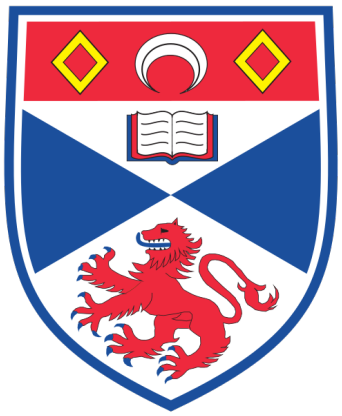

University of
St Andrews

Visual Transformation Based Preprocessing and Depth Estimation for Robust Mapping in SLAM system

CS5098 – Group Project and Dissertation

Kumar Gaurav Atram
Student ID – 220025456
Date – 15th Aug 2023

Supervised by
Dr. Christopher Stone
Dr. Juan Ye

Abstract

Simultaneous Localization and Mapping (SLAM) is a critical technology enabling autonomous navigation and scene reconstruction in various fields. This dissertation introduces a novel approach to enhance the robustness of SLAM systems by leveraging Visual Transformation (ViT) for Depth Estimation. The proposed methodology builds upon the innovative Meta AI DINO v2 visual transformer architecture, harnessing its self-supervised learning capabilities to extract high-level feature representations from raw visual data.

By incorporating deep learning techniques with self-attention of ViT DINO v2, which leverages monocular image sequences, the proposed approach enables accurate depth estimation without the need for costly depth sensors. This allows for robust mapping and localization even in challenging environments where depth information is limited or unreliable.

To evaluate the effectiveness of the proposed method, comprehensive experiments were conducted using benchmark datasets and real-world scenarios. The experimental results demonstrate that the integration of depth estimation technique using visual transformer could be a promising component in SLAM pipeline.

Declaration

I declare that the material submitted for assessment is my own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the current academic year except where otherwise stated.

This Group project is completed in collaboration with Ashwin Kashyap H K (220033001) and under the supervision of Dr. Christopher Stone and Dr. Juan Ye.

The main text of this project report is approximately 9,702 words long.

In submitting this project report to the University of St Andrews, I give permission for it to be made available for use in accordance with the regulations of the University Library. I also give permission for the title and abstract to be published and for copies of the report to be made and supplied at cost to any bona fide library or research worker, and to be made available on the World Wide Web. I retain the copyright in this work. If there is a strong case for the protection of confidential data, the parts of the declaration giving permission for its use and publication may be omitted by prior permission of the Director of Postgraduate Teaching or Honours coordinator (as appropriate).

Acknowledgements

I extend my gratitude to my supervisor, Dr. Christopher Stone, for his support, guidance, debugging sessions, thoughtful suggestions, and explanations that have been instrumental in shaping the course of this project.

Additionally, I would like to express my sincere appreciation to my second supervisor, Dr. Juan Ye, for her invaluable assistance, constructive suggestions, and expert guidance, all of which have greatly contributed to the development and success of this project.

Table of Contents

Table of Figures

Index of Tables

1. Introduction

Visual simultaneous localization and mapping (SLAM) is a fundamental technology that enables robots, augmented reality systems, and autonomous vehicles to navigate and understand their surroundings. It involves the simultaneous estimation of the robot's position and orientation (localization) and the creation of a map of the environment (mapping) using visual sensor data. Accurate depth estimation is crucial for the reliable operation of SLAM systems in real-world scenarios.

Common challenge in visual SLAM is the inconsistent estimation of depth information. Depth estimation plays a vital role in accurately reconstructing the environment and performing reliable localization. Inaccurate or inconsistent depth estimates can lead to incorrect mapping, alignment errors, and incorrect scale estimation. While depth sensors such as LiDAR and stereo cameras can provide reliable depth information, they are often costly and may have limited availability or be prone to occlusion in certain scenarios.

To address the challenge, this dissertation proposes a novel approach titled "Visual Transformation Based Preprocessing and Depth Estimation for Robust Mapping in SLAM system" The objective of this research project is to leverage the power of DINO v2, a state-of-the-art deep learning framework, to improve the accuracy and robustness of visual SLAM systems.

DINO v2 is an unsupervised learning framework that can learn meaningful representations from raw image data. By incorporating DINO v2 into the SLAM pipeline, we can enhance the preprocessing of visual inputs, improving the robustness and accuracy of feature tracking and matching. This, in turn, addresses the issue of drift correction and contributes to more accurate depth estimation.

Furthermore, this research project aims to utilise the power of visual transformer framework for depth estimation. DINO v2's self-supervised training scheme, leveraging monocular image sequences, enables accurate depth estimation without the need for costly depth sensors. By incorporating DINO v2's depth estimation capabilities, the proposed approach can overcome inconsistent depth estimations, leading to more robust mapping and accurate localization, even in environments with limited or unreliable depth information.

To evaluate the effectiveness of the proposed approach, comprehensive experiments will be conducted using benchmark datasets and real-world scenarios.

The findings of this research project have significant implications for various applications that rely on reliable visual SLAM systems, including robotics, augmented reality, and autonomous navigation. By potential use of proposed depth estimation technique in visual SLAM, this research

contributes to the advancement of these fields, enabling more accurate and robust perception in dynamic environments.

In summary, this dissertation aims to investigate and develop a deep learning-based approach using DINO v2 ViT for preprocessing and depth estimation. By addressing inconsistent depth estimations, the proposed approach holds the potential to significantly improve the accuracy and robustness of SLAM system, opening doors to enhanced capabilities in robotics, augmented reality, and autonomous navigation.

1.1 Objective

1.1.1 Primary Objectives

The primary objective of this research project is to enhance depth estimation in visual SLAM by leveraging the power of self-attention self-supervised visual transformer DINO v2. Specifically, we aim to:

- Investigate the current landscape of state-of-the-art SLAM methodologies and pinpoint opportunities for potential enhancements.
- Investigate diverse neural network architectures suitable for serving as feed-forward networks to complement the DINO v2 visual transformer.

1.1.2 Secondary Objectives

- Assessing the system's latency to determine its suitability for real-time operational contexts.
- Conducting thorough testing and evaluation of the network's performance within indoor environments.

1.1.3 Tertiary Objectives

- Improve latency to be useful in real world dynamic environments.
- Integrate transformer based network in SLAM system.

2. Context Survey

Visual SLAM (Simultaneous Localization and Mapping) is a field of research that combines computer vision, sensor fusion, and robotics to enable a system to simultaneously estimate its own pose (position and orientation) in an environment while constructing a map of that environment. Visual SLAM primarily utilizes visual data, such as camera images or video sequences, to perform localization and mapping tasks.

2.1 SLAM Architecture

A SLAM system comprises two primary components: the front-end and the back-end. The front-end is responsible for transforming sensor data into suitable models for estimation. On the other hand, the back-end performs inference on the transformed data generated by the front-end.

2.1.1 Front-end

The front-end of the SLAM system is responsible for extracting pertinent features from the sensor data. In the case of vision-based SLAM, the front-end identifies distinguishable points in the environment and captures their pixel locations. These pixel observations are then readily modelled in the back-end. Additionally, the front-end handles the crucial task of associating each measurement with a specific landmark or 3D point in the environment, a process known as data association. More broadly, the data association module links each measurement (z_k) with a subset of unknown variables (X_k) in such a way that $z_k = h_k(X_k) + \varepsilon_k$. Furthermore, the front-end may provide an initial estimate for the variables in the nonlinear optimization process. For example, in feature-based monocular SLAM, the front-end typically initiates landmark positions by triangulating them from multiple viewpoints. (Cadena et al., 2016)6

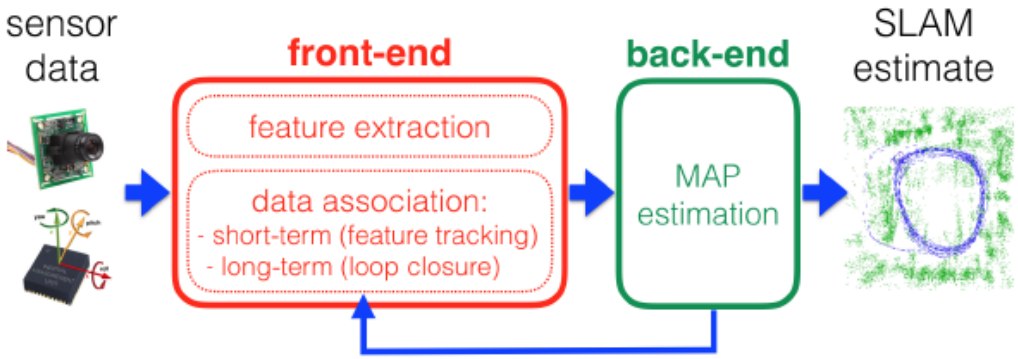

Figure 1: Front-end and back-end in a typical SLAM system. (Cadena et al., 2016)

Figure1 illustrates a visual representation of a typical SLAM system. Within the front-end, the data association module consists of two components: short-term data association and long-term data association. The short-term data association is responsible for establishing correspondences between features in consecutive sensor measurements. For example, it tracks the relationship between two pixel measurements in consecutive frames that represent the same 3D point. On the other hand, the long-term data association, also known as loop closure, associates new measurements with previously observed landmarks. It should be noted that the back-end of the SLAM system often provides feedback to the front-end. This feedback facilitates tasks such as loop closure detection and validation. (Cadena et al., 2016)6

2.1.2 Back-end

The backend of the system receives the raw data captured by the visual sensor from the frontend and conducts calculations and optimizations. The optimization process in the backend primarily involves two approaches: the filter-based backend, such as the Extended Kalman Filter (EKF), and the nonlinear optimization backend, such as graph optimization. In the context of Visual Simultaneous Localization and Mapping (V-SLAM), the nonlinear optimization method is commonly preferred due to its superior effectiveness and stability. (Jia et al., 2022)24

Mapping plays a crucial role in achieving precise positioning, navigation, obstacle avoidance, reconstruction, and interaction capabilities. Both the frontend and backend optimizations play a significant role in preparing the data for the mapping process. These optimization steps ensure that the data is refined and optimized to provide accurate and reliable mapping results.(Jia et al., 2022)24

2.2 Feature Extraction in Visual SLAM

Camera-based SLAM involves detecting visual features in the environment that can be recognized by human vision systems, such as corners, edges, colours, shadows, and depths. The process of feature extraction plays a crucial role in image matching, enabling the creation of maps and object detection techniques. However, the increasing use of high-quality and high-resolution cameras has led to the generation of large amounts of data, posing challenges in terms of processing speed and the need for powerful and expensive processors. These processors also consume significant power, making them less suitable for mobile robots.

Feature extraction serves the purpose of reducing the size of input data by selecting relevant data and features from the input images. Features in an image are pixels that possess common properties and are distinctive from neighbouring pixels. For object detection techniques, these features can include image details such as texture, mean pixel values, colour, corners, and more. In the case of vision-based SLAM, the features should be invariant to rotation, orientation, translation, scaling, and changes in luminous intensity.

In the context of mobile robots, the vision system plays a crucial role in perceiving the surrounding objects and finding collision-free paths. Therefore, it is essential to accurately determine the pose of objects in the environment. This necessitates comparing each frame of the video stream captured by the cameras to identify relevant features and track their movements. This real-time system requires a fast algorithm capable of efficiently identifying and extracting the most relevant image features while filtering out irrelevant data. The accurate identification of the current robot position is crucial for ensuring obstacle avoidance and precise mapping. When executed effectively, the robot's path will be free from collisions, and the system will generate a comprehensive map that outlines the area's boundaries and the obstacles it successfully manoeuvred around.

Within SLAM, the ability to capture various angles, orientations, and patterns of a specific object within a single image is crucial. Texture analysis, as a method for objectively assessing and characterizing texture attributes, proves beneficial in facilitating object recognition and classification tasks (Xu, Zhang, Li, Liu, & Zhu, 2021)52.

2.2.1 PCA

The computational efficiency of feature vectors representing texture attributes in high-quality images is hindered by their high dimensionality. To address this issue, it becomes necessary to employ a method that combines texture representation with dimensionality reduction. **Principle Component Analysis (PCA),** a widely utilized technique in object-based analysis, is employed for data reduction. PCA transforms multidimensional data into a linear vector by identifying linear variables with the highest variance. Current versions of PCA are employed for enhanced feature extraction and dimension reduction of image matrices.

The **(2D)2PCA** technique, which stands for 2-directional 2-dimensional principal component analysis, is an approach that involves applying PCA separately in both the row and column directions of images. This method allows for simultaneous size reduction in both the rows and columns of images. (Kazerouni & Haddadnia, 2014)23. (2D)2PCA is an efficient technique for data representation, feature extraction, and dimensional reduction of image matrices. It aims to find a more compact and less redundant representation of image data, where a reduced number of components can independently account for the variations in the image data (Abaspur Kazerouni, Dooly, & Toal, 2020)1.

Given a 2-dimensional PCA operator in an image A with m rows and n columns we can define the covariance matrix C as:

$$C = \frac{1}{M}\sum_{k=1}^{M}\sum_{i=1}^{m}\left(A_k^{(i)} - \overline{A}^{(i)}\right)\left(A_k^{(i)} - \overline{A}^{(i)}\right)^T \text{AA}$$

where M is the training sample with m by n matrices, which are shown by A_k (k = 1, 2,..., M) and $\overline{A}$ and C represent the average matrix and covariance matrix respectively and $A_k^{(i)}A_k^{(i)}$ and $\overline{A}^{(i)}$ $\overline{A}^{(i)}$ denote the i-th row vectors of A_k and $\overline{A}$ $\overline{A}$ respectively. Another 2-dimensional PCA can be applied in the image columns as:

$$C = \frac{1}{M}\sum_{k=1}^{M}\sum_{j=1}^{n}\left(A_k^{(j)} - \overline{A}^{(j)}\right)\left(A_k^{(j)} - \overline{A}^{(j)}\right)^T \text{AA}$$

where $A_k^{(j)}A_k^{(i)}$ and $\overline{A}^{(j)}$ $\overline{A}^{(i)}$ denote the j-th column vectors of A_k and $\overline{A}$ $\overline{A}$ respectively, and q first high eigenvalues of matrix C are located as columns in the matrix Z which $Z \in R^{m\times q}$Z∈$R^{m\times q}$. Projecting the random matrix A onto Z yields a 'q by n' matrix $Y = Z^TAY = Z^TA$ and projecting the matrix A onto Z and X generates a 'q by d' matrix $= Z^TAX$ Y = Z^TAX.

(Abaspur Kazerouni et al., 2022)2

To achieve higher accuracy and faster results in navigation and SLAM, it is crucial to identify the most suitable features in images that are invariant to orientation, angle, and contrast. Various modern techniques have been proposed in the past for keypoint feature extraction and matching. These techniques include Scale Invariant Feature Transform (SIFT), Speeded Up Robust Features (SURF), Features from Accelerated Segment Test (FAST), Binary Robust Independent Elementary Features (BRIEF), Oriented FAST and Rotated BRIEF (ORB), Learned Invariant Feature Transform (LIFT), Fast Retina Keypoint (FREAK), Binary Robust Invariant Scalable Keypoints (BRISK), Histogram of Oriented Gradients (HOG), A Fast-Local Descriptor for Dense Matching (DAISY), Gradient Location and Orientation Histogram (GLOH), Local Intensity Order Pattern (LIOP), MatchNet, MROGH, KAZE, and A-KAZE. When analyzing a sequence of frames from a video stream in real-time, a smart system employs feature matching algorithms to identify common features across the frames. Using information about the orientation and depths of the images, the system estimates the position based on these features.

2.2.2 SIFT

Lowe (2004)27 introduced **SIFT** (Scale-Invariant Feature Transform), an algorithm that addresses the challenges of image rotation and scale invariance. SIFT is commonly employed for key-point feature matching and exhibits strong robustness in handling image scaling and rotation.

Additionally, it offers partial invariance to variations in illumination and affine transformations. The algorithm achieves accurate key-point localization by utilizing the Taylor expansion of the Difference-of-Gaussian (DOG) scale-space function, denoted as D(x, y, σ), which is shifted to have its origin at the candidate point (Lowe, 2004)27.

$$D(\mathrm{x}) = D + \frac{\partial D^T}{\partial \mathrm{x}}\mathrm{x} + 0.5\mathrm{x}^T\frac{\partial^2 D}{\partial \mathrm{x}^2}\mathrm{x},$$

where D and its derivatives are evaluated at the candidate point, x= (x, y, σ) $^{\mathrm{T}}$ is the offset from this point and the location of the extremum is defined as:

$$\hat{\mathrm{x}} = -\frac{\partial^2 D^{-1}}{\partial \mathrm{x}^2}\frac{\partial D}{\partial \mathrm{x}}$$

And for rejecting unstable extrema with low contrast, the function value at the extremum is expressed by:

$$D(\hat{\mathrm{x}}) = D + 0.5\frac{\partial D^T}{\partial \mathrm{x}}\hat{\mathrm{x}}$$

(Abaspur Kazerouni et al., 2022)2

SIFT is a valuable tool for feature matching and image mosaicking applications (Rublee, Rabaud, Konolige, & Bradski, 2011)42. However, its extensive computational requirements make it challenging to utilize in real-time systems. SIFT tends to perform more effectively in regions with abundant blobs rather than regions with numerous corners.

2.2.3 FAST

In order to apply SLAM techniques to mobile robots, which often have limited computational resources, the **FAST** (Features from Accelerated Segment Test) algorithm can be a valuable solution to address the challenges associated with SIFT. FAST, proposed by Rosten and Drummond (2006)41, is a corner detection algorithm that proves useful in this context. It examines neighboring pixels within a defined radius around each pixel in the image. By comparing the intensities of these contiguous pixels with a threshold, the algorithm determines if they are brighter or darker than the candidate pixel. If a specific set of contiguous pixels are all brighter or all darker than the candidate pixel, it is classified as a corner pixel. The FAST corner detector is known for its high speed, but it may struggle to extract accurate features in the presence of image noise. Additionally, selecting an appropriate threshold level can be challenging for certain images.

2.2.4 SURF

Speeded Up Robust Features (SURF), proposed by Bay, Ess, Tuytelaars, and Van Gool (2008)5, is a faster alternative to SIFT. It operates by generating multiple levels of image and descriptor pairs. Similar to SIFT, SURF analyses the image but instead of using the Difference-of-Gaussians, it employs the Haar wavelet. The SURF algorithm leverages the determinant of the Hessian matrix to select the location and scale, enabling the generation of a fast and accurate descriptor. An approximation for the determinant of the Hessian is defined as

$$\det(H_{approx}) = D_{xx}D_{yy} - 0.81D_{xy}\text{^}2$$

where D_{xx}, D_{yy}, and D_{xy} represent approximations for Gaussian second-order derivatives at the highest spatial resolution. The computational cost and calculations involved in SURF are more suitable for real-time applications, although SIFT tends to exhibit greater robustness in capturing important features and details.

2.2.5 BRIEF

A descriptor algorithm called BRIEF was introduced by Calonder et al. (2011)7 to improve efficiency in real-time applications. BRIEF stands for **Binary Robust Independent Elementary Features**. This algorithm is sensitive to noise and utilizes a Gaussian kernel to smooth the image before converting it into a binary feature vector. The BRIEF feature descriptor is defined as the summation of binary tests applied to the image patch (the neighbourhood around a pixel). The BRIEF feature descriptors can be defined as:

$$f_n(p) = \Sigma\, 2^{i-1}\, \tau(p; x_i, y_i)$$

Where p is image patch (the neighbourhood around pixel), τ is binary test for the selected set of n(x, y). Choosing the proper test points is vital for this algorithm.

2.2.6 ORB

Rublee et al. (2011)42 introduced the **ORB** (Oriented FAST and Rotated BRIEF) algorithm as a descriptor. This approach builds upon the BRIEF descriptor and utilizes the FAST key-point detector. Additionally, ORB measures the corner orientation by employing an intensity centroid. The moments of an image patch can be expressed as:

$$m_{pq} = \Sigma\, x^p\, y^q\, I(x,y)$$

With these moments, the center of mass of the patch can be defined as:

$$C = (m_{10}/m_{00},\ m_{01}/m_{00})$$

Using C, the vector from the corner's center, O is computed as OC. The orientation of the patch is defined as:

$$\theta = \text{atan2}(m01, m10)$$

By utilizing θ, the feature can be rotated to a canonical orientation, enabling the computation of the descriptor with some degree of rotation invariance. Although the ORB algorithm is highly efficient, it is less effective in terms of scale. This approach is implemented in SLAM methodologies known as ORB-SLAM (Mur-Artal et al., 2015)33 and ORB-SLAM2 (Mur-Artal & Tardós, 2017)34 35. These techniques are feature-based SLAM methods that employ the ORB feature descriptor to generate keyframes based on a graph structure.

2.2.7 HOG

Dalal and Triggs (2005)14 introduced a fundamental model known as **Histogram of Oriented Gradients** (**HOG**) for human detection. The HOG algorithm utilizes image gradients to extract and define the main features of objects, specifically the distribution of directions of gradients, also known as oriented gradients. By analysing the gradients, the algorithm can identify the corners and edges of objects, which serve as crucial features. It is important to note that HOG is considered one of the fundamental algorithms in feature descriptors and predates the introduction of deep learning techniques in this domain.

2.3 Current Visual SLAM

2.3.1 MonoSLAM

The pioneering work in monocular Visual SLAM (vSLAM) can be attributed to Davison et al. (2007)16 with the development of MonoSLAM, which stands as a representative method in filter-based vSLAM algorithms. MonoSLAM utilizes an extended Kalman filter (EKF) to simultaneously estimate camera motion and the 3D structure of an unknown environment. The camera's six degrees of freedom (DoF) motion and the 3D positions of feature points are represented as a state vector within the EKF framework. The prediction model assumes uniform motion, while the observations are derived from feature point tracking. As the camera moves, new feature points are incorporated into the state vector. It is important to note that the initial map is built incrementally, and subsequent observations are used to refine the map over time (Davison et al., 2007)16.

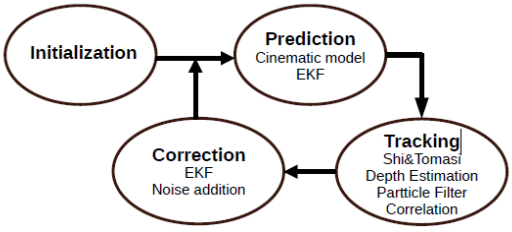

Figure 2: MonoSLAM algorithm (Macario Barros et al., 2022)

One limitation of the MonoSLAM method is its computational cost, which scales with the size of the environment being mapped (Taketomi, Uchiyama and Ikeda, 2017)48. As the number of feature points increases in larger environments, the size of the state vector grows significantly. Consequently, real-time computation becomes challenging due to the increased computational requirements associated with processing a large state vector.

2.3.2 ORB-SLAM

ORB-SLAM3 builds upon the foundations of ORB-SLAM2 (Mur-Artal and Tardos, 2017a)34 and ORB-SLAM-VI (Mur-Artal and Tardos, 2017b)35. It represents a comprehensive system that supports multiple maps and sessions, allowing it to operate in both visual and visual-inertial modes. It is compatible with various sensor configurations, including monocular, stereo, and RGB-D sensors, and supports both pin-hole and fisheye camera models. This versatility enables ORB-SLAM3 to adapt to different scenarios and sensor setups, making it a highly flexible and adaptable visual SLAM solution. (Campos et al., 2021)8.

ORB (Oriented FAST and Rotated BRIEF) features (Rublee et al., 2011)42 are multiscale FAST corners with associated 256-bit descriptors. These features are known for their exceptional computational speed in both computation and matching, making them highly efficient for visual SLAM applications. Moreover, ORB features exhibit robustness to changes in viewpoint, enabling reliable matching even across wide baselines. This property contributes to improved accuracy in Bundle Adjustment (BA), a crucial optimization step in visual SLAM algorithms. (Mur-Artal, Montiel and Tardos, 2015)33.

The algorithm is widely regarded as a leading feature-based approach in the field of visual SLAM. It operates in three concurrent threads: tracking, local mapping, and loop & map merging. The tracking thread is responsible for determining the sensor's location by establishing correspondences between features and minimizing the re-projection error. The local mapping thread manages the operations related to map creation and maintenance. The loop closing thread focuses on identifying new loops and rectifying drift errors within those loops. To ensure overall structure consistency and motion estimation accuracy, the algorithm incorporates a full bundle adjustment step. Figure 3 illustrates the individual threads that constitute the algorithm. This approach supports monocular, stereo, and RGB-D setups and incorporates global optimization and loop closure techniques. (Macario Barros et al., 2022)28. However, authors in (Seiskari et al., 2022)45 demonstrated significant errors results of ORB-SLAM3 online performance. In (Merzlyakov and Macenski, 2021)30, the algorithm obtained a good performance, but failed to process all the sequences, and obtained inaccurate estimates in outdoor sequences.

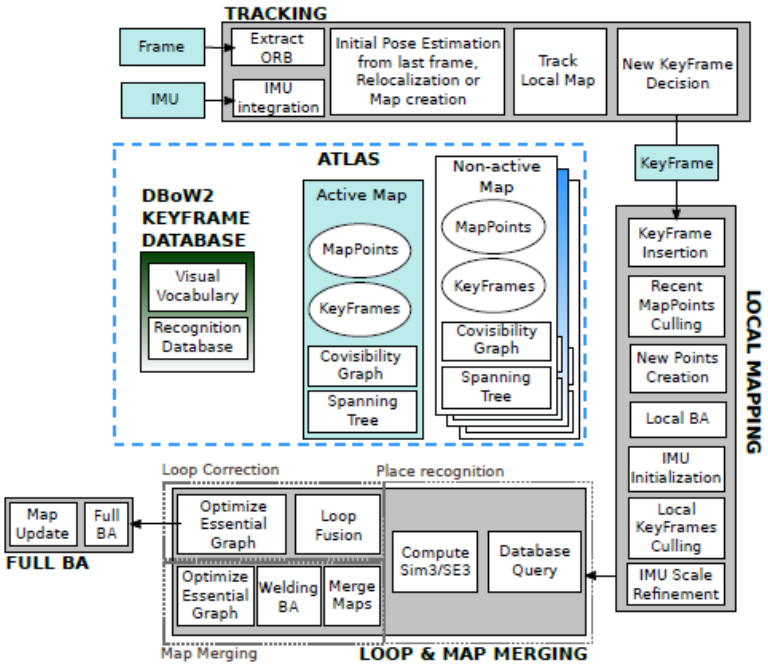

Figure 3: ORB-SLAM3 algorithm (Macario Barros et al., 2022)

2.3.3 DeepSLAM

DeepSLAM is an innovative monocular SLAM system that leverages unsupervised deep learning techniques. This system takes monocular colour images as input and simultaneously generates the pose trajectory, depth map, and 3D point cloud (refer to Figure 4). DeepSLAM can be summarized as follows:

- A novel visual SLAM framework that combines unsupervised deep learning and geometric constraints to enhance performance.
- A Deep Recurrent Convolutional Neural Network (RCNN) specifically for modelling ego-motion. This network leverages the spatial and temporal properties of a sequence of stereo images during training.
- DeepSLAM integrates the DL-based tracking results and loop closure detection with a graph-based optimization mechanism, resulting in a comprehensive visual SLAM system.
- To improve robustness in challenging scenes, DeepSLAM incorporates outlier rejection using uncertainty estimates derived from error maps based on both geometric and photometric consistencies.

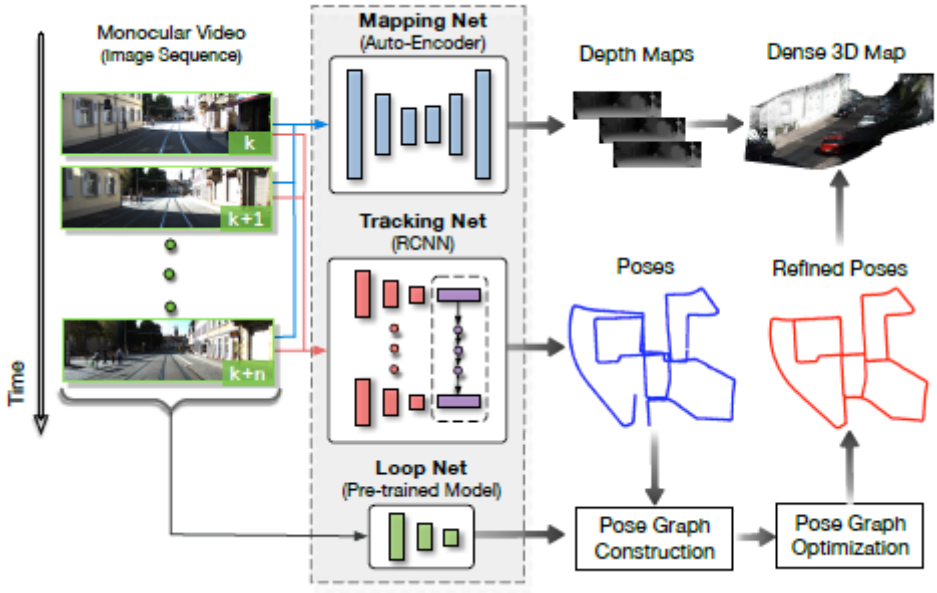

Figure 4: DeepSLAM framework ((Li, Wang and Gu, 2021)

By utilizing the Mapping-Net, Tracking-Net, and Loop-Net, DeepSLAM takes monocular colour images as input and generates depth maps, poses, and point clouds as outputs. The system employs specific neural network architectures for each task:

- Mapping-Net: This autoencoder architecture is responsible for depth estimation, providing accurate depth maps based on the input images.
- Tracking-Net: Built upon a RCNN (Recurrent Convolutional Neural Network), this architecture focuses on pose estimation. It utilizes both spatial and temporal properties of the input sequence to estimate the camera's motion and position.
- Loop-Net: Designed as a CNN (Convolutional Neural Network), Loop-Net is responsible for loop closure detection. It analyses the visual features of the input images to identify potential loop closures within the environment.

DeepSLAM presented a better performance than other monocular algorithms, as the ORB-SLAM, and better robustness than ORB-SLAM and LSD-SLAM. (Macario Barros et al., 2022)28

2.4 Depth Estimation in Visual SLAM

Depth estimation in vSLAM refers to the process of estimating the distances of objects in a scene from the camera's viewpoint. Accurate depth information is crucial for scene understanding, 3D reconstruction, and navigation tasks in visual SLAM systems. Perceiving and recognizing the physical surroundings and environment is a crucial requirement for SLAM and robotic systems. To achieve this, autonomous robots need sensors capable of capturing and analyzing accurate information from the environment to construct a map. Cameras are commonly employed as sensors in SLAM applications, offering several advantages. They enable users to incorporate additional

functionalities like object detection, segmentation, and other vision-based systems. Alongside cameras, other sensor types like RGB-D cameras and LiDAR are also utilized in SLAM for their unique capabilities.

2.4.1 SONAR

SONAR sensors are commonly used in mobile robots as they can detect objects by analysing the echo of ultrasonic signals that bounce off them. These sensors rely on sound signals, making them suitable for environments with low visibility, such as underwater settings. SONAR sensors offer extensive coverage for mapping and navigation purposes. However, they are not ideal for precise object detection tasks, as they may struggle to accurately detect object corners. Commercial acoustic sensors are available in various power consumption levels and operating frequencies, providing flexibility for different applications. (Abaspur Kazerouni et al., 2022)2

2.4.2 LiDAR

LiDAR-based SLAM has become a widely used mapping system in modern robotic applications. While earlier versions of LiDAR sensors were typically large and heavy, there are now compact and lightweight LiDAR options available that enable fast and accurate imaging. LiDAR operates similar to RADAR but utilizes laser light instead of radio waves to detect and image an area, objects, and depth up to a range of 300 meters. With its capability, LiDAR can generate precise maps of both indoor and outdoor environments (Hess, Kohler, Rapp, & Andor, 2016)22.

2.4.3 Monocular camera

Monocular cameras are widely used and easily accessible sensors that are commonly found as standard components in many robots. These cameras can be effectively employed in SLAM techniques. There is a vast array of algorithms, source code, books, and technical papers available for image processing, machine vision, and deep learning, which leverage the RGB images captured by cameras. Numerous algorithms exist for constructing environment maps and performing object detection through data fusion and image analysis. Monocular cameras are particularly advantageous for low-power and cost-effective SLAM projects (Civera, Davison, & Montiel, 2008; Engel et al., 2014; Mur-Artal et al., 2015)10 20 33.

Over the past few years, there has been an integration of object detection methods into robotic systems in order to create comprehensive maps of areas while also being able to recognize and differentiate objects within the environment. RGB-D cameras have emerged as a popular choice for this purpose as they enable the detection of objects, corners, and contours through clustering techniques. These cameras consist of infrared (IR) transmitters, IR receivers, and a monocular camera, which combine to produce an RGB image by projecting structured light patterns in the infrared spectrum onto the scene and capturing the reflected light to determine the depth or range of each pixel in the image (Sturm, Engelhard, Endres, Burgard, & Cremers, 2012)46.

2.4.4 Stereo camera

Stereo cameras provide an alternative vision system for SLAM that utilizes two captured images of an object taken from different angles to estimate depth. The introduction of RGB-D sensors has significantly advanced SLAM techniques (Dryanovski, Valenti, & Xiao, 2013)17. These systems offer advantages such as low cost and high mobility. However, RGB-D sensors also have limitations in dense 3D mapping systems. They have a restricted range and field of view (FoV), which can lead to tracking difficulties as they lack the necessary spatial structure to constrain iterative closest point (ICP) alignments (Tang et al., 2016)49.

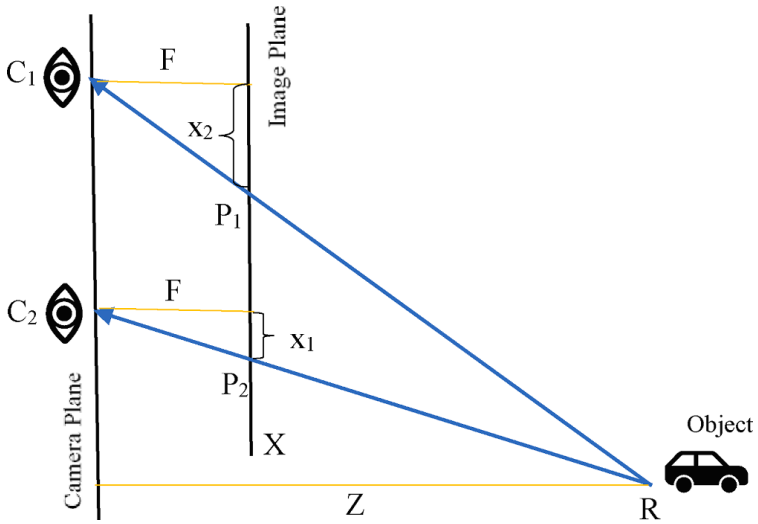

Figure 5: Shows the stereo vision geometry for stereo camera. (Abaspur Kazerouni et al., 2022)

the depth of the object point (Z) can be obtained as:

$$X/Z = x_2/F$$

$$1/Z(X-C_1-C_2) = x_1/F$$

$$Z = \frac{F(C_1 - C_2)}{x_2 - x_1}$$

where F = focal length of the cameras.

C1 and C2 are Camera1 and Camera2 positions.

x1 and x2 are image space coordinates.

A 3D view of an area can be obtained by knowing depth of each point in stereo imaging. (x2 – x1) is the disparity defined as the distance between points in image plane corresponding to camera center and the scene point 3D. (Abaspur Kazerouni et al., 2022)2

The successful implementation of SLAM relies on the selection of appropriate sensing technologies, taking into consideration factors such as price, power consumption, real-time processing capabilities, size, and noise reduction ability. Acoustic sensors, while useful in certain scenarios, are generally unable to detect and extract detailed features of an unknown area with sufficient resolution. LIDAR sensors, although effective, are often bulky, expensive, and can pose challenges in pose estimation. On the other hand, monocular cameras lack depth information, while RGB-D cameras can provide depth information and object features but are limited in their range (Zaffar, Ehsan, Stolkin, & Maier, 2018)53.

2.4.5 RGB-D camera

Lately, robotic systems have integrated object detection techniques to create a detailed map of a given area, enabling the identification of discernible objects within the surroundings. Among the preferred options for achieving this goal, RGB-D cameras have gained popularity. These cameras utilize a combination of infrared (IR) transmitters, IR receivers, and a monocular camera. By projecting structured light patterns in the infrared spectrum onto a surface and capturing the resulting reflections, an RGB image is generated. This process provides depth or range information for each pixel in the image, facilitating object identification, corner and contour detection, as well as clustering. This approach was detailed by Sturm, Engelhard, Endres, Burgard, and Cremers in 201246.(Abaspur Kazerouni et al., 2022)2

SLAM sensors.

Sensor	Application	Range (Frame Rate for cameras and Operating Frequency for acoustics)	Price (€)	SLAM Type	Power Consumption (W)
Sonar	Underwater	1 kHz and 500 kHz	500	Bat SLAM	0.01–5
Laser	Indoor, Outdoor and Drones	1–100 Hz	200	Laser-based SLAM, TinySLAM	1–100
LiDAR	Indoor, Outdoor and Drones	1–50 Hz	100–5 K	LiDAR SLAM, CT-SLAM	5–200
RGB-D	Indoor	10–400 Hz	150–500	RGB-D SLAM	0.3–5
Monocular Camera	Indoor and Outdoor	20–200 Hz	100–5 K	Mono SLAM	0.01–10
Stereo	Indoor, Outdoor and UAV	1–400 Hz	400–4 K	Stereo LSD-SLAM	2–15

Figure 6: Summary of Sensors (Abaspur Kazerouni et al., 2022)

2.5 Depth Estimation using Deep Learning

2.5.1 DINO v2: Self-distillation with no labels

Meta AI (ai.facebook.com, n.d.)3 has developed DINOv2, an innovative approach for training advanced computer vision models. DINOv2 demonstrates exceptional performance without the need for fine-tuning, making it an excellent choice as a fundamental component for various computer vision applications.

Utilizing self-supervision, DINOv2 has the ability to learn from diverse image datasets, making it highly versatile. Moreover, it surpasses the capabilities of the existing conventional methods by acquiring additional features like depth estimation.

DINOv2 offers powerful features that can be directly employed as inputs for straightforward linear classifiers. This inherent flexibility allows DINOv2 to serve as a versatile foundation for a wide range of computer vision tasks. Extensive evaluations indicate that DINOv2 exhibits exceptional predictive capabilities in tasks like classification, segmentation, and image retrieval.

Remarkably, when it comes to depth estimation, the features derived from DINOv2 outperform specialized state-of-the-art pipelines, both within the domain and outside of it. This impressive out-of-domain performance is believed to be a result of the combination of self-supervised feature learning and the utilization of lightweight task-specific modules, such as linear classifiers.

Importantly, since DINOv2 doesn't rely on fine-tuning, the backbone remains generalizable, allowing the same features to be concurrently utilized across diverse tasks. This attribute enhances the efficiency and versatility of DINOv2, making it an advantageous choice for various computer vision applications.

DINO v2 is based on its predecessor DINO. Figure 7 illustrates the process of self-distillation. (Caron et al., 2021)9 showcase the DINO approach in a simplified scenario involving a single pair of views (x1, x2). In this setup, an input image undergoes two distinct random transformations, and these transformed images are passed through both the student and teacher networks. Despite having identical architectures, these networks possess different parameters.

The output of the teacher network is adjusted to have a mean computed across the batch. Both networks generate a K-dimensional feature, which is then normalized using a temperature-based softmax applied over the feature dimension. Their similarity is quantified using a cross-entropy loss. To direct gradients solely through the student network, we employ a stop-gradient (sg) operator on the teacher network. The teacher's parameters are updated through exponential moving averages (ema) derived from the student's parameters.

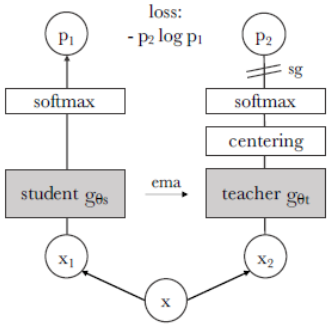

Figure 7: Self-distillation with no labels (Caron et al., 2021)

Figure 8 shows performance on eight types of vision tasks, and average metrics with each type. Features are extracted from self-supervised encoders, DINOv2 (dark blue), and compared with self-supervised methods(pale orange), as well as weakly-supervised methods (dark pink). (ai.facebook.com, n.d.)3

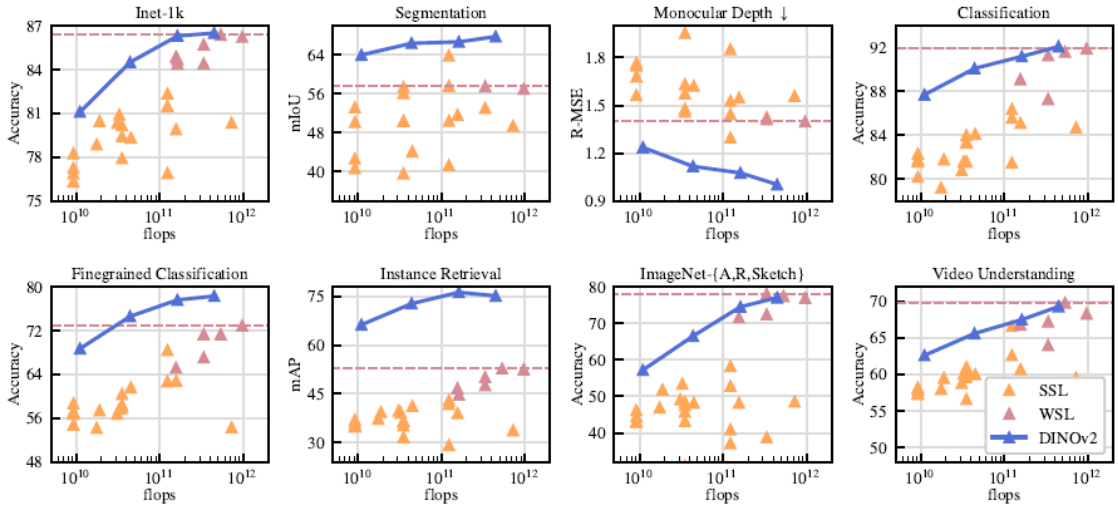

Figure 8: shows performance on eight types of vision tasks.(ai.facebook.com, n.d.)

2.5.2 NeRF: Neural Radiance Fields

Neural Radiance Fields (NeRF) are fully-connected neural networks capable of generating new perspectives of intricate 3D scenes, even with a limited number of 2D images. Through training, NeRF learns to utilize a rendering loss to faithfully reproduce the input views of a scene. It accomplishes this by taking the input images representing the scene and employing interpolation

techniques to generate a complete scene. NeRF proves highly effective in generating synthetic data images.(Datagen, n.d.) 15

By training a NeRF network, it becomes proficient in directly mapping viewing direction and spatial location (5D input) to opacity and color (4D output). This mapping is employed in volume rendering to generate fresh views. NeRF, due to its computational complexity, can require substantial processing time, ranging from hours to days, particularly for complex scenes. Nonetheless, there have been recent advancements in algorithms that significantly enhance NeRF's performance.

Basic Concepts

To understand how NeRF works, lets review some basic concepts

1. **Rendering** involves the creation of a visual image from a 3D model. This model encompasses various elements such as textures, shading, shadows, lighting, and viewpoints. The responsibility of a rendering engine is to process these elements to produce a lifelike image.

There are three commonly employed rendering algorithms:

- Rasterization: This algorithm projects objects geometrically based on the information present in the model, without incorporating optical effects.
- Ray casting: This algorithm calculates an image from a particular viewpoint by employing basic optical laws of reflection.
- Ray tracing: This algorithm utilizes Monte Carlo techniques to generate a realistic image in significantly less time. Ray tracing is employed to enhance rendering performance in NVIDIA GPUs.

2. **Volume rendering** allows the creation of a 2D representation of a 3D dataset that has been sampled discretely.

In volume rendering, an algorithm takes into account the camera position and casts rays from the camera through the space. For each voxel encountered by these rays, the algorithm retrieves the RGBα (Red, Green, Blue, and Alpha channel) values. These RGBα values are then converted to RGB colors and stored in the corresponding pixel of the resulting 2D image. This process continues for every pixel until the entire 2D image is rendered.

3. **View synthesis** is the inverse process of volume rendering, as it entails generating a 3D view from a collection of 2D images. This can be achieved by utilizing a sequence of photographs capturing an object from various angles. By constructing a hemispheric representation of the object, each image is positioned accurately around it. A view synthesis function aims to estimate the depth based on the series of images that depict different viewpoints of the object.

NeRF Working

NeRF leverages a limited set of input views to optimize a continuous volumetric scene function, enabling the generation of new perspectives of intricate scenes. Input for NeRF can be supplied in the form of a fixed collection of images.

The continuous scene in NeRF is represented by a 5D vector-valued function, possessing the following attributes:

- Input: A 3D location x = (x, y, z) and a 2D viewing direction (θ, Φ).
- Output: An emitted color c = (r, g, b) and volume density (α).

Figure 9: 5D NeRF representation of a scene from a set of input images (Mildenhall et al., 2020)

In essence, NeRF utilizes this 5D function to transform input coordinates and viewing directions into corresponding emitted colors and volume densities, enabling the generation of diverse and realistic views of the scene.

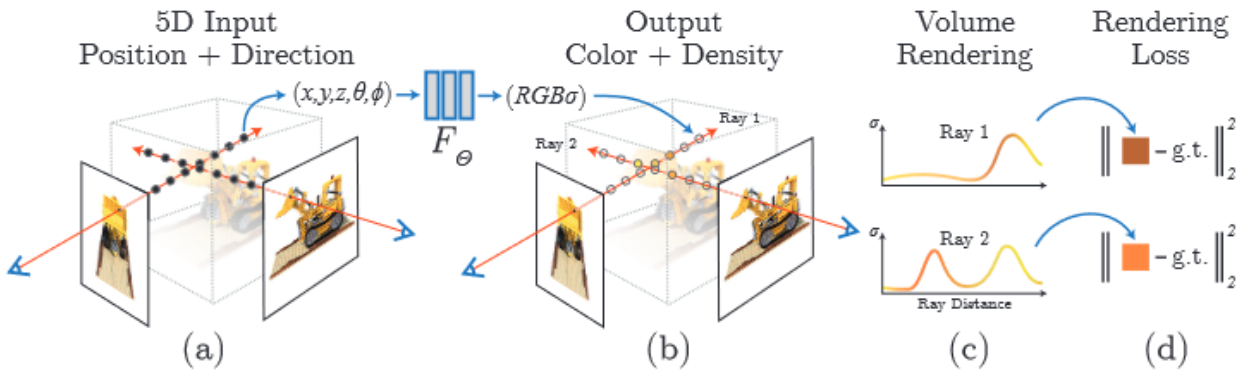

Figure 10: 5D NeRF representation of a scene from a set of input images (Mildenhall et al., 2020)

Figure 10 provides an overview of approach to scene representation using neural radiance fields (NeRF) and the differentiable rendering process. The steps involved are as follows:

(a) Sampling 5D coordinates along camera rays: sample coordinates in 5D, comprising both the location and viewing direction, along the camera rays.

(b) MLP-based generation of colour and volume density: These sampled coordinates are inputted into a multi-layer perceptron (MLP), which generates values for colour and volume density.

(c) Volume rendering for image composition: The obtained colour and density values are combined using volume rendering techniques to compose a final image.

(d) Differentiable rendering for optimization: The rendering procedure is differentiable, enabling us to optimize the scene representation. We achieve this by minimizing the difference between the synthesized images and the ground truth observed images, using the residual as the optimization objective.

By leveraging this differentiable rendering framework, we can refine the NeRF scene representation by iteratively minimizing the discrepancy between the synthesized images and the ground truth, improving the fidelity and accuracy of the rendered scenes.

The aforementioned process involves optimizing a deep, fully-connected multi-layer perceptron (MLP) without the need for convolutional layers. It employs gradient descent as the optimization method to minimize errors between each observed image and all the corresponding views that are rendered from the scene representation.

NeRF Performance

The initial NeRF model exhibited several limitations, including slow training and rendering speeds, its restricted applicability to static scenes, and the lack of flexibility in using a pre-trained NeRF model for different scenes.

To address these challenges, several methods have been developed that build upon NeRF and aim to overcome these issues.

RegNeRF (Niemeyer et al., 2021)36, short for "regularizing neural radiance fields," focuses on view synthesis from sparse inputs. It tackles a specific challenge encountered in NeRF where the performance suffers when the number of input views is limited.

PixelNeRF (Alex W.Y. Yu et al., 2021)4 – The pixelNeRF learning framework enables the prediction of a continuous neural scene representation using one or multiple input images. Unlike traditional NeRF construction methods that require optimizing the representation for each scene individually, pixelNeRF streamlines the process by reducing the reliance on numerous calibrated views and minimizing the computational time involved.

Mega-NeRF (Haithem Turki, Ramanan and Mahadev Satyanarayanan, 2022)21 introduces a learning framework that utilizes NeRFs for constructing interactive 3D environments from extensive visual captures, including structures like buildings and multiple city blocks, primarily obtained through drone-based data collection. Traditionally, NeRFs have been evaluated using single-object scenes, but this approach presents several challenges.

LOLNeRF (Rebain et al., 2022)39 is an acronym for "Learn from One Look" (LOL), a learning approach that leverages NeRF for generative 3D modeling. It focuses on training solely from data consisting primarily of single views of each object. This method aids in generating the corresponding 3D structure of objects in a manner that allows them to be rendered from various viewpoints.

2.6 Summary

In recent years, Deep Learning has made significant advancements in various academic and industrial domains, revolutionizing fields like decision making, image analysis, and image manipulation. These breakthroughs have showcased remarkable autonomy and speed, making them a promising avenue for Visual Simultaneous Localization and Mapping (V-SLAM) systems and SLAM in general.

Cutting-edge deep learning techniques can be leveraged to enhance multiple aspects of the SLAM process, including depth estimation. By incorporating these techniques, future SLAM systems can achieve higher accuracy and faster performance.

DINOv2 exhibits impressive performance without the need for fine-tuning, making it a versatile backbone for various computer vision tasks. By leveraging self-supervision, DINOv2 has the capability to learn from diverse image datasets, enabling it to acquire features that surpass the capabilities of the current standard approach, including depth estimation. Additionally, Meta AI has made their model open-source.

NeRF has illustrated that representing scenes as 5D neural radiance fields, which utilize a multi-layer perceptron (MLP) to generate volume density and view-dependent emitted radiance based on 3D location and 2D viewing direction, yields superior renderings compared to the previous prevailing approach of training deep convolutional networks for discrete voxel representations. While NeRF incorporates a hierarchical sampling strategy to enhance sample efficiency during training and testing, there is still ample room for further exploration of techniques aimed at optimizing and rendering neural radiance fields efficiently.

In summary, the progress made in Deep Learning offers a promising path for advancing VSLAM systems and SLAM as a whole. By integrating state-of-the-art Neural network deep learning techniques, we can enhance SLAM processes to achieve higher accuracy and faster performance.

3. Ethics

This research endeavour employs a secondary dataset, abstaining from the acquisition of fresh video data, to delve into an innovative approach for addressing depth estimation in visual SLAM systems. To enhance feature extraction and depth estimation, this study capitalizes on DINO v2 visual transformer, an advanced deep learning visual transformer made publically available for use by Meta AI. Moreover, experiments will be carried out in a controlled indoor setting, utilizing non-sensitive objects like chairs, wires, and tables.

Our research initiative entails the utilization of publicly accessible benchmark datasets from TUM (RGB-D) (cvg.cit.tum.de, n.d.)13 and Hypersim (Roberts et al., n.d.)40, encompassing real-world visual inputs. While these datasets are devoid of personal information or identifiable individuals, should any such information be encountered, precautionary measures will be implemented to obscure pertinent details, ensuring anonymity and privacy safeguards. Given the non-involvement of human participants in our study, the concept of consent does not apply, and since the datasets are publicly available, confidentiality and data management pose minimal concerns.

Please refer to appendix A for ethics approval of the project.

4. Software Engineering Practise

4.1 Agile Methodology

Throughout the course of the project, we embraced agile software engineering practices. We followed two weeks of sprint span having bi-weekly retrospective meetings with our supervisor. These meetings provided a platform for us to discuss any challenges or roadblocks we encountered, as well as to provide regular updates on our progress. This iterative feedback and suggestion proved invaluable in ensuring that we stayed on track and could address any issues in timely manner.

4.2 Project Sprint Planning

While our initial project plan outlined three distinct sprints, the dynamic nature of working with neural networks prompted us to evolve our approach. The first sprint was dedicated to the meticulous analysis and testing of various datasets. Recognizing the intricacies of neural network behavior, we made a deliberate choice to prioritize dataset exploration before embarking on architecture development. As we progressed to the second sprint, our focus shifted to the careful

experimentation with different neural network architectures. This phase was pivotal in identifying an optimal and stable architecture that aligned with our project goals.

Running in parallel with our sprints, our thesis write-up was an ongoing process that evolved alongside our project's development. This approach allowed us to seamlessly integrate our findings, and outcomes into the thesis.

4.3 Effective Communication and Coordination

A cornerstone of our project's success was effective communication and coordination. Our commitment to keeping our supervisors well-informed at each stage of the project proved pivotal. Regular updates allowed us to leverage their expertise in identifying successful strategies and tackling challenges. Our meetings, a blend of virtual and in-person interactions, provided a platform for discussions and collaborative problem-solving. In-person sessions, in particular, played a crucial role in debugging complex network issues and fine-tuning our approaches.

4.4 Version Control

In terms of version control, we employed Git. Our version control strategy encompassed two primary branches: a master branch housing the stable and tested code, and a development branch where we continuously integrated new and potentially unstable features. This enabled us to maintain a reliable baseline while also facilitating ongoing development and experimentation.

4.5 Code Quality

Emphasizing clean code principles, we ensured that our codebase remained organized, maintainable, and comprehensible. This practice not only streamlined our development process but also enhanced collaboration among team members.

5. Design

This section details the design considerations that underpin the creation of this neural network, offering insight into the core components of neural network.

5.1 Network Architecture

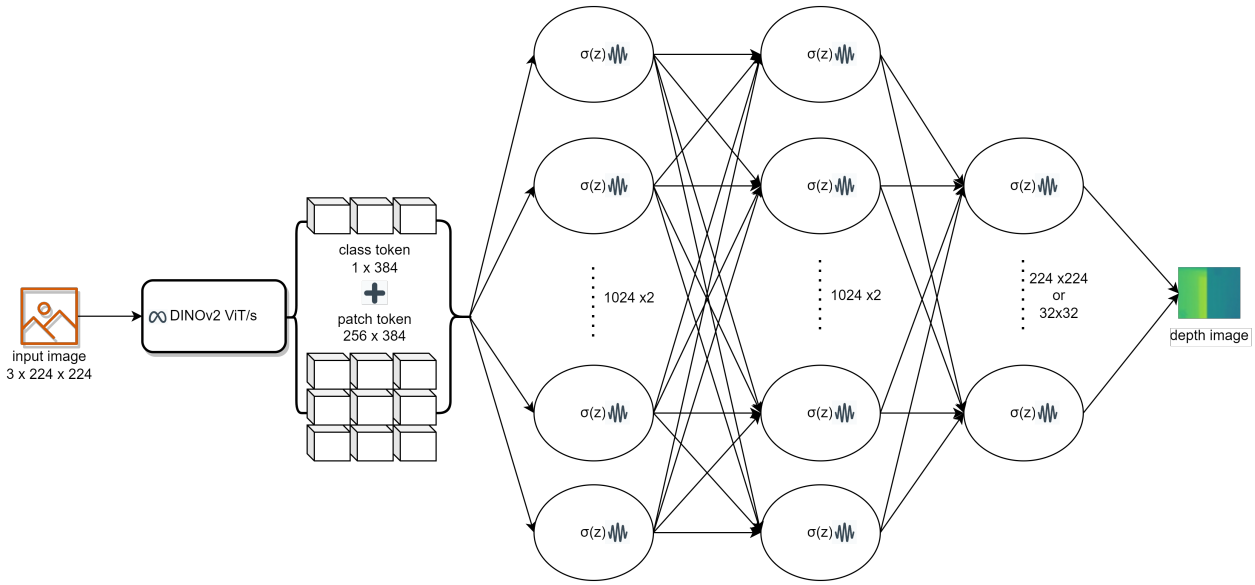

Figure 11: ANN architecture using DINO v2 ViT for depth estimation

The neural network architecture presented in figure 11 is designed for depth estimation using a monocular camera and the DINO v2 visual transformer. This architecture employs three layers of fully connected (FC) layers, followed by non-linear activation functions (Softplus), to transform the input data and ultimately predict depth values. The primary components of neural networks are:

5.1.1 DINO v2 ViT

The Vision Transformer model presented in this work has been pre-trained using the approach detailed in the publication titled 'DINOv2: Learning Robust Visual Features without Supervision.' We utilized ViT-S model in our architecture which is distilled model offering of ViT-g model trained from the ground up. The input to the ViT-S consists of an RGB image, and in response, it generates a class token and patch tokens. The embedding dimension for the ViT-S model is 384. ViT-S model adhere to a Transformer architecture, employing a patch size of 14. For an image with dimensions of 3x224x224, this configuration results in the production of 1x384 class tokens alongside 256x384 patch tokens.

The models have the capacity to accommodate larger images, contingent upon the image dimensions being divisible by the patch size (14). Should this condition not be met, the model will perform cropping to align with the nearest smaller multiple of the patch size.

This ViT-S model serve as foundational vision components, offering versatile features applicable to a range of subsequent objectives. The transformer can be directly employed without the need for further fine-tuning, enabling feature utilization in downstream linear layers. This straightforward approach is the movitation of using DINO v2 for depth estimation task as the image features given by the transformer provide good performance out-of-the-box.

5.1.2 Fully connected Layers

The network's core element is a sequential stack of fully connected three layers, encapsulated within the nn.Sequential container. This stack is designed to learn hierarchical representations from the input data. The structure of the FC layers involves three successive layers of linear transformations (fully connected layers) followed by softplus activation functions with hidden layers having 1024 x2 nodes and output layer having 224 x224 nodes. The input to the first layer is concatenation of class token and patch token output of DINO v2 transformer and expected to be in the dimensions of 257 x 384. FC layer configuration is derived empirically over iterative process.

5.1.3 Activation Function

The network utilises Softplus as the activation function. Softplus serves as a smoothed approximation of the ReLU function, offering the capability to ensure that the output of machine remains consistently positive. Softplus function is defined as

$$\text{Softplus}(x) = \frac{1}{\beta} * \log(1 + \exp(\beta * x))$$

where β is Softplus formulation and defaults to 1.

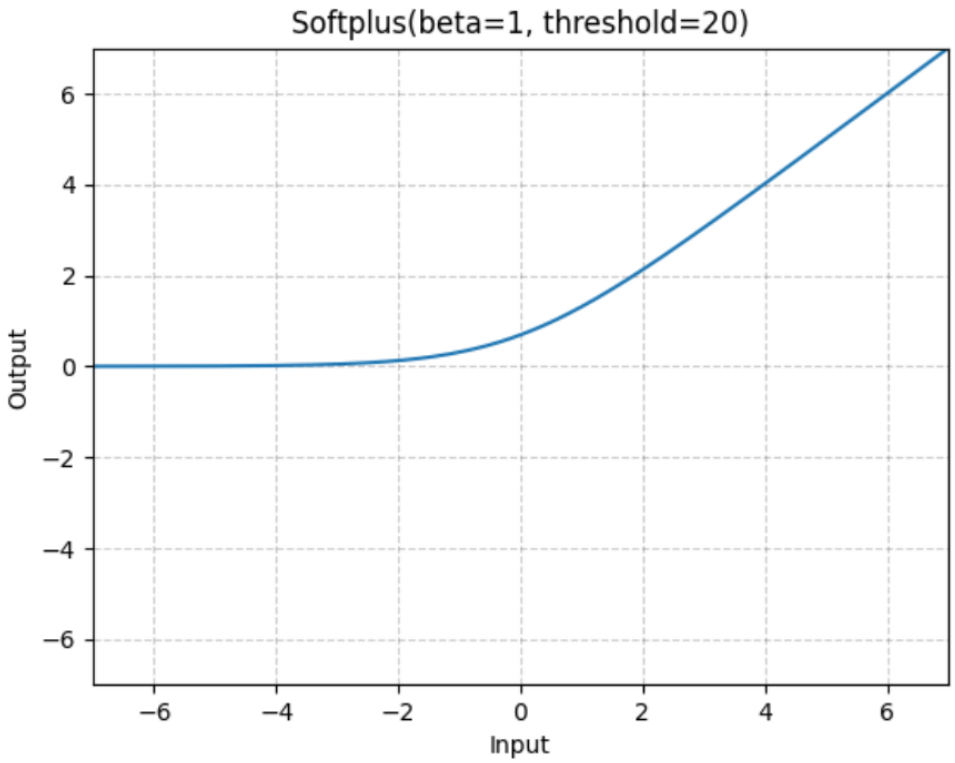

Figure 12: Softplus activation function (pytorch.org, n.d.)

When an input value 'x' is passed through the Softplus activation layer, the function computes the logarithm of the sum of one and the exponential of 'x'. This process ensures that the output value is always positive, as both the exponential and logarithm functions are inherently positive. The result is a gently increasing curve that smoothly approaches larger values, which aids in avoiding the vanishing gradient problem often observed in traditional activation functions like sigmoid or hyperbolic tangent (tanh).

5.1.4 Dataset decision

Our choice was to employ Hypersim (Roberts et al., n.d.)40, a photorealistic synthetic dataset tailored for indoor scenes. This dataset makes use of an extensive collection of synthetic scenes, encompassing a total of 77,400 images across 461 distinct indoor settings. Each image comes with per-pixel annotations and corresponding ground truth geometry. Importantly, Hypersim is a publicly accessible resource, offered for use, and it strictly comprises non-personal material information. For these reasons Hypersim dataset was considered “fit-for-purpose” for this project.

An overview of dataset is shown in figure 13. For each color image (a), the dataset provides grouth truth

- Depth information (b)
- Surface normals (c)
- Instance-level semantic segmentations (d, e)

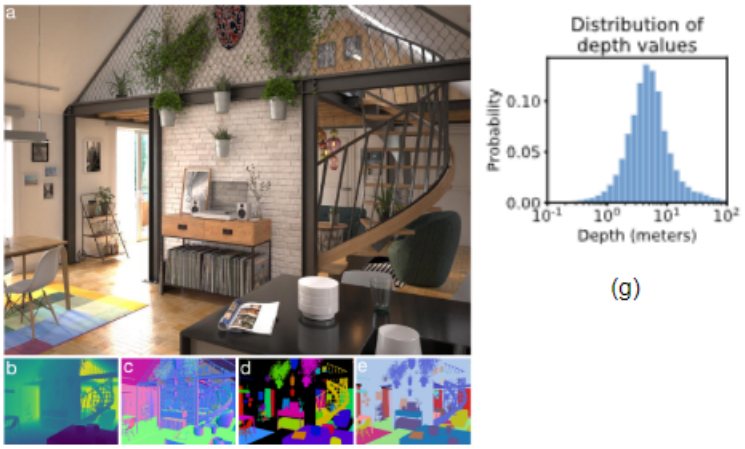

Figure 13: Hypersim dataset overview (Roberts et al., n.d.)

Depth value provided by dataset is log-normally distributed with an average depth of 5.4 meters (fig 13 g) (Roberts et al., n.d.)40

6. Implementation

We trained two linear regression models for depth estimation on Hypersim dataset using Pytorch 1.10.0-cu113 framework on cuda enabled NVIDIA GeForce GTX 1060 machine. The machine has 12 CPUs and 1 GPU available. The two models differentiate on output layer with one having 224x224 perceptrons while other having 32x32 perceptrons on the output layer. Training of both the models is done using Adam optimization with batch size of 32 for total of 500 epochs over 2500 images with learning rate (lr) = $1e^{-4}$. We used Optuna library (Optuna, n.d.)37 for hyper-parameter tuning for initial configuration of the models however the current stable configuration is empirically validated over iterative cycles. Refer Appendix E for hyper-parameter tuning result using Optuna library (Optuna, n.d.)37. Refer Appendix B for details on hardware configuration used for model training.

6.1 Data processing

Per requirement of DINO v2 ViT-S (ai.facebook.com, n.d.)3, the input RGB image is resize to 224x224 to produce visual feature embeddings of 384 dimensions with 1 class token and 256 patch tokens. The class token and patch token is then concatenated and normalised using sklearn.preprocessing.MinMaxScaler module (scikit-learn, n.d.)44 to produce normalised feature inputs for the linear regression models for depth estimation. The flowchart below outlines the data processing steps

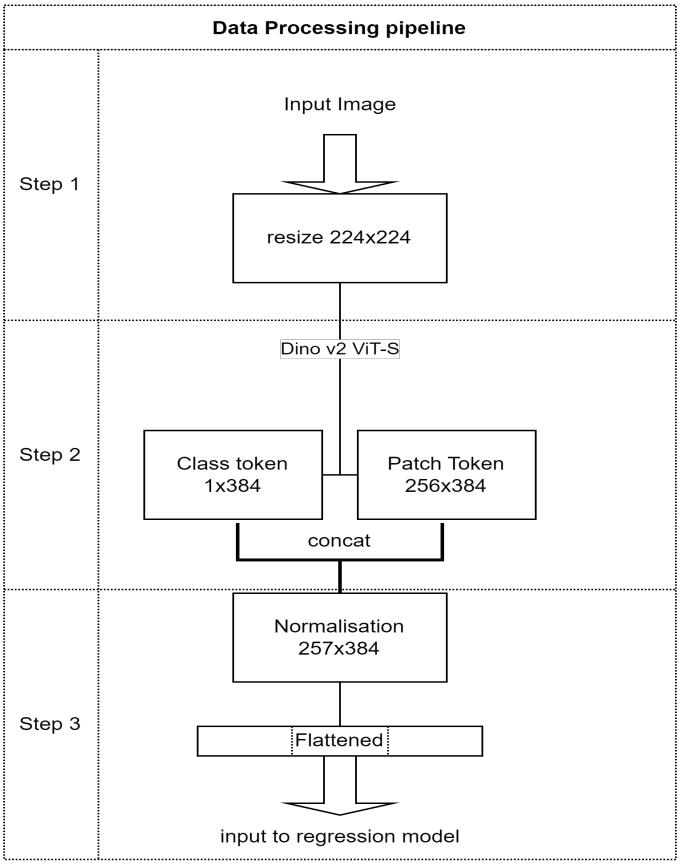

Figure 14: Data processing pipeline

6.2 Loss Component

Our traning model uses custom loss function to penalise MSELoss using Structual Similarity Index (SSIM) of an image. SSIM is calculate using scikit image structural similarity module (scikit-image.org, n.d.)43. The index value closer to 1 is considered to have high similarity between input and output image. Since it is desirable to achieve output image closely similar to input image we considered using SSIM to penalise MSELoss where model fails to capture essential details of an image. SSIM loss is given as

$$SSIM_{Loss} = 1 - SSIM$$

$$Custom\_Loss = MSELoss * \exp(SSIM_{Loss})$$

6.3 Model Training hours

On an average 224x224 regression model took approximately 2800 secs for each 100 epochs of training. The total training time for 500 epochs took nearly 31 hours. Following are details of training hours:

- Epochs 0 to 100 – 21219.41 seconds
- Epochs 100 to 300 – 43642.50 seconds
- Epochs 300 to 400 – 23426.80 seconds
- Epochs 400 to 500 – 23733.25 seconds

Similary, average time calculated for 32x32 depth estimation model is 23500 secs for each 100 epochs and training time it took is nearly 26 hours. The details are as below

- Epochs 0 to 100 – 18396.90 seconds
- Epochs 100 to 300 – 37558.22 seconds
- Epochs 300 to 400 – 19288.97 seconds
- Epochs 400 to 500 – 19053.75 seconds

7. Evaluation

7.1 Dataset comparison

We conducted a comparative analysis between the Hypersim dataset, employed for training and evaluating our depth estimation model, and the well-established TUM dataset (cvg.cit.tum.de, n.d.)13, commonly used for visual odometry evaluation. The histogram comparison of the ground-truth depth images is presented in Figure 15.

The Hypersim dataset displays a pixel value distribution that follows a normal pattern, exhibiting fewer instances of value counts at both the lower and higher bins. In contrast, the TUM dataset demonstrates a notable concentration of values in the lower bins, indicating an uneven distribution in the dataset and sparsity. Furthermore, the value range spans from 0 to 50,000 in the TUM dataset. This inherent sparsity and the broader value range may contribute to extended training times for the model.

On the other hand, the Hypersim dataset stands out due to its absence of sparsity and narrow pixel value range of 0 to 250. This characteristic renders the Hypersim dataset more suitable for the training and evaluation of our depth estimation model.

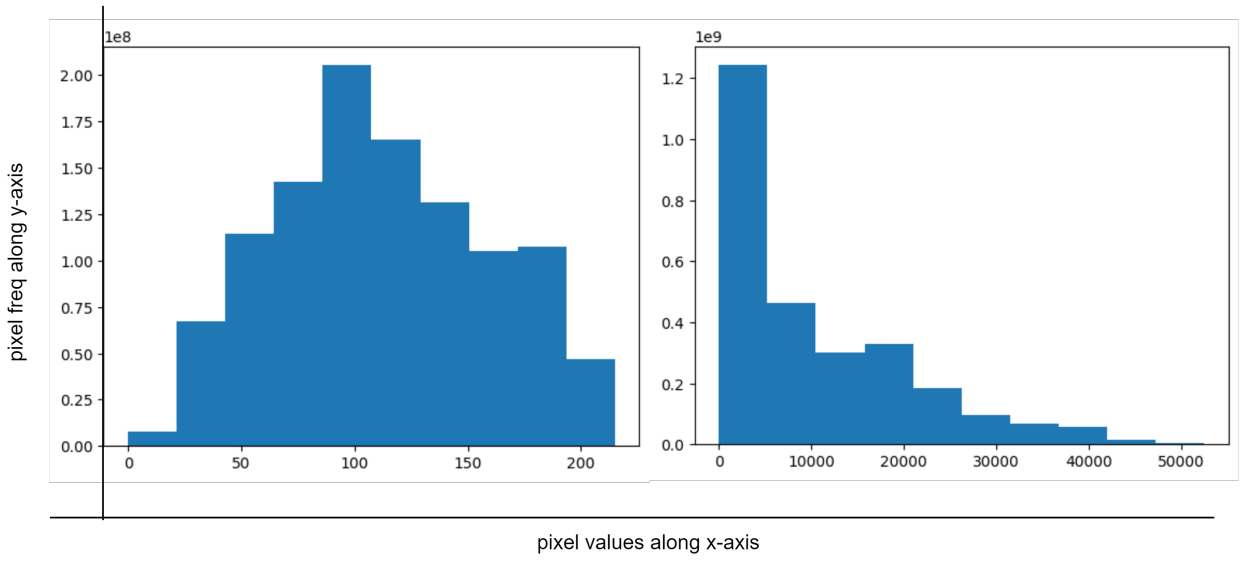

Figure 15: Histogram comparison of Hypersim (left) and TUM (right) datset

7.2 Effectiveness of visual transformer

To study the effectiveness of DINO v2 ViT-S transformer we conducted an experiment of training our 224x224 depth estimation regression model without DINO v2 features. The inference is shown

in Figure 16. It is evident that without use of visual transformer our model fails to predict depth accurately. Figure 17 shows training loss of our model when trained without the transformer. While the training loss seems to exhibit a declining trend over 100 epochs, the model's predictions remain less than optimal.

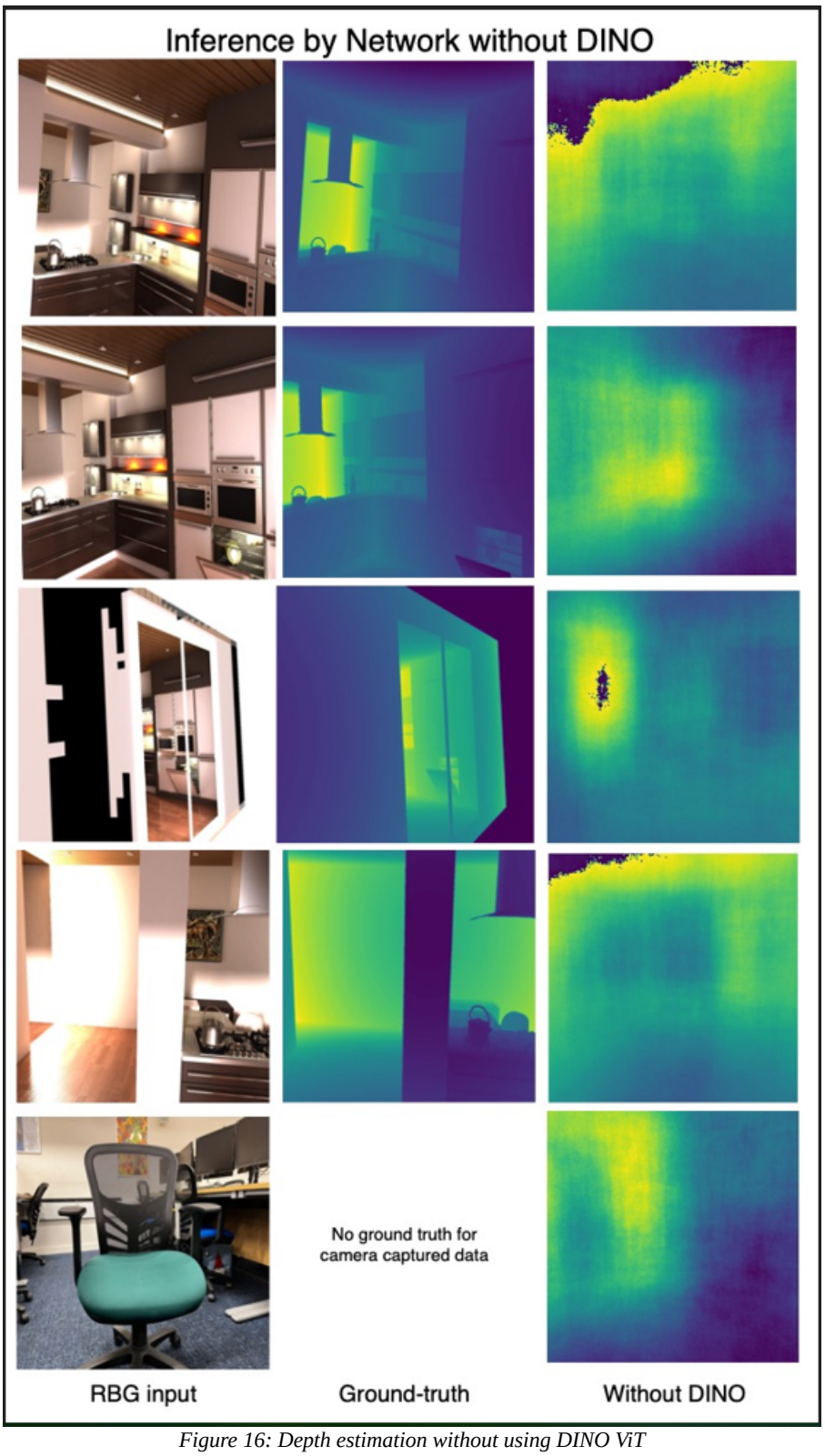

Figure 16: Depth estimation without using DINO ViT

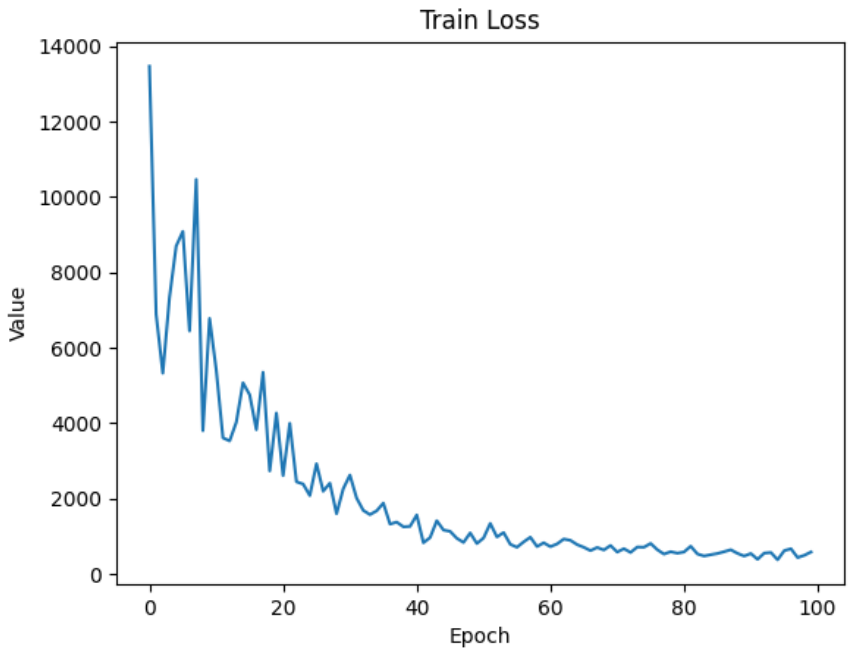

Figure 17: Training loss without DINO v2 transformer

7.3 Evaluating Custom Loss function

We compare our custom loss function which additionally calculates Structural Similarity Index to that of standard MSELoss function. Time taken to train the model using custom loss function is 1126.116 secs having model configuration as 3 Layer Linear model, softplus, Input 257*384, output 32*32, lr=1e-4, run 30 times of 32 batch size

Time taken to train the model using standard MSELoss function is 1086.306 secs having model configuration as 3 Layer Linear model, softplus, Input 257*384, output 32*32, lr=1e-4, run 30 times of 32 batch size

As noted, when utilizing the custom Loss function during training, the model experiences a modest increase of 1.3 seconds for each epoch. This additional time consumption is not excessively overhead in terms of computational resources.

Figure 18 depicts a comparison of Loss plots between Custom Loss and MSE Loss during the training phase. At the end of 30 epochs, Custom Loss yields higher loss values than MSE Loss. This outcome aligns with our expectations, given that Custom Loss incorporates an additional element wherein input is penalized according to its structural similarity index with the ground-truth. It is worth noting that when the $SSIM_{Loss}$ attains a value of 0, the resulting Custom Loss equates to the MSE Loss.

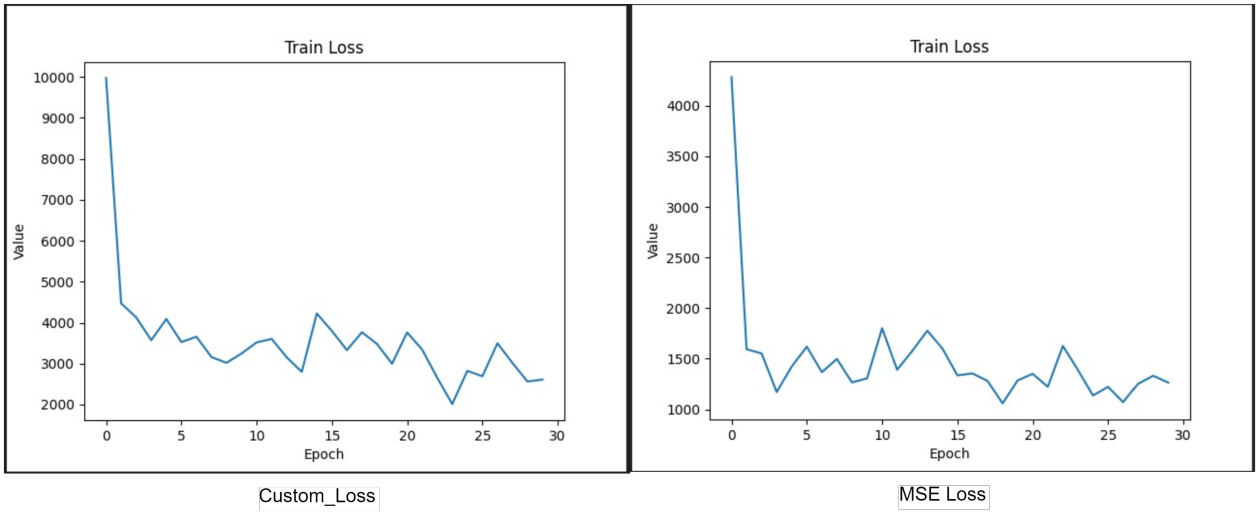

Figure 18: Training Loss - Custom Loss Vs MSE Loss

7.4 Model comparison

We trained two model for depth estimation. The models differ in output layer, one having 32x32 preceptrons on output layer while other having 224x224. Fig. 19 shows the inference of 224x224 model in the middle. The output depth image is smooth and closely follows ground-truth image shown on the right whereas in case of 32x32 model the resultant image (left) is pixelated. Both the models equally perform well and exhibits very little difference in training loss comparison (Fig 20). Applications with need for smaller network model can deploy 32x32 model.

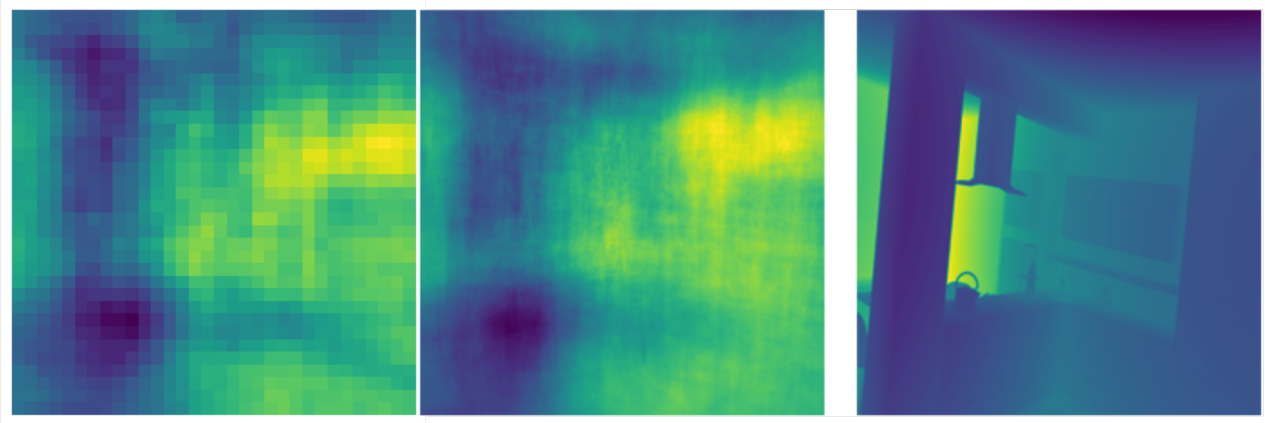

Figure 19: Inference on 32x32 model (L) 224x224 model (M) and ground-truth (R)

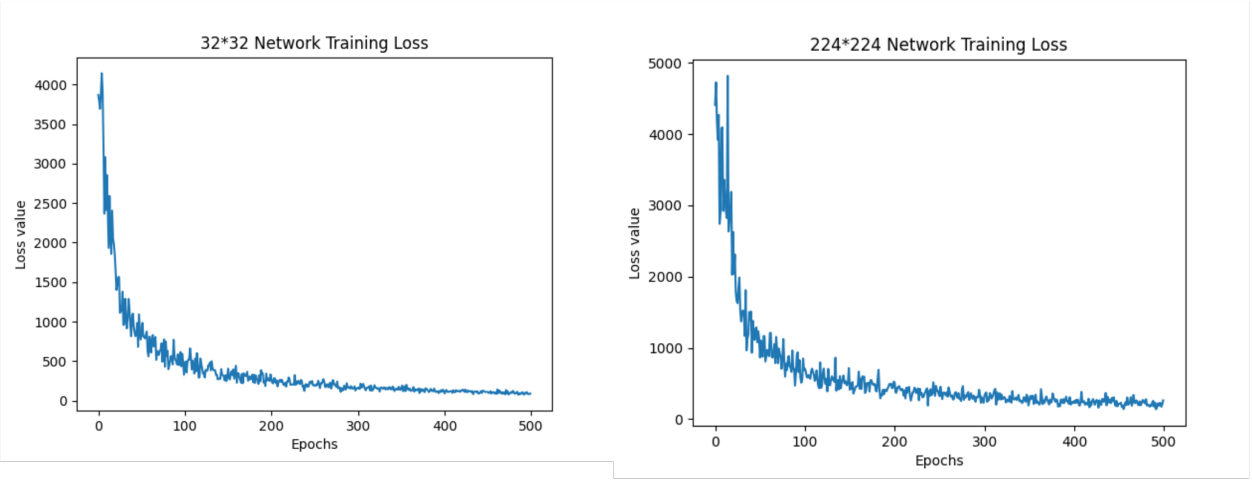

Figure 20: Training loss comparison of 32x32 and 224x224 model

7.5 Depth prediction

Figure 21 presents the visual representation of the depth images predicted by the models when evaluated on the Hypersim test dataset and real-world image respectively. Notably, the depicted depth images utilize a colour gradient scheme to convey the spatial relationships between objects within the scene.

The immediate foreground objects within the images are denoted by a distinct dark blue color, effectively capturing their proximity to the observer. Conversely, objects situated at a greater distance from the viewer are portrayed in a vivid yellow hue, reflecting their increased distance. The intermediate region, encompassing objects falling between the forefront and background, is depicted using a combination of green tones, effectively indicating their intermediate placement within the scene.

This colour-coded visualization captures the depth variations within the images, offering a clear and intuitive representation of the spatial distribution of objects across different depths.

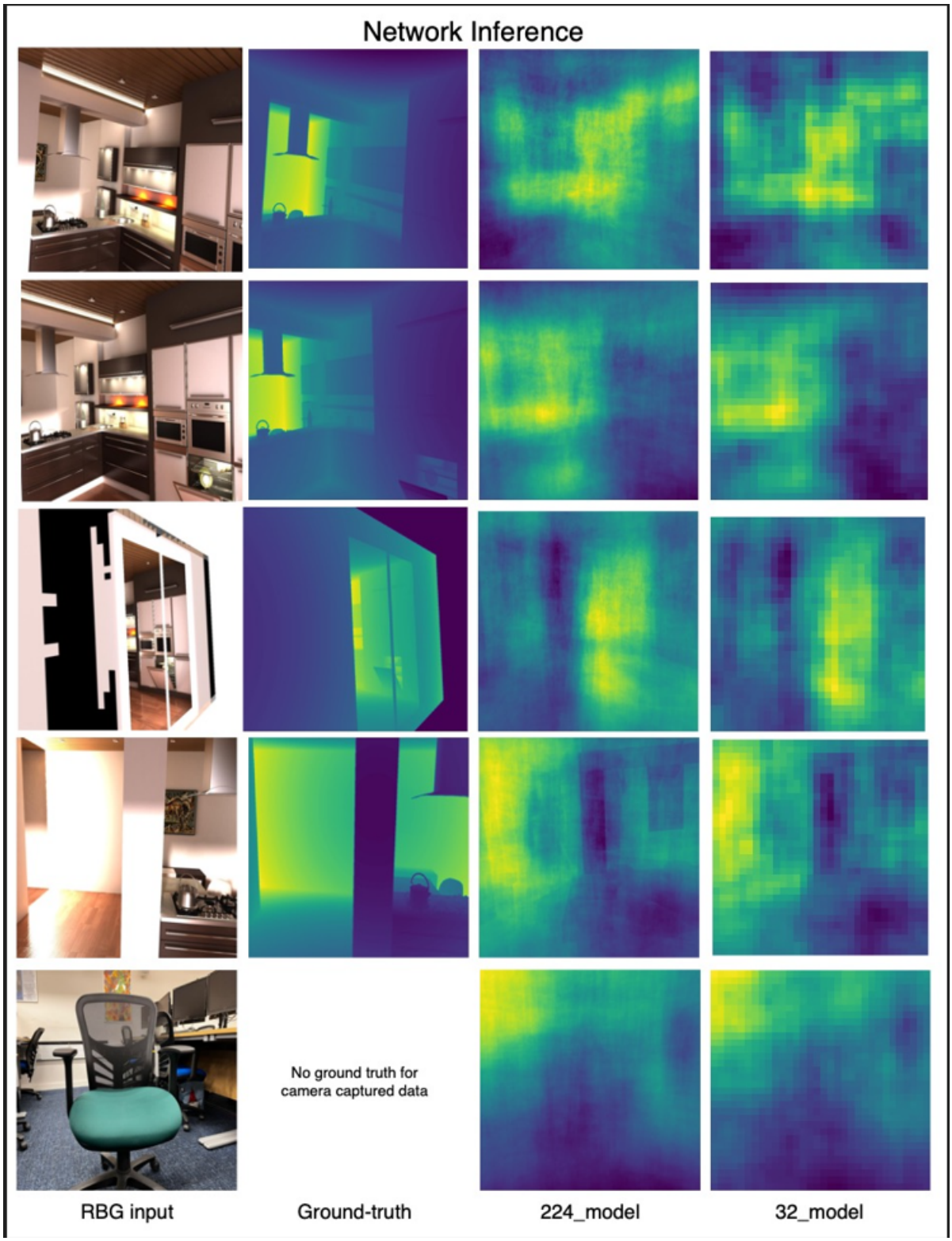

Figure 21: Depth Inference

Following observations are noted for depth prediction on real-world image. Note for real-world image there is no ground-truth depth image available.

Time taken for inference is 102 ms

Predicted pixel values:

```
tensor([[175.1576, 175.1674, 176.6721,  ...,  98.7597, 100.8860, 100.5416],
        [175.9549, 176.2772, 176.4510,  ...,  99.4319, 101.2737,  99.9729],
        [175.1356, 176.4128, 175.5481,  ...,  99.6075, 100.2125,  98.3128],
       ...,
        [ 58.2703,  58.0374,  56.7287,  ...,  48.5954,  48.9769,  49.6903],
        [ 57.6249,  57.0658,  56.2805,  ...,  48.3070,  48.5302,  48.8292],
        [ 56.8583,  55.8544,  56.3548,  ...,  47.4483,  48.0697,  48.2288]],
      grad_fn=<SqueezeBackward1>)
```

Predicted depth estimates in meters:

```
tensor([[3.7092, 3.7094, 3.7413,  ..., 2.0914, 2.1364, 2.1291],
        [3.7261, 3.7329, 3.7366,  ..., 2.1056, 2.1446, 2.1171],
        [3.7088, 3.7358, 3.7175,  ..., 2.1093, 2.1221, 2.0819],
       ...,
        [1.2340, 1.2290, 1.2013,  ..., 1.0291, 1.0372, 1.0523],
        [1.2203, 1.2085, 1.1918,  ..., 1.0230, 1.0277, 1.0340],
        [1.2041, 1.1828, 1.1934,  ..., 1.0048, 1.0179, 1.0213]],
      grad_fn=<DivBackward0>)
```

7.6 Latency

Following observations are recorded as wall time latency (in ms) during inference over ten iterations on input image. Both the models shows relatively low latency on training and test dataset than real-world images. This could be due to higher resolution of camera image.

Table 1: Wall time latency in ms

	Train dataset	Test dataset	Real world Image
32x32 model	[78.0, 73.0, 71.0, 71.0, 69.0, 70.0, 79.0, 77.0, 72.0, 69.0] average = 73.0 ms	[73.0, 68.0, 73.0, 71.0, 73.0, 73.0, 73.0, 73.0, 76.0, 71.0] average = 72.0 ms	[98.0, 97.0, 95.0, 98.0, 97.0, 97.0, 95.0, 94.0, 98.0, 98.0] average = 97.0 ms
224x224 model	[75.0, 80.0, 77.0, 80.0, 78.0, 79.0, 80.0, 79.0, 82.0, 80.0] average = 79.0 ms	[82.0, 80.0, 81.0, 79.0, 78.0, 81.0, 81.0, 81.0, 79.0, 81.0] average = 80.0 ms	[127.0, 131.0, 127.0, 106.0, 107.0, 107.0, 106.0, 106.0, 107.0, 107.0] average = 113.0 ms

The two model is observed to perform equally on train and test dataset however on real world image 32x32 model shows low latency than 224x224 model.

Table 2: Sub-process latency in ms

Model	Sub-process	Train dataset (avg in ms)	Test dataset (avg in ms)	Real world image (avg in ms)
32x32	Image resizing	7.2	7.2	44.1
	ViT processing	26.1	26.1	26.2
	ANN processing	19.7	19.8	19.6
224x224	Image resizing	6.8	7.04	49.2
	ViT processing	25.9	26.3	26.0
	ANN processing	28.5	28.5	28.3

Table 2 provided offers a comprehensive breakdown of processing times (averaged in milliseconds) for different models across various sub-processes, utilizing different input images from training and test datasets, as well as real-world image. 32x32 and 224x224 models are evaluated in this context along with associated sub-processes such as image resizing, ViT (Visual Transformer) processing, and ANN (Artificial Neural Network) processing.

The processing time to re-size real-world image is relatively high in both the model when compared to train and test dataset. This could be due to high image resolution of input image. For both the models, transformer and ANN processing latency do not change significantly with change in input image.

8. Critical Appraisal

The project's depth estimation models have yielded satisfactory results, showcasing the feasibility of the approach taken. However, a notable limitation has been the quality of the depth images produced by these models. The generated depth images exhibit a lack of sharpness, and the edges within the images are not distinctly defined. This deficiency in image quality potentially impacts the model's overall utility and its ability to provide accurate depth information for various applications.

While the project has made considerable progress, further improvement can be made in observed latency. Latency values ranging from 90 to 100 ms have been recorded, indicating room for optimization in terms of real-time processing. Addressing and reducing this latency could

significantly enhance the model's practical applicability, especially in scenarios that require rapid decision-making and responsiveness.

One notable aspect for future enhancement lies in the integration of the developed depth estimation model with Simultaneous Localization and Mapping (SLAM) systems. Although not realized within the current project scope, this integration holds significant potential to extend the utility of the model. Demonstrating this integration would have provided a more comprehensive view of the model's practicality and its value in real-world scenarios.

Another avenue for improvement centers around the model's loss function, which is a fundamental component of training. While the current project has made strides in implementing a custom loss function, there remains room for refining and optimizing this aspect further. Future work dedicated to enhancing the loss function could potentially lead to improved model performance and accuracy.

It's crucial to acknowledge that the project has brought to light a potential limitation related to the training of the model. Specifically, the model's performance might be hindered if the ground-truth depth images possess sparsity. This emphasizes the importance of robust data preprocessing and handling techniques, which could be explored in future iterations of the project to ensure the model's versatility across various data scenarios.

9. Conclusions

Our primary objectives, aimed at researching the forefront of SLAM methodologies and identifying avenues for innovation, have been successfully achieved. Our rigorous research and thorough analysis have provided us with an understanding of the current SLAM systems. This exploration led us to pinpoint depth estimation as a critical area for enhancement within the SLAM system framework.

In our pursuit of advancing depth estimation, we undertook the design and implementation of robust neural network architectures. Leveraging the cutting-edge capabilities of the DINO v2 visual transformer, we harnessed the potential for significant improvements in depth estimation. The resultant model not only demonstrates stability but also yields satisfactory results, all while maintaining low latency.

The network's capability was rigorously examined within the context of indoor environments, showcasing its potential for seamless integration into real-world applications.

While our project was carried out within a constrained time frame, it has illustrated how the strategic application of visual transformers in neural networks can yield transformative outcomes, specifically in the realm of depth estimation. By effectively laying this groundwork, we pave the way for future projects to harness the power of visual transformers within the broader context of SLAM systems.

In conclusion, the project has yielded valuable insights and outcomes, including satisfactory results from depth estimation models. However, previously mentioned challenges are some of the aspects that can guide the project's future trajectory towards greater applicability.

9.1 Future Work

While the current project has made significant effort in enhancing depth estimation through the integration of visual transformer, there remain promising avenues for future exploration and refinement. The limitations identified within the project can be effectively addressed through the following avenues:

1. Enhanced Model Refinement for Improved Generalization

Refining the existing model through meticulous fine-tuning stands out as a critical pathway for advancing the project's capabilities. By systematically adjusting and optimizing the model's parameters, the model can be enhanced to generalize better. This refinement process has the potential to produce depth images characterized by sharper details and well-defined edges. By addressing the current limitations in image quality, this approach would significantly elevate the overall performance of the depth estimation model.

2. Latency Optimization for Real-Time Applications

Improving the latency of the network represents another promising direction for future work. Concentrating on minimizing the processing time of the network would render it more suitable for real-time applications. The optimization of latency would extend the practical utility of the depth estimation model, making it a valuable tool for real-world, dynamic environments.

3. Integration of Deep Learning ViT-Based Techniques in SLAM systems

The groundwork laid through our efforts holds the promise of influencing the broader SLAM landscape. Future projects can explore the integration of deep learning ViT-based techniques within the SLAM pipeline. By doing so, the system's robustness and performance could be significantly enhanced, translating our endeavours into real-world applications.

References

1. Abaspur Kazerouni, I., Dooly, G. and Toal, D. (2020). Underwater Image Enhancement and Mosaicking System Based on A-KAZE Feature Matching. *Journal of Marine Science and Engineering*, [online] 8(6), p.449. doi:https://doi.org/10.3390/jmse8060449.

2. Abaspur Kazerouni, I., Fitzgerald, L., Dooly, G. and Toal, D. (2022). A survey of state-of-the-art on visual SLAM. *Expert Systems with Applications*, [online] 205, p.117734. doi:https://doi.org/10.1016/j.eswa.2022.117734.

3. ai.facebook.com. (n.d.). *DINOv2: State-of-the-art computer vision models with self-supervised learning*. [online] Available at: https://ai.facebook.com/blog/dino-v2-computer-vision-self-supervised-learning/

4. Alex W.Y. Yu, Ye, V., Tancik, M. and Kanazawa, A. (2021). pixelNeRF: Neural Radiance Fields from One or Few Images. doi:https://doi.org/10.1109/cvpr46437.2021.00455.

5. Bay, H., Ess, A., Tuytelaars, T. and Van Gool, L. (2008). Speeded-Up Robust Features (SURF). *Computer Vision and Image Understanding*, [online] 110(3), pp.346–359. doi:https://doi.org/10.1016/j.cviu.2007.09.014.

6. Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., Reid, I. and Leonard, J.J. (2016). Past, Present, and Future of Simultaneous Localization And Mapping: Towards the Robust-Perception Age. *IEEE Transactions on Robotics*, [online] 32(6), pp.1309–1332. doi:https://doi.org/10.1109/TRO.2016.2624754.

7. Calonder, M., Lepetit, V., Ozuysal, M., Trzcinski, T., Strecha, C. and Fua, P. (2012). BRIEF: Computing a Local Binary Descriptor Very Fast. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(7), pp.1281–1298. doi:https://doi.org/10.1109/tpami.2011.222.

8. Campos, C., Elvira, R., Rodriguez, J.J.G., M. Montiel, J.M. and D. Tardos, J. (2021). ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual–Inertial, and Multimap SLAM. *IEEE Transactions on Robotics*, pp.1–17. doi:https://doi.org/10.1109/tro.2021.3075644.

9. Caron, M., Touvron, H., Misra, I., Jégou, H., Mairal, J., Bojanowski, P. and Joulin, A. (2021). Emerging Properties in Self-Supervised Vision Transformers. *arXiv:2104.14294 [cs]*. [online] Available at: https://arxiv.org/abs/2104.14294.

10. Civera, J., Davison, A.J. and Montiel, J. (2008). Inverse Depth Parametrization for Monocular SLAM. *IEEE Transactions on Robotics*, 24(5), pp.932–945. doi:https://doi.org/10.1109/tro.2008.2003276.

11. Cummins, M. and Newman, P. (2008). FAB-MAP: Probabilistic Localization and Mapping in the Space of Appearance. *The International Journal of Robotics Research*, 27(6), pp.647–665. doi:https://doi.org/10.1177/0278364908090961.

12. Cummins, M. and Newman, P. (2010). Appearance-only SLAM at large scale with FAB-MAP 2.0. *The International Journal of Robotics Research*, 30(9), pp.1100–1123. doi:https://doi.org/10.1177/0278364910385483.

13. cvg.cit.tum.de. (n.d.). *Computer Vision Group - Datasets - RGB-D SLAM Dataset and Benchmark*. [online] Available at: https://cvg.cit.tum.de/data/datasets/rgbd-dataset.

14. Dalal, N. and Triggs, B. (2005). Histograms of Oriented Gradients for Human Detection. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, 1. doi:https://doi.org/10.1109/cvpr.2005.177.

15. Datagen. (n.d.). *Neural Radiance Field (NeRF): A Gentle Introduction*. [online] Available at: https://datagen.tech/guides/synthetic-data/neural-radiance-field-nerf/.

16. Davison, A.J., Reid, I.D., Molton, N.D. and Stasse, O. (2007). MonoSLAM: Real-Time Single Camera SLAM. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, [online] 29(6), pp.1052–1067. doi:https://doi.org/10.1109/tpami.2007.1049.

17. Dryanovski, I., Valenti, R. and Xiao, J. (2013). Fast visual odometry and mapping from RGB-D data. doi:https://doi.org/10.1109/icra.2013.6630889.

18. Durrant-Whyte, H. and Bailey, T. (2006). Simultaneous localization and mapping: part I. *IEEE Robotics & Automation Magazine*, [online] 13(2), pp.99–110. doi:https://doi.org/10.1109/mra.2006.1638022.

19. Engel, J., Koltun, V. and Cremers, D. (2018). Direct Sparse Odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, [online] 40(3), pp.611–625. doi:https://doi.org/10.1109/tpami.2017.2658577.

20. Engel, J., Schöps, T. and Cremers, D. (2014). LSD-SLAM: Large-Scale Direct Monocular SLAM. *Computer Vision – ECCV 2014*, pp.834–849. doi:https://doi.org/10.1007/978-3-319-10605-2_54.

21. Haithem Turki, Ramanan, D. and Mahadev Satyanarayanan (2022). Mega-NeRF: Scalable Construction of Large-Scale NeRFs for Virtual Fly- Throughs. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. doi:https://doi.org/10.1109/cvpr52688.2022.01258.

22. Hess, W., Kohler, D., Rapp, H. and Andor, D. (2016). *Real-time loop closure in 2D LIDAR SLAM*. [online] IEEE Xplore. doi:https://doi.org/10.1109/ICRA.2016.7487258.

23. Iman Abaspur Kazerouni and Javad Haddadnia (2014). A mass classification and image retrieval model for mammograms. 62(7), pp.353–357. doi:https://doi.org/10.1179/1743131x13y.0000000054.

24. Jia, G., Li, X., Zhang, D., Xu, W., Lv, H., Shi, Y. and Cai, M. (2022). Visual-SLAM Classical Framework and Key Techniques: A Review. *Sensors*, [online] 22(12), p.4582. doi:https://doi.org/10.3390/s22124582.

25. Klein, G. and Murray, D. (n.d.). *Parallel Tracking and Mapping for Small AR Workspaces*. [online] Available at: https://www.robots.ox.ac.uk/~gk/publications/KleinMurray2007ISMAR.pdf.

26. Li, R., Wang, S. and Gu, D. (2021). DeepSLAM: A Robust Monocular SLAM System With Unsupervised Deep Learning. *IEEE Transactions on Industrial Electronics*, [online] 68(4), pp.3577–3587. doi:https://doi.org/10.1109/TIE.2020.2982096.

27. Lowe, D.G. (2004). Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2), pp.91–110. doi:https://doi.org/10.1023/b:visi.0000029664.99615.94.

28. Macario Barros, A., Michel, M., Moline, Y., Corre, G. and Carrel, F. (2022). A Comprehensive Survey of Visual SLAM Algorithms. *Robotics*, 11(1), p.24. doi:https://doi.org/10.3390/robotics11010024.

29. Memon, A.R., Wang, H. and Hussain, A. (2020). Loop closure detection using supervised and unsupervised deep neural networks for monocular SLAM systems. *Robotics and Autonomous Systems*, 126, p.103470. doi:https://doi.org/10.1016/j.robot.2020.103470.

30. Merzlyakov, A. and Macenski, S. (2021). A Comparison of Modern General-Purpose Visual SLAM Approaches. *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. doi:https://doi.org/10.1109/iros51168.2021.9636615.

31. Mildenhall, B., Srinivasan, P.P., Tancik, M., Barron, J.T., Ramamoorthi, R. and Ng, R. (2020). NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. *arXiv:2003.08934 [cs]*. [online] Available at: https://arxiv.org/abs/2003.08934.

32. Milz, S., Arbeiter, G., Witt, C., Abdallah, B. and Yogamani, S. (2018). *Visual SLAM for Automated Driving: Exploring the Applications of Deep Learning*. [online] IEEE Xplore. doi:https://doi.org/10.1109/CVPRW.2018.00062.

33. Mur-Artal, R., Montiel, J.M.M. and Tardos, J.D. (2015). ORB-SLAM: A Versatile and Accurate Monocular SLAM System. *IEEE Transactions on Robotics*, [online] 31(5), pp.1147–1163. doi:https://doi.org/10.1109/tro.2015.2463671.

34. Mur-Artal, R. and Tardos, J.D. (2017a). ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras. *IEEE Transactions on Robotics*, 33(5), pp.1255–1262. doi:https://doi.org/10.1109/tro.2017.2705103.

35. Mur-Artal, R. and Tardos, J.D. (2017b). Visual-Inertial Monocular SLAM With Map Reuse. *IEEE Robotics and Automation Letters*, [online] 2(2), pp.796–803. doi:https://doi.org/10.1109/lra.2017.2653359.

36. Niemeyer, M., Barron, J.T., Mildenhall, B., Mehdi, Geiger, A. and Radwan, N. (2021). RegNeRF: Regularizing Neural Radiance Fields for View Synthesis from Sparse Inputs. doi:https://doi.org/10.1109/cvpr52688.2022.00540.

37. Optuna. (n.d.). *Optuna - A hyperparameter optimization framework*. [online] Available at: https://optuna.org/

38. pytorch.org. (n.d.). *PyTorch documentation — PyTorch 1.10 documentation*. [online] Available at: https://pytorch.org/docs

39. Rebain, D., Matthews, M., Kwang Moo Yi, Dmitry Lagun and Tagliasacchi, A. (2022). LOLNeRF: Learn from One Look. doi:https://doi.org/10.1109/cvpr52688.2022.00161.

40. Roberts, M., Ramapuram, J., Ranjan, A., Kumar, A., Bautista, M., Paczan, N., Webb, R. and Susskind Apple, J. (n.d.). *Hypersim: A Photorealistic Synthetic Dataset for Holistic Indoor Scene Understanding*. [online] Available at: https://arxiv.org/pdf/2011.02523.pdf.

41. Rosten, E. and Drummond, T. (2006). Machine Learning for High-Speed Corner Detection. *Computer Vision – ECCV 2006*, pp.430–443. doi:https://doi.org/10.1007/11744023_34.

42. Rublee, E., Rabaud, V., Konolige, K. and Bradski, G. (2011). *ORB: An efficient alternative to SIFT or SURF*. [online] IEEE Xplore. doi:https://doi.org/10.1109/ICCV.2011.6126544.

43. scikit-image.org. (n.d.). *Structural similarity index — skimage 0.21.0 documentation*. [online] Available at: https://scikit-image.org/docs/stable/auto_examples/transform/plot_ssim.html

44. scikit-learn. (n.d.). *sklearn.preprocessing.MinMaxScaler*. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html#sklearn.preprocessing.MinMaxScaler.fit

45. Seiskari, O., Pekka Rantalankila, Juho Kannala, Ylilammi, J., Esa Rahtu and Solin, A. (2022). HybVIO: Pushing the Limits of Real-time Visual-inertial Odometry. doi:https://doi.org/10.1109/wacv51458.2022.00036.

46. Sturm, J., Engelhard, N., Endres, F., Burgard, W. and Cremers, D. (2012). A benchmark for the evaluation of RGB-D SLAM systems. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. doi:https://doi.org/10.1109/iros.2012.6385773

47. Sumikura, S., Shibuya, M. and Sakurada, K. (2023). OpenVSLAM: A Versatile Visual SLAM Framework. doi:https://doi.org/10.1145/3343031.3350539.

48. Taketomi, T., Uchiyama, H. and Ikeda, S. (2017). Visual SLAM algorithms: a survey from 2010 to 2016. *IPSJ Transactions on Computer Vision and Applications*, 9(1). doi:https://doi.org/10.1186/s41074-017-0027-2.

49. Tang, S., Zhu, Q., Chen, W., Darwish, W., Wu, B., Hu, H. and Chen, M. (2016). Enhanced RGB-D Mapping Method for Detailed 3D Indoor and Outdoor Modeling. *Sensors*, 16(10), p.1589. doi:https://doi.org/10.3390/s16101589.

50. Thrun, S., Burgard, W. and Fox, D. (2010). *Probabilistic robotics*. Cambridge, Mass.: Mit Press.

51. Whelan, T.A., Leutenegger, S., Salas-Moreno, R.F., Glocker, B. and Davison, A.J. (2015). ElasticFusion: Dense SLAM Without A Pose Graph. doi:https://doi.org/10.15607/rss.2015.xi.001.

52. Xu, Y., Zhang, S., Li, J., Liu, H. and Zhu, H. (2021). Extracting Terrain Texture Features for Landform Classification Using Wavelet Decomposition. *ISPRS International Journal of Geo-Information*, 10(10), p.658. doi:https://doi.org/10.3390/ijgi10100658.

53. Zaffar, M., Ehsan, S., Stolkin, R. and Maier, K.M. (2018). Sensors, SLAM and Long-term Autonomy: A Review. *2018 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. doi:https://doi.org/10.1109/ahs.2018.8541483.

Appendix A – Ethics Approval

University of St Andrews | FOUNDED 1413

School of Computer Science Ethics Committee

03 July 2023

Dear Ashwin,

Thank you for submitting your ethical application which was considered by the School Ethics Committee.

The School of Computer Science Ethics Committee, acting on behalf of the University Teaching and Research Ethics Committee (UTREC), has approved this application:

Approval Code:	CS17127	**Approved on:**	03.07.23	**Approval Expiry:**	03.07.28
Project Title:	Enhancing Long-Term Estimation in Visual SLAM using DINO v2: Deep Learning-based Preprocessing and Depth Estimation for Drift Correction and Robust Mapping				
Researcher(s):	Ashwin Kashyap Haresamudram Krishna Rao and Kumar Gaurav Atram				
Supervisor(s):	Dr Christopher Stone and Dr Juan Ye				

The following supporting documents are also acknowledged and approved:

1. Application Form

Approval is awarded for 5 years, see the approval expiry data above.

If your project has not commenced within 2 years of approval, you must submit a new and updated ethical application to your School Ethics Committee.

If you are unable to complete your research by the approval expiry date you must request an extension to the approval period. You can write to your School Ethics Committee who may grant a discretionary extension of up to 6 months. For longer extensions, or for any other changes, you must submit an ethical amendment application.

You must report any serious adverse events, or significant changes not covered by this approval, related to this study immediately to the School Ethics Committee.

Approval is given on the following conditions:

- that you conduct your research in line with:
  - the details provided in your ethical application
  - the University's Principles of Good Research Conduct
  - the conditions of any funding associated with your work
- that you obtain all applicable additional documents (see the 'additional documents' webpage for guidance) before research commences.

You should retain this approval letter with your study paperwork.

Yours sincerely,

Wendy Boyter

SEC Administrator

School of Computer Science Ethics Committee
Dr Olexandr Konovalov/Convenor, Jack Cole Building, North Haugh, St Andrews, Fife, KY16 9SX
Telephone: 01334 463273 Email: ethics-cs@st-andrews.ac.uk
The University of St Andrews is a charity registered in Scotland: No SC013532

Appendix B – H/w Configuration

Lab machine hardware configuration used for ANN training

Architecture:	x86_64
CPU op-mode(s):	32-bit, 64-bit
CPU(s):	12
Model name:	12th Gen Intel(R) Core(TM) i5-12400
Thread(s) per core:	2
CPU max MHz:	5600.0000
CPU min MHz:	800.0000
GPU model:	NVIDIA GeForce GTX 1060
Driver Version:	525.116.04
CUDA Version:	12.0

Appendix C – User Guide

To run the inference for depth estimation

1. Navigate into the submitted folder directory.

2. Download the '**.pt**' files of the networks from https://universityofstandrews907-my.sharepoint.com/:f:/g/personal/akhkr1_st-andrews_ac_uk/Eh0rOhTeM09Cmw3SIr13z2wBg8Vjpk2lDX7HbvJwuQylsA?e=8NI66j

and save them in the submitted folder.

3. Install the dependencies by running the command "pip install -r requirements.txt" and launch Jupyter notebook in the "submission" folder.

4. Open the python notebook named '**Inference.ipynb**'.

5. Enter the path of ".pt" file in 'model_name' as a string either '224_model' or '32_model'.

6. Enter the op_size (output size should match the model chosen) as 224 (for 224_model) or 32 (for 32_model) (integer).

7. Enter the input_image_path as a string with the path to the input RGB image file.

8. If the inference is only with input image then the next variable target_image_path can be left blank and run cells 1, 2, 3.

9. If the target image is available for inference, enter the target_image_path as a string with the path to target depth image and run cells 1, 2, 4.

10. The inference displays predicted pixel values, depth converted values for each pixel, plots the predicted and input images, prints loss if target is also given and plots the ground-truth images as well.

Appendix D – Project Structure

```
Training code for 224_model                                          ├── 224_model.py
Training code for 32 model                                           ├── 32_model.py
Training code for network without DINO                               ├── without_dino.py
Instructions to run inference                                        ├── README.md
Serialised scaler object                                             ├── scaler.pkl
Sample test image for inference                                      ├── test_input.jpg
Sample test target for inference                                     ├── test_target.png
Sample camera image for inference                                    ├── chair.jpeg
Dependencies list to run inference                                   ├── requirements.txt
Interactive python notebook for running inference                    ├── inference.ipynb
Folder contains code of experimentations                             ├── experimental_networks
        Code for 4 layers network                                    │   ├── 4_layers_network.py
        Code for feeding grayscale image to DINO                     │   ├── grayscale_to_dino.py
        Code for [64,64] grayscale target                            │   ├── gs_64.py
        Code that feeds camera input to DINO network                 │   ├── interface_dino.py
        Code for 5_5_5 network                                       │   ├── original_size-2.py
        Code for generating downsized depth estimates                │   ├── pred_dataset.py
        Code for super resolution model                              │   ├── super_res.py
        Code for training super resolution model                     │   └── train_super_res.py
Folder contains miscellaneous files for reference                    └── misc
        Code used for Histogram generation                               ├── histogram.ipynb
        Code used for latency evaluation                                 ├── latency_check.ipynb
        Code used for normalisation                                      ├── normalisation.ipynb
        Folder contains inference drawn from 224 model with test data    ├── 224_test
                                                                         │   ├── 224_network_test.txt
                                                                         │   ├── input_0.jpg
                                                                         │   ├── input_80.jpg
                                                                         │   ├── input_90.jpg
                                                                         │   ├── pred_0.png
                                                                         │   ├── pred_80.png
                                                                         │   ├── pred_90.png
                                                                         │   ├── target_0.png
                                                                         │   ├── target_80.png
                                                                         │   └── target_90.png
        Code for bayesian optimisation using optuna                      ├── bayes_optim_depth.py
        Code for computing features dataset using SIFT                   ├── compute_pyramid_and_sift.ipynb
        Draft for the same                                               ├── compute_pyramid_and_sift_draft.ipynb
        Pdf which shows the data flow                                    ├── data_flow.pdf
        DINO outputs for experimenting with it                           ├── dino_output.txt
        Plotting the attention applied by DINO                           ├── dinov2_with_image_experiment.ipynb
        Experimenting with NYU labeled dataset                           ├── labeled_dataset.ipynb
        Code for syncing the RGB images with corresponding targets       ├── sync.ipynb
        Code for merging the tum dataset                                 └── tum_merge.ipynb
```

Appendix E – Hyper-parameter tuning

Following are Hyper-parameter tuning results using Optuna library

States:
Number of finished trials: 100
Number of pruned trials: 90
Number of completed trials: 10
Best trial:
Value: 1189.0291748046875
Params:
n_layers: 7
n_units_l0: 635
n_units_l1: 450
n_units_l2: 653
n_units_l3: 624
n_units_l4: 259
n_units_l5: 607
n_units_l6: 203
dropout_l6: 0.29003892869665815
lr: 0.019440328519667247

[I 2023-07-18 17:23:12,581] Trial 0 finished with value: 4580.38525390625 and parameters: {'n_layers': 9, 'n_units_l0': 252, 'n_units_l1': 648, 'n_units_l2': 687, 'n_units_l3': 584, 'n_units_l4': 610, 'n_units_l5': 395, 'n_units_l6': 487, 'n_units_l7': 680, 'n_units_l8': 255, 'dropout_l8': 0.36448631009458143, 'lr': 1.7839127381220633e-05}. Best is trial 0 with value: 4580.38525390625.
[I 2023-07-18 17:24:36,249] Trial 1 finished with value: 2623.03564453125 and parameters: {'n_layers': 5, 'n_units_l0': 799, 'n_units_l1': 578, 'n_units_l2': 516, 'n_units_l3': 399, 'n_units_l4': 777, 'dropout_l4': 0.4545033843297115, 'lr': 0.00012508107579489193}. Best is trial 1 with value: 2623.03564453125.
[I 2023-07-18 17:25:57,330] Trial 2 finished with value: 1376.8817138671875 and parameters: {'n_layers': 11, 'n_units_l0': 490, 'n_units_l1': 740, 'n_units_l2': 362, 'n_units_l3': 353, 'n_units_l4': 408, 'n_units_l5': 660, 'n_units_l6': 469, 'n_units_l7': 390, 'n_units_l8': 493, 'n_units_l9': 236, 'n_units_l10': 249, 'dropout_l10': 0.2403835117366536, 'lr': 0.01685049410988646}. Best is trial 2 with value: 1376.8817138671875.
[I 2023-07-18 17:27:20,863] Trial 3 finished with value: 1050.91064453125 and parameters: {'n_layers': 8, 'n_units_l0': 288, 'n_units_l1': 344, 'n_units_l2': 603, 'n_units_l3': 275, 'n_units_l4': 494, 'n_units_l5': 613, 'n_units_l6': 477, 'n_units_l7': 718, 'dropout_l7': 0.46113381451115626, 'lr': 0.0003890964787601002}. Best is trial 3 with value: 1050.91064453125.
[I 2023-07-18 17:28:43,564] Trial 4 finished with value: 1148.2685546875 and parameters:

{'n_layers': 8, 'n_units_l0': 249, 'n_units_l1': 736, 'n_units_l2': 439, 'n_units_l3': 701, 'n_units_l4': 519, 'n_units_l5': 597, 'n_units_l6': 701, 'n_units_l7': 461, 'dropout_l7': 0.4880974477418865, 'lr': 5.489730931988637e-05}. Best is trial 3 with value: 1050.91064453125.
[I 2023-07-18 17:28:44,465] Trial 5 pruned.
[I 2023-07-18 17:28:44,509] Trial 6 pruned.
[I 2023-07-18 17:28:44,552] Trial 7 pruned.
[I 2023-07-18 17:28:45,408] Trial 8 pruned.
[I 2023-07-18 17:28:45,454] Trial 9 pruned.
[I 2023-07-18 17:30:07,001] Trial 10 finished with value: 1123.073974609375 and parameters: {'n_layers': 4, 'n_units_l0': 386, 'n_units_l1': 414, 'n_units_l2': 602, 'n_units_l3': 200, 'dropout_l3': 0.3918041109818822, 'lr': 0.0010152862932548898}. Best is trial 3 with value: 1050.91064453125.
[I 2023-07-18 17:30:07,058] Trial 11 pruned.
[I 2023-07-18 17:30:07,121] Trial 12 pruned.
[I 2023-07-18 17:31:29,266] Trial 13 finished with value: 1048.182861328125 and parameters: {'n_layers': 4, 'n_units_l0': 339, 'n_units_l1': 225, 'n_units_l2': 593, 'n_units_l3': 292, 'dropout_l3': 0.27202773630568, 'lr': 0.0020619558597725633}. Best is trial 13 with value: 1048.182861328125.
[I 2023-07-18 17:31:31,003] Trial 14 pruned.
[I 2023-07-18 17:31:31,067] Trial 15 pruned.
[I 2023-07-18 17:31:31,130] Trial 16 pruned.
[I 2023-07-18 17:31:31,202] Trial 17 pruned.
[I 2023-07-18 17:31:31,263] Trial 18 pruned.
[I 2023-07-18 17:31:31,330] Trial 19 pruned.
[I 2023-07-18 17:31:31,396] Trial 20 pruned.
[I 2023-07-18 17:31:31,456] Trial 21 pruned.
[I 2023-07-18 17:31:31,515] Trial 22 pruned.
[I 2023-07-18 17:31:31,577] Trial 23 pruned.
[I 2023-07-18 17:31:33,291] Trial 24 pruned.
[I 2023-07-18 17:31:33,350] Trial 25 pruned.
[I 2023-07-18 17:31:33,416] Trial 26 pruned.
[I 2023-07-18 17:31:33,478] Trial 27 pruned.
[I 2023-07-18 17:31:33,538] Trial 28 pruned.
[I 2023-07-18 17:31:33,610] Trial 29 pruned.
[I 2023-07-18 17:31:33,678] Trial 30 pruned.
[I 2023-07-18 17:31:44,666] Trial 31 pruned.
[I 2023-07-18 17:31:55,722] Trial 32 pruned.
[I 2023-07-18 17:31:55,793] Trial 33 pruned.
[I 2023-07-18 17:31:55,869] Trial 34 pruned.
[I 2023-07-18 17:31:56,791] Trial 35 pruned.
[I 2023-07-18 17:31:58,538] Trial 36 pruned.
[I 2023-07-18 17:32:09,454] Trial 37 pruned.

[I 2023-07-18 17:32:09,532] Trial 38 pruned.
[I 2023-07-18 17:32:11,298] Trial 39 pruned.
[I 2023-07-18 17:32:22,309] Trial 40 pruned.
[I 2023-07-18 17:32:22,388] Trial 41 pruned.
[I 2023-07-18 17:32:22,464] Trial 42 pruned.
[I 2023-07-18 17:33:46,150] Trial 43 finished with value: 1329.5771484375 and parameters: {'n_layers': 8, 'n_units_l0': 393, 'n_units_l1': 682, 'n_units_l2': 277, 'n_units_l3': 225, 'n_units_l4': 384, 'n_units_l5': 644, 'n_units_l6': 452, 'n_units_l7': 549, 'dropout_l7': 0.21816439906728197, 'lr': 0.0019024642357299054}. Best is trial 13 with value: 1048.182861328125.
[I 2023-07-18 17:33:46,225] Trial 44 pruned.
[I 2023-07-18 17:33:46,301] Trial 45 pruned.
[I 2023-07-18 17:33:46,373] Trial 46 pruned.
[I 2023-07-18 17:33:46,463] Trial 47 pruned.
[I 2023-07-18 17:35:10,535] Trial 48 finished with value: 2605.78662109375 and parameters: {'n_layers': 10, 'n_units_l0': 338, 'n_units_l1': 446, 'n_units_l2': 409, 'n_units_l3': 274, 'n_units_l4': 451, 'n_units_l5': 420, 'n_units_l6': 506, 'n_units_l7': 726, 'n_units_l8': 565, 'n_units_l9': 652, 'dropout_l9': 0.3051120989366396, 'lr': 0.0008165976178608769}. Best is trial 13 with value: 1048.182861328125.
[I 2023-07-18 17:35:10,609] Trial 49 pruned.
[I 2023-07-18 17:35:10,705] Trial 50 pruned.
[I 2023-07-18 17:35:10,787] Trial 51 pruned.
[I 2023-07-18 17:36:33,788] Trial 52 finished with value: 1006.415283203125 and parameters: {'n_layers': 8, 'n_units_l0': 574, 'n_units_l1': 711, 'n_units_l2': 386, 'n_units_l3': 207, 'n_units_l4': 480, 'n_units_l5': 562, 'n_units_l6': 476, 'n_units_l7': 425, 'dropout_l7': 0.4034238275281995, 'lr': 0.003526864272316972}. Best is trial 52 with value: 1006.415283203125.
[I 2023-07-18 17:36:33,865] Trial 53 pruned.
[I 2023-07-18 17:36:33,944] Trial 54 pruned.
[I 2023-07-18 17:36:34,019] Trial 55 pruned.
[I 2023-07-18 17:36:34,084] Trial 56 pruned.
[I 2023-07-18 17:36:34,162] Trial 57 pruned.
[I 2023-07-18 17:36:34,241] Trial 58 pruned.
[I 2023-07-18 17:36:34,328] Trial 59 pruned.
[I 2023-07-18 17:36:34,403] Trial 60 pruned.
[I 2023-07-18 17:36:34,470] Trial 61 pruned.
[I 2023-07-18 17:36:34,549] Trial 62 pruned.
[I 2023-07-18 17:36:34,636] Trial 63 pruned.
[I 2023-07-18 17:36:34,719] Trial 64 pruned.
[I 2023-07-18 17:36:34,803] Trial 65 pruned.
[I 2023-07-18 17:36:34,879] Trial 66 pruned.
[I 2023-07-18 17:36:34,947] Trial 67 pruned.
[I 2023-07-18 17:36:35,021] Trial 68 pruned.
[I 2023-07-18 17:36:35,104] Trial 69 pruned.

[I 2023-07-18 17:36:35,186] Trial 70 pruned.
[I 2023-07-18 17:36:35,274] Trial 71 pruned.
[I 2023-07-18 17:36:35,362] Trial 72 pruned.
[I 2023-07-18 17:36:35,450] Trial 73 pruned.
[I 2023-07-18 17:36:35,542] Trial 74 pruned.
[I 2023-07-18 17:36:35,631] Trial 75 pruned.
[I 2023-07-18 17:36:35,719] Trial 76 pruned.
[I 2023-07-18 17:36:35,804] Trial 77 pruned.
[I 2023-07-18 17:36:37,560] Trial 78 pruned.
[I 2023-07-18 17:38:01,550] Trial 79 finished with value: 3312.99072265625 and parameters: {'n_layers': 5, 'n_units_l0': 363, 'n_units_l1': 333, 'n_units_l2': 516, 'n_units_l3': 239, 'n_units_l4': 473, 'dropout_l4': 0.20875086813706376, 'lr': 0.0004726170123446892}. Best is trial 52 with value: 1006.415283203125.
[I 2023-07-18 17:38:14,289] Trial 80 pruned.
[I 2023-07-18 17:38:16,024] Trial 81 pruned.
[I 2023-07-18 17:38:16,098] Trial 82 pruned.
[I 2023-07-18 17:38:16,170] Trial 83 pruned.
[I 2023-07-18 17:38:16,246] Trial 84 pruned.
[I 2023-07-18 17:38:16,316] Trial 85 pruned.
[I 2023-07-18 17:38:16,403] Trial 86 pruned.
[I 2023-07-18 17:38:16,497] Trial 87 pruned.
[I 2023-07-18 17:38:16,586] Trial 88 pruned.
[I 2023-07-18 17:39:40,550] Trial 89 finished with value: 874.1036987304688 and parameters: {'n_layers': 4, 'n_units_l0': 302, 'n_units_l1': 431, 'n_units_l2': 629, 'n_units_l3': 531, 'dropout_l3': 0.27982927029179305, 'lr': 0.000996013598014235}. Best is trial 89 with value: 874.1036987304688.
[I 2023-07-18 17:39:40,623] Trial 90 pruned.
[I 2023-07-18 17:39:40,695] Trial 91 pruned.
[I 2023-07-18 17:39:40,768] Trial 92 pruned.
[I 2023-07-18 17:39:40,844] Trial 93 pruned.
[I 2023-07-18 17:41:05,106] Trial 94 finished with value: 1517.8016357421875 and parameters: {'n_layers': 4, 'n_units_l0': 371, 'n_units_l1': 723, 'n_units_l2': 645, 'n_units_l3': 619, 'dropout_l3': 0.367847188341626, 'lr': 0.0006740556528763471}. Best is trial 89 with value: 874.1036987304688.
[I 2023-07-18 17:41:05,181] Trial 95 pruned.
[I 2023-07-18 17:41:05,281] Trial 96 pruned.
[I 2023-07-18 17:41:05,355] Trial 97 pruned.
[I 2023-07-18 17:41:05,445] Trial 98 pruned.
[I 2023-07-18 17:41:05,532] Trial 99 pruned.
States:
Number of finished trials: 100
Number of pruned trials: 87

Number of completed trials: 13
Best trial:
Value: 874.1036987304688
Params:
n_layers: 4
n_units_l0: 302
n_units_l1: 431
n_units_l2: 629
n_units_l3: 531
dropout_l3: 0.27982927029179305
lr: 0.000996013598014235

[I 2023-07-18 17:49:28,033] A new study created in memory with name: no-name-87f96e5f-7ce7-47f5-a9b3-4263fd488e47
[I 2023-07-18 17:50:51,350] Trial 0 finished with value: 835.2402954101562 and parameters: {'n_layers': 10, 'n_units_l0': 656, 'n_units_l1': 644, 'n_units_l2': 514, 'n_units_l3': 718, 'n_units_l4': 675, 'n_units_l5': 277, 'n_units_l6': 460, 'n_units_l7': 342, 'n_units_l8': 498, 'n_units_l9': 584, 'dropout_l9': 0.35274482626577036, 'lr': 0.05399352914460215}. Best is trial 0 with value: 835.2402954101562.
[I 2023-07-18 17:52:19,975] Trial 1 finished with value: 965.5225219726562 and parameters: {'n_layers': 8, 'n_units_l0': 421, 'n_units_l1': 491, 'n_units_l2': 543, 'n_units_l3': 384, 'n_units_l4': 511, 'n_units_l5': 613, 'n_units_l6': 418, 'n_units_l7': 467, 'dropout_l7': 0.40314262700748593, 'lr': 0.002175544038801345}. Best is trial 0 with value: 835.2402954101562.
[I 2023-07-18 17:53:43,531] Trial 2 finished with value: 3624.4013671875 and parameters: {'n_layers': 8, 'n_units_l0': 405, 'n_units_l1': 366, 'n_units_l2': 611, 'n_units_l3': 528, 'n_units_l4': 564, 'n_units_l5': 487, 'n_units_l6': 550, 'n_units_l7': 626, 'dropout_l7': 0.31888819971302435, 'lr': 0.0008668653916489756}. Best is trial 0 with value: 835.2402954101562.
[I 2023-07-18 17:55:06,899] Trial 3 finished with value: 1914.3046875 and parameters: {'n_layers': 6, 'n_units_l0': 474, 'n_units_l1': 324, 'n_units_l2': 379, 'n_units_l3': 457, 'n_units_l4': 466, 'n_units_l5': 555, 'dropout_l5': 0.32630571137910797, 'lr': 0.0015384842965336 13}. Best is trial 0 with value: 835.2402954101562.
[I 2023-07-18 17:56:30,693] Trial 4 finished with value: 871.7032470703125 and parameters: {'n_layers': 8, 'n_units_l0': 407, 'n_units_l1': 381, 'n_units_l2': 339, 'n_units_l3': 623, 'n_units_l4': 474, 'n_units_l5': 496, 'n_units_l6': 709, 'n_units_l7': 613, 'dropout_l7': 0.38332140805725323, 'lr': 0.0001801619582599 1263}. Best is trial 0 with value: 835.2402954101562.
[I 2023-07-18 17:56:31,578] Trial 5 pruned.
[I 2023-07-18 17:56:31,622] Trial 6 pruned.
[I 2023-07-18 17:56:31,666] Trial 7 pruned.
[I 2023-07-18 17:56:31,711] Trial 8 pruned.
[I 2023-07-18 17:56:31,756] Trial 9 pruned.
[I 2023-07-18 17:56:31,801] Trial 10 pruned.

[I 2023-07-18 17:57:56,030] Trial 11 finished with value: 1998.15283203125 and parameters: {'n_layers': 7, 'n_units_l0': 596, 'n_units_l1': 789, 'n_units_l2': 580, 'n_units_l3': 467, 'n_units_l4': 603, 'n_units_l5': 480, 'n_units_l6': 729, 'dropout_l6': 0.24906037794912436, 'lr': 0.004471356265507967}. Best is trial 0 with value: 835.2402954101562.
[I 2023-07-18 17:57:56,075] Trial 12 pruned.
[I 2023-07-18 17:57:56,122] Trial 13 pruned.
[I 2023-07-18 17:57:56,166] Trial 14 pruned.
[I 2023-07-18 17:57:56,209] Trial 15 pruned.
[I 2023-07-18 17:57:56,252] Trial 16 pruned.
[I 2023-07-18 17:57:56,297] Trial 17 pruned.
[I 2023-07-18 17:57:56,342] Trial 18 pruned.
[I 2023-07-18 17:57:56,387] Trial 19 pruned.
[I 2023-07-18 17:57:56,432] Trial 20 pruned.
[I 2023-07-18 17:57:56,476] Trial 21 pruned.
[I 2023-07-18 17:57:56,520] Trial 22 pruned.
[I 2023-07-18 17:57:57,388] Trial 23 pruned.
[I 2023-07-18 17:57:57,432] Trial 24 pruned.
[I 2023-07-18 17:57:57,477] Trial 25 pruned.
[I 2023-07-18 17:57:57,521] Trial 26 pruned.
[I 2023-07-18 17:57:58,381] Trial 27 pruned.
[I 2023-07-18 17:57:59,256] Trial 28 pruned.
[I 2023-07-18 17:57:59,302] Trial 29 pruned.
[I 2023-07-18 17:57:59,349] Trial 30 pruned.
[I 2023-07-18 17:57:59,394] Trial 31 pruned.
[I 2023-07-18 17:57:59,439] Trial 32 pruned.
[I 2023-07-18 17:57:59,483] Trial 33 pruned.
[I 2023-07-18 17:57:59,528] Trial 34 pruned.
[I 2023-07-18 17:58:00,401] Trial 35 pruned.
[I 2023-07-18 17:58:00,446] Trial 36 pruned.
[I 2023-07-18 17:58:00,491] Trial 37 pruned.
[I 2023-07-18 17:58:00,536] Trial 38 pruned.
[I 2023-07-18 17:58:00,581] Trial 39 pruned.
[I 2023-07-18 17:58:00,624] Trial 40 pruned.
[I 2023-07-18 17:58:00,668] Trial 41 pruned.
[I 2023-07-18 17:58:00,713] Trial 42 pruned.
[I 2023-07-18 17:58:00,758] Trial 43 pruned.
[I 2023-07-18 17:58:01,625] Trial 44 pruned.
[I 2023-07-18 17:58:02,497] Trial 45 pruned.
[I 2023-07-18 17:58:02,541] Trial 46 pruned.
[I 2023-07-18 17:58:02,585] Trial 47 pruned.
[I 2023-07-18 17:58:03,470] Trial 48 pruned.
[I 2023-07-18 17:58:03,516] Trial 49 pruned.

[I 2023-07-18 17:58:03,561] Trial 50 pruned.
[I 2023-07-18 17:58:03,605] Trial 51 pruned.
[I 2023-07-18 17:58:04,496] Trial 52 pruned.
[I 2023-07-18 17:58:04,542] Trial 53 pruned.
[I 2023-07-18 17:58:04,587] Trial 54 pruned.
[I 2023-07-18 17:58:04,632] Trial 55 pruned.
[I 2023-07-18 17:58:05,519] Trial 56 pruned.
[I 2023-07-18 17:58:05,564] Trial 57 pruned.
[I 2023-07-18 17:58:05,609] Trial 58 pruned.
[I 2023-07-18 17:58:09,049] Trial 59 pruned.
[I 2023-07-18 17:58:09,094] Trial 60 pruned.
[I 2023-07-18 17:58:09,138] Trial 61 pruned.
[I 2023-07-18 17:58:09,184] Trial 62 pruned.
[I 2023-07-18 17:58:09,228] Trial 63 pruned.
[I 2023-07-18 17:58:09,272] Trial 64 pruned.
[I 2023-07-18 17:58:09,316] Trial 65 pruned.
[I 2023-07-18 17:58:09,361] Trial 66 pruned.
[I 2023-07-18 17:58:09,405] Trial 67 pruned.
[I 2023-07-18 17:58:09,451] Trial 68 pruned.
[I 2023-07-18 17:58:10,338] Trial 69 pruned.
[I 2023-07-18 17:58:10,382] Trial 70 pruned.
[I 2023-07-18 17:58:10,446] Trial 71 pruned.
[I 2023-07-18 17:58:10,492] Trial 72 pruned.
[I 2023-07-18 17:58:11,456] Trial 73 pruned.
[I 2023-07-18 17:58:11,501] Trial 74 pruned.
[I 2023-07-18 17:58:12,363] Trial 75 pruned.
[I 2023-07-18 17:58:12,409] Trial 76 pruned.
[I 2023-07-18 17:58:12,454] Trial 77 pruned.
[I 2023-07-18 17:58:12,499] Trial 78 pruned.
[I 2023-07-18 17:58:12,544] Trial 79 pruned.
[I 2023-07-18 17:59:37,532] Trial 80 finished with value: 2288.10693359375 and parameters: {'n_layers': 9, 'n_units_l0': 568, 'n_units_l1': 429, 'n_units_l2': 475, 'n_units_l3': 603, 'n_units_l4': 577, 'n_units_l5': 461, 'n_units_l6': 453, 'n_units_l7': 218, 'n_units_l8': 766, 'dropout_l8': 0.3280456242410224, 'lr': 0.006345605729256616}. Best is trial 0 with value: 835.2402954101562.
[I 2023-07-18 17:59:37,577] Trial 81 pruned.
[I 2023-07-18 17:59:37,622] Trial 82 pruned.
[I 2023-07-18 17:59:37,667] Trial 83 pruned.
[I 2023-07-18 17:59:37,712] Trial 84 pruned.
[I 2023-07-18 17:59:37,757] Trial 85 pruned.
[I 2023-07-18 17:59:40,336] Trial 86 pruned.
[I 2023-07-18 17:59:41,232] Trial 87 pruned.

[I 2023-07-18 17:59:41,278] Trial 88 pruned.
[I 2023-07-18 17:59:42,163] Trial 89 pruned.
[I 2023-07-18 17:59:42,208] Trial 90 pruned.
[I 2023-07-18 17:59:43,101] Trial 91 pruned.
[I 2023-07-18 17:59:43,966] Trial 92 pruned.
[I 2023-07-18 17:59:46,533] Trial 93 pruned.
[I 2023-07-18 17:59:47,455] Trial 94 pruned.
[I 2023-07-18 18:01:11,984] Trial 95 finished with value: 2309.37646484375 and parameters: {'n_layers': 7, 'n_units_l0': 545, 'n_units_l1': 475, 'n_units_l2': 355, 'n_units_l3': 562, 'n_units_l4': 557, 'n_units_l5': 547, 'n_units_l6': 471, 'dropout_l6': 0.27380122504013077, 'lr': 0.003750930333503988}. Best is trial 0 with value: 835.2402954101562.
[I 2023-07-18 18:01:12,028] Trial 96 pruned.
[I 2023-07-18 18:01:12,073] Trial 97 pruned.
[I 2023-07-18 18:01:12,119] Trial 98 pruned.
[I 2023-07-18 18:01:12,164] Trial 99 pruned.

States:
Number of finished trials: 100
Number of pruned trials: 92
Number of completed trials: 8
Best trial:
Value: 835.2402954101562
Params:
 n_layers: 10
 n_units_l0: 656
 n_units_l1: 644
 n_units_l2: 514
 n_units_l3: 718
 n_units_l4: 675
 n_units_l5: 277
 n_units_l6: 460
 n_units_l7: 342
 n_units_l8: 498
 n_units_l9: 584
 dropout_l9: 0.35274482626577036
 lr: 0.05399352914460215