Towards Full Deep Learning-based SLAM

by

Abhishek Khoyani

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs in partial fulfillment of the requirements for the degree of

Master of Applied Science

in

Electrical and Computer Engineering

Carleton University
Ottawa, Ontario

© 2023
Abhishek Khoyani
Abstract

Applications of the robots are increasing in routine for shop floor activity, transportation, and many other areas. Simultaneous localization and mapping (SLAM) is one of the algorithm that gives a comprehensive understanding of robot’s environment in form of map to perform various operation like navigation, and space reconstruction along with current state of estimation of robot i.e., localization in form of pose and orientation. Deep learning-based SLAM can be employed to overcome challenges faced by conventional SLAM algorithms, such as dynamic environments (moving objects, lighting variations) and long-scale mapping scenarios. In absence of fully deep learning-based SLAM system, this thesis presents a loop closure approach with constructing a real-time graph that can be developed further in fully SLAM system. The proposed approach involves constructing a real-time graph using features extracted from deep learning-based backbone models to achieve robust loop closure detection. This modularized parameter-free system framework extracts features from input images, calculates matching scores between the features, and generates a graph structure that represents the connectivity of nodes based on the matching criteria. The proposed framework is evaluated using various deep learning-based feature extractors on 11 sequences from the 5 open-source dataset. The results show that the proposed framework achieves faster processing times whereas achieving higher precision-recall
scores compared to the other methods.

Once the graph is created, it can be utilized to improve the performance metrics by leveraging power of graph neural network (GNN). In light of this, a supervised offline method was explored to investigate the applicability of GNN in context of loop closure detection, for the first time in literature. The feature extractor is modified by adding graphSAGE layers on top of convolutional backbone models, which helps to learn the embeddings from the node’s neighborhood. The model was trained and tested on a combined graph of five datasets. Evaluation results exhibit improved performance, particularly in reducing false positives, which is a crucial objective in robust loop closure algorithms.
To my parents,

ॐ पूर्णमद: पूर्णमिदं पूर्णात्पूर्णमुदच्यते ।
पूर्णस्य पूर्णमादाय पूर्णमिवावशिष्यते ॥ ॐ शान्ति: शान्ति: शान्ति: ॥

- ईशावास्य उपनिषद

Translation: Om, That is Full, This also is Full, from this Fullness comes that Fullness, Taking Fullness from Fullness, Fullness Indeed Remains.
Declaration

In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of Carleton University’s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.
Acknowledgements

Firstly, I would like to express my heartfelt appreciation to my supervisor, Dr. Marzieh Amini, for placing her trust in my abilities and affording me the opportunity to pursue my MASc degree. This important milestone would not have been attainable without her unwavering guidance, encouragement, and motivation.

I am profoundly grateful for my wonderful family. To my late father, whose numerous sacrifices have paved the way for my current achievements, I owe a debt of gratitude. To my mother, sisters Pinal, Samapti, and Kajal, and brother Akshay, your boundless love and prayers have continuously motivated me to surpass my limits. I am also grateful to all my friends in India and Canada, who have stood by me through the ups and downs of this journey. Ankita, completing this thesis without you would have been an entirely different experience; our discussions and brainstorming sessions provided me with valuable insights and perspectives.
# Contents

Abstract i

Acknowledgements v

List of Tables x

List of Figures xi

1 Introduction 1
   1.1 Problem Statement 2
   1.2 Contribution 4
   1.3 Publications 5
   1.4 Thesis Structure 7

2 Theoretical Background 9
   2.1 Deep Learning 9
      2.1.1 Feed Forward Neural Network 10
      2.1.2 Convolutional Neural Network 10
      2.1.3 Graph Neural Network 12
         2.1.3.1 GraphSAGE 12
      2.1.4 Loss Function 15
2.2 Computer Vision ........................................... 16
  2.2.1 Feature Extraction ................................... 17
    2.2.1.1 Conventional Feature Extractors .............. 17
    2.2.1.2 Deep Learning-based Feature Extraction .... 18
    2.2.1.3 Backbone Models ................................ 20
  2.2.2 Bag-of-Words ......................................... 24
  2.2.3 Euclidean Distance .................................. 25
  2.2.4 Cosine Similarity ................................... 26

2.3 SLAM ....................................................... 27
  2.3.1 Visual Odometry ..................................... 29
  2.3.2 Loop Closing Detection ............................. 30
  2.3.3 Pose Graph Optimization ............................ 30

2.4 Visual SLAM Algorithms ................................ 31
  2.4.1 Conventional SLAM Algorithms ..................... 32
    2.4.1.1 DVO-SLAM .................................... 32
    2.4.1.2 RGBDSLAMv2 ................................ 32
    2.4.1.3 RTAB-MAP ................................... 34
  2.4.2 Deep Learning-based SLAM Algorithms ............ 35
    2.4.2.1 CodeSLAM ................................... 35
    2.4.2.2 CNN-SLAM .................................. 37

2.5 Conclusion ................................................ 38

3 Related Work ................................................ 39
  3.1 LCD ...................................................... 39
    3.1.1 Conventional LCD Methods ........................ 40
    3.1.2 Deep Learning-based LCD Methods ............... 42
3.1.3 Graph-based LCD methods ........................................... 44
3.2 Conclusion ................................................................. 44

4 Methodology ................................................................. 46
  4.1 Introduction .............................................................. 46
  4.2 Graph SLAM Structure .................................................. 46
    4.2.1 Graph Optimization ............................................... 47
  4.3 Benchmark .............................................................. 50
    4.3.1 Datasets ............................................................ 51
    4.3.2 Groundtruth ....................................................... 52
      4.3.2.1 Groundtruth for Image-2-Image Dataset ............... 53
      4.3.2.2 Groundtruth for Sequential Dataset ................. 54
    4.3.3 Evaluation Metrics ............................................. 55
  4.4 Conclusion .............................................................. 56

5 Loop Closing Detection With Constructing Graph Structure ........ 57
  5.1 Proposed Method ...................................................... 57
    5.1.1 Environmental Setup ........................................... 59
    5.1.2 Python Packages ............................................... 60
  5.2 Experimental Results ................................................ 62
    5.2.1 Method Evaluation ............................................ 62
    5.2.2 Comparative Results ......................................... 69
    5.2.3 Time Complexity ............................................... 70
  5.3 Conclusion .............................................................. 72

6 Learning Embeddings Using Graph Neural Network ................. 73
  6.1 Introduction ............................................................ 73
6.2 Problem Formation .................................................. 75
6.3 Methodology and Experiments .................................... 76
6.4 Experimental Results ................................................. 79
6.5 Conclusion ............................................................. 81

7 Conclusion and Future Work ......................................... 82

7.1 Future Works .......................................................... 83

References ................................................................. 87
List of Tables

4.1 Datasets Summary .................................................. 52
4.2 Loop closure confusion matrix .................................. 55

5.1 The performance comparison of the different feature extractors in the term of average precision ........................................ 63
5.2 The performance comparison between methods proposed in [1], [2], [3] and ours in terms of average precision. ............................... 69
5.3 The performance comparison of the different feature extractors in the term of time complexity (in ms) ................................. 71
5.4 Overall time-complexity comparison between our proposed method and FABMAP [1] ................................................................. 71

6.1 Final hyperparameter values selected after 128 iteration monitored to minimize validation loss. ............................................. 77
6.2 Training parameter and evaluation metrics comparison of two training iteration. ................................................................. 79
# List of Figures

1.1 Odometry correction after adjusting detected loops. .............................. 3

2.1 Convolution operation of a $3 \times 3$ kernel with a $4 \times 4$ input with stride=1. 
   [4] .................................................................................. 11

2.2 A layer representation of a simple graph neural network where $V$, $E$, and $U$ 
   gets updated from the information stored in itself. Update function $f$ 
   can be different to update each part individually [5]. .......................... 13

2.3 “Pooling” function gathers the information on the edges from the connected 
   nodes and pass it back to the node with “update” function [5]. ............... 13

2.4 Overall methodology of the GraphSAGE architecture explaining sampling 
   neighbourhood features, aggregate it, and final prediction, (a) Sample 
   neighborhood, (b) Aggregate feature information from neighbors, (c) Predict 
   graph context and label using aggregated information [6]. ........................ 14

2.5 An example of keypoints detected features using SIFT on a sample image 
   from New College dataset. ......................................................... 18

2.6 An example of layer wise extracted features of a sample image from the New 
   College dataset using ResNet. .................................................... 19
2.7 Naive visual representation of BoVW. ........................................... 25
2.8 Categorization of SLAM methods based on the working methodology. 28
2.9 Basic RGB-D SLAM architecture divided in front-end and back-end
   based on the task it performs. ...................................................... 29
2.10 (a) Corrupted Manhattan world pose graph with 3500 nodes. (b)
       Optimized map using g2o [7]. ................................................ 31
2.11 DVO-SLAM output graph structure with estimated and ground truth
       trajectory [8]. ........................................................................ 33
2.12 RGBDSLAMv2 scene reconstruction output [9]. ................................. 34
2.13 OctoMap output generated from RTAB-MAP [10]. ............................. 35
2.15 3D reconstruction of ICL-NUIM “living0” sequence using (a) CNN-
       SLAM [12] and (b) LSD-SLAM [13]. ......................................... 38

4.1 Constructing graph based on calculated matching score. (a) Matching
       score is calculated between the current node \( I_i \) and all the existing \( N \)
       nodes using Eq. 5.1. (b) False positives shall be removed by applying
       threshold on matching score, once the whole graph is constructed. ... 48
4.2 Example of a pair of nodes \( N_i \) and \( N_j \) with its predicted relative
       transformation \( \hat{E}_{ij} \) and measurement \( E_{ij} \) with it’s uncertainty referred
       by information matrix \( \Omega_{ij} \). [14] ........................................... 49
4.3 Example pair from each of the dataset used in experiment. First row shows the query images, second row shows the corresponding matched images. The sequence shown are from (a) City Centre, (b) New College [1], (c) KAIST North, (d) KAIST East, (e) KAIST West [15], (f) KITTI 00, (g) KITTI 05 [16], (h) MALAGA Parking 6L [17], and (i) EuROC Machine Hall 05 [18].

4.4 Visualization of 394 × 394 sized groundtruth matrix of NC-reduced. Light pixels indicate loop matching for the corresponding indices, while dark pixels represent no loop matching.

4.5 Custom class representation of the City Centre dataset as the sequential groundtruth where each entry consists of the query node ID, start matching node ID, and end matching node ID from the whole sequence.

5.1Visualization of correspondence matrix predicted by various backbone models on the New College dataset.

5.2 Performance comparison of the various feature extractors based on the achieved mean average precision and standard variation on five sequence of image-2-image dataset.

5.3 (a)-(e) The performance comparison of the different feature extractor with image-2-image corresponding labels in terms of PR curves on different experimental datasets. (f) The performance comparison in terms of PR curve on different sequential datasets after selecting the best feature extractor from previous results.
5.4 First column is the ground truth trajectories, second column is the final predicted trajectories, and the third column is the final correspondence matrices of the (a) New College, (b) City Centre, (c) KITTI-00, and (d) KITTI-05 datasets. 68

6.1 Model architecture to learn the embeddings using graphSAGE layer on top of ShuffleNet. 77

6.2 Cosine embedding loss curves for the hyperparameter tuning and training iterations. (a) The comparison of multiple iteration in terms of cosine embedding loss against training steps during hyperparameter tuning. Each curves represent the training curve with different hyperparameters set. Legend is avoided in figure for better visualization. (b) The comparison of Iteration 1 and Iteration 2 during training in terms of cosine embedding loss. Iteration 2 was performed in continuation of Iteration 1 to optimize model as its clear the loss is converging at the end of Iteration 1. 78

6.3 The performance comparison of GNN versus vanilla backbone models in terms of PR curves obtained on combined dataset. 80
Chapter 1

Introduction

Robotics encompasses various disciplines involved in the design, construction, algorithms, and operation of robots. Robots play a critical role in industries such as manufacturing, automation, transportation, and space exploration, prompting extensive research to enhance their performance. The concept of simultaneous localization and mapping (SLAM) [19] involves a robot creating a map of its surroundings while simultaneously determining its own position within that environment. In-depth analysis conducted in [20] explored the challenges, different approaches, and state-of-the-art methods related to SLAM. The conventional SLAM algorithm is known to have limitations in terms of tracking failures, long-term scenarios, and dynamic environments. Deep learning-based approaches offer a promising solution to address these challenges, leveraging their remarkable performance in diverse domains of computer vision. However, as of the time of writing, there is no fully deep learning-based SLAM algorithm available. Nevertheless, hybrid approaches combining deep learning with other techniques have been explored [21]. One potential approach is the implementation and optimization of graph SLAM [14] by utilizing deep learning based framework. Graph SLAM represents the environment as a graph consisting of vertices representing
robot poses or landmarks and edges representing the constraints between these poses or landmarks. Further details about graph SLAM and its optimization technique are outlined in Section 4.2.

1.1 Problem Statement

While developing a complete SLAM algorithm can be a time-consuming endeavor, this thesis focuses on a specific aspect of SLAM known as loop closing detection (LCD), which plays a crucial role in the overall SLAM process. LCD is responsible for identifying instances when a mobile robot revisits a previously encountered location, thereby closing a loop in its trajectory. This detection is essential as it allows the robot to correct any accumulated errors in its pose estimation and minimize drift in the generated map, as depicted in Fig. 1.1. Once a loop closure is detected, the robot can refine its trajectory estimate and update the map accordingly. By effectively identifying loop closures, the robot can optimize map generation and enhance the accuracy of the final map.

Various sensor types can be utilized in SLAM, including monocular and stereo cameras, light detection and ranging (LiDAR), radio detection and ranging (radar), and inertial measurement unit (IMUs). Each sensor type possesses its own advantages and limitations, and their combinations can be tailored to specific application requirements and environmental conditions. Visual SLAM, in particular, solely relies on camera input for pose estimation and map construction. Visual LCD or visual place recognition addresses the challenge of identifying previously visited locations within a visual SLAM system. This involves comparing the current camera image with images captured during the robot’s past trajectories to detect loop closures and rectify map errors.
Visual LCD poses challenges due to scene variations such as changes in lighting, viewpoint, and occlusions. While IMUs and GPS can provide valuable information for pose estimation and localization, their accuracy and reliability may be compromised, particularly in indoor environments with weak or absent GPS signals. Visual SLAM has garnered significant attention in research due to its ability to achieve accurate and robust pose estimation and mapping using only visual input from cameras. However, many existing visual LCD methods do not explicitly construct a graph structure to represent the environment.

The process of LCD can be divided into three main sub-parts,

1. **Feature Extraction:** This step involves extracting distinctive features from the images that capture important characteristics of the environment. These features can include points, edges, or other patterns that exhibit robustness against changes in lighting conditions, viewpoints, and occlusions. The selection of appropriate features plays a crucial role in accurately identifying loop closures.

2. **Feature Matching Criteria:** In this step, a matching criterion is applied
to compare the features extracted from the current image with those obtained from previous images in the robot’s trajectory. The goal is to identify potential loop closures by finding similar or corresponding features between different images. Various distance measures or similarity measures can be employed as the matching criterion, depending on the specific application requirements.

3. **Image Retrieval**: Once potential loop closures are identified based on the feature matching, the corresponding images from the robot’s past trajectory need to be retrieved for further analysis and processing. These retrieved images provide additional information to refine the loop closure detection and perform subsequent tasks such as pose estimation and map correction.

All of these sub-parts need to be performed efficiently and in real-time to enable loop closure detection to be performed on-board the robot. This can be challenging, especially in large-scale environments with many possible loop closures.

### 1.2 Contribution

The primary contributions of this thesis can be summarized as follows:

- Extensive investigation of both conventional and deep learning-based SLAM and LCD algorithms, providing a comprehensive understanding of the existing methodologies in the field.

- Introduction of a novel loop closing technique based on a graph structure, which can serve as a foundation for a complete deep learning-based SLAM algorithm by incorporating additional components such as visual odometry and pose-graph optimization.
• Development of a modularized LCD system that allows for experimentation with various feature extraction methods and matching criteria, providing flexibility in adapting the system to different scenarios and datasets.

• Unlike other approaches that often require manual parameter tuning for different datasets, the proposed system eliminates the need for such manual adjustments.

• The formulation of the loop closing problem as a link prediction task enables the learning of similarity embeddings from prior data using graph neural network (GNN), facilitating more accurate and robust loop closure predictions.

• Exploration of the potential applicability of GNN in the domain of loop closure detection, potentially paving the way for future research and advancements in this area of study.

1.3 Publications

Work conducted in this thesis resulted in the following publications:


• A. Khoyani and M. Amini, “Learning embeddings for loop closing detection using graph neural network,” in IROS 2023 Workshop on Closing the Loop
on Localization: What Are We Localizing For, and How Does That Shape Everything We Should Do? [23]

In addition to the above published work, I’ve contributed to several other research projects which are out-of-the scope of this thesis.


1.4 Thesis Structure

- **Chapter 2**: This chapter outlines the fundamentals of Deep learning with a focus on Convolutional Neural Networks (CNN) and Computer Vision. Additionally, it throws light on SLAM algorithms and their applications.

- **Chapter 3**: This chapter dives into the work accomplished on LCD in detail as well as the state-of-the-arts methods in LCD.

- **Chapter 4**: The methodology chapter gives information on the graph SLAM structure and the benchmark elements used for the experiment including the Dataset, the Ground truth, and the evaluation metrics.
• **Chapter 5**: It consists of the primary view for the proposed project including the minute details about the method used; the experiment setup and the results achieved. Furthermore, it goes over the model evaluation and comparative results.

• **Chapter 6**: In this chapter, the loop closing problem is extended as a link prediction task in graph theory, and the utilization of GNN for robust loop closure detection is proposed.

• **Chapter 7**: Concludes with summarizing the thesis’s objective, and debates viable future work in the field.
Chapter 2

Theoretical Background

The key concepts used throughout this thesis consist of deep learning, certain concepts of computer vision, and SLAM are covered in this chapter with a short description of each is provided as background. Furthermore, an in-depth discussion is provided on the most popular conventional and deep-learning based SLAM techniques.

2.1 Deep Learning

Deep learning [30] is a sub-field of machine learning which uses artificial neural networks to train, learn and extract representation from a raw form of data. Unlike machine learning, deep learning-based algorithms are known for improvised selecting complex attributes and patterns from large datasets. These huge datasets can have any type of input data including texts, images or audios. Thus, deep learning is well-known for its versatility and potential to learn hierarchical patterns and intricate representations from raw data. Deep learning models have higher precision and performance rate and have attained significant growth in various branches like computer vision, natural language processing (NLP), image detection and classification and speech recognition.
2.1.1 Feed Forward Neural Network

Feed forward neural network is one of the elementary algorithms in the field of deep learning. It works by processing the raw input data through its multiple layers to represent unique features and dimensions of the data and computes activation function to detect non-linear patterns form the input data. The flexibility of this algorithm has shown successful outcomes for various applications in the deep learning domain such as NLP, image classification and regression, and speech recognition. It simply applies eq. 2.1 on the input data $x$ in order to learn the mapping represented by $w$ to generate output $y$.

\[ y = \sum_{i}^{n} w_{i}x_{i} + b \]  

(2.1)

2.1.2 Convolutional Neural Network

Out of all the deep learning models, CNN [31] is one of the most prevailing models for tasks related to image containing data and performs meticulously when it comes to image classification, image or object detection and image segmentation. It mainly focuses on feature extraction by representing complex patterns and structural relationships in an image. It is known for its ability to learn and extract primary features without disturbing the meaningful information from the raw data. The working of CNN depends on its three predominant parts namely convolutional layer which applies convolution operations to extract the key features and dimensional patterns in the data, non-linear activation function to learn complex relationships, and pooling layer reduce this dimension identified by the above-mentioned layer. Mathematically,
convolution operator ($\otimes$) between two 2-dimensional function $f$ and $g$ is written as,

$$(f \otimes g)(p, q) = \sum_{p' \in W} \sum_{q' \in H} f(p', q')g(p - p', q - q')$$  \hspace{1cm} (2.2)

where $W$, and $H$ are range of $p'$, and $q'$, respectively, which is usually width and height of the convolutional filter. Whereas, $p$, and $q$ are pixel coordinates of the image.

Convolution layer is an integral part of the network having a crucial role in the training of CNN. This layer has convolution operations which applies kernel-sized filter on the raw image to generate feature maps to identify dimensional patterns in the image. Each filter is a small grid-like matrix, and they examine small regions called local receptive fields to detect local features and patterns. These filters share weights with the input data as shown in Fig. 2.1. This layer uses strides to determine the step-size for the filters and padding to add extra border pixels to the input for easier computation.

The Pooling layer essentially down-samples the feature maps generated by the above layer to learn a less-complex representation of the image. This layer divides these feature maps into overlapping and non-overlapping local receptive fields known as pooling windows. This pooling window either select maximum values within the fields for max pooling or average values for average pooling. The size of this pooling windows calculates the number of down-samples transpired.
2.1.3 Graph Neural Network

GNN [32] are deep learning models that are used on graph based structured input data. This neural network functions by interpreting the convoluted associations in the input data. It gathers information across every node, edge and global context linked with the feature vectors. If we denote the node features as $V$, Edge features as $E$ and, whole global graph feature as $U$, it uses separate multilayer perceptron to learn about each node vector, edge vector, and global vector to updates the representation layer wise. GNN has various approaches when it comes to different situations. The approaches mentioned below are the commonly used approaches in GNN [5].

**Pooling information:** If the prediction is on nodes, the graph already consists of the information needed on the nodes and is the simplest case, as shown in Fig. 2.2, which can be solved by linear classification. In case of the edges having information and the model needs node-level feature prediction, as shown in Fig. 2.3, the GNN uses pooling method to get needed information from the edges and aggregate them to the nodes and vice versa. However, if we have node level feature information and the model needs to predict the global characteristics then, the GNN gathers all the node level information and groups them and uses pooling.

**The message passing approach to exchange information:** This approach is used to neighboring nodes embeddings, aggregate them via sum or mean functions and pass them through an update function. This can be applied to nodes and edges and can occur between them and hence increases the connectivity of the entire graph.

2.1.3.1 GraphSAGE

The graphSAGE [6] layer is a key component in GNN that enables effective information propagation and aggregation within a graph structure. It operates by aggregating
Figure 2.2: A layer representation of a simple graph neural network where $V$, $E$, and $U$ gets updated from the information stored in itself. Update function $f$ can be different to update each part individually [5].

Figure 2.3: “Pooling” function gathers the information on the edges from the connected nodes and pass it back to the node with “update” function [5].
feature information from a node’s local neighborhood and updating the node’s own features based on this aggregated information. Fig. 2.4 shows the step-by-step visualization of the sampling, aggregation, and update functions.

During the information aggregation process, the graphSAGE layer considers the features of a node’s neighboring nodes and combines them using a pooling or aggregation function, such as mean pooling or max pooling. This aggregation step allows each node to incorporate information from its surrounding nodes.

After the aggregation, the updated features are combined with the node’s own features and transformed using a neural network layer. This transformation enables the node to update its own representation based on both its own features and the aggregated neighborhood information.

\[ x_i = w_1 x_i + w_2 \cdot \rho_{j \in \mathcal{N}(i)} x_j \]  

(2.3)

In eq. 2.3, the \( x_i \) nodes features are getting updated using its own features with learn-able parameters \( w_1 \) and from the neighbor’s features \( x_j \) with parameters \( w_2 \). Parameter, \( \rho \), is the pooling or aggregate function, which can be “mean”, “max”, “sum”, or any other permutation invariance operator. Based on this aggregated learned
features, node-level, edge-level or graph-level predictions can be made.

2.1.4 Loss Function

A loss function, also known as a cost function, quantifies the deviation between predicted value $\hat{y}$ by the model and ground true values $y$. It measures the quality of the model’s predictions and serves as a basis for optimizing the model’s parameters. The goal is to minimize the loss function to improve the model’s performance.

The choice of loss function depends on the specific task and the nature of the data. For example, in binary classification tasks, common loss functions include the binary cross-entropy loss (BCE) (Eq. 2.4).

$$
\text{BCE}(y, \hat{y}) = [y \cdot \log(\hat{y}) + (1 - y) \cdot \log(1 - \hat{y})] 
$$ (2.4)

For similarity embedding learning, cosine embedding loss [33] (Eq. 2.5) can be used. Whereas margin value is used for additional penalty when $y$ is negative.

$$
\mathcal{L}_{cos}(y, \hat{y}) = \begin{cases} 
1 - \cos(y, \hat{y}), & \text{if } y = 1 \\
\max(0, \cos(y, \hat{y}) - \text{margin}), & \text{if } y = -1
\end{cases} 
$$ (2.5)

The optimization process involves finding the optimal values for the model’s parameters that minimize the loss function. This is achieved through an optimization algorithm, typically gradient-based methods like gradient descent or its variants. These algorithms iteratively update the parameters based on the gradients of the loss function ($\nabla \mathcal{L}(y, \hat{y})$) with respect to the parameters. The learning rate, which determines the step size of parameter updates, is an important hyperparameter that affects the optimization process.
The optimization process seeks to find the global minimum of the loss function, but it is not always guaranteed to converge to the global minimum. The algorithm may get stuck in a sub-optimal solution or encounter other challenges such as over-fitting or vanishing/exploding gradients. Therefore, careful selection of the loss function and fine-tuning of the optimization process are crucial for training effective machine learning models.

2.2 Computer Vision

Computer vision is an interdisciplinary field of study that involves using computers to process, analyze, and interpret images and video data. It is concerned with developing algorithms and techniques that enable computers to gain a high-level understanding of visual information, such as recognizing objects, detecting patterns, and identifying specific features in images.

At a fundamental level, computer vision involves extracting meaningful information from raw visual data. This process typically involves several stages, such as image acquisition, preprocessing, feature extraction, and classification. Feature extraction is one of the most critical stages in computer vision, as it involves identifying and extracting relevant and discriminative features from the raw visual data, such as images or point clouds. This process typically involves identifying distinctive patterns or structures in the visual data and encoding them as a set of features that can be used for further analysis and interpretation of various tasks, such as object recognition, image matching, image retrieval, and place recognition in the context of SLAM. Examples of visual features that are commonly used in computer vision include edges, corners, textures, and color histograms.
2.2.1 Feature Extraction

Feature extraction techniques typically involve processing the raw data to identify points or regions of interest, and then computing numerical descriptors that capture the visual or structural characteristics of these points or regions. These descriptors are often designed to be invariant to changes in scale, rotation, illumination, and viewpoint, so that they can be reliably matched and compared across different images or scenes.

Feature extraction techniques typically involve processing the raw data to identify points or regions of interest, and then computing numerical descriptors that capture the visual or structural characteristics of these points or regions. These descriptors are often designed to be invariant to changes in scale, rotation, illumination, and viewpoint, so that they can be reliably matched and compared across different images or scenes.

2.2.1.1 Conventional Feature Extractors

Conventional feature extraction techniques have been widely used in computer vision for many years and have provided good performance in various applications. Commonly used feature extraction techniques include:

- Point-based features: These include identifying points of interest, such as corners or keypoints, and computing descriptors that capture the local visual characteristics around these points. Examples of point-based features include scale-invariant feature transform (SIFT) [34], speeded-up robust features (SURF) [35], and oriented FAST and rotated BRIEF (ORB) [36]. Refer Fig. 2.5, which illustrate an example of SIFT keypoint features extracted from a sample image of the New College dataset.
• Region-based features: These involve segmenting the image into regions or objects, and computing descriptors that capture the visual characteristics of these regions. Examples of region-based features include histogram of oriented gradient (HOG) [37], and local binary patterns (LBP) [38] based features.

Conventional methods often require careful design and manual engineering of feature extraction algorithms, which can be time-consuming and limited to the specific characteristics that the algorithms are designed to capture. With the rise of deep learning and the ability of neural networks to learn features automatically, conventional feature extraction methods have been partly replaced by end-to-end deep learning models that can learn complex representations directly from raw image data.

2.2.1.2 Deep Learning-based Feature Extraction

Deep learning-based feature extractors refer to methods that utilize deep neural networks to automatically learn feature representations from raw data. It is common practice to employ features extracted from intermediate layers of CNNs as foundational
models for subsequent processing. Fig. 2.6 shows the extracted patterns from the different layers of the ResNet \cite{39} architecture of a sample image from the New College dataset. These features serve as the backbone of the network in further processing tasks. Several prominent classification networks have emerged as widely adopted choices for feature extraction in various applications. These networks have undergone extensive research and have been trained on large-scale image datasets to acquire comprehensive representations capable of capturing both low-level details and high-level semantics. Below are the some of the widely used classification networks which we can use for the feature extraction.

Figure 2.6: An example of layer wise extracted features of a sample image from the New College dataset using ResNet.

Feature extraction is a critical step in many computer vision tasks, as the quality and discriminative power of the extracted features directly affect the performance and
accuracy of subsequent processing steps, such as feature matching, object recognition, and SLAM.

2.2.1.3 Backbone Models

In this section, we briefly explained some of the commonly used backbone models, in chronological order, used for deep learning-based feature extractor. We’ll be using these models for extracting features in Chapter 5. These models are compared on the basis of the Top-1 and Top-5 accuracy achieved on ImageNet [40] multi-class classification problem. Top-1 accuracy represents the conventional accuracy, where the model’s highest predicted label corresponds to the ground truth label, whereas top-5 accuracy considers a match as correct when the ground truth label matches with any of the top five predictions made by the model.

- **AlexNet (2012) [41]**: A deep CNN architecture that played a significant role in advancing the field of computer vision, known for its successful application in image classification tasks. It introduced concepts such as rectified linear units (ReLU) and dropout, leading to improved performance. Achieved accuracy on ImageNet: Top-1: 56.52%, Top-5: 79.07%.

- **VGG (2014) [42]**: A deep CNN architecture known for its simplicity and effectiveness, featuring a stack of convolutional layers with small filter sizes and max-pooling layers. Achieved accuracy on ImageNet: Top-1: 73.36%, Top-5: 91.52%.

- **GoogLeNet (2015) [43]**: Also known as Inception V1, it introduced the inception module that consists of multiple parallel convolutional layers with different filter sizes to capture different levels of spatial information. It was prone
to vanishing gradient problem which is taken care in Inception V3. Achieved accuracy on ImageNet: Top-1: 69.78%, Top-5: 89.53%.

- **Inception V3 (2016) [44]**: An enhanced version of GoogLeNet, incorporating improvements such as factorized convolution and auxiliary classifiers, leading to better accuracy and improved training efficiency. Achieved accuracy on ImageNet: Top-1: 77.29%, Top-5: 93.45%.

- **ResNet (2016) [39]**: A groundbreaking architecture that introduced the concept of residual learning, using skip connections to enable the training of very deep neural networks and mitigate the vanishing gradient problem, which enable training of extremely deep networks. Achieved accuracy on ImageNet: Top-1: 76.13%, Top-5: 92.86%.

- **SqueezeNet (2016) [45]**: An architecture focused on reducing the number of parameters while maintaining high accuracy by utilizing various techniques like 1x1 convolutions and aggressive down-sampling. Achieved accuracy on ImageNet: Top-1: 58.18%, Top-5: 80.62%.

- **Wide ResNet (2016) [46]**: A variant of ResNet that increases the width of the layers, providing a wider feature space and enabling better feature learning capabilities. With such capability it introduces computational complexity compared to standard ResNet. Achieved accuracy on ImageNet: Top-1: 78.85%, Top-5: 94.28%.

- **DenseNet (2017) [47]**: A network architecture that promotes feature reuse by connecting each layer to every other layer in a feed-forward manner, enabling efficient information flow and better gradient propagation. It may increases
memory consumption as the networks grows deeper considering its feature reusability. Achieved accuracy on ImageNet: Top-1: 77.14%, Top-5: 93.56%.


- **MobileNet V2 (2018) [49]**: A lightweight architecture for mobile devices, leveraging depthwise separable convolutions and inverted residual blocks to achieve a good balance between model size and accuracy. Achieved accuracy on ImageNet: Top-1: 71.88%, Top-5: 90.29%.

- **ShuffleNet V2 (2018) [50]**: An architecture that employs channel shuffling and pointwise group convolutions to reduce computational complexity and achieve efficient model design. Achieved accuracy on ImageNet: Top-1: 76.23%, Top-5: 93.01%.

- **EfficientNet (2019) [51]**: A family of CNN architectures that achieve high accuracy while maintaining efficiency by scaling depth, width, and resolution in a balanced manner. Achieved accuracy on ImageNet: Top-1: 84.01%, Top-5: 96.92%.

- **MNASNet (2019) [52]**: A mobile-oriented architecture designed to achieve high performance on resource-constrained devices by utilizing efficient depthwise separable convolutions and model scaling techniques. Achieved accuracy on ImageNet: Top-1: 76.51%, Top-5: 93.52%.

- **MobileNet V3 (2019) [53]**: The next iteration of MobileNet, introducing
new features like a combination of mobile-friendly architecture design, efficient operations, and improved performance. Achieved accuracy on ImageNet: Top-1: 74.04%, Top-5: 91.34%.

- **RegNet (2020) [54]**: A family of neural network architectures designed for scalable and efficient model design, enabling effective trade-offs between model size, accuracy, and computational resources. Achieved accuracy on ImageNet: Top-1: 80.88%, Top-5: 95.34%.

- **VisionTransformer (2020) [55]**: A purely transformer-based architecture for vision tasks, utilizing self-attention mechanisms to capture global and local image information effectively. Due to attention mechanism it requires more computational resource compared to convolutional models. Achieved accuracy on ImageNet: Top-1: 81.07%, Top-5: 95.32%.

- **EfficientNetV2 (2021) [56]**: An updated version of EfficientNet with improved performance and efficiency, achieved through the use of compound scaling, convolutional layers with efficient operations, and advanced regularization techniques. Achieved accuracy on ImageNet: Top-1: 85.11%, Top-5: 97.16%.

- **SwinTransformer (2021) [57]**: A vision transformer architecture that introduces the concept of shifted windows, enabling efficient modeling of long-range dependencies in images. Achieved accuracy on ImageNet: Top-1: 83.71%, Top-5: 96.82%.

- **ConvNeXt (2022) [58]**: An architecture that combines the concepts of depth-wise separable convolutions and grouped convolutions to effectively capture spatial and channel-wise dependencies in an input. Achieved accuracy on ImageNet: Top-1: 85.34%.
geNet: Top-1: 84.06%, Top-5: 96.87%.

- **MaxVit (2022) [59]:** A vision transformer architecture that combines the advantages of both CNN and transformers, employing a hybrid architecture for capturing spatial and channel-wise information. Achieved accuracy on ImageNet: Top-1: 83.70%, Top-5: 96.72%.

Above mentioned backbone models will be used in chapter 6 for extracting features for the loop closing detection.

### 2.2.2 Bag-of-Words

Bag-of-words (BoW) is a simple and popular approach for feature representation in natural language processing and computer vision. It involves representing a text document or an image as a “bag” of its constituent words or visual features, respectively, ignoring their order but keeping track of their frequencies. In computer vision, BoW, or widely known as bag-of-visual-words (BoVW), can be applied to image classification, object recognition, and scene understanding tasks. The visual features extracted from an image are represented as a vector of feature counts where each entry in the vector corresponds to a unique visual feature in the vocabulary, and its value corresponds to the frequency of that feature in the image, as shown in Fig. 2.7. This approach can capture the global visual properties of an image, but loses the spatial information about where the features appear in the image.

Overall, the BoW approach is simple and efficient, but it has limitations, such as the loss of sequential or spatial information and the inability to capture the semantic relationships between visual features. However, it is still widely used as a baseline for comparison with more advanced feature extraction and representation methods.
such as deep learning-based CNNs or feature encoding methods like vector of locally aggregated descriptors (VLAD) [60].

2.2.3 Euclidean Distance

Euclidean distance is a commonly used distance metric in mathematics and data analysis. It measures the straight-line distance between two points in euclidean space. This distance corresponds to the length of the shortest path between two points, which is a straight line.

Mathematically, the euclidean distance between two vectors A and B can be calculated as the square root of the sum of the squares of the difference between their corresponding coordinates:

$$d(A, B) = \sqrt{\sum_{i=1}^{n} (B_i - A_i)^2} \quad (2.6)$$

Equation 2.6 elucidates the computation of euclidean distance between vector A
and B in n-dimensional Euclidean space. However, in the context of LCD, Euclidean distance is not widely used as it calculates the absolute distance between two vector and not the angular distance which may result in the lower similarity score for the similar images with minor scene variations.

### 2.2.4 Cosine Similarity

Cosine similarity is a measure of similarity between two vectors in a multi-dimensional space, often used in information retrieval, natural language processing, and recommendation systems. It calculates the cosine of the angle between two vectors, which can be interpreted as the similarity of their orientation or direction, independent of their magnitude or length.

Mathematically, the cosine similarity between two vectors A and B can be calculated as the dot product of the vectors divided by the product of their magnitudes:

\[
\text{cosine similarity}(A, B) = \frac{(A \cdot B)}{\|A\| \|B\|} \tag{2.7}
\]

where “\(\cdot\)” denotes the dot product and “\(\|i\|\)” denotes the magnitude or Euclidean norm of a vector \(i\).

The resulting value ranges from 0 to 1, where 1 indicates that the two vectors are identical or very similar, 0 indicates that the two vectors are orthogonal or dissimilar.

Cosine similarity is a commonly used metric in place recognition to compare the visual appearance of two images. In place recognition, the goal is to determine whether a given query image corresponds to a previously visited location or not. To achieve this, a BoW representation is created for each image, where the image is represented as a histogram of visual words, which are essentially clusters of visual features such as
SIFT or SURF.

Once the BoW representation is obtained for both the query image and a set of reference images, cosine similarity can be used to compute the similarity between them. The cosine similarity score ranges between -1 and 1, where a score of 1 indicates that the two images are identical and a score of -1 indicates that they are completely dissimilar. A higher cosine similarity score between the query and reference images indicates that they are more likely to correspond to the same location.

In practice, place recognition systems often use a threshold on the cosine similarity score to determine whether a match has been found or not. If the score exceeds the threshold, the system may consider the query image as a match to the reference image, indicating that the robot has revisited a previously seen location.

In summary, cosine similarity is a widely used similarity measure that captures the directional similarity of two vectors, and can be applied in various domains such as information retrieval, natural language processing, and recommendation systems.

### 2.3 SLAM

In the field of robotics, SLAM poses a conundrum [61]. For mobile robots to navigate safely in any given environment, they must first be able to understand their surroundings by creating a map of the area. To generate a map, the robot must gather data using a variety of sensors and gradually construct the map. However, the challenge arises after the data has been collected. To assign each piece of data to a particular landmark or location, the robot must determine the location of the data within the map. In essence, the robot must be able to localize itself within the map to generate the map, hence the name “simultaneous localization and mapping”
In literature, there are two main approaches to SLAM: filter-based SLAM and graph-based SLAM, refer Fig. 2.8. Filter-based SLAM uses recursive Bayesian filtering methods to estimate the robot’s pose and the map of the environment. It is one of the early approaches to SLAM and is based on the concept of filtering, where the current state estimate is updated recursively as new sensor measurements arrive. In graph-based SLAM, the trajectory of the robot and the positions of the landmarks in the environment are represented as nodes in the graph.

Additionally, SLAM can be classified based on the type of sensor used, such as visual SLAM, visual-inertial SLAM, and LiDAR SLAM. This thesis focuses on a specific part of graph-based visual SLAM, as it has demonstrated better performance over time. In chapter 4, we will discuss in detail how the generated map is defined in graph-based SLAM [14]. The Fig. 2.9 illustrates the three primary sub-parts of the
full SLAM algorithm. The camera inputs are pre-processed in the front end, which enables the camera to track and estimate its current pose and orientation based on the known environment. This is also referred to as visual odometry. In case the robot revisits a particular location, it should be able to identify that it has been there before based on the generated map and collected data. Moreover, it should be able to determine which part of the map it is correlating with, known as loop closing detection or visual place recognition. Once the entire graph is constructed, it must be optimized based on the relative constraints for minimal translational and rotational loss. This algorithm is called pose graph optimization.

Figure 2.9: Basic RGB-D SLAM architecture divided in front-end and back-end based on the task it performs.

### 2.3.1 Visual Odometry

Visual odometry (VO) involves estimating the ego-motion of a robot using a sequence of images. VO has become increasingly important in robotics, especially when compared to wheel odometry, as it is not affected by wheel slip in uneven or slippery terrain. Furthermore, in GPS-denied environments such as underwater, space exploration, or indoor applications where GPS is available but cannot provide accurate results, VO can be used to estimate relative motion from previously visited locations. However,
VO has its own limitations, such as the need for a sequence of images with sufficient common overlapping texture to calculate geometric relevance. Additionally, it can only function in a well-illuminated environment with visible extractable features. For graph-based SLAM, VO provides the initial estimation of the node and edge attributes.

2.3.2 Loop Closing Detection

LCD is a crucial part of the SLAM algorithm that matches non-sequential entries in the estimated graph. This process helps to prevent the generation of false and redundant graphs and reduces the overall drift error. Fig. 1.1 provides an example of an estimated trajectory from VO, where the red solid lines represent the detected loop closures. Using these detected loops between query and matching nodes, new edges connecting these nodes must be added to the graph in order to overall pose-graph optimization. Fig. 1.1(b) depicts the corrected odometry based on the LCD, which clearly demonstrates its importance.

2.3.3 Pose Graph Optimization

Pose graph optimization is the final step of the graph-based SLAM algorithm. At this stage, all the nodes have their pose (position and orientation), and the edges have the relative transformation from the source node to the destination node. The objective is to maximize the likelihood of the graph configuration such that all the nodes and their relative transformations satisfy the minimum log likelihood, this will be demonstrated in Chapter 4. Fig. 2.10 shows the importance of the pose graph optimization which can optimize the map even with the noisy odometry.
2.4 Visual SLAM Algorithms

Since the inception of SLAM almost three decades ago [64], a plethora of algorithms have been proposed. To streamline the discussion, this section focuses solely on visual SLAM approaches that can generate both 2D and 3D output, considering their applicability in real-time environments for mobile robots and autonomous vehicles.

Visual SLAM algorithms utilize image inputs captured from cameras. Depending on the type of camera used, these algorithms can be classified into mono, stereo, and RGB-D. Mono camera-based algorithms are prone to errors as they are unable to provide depth information, resulting in scale ambiguity. Stereo cameras provide left and right images simultaneously, allowing for depth information to be calculated using various algorithms, but this increases computational cost. The introduction of RGB-D cameras, such as Microsoft’s Kinect, has opened a new branch of research in SLAM, as RGB-D inputs provide feature-rich images with per-pixel depth information in real-time, eliminating the need for many pre-processing steps. Since RGB-D cameras provide depth information as a point cloud, similar to LiDAR, it is possible to implement all LiDAR-based algorithms on RGB-D inputs.
2.4.1 Conventional SLAM Algorithms

2.4.1.1 DVO-SLAM

Dense visual odometry SLAM was a direct SLAM approach that works well in scenes with minimal textures and structures but mainly designed for indoor environment. The rationale is that it minimizes both errors, photo-metric error and depth error to utilize the feature-rich image data directly for higher pose accuracy. DVO-SLAM [8] is a graph-based SLAM algorithm [14] where the pose estimation is proposed by entropy-based key-frame selection and for the backend it utilized general framework for graph-optimization (g2o) [7]. Map is presented in a form of keyframe for specific camera pose, the consecutive keyframes added an edge based on the relative transformation. All the candidate around current keyframe in a sphere with predefined radius is considered for a loop closing detection. Entropy metric is used for keyframes selection in validating loop closure. New detected and validated loop closures added additional edges in the graph. In final stage, the corrected trajectory enabled to construct a consistent point cloud model of a scene. Fig. 2.11 shows DVOSLAM output graph of “fr3/office” sequence of TUM-RGBD dataset.

2.4.1.2 RGBDSLAMv2

RGBDSLAMv2 [9] is a full SLAM algorithm that generates highly accurate 3D maps using an RGB-D camera. This graph-based SLAM extracted standard features such as SIFT [34], SURF [35], ORB [36], and the combination of Shi-Tomashi and SURF. Then, the various distance was computed based on the feature used in estimating the relative transformation. In [9], a beam-based environment measurement model (EMM) has been proposed to validate transformation irrespective to the estimation
methods, RANSAC or ICP. EMM penalizes the poses and transformation that might be happened during the image capturing process such as occlusion, motion blur, or low overlap between the frames. To detect loop closure, the current frame was matched with three different types of candidates: $n$ immediate neighbor in egomotion estimation, graph-neighbor of the previous frame, and randomly $k$ biased frames to the earlier frames [9]. Due to this strategy, runtime increased linearly with the value of parameter $n$ and $k$. So, it was crucial to choose the value of $n$ and $k$ for better system performance. Both of these value can be lower for a short and feature-rich environment, but they should be larger for a complex and longer environment. At last, the graph generated by front-end was optimized by g2o framework to compute consistent global trajectory. Point cloud representation can be created by projecting original point measurement into a common coordinate frame of the global trajectory. RGBDSLAMv2 reconstruction output for TUM-RGBD dataset sequence “fr1/desk” is shown in Fig. 2.12.
2.4.1.3 RTAB-MAP

RTAB-MAP [10] is a graph-based SLAM algorithm that works with any kind of sensor that is able to provide odometry data, which means it follows sensor agnostic odometry and can work with stereo, RGB-D as well as 2D or 3D LiDAR inputs. Map is structured in graph with nodes and links and to control graph optimization processing time for long-term and large-scale environment. It is implemented with memory management approach that divides available memory in two parts: short-term memory (STM) and long-term memory (LTM). Node stores the current robot states such as odometry pose, orientation, sensors’ input and other additional information that can be useful in the loop closer detection and local/global mapping optimization. A link contains the rigid transformation between two nodes. Moreover, these links are divided in three different categorizes based on its location and purpose, namely, neighbor, loop closure, and proximity links. All the links will be used as a constraint for graph optimization. Tuning a lot of parameters to apply this method might be cumbersome at some moment. 3D reconstruction mapping in the form of OctoMap [65] is shown in Fig. 2.13.
2.4.2 Deep Learning-based SLAM Algorithms

Lot of current research is going on in learning-based SLAM methods including the end-to-end deep learning framework and hybrid models. However, none of them are able to implement all the important aspects of full SLAM system, but considering the superior performance of the CNN in image classification task, it is important to study the progress in this direction. Some of the approaches can generate 3D output with integration with conventional methods, such two approaches namely CodeSLAM [11] and CNN-SLAM [12] are explained in more depth.

2.4.2.1 CodeSLAM

CodeSLAM [11] is a compact, yet dense scene geometry representation formed by training an auto-encoder network with a small number of parameters on depth images. While a simple auto-encoder may oversimplify natural scene reconstruction, this approach used image intensity data to condition the training. The depth map estimate
for a key-frame is a function of the corresponding intensity image and a compact representation. Because of its small size, the code can be used as a dense representation of the geometry in inference algorithms, allowing a full joint estimation of both camera poses and dense depth maps for multiple overlapping keyframes. As the author claimed, CodeSLAM is the first real-time targeted monocular system to achieve such a tight joint optimization of motion and dense geometry. However, the fact that this framework can only deal with rotational motion is a limitation and it is not considered as a full SLAM. But, the map representation learning can be implemented with keyframe-based monocular dense SLAM such as PTAM [66] and LSD-SLAM [13].

Figure 2.14: 3D rendering of a scene by CodeSLAM by monocular image [11].
2.4.2.2 CNN-SLAM

CNN-SLAM [12] is a direct monocular SLAM approach that used CNN for predicting depth. This estimated depth is fused with direct monocular SLAM method such as LSD-SLAM to produce a dense scene reconstruction that is both unambiguous in terms of absolute scale and robust tracking. CNN-SLAM is a direct monocular SLAM approach that uses highly appreciated CNN for computer vision tasks to predict depth and fusing it with depth estimated from direct monocular SLAM method such as LSD-SLAM to produce a dense scene reconstruction that is both unambiguous in terms of absolute scale and robust tracking.

Specifically, in [12], a ResNet-50 [39] structure in backbone and up-sampling, dropout before final convolutional layers has been used and the output was a 1-channel depth map. Huber loss function with stochastic gradient descent (SGD) was used for network optimization. Moreover, the ratio between focal length of the current camera, $f_{\text{cur}}$ and the camera used for training was introduced to overcome the absolute scale inaccuracy for the 3D reconstruction resulting from different intrinsic parameters of the cameras. Since, the depth prediction using CNN and depth refinement using monocular SLAM can be carried parallelly on the different computational resources called GPU and CPU, respectively, this framework can run in real-time. Due to this fusion, CNN-SLAM is capable of obtaining a dense depth along with a texture-less environment and deals with pure rotational motions. In addition, CNN-based method was utilized to fuse learned semantic segmentation to create 3D global model with semantic labels. Again, just like CodeSLAM, CNN-SLAM is not a full SLAM system and it needs to integrate with other monocular SLAM methods like LSD-SLAM. Fig. 2.15 shows an accurate scale estimation using CNN-SLAM and LSD-SLAM.
2.5 Conclusion

In this chapter, a summary is presented that encompasses the fundamental concepts required to comprehend deep learning and its applications in computer vision tasks within the context of SLAM and loop closing detection. An introductory examination of model architectures employed in deep learning, including feed-forward neural networks, CNNs, and GNNs, is provided. Subsequently, the conventional and deep learning-based techniques for feature extraction in the computer vision domain are expounded upon, along with an elucidation of commonly utilized backbone models for feature extraction. Additionally, an overview of the sub-components of SLAM and a discussion on state-of-the-art SLAM methods are presented to establish the foundation for the necessity of a fully deep learning-based SLAM system. The subsequent chapter delves into a detailed exploration of the LCD algorithm.
Chapter 3

Related Work

In order to comprehend the methodology and implementation of graph-based loop closing detection, an in-depth study was conducted on the current state-of-the-art approaches, which are summarized here along with a detailed overview of conventional, deep learning-based, and graph-based loop closing detection methods.

3.1 LCD

In order to achieve a comprehensive map derived solely from visual sensing, it is imperative to establish an appropriate representation of the recorded data. Consequently, it is common practice for numerous pipelines to rely on feature vectors extracted from images to effectively describe the path taken, capitalizing on their discriminative capabilities. This approach is further extended to the domain of visual loop closure detection, where the careful selection of an efficient visual feature encoder assumes utmost importance. While traditional methodologies encompass the manual design of hand-crafted features tailored to extract specific image attributes, recent advancements in deep learning have prompted a shift in focus towards learned features extracted
from CNN activations. These learned features have exhibited remarkable performance across various computer vision tasks. In the context of graph SLAM, which utilizes a graph data structure to represent the neighborhood of each visual feature, several methods have leveraged graph techniques to achieve loop closure prediction. Overall, loop closure detection techniques have been categorized into conventional methods, deep learning-based methods, and graph-based methods, each leveraging distinct characteristics to tackle the challenge at hand.

3.1.1 Conventional LCD Methods

In the context of visual loop closure detection, hand-crafted feature-based representations are commonly employed. These representations involve algorithms designed to extract features from images, enabling effective representation. There are two primary types of feature extractors discussed: local and global.

The local feature extractors are more concentrated on particular region-of-interest in the image. The SIFT [67] identifies region-of-interests and thereafter describes them by using difference of gaussian (DoG) function to compute gradient and neighborhood interpretation of the image; SURF [35] gives faster extraction by using approximation of the determinant of Hessian blob detector to find the region-of-interest and then use the sum of Haar wavelet response to describe the features. The KAZE [68] detects and describes the image in a nonlinear scale space. Various extensions of binary descriptor BRIEF [69], such as ORB [36], BRISK [70], FREAK [71], and M-LDB [72] were introduced to reduce complexity and memory requirements by using simple intensity difference tests to describe the region-of-interest. The local extractors are more known for their robustness against the change of scale and rotation.

The global features are known to extract an all-inclusive interpretation of an
image. Gist [73] and WI-SURF [74] are two global feature extractors. Gist is the most commonly used feature extractor in LCD pipelines and it uses image gradients extracted from Gabor filters to create a compact representation of the image. The WI-SURF is a descriptor that describes the whole image using the SURF algorithm. These extractors are well-known for their computational efficiency, faster indexing and lower storage consumption. Moreover, there are several techniques based on histogram statistics such as color histograms [75] which captures color distribution, HOG which computes pixel gradients [76], [77], pyramid-of-HOG (PHOG) [78] which considers local shape and spatial layout for image description. In several works, customized descriptors [79], PCA-based descriptors [80] were employed to reduce the descriptor size for memory optimization. FABMAP [1] is the standard benchmark LCD algorithms in which two publicly available datasets, namely, New College (NC) and City Center (CC) with GPS ground truth were released. In FABMAP algorithm, the Chow-Liu tree searching algorithm was used on SURF-extracted features for the feature matching and also for fast image matching retrieval for the queried image. VBoW [81] model represents images as an aggregation of quantized local features called ”visual words” using a ”visual vocabulary” database. FABMAP used real valued visual words while [82] used binary ones. The use of VBoW is an efficient method for loop closure detection with local features, however, it has two main weaknesses. Firstly, the visual vocabulary is usually generated beforehand and remains fixed during navigation, which limits adaptability to the operational environment and hampers overall performance. Secondly, vector quantization employed in BoW disregards geometric information, leading to reduced discriminative capabilities, particularly in cases of perceptual aliasing. To overcome these limitations, incremental approaches have been proposed, gradually generating the visual vocabulary along the navigation
3.1.2 Deep Learning-based LCD Methods

The learned feature-based representation, as the name suggests involves deep learning strategy because the features can be extracted through data-driven methods by capturing complex patterns and relations in an image from huge datasets. This representation technique can effortlessly deal with occlusions, incorporate geometric information, and exhibit invariance to image transformations. Hence, deep learning models are acknowledged to have superior adaptability and generalization. With the advent of more computational power and the excellent performance of CNN-based deep learning models in image classification tasks [40], many LCD approaches have been developed utilizing various CNN architectures.

Utilization of CNN for learning visual features with a high level of abstraction is remarkable where it introduces four paradigms of feature extractions using trained CNN models, namely; image-based features, pre-defined region-based features, extracted region-based features and lastly, extracted simultaneously image-based and region-based features.

The first paradigm involves feeding the entire image to the network and using activations from the last hidden layer which serves as a compact representation of the image and acts as image descriptor to extract relevant features. The utilization of learned features extracted from all layers of a pre-trained network [84] for object recognition to detect similar locations was pioneered by Chen et al. [85]. However, experimentation with convolution architecture [86] to extract features from the intermediate layers was carried out in [87], and [88] in which they showed that with or without the fully connected layers of the CNN, high-performance results can be
achieved.

The second paradigm focuses on identifying specific region-of-interest by perceiving image landmarks and performing semantic segmentation to extract local features for creating comprehensive representation. One of the common methods [89] used is concatenating descriptors from semantic histograms and HoG into one vector, VLASE [90] relied on semantic edges for the image’s description.

While the above methods extract learned pre-defined features from the landmarks, the third paradigm identifies salient regions and uses them as peculiar features. It includes various methods like regions of maximum activated convolutions (R-MAC) [91], multiscale super-pixel grid (SP-Grid) [92], deep local features (DELF) [93] and D2-net [94] which selected prominent regions and extracts localized features in an image for effective LCD. In [95], a stacked denoising auto-encoder (SDA) was trained on the TUM-RGBD [96] dataset for feature extraction. In [97] and [98], the training strategy was modified to obtain local features. In [97], an unsupervised auto-encoder was used to extract HOG descriptors. Whereas in [98] a combination of supervised-unsupervised model was trained. The supervised model was used to identify mobile objects in image patches, and the unsupervised model was used for unseen image detection, so called novelty detection. In this method, in order to have a fast image retrieval, a super-dictionary was used. The drawback of both [97], and [98] methods is that they require rigorous training before utilizing them for LCD. In [2], in order to extract global features, DCGAN descriptor was trained on the Places dataset [99] using local patches extracted from SURF descriptor. In [3], GAN model architecture was modified to extract binary feature descriptor. Due to the complex architecture of DCGAN, this method is computationally expensive and requires more computational resources as well.
The final paradigm is the combination of the previous paradigms to simultaneously extract global features by capturing crucial attributes of the entire image in a robust and distinctive manner as well as localize features by prioritizing informative regions of interest and intensifying discriminative power of representation. In models such as HF-Net [100] and DELG [101], in order to create holistic representation a generalized mean pooling was used for global features extraction and attentive selection was used for local features extraction.

3.1.3 Graph-based LCD methods

A number of graph-based algorithms have been proposed in the literature, but they do not construct a graph that can be used in graph SLAM. For instance in [102], a proximity graph was proposed for visual vocabulary called hierarchical navigable small world (HNSW) instead of BoW. Graph diffusion was implemented in [103] to pass feature information to the neighboring nodes and checked temporal consistency to avoid false positives. A graph-based image representation was proposed in X-view [104], which incorporated both the geometry and semantics of the scene. Same kind of approach was followed in SymbioLCD2 [105]. To learn the spatial relationship between extracted features from the image, SymbioLCD2 generated multi-tier graph from the features and compared them to predict the loop closure.

3.2 Conclusion

In conclusion, this chapter provided a comprehensive understanding of various LCD algorithms in the literature. The review of current state-of-the-art approaches, including conventional, deep learning-based, and graph-based methods, has laid a strong
foundation for further research in this field. The forthcoming chapter provides an in-depth overview of the graph SLAM structure and the fundamental components necessary for benchmarking the LCD algorithm.
Chapter 4

Methodology

4.1 Introduction

This chapter focuses on the methodology employed to construct a graph structure for SLAM and the associated optimization techniques. Additionally, it explores the benchmarks for LCD algorithms, encompassing the dataset utilized in this thesis, the ground truth information, and the evaluation metrics employed for assessing the performance of these algorithms.

4.2 Graph SLAM Structure

In graph-based SLAM, the map is represented as a graph data structure as shown in Fig. 4.1, where each pose is defined as a node in the graph and the relative transformations between the nodes are represented as connecting edges. The nodes can have different types of data storage attributes based on the sensor configuration, such as RGB images, RGB-D images, point cloud information, GPS/IMU information, position and orientation details in special orthogonal group in 2 dimension (SO2) or
SO3 for 2D and 3D SLAM respectively.

The edges represent the connectivity between source and destination nodes, which represents the relative movement of the robotic platform. In addition to the relative transformation in terms of rotational and translational motion, the edges also include an information matrix that represents the uncertainty of the measurement error. This information matrix is the inverse of the covariance matrix and is symmetric and positive semi-definite.

During graph optimization, these edge attributes are used as constraints to optimize the entire graph. The optimization process aims to maximize the likelihood of the graph configuration, ensuring that all the nodes and relative transformations satisfy the minimum log-likelihood constraint. Following section explain more in details about pose-graph optimization technique.

### 4.2.1 Graph Optimization

Graph optimization is a non-linear least-square problem. Let, the node $i$ defined as $N_i$, mean “virtual measurement” between nodes $i$, and $j$ defined as $E_{ij}$, and information matrix as $\Omega_{ij}$. Eq. 4.1, and Eq. 4.2 denotes the information stored in nodes and edges of the graph for the 2D and 3D, respectively. Where, $x, y,$ and $z$ are the translational components, and $\theta$ and $q$ are the rotational components, degrees in 2D and quaternion in 3D.

\[
N_i^T = [x_i, y_i, \theta_i]
\]
\[
E_{ij}^T = [x_{ij}, y_{ij}, \theta_{ij}]
\]
\[
\Omega_{ij} = 3 \times 3 \text{ sized information matrix}
\]
Figure 4.1: Constructing graph based on calculated matching score. (a) Matching score is calculated between the current node $I_i$ and all the existing $N$ nodes using Eq. 5.1. (b) False positives shall be removed by applying threshold on matching score, once the whole graph is constructed.
Figure 4.2: Example of a pair of nodes $N_i$ and $N_j$ with its predicted relative transformation $\hat{E}_{ij}$ and measurement $E_{ij}$ with it’s uncertainty referred by information matrix $\Omega_{ij}$. [14]

$$
N_i^T = [x_i, y_i, z_i, q_{xi}, q_{yi}, q_{zi}, q_{wi}]
$$

$$
E_{ij}^T = [x_{ij}, y_{ij}, z_{ij}q_{xij}, q_{yij}, q_{zij}, q_{wij}]
$$

(4.2)

$$
\Omega_{ij} = 7 \times 7 \text{ sized information matrix}
$$

As shown in the Fig. 4.2 virtual measurement $E_{ij}$ is a transformation that makes the observations acquired from $i$ maximally overlap with the observation from $j$, and $\Omega_{ij}$ is the information matrix, which is just the inverse of covariance matrix with the uncertainty of the measurements.

If the predicted relative transformation between two nodes is indicated by $\hat{E}_{ij(N_i,N_j)}$, which solemnly depends on node configuration $N_i$ and $N_j$, the error $e(N_i, N_j, E_{ij})$ can be calculated as per Eq. 4.4. The objective of the optimization approach is to minimize the negative log-likelihood $F(N)$ of all the observations from the set of constraints $C$ and find the node configuration $N^*$. Here, $C$ is the set of all the edges for which a measurement $E$ exists.
\( e_{ij}(N_i, N_j) = E_{ij} - \hat{E}_{ij}(N_i, N_j) \) (4.3)

\[ l_{ij} \propto e_{ij}^T \Omega_{ij} e_{ij} \] (4.4)

\[
F(N) = \sum_{(i,j) \in C} e_{ij}^T \Omega_{ij} e_{ij} \tag{4.5}
\]

\[
N^{*} = \arg\min_{N} F(N) \tag{4.6}
\]

TORO [106], g2o [7], and GTSAM [107] are some of the approaches to optimize pose graph using non-linear solvers. TORO is a tree-based parameterized system that increases the convergence speed to minimize loss function using gradient descent. g2o is a general framework for graph optimization that can be used to optimize a common sparse graph structure in various problem such as SLAM, and bundle adjustment [108]. GTSAM optimize the graph by using factor graph [109] and sparse direct linear solvers [110].

### 4.3 Benchmark

There are three main elements needed for benchmarking any loop closing detection algorithm: 1) Dataset, 2) Groundtruth, and 3) Evaluation metrics. The fundamental overview of every component will be provided in this section.
4.3.1 Datasets

Recent advances in robotics and autonomous vehicle research have made a variety of datasets including groundtruth data readily available in the open-source community. These datasets differ primarily in the type of environment they were collected in—static or dynamic, indoors or outdoors, urban or rural—as well as the platform type—handheld cameras, mobile robots, car mounts, or micro aerial vehicles (MAV) mounts. A sample pair of query and matched images from each dataset is shown in Fig. 4.3, where the first and the second row presents the query and corresponding matched images, respectively. The summary of each dataset is presented in Table 4.1. KITTI [16] vision suits are a well-known benchmark environment in the field of robotics and computer vision. With accurate odometry groundtruth information and data gathered from multisensory suits, including high-resolution stereo cameras, it offers a wide range of sequences. In particular, sequences 0 and 5 are frequently utilized for LCD since they exhibit the actual loop closure frames compared to all 11 sequences that are accessible with odometry data. Another often used dataset in LCD is City Centre and New College [1], which is acquired using a lateral-mounted stereo camera on a robotic platform as it moves over the Oxford campus. With its extensive repeating stretch of medieval buildings and gardens, New College is used to test the system’s robustness against perceptual aliasing. Due to urban traffic and pedestrian movement, City Center has dynamic scene variation. Smith et. el. [111] presented an additional version of the New College vision suite that included a camera with a better frame rate and more sensors. The stereo vision of an electric buggy-type vehicle is used to capture Malaga parking 6L [17] in an outdoor parking lot. A high frame rate camera mounted on an MAV was used to collect the EuROC MAV [18] dataset, which produced sequences with rapidly changing velocity and illumination in
an indoor environment. For LCD, a specific EuROC machine hall 05 sequence is used. KAIST [15] is a collection of images captured along the same route, including both day and night environments.

Table 4.1: Datasets Summary

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Ground truth received from</th>
<th>Description</th>
<th>Image Resolution</th>
<th># Images</th>
<th># Loop Closure Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>New College (NC - reduced) [1]</td>
<td>[2]</td>
<td>Outdoor, dynamic</td>
<td>$480 \times 640$</td>
<td>394</td>
<td>640</td>
</tr>
<tr>
<td>City Center (CC - reduced) [1]</td>
<td>[2]</td>
<td>Urban, dynamic</td>
<td>$480 \times 640$</td>
<td>274</td>
<td>417</td>
</tr>
<tr>
<td>KAIST North (KN) [15]</td>
<td>[2]</td>
<td>Outdoor, Day &amp; Night</td>
<td>$900 \times 1200$</td>
<td>204</td>
<td>510</td>
</tr>
<tr>
<td>KAIST East (KE) [15]</td>
<td>[2]</td>
<td>Outdoor, Day &amp; Night</td>
<td>$900 \times 1200$</td>
<td>200</td>
<td>500</td>
</tr>
<tr>
<td>KAIST West (KW) [15]</td>
<td>[2]</td>
<td>Outdoor, Day &amp; Night</td>
<td>$900 \times 1200$</td>
<td>200</td>
<td>500</td>
</tr>
<tr>
<td>New College (NC) [1]</td>
<td>[1]</td>
<td>Outdoor, dynamic</td>
<td>$480 \times 640$</td>
<td>1073</td>
<td>46723</td>
</tr>
<tr>
<td>City Center (CC) [1]</td>
<td>[1]</td>
<td>Urban, dynamic</td>
<td>$480 \times 640$</td>
<td>1237</td>
<td>8252</td>
</tr>
<tr>
<td>KITTI 00 [16]</td>
<td>[112]</td>
<td>Outdoor, dynamic</td>
<td>$1241 \times 376$</td>
<td>4541</td>
<td>31922</td>
</tr>
<tr>
<td>KITTI 05 [16]</td>
<td>[112]</td>
<td>Outdoor, dynamic</td>
<td>$1241 \times 376$</td>
<td>2761</td>
<td>12348</td>
</tr>
<tr>
<td>Malaga 2009 Parking 6L [17]</td>
<td>[82]</td>
<td>Outdoor, Slightly dynamic</td>
<td>$1024 \times 768$</td>
<td>3474</td>
<td>18919</td>
</tr>
<tr>
<td>EuROC MH 05 [18]</td>
<td>[82]</td>
<td>Indoor, Static</td>
<td>$752 \times 480$</td>
<td>2273</td>
<td>64879</td>
</tr>
</tbody>
</table>

4.3.2 Groundtruth

Groundtruth is an important aspect for evaluating any loop closure techniques. As indicated in Table 4.1, the ground truth data for these datasets is received from the author of [1, 82, 112, 2]. It is worth mentioning that our approach is evaluated using
two different types of ground truth for the City Centre and New College datasets. The ground truth provided in [1] was prepared based on the GPS proximity of images. However, this approach may sometimes classify images as true closures even when they show different scenes, particularly in turning scenario. To address this limitation, we also use a subset of City Centre and New College datasets provided in [2] where the ground truth labels created manually by multiple human. These subsets are referred to as CC-reduced and NC-reduced in our subsequent discussions, while the experiments conducted using the original data are referred to as NC and CC, respectively. NC-reduced, CC-reduced, and KAIST dataset are considered as the image-to-image corresponding matching label, while rest of the datasets are continuous sequential frames.

4.3.2.1 Groundtruth for Image-2-Image Dataset

As shown in Fig. 4.4, an $N \times N$ boolean matrix for $N$ timestamps is used to represent ground truth for the LCD. In this matrix, a cell value of 1 indicates a true loop closing event at the corresponding row and column indices, and 0 otherwise. This ground truth matrix, along with the similarity matrix generated from the system in the experiment, is used to calculate evaluation metrics.
4.3.2.2 Groundtruth for Sequential Dataset

Groundtruth for the sequential dataset is represented as the list of tuples containing starting node and ending node for the query node. As shown in Fig. 4.5, each entry in the list is the individual groundtruth-ID comprised of the query node ID with its matching start node ID and end node ID. The prediction is considered positive if it is in the range between the start and end range or negative otherwise.

Figure 4.5: Custom class representation of the City Centre dataset as the sequential groundtruth where each entry consists of the query node ID, start matching node ID, and end matching node ID from the whole sequence.
4.3.3 Evaluation Metrics

If the two images originated from the same place as indicated by the correspondence matrix of ground truth, then the loop closure event can be classified as positive or negative. A loop closure detection algorithm’s output can also be classified as true or false depending on the data the ground truth provides. The four detection labels that result are true-positive (TP), false-positive (FP), true-negative (TN), and false-negative (FN). True positives are positive loop closing events that the system accurately predicts, whereas false positives are the events that are still predicted even when the image pair is not from the same region. False Negatives are a kind of image pair that actually closes the loop but isn’t recognized by the system. Images that are neither truly closed nor recognized by the system are considered true negatives.

Table 4.2: Loop closure confusion matrix

<table>
<thead>
<tr>
<th>Detected Loop Closure</th>
<th>Positive Loop Closure</th>
<th>Negative Loop Closure</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TP</td>
<td>FP</td>
</tr>
<tr>
<td>Not Detected Loop Closure</td>
<td>FN</td>
<td>TN</td>
</tr>
</tbody>
</table>

Since only true loop closing events are of concern, LCD methods are assessed using precision-recall curves. Precision is the ratio of the system’s correctly closed loops to all of the true loops it has discovered, whereas recall is the ratio of the system’s correctly closed loops to all of the ground truth’s actual loop closures. Mathematically, precision and recall are defined as,

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

The true or false match between the images is defined by an internal system parameter that may be varied to generate the precision-recall curve, which displays
the relationship between these two metrics. Instead of an absolute true (1) or false (0) result, LCD algorithms often produce a similarity matrix with scores ranging from 0 to 1. In order to obtain different combinations of precision and recall, we may also build PR curves by altering the hypothesis threshold over the similarity score. The area under the curve (AUC) is an assessment matrix used to compare two curves, however it does not provide information on the highest precision or recall attained with regard to each other. Maximum recall with 100% precision ($R@P_{100}$) is the crucial indication to assess LCD method because any negative loop closure event recorded as true event by the system (false-positives) can result in a complete failure of the SLAM. When a method is unable to achieve 100% precision, average precision may also be used to illustrate how well it is performing. A better precision for all recall values is one of LCD’s objectives, and average precision can reflect this characteristics.

4.4 Conclusion

In conclusion, this chapter has provided a comprehensive explanation of the fundamental graph structure employed in SLAM and the associated optimization process. Furthermore, it has highlighted the key components required for benchmarking loop closure methods. Chapter 5 outline the procedure to construct the graph structure with feature extraction and calculating similarity score.
Chapter 5

Loop Closing Detection With Constructing Graph Structure

This chapter presents a comprehensive introduction to the proposed framework, experimental setup, and a detailed analysis of the performance comparison based on average precision and run-time. The framework encompasses various components and methodologies that contribute to its overall effectiveness in addressing the research problem. Finally, the performance comparison sheds light on the framework’s efficacy in comparison to alternative approaches.

5.1 Proposed Method

As shown in Fig. 4.1, this chapter proposes a constructive graph framework that is similar to graph SLAM. The framework is modularized into two separate modules: feature extraction and feature matcher. This allows a flexibility in using any feature extraction technique and any feature matcher criteria, independently. In this work, twenty distinct deep learning-based architectures presented in section 2.2.1.3,
namely; AlexNet, VGG, GoogLeNet, Inception V3, ResNet, SqueezeNet, Wide ResNet, DenseNet, ResNeXt, MobileNet V2, ShuffleNet V2, EfficientNet, MNASNet, MobileNet V3, RegNet, Vision transformer, EfficientNet V2, Swin transformer, ConvNeXt, and MaxVit, are utilized for feature extraction. To perform feature matching, a rapid cosine similarity approach is adopted, which calculates a matching score against all other nodes in the graph. Consequently, the retrieval of the matching node is carried out based on this score with minimal additional time. Finally, the current image frame was incorporated into the graph with its calculated matching scores serving as their edge attribute. This approach helps in detecting other nearest neighbors that can be considered as a potential loop closure candidate for the next image frame.

1. Feature Extractor: This module provides flexibility in choosing any method of feature extraction $F(I)$ that takes an input image and produces an array of features as output. In our experiments, we utilized twenty various backbone models based on convolution, visual transformer, and hybrid architecture pre-trained on Imagenet dataset for the task of image classification. We removed the classification head layers and extracted features from the flattened layer after the base feature extractor module.

2. Feature Matching + Node addition: After obtaining all the features from the current image, the next step is to calculate the feature matching score with all the nodes in the existing graph using the cosine similarity method. To achieve fast computation, a modification was made to the traditional one-by-one score calculation method used in [98]. Instead, we converted it into a matrix multiplication form as described in equation 5.1. After calculating the similarity score, the current node is incorporated into the graph by assigning the corresponding matching score as the edge attribute, which determines the
where, $f_{(N \times n)}$ is $n$ dimensional extracted features from already seen $N$ images, $f_{i(1 \times n)}$ is current queried image $i^{th}$ feature, and $S^i$ represents the matching score corresponding to every previously seen image node. Matrix multiplication ($\times$) was utilized to get the final $N$ dimensional vector that represents the dot operation of $N$ vector with current image’s feature. $\|f_{N \times n}\|$ is norm-2 of metrics $f_{(N \times n)}$ calculated on its second dimension. Once $N$ nodes are added to the existing graph, there will be $\frac{n!}{(n-2)!}$ edges with matching scores. To finalize the loop closure, a threshold based on the PR-curve is applied to remove any extra edges.

Algorithm 1 Full pipeline of proposed framework

1: $g \leftarrow \emptyset$ \{Initialize empty graph\}
2: for all image $I$ in Dataset do
3: \hspace{1em} $f = \mathcal{F}(I)$
4: \hspace{1em} $S_i = Mx(g, f)$ \{Mx is implementation of equation 5.1.\}
5: \hspace{1em} $g$.add_node($f, S_i$) \{Add current node to the graph with its features and calculated score.\}
6: end for
7: \hspace{1em} return Constructed graph $g$

5.1.1 Environmental Setup

Before starting any experiment, the environmental setup plays a crucial role in the performance and time requirements for the models. During the initial stages of the research, the experiments were performed on the local machine using the Jupyter Notebook environment. Due to the advantage of cloud resources, which are faster, the
research experiments were moved to Compute Canada server. The availability of a cloud environment and GPU reduced the time required for training significantly. The environment used during the experiments is discussed below.

- IDE: Visual Studio Code with extension: Jupyter Notebook, Python
- GPU: 12GB Tesla P100
- CPU: 2×Intel E5-2683v4, 2.1 GHz

In the next section, the packages used during the research are analysed and described in detail.

5.1.2 Python Packages

During the experiments, different Python packages/libraries are used.

- Numpy: A fundamental package for scientific computing in Python, providing support for powerful array operations and mathematical functions. This package is used to perform all kind of array manipulations.

- SciPy: A library built on top of Numpy, offering a wide range of scientific and numerical computing tools, including optimization, linear algebra, and signal processing. This library is used to save and load prediction matrices and groundtruth in .mat files. Furthermore, it enables the conversion of matrices to sparse matrices, which is useful for providing input to the edges of a torch-geometric graph.

- Matplotlib: A comprehensive plotting library for creating static, animated, and interactive visualizations in Python, enabling the creation of various types
of charts, graphs, and plots. This library is utilized to plot PR-curve, and
visualizing prediction matrices.

- **Scikit-learn**: A popular machine learning library that provides a wide range
  of algorithms and tools for tasks such as classification, regression, clustering,
  dimensionality reduction, and model evaluation. It is utilized to calculate
  performance matrices such as precision, and recall.

- **PyTorch**: A deep learning framework that provides efficient tensor computations
  and automatic differentiation, enabling the implementation of neural networks
  and other machine learning models. PyTorch is utilized as a base of deep-learning
  framework for all of the tensor operations and manipulation.

- **torchvision**: A package that provides datasets, model architectures, and com-
  mon image transformations for computer vision tasks, designed to work seam-
  lessly with PyTorch. This library is utilized to prepare data loader, backbone
  models, and pre-processing transforms.

- **torch-geometric**: A PyTorch-based library for deep learning on graphs and
  other irregular structures, offering a wide range of graph operations, dataset
  handling, and model building blocks. Pytorch-geometric is used extensively for
  constructing incremental graphs and building GNN.

- **wandb**: A tool for visualizing, tracking, and analyzing machine learning exper-
  iments, allowing users to log and monitor experiments, compare models, and
  collaborate with team members. All the experiments mentioned in this work are
  saved and tracked on wandb, it very convenient to compare the experimental
  results and visualize the predictions, especially while working in remote server.
5.2 Experimental Results

This section presents a concise overview of the experimental procedure, and the comparative analysis of results with different performance metrics.

5.2.1 Method Evaluation

In this work, twenty distinct deep learning-based architectures, as explained in section 2.2.1.3, are implemented for feature extraction, and evaluated on five image-2-image matching paired datasets. In order to compare the effect of different feature extractor on the performance of the proposed LCD method, we compare them in the terms of average precision as presented in Table 5.1, and PR curve as shown in 5.3. To check the consistency of the experimented feature extractors over all the dataset, box plot of average precision is shown in Fig. 5.2. As it is shown in Table 5.1, the maximum average precision can be achieved using different feature extractors on different datasets, namely, NC-reduced, CC-reduced, KN, KE, and KW are 75.38%, 84.78%, 63.98%, 74.68%, and 73.70%. Based on this result, it can be assumed that KN is the most complex dataset that makes it difficult to achieve higher average precision compared to the other datasets. Among all the tested models, ShuffleNet based models perform better than others across three dataset followed by AlexNet and VGG. On the other hand, transformer-based models like ViT, MaxViT, and SWIN, known for their superior performance in classification tasks, do not yield comparable results in this context. This discrepancy could be attributed to the fact that the intermediate feature vectors extracted by these models do not exhibit similar behavior to CNN-based features. Essentially, it could be a consequence of the fundamental distinction between the operations of CNNs (convolutional operations)
and transformers (attention operations). CNNs utilize convolution operations to capture local patterns and spatial hierarchies in images. They are designed to exploit the grid-like structure of images. ViTs, on the other hand, are based on self-attention mechanisms. They process the entire image as a sequence of patches and can capture global context and long-range dependencies between image regions. Generalization property of CNNs and ViTs also can be an important factor as ViTs requires more task-specific fine-tuning considering their larger number of parameters compared to CNNs [113].

Table 5.1: The performance comparison of the different feature extractors in the term of average precision

<table>
<thead>
<tr>
<th>Feature Extractor</th>
<th>Dataset</th>
<th>NC-reduced</th>
<th>CC-reduced</th>
<th>KN</th>
<th>KE</th>
<th>KW</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet</td>
<td></td>
<td>0.73038</td>
<td><strong>0.84783</strong></td>
<td>0.61983</td>
<td>0.67869</td>
<td>0.71510</td>
</tr>
<tr>
<td>ConvNext</td>
<td></td>
<td>0.74387</td>
<td>0.81433</td>
<td>0.56564</td>
<td>0.58683</td>
<td>0.62304</td>
</tr>
<tr>
<td>DenseNet</td>
<td></td>
<td>0.75196</td>
<td>0.82732</td>
<td>0.60088</td>
<td>0.60907</td>
<td>0.64755</td>
</tr>
<tr>
<td>EfficientNet</td>
<td></td>
<td>0.73813</td>
<td>0.81510</td>
<td>0.57422</td>
<td>0.57531</td>
<td>0.62565</td>
</tr>
<tr>
<td>EfficientNetV2</td>
<td></td>
<td>0.72986</td>
<td>0.79980</td>
<td>0.54751</td>
<td>0.56519</td>
<td>0.60431</td>
</tr>
<tr>
<td>GoogLeNet</td>
<td></td>
<td>0.74205</td>
<td>0.82191</td>
<td>0.57689</td>
<td>0.60982</td>
<td>0.64997</td>
</tr>
<tr>
<td>InceptionV3</td>
<td></td>
<td>0.74644</td>
<td>0.82173</td>
<td>0.56749</td>
<td>0.61719</td>
<td>0.67352</td>
</tr>
<tr>
<td>MaxVit</td>
<td></td>
<td>0.73138</td>
<td>0.79456</td>
<td>0.54796</td>
<td>0.54568</td>
<td>0.59411</td>
</tr>
<tr>
<td>MNASNet</td>
<td></td>
<td>0.74559</td>
<td>0.82897</td>
<td>0.56232</td>
<td>0.57098</td>
<td>0.60499</td>
</tr>
<tr>
<td>MobileNetV2</td>
<td></td>
<td>0.74648</td>
<td>0.82916</td>
<td>0.56466</td>
<td>0.59486</td>
<td>0.62011</td>
</tr>
<tr>
<td>MobileNetV3</td>
<td></td>
<td>0.75302</td>
<td>0.82698</td>
<td>0.57356</td>
<td>0.57430</td>
<td>0.60789</td>
</tr>
<tr>
<td>RegNet</td>
<td></td>
<td>0.75226</td>
<td>0.84451</td>
<td>0.59913</td>
<td>0.60346</td>
<td>0.60524</td>
</tr>
<tr>
<td>ResNet</td>
<td></td>
<td>0.73878</td>
<td>0.82412</td>
<td>0.53655</td>
<td>0.57300</td>
<td>0.59432</td>
</tr>
<tr>
<td>ResNeXt</td>
<td></td>
<td><strong>0.75385</strong></td>
<td>0.82897</td>
<td>0.57673</td>
<td>0.59530</td>
<td>0.63220</td>
</tr>
<tr>
<td>ShuffleNetV2</td>
<td></td>
<td>0.72473</td>
<td>0.84440</td>
<td><strong>0.63987</strong></td>
<td><strong>0.74684</strong></td>
<td><strong>0.73698</strong></td>
</tr>
<tr>
<td>SqueezeNet</td>
<td></td>
<td>0.73904</td>
<td>0.82830</td>
<td>0.56597</td>
<td>0.57134</td>
<td>0.61261</td>
</tr>
<tr>
<td>SWIN</td>
<td></td>
<td>0.74760</td>
<td>0.80589</td>
<td>0.51896</td>
<td>0.53047</td>
<td>0.56503</td>
</tr>
<tr>
<td>VGG</td>
<td></td>
<td>0.73409</td>
<td>0.83976</td>
<td>0.61487</td>
<td>0.67321</td>
<td>0.6860</td>
</tr>
<tr>
<td>ViT</td>
<td></td>
<td>0.73549</td>
<td>0.80819</td>
<td>0.54917</td>
<td>0.54669</td>
<td>0.59033</td>
</tr>
<tr>
<td>WideResNet</td>
<td></td>
<td>0.74842</td>
<td>0.82263</td>
<td>0.57156</td>
<td>0.59852</td>
<td>0.63451</td>
</tr>
</tbody>
</table>

For the visual comparison, correspondence matrix predicted by all the models
for the NC-reduced dataset is shown in Fig. 5.1, refer Fig. 4.4 for the respective
groundtruth matrix. The findings presented in Fig. 5.1(n) align with the results
depicted in Table 5.1, as the correspondence matrix predicted by ResNeXt clearly
highlights the loop closure pattern with a distinct bright stretch. In contrast, such
resemblance is not apparent in the case of SqueezeNet and ShuffleNet, leading to lower
average precision scores for that specific dataset. PR curve of the five models, to avoid
in-distinctive presentation, is presented in Fig. 5.3(a)-(e). ShuffleNet stands out as the
top performer in the KAIST dataset, exhibiting a notably higher margin compared to
the other models. Additionally, it achieves relatively similar results for the CC-reduced
and NC-reduced datasets, demonstrating its consistency across different scenarios.

Based on the obtained results as shown in Fig. 5.2, we utilized ShuffleNet backed
models to extract features and evaluated their performance on the remaining sequential
datasets. ShuffleNet consistently outperformed other methods in terms of mean average
precision over all the dataset in experiment, as seen in Fig. 5.2. The PR curve for
these datasets is presented in Fig. 5.3(f). The PR curve clearly demonstrates that
the proposed model performs well on the City Center and KITTI-00/05 sequences.
However, it exhibits a degradation in performance on the New College, MALAGA, and
EuROC datasets. This degradation can be attributed to the high similarity present
in these scenes, leading to false positives. For instance, in the New College dataset,
the majority of the images consist of plain greenery with distant trees, while the
MALAGA dataset contains repetitive parking lot images. Additionally, the EuROC
MH 05 sequence captures indoor scenes with limited exposure to new scenes. The
ground truth annotations fail to capture this information accurately, as they strictly
rely on GPS location within a certain radius to determine true matches. Despite these
challenges, the proposed method shows improved performance compared to other
Figure 5.1: Visualization of correspondence matrix predicted by various backbone models on the New College dataset.
Figure 5.2: Performance comparison of the various feature extractors based on the achieved mean average precision and standard variation on five sequence of image-2-image dataset.

methods, as illustrated in Section 5.2.2.

Finally, Fig. 5.4 displays the predicted loop closures for the CC, NC, KITTI-00, and KITTI-05 sequences by utilizing odometry information from each dataset. The first column shows the ground truth for each sequence, while the second column shows the predicted loop closures generated by the proposed framework. Following that, a binary matrix is presented, obtained by applying a threshold to the calculated matching scores to maximize the average precision. Each entry in this matrix signifies a predicted loop closure between nodes $i$ and $j$ if the value in the corresponding cell is 1 at index $i$ of the row and index $j$ of the column.
Figure 5.3: (a)-(e) The performance comparison of the different feature extractor with image-2-image corresponding labels in terms of PR curves on different experimental datasets. (f) The performance comparison in terms of PR curve on different sequential datasets after selecting the best feature extractor from previous results.
Figure 5.4: First column is the ground truth trajectories, second column is the final predicted trajectories, and the third column is the final correspondence matrices of the (a) New College, (b) City Centre, (c) KITTI-00, and (d) KITTI-05 datasets.
5.2.2 Comparative Results

Table 5.2 provides a comparison of the average precision and recall of the proposed method with that of [2], [1], and [3]. [2] and [3] was used to compare the results with the proposed method as they’re using the same reduced subset of New College and City Center dataset. We achieved higher average precision in comparison with [1] on all tested datasets. The results show that [3], and [2] performs slightly better on four sequences where the features are extracted from a GAN [114] based models trained on Places360 [115] dataset, whereas we have used models trained on Imagenet [40] which is more suited for object classification rather than place identification [87]. Despite these limitations, we achieved superior results by providing higher average precision on KAIST-East and KAIST-West datasets in comparision to [2].

Table 5.2: The performance comparison between methods proposed in [1], [2], [3] and ours in terms of average precision.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NC-reduced</td>
<td>0.2337</td>
<td><strong>0.7890</strong></td>
<td>0.7520</td>
<td>0.7247</td>
</tr>
<tr>
<td>CC-reduced</td>
<td>0.3413</td>
<td>0.8500</td>
<td><strong>0.8580</strong></td>
<td>0.8440</td>
</tr>
<tr>
<td>KAIST-North</td>
<td>0.1720</td>
<td>0.6400</td>
<td><strong>0.6560</strong></td>
<td>0.6398</td>
</tr>
<tr>
<td>KAIST-East</td>
<td>0.2098</td>
<td>0.6640</td>
<td>0.7190</td>
<td><strong>0.7460</strong></td>
</tr>
<tr>
<td>KAIST-West</td>
<td>0.3751</td>
<td>0.7007</td>
<td><strong>0.7500</strong></td>
<td>0.7369</td>
</tr>
<tr>
<td>NC</td>
<td>0.1640</td>
<td>-</td>
<td></td>
<td><strong>0.3240</strong></td>
</tr>
<tr>
<td>CC</td>
<td>0.0825</td>
<td>-</td>
<td></td>
<td><strong>0.7134</strong></td>
</tr>
<tr>
<td>KITTI-00</td>
<td>0.0096</td>
<td>-</td>
<td></td>
<td><strong>0.7173</strong></td>
</tr>
<tr>
<td>KITTI-05</td>
<td>0.0074</td>
<td>-</td>
<td></td>
<td><strong>0.6932</strong></td>
</tr>
<tr>
<td>MALAGA</td>
<td>0.0065</td>
<td>-</td>
<td></td>
<td><strong>0.1760</strong></td>
</tr>
<tr>
<td>EuROC</td>
<td>0.0186</td>
<td>-</td>
<td></td>
<td><strong>0.0292</strong></td>
</tr>
</tbody>
</table>
5.2.3 Time Complexity

Based on the experimental setup, our proposed method was evaluated using 12GB Tesla P100 GPU and 2×Intel E5-2683 v4, 2.1 GHz CPU. The results are presented in the Table 5.3, which shows the comparison of execution time per instance for different models. As per Table 5.3, AlexNet model is the fastest among all the models evaluated, surpassing even the SqueezeNet model, despite SqueezeNet having a lower number of parameters, however, the runtime difference between these two model is not significant compared to the runtime difference between AlexNet with rest of the models. The slowest model was SWIN, which took average $\sim$60ms per instance. However, this is still $\sim$30 times faster than the method proposed in [2], which has an execution time of 1956ms per image, and achieved matching speed of [102].

This fast execution time makes our method suitable for real-time loop closure detection. It is worth noting that this time includes the operation of adding new nodes to the existing graph with predicted similarity scores. Table 5.4 provides a comparison of the time complexity results between the proposed method with and FABMAP [1]. The table clearly demonstrates that the presented method surpasses FABMAP by a significant margin. The proposed method achieves a minimum FPS of approximately 22 frames per second, making it highly suitable for real-time loop closure detection applications. It is important to acknowledge that as the number of images increases, the computation time required to calculate the similarity scores also increases. In other words, the runtime of this method grows linearly with the number of images, making it slower over time, similar to other existing approaches.
Table 5.3: The performance comparison of the different feature extractors in the term of time complexity (in ms)

<table>
<thead>
<tr>
<th>Feature Extractor</th>
<th>Dataset</th>
<th>NC-reduced</th>
<th>CC-reduced</th>
<th>KN</th>
<th>KE</th>
<th>KW</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet</td>
<td></td>
<td>10.65428</td>
<td>14.36624</td>
<td>30.06843</td>
<td>30.46689</td>
<td>30.76174</td>
</tr>
<tr>
<td>ConvNext</td>
<td></td>
<td>25.51384</td>
<td>27.81957</td>
<td>37.59844</td>
<td>39.10682</td>
<td>38.76155</td>
</tr>
<tr>
<td>DenseNet</td>
<td></td>
<td>45.29608</td>
<td>48.73569</td>
<td>59.04711</td>
<td>54.77611</td>
<td>57.99073</td>
</tr>
<tr>
<td>EfficientNet</td>
<td></td>
<td>49.07564</td>
<td>50.02732</td>
<td>60.64234</td>
<td>59.98627</td>
<td>62.84634</td>
</tr>
<tr>
<td>EfficientNetV2</td>
<td></td>
<td>52.04248</td>
<td>57.07138</td>
<td>65.00323</td>
<td>63.92730</td>
<td>64.84694</td>
</tr>
<tr>
<td>GoogLeNet</td>
<td></td>
<td>21.80536</td>
<td>23.45341</td>
<td>34.58851</td>
<td>34.53454</td>
<td>34.54299</td>
</tr>
<tr>
<td>InceptionV3</td>
<td></td>
<td>28.70002</td>
<td>31.10860</td>
<td>40.70237</td>
<td>42.46590</td>
<td>40.52316</td>
</tr>
<tr>
<td>MaxVit</td>
<td></td>
<td>48.99947</td>
<td>50.32503</td>
<td>60.85149</td>
<td>60.33203</td>
<td>62.54698</td>
</tr>
<tr>
<td>MNASNet</td>
<td></td>
<td>18.84381</td>
<td>20.36932</td>
<td>33.10469</td>
<td>33.52211</td>
<td>33.42363</td>
</tr>
<tr>
<td>MobileNetV2</td>
<td></td>
<td>17.97042</td>
<td>19.68180</td>
<td>33.50924</td>
<td>33.92972</td>
<td>33.80170</td>
</tr>
<tr>
<td>MobileNetV3</td>
<td></td>
<td>20.52348</td>
<td>21.66523</td>
<td>33.54364</td>
<td>33.87035</td>
<td>33.71751</td>
</tr>
<tr>
<td>RegNet</td>
<td></td>
<td>33.81929</td>
<td>35.39069</td>
<td>45.63184</td>
<td>45.45267</td>
<td>45.64172</td>
</tr>
<tr>
<td>ResNet</td>
<td></td>
<td>19.87066</td>
<td>21.97088</td>
<td>35.61298</td>
<td>34.69685</td>
<td>34.09175</td>
</tr>
<tr>
<td>ResNeXt</td>
<td></td>
<td>33.05311</td>
<td>34.03183</td>
<td>43.74256</td>
<td>44.20166</td>
<td>44.86707</td>
</tr>
<tr>
<td>ShuffleNetV2</td>
<td></td>
<td>18.21042</td>
<td>20.21818</td>
<td>32.62840</td>
<td>33.06572</td>
<td>35.33777</td>
</tr>
<tr>
<td>SWIN</td>
<td></td>
<td>54.78252</td>
<td>55.43728</td>
<td>66.99387</td>
<td>64.84659</td>
<td>67.39919</td>
</tr>
<tr>
<td>VGG</td>
<td></td>
<td>14.06294</td>
<td>15.54583</td>
<td>31.54195</td>
<td>31.17478</td>
<td>33.95584</td>
</tr>
<tr>
<td>ViT</td>
<td></td>
<td>20.79017</td>
<td>22.27780</td>
<td>33.03259</td>
<td>33.95116</td>
<td>33.21701</td>
</tr>
<tr>
<td>WideResNet</td>
<td></td>
<td>31.75418</td>
<td>34.60004</td>
<td>46.43120</td>
<td>43.81653</td>
<td>44.23269</td>
</tr>
</tbody>
</table>

Table 5.4: Overall time-complexity comparison between our proposed method and FABMAP [1]

<table>
<thead>
<tr>
<th>FABMAP [1]</th>
<th>Proposed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total time (s)</td>
<td>FPS</td>
</tr>
<tr>
<td>KITTI00</td>
<td>1360</td>
</tr>
<tr>
<td>KITTI05</td>
<td>834.8</td>
</tr>
<tr>
<td>EuROC</td>
<td>1627.9</td>
</tr>
<tr>
<td>NC</td>
<td>294.25</td>
</tr>
<tr>
<td>CC</td>
<td>379</td>
</tr>
<tr>
<td>MALAGA</td>
<td>1267.2</td>
</tr>
</tbody>
</table>

71
5.3 Conclusion

In conclusion, this chapter introduces a framework for efficient loop closure detection in SLAM. The proposed framework addresses the need for a parameter-free and modularized system that can be utilized in real-time scenarios, leveraging the computational power of GPUs. By extracting features using deep-learning based backbone from input images and calculating matching scores, the framework constructs a graph structure that captures the connectivity of nodes based on the matching criteria. This graph structure enables rapid retrieval of matching nodes. Through extensive evaluation using various deep learning-based feature extractors on a set of sequences from an open-source dataset, the proposed framework demonstrates faster processing times with higher precision-recall scores when compared to alternative approaches. These findings highlight the effectiveness and efficiency of the proposed framework in performing loop closure detection tasks in real-time robotics applications.
Chapter 6

Learning Embeddings Using Graph Neural Network

6.1 Introduction

The graph structure is a powerful non-linear data structure that can effectively represent various real-world problems, including bio-chemical networks, social media networks, traffic signal networks, and maps, among others. By leveraging state-of-the-art GNN techniques, we can learn the relationships between nodes and their neighbors to make predictions about node attributes (node classification), node linkages (edge prediction), or even overall graph properties (graph classification). In the context of loop closing, the goal is to predict the similarity between a query node and a matching node. While many existing approaches focus solely on the features extracted from the query and matching nodes, [103] introduced a conventional graph diffusion mechanism to learn features from the surrounding neighbors. In Chapter 5, we have constructed a real-time graph from the dataset and computed the cosine similarity directly from the features extracted using a convolutional backbone. The constructed
graph imitates the structure of the graph SLAM, which can be easily transformed into full SLAM with additional components like odometry and pose graph optimization. However in chapter 5, we extracted the features solely from query or matching images, without leveraging its neighborhood image information. GNN makes it possible to learn a new features by gathering neighborhood information that helps to distinguish similarity and dissimilarity clearly. This chapter presents a novel approach to learn a new embeddings that can enhance the similarity score for the loop closure queries and matching nodes whereas diminishing the score for dissimilar nodes, finally resulting in reduced false positives and false negatives. This is achieved by utilizing a graphSAGE [6] layer, which is a variant of graph convolutional neural networks explained in Section 2.1.3. The contributions of this work can be summarized as follows:

- Formulating the loop closing problem as a link prediction problem, allowing us to learn similarity and dissimilarity embeddings from prior data.

- Investigating the applicability of GNN in the domain of loop closing detection, potentially pioneering this approach in the literature.

To implement this strategy, the problem of loop closing detection is framed as a link prediction problem between two nodes within a graph. This formulation is detailed in Section 6.2, which also covers the preparation of the dataset for training and evaluation purposes. Subsequently, in Section 6.3, the model architecture, hyperparameter tuning, and training strategy are elaborated upon. This includes providing insights into the design choices of the model and optimizing its performance through parameter adjustments. Finally, the experimental results are presented and discussed in Section 6.4, highlighting the outcomes and insights gained from the evaluation. The chapter concludes by summarizing the findings and implications of the study.
6.2 Problem Formation

Link prediction is a supervised binary classification problem that involves predicting whether there should be a connection between two nodes within a graph [116]. In the context of loop closing detection, the objective is to predict the similarity between two images and subsequently add an edge to the graph based on this similarity [117]. In order to train a GNN in a supervised manner, the edges in the graph are divided into separate train, validation, and test datasets. The validation and test datasets exclusively contain “supervision” edges, which are utilized for evaluation purposes. On the other hand, the train dataset consists of both “supervision” edges and “message-passing” edges. The “message-passing” edges are utilized during training to facilitate the learning of neighborhood information.

The dataset preparation involved using five sequences of image-to-image data along with their corresponding ground truth to construct a graph. Sequential datasets were not used to avoid potential issues of data leakage in the training and testing pipeline, as they often contain similar pairs of images in sequence for both the matching and query nodes. By combining these five datasets a graph with a total of 1272 nodes and 2567 positive edges was created. To simplify the process, the graph was constructed using features extracted from ShuffleNet, rather than the original images. It is important to note that for this problem, there are a total of 1,616,712 possible edges between the nodes (1272 × 1271), out of which only 2567 are positive edges. It was observed that, when adding a new untrained layer of graphSAGE on top of the ShuffleNet features, untrained models tend to predict most edges as positive due to the random weights in newly added layers. This characteristic makes the problem highly unbalanced in terms of classification. To address this issue, negative edges are dynamically added during training based on the ratio of total possible edges to positive edges. The
dataset was split into train, validation, and test sets in a proportion of 80%, 10%, and 10%, respectively. Considering the imbalanced nature of the problem, evaluation metrics such as specificity, precision, and F1-score were used. Specificity helps to assess the model’s ability to identify dissimilarities between images, precision measures the model’s ability to correctly predict true loop closures with minimal false positives, and the overall evaluation is determined based on the F1-score, which takes into account the data imbalance.

6.3 Methodology and Experiments

The primary objective of utilizing a GNN to learn new embeddings on top of convolutional backbone is to effectively distinguish between similar and dissimilar nodes. This distinction is crucial as it allows the embeddings’ cosine similarity scores to accurately reflect the level of similarity between nodes, leading to improved evaluation metrics. The basic architecture, as depicted in Fig. 6.1, involves adding two graphSAGE [6] layers with intermediate ReLU activation and dropout layers on top of the ShuffleNet features. GraphSAGE layers accumulates the features from the neighbor nodes from the training set which exhibits the higher similarity and learns a new embeddings during training phase to differentiate similar and dissimilar neighbors. The output of the final graph convolutional layer represents the learned embeddings, which are subsequently used to predict loop closures by calculating the cosine similarity between pairs of vertices. The number of nodes in each graph convolutional layer and the dropout coefficient are determined through a hyperparameter tuning strategy [118], aiming to optimize the model’s performance.

The selection of hyperparameters is performed using the wandb sweep agent, which
iteratively runs the training loop with different combinations of input parameter settings. The goal is to optimize a specific metric, in this case, the cosine embedding loss [33] on the validation set. The optimization process spans 40 epochs. Various hyperparameters, including the number of nodes in each graph convolutional layer and the dropout coefficient for the model architecture, are tuned. Additionally, hyperparameters related to the training strategy, such as the initial learning rate, L2 normalization (decay factor in the Adam optimizer) [119], and the patience number to reduce the learning rate on a plateau, are also optimized. After conducting multiple runs, the values presented in Table 6.1 are finalized as the hyperparameter settings for the final experiments.

Figure 6.1: Model architecture to learn the embeddings using graphSAGE layer on top of ShuffleNet.

Table 6.1: Final hyperparameter values selected after 128 iteration monitored to minimize validation loss.

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCN-1</td>
<td>1000</td>
</tr>
<tr>
<td>GCN-2</td>
<td>2000</td>
</tr>
<tr>
<td>Learning Rate</td>
<td>1e-3</td>
</tr>
<tr>
<td>Dropout</td>
<td>0.5</td>
</tr>
<tr>
<td>Decay</td>
<td>1e-3</td>
</tr>
</tbody>
</table>

Once the final model architecture parameters and initial training parameters are established, the model is trained in two iterations. The first iteration consists of
training for 30 epochs, while the second iteration extends to 50 epochs. In the second iteration, the learning rate is reset to its initial value, but the model continues training with the learned weights from the previous iteration. This two-part training approach is employed to address the issue of loss convergence observed after 20-25 epochs, which occurs when the learning rate is reduced to lower values. As depicted in Fig. 6.2(a) and Fig. 6.2(b), the loss does not exhibit significant improvement after 1.5k steps in the first iteration, but there is still noticeable improvement in the second iteration. Although the improvement may appear modest, it contributes to enhancing the evaluation metrics.
6.4 Experimental Results

As indicated in Table 6.2, the first iteration achieves a validation loss of 0.0848. Although the loss could be further reduced by extending the number of epochs, the downward slope of the loss curve, refer Fig. 6.2(b) has not yet reached saturation. Consequently, a second training iteration is conducted with the same initial parameters but utilizing the learned weights from the first iteration and an increased number of epochs (50). In the second iteration, the lower slope of the training and validation loss may appear insignificant; however, it contributes to enhancing positive edge prediction by reducing false positives. Since the number of positive edges is considerably smaller than the number of negative edges, the average loss of both types of edges is not distinctly visible. The final evaluation metrics support the hypothesis, with the precision of the positive class improving from 88.8% to 95.81%. The final prediction results on the test set demonstrate a specificity of 96.48%, precision of 95.81%, and F1-score of 87.47%. These results indicate that the proposed model effectively reduces false positives while improving true positives and true negatives.

Table 6.2: Training parameter and evaluation metrics comparison of two training iteration.

<table>
<thead>
<tr>
<th>Metric</th>
<th>1st Iteration</th>
<th>2nd Iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Epoch</td>
<td>30</td>
<td>50</td>
</tr>
<tr>
<td>Start_lr</td>
<td>1e-3</td>
<td>1e-3</td>
</tr>
<tr>
<td>End_lr</td>
<td>4.78e-4</td>
<td>2.22e-6</td>
</tr>
<tr>
<td>Validation Loss</td>
<td>0.0848</td>
<td>0.0836</td>
</tr>
<tr>
<td>Validation F1</td>
<td>0.9101</td>
<td>0.9173</td>
</tr>
<tr>
<td>Precision</td>
<td>0.8880</td>
<td>0.9581</td>
</tr>
<tr>
<td>Recall</td>
<td>0.8671</td>
<td>0.8046</td>
</tr>
<tr>
<td>Specificity</td>
<td>0.8906</td>
<td>0.9648</td>
</tr>
<tr>
<td>F1-Score</td>
<td>0.8774</td>
<td>0.8747</td>
</tr>
</tbody>
</table>

Fig. 6.3 presents a comparison between the results obtained from the direct extracted features from the backbone models discussed in the previous chapter and the
results achieved using the proposed GNN method on the combined dataset. Notably, Fig. 6.3 reveals that the proposed GNN-based embedding learning outperform the directly extracted features from the ConvNet and transformer, achieving an average precision of 93.94% which is at least 20% higher than the average precision of 73.85% achieved by ShuffleNet extracted features.

It is important to note that the proposed models require the availability of some neighboring information initially in order to generate suitable embeddings. As a result, in order to use this model in real-time applications, binary edges needs to be added into graph by applying a threshold on the predicted similarity score. The threshold is obtained from training of the model offline. In addition, they can be employed offline method to enhance the results obtained using online methods, particularly by reducing the occurrence of falsely identified loop closures, which can have detrimental effects on the overall SLAM system.
6.5 Conclusion

In conclusion, this chapter has focused on the implementation and evaluation of a GNN approach for loop closing detection in SLAM. The challenge of predicting loop closures in visual SLAM systems has been successfully addressed by leveraging the power of graph structures and GNN techniques. Through the use of learned embeddings and cosine similarity, our proposed method demonstrates improved performance in separating similar and dissimilar nodes, leading to enhanced evaluation metrics, specifically precision and specificity.
In conclusion, the problem of loop closure detection in SLAM has been thoroughly investigated in this thesis, and a novel approach using real-time graph construction and performance optimization through learning embeddings from GNN has been proposed. A comprehensive review of conventional and deep learning-based SLAM algorithms was conducted, highlighting their limitations and potential areas for improvement. In Chapter 5, a modularized and parameter-free framework was introduced to enable efficient real-time loop closure. By extracting features using deep learning-based backbones from input images and computing matching scores, the framework constructs a graph structure that captures the connectivity of nodes based on the matching criteria. This graph structure facilitates rapid retrieval of matching nodes. Through extensive evaluation using various deep learning-based feature extractors on a set of sequences from an open-source dataset, the proposed framework demonstrates faster processing times and higher precision-recall scores compared to alternative approaches. These findings underscore the effectiveness and efficiency of the proposed framework for real-time loop closure detection tasks. In Chapter 6, the loop closure approach was modified to link prediction in the graph data structure, exploring the potential
applicability of GNN in loop closure detection. A novel approach was proposed to learn effective embeddings by aggregating information from the neighborhood for robust loop closure. Training and testing were conducted on a combined dataset comprising five datasets. Promising results have been observed with the proposed graph-based loop closure detection system, as it has demonstrated improvements in both precision and specificity compared to approaches that utilize features directly extracted from convolutional backbone models. This indicates the effectiveness of the graph-based approach in capturing the intricate relationships between nodes and leveraging them for more accurate loop closure detection. By incorporating the graph structure and utilizing GNN, the proposed system enhances the ability to discriminate between loop closures and non-loop closures, leading to more reliable and precise predictions. These findings highlight the potential of graph-based methods in advancing loop closure detection systems and improving their overall performance.

7.1 Future Works

- **GNN on sequential dataset:** As mentioned in chapter 6, the use of sequential datasets for training and testing purposes was avoided due to the data leakage problem. However, further studies are required to determine a suitable approach for splitting the training and testing sequences, ensuring that GNN can be used without information leakage into the blind dataset.

- **Feature Engineering:** In this thesis, our approach involved exclusively extracting features from the final flattened layers of the backbone models. However, it’s important to note that there exist multiple methods for feature extraction in deep learning-based models. These alternatives encompass extracting features
from intermediate layers, fusing features from various layers, and even combining features from different models. Additionally, the choice of pre-trained models can be adapted to better suit the dataset’s relevance in the experiment. For instance, while this thesis utilized ImageNet-trained models, an alternative approach could involve training on datasets like Kitti360 or Places365, which are more aligned with outdoor and dynamic environments.

- **Experiment with matching criteria:** In this thesis, our experimentation primarily focused on utilizing cosine similarity as the matching criterion for LCD. Cosine similarity is a widely adopted metric known for its effectiveness in previous LCD research. However, it’s important to acknowledge that there are alternative matching criteria that warrant further investigation. These include metrics such as Mahalanobis distance, Euclidean distance, and Siamese networks. Each of these approaches may offer unique advantages and could potentially enhance the performance of loop closure detection under specific conditions or datasets.

- **Sensor Fusion:** Sensor fusion is an emerging technology that leverages the complementary functionalities of various types of sensors. In outdoor applications, the combination of cameras and GPS can be harnessed to enhance loop closure detection. The utilization of a combination of cameras and LIDAR sensors can be employed to acquire depth-wise features, ultimately leading to improved localization accuracy.

- **Memory Optimization:** One area of focus for future research is memory optimization. As the number of images increases, the size of the graph also grows, leading to increased memory consumption. While deep learning-based
solutions using GPUs may provide runtime optimization, addressing the challenge of memory consumption is crucial for long-term scenarios. Further research is needed to explore memory-efficient approaches for handling large-scale graphs in real-time applications.

- **Dataset and Groundtruth:** Another aspect that requires attention is the availability of standardized datasets and groundtruth for LCD. Currently, there are various sequential datasets available in the open-source community; however, the groundtruth varies between different works, making it challenging to compare and evaluate algorithms consistently. Additionally, variations in viewpoints and the preparation of groundtruth from GPS coordinates introduce additional complexities. Establishing a standardized set of datasets with reliable groundtruth is essential for ensuring valid comparisons and facilitating advancements in LCD techniques. Furthermore, for GNN, larger image-2-image datasets would be beneficial to enhance the training and performance of GNN models.

- **Full DL-based SLAM:** A significant future direction involves the integration of visual odometry and pose graph optimization to develop a fully functional deep learning-based SLAM system. Visual odometry plays a crucial role in estimating the robot’s motion by tracking features in consecutive frames, while pose graph optimization aims to refine the estimated poses based on the constraints provided by loop closures. Incorporating these components into the SLAM system will enable comprehensive scene understanding and accurate mapping. For non-linear optimization tasks such as pose graph optimization, unsupervised networks hold promise as ideal candidates, as they focus on minimizing cost functions through iterative refinement.
Addressing these areas of future work will contribute to advancing the field of loop closure detection and deep learning-based SLAM, leading to more efficient and accurate mapping and localization capabilities in robotic systems.
References


ceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 1–9.


[101] B. Cao, A. Araujo, and J. Sim, “Unifying deep local and global features for image search,” in Computer Vision–ECCV 2020: 16th European Conference,


