Imperial College London

MSc Individual Project

Imperial College London

Department of Computing

Unsupervised Deep Loop Detection With Binary Edge Images

Author:
Zheng En Nicholas Goh

Supervisor:
Prof. Paul Kelly
Riku Murai

Second Marker:
Dr. Ronald Clark

Submitted in partial fulfillment of the requirements for the MSc degree in Computing Science of Imperial College London

September 4, 2020

Abstract

Appearance-Based Loop Closure Detection has always been a key aspect within the SLAM methodology for robots to acquire accurate positional awareness through the matching of images captured by their cameras across different points in time. In this project, we developed a reliable and effective loop closure detection algorithm on the CPU which only utilises a very sparse set of handcrafted features, namely Binary Edge Maps which were designed by Murai et al.[32]. Our approach integrates both handcrafted features and a neural network in the form of a denoising variational autoencoder to do loop closure detection, where we add noise in the form of homography image deformations. This hybrid approach is novel as most of the other state of the art algorithms for loop closure detection uses either a neural network or handcrafted features, but never both. Binary Edge Maps as a feature has also not been used before within the context of loop closure detection. We demonstrated the algorithm to have a good performance via its ability to identify all unique loop closure sequences within the New College data set and majority of those within the City Center data set with no false positives. This was further benchmarked against a similar unsupervised neural network method which uses RGB images developed by Gao et al.[14], namely a stacked denoising autoencoder (SDA). We show that with the additional false positive checks, our model was able to attain a maximum recall of 36.7% and 30.1% at 100% precision on the New College data set and City Center data set respectively. which is higher than the one attained by the SDA. Also, the absence of such additional checks still allowed the algorithm to perform at a level close to that of the SDA in terms of precision, considering the fact that we are using inputs that contain much less information than full-scale RGB images. Lastly, we displayed the flexibility of such a system to generalise well to multiple real-life data sets by training exclusively on binary edge maps of synthetically produced images and showing that it still successfully and reliably detects loop closures on a variety of real-life data sets i.e. binary edge maps of RGB images captured in the New College and City Center data sets and binary edge outputs of an actual SCAMP-5 device.

Acknowledgements

This project wouldn't have been possible without the kind assistance provided by both Professor Paul Kelly and Riku Murai who were very generous with their time and very patient in their guidance despite the timezone and geographical differences that we were operating in. I would also like to thank Riku for kindly recording images of his own home using the SCAMP-5 device. Last but not least, I would like to thank my family and friends for their endless support and motivation throughout the entire duration of this project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

From operating the Mars Rover to autonomous vehicles, spatial awareness has always been a key interest in the field of robotics. The key for robots to attain such awareness can be found in solving the Simultaneous Localisation and Mapping (SLAM) problem - "which asks if it is possible for a mobile robot to be placed at an unknown location in an unknown environment and for the robot to incrementally build a consistent map of this environment while simultaneously determining its location within this map".[12]

Today, with the development of better visual sensors, SLAM solutions have moved into using visual odometry - defined by Nister et al. as "a system that estimates the motion of a stereo head or a single moving camera based on video input".[33] Usual approaches include the identification of features within frames, matching of such features over multiple frames and estimating motion through optical flow. Unfortunately, visual odometry methods accumulates drift over long periods of time, increasing the error of motion estimates as time progresses.[33] This highlights the importance of loop closure detection. By allowing a robot to correctly identify previously visited places from sensor data, one is able to correct inaccuracies in the map created by drift, ultimately generating a more consistent representation and reducing uncertainty. In this paper, we will focus only on detecting if a loop has occurred (i.e. loop closure detection) and will not cover issues surrounding pose-correction of the robot. Also, we will focus on the use of appearance-based methods, which are methods that use images as the main source of information.[15]

This research is an extension of the efficient and lightweight visual odometry system designed by Murai et al..[32] Focal-plane Sensor-processors(FPSP), are general purpose vision chip technology which allow for parallel computation. This creates a highly efficient system – in terms of both power consumption and frame-rate by reducing the amount of data transferred. Using the SCAMP-5, a general-purpose FPSP, Murai et al. were able to create an efficient BInary feaTure Visual Odometry system, BIT-VO by calculating binary edge based descriptors on the FPSP chip itself and transmitting this set of reduced features to the host device for further processing. In this research, we aim to perform loop closure detection on the CPU using only this set of reduced binary features as input. If successful, this algorithm can then be ported over to the SCAMP-5 device in future works to enhance the robustness of this visual odometry system while still retaining the low power con-

sumption and high frame rate.

To summarise, our main research problem is as follows: To design an effective loop detection algorithm by utilising only a reduced set of binary edge features. Deployment will be done on a CPU and testing will be carried across multiple data sets, including a small unlabelled data set produced by the SCAMP-5 device.

1.1 Contributions

Below highlights the main contributions that have been achieved through this paper:

1. We successfully developed a reliable loop closure detection system implemented on the CPU utilising only binary edge images as input. The algorithm was novel in that it utilises hand-crafted descriptors as inputs to a neural network. More specifically, we use a denoising variational auto-encoder to achieve this, where we added noise to the inputs in the form of homography deformations. Furthermore, our model is unsupervised meaning that no extra work is required to label the loop closure data sets during the training phase which can be a very time-consuming task.

2. The performance of this algorithm with the additional false positive detection mechanisms (i.e. Smith-Waterman and Rank Reduction) was validated through it being able to identify all unique loop closure sequences within the New College data set and majority of the unique sequences within the City Center data set, with none of these sequences being a false positive. The algorithm was also shown to obtain loop closure sequences from the output of the SCAMP-5 device with no false positives.

3. We then demonstrated the algorithms capability through attaining a higher maximum recall rate of 36.7% and 30.1% at full precision on the New College data set and City Center data set respectively. This provided us with better results against the state-of-the-art unsupervised stacked denoising autoencoder (SDA) framework designed by Gao et al.[14] which utilised RGB images. Performance without additional detection mechanisms was also shown to perform at a level close to that of the SDA in terms of precision, considering the fact that we are using inputs that contain much less information than full-scale RGB images.

4. Finally, we demonstrated the flexibility of such a system to generalise well to multiple real-life data sets by training exclusively on binary edge maps of synthetically produced images and showing that it still successfully and reliably detects loop closures on a variety of real-life data sets i.e. binary edge maps of RGB images captured in the New College and City Center data set and binary edge outputs of an actual SCAMP-5 device.

1.2 Report Structure

This thesis is structured in the following manner:

1. **Background**: In this chapter, we provide the reader with an understanding of the core concepts that we build our loop detection algorithm upon such as the SCAMP-5 device, handcrafted binary edge map features, neural networks, evaluation metrics and dimensonality reduction techniques (with a specific emphasis on auto-encoders).

2. **Related Works**: In this chapter, we provide an overview of the various methods currently employed to solve the loop closure detection problem, ranging from hand-crafted features to the recent usage of neural networks.

3. **Experiment Setup**: In this chapter, we give an overview of how we setup the data sets that we use for training and validation of our neural network. We also give an overview of the evaluation metrics that we use to assess the performance of our algorithm.

4. **Methodology**: In this chapter, we highlight our loop closure detection algorithm including the reasons as to why we made those specific adjustments. This is broken down into two major phases - firstly, the production of a compressed latent space from our binary edge map inputs and secondly, the matching of these latent spaces to identify a loop closure.

5. **Results and Evaluation**: In this chapter, we summarise the results of our testing on a variety of data sets as well as give an evaluation of the strengths and weaknesses of our algorithm. We also attempt to benchmark our algorithm against other state of the art methods.

6. **Conclusion and Future Works**: In this chapter, we round up the report by highlighting how we have achieved each of these contributions as well as provide a brief overview on possible future works that might be pursued to further this piece of research.

Chapter 2

Background

In this chapter, we will cover the broad concepts that are related to our work such as the FPSP device that we are drawing the inputs from and the handcrafted binary edge map features that we intend to use. We then provide an overview of neural networks and their evaluation metrics, specifically on autoencoders which forms the core of our detection algorithm.

2.1 FPSP

Focal-plane sensor-processor (FPSP) image devices are sensor arrays and processor arrays embedded in each other on the same silicon chip. By allowing for the processor and sensor to be closer, processing efficiency is greatly increased as we reduce the amount of transfer of sensory data which is a slow and energy-intensive process.[50]

2.1.1 SCAMP-5

The SCAMP-5 is the specific FPSP device used by Murai et al.[32] in their research. This device was developed by the University of Manchester. The SCAMP-5 vision chip contains a 256x256 processor array which operates in a Single Instruction Multiple Data (SIMD) mode - "able to perform the same operation on multiple data operands concurrently"[6]. Each pixel houses a processing element(PE) which contains a few registers, local memory and an arithmetic logic unit that can execute simple operations. Operations can be done in an analog space, which eliminates the need for A/D conversion, thus saving power.[7] Figure 2.1 highlights the architecture of the SCAMP-5 vision chip.

There is no global memory for the SCAMP-5 vision chip and this coupled with its limited hardware components limits the computational complexity of the device. This is also the reason why the output that it produces (i.e. the inputs we work with to perform the loop closure detection in this research) is limited in terms of the information it contains.[7]

2.2 Feature Detection

Features are condensed representations of an image which contain rich information about the image. By comparing the features between two images, we can create a

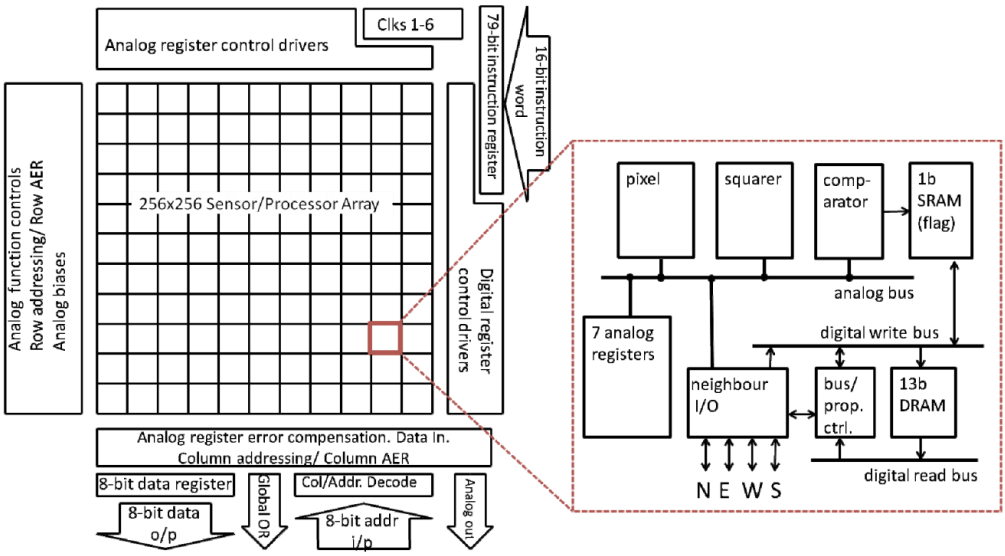

Figure 2.1: Architecture of the SCAMP-5 vision chip. Obtained from [7].

set of correspondence between images, allowing us to effectively match an image. One of the most popular class of features are locally hand-crafted features. Using such features for detection can be executed via two general steps[46]:

1. Features such as interest points / key-points are identified. Corners is an example of such key-points.

2. A local descriptor for each of this point is tabulated.

Local features are more robust against occlusion and clutter but will result in multiple descriptors per image.[49] Examples include SIFT[28] and ORB[40] which covers both steps highlighted above. There are also other algorithms like Harris Corner Detector[16] which only captures the interest point detector step.

Apart from local hand-crafted features, another class of popular feature detection methods are global handcrafted features. Global features use a single local descriptor to describe the appearance of a complete scene rather than having one descriptor per interest point. However, they are known to be less resilient against occlusion and clutter within an image.[49] Methods include Histogram Oriented Gradients (HOG)[11] and GIST[36]. With the advent of deep learning, researchers have also been able to obtain abstract feature descriptors via the training of a neural network. A more in-depth view of the different features considered in various loop closure algorithms will be found in Section 3.

2.2.1 Binary Edge Features

Here we give an overview of the binary edge features designed by Murai et al.[32] that we will be using to perform loop detection. There are 2 set of features that are currently being identified by the BIT-VO system: Coordinates of the corner key-points that are generated by the FAST detection algorithm and a binary edge map containing the location of all edges. For our approach, we will only be using

the binary edge map.

The binary edge map is a global feature where the value of a pixel is 1 if an edge is present at that pixel and 0, otherwise. Edges are defined as points where the intensity of the pixel varies greatly. These points can be identified through functions that can approximate gradient magnitude. A more in depth process on how we extract edge maps from a RGB-image can be found in Section 4.1.1.

The main benefit of using an edge map for image similarity matching, apart from the increased in efficiency of data transmission is that edges are robust towards slight changes in illumination and are less noisy. However, there are still remains other key challenges that causes discrepancies in image matching despite the image being of the same scene:

1. **Scale**: The changes in the scale of an image from a camera capturing a scene from different distances.

2. **Viewpoint**: The changes in the viewpoints from a camera capturing a scene at different angles.

3. **Occlusion and Clutter**: Occlusion occurs when certain key objects get blocked out from the camera's vision. This can be caused by changes in viewpoints and scale or the inclusion of objects when capturing the same scene at different time frames. Occlusion occurs more commonly when a scene is very cluttered.

Also, there are additional issues of using binary edge images:

1. **Reduced information**: There is a substantial decrease in information contained within such feature descriptors. Firstly, there is a loss of colour information. However, this is less worrying as most image matching algorithms today work even with gray-scale images, albeit a slight decrease in performance. In fact, the main difficulty we face while using such features is in the loss of intensity values. While we are able to still retain some information about where there are large intensity changes (i.e. gradient magnitudes), these are binarised into 2 buckets based on the threshold and therefore, there is reduced granularity of intensity change information. Also, the SCAMP-5 device is unable to produce orientation information for such gradients.

2. **Sparse and extreme image pixel values**: Image values are now binary and since the occurrence of edges are less common within an image, the image will mostly consist of 0 values, with the 1 value being spaced far apart from each other, creating a sparse image.

2.2.2 Appearance-Based Loop Detection

Appearance-Based loop detection is the use of images to identify loops. Loop detection algorithms are usually an extension of the Feature Detection algorithm used in Section 2.2 with the added task of doing image matching, where we attempt to find similar images using the feature descriptor. Section 3 will discuss the range of algorithms that researchers have used to solve the loop detection problem in more detail.

The main challenges of Appearance-Based Loop Detection are the following[15][25]:

1. **Perceptual Aliasing**: This is a problem where two scenes look the same but are different places in reality. Perceptual Aliasing in loop detection is a huge issue in visual odometry as an incorrect match can potentially corrupt the localisation map of the robot quite significantly.

2. **Scalability**: As the robot continues running over long periods of time, the number of existing frames that we have to compare to the new incoming frame increases. This increases the complexity of the algorithm.

3. **Sensor Noise**: Images output by camera sensors are corrupted by noise, which in turn makes it difficult to match images.

4. **Environmental Changes**: Due to various environmental changes (i.e. rain, day-night, etc.), same scenes could potentially look different, once again affecting the matching process.

2.3 Neural Networks

A neural network aims to emulate the method in which neurons work within a human's brain. This is done via the connection of artificial neurons through a network.

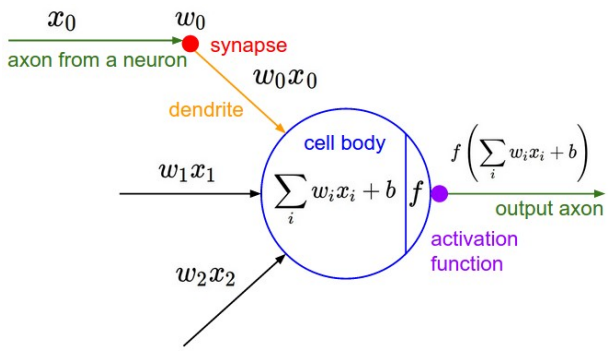

Figure 2.2: Structure of a neuron. Image obtained from [26].

An artificial neuron as shown in Figure 2.1 takes in a certain set of N inputs x_i and constructs a linear relationship between these inputs and the output y for a single sample, as shown in Equation 2.1. Here w_i are the assigned weights given to each input moving into a neuron and b is the overall bias of this linear relationship. In order to extend the model to non-linear relationships, an activation function $f(.)$ is then applied to the output of such neuron.

$$\begin{aligned} y &= \sum_{i=1}^{N} w_i x_i + b \\ y &\rightarrow f(y) \end{aligned} \tag{2.1}$$

Common activation functions include the sigmoid function, softmax function and ReLU function. These functions are all non-linear in nature and an example of the sigmoid function is shown in Equation 2.2.

$$sigmoid(x) := \frac{1}{1 + e^{-x}} \tag{2.2}$$

Using these artificial neurons as building blocks, a neural network can then be constructed by linking the output of a layer of neurons as the inputs of the next layer of neurons, essentially forming a network as show in Figure 2.3. This is usually done in a "fully-connected" fashion where each neuron in the previous layers is linked to all other neurons in the next layer.

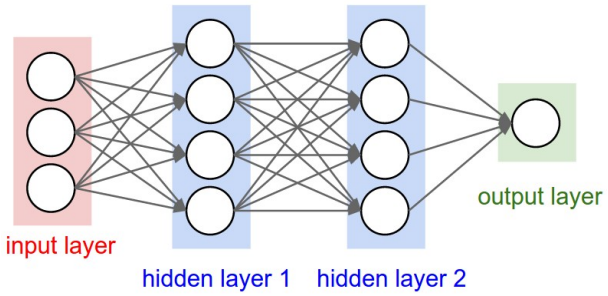

Figure 2.3: Structure of a neural network. Image obtained from [26].

The next step would be to then define a loss function between the predicted output produced by this network and its corresponding true value for the entire sample. This is usually chosen to be the Mean Squared Error for the case of real values or the Cross Entropy Loss for the case of discrete values. An example of the Mean Squared Error for a sample size of n is shown in Equation 2.3.

$$MSE(\mathbf{y}^{true}, \mathbf{y}^{pred}) := \frac{1}{n}\sum_{i=1}^{n}(y_i^{pred} - y_i^{true})^2 \tag{2.3}$$

In order to minimise the following loss function, we need to find the partial derivative of the loss function with respect to the individual weights and biases. This can be done through back-propagation. Finally, after this derivative is obtained, the loss function can then be minimised via stochastic gradient descent as shown in Equation 2.4 where we collect the individual weights and bias into matrices and vectors $\mathbf{W}$ and $\mathbf{b}$.

$$(\mathbf{W}*, \mathbf{b}^*) = (\mathbf{W}, \mathbf{b}) - \gamma(\frac{\delta Loss}{\delta \mathbf{W}}, \frac{\delta Loss}{\delta \mathbf{b}}) \tag{2.4}$$

We note here on the γ value which is also known as the learning rate. This will be explained more in Section 2.3.1 about hyperparameters.

2.3.1 Neural Network Hyperparameters

Hyperparameters are parameters that are external to the training of the model, which means that they need to be initialised before training of the model begins. Hyperparameters can greatly influence the result of a neural network. We explain some key hyperparameters for neural networks here, including its effects:

1. **Batch Size**: The batch size is the number of samples you are passing through the neural network before it updates its internal parameters via mini-batch gradient descent. By splitting the entire network into batches, and updating the weights and biases of a neural network using data from a single batch, one is able to speed up computations during the back-propagation phase. Having too small of a batch size will result in the gradient estimation becoming very noisy reducing the ability for a model to reach a proper local minima. Conversely, having too large of a batch size will lower the speed of computation.

2. **Number of Epochs**: The number of epochs represent the number of times you will train your entire data set through the neural network. Having a large number of epochs might help lower the loss for the model, but this can potentially result in the model to over-fit. Choosing the right number of epochs will allow the model to make good predictions while at the same time allowing it to be generalisable to data points outside of the training set.

3. **Learning Rate**: The learning rate determines the step size when doing gradient descent. This is represented by γ as seen in Equation 2.4. Too small a learning rate will result in small changes to the weights and biases of the model which in turn lengthens the time required to find a minima. On the flip side, a high learning rate might cause the model to diverge as one might potentially step past a minima during gradient descent.

4. **Weight Decay**: Weight decay represents the weight applied to the L2 regularisation term within the loss function and is represented by λ in Equation 2.5. Weight decay acts as a regularisation term to prevent over-fitting, but choosing too high of a value might conversely result in under-fitting the data.

$$Loss = MSE(\mathbf{y}^{true}, \mathbf{y}^{pred}) + \lambda \sum_{i=1}^{N} w_i^2 \tag{2.5}$$

2.3.2 Convolutional Neural Network

Convolutional Neural Networks (CNNs) are a key tool in many of today's Computer Vision tasks as they work better than plain fully-connected neural networks on images. While plain neural networks connect each pixel of every channel in the input to every pixel of the output image, CNNs reduce the amount of connections through the mechanism of convolution which is depicted in Figure 2.4. Each convolution layer of the network consists of width, height and depth parameters. The width and height together forms the receptive field or the filter size of that convolution layer. As the filter slides over the width and height of the input volume for every depth slice, each filter will tabulate a set of weights for each pixel within itself. This means that each pixel / neuron in the output volume is connected to only a local region which is the size of the receptive field but to every depth layer of the input. For example, a 4 x 4 (H x W) filter applied to a 64 x 64 x 3 input will have $4 \times 4 \times 1 = 16$ weight parameters per neuron in the output volume. For output volumes with non-singular depth, it is assumed that the neurons in each depth will share the same weights.

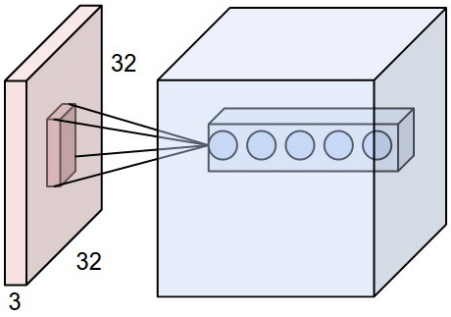

Figure 2.4: Convolution Mechanism. Image obtained from [26].

There are a few hyperparameters of the filters which control the output size of the output volume[26], which will be elaborated below:

1. **Depth**: The depth of the filter determines the depth of the output produced after the convolutional filter is applied to it.

2. **Stride**: The stride determines the step size of the filter with each slide. A stride of 2 means that the filter moves by 2 pixel every turn. An example is shown in Figure 2.5.

3. **Padding**: Padding refers to the number of layer of surrounding zero-valued pixels we add to the input before applying convolution. An example is shown in Figure 2.6.

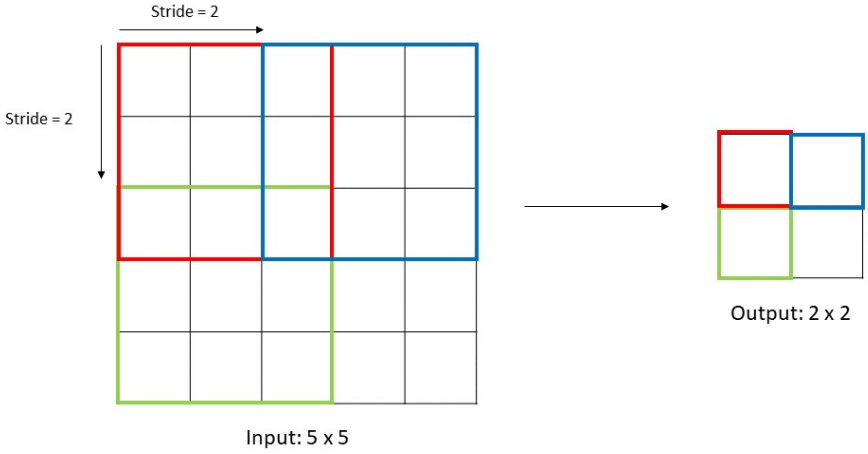

Figure 2.5: Input and output of single depth layer after convolution with 3 x 3 filter, stride = 2 and padding = 0.

Another key layer within CNNs that often appear are pooling layers. These layers allows the network to down-sample the image to reduce the number of parameters and to speed up training and are usually used when the input image is large. Pooling is applied to every depth layer of the input, therefore the output it produces will be of similar depth. Pooling downsizes the image by inputting multiple pixel values within its filter and outputting only a single pixel value. The most popular form of pooling is max-pooling which outputs the maximum value out of all values within

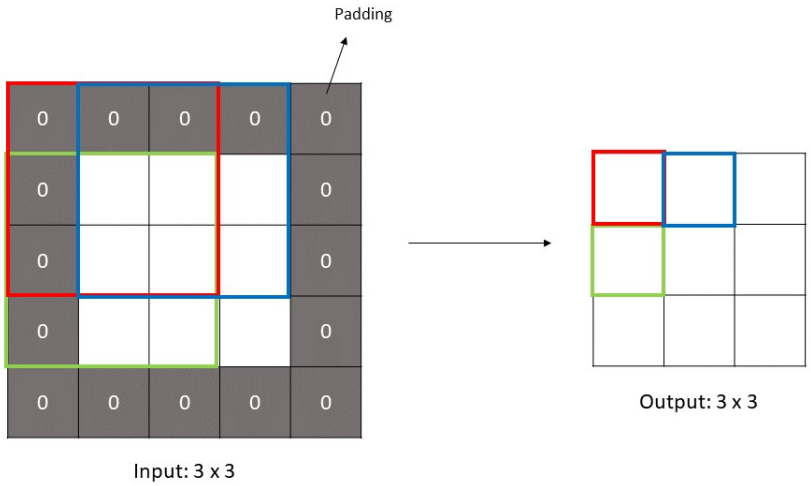

Figure 2.6: Input and output of single depth layer after convolution with 3 x 3 filter, stride = 1 and padding = 1.

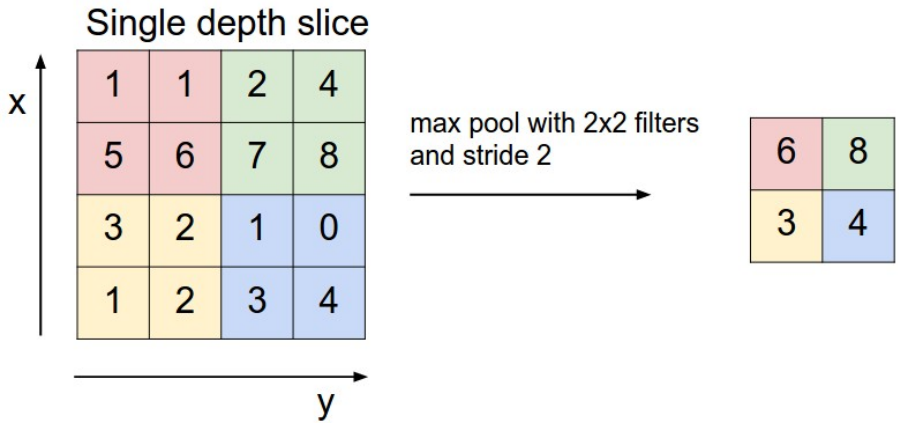

Figure 2.7: 2 x 2 Max Pooling. Image obtained from [26].

its filter. An example of a max-pooling is shown in Figure 2.7.

Apart from convolutional and pooling layers, CNNs also contains fully connected layers. These are layers where every pixel in the previous layer is connected to every pixel in the next layer. These are usually placed at the end of the network once the size of image inputs are relatively small after several convolution and pooling layers.

2.3.3 Evaluation of neural networks

A confusion matrix is a table that summarises the performance of a machine learning classification task. Figure 2.8 shows an example of a confusion matrix.
The semantics of each item within the table in the context of loop detection are as follows:

1. **True Positive (TP)**: Number of image pairs classified to be loop candidates where there is an actual correspondence between the pair.

		Prediction	
		Class 1	Class 2
Ground Truth	Class 1	True Positive (TP)	False Negative (FN)
	Class 2	False Positive (FP)	True Negative (TN)

Figure 2.8: Sample confusion matrix.

2. **False Negative (FN)**: Number of image pairs classified as non-loop candidates where there is an actual correspondence between the pair.
3. **False Positive (FP)**: Number of image pairs classified to be loop candidates where there is no actual correspondence between the pair.
4. **True Negative (TN)**: Number of image pairs classified as non-loop candidates where there is no actual correspondence between the pair.

The information from this table then allows us to calculate the Accuracy, Precision and Recall values whose formulas are shown in Equations 2.6, 2.7, 2.8.

$$Accuracy = \frac{TP + TN}{TP + FN + FP + TN} \tag{2.6}$$

$$Precision = \frac{TP}{TP + FP} \tag{2.7}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.8}$$

Equations 2.6, 2.7, 2.8 allows us to infer the following:

1. Accuracy is the proportion of total number of examples which are correctly identified.
2. Precision is the proportion of total positive predictions which are correct. Precision is a direct result of perceptual aliasing as mentioned in Section 2.2.2. A high degree of perceptual aliasing leads to larger numbers of FPs which lead to low precision.
3. Recall is the proportion of total positive outcomes that we correctly predict. Recall is a direct result of sensor noise and environmental changes as mentioned in Section 2.2.2. A high degree of this will result in larger number of FNs which lead to low recall.

2.4 Dimensionality Reduction

Dimensionality Reduction in Computer Vision is important in capturing a compact set of features that saliently describes an image. Here we highlight one classic dimensionality reduction algorithm commonly used - Principal Component Analysis, before delving into dimensionality reduction in the age of neural networks - Autoencoders.

2.4.1 Principal Component Analysis

Principal Component Analysis (PCA) is a linear dimensionality reduction process where high dimension data is orthogonally projected onto a lower dimensional subspace that maximises the variance of the lower dimensional data. An example of PCA is shown in Figure 2.9.

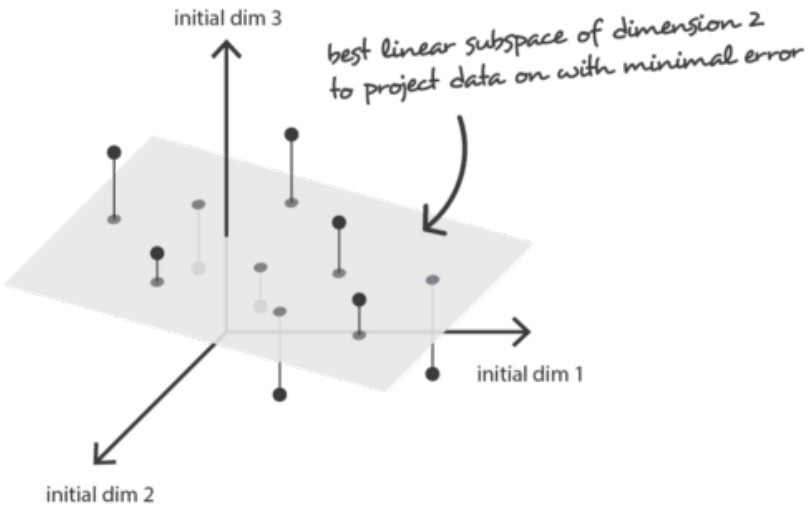

Figure 2.9: Example of PCA process. Image obtained from [39].

PCA is done in the following manner - Firstly, the data needs to standardised according to Equation 2.9 where n is the sample size, μ is the mean and σ is the standard deviation. This has to be done for each variable m.

$$x_i^{(m)} = \frac{x_i^{(m)} - \mu}{\sigma} \text{ where}$$
$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i^{(m)} \text{ and } \sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i^{(m)} - \mu)^2} \text{ for all } m \tag{2.9}$$

The next step would be to then compute the covariance matrix, Σ for all data points as in Equation 2.10. Note that $\mathbf{x}_i$ here is a vector containing all variables for the particular sample i, i.e. $\mathbf{x}_i = \begin{bmatrix} x_i^{(1)} & \dots & x_i^{(m)} \end{bmatrix}^T$

$$\Sigma = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T \text{ where}$$
$$\bar{\mathbf{x}} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i \tag{2.10}$$

We then tabulate the m eigenvalues and corresponding eigenvectors and of the covariance matrix as per Equation 2.11 where λ_i is one possible eigenvalue and $\mathbf{v}_i$ is

the corresponding eigenvector. Here, we sort the eigenvalues from the largest to the smallest such that $\lambda_1 > \lambda_2 > \ldots > \lambda_m$, with their corresponding eigenvectors $\mathbf{v}_1, \ldots \mathbf{v}_m$.

$$\Sigma \mathbf{v}_i = \lambda_i \mathbf{v}_i \tag{2.11}$$

In order to project our data into a smaller k dimensional subspace, we pick the k eigenvectors with the largest eigenvalues as the new orthogonal basis of the data. This forms the set of principal components. The revised condensed representation can then be computed using Equation 2.12.

$$\mathbf{x}_i = \begin{bmatrix} \mathbf{v}_1^T \mathbf{x}_i \\ \vdots \\ \mathbf{v}_k^T \mathbf{x}_i \end{bmatrix} \tag{2.12}$$

2.4.2 Autoencoders

Autoencoders are neural networks that learn how to create compressed representations of the original inputs. It is mainly used in an unsupervised learning context where one does not have prior knowledge about the truth labels. From a given input, we train the network to provide a similar output through the encoder $e(.)$, and the decoder $d(.)$. Training of an autoencoder proceeds as per that of a typical neural network - we set a loss function which represents the distance between x and x^{pred} (e.g. Mean Squared Error for real labels or Cross-Entropy for binary labels) and we minimise with respect to its weight and bias parameters via stochastic gradient descent. Figure 2.10 highlights the overall structure of an autoencoder, with Equation 2.13 highlighting its process.

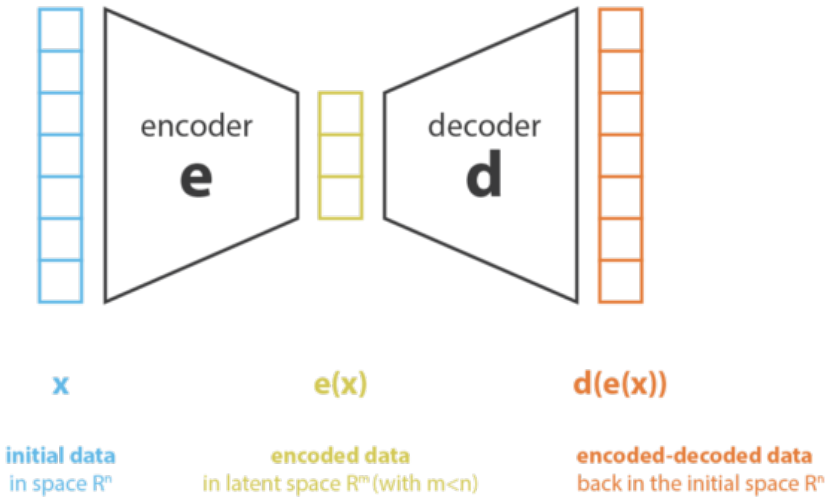

Figure 2.10: Structure of an auto-encoder. Image obtained from [39].

$$(d^*(.),\ e^*(.)) = \underset{d(.),\, e(.)}{\arg\min}\ Loss(x, d(e(x)) \tag{2.13}$$

Once training is completed, in order to get a prediction of the latent space on the test set, we simply discard the decoder and use the trained encoder as seen in Equation 2.14.

$$x^{latent} = e^*(x^{test}) \tag{2.14}$$

The key characteristic as to how an autoencoder obtains a compressed latent representation is via a "reduced" hidden layer. This is most commonly done by restricting the number of nodes within the hidden layer to be less than that of the input and output layers (i.e. an under-complete auto-encoder), which in turn limits the dimensionality of the hidden layer.

Autoencoders differ from PCA mainly in the way that it can model more complicated non-linear low dimensional representations of actual data, whereas those of PCA are confined to linear relationships. Also, the latent variables within autoencoders can be correlated, unlike PCA which enforces orthogonality of each of these variables.

2.4.3 Convolutional autoencoders

The encoder of an autoencoder follows a structure similar to a normal CNN. However, for the non-fully connected portions of the decoder, there needs to be a method of upsampling the image. One method is through the usage of transposed convolution filters. These work similarly to normal convolution filters with regards to depth connections, however there is a subtle difference in how the local connections work, especially with regards to stride and padding. Figure 2.11 shows the difference between such transposed convolution and regular convolution.

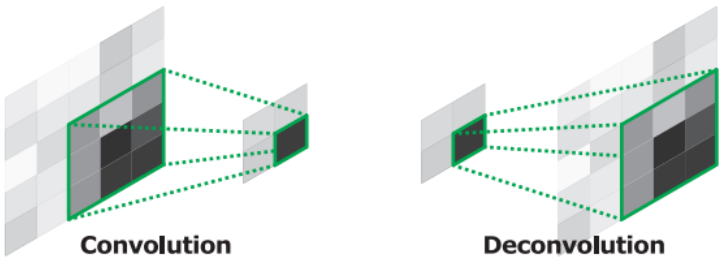

Figure 2.11: Convolution vs transposed convolution. Image obtained from [35].

For transposed convolution, layers of zero-valued pixels will be added to the input image until the bottom right pixel within the filter will be aligned to the top left pixel of the input. This can be seen in Figure 2.12 where A, the top left pixel of the input is in the bottom right of the 2 x 2 filter after adding 1 layer of zero pixels. The next step would be to then normally convolve this intermediate output with the 2 x 2 filter without stride to produce the final output. A few of the local connections are highlighted in Figure 2.12.

With regards to strides, the transposed convolution will place additional zero-valued pixels in between the input pixels when forming the intermediate image. The number of such pixels added is equivalent to the value of stride - 1. This can be seen in

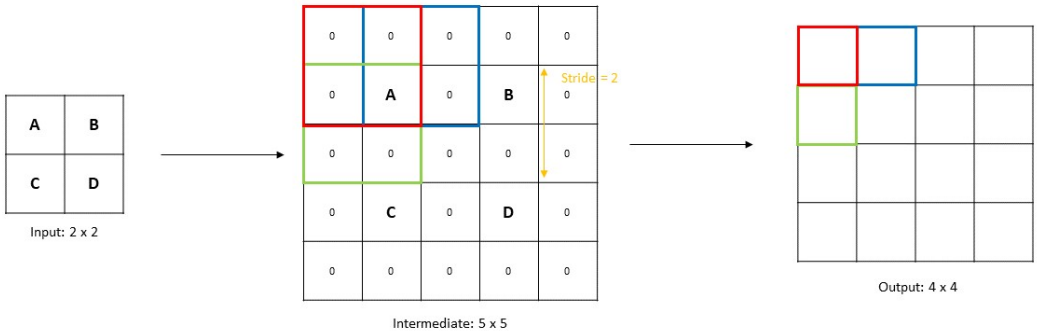

Figure 2.12: Input and output of a single depth layer after transposed convolution with 2 x 2 filter, stride = 2, padding = 0.

Figure 2.12 where $2 - 1 = 1$ zero-valued pixel is placed between each input pixel.

For padding, transposed convolution has the reverse effect, where padding is used to reduce the size of the output image. The number of padding will reduce the number of outer layers of the intermediate padding before the non-strided convolution to attain the final output. An example is seen in Figure 2.13.

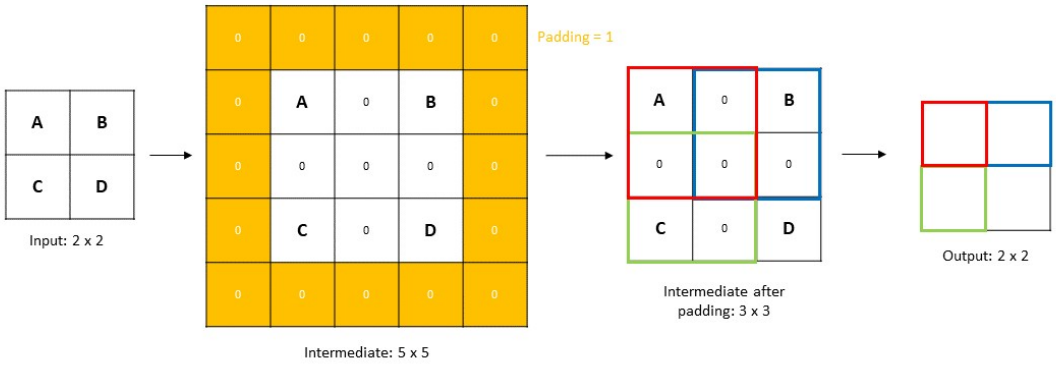

Figure 2.13: Input and output of a single depth layer after transposed convolution with 2 x 2 filter, stride = 2, padding = 1.

Transposed convolution also has an additional hyperparameter called output padding. This involves adding layers equal to the number of padding on the output of the transposed convolution.

Apart from transposed convolution, there is another form of upsampling, namely switch unpooling. This is the upsampling counterpart of the max-pooling described in Section 2.3.2. Switch unpooling involves saving the original coordinates during the encoding process where the max pooling occurs. During the decoding process,

these coordinates is fed into the system as a switch, where they are set to the unpooled value, while the others are set to zero. An example of switch unpooling is shown in Figure 2.14.

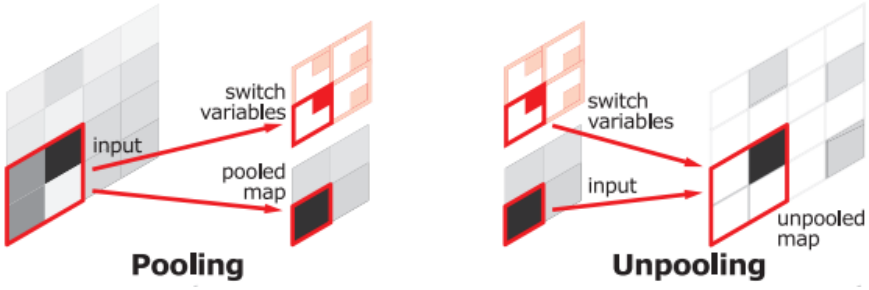

Figure 2.14: Example of pooling and switch unpooling. Image obtained from [35].

The alternative to switch unpooling is nearest neighbour unpooling which involves setting the non-switches to the same value as the switch as well. Nearest neighbour unpooling do not require the saved coordinate values, since all values within the unpooling filter is set to the same value.

2.4.4 Denoising autoencoder

In order to account for noise and to produce a generalisable model, one approach is to slightly corrupt the input data and maintain the uncorrupted data as the output. This is the structure of what we call a denoising autoencoder. One possible corrupting process is where we randomly select a fixed number of input components and set them to 0 while keeping the rest the same. The denoising auto-encoder will then be forced to recover these missing inputs.[47] The loss function then becomes a comparison between this corrupted input $widetildex$ and the uncorrupted output x as seen in Equation 2.15.

$$Loss(x, f(g(x))) \rightarrow Loss(x, f(g(\widetilde{x}))) \tag{2.15}$$

2.4.5 Variational autoencoder

Compared to a normal vanilla autoencoder whose latent representation is a deterministic value, variational autoencoders (VAE) try to regularise this latent space by modelling the latent space as a probabilistic distribution. By setting the latent space as a distribution, the latent space becomes more continuous as we force the encoded data to overlap more within the latent space, unlike the case of a vanilla autoencoder which might model data to be very far apart.

VAE was first implemented by Kingma et al. Here we highlight some of the key mathematical concepts within their paper with regards to the VAE.[24] First, we need to model our encoder and decoder in a probabilistic manner, where $q_{\phi}(\mathbf{z}|\mathbf{x})$ is our probabilistic encoder, encoding the data, $\mathbf{x}$ to its latent representation $\mathbf{z}$. Here, $\boldsymbol{\phi}$ are the parameters that will govern how this encoding function works. On the other hand, we let $q_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ represent our probabilistic decoder, where $\boldsymbol{\theta}$ are the

parameters of this function. We start off by expressing marginal likelihood as in Equation 2.16.

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}_i) = KL(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}_i) \,||\, p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}_i)) + ELBO(\boldsymbol{\theta}, \boldsymbol{\phi}) \tag{2.16}$$

Here the Kullbeck-Liebler divergence term (KL) measures the distance between probability distributions. Intuitively, the we would like to minimise the KL term so that the estimated latent space, $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}_i)$ matches the true latent space distribution $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}_i)$. Since the KL term is always positive, by Jensen inequality, minimising the KL term is equivalent to maximising the ELBO term. In that case, we reformulate the equation to that of Equation 2.17.

$$ELBO(\boldsymbol{\theta}, \boldsymbol{\phi}) = -KL(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}_i) \,||\, p_{\boldsymbol{\theta}}(\mathbf{z})) + \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}_i)}[\log(p_{\boldsymbol{\theta}}(\mathbf{x}_i|\mathbf{z})] \tag{2.17}$$

However, the difficulty of directly transforming this to an autoencoder lies in the fact that we need to be able to sample $\mathbf{z}$ form the encoder, while still allowing the error to be backpropogated to the network. For that, we need to adopt a reparameterisation trick, where we make the additional simplifying assumptions that both $p_{\boldsymbol{\theta}}(\mathbf{z})$ and $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}_i)$ are multivariate Gaussians. These assumptions and the assumed Gaussian distributions are shown in Equation 2.18.

$$\begin{gathered}\text{Assuming } p_{\boldsymbol{\theta}}(\mathbf{z}) \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \text{ and } q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}_i) \sim \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\sigma}_i^2\mathbf{I}), \\ \mathbf{z}_i^{(l)} = \boldsymbol{\mu}_i + \boldsymbol{\sigma}_i \odot \boldsymbol{\epsilon}^{(l)} \text{ and } \boldsymbol{\epsilon}^{(l)} \sim \mathcal{N}(0, \mathbf{I})\end{gathered} \tag{2.18}$$

Using the assumptions and the reparameterisation trick above, we can analytically evaluate the KL-divergence and putting everything together, we get the final form in Equation 2.19, where j is the dimensionality of $\mathbf{z}$. Note that we are maximising the ELBO term and therefore the loss function will be the negative of the ELBO.

$$\begin{aligned}&ELBO(\boldsymbol{\theta}, \boldsymbol{\phi}) \simeq \frac{1}{2}\sum_{j=1}^{J}(1 + \log((\sigma_i^{(j)})^2) - (\mu_i^{(j)})^2 - (\sigma_i^{(j)})^2) + \frac{1}{L}\sum_{l=1}^{L}\log(p_{\boldsymbol{\theta}}(\mathbf{x}_i|\mathbf{z}_i^{(l)})) \\ &\text{where } \mathbf{z}_i^{(l)} = \boldsymbol{\mu}_i + \boldsymbol{\sigma}_i \odot \boldsymbol{\epsilon}^{(l)} \text{ and } \boldsymbol{\epsilon}^{(l)} \sim \mathcal{N}(0, \mathbf{I})\end{aligned} \tag{2.19}$$

Like the vanilla autoencoder, the structure of the VAE has to contain both an encoder and a decoder. However, the encoder should be producing two outputs, $\boldsymbol{\mu}_i$ and $\boldsymbol{\sigma}_i$. This is followed by a reparameterisation trick being applied on these outputs to get $\mathbf{z}$ and feeding the result back into the decoder to attain the final structure. Together with the output, $\boldsymbol{\mu}_i$, $\boldsymbol{\sigma}_i$ should be fed into the loss function to calculate the final loss which will be optimised using gradient descent. A pictorial representation is shown in Figure 2.15.

2.5 Summary of Background

In this chapter, we introduced the following concepts - the FPSP camera and the binary edge map that it produces. Due to a lack of access to the SCAMP-5 device, we will recreate such edge maps from RGB images on the CPU for us to use as inputs in our loop detection. We then covered some brief characteristics of the

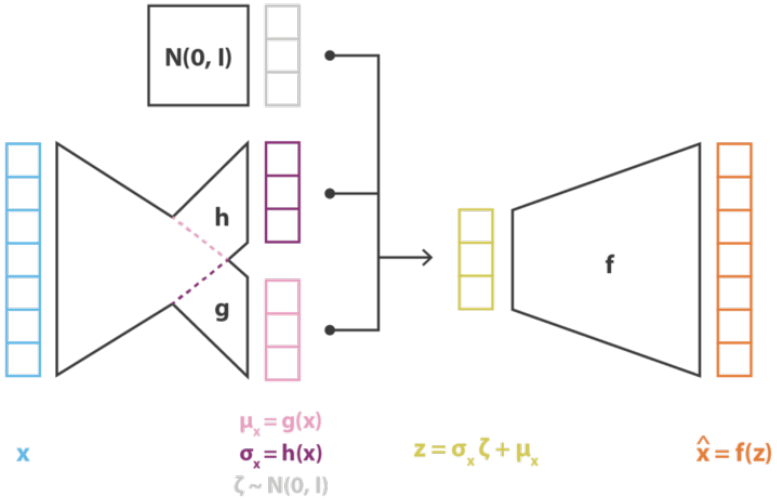

Figure 2.15: Example structure of a VAE. Image obtained from [39].

binary edge maps and highlighted the main difficulties in using them for loop closure detection. This was followed by an overview of neural networks, with an emphasis on convolutional autoencoders and the more advanced VAE which we will be using as our main feature extraction tool. Finally, we also highlighted some key metrics that we use to evaluate the efficacy of neural network models.

Chapter 3

Related Works

Due to the pertinence of the loop closure problem, there are many pre-existing developed algorithms tackling it. Although many of the research that follows usually cover the full range of the loop closure problem which includes the adjustment of the robots localisation post-detection of the loop, we scope into only covering the loop detection step as that is our object of interest. This section will give a brief overview of approaches utilised in this research area and how they have evolved over time.

3.1 Hand-Crafted Features Algorithms

Before the advent of neural networks, most of appearance-based approaches focused on first creating a set of hand-crafted features that describes a scene. As explained in Section 2.2, such features can be generally classified into local or global features. The next step is to then index the images. Indexes allows for quick retrieval of similar images based on some criteria derived from these hand-crafted features. Algorithms have also been designed for indexing to avoid a brute-force search method which can be inefficient. Criteria for good indexing algorithms include search accuracy, its efficiency and the memory usage.

3.1.1 Local Feature Algorithms

Local image features describe the appearance of a scene through identifying distinct patches within the image and computing a representation of each patch. Commonly used local descriptors include SIFT[28], SURF[3], BRIEF[5] and ORB[40]. Such local features usually differ in terms of its robustness - images with transformations and deformations applied to it should still have same local feature descriptions. Due to their level of detail, they also differ in computational time and complexity.[23] For example, the following research has shown that "ORB is the fastest algorithm while SIFT performs the best in the most scenarios. In the special case where the angle of rotation is proportional to 90 degrees, ORB and SURF outperforms SIFT and in the noisy images, ORB and SIFT show almost similar performances."[23]

Indexing algorithms that are used alongside local image include techniques such as "bag-of-words", Fisher Kernel and VLAD[22]. The most commonly and successfully used indexing algorithm for local image features is the "bag-of-words" algorithm.

This algorithm was modelled after text retrieval systems where a visual vocabulary is built through quantizing the descriptors into clusters which will be the visual 'words' for text retrieval. A histogram of word occurrences is then used to find the similarity between a query image with the existing images.[42]

An example is the widely known FAB-MAP which uses a combination of SURF descriptors and an extension of the "bag-of-words" to solve the loop closure problem. They do this by constructing and using a generative model of visual words based on their probabilistic relationships which were approximated by their use of a Chow Liu tree. This allowed the system to account for perceptual aliasing and attain linear time complexity.[9] FAB-MAP 2.0 was an improvement to the FAB-MAP achieved by defining a sparse approximation using an inverted index, which was "demonstrated to perform reliable online appearance mapping and loop closure detection over a 1,000 km trajectory, with mean filter update times of 14 ms."[10]

Many other algorithms that are based off the "bag-of-words" approach have been developed to outperform the FAB-MAP, albeit with some slight modifications to the type of local descriptors they use or some further improvement to the "bag-of-words" algorithm. An example is Galvez-L'opez and Tardos, who utilised binary local descriptors FAST and BRIEF to further speed up the loop matching process. Such local descriptors while not invariant to scale and rotation, are still very robust for loop detection with planar camera motions. On the other hand, these descriptors are quick to compute which greatly reduces computation time. They reported that "their whole technique, including feature extraction, requires 22ms per frame in a sequence with 26300 images, being one order of magnitude faster than previous approaches."[13]

3.1.2 Global Feature Algorithms

Global descriptors unlike local descriptors, describe the appearance of the complete scene and not of single points in it. Examples of such descriptors include GIST[36] and HOG[11].

GIST creates a low-dimension representation of the entire image called a spatial envelope. This is built from the responses of steerable filters at different orientations and scales. One global image feature algorithm is the BRIEF-GIST, which is an adaptation of the GIST descriptor which uses BRIEF as a holistic descriptor for a complete image. BRIEF-GIST is done through down-sampling the image to a suitable size close to the descriptor patch size and then calculating the BRIEF descriptor around the center of the down-sampled image. Comparing the similarity of two descriptors can then be easily done using the Hamming distance. Since operations are binary in nature, this allows for a highly efficient algorithm.[45]

SEQ-SLAM is another algorithm which uses the complete scene as input. The uniqueness of SEQ-SLAM is in its local best match approach where instead of comparing all images globally, comparison is done within a sequence of images (i.e. its neighbourhood). The author's then proceeded to do loop closure detection via identifying coherent sequences of the local best matches. SEQ-SLAM has been highly

successful algorithm for the loop closure problem because of its ability to remain robust in extreme perceptual changes.[31]

Lastly, Wang et al.[48] similar to us, also developed a binary image map which they then apply image to image comparisons to via a logical operation. However, rather than tabulating edges, their binary image was tabulated using a log spectral residual method which requires more computation than simple edge thresholding. Also, unlike their approach, we utilised a neural network within our matching process.

3.1.3 Binary Edge Maps

Due to the nature of the project described in Section 1, the objective is to extend the work of Murai et al.[32] to increase the robustness of the visual odometry system running on the FPSP device by enabling it to perform loop detection. Murai et al. were able to achieve a low-power and high-frame rate visual odometry system through the use of FPSPs which allowed them to perform quick parallel computations within each pixel, allowing them to transfer a reduced set of features from the vision chip to the device for further processing.

This reduced set of features consist of the coordinates of corner key-points produced by the FAST algorithm and a binary edge map produced by the use of sobel filters. For our inputs, we constrain our inputs to only using the binary edge map produced by the FPSP feature detection algorithm rather than full-scale RGB images as it will allow us to extend the localisation capabilities of the device while still retaining its advantages of having a low energy consumption and a high frame-rate.

3.2 Neural Networks

With the advent of neural networks, research has moved towards trying to exploit learned features instead of utilising hand-crafted traditional features to tackle the loop detection problem.

Neural network approaches usually consists of 2 steps: Firstly, a neural network is trained to obtain learned features and secondly, a comparison is required to determine the similarity of two scenes. Furthermore, such approaches can be broadly classified into supervised and unsupervised methods.

3.2.1 Supervised Methods

Supervised neural networks are those with truth labels which describe if a true loop closure has occurred.

Initial works surrounding supervised methods utilised pre-trained convolutional neural networks to obtain learned features. The first users of such a method were Chen et al. who used an Overfeat network to extract such learned features. Using these learned features, image similarity was then tabulated to determine potential loop closures. These hypotheses are then filtered down via a Spatial Continuity Check

and a Sequential Check to determine loop closure candidates.[8]

Another example is Hou et al. who used a pre-trained CNN model to create individual descriptors at every layer of the network. In order to gauge similarity, a nearest neighbor image is identified based on Euclidean distance. Finally, a threshold is then applied to determine if loop closure has occurred. The authors reported that "such learned features were shown to perform similarly to hand-crafted descriptors in environments without illumination change, but outperforms hand-crafted descriptors by a significant margin when the robot navigation environment experiences inevitable illumination change in long term operations."[19]

Due to the large dimension of such learned features, the image matching process remained computationally expensive. For this, Hou et al. improved their algorithm by using the "bag-of-words" approach highlighted in Section 3.1.1.[20]

However, these studies all utilize pre-trained CNNs that are trained on an object-centric dataset, which is different in nature from the place recognition task. In order to solve this, researchers began to train their own network for place recognition. Arandjelovic et al. did this by using the Google Street View Time Machine to obtain a large dataset of multiple panoramic images depicting the same place from different viewpoints over time. They then used a triplet loss scheme to train the network, where a triplet consists of two matching images and one non-matching image.[1]

3.2.2 Unsupervised Methods

As the neural network needs to be trained on large varied environment datasets to perform well, to minimise human effort in trying to label data sets, unsupervised deep learning methods were also used to solve the loop closure problem. The first attempt at this was by Gao et al. who built a stacked de-noising auto-encoder (SDA). Regions of interest or patches were first identified via patches extracted from sparse key point detection algorithms like SIFT, FAST or ORB and fed into the SDA. A similarity score is tabulated between current frames and previous frame in which loops are detected if the similarity scores exceeds a threshold.[14]

To further enhance the SDA's invariance to variations in viewpoint. Merill et al. adjusted their network to mimic the viewpoint variations that it will encounter in reality. They do this by corrupting their input images using homography deformations and trained the network to recover the HOG of the original image using these corrupted images, which they then compared across images. Their approach is also meant to be more lightweight than the approach utilised by Gao et al.[30]

3.2.3 Summary of Related Works

While there are many existing approaches which uses some form of binary features, like BRIEF and spectral residual methods, none of the past works involved the use of binary edge images as a feature for loop detection. Also, those works are restricted to hand-crafted features, whereas in our work, we will attempt to obtain a

more abstract and compressed binary edge image representation by combining such features with a neural network autoencoder. As far as we know, our approach is unique in nature and has not been attempted in the past.

Since manual annotation of loop closure matches are extremely time consuming, our goal is to train the neural network in an unsupervised manner. As such, the work of Gao et al[14] forms the starting point of our research, where we will adapt their autoencoder model to work with binary edge maps instead of full-scale RGB images.

Chapter 4

Experiment Setup

The following section highlights how the data sets were setup, followed by an outline of the evaluation metrics we use to assess the efficacy of our framework.

4.1 Data and Pre-processing

The main data set used for training and validation in this particular experiment was the deer-walk data set and diamond-walk data set produced by Saeedi et al. from Imperial College London[41]. Each data set consists of 1281 RGB scene images, produced synthetically by artificially modelling a robot's movement within a high quality rendered 3D space. The deer-walk scenes mainly suffer from crowding and occlusion due to the large number of objects that are captured within each scene, whereas the diamond-walk scenes suffer from poor lighting conditions. Figure 4.1 shows sample images from these particular data sets.

Figure 4.1: Top: Sample Images from deer-walk data set. Bottom: Sample Images from diamond-walk data set. Image obtained from [41].

4.1.1 Modelling SCAMP-5 outputs

Here we outline how we transform RGB images to match the binary edge images produced by the BIT-VO. RGB images were first resized into 256x256 images before being converted to gray-scale images. The next step then involved using sobel filters. Sobel filters were used as its most closely resembled the edge detection algorithm applied on the BIT-VO itself. Figure 4.2 shows the structure of the 3 x 3 x- and y-axis sobel filters.

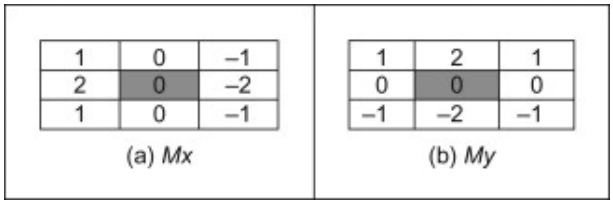

Figure 4.2: 3x3 x-axis sobel filter(left), 3x3 y-axis sobel filter(right). Image obtained from [34].

Sobel filters are approximations to derivatives of an image. By approximating gradients in the x and y directions and computing the magnitude of the gradient, we are able to identify potential regions with edges. This is because an edge is a point where the intensity values of its neighbouring pixels vary the most significantly. After tabulating a gradient magnitude for each pixel of the original image, a thresholding can then be applied to select edges from the full set of potential edges. This is done by simply categorising the points whose gradient magnitude is larger than the threshold as edges. Here we use simple thresholding, similar to the one found in the BIT-VO edge algorithm, where points above the threshold is set to a value of 1 and those below are set to a value of 0, hence creating a binary edge map. An overview of the algorithm for tabulating the binary edge map is shown in Algorithm 1.

Algorithm 1 Pseudocode: Binary Edge Map

procedure BINARYEDGEMAP($image, threshold$)
    Resize $image$ to 256x256 pixels
    Convert $image$ from RGB to gray-scale
    $g_x = image * sobel_x$ ▷ $sobel_x$: x-axis sobel filter, $*$: Convolution
    $g_y = image * sobel_y$ ▷ $sobel_y$: y-axis sobel filter
    $magnitude = \sqrt{g_x^2 + g_y^2}$
    for each $pixel$ in $magnitude$ **do**
        if $pixelvalue >= threshold$ **then**
            $pixelvalue = 1$
        else
            $pixelvalue = 0$
    return $magnitude$

Figure 4.3 shows the corresponding gradient magnitude map and binary edge map produced from an example image via the process described above. We note that the binary edge map here has been re-scaled to intensity values of 0 and 255 for easier visualisation.

Figure 4.3: Binary edge image conversion on deer-walk sample image.

Also for these RGB images, we will ignore the noise produced by the SCAMP-5 device on images. This is mainly because of a lack of access to the device, coupled with the fact that it is hard to specifically model noise that will be similar to those produced specifically by that device. Furthermore, we will be testing on a set of images produced by the SCAMP-5 device itself to ensure that the algorithm remains robust to the noise produced by the SCAMP-5 device.

4.1.2 Constructing approximate truth labels

As both deer-walk and diamond-walk data sets lacked truth labels of when loop closure occurred, a rough estimate of loop closure was constructed using positional and rotational data captured by the data set. Such positional and rotational data includes x-, y- and z-coordinates and w-, x-, y-, z-quarternions for each image.

The first step of this estimation process involved clustering of the images using their x- and y-coordinates. We have opted to remove the z-coordinate from consideration as the variation of this coordinate among all data points within the deer walk data set was minimal. Minimum and maximum values were tabulated for both the x- and y-coordinates of all data points to determine the range of values which encompasses the entire field of movement of the robot. Regions were then created by splitting this field into equal intervals along each of these axes. Images that fall within the same region are determined to be loop closure candidates. This clustering process is outlined in Figure 4.4.

From here, we first initialise an initial truth matrix. A 1281x1281 matrix was created with each index representing an image (starting from an index of 0). If two images are considered a loop closure match i.e. falls within the same cluster above, then the corresponding entry in the matrix will be 1. Similarly, if they do not belong to the same cluster, the corresponding entry will be 0. For example, if image 2 and image 4 belongs to the same cluster, then the value of the matrix at (2, 4) and (4, 2) will both be 1. An example of such a truth matrix is shown in Figure 4.5.

Here are few of the properties we note about the truth matrix: the value at similar indexes represent the same image i.e. the diagonals of the matrix will always be labelled as 1. Also, the truth matrix is symmetric since a positive match between

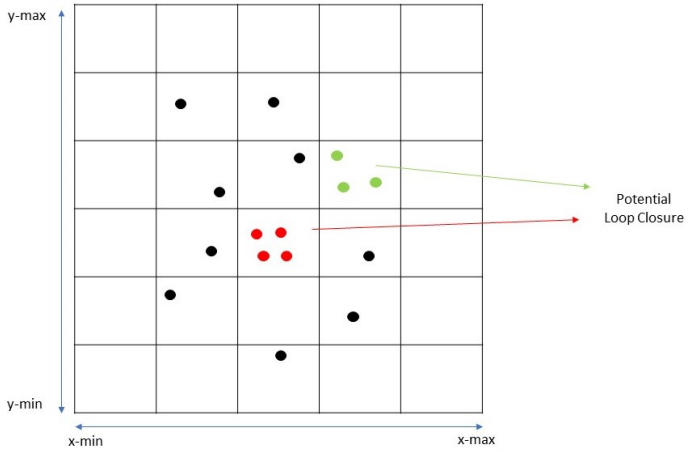

Figure 4.4: Clustering of images based on x-, y-coordinates.

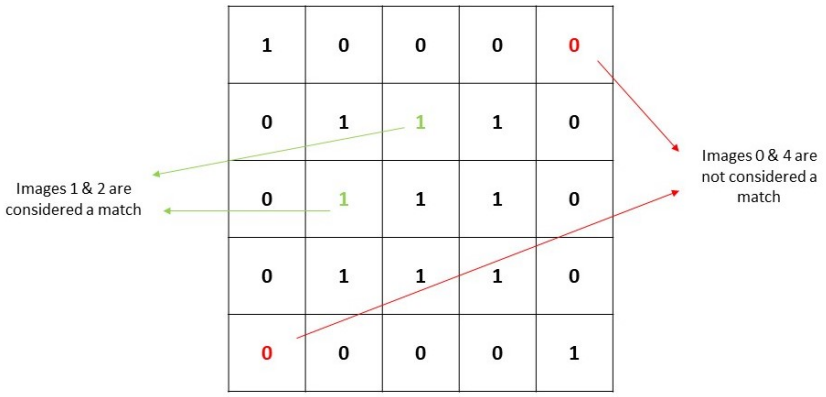

Figure 4.5: Example of a truth matrix.

image 1 and 3 will also mean that the converse is true.

The next step would be to then consider the rotational data. For each cluster of images determined from the previous step, we tabulate the cosine distance of every possible pair of images within that cluster. This was done by converting the quarternions into 3D rotational vectors before tabulating cosine distance as seen in Equation 4.1.

$$CosineDistance(\mathbf{x}, \mathbf{y}) = 1 - \frac{\sum_{i=1}^{n} \mathbf{x}_i \mathbf{y}_i}{\sqrt{\sum_{i=1}^{n} \mathbf{x}_i^2} \sqrt{\sum_{i=1}^{n} \mathbf{y}_i^2}} \tag{4.1}$$

Image pairs within a cluster with a cosine distance of more than a threshold had their truth matrix value changed from 1 to 0 i.e. were no longer considered matches, creating the final truth matrix. In order to find the proper threshold for the cosine distance, image pairs which have cosine distances larger than and closest to

the threshold were printed and visually inspecte to determine if they were matches. Here, we ensure that the threshold selected was loose enough to allow for images with slight viewpoint changes to be considered a match as this is a desirable quality of a loop closure detection system. Figure 4.6 shows example of these images after deciding on an orientation threshold.

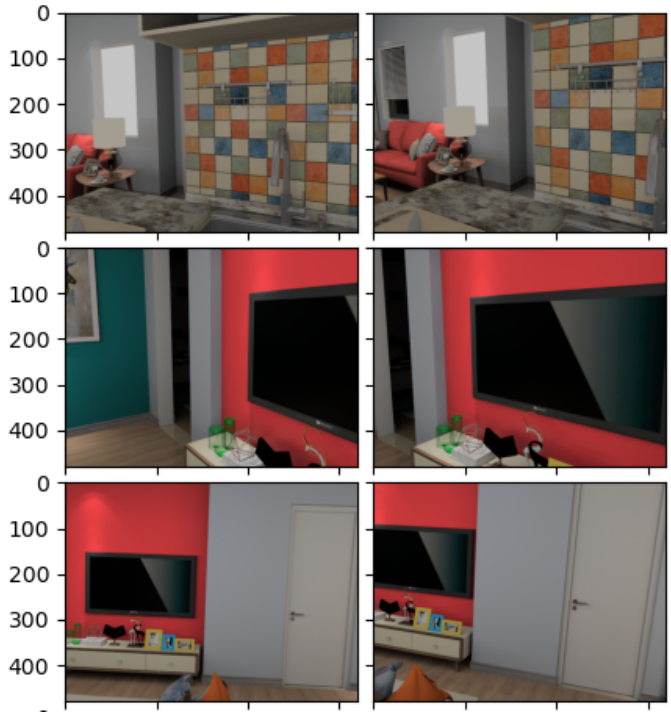

Figure 4.6: Accepted image pairs with cosine distances close to threshold.

When deciding on the positional thresholds, we set a loose segmentation for positional data. This is to allow for more loop closures that are not within immediate frames to be identified as these are the loop closures that we are interested in. This is because loop closures of images taken within a short timeframe are already handled by existing detection systems and are threfore less of interest than those which are of a significant timeframe apart i.e. loop closures away from the main diagonal of the truth matrix. As a result, in order to prevent too many erroneous matches, we correspondingly set a stricter restriction on rotational threshold. The truth matrix before and after applying this rotational thresholding is shown in Figure 4.7.

4.2 Evaluation tools

To evaluate how effective the various adjustments to our neural network has on loop closure detection, as well as provide a quantitative evaluation of the final framework, we utilise the following evaluation tools.

4.2.1 Precision-Recall Curve

The main tool that we will be using would be a precision-recall curve. The precision-recall curve plots precision against recall. See Section 2.3.3 for more details on Pre-

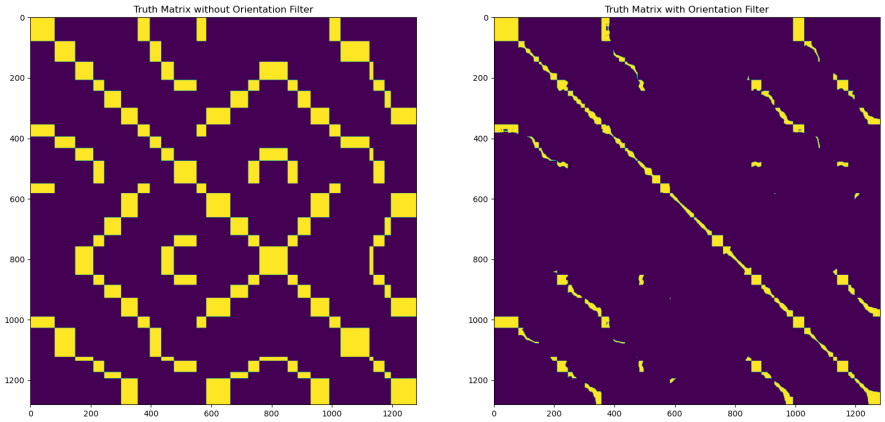

Figure 4.7: Truth matrix before and after rotational thresholding.

cision and Recall.

Precision and Recall values can be obtained via computing probabilities of whether a loop closure has occurred. This probabilities can be determined by designing an algorithm which tabulates a similarity score between a pair of images for all possible images. By setting a threshold on the probability, we are then able to classify all predictions into 2 categories - loop closure for image pairs whose similarity score is above the threshold and non-loop closure for image pairs whose similarity score is below the threshold. This can then be compared to the truth value which can be extracted from the truth matrix highlighted in the previous section. From there, we are able to tabulate both precision and recall scores.

We then plot the precision-recall curve by varying the threshold probability. For example, setting a high threshold, will reduce the number of positive loop closure predictions. This is likely to reduce the value of precision and increase the value of recall since the number of false positives will increase and the number of false negatives will decrease. Conversely, a low threshold will increase the number of positive loop closure predictions, increasing the value of precision and decreasing the value of recall. The precision-recall curve can then be plotted by plotting the different trade-off values between precision and recall as we vary the threshold.

Ideally, a sign of a good classification algorithm will have high precision without sacrificing too much recall and vice-versa. Getting a single-valued proxy which gauges the overall quality of an algorithm is useful for quick evaluation of the algorithm. For this case, we use the area under curve (AUC) metric i.e. a larger area under the curve is indicative of a better classification algorithm since it is indicative of a low trade-off between precision and recall. Figure 4.8 shows an example of such a curve and the AUC metric.

Precision-recall curves are used over accuracy for two main reasons:

1. Loop detection algorithms most of the time leads to imbalanced data-sets (i.e. less loops than non-loops). Calculating accuracy in such cases tend to be misleading since it becomes heavily influenced by the majority class.

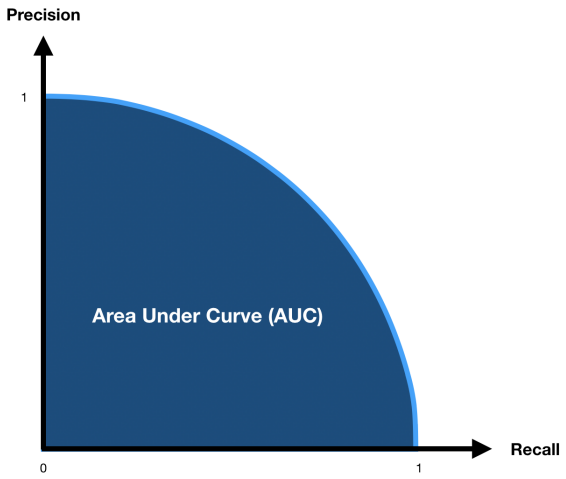

Figure 4.8: Sample Precision-Recall Curve and AUC metric.

Therefore, examining precision and recall will help us to avoid such a trap.

2. As mentioned previously, perceptual aliasing is especially important in the context of a loop closure detection algorithm. Being able to track the precision value will give us a better visibility on whether our algorithm is performant against such factors.

Since our algorithm provides us with predictive similarity score matrices with values [0, 1] structured in the same way as the truth matrix outlined in Section 4.1.2 where the value in each row i and column j represents the similarity score of images i and j, we can just treat the similarity scores as probabilities and compute the precision-recall curves accordingly. Since these matrices are symmetric, we only need to use the lower bottom left triangle (excluding the main diagonal) of the matrices to compute these curves. Also, in loop closure detection, we are more interested in finding loop closures between images that are taken after a significant time frame has elapsed i.e. similarity matches away from the main diagonal of the matrices. Therefore, in most cases we only compare a new image with a previous one if there exists a minimum number of frames taken in between both images. We term this minimum number as frame-lapse. An example is shown in Figure 4.9 shows the values that we should consider when we tabulate the precision-recall curves on loop closures with a frame-lapse of 7 i.e. the yellow region.

4.2.2 Post-PCA 2D Latent Space Visualisation

To visualise the organisation of the latent space, we normalise the data and apply a PCA to the current set of latent features to reduce it to a 2-dimensional latent space. (See Section 2.4.1 for more information on PCA) We then print this latent space on a 2D scatter-plot image. Points of the same colour belong to the same region based on their positional segmentation as specified in 4.1.2. For each sector, orientation information is then incorporated into this visualisation via picking 3 reference images within that region. Images within each positional sector are then further broken down into 3 sub-groups based on their cosine distances with each of the 3 reference images i.e. it was classified into the same subgroup as the reference image with the

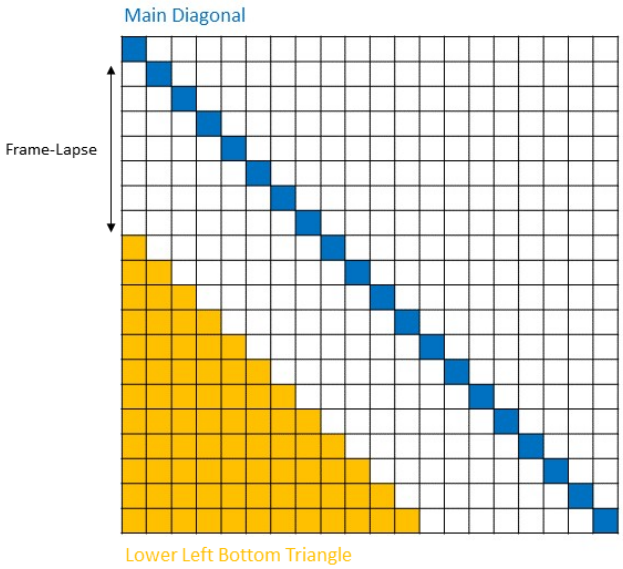

Figure 4.9: Example matrix evaluation - yellow cells: lower bottom left triangle with frame-lapse of 7, blue cells: main diagonal.

shortest cosine distance. Images that do not fall within the orientation threshold of any of the 3 images were just removed from the visualisation diagram. As a result, the visualisation may not be representative of every single test point. In order to mitigate this effect and maximise the number of images covered, reference images were chosen to be the first, middle and last image arranged by the time they were taken. An example of the latent space visualisation is shown in Figure 4.10.

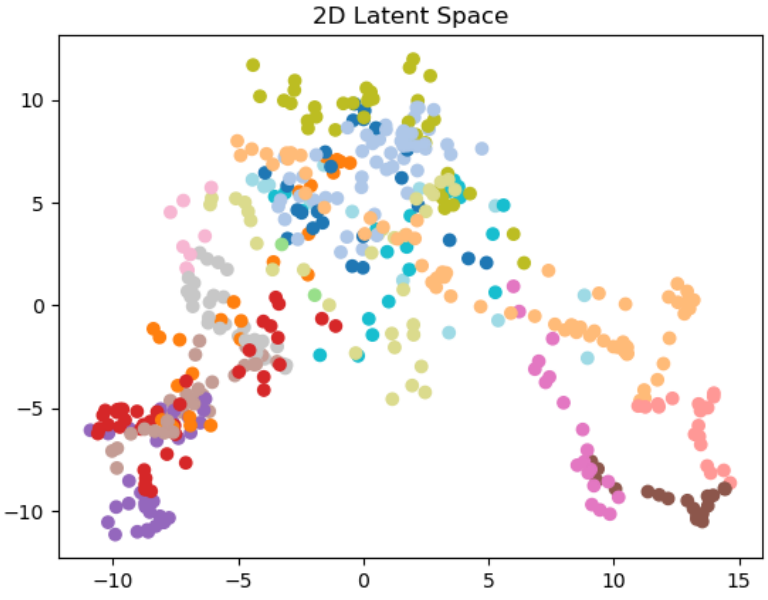

Figure 4.10: Example of latent space visualisation (Post-PCA).

4.3 Summary of Experiment Setup

Here we highlighted how we pre-processed the data mainly for the deer-walk and diamond-walk data sets to match those of the BIT-VO outputs (i.e. binary edge maps). This will extend to all test data sets within Section 6 that uses RGB images as well. Next, we also provided an overview on how we approximated truth labels for these synthetic data sets which do not come with loop closure labels. Finally, we discussed two main tools that we will be using to evaluate our framework, namely the precision-recall curve as well as the post-PCA 2D latent space visualisation.

Chapter 5

Methodology

The following section highlights the methodology used in our loop closure detection algorithm. Our methodology consists of two main steps, the first of which is the usage of an autoencoder to produce a compressed latent space of each image that captures the most salient points of that image. The second step is to then match the latent space in order to determine if an actual loop closure occurs.

5.1 Compressing into latent features

Here we highlight the structure of our autoencoder that we used to compress the initial image into a smaller latent space. We will also explain how we derived the final autoencoder structure using some of the evaluation tools that was explained in Section 4.2. Results here are based on training the various models on a subset of the deer-walk data set and then testing on a separate blind subset of data from the same data set, before plotting the precision-recall curves. While training data was shuffled, testing data was fed into the encoder sequentially based on their timestamp in order to mimic the processing of an actual camera. For this portion, we tested on loop closures near the main diagonal (i.e. frame-lapse of 0) as well as those away from it, since in this stage, we are less focused on identifying loops but rather more interested in obtaining a good latent representation of similar looking images. See Section 4.2.1 on details of frame-lapse.

5.1.1 Global Image vs Local Patches

The unsupervised method executed by Gao et al.[30] was a method that utilised local features as they used the SIFT methodology to first extract key-points before taking a surrounding patch over each of those key-points. Comparison of two images will hence involve an additional step of first matching these patches. Here, they used a brute force match and a fast approximate nearest neighbor algorithm to execute this step. For our case, since the BIT-VO system can also obtain corner coordinates via FAST detection, we can obtain similar patches of our full binary edge image. An example can be seen in Figure 5.1, where we took 40x40 patches of the top 30 key-points extracted by the FAST algorithm.

However, the difficulty in using local patches for our use case is the fact that binary edge images already contain very little information, hence making it very dif-

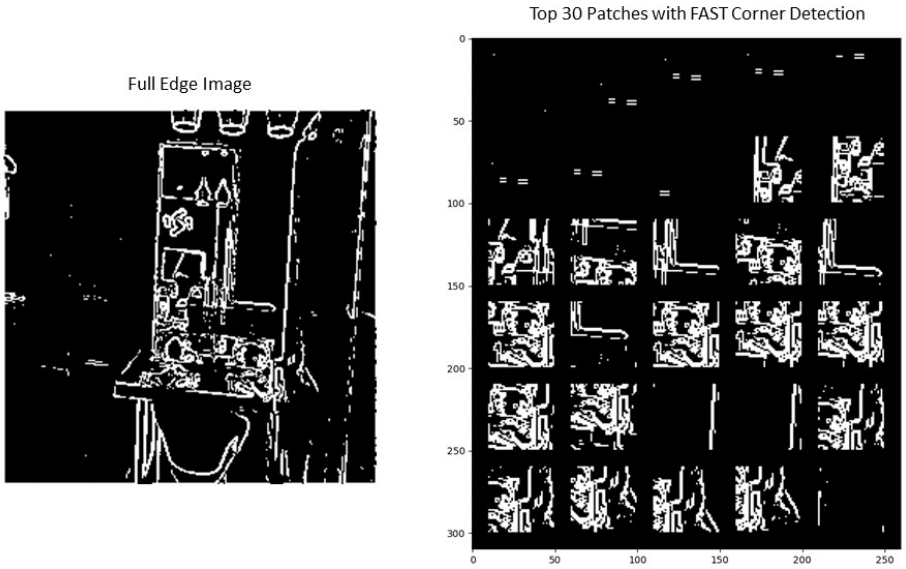

Figure 5.1: Example of local patches extracted using FAST corner detection for binary edge images.

ficult to distinguish or accurately match local patches. As we can see, some of the patches contains single lines which can be present in many patches of other edge images. While there are algorithms that uses local patches to do matching for edge images[51], they utilise additional gradient orientation information which is unfortunately not an output of the BIT-VO device. As such, we chose to feed in full images into our autoencoder for compression instead.

5.1.2 Reference autoencoder structure

Due to the fact that we use global images instead of local patches, using a fully connected autoencoder like the one used by Gao et al., will result in a very slow training process. As such, we opted to use a convolutional autoencoder instead. The convolutional autoencoder we choose to base our network off is the SegNet[2]. Since the SegNet is able to effectively perform segmentation task for a scene, we hypothesise that the representation it learns within its latent space will also contain sufficient information for us to compare between two different scenes. The structure of the SegNet is shown in Figure 5.2.

The encoder portion of the SegNet consists of 13 convolutional layers with filter size of 3 x 3, with padding of 1 and a stride of 1. As a result, each convolutional layer outputs an image of similar height and width of its input. It also contains 5 2 x 2 max pooling layers which are responsible for downsizing the image by half at each of this layer. The depth of the 13 convolutional layers increase as we move deeper into the network, with convolutional filters before each pooling layer sharing the same depth i.e. the first 2 convolutional filters has a depth of 64, the second two has a depth of 128, etc. Depth information has been added below each of the convolutional layers in Figure 5.2, for easier reference. Also, a batch normalisation, followed by a ReLU activation function is applied at the end of each of the convo-

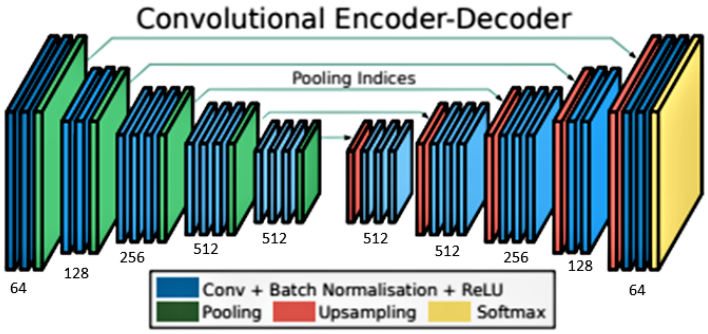

Figure 5.2: Structure of the SegNet Network. Image obtained and adjusted from [2].

lutional layers.

For the decoder portion, switch unpooling is used to upsample the image where the pooled indices are obtained from the encoder. The SegNet consists of the same 13 convolutional layers just positioned in reverse. This is unique to the SegNet as most other convolutional autoencoder uses transposed convolutions in their decoder. The rationale here is that with post-switch unpooling, the image becomes very sparse and that adding convolutional layers will allow it to make the feature map denser. Finally, a softmax activation is applied on the output.

Apart from an increase in efficency of using a CNN, advantages of using the general structure of the SegNet include the following:

1. The multiple layers of downsampling through the pooling layers will make our images more robust to viewpoint and scale changes which was one of the main problems we highlighted in Section 2.2.1. This high number of layers will also make our feature map more dense which is important because of the sparse nature of binary edge maps where there are little positive values which are all far apart.

2. SegNet does not utilise fully connected layers as its latent space is just the output of the last pooling layer within the encoder. This allows us to retain spatial information of the image within the latent space, which was shown by Hou et al.[19] to have improved loop detection performance compared to outputs of fully-connected layers.

In the next few sections, we dive into how we adjust specific portions of this architecture to fit our use case of binary edge inputs and the results that we obtain from these adjustments.

5.1.3 Loss function and output activation

Since our input to the autoencoder are binary images instead of full-scale RGB images, certain changes were required for both the output activation layer and the loss function. For autoencoders, in general, the input image $\mathbf{x}$ and its reconstructions $\mathbf{x}'$

should be viewed in a probabilistic framework rather than a deterministic one. Vincent et al.[47] gave a derivation of how this will affect the loss function and output activation which we will cover briefly here.

Using such a probabilistic framework, our loss function changes into the form we see in Equation 5.1. Intuitively, we want to get a reconstruction that has the highest probability of generating the original image, which in turn reduces the loss.

$$Loss(\mathbf{x}, \mathbf{x}') \propto -\log p(\mathbf{x}|\mathbf{x}') \tag{5.1}$$

Since our inputs are binary images, $\mathbf{x} \in \{0,1\}^d$ we model each random variable X_i conditional on a deterministic $\mathbf{x}'$ as a Bernoulli distribution as seen in Equation 5.2.

$$P(X_i|\mathbf{x}') \sim Bernoulli(\mathbf{x}'_j) \tag{5.2}$$

Next, because $\mathbf{x}'_i$ is a parameter of a Bernoulli distribution, it needs to be within the interval 0 and 1 (i.e. $\mathbf{x}'_i \in [0,1]^d \, \forall \, i$). Therefore, we choose a sigmoid activation layer for our output to ensure that this constraint is fulfilled. The sigmoid function is shown in Equation 2.2.

Correspondingly, the derivation of the loss function will be as seen in Equation 5.3, where n is the number of samples in the training dataset. This is otherwise known as a Binary Cross Entropy loss function.

$$\begin{aligned} Loss(\mathbf{x}, \mathbf{x}') &= -\log p(\mathbf{x}|\mathbf{x}') \\ &= -\log \prod_{i=1}^{n} p(\mathbf{x}_i|\mathbf{x}'_i) \\ &= -\log \prod_{i=1}^{n} (\mathbf{x}'_i)^{\mathbf{x}_i} (1-\mathbf{x}'_i)^{1-\mathbf{x}_i} \\ &= -\sum_{i=1}^{n} \mathbf{x}_i \log \mathbf{x}'_i + (1-\mathbf{x}_i) \log(1-\mathbf{x}'_i) \\ &:= BCELoss(\mathbf{x}, \mathbf{x}') \end{aligned} \tag{5.3}$$

5.1.4 Activation functions of hidden layers

The next thing we consider would be the activation functions used in the hidden layers. For this, SegNet uses a ReLU function whose equation can be seen in Figure 5.3.

The problem with using the ReLU function in our use case is the fact that majority of the input image pixels are at a value of zero. This increases the chance of them being pushed to the negative region during training, to which the ReLU function just sets to zero. Ratios and magnitude of these negative values with respect to positive values are important information as it contains information like the number of non-edges or how densely packed edges are within the image, respectively. By setting them all to zero, we essentially lose this information. Furthermore, if majority of the values across the samples start reaching the zero region, the positive values will have smaller updates as well and the latent space will eventually converge

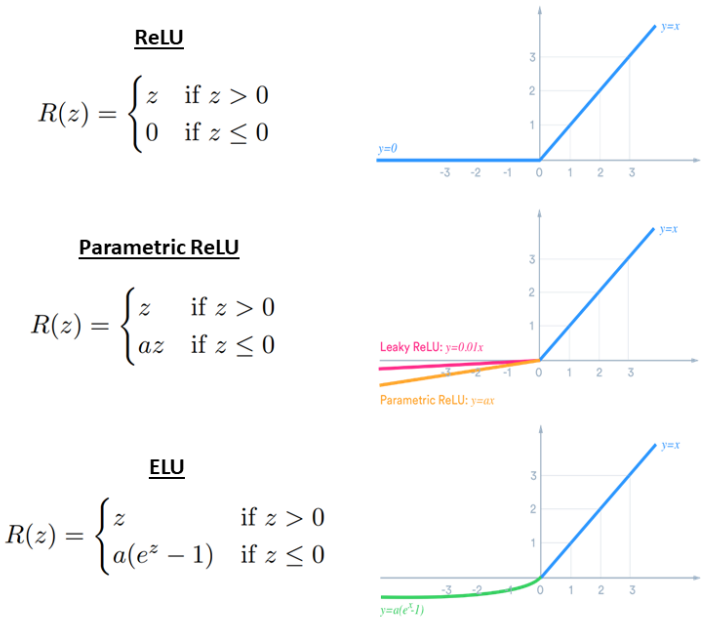

Figure 5.3: Various ReLU functions and their plotted graphs. Images obtained and adjusted from [27].

for all samples making different images indistinguishable. This phenomenon can be seen in Figure 5.4 where we trained our network with a ReLU function. This image shows the printed output from the intermediate layers within the encoder portion, we can see that some of them have gone to zero (i.e. complete black patches). There are also many grey / white patches with minimal intensity change, showing how the values have started to converge.

Therefore, we consider using either the Parametric ReLU function or the ELU function to avoid this problem. Their equations can also be found in Figure 5.3. As we can see from their graphs, both functions allows for some slight variation within the negative region, although for the Parametric ReLU function, this relationship is linear while for the ELU function, it is exponential. We tested using both of these functions on the same network where we use a value of $a = 0.05$ for the Parametric ReLU function and $a = 1$ for the ELU function, where the results are captured in Figure 5.5.

To analyse the reasoning behind why the ELU performs better than the Parametric ReLu, we plot the values of each element of the latent vector against their index for 5 sample latent vectors for separate models trained on each of these functions as seen in Figure 5.6.

As we can see, for ELU the negative values beyond a certain point are forced to converge to the value of $a = 1$ whereas for the Parametric ReLU, there wasn't such a restriction. Most of the time this negative values are representative of the 0 valued pixels of the original image which were pushed down while the network trains. Therefore, while it is important for such values to exist to allow for the model to capture some information about the amount of edges that exist within the image, in

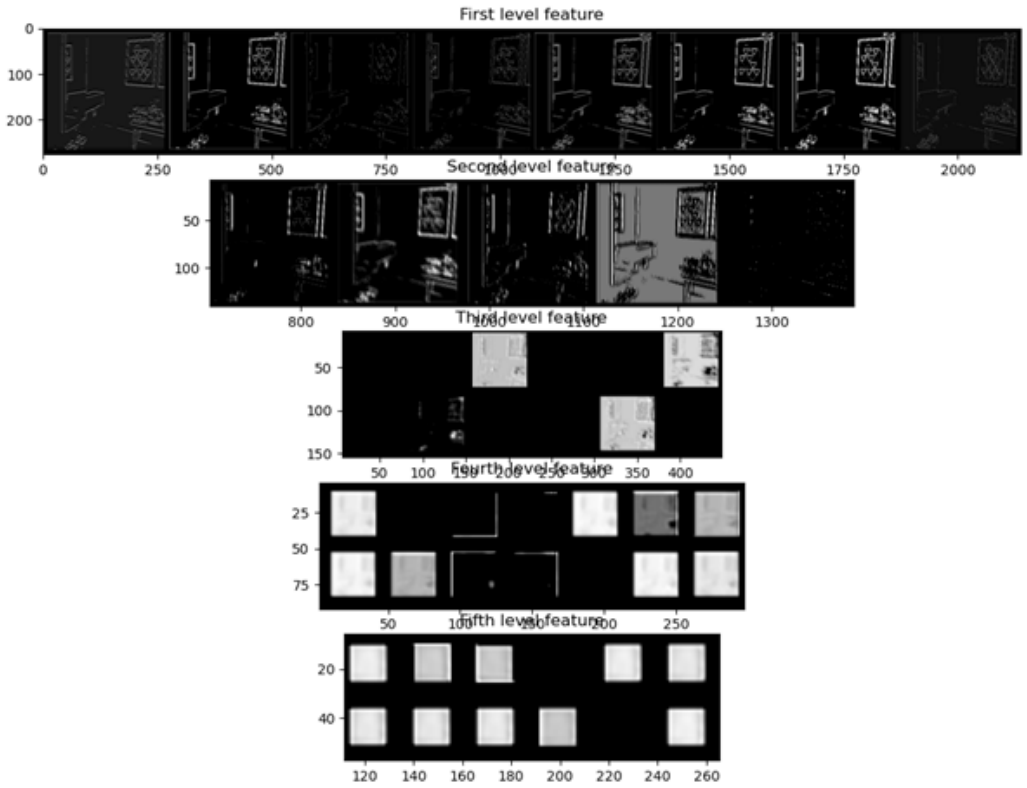

Figure 5.4: ReLU: Subset of output of intermediate layers.

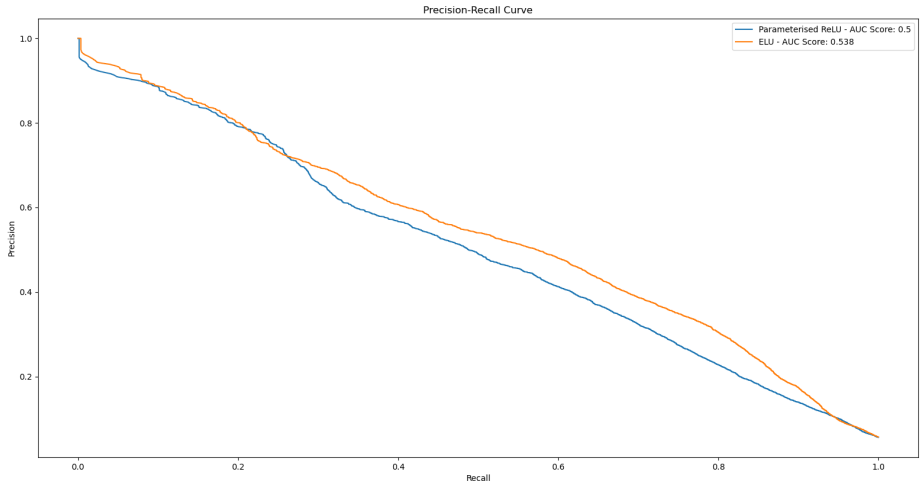

Figure 5.5: Precision-Recall Curve and AUC of Parametric ReLU($a = 0.05$) versus ELU($a = 1$).

cases where this black spot in shared among all images (i.e. corner of image where there are no distinguishing edges), they just contribute to noise. By not setting a cap on the negative values, these negative values start to diverge as seen in the

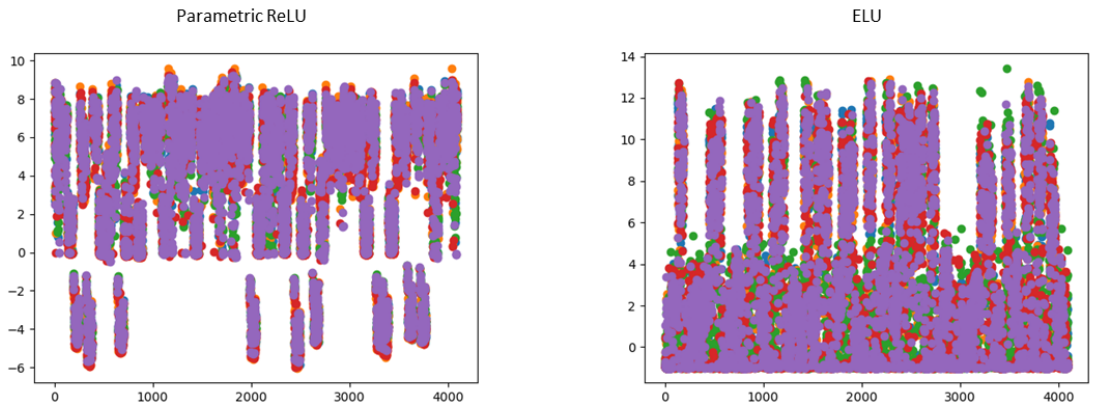

Figure 5.6: Latent vector value distribution of Parametric ReLU($a = 0.05$) versus ELU($a = 1$) for 5 samples.

case of the Parametric ReLU and the increased distance starts contributing more to the total distance between images, essentially providing more weight to the noise. The ELU on the other hand, restricts the effect this noise has on the total value by forcing the negative value to converge, but still continues to allow for negative values to exist, retaining useful information such as proportion and slight variations in the magnitude of negative values within the latent space, which might explain its better performance. To cap the amount of noise for the Parametric ReLU, we also tested a smaller value of $a = 0.01$ for the parametric ReLU function, which is otherwise termed as the Leaky ReLU function. However, results for this was very similar to that of the ReLU function, likely due to the sheer amount of zero pixels within the original image and the value of the negative gradient being way too small to distinguish the negative values.

We plot the same output of intermediate layers for the ELU function in Figure 5.7. Contrasted with Figure 5.4, we see intermediate outputs with more varied intensity values. Therefore, we replaced the ReLU activation function of all hidden layer in the SegNet with the ELU activation function for our revised architecture.

5.1.5 Strided convolutions

Another thing we attempted to vary was to use strided convolutions in place of the pooling layers and strided transposed convolutions in place of the unpooling ones. The rationale to using such convolutions is for the network to learn more information during the down-sampling phase rather than simply use a pooling layer which does not update the weight parameters. Such a replacement is shown to achieve better results by Springenberg et al.[44]. By using a stride of 2 and a padding of 1, with a 3 x 3 filter, we will be able to downsize the image to half its size, similar to a 2 x 2 pooling method.

For upsampling, since we no longer have access to switches, there are two alternatives that we consider, the first of which is using transposed convolutions to re-upsample the image. Transposed convolutions involved using a stride of 2 and a

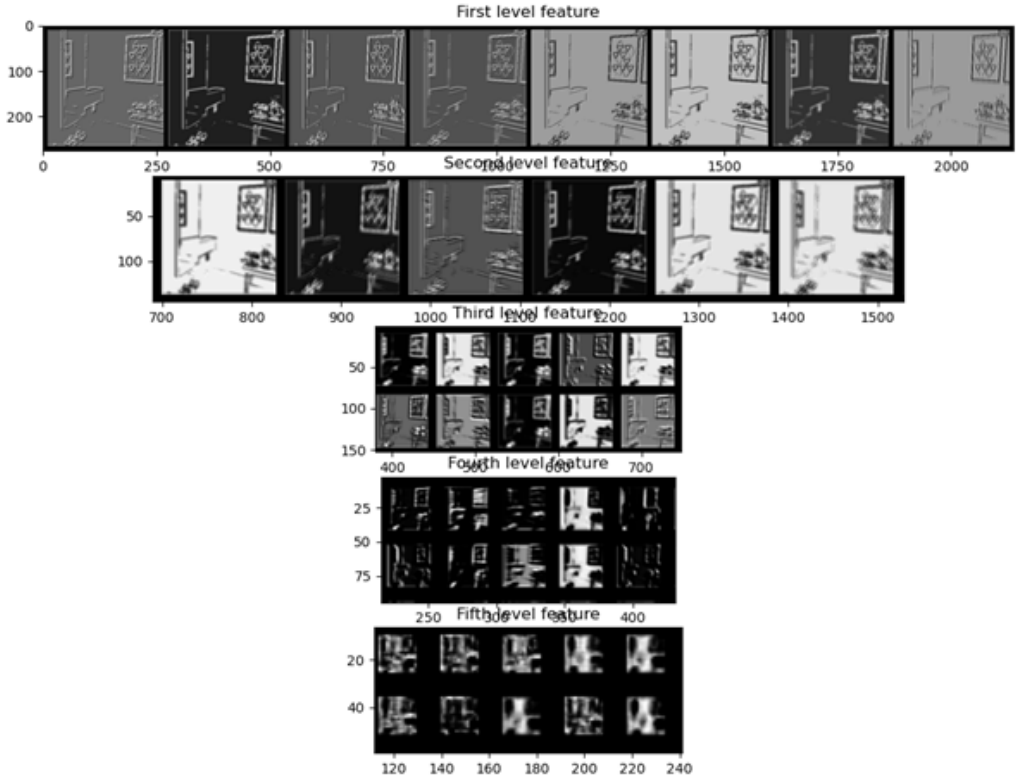

Figure 5.7: ELU: Subset of output of intermediate layers.

padding of 1 with a 3 x 3 filter and an added output padding of 1 which allows the convolution to double the size of the image. The second alternative is to use nearest neighbour unpooling where we just set all 2 x 2 pixels in the unpooled output to be of the same value as that of input.

Results of these tests on all three configurations are shown in Figure 5.8. From here we can see that pooling greatly outclasses both strided convolution methods which is in contrast with the initial hypothesis.

Upon closer inspection of the outputs of the intermediate layer within the encoder and the final latent representation produced as seen in Figure 5.9, we attribute this difference in performance to overfitting from the strided convolutions. As seen from the images on the right, striding creates uneven intensity gradients towards the bottom of the network. This can be seen where intensity in the latent layer looks much less smooth and almost checkerboard-like compared to those in the pooling case. This might be because striding convolutional layer causes the network to try to model even the smallest changes in gradients in the previous frame, which is a huge problem because of the extreme values of our inputs. This is evidenced by the fact that we obtain extreme gradient values within our latent space. This is contrasted with pooling whose latent space only models the general shape of the images, gradient values are pretty smooth within the prominent edges of the image. It is this ability to generalise which makes the pooling model better for our use case.

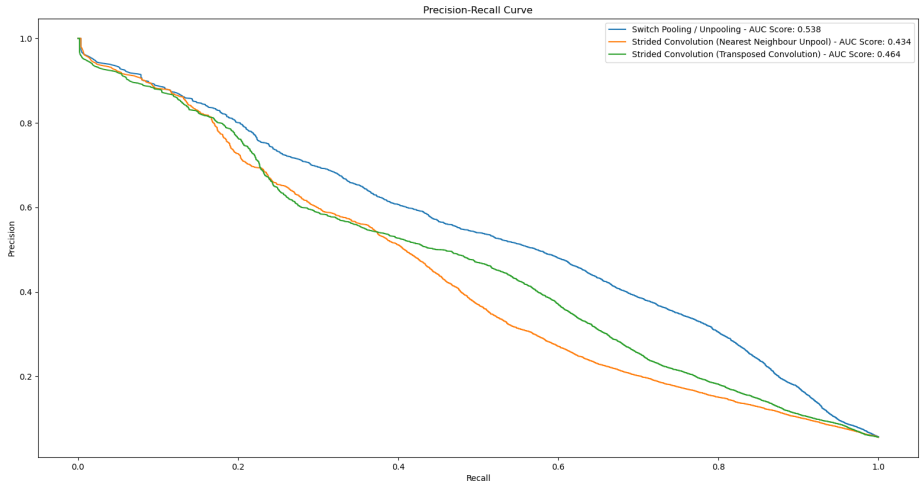

Figure 5.8: Precision-Recall Curve and AUC comparison of switch pooling-unpooling, strided convolution-strided transposed convolution and strided convolution-nearest neighbour unpooling.

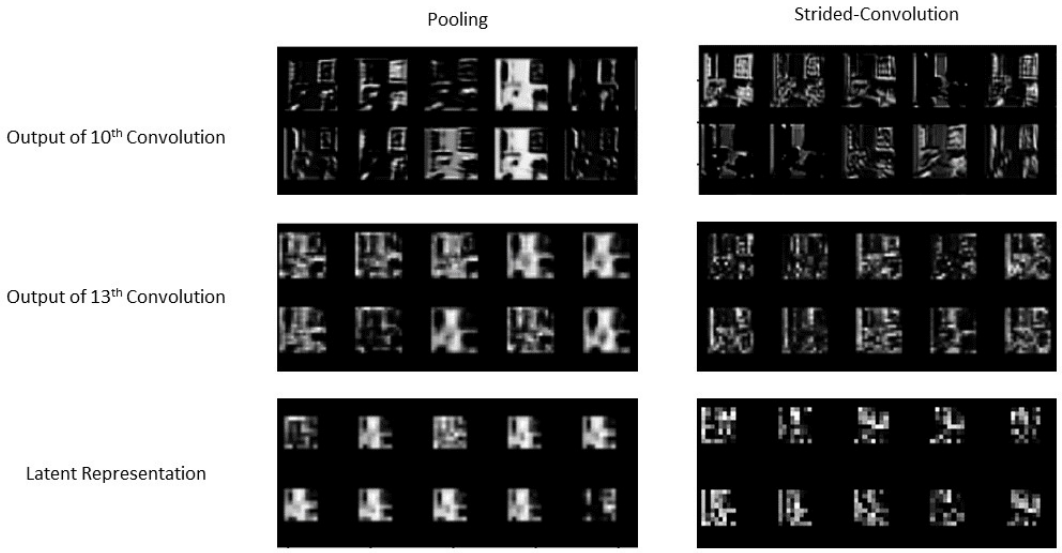

Figure 5.9: Intermediate outputs and latent representation of switch pooling-unpooling versus strided convolution-strided transposed convolution.

5.1.6 Depth of latent space

The depth of convolutional layers within the SegNet increases to a depth of 512 at the end of the encoder. Given the fact that SegNet uses full scale RGB images as input, it is expected that they will require a larger latent space to store information such as colour, intensity values, gradient, etc. On the other hand binary edge images are

much more compact in information compared to such RGB images and thus using a latent space of the same size for our use case will result in overfitting, where each image has its own unique latent representation. This will result in even the slightest change in the original image (i.e. a slight translation) to have a drastic change in its latent space representation, which is not a desirable quality of loop closure detection.

As such, we experimented with reducing this latent space. We firstly started off by dividing the depth of all convolutional layers by 4. This will give us a depth of 16-32-64-128-128 within the encoder and 128-128-64-32-16 within the decoder. We continued halving this latent space with every different network with a minimum depth of 8, meaning that no convolutional layers should have a depth smaller than 8. This minimum is in place so as to prevent losing too much information at the start before the later layers try to extract higher order information. Displayed in Figure 5.10 are the Precision-Recall Curves and AUC of these various setups.

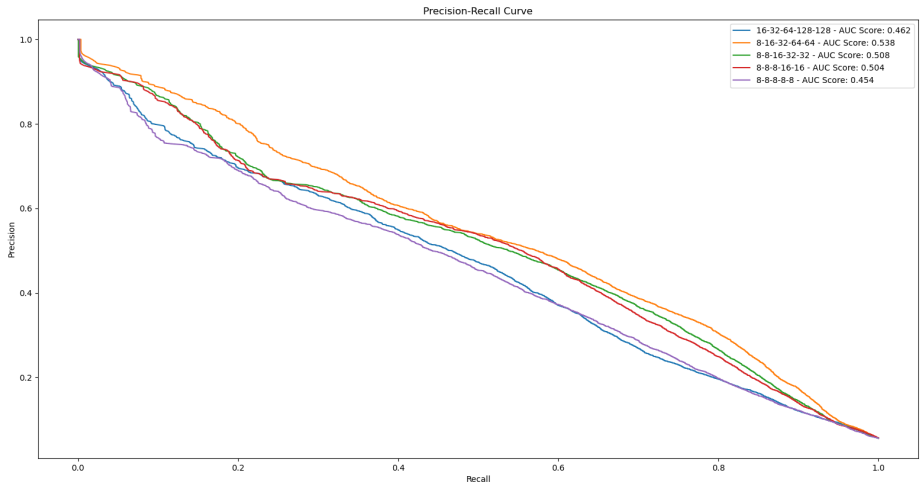

Figure 5.10: Precision-Recall Curve and AUC comparison of different depth latent space. Depth configuration for encoder is stated in legend with convolutional layers before each pooling layers sharing the same depth.

From here, we can see that the configuration of 8-16-32-64-64 within the encoder produces the best results. Configurations with smaller latent space than that tend to lose too much information and configurations larger than that tend to over-fit. This is confirmed by the quality of reconstructions produced by the decoder for different latent space sizes as seen in Figure 5.11, where the latent space of size 128 created a near perfect reconstruction, having only an average item loss of $\sim$23 compared to $\sim$244 from the one with a latent space of size 64. On the other end of the spectrum are those of latent space with size 8 and 16 which gave almost double the average loss per item, implying that it has lost too much information.

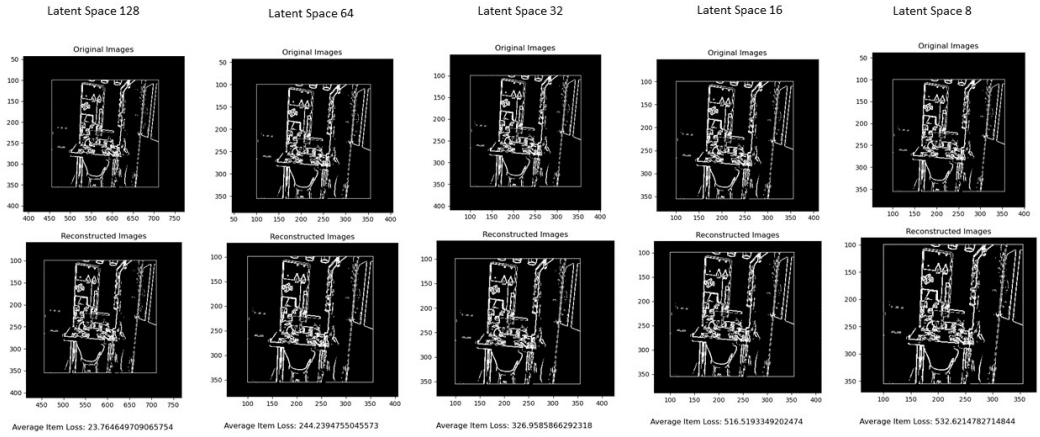

Figure 5.11: Reconstructions produced by decoder for different latent space sizes.

5.1.7 Regularising the structure of latent space

Due to the sparse and extreme image value property of binary edge images as explained in 2.2.1, the latent space of such images tend to be irregularly shaped as seen in the left panel of Figure 5.13, which is the projected 2D latent space of a normal autoencoder. As we can see the orange dots in the top left corner and in the center are placed very far apart with a large gap between them despite them having the same labels. This irregularly shaped latent space makes comparison via a distance metric very difficult.

As such in order to regularise the latent space to a more compact structure, we opted to use a VAE, whose mathematical details are highlighted in Section 2.4.5. We define the loss function as the negative of the ELBO function found in Equation 2.17 and make the same simplifying assumptions about the prior and variational estimated posterior being Gaussian. Since our decoder produces values within the range of (0, 1), we can model the decoder $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ as a multivariate Bernoulli distribution. Then the loss function will be as seen in Equation 5.4.

$$\begin{aligned} Loss(\mathbf{x}, \mathbf{x'}) &= -\frac{1}{2}\sum_{j=1}^{J}(1 + \log((\sigma_i^{(j)})^2) - (\mu_i^{(j)})^2 - (\sigma_i^{(j)})^2) - \sum_{i=1}^{n}\mathbf{x}_i \log \mathbf{x}'_i + (1-\mathbf{x}_i)\log(1-\mathbf{x}'_i) \\ &:= -KL(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}_i) \,||\, p_{\boldsymbol{\theta}}(\mathbf{z})) + BCELoss(\mathbf{x}, \mathbf{x'}) \\ &\text{where } \mathbf{z}_i^{(l)} = \boldsymbol{\mu}_i + \boldsymbol{\sigma}_i \odot \boldsymbol{\epsilon}^{(l)} \text{ and } \boldsymbol{\epsilon}^{(l)} \sim \mathcal{N}(0, \mathbf{I}) \end{aligned} \tag{5.4}$$

Rather than just applying the multivariate normal distribution on the current latent space, which will cause the model to become over-regularised, we downsize this latent space to a smaller latent dimensional representation to which we apply the multivariate normal distribution, by tabulating the KL-divergence within the loss term with the mean and variance of this smaller latent dimension. We would then continue to use the 64 x 8 x 8 latent space for similarity comparisons. Therefore, for downsizing, we added a layer of 1 x 1 convolutional filter which has a depth of 16 reducing the latent space to a 16 x 8 x 8 output first, before further compressing via a fully-connected layer to a 8 dimensional latent representation to represent the

mean and the-log variance. The reverse upsampling was then performed through fully-connected layers. This is represented in Figure 5.12.

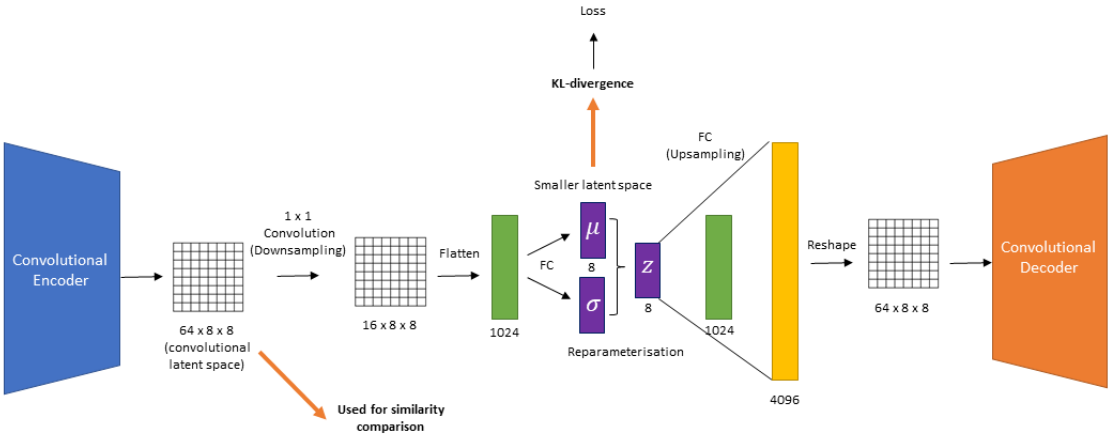

Figure 5.12: Convolutional VAE structure.

We chose this format of downsizing for two purpose: firstly we allow for a smoother transition into the smaller latent space, ensuring that the smaller latent space contains the most relevant information within the larger one and is hence representative of it. Also, we chose to use a convolutional filter so that our intermediate layer retains the spatial distribution of the larger latent space. This is to ensure that during back-propogation, the spatial distribution within the larger latent space still mimics the intermediate layers.

Unfortunately, as seen from Figure 5.13, while the initial VAE allows for better comparison of the latent representations of images as mentioned above, this came with a tradeoff of over-compressing the latent space.

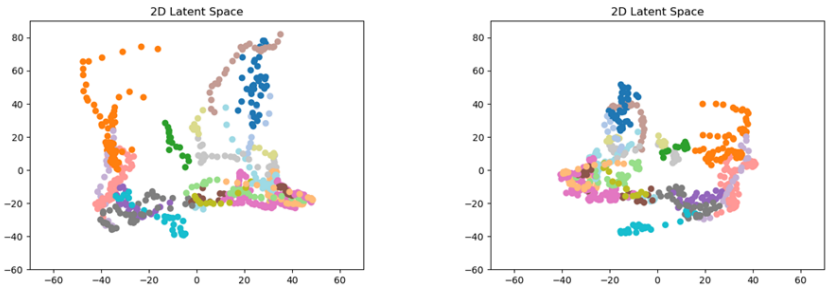

Figure 5.13: Left: Post-PCA 2D latent space of normal autoencoder. Right: Post-PCA 2D latent space of variational autoencoder.

Therefore, there was only a slight increase in performance compared to the normal autoencoder as seen by the comparison of their precision recall curves as seen

in Figure 5.14.

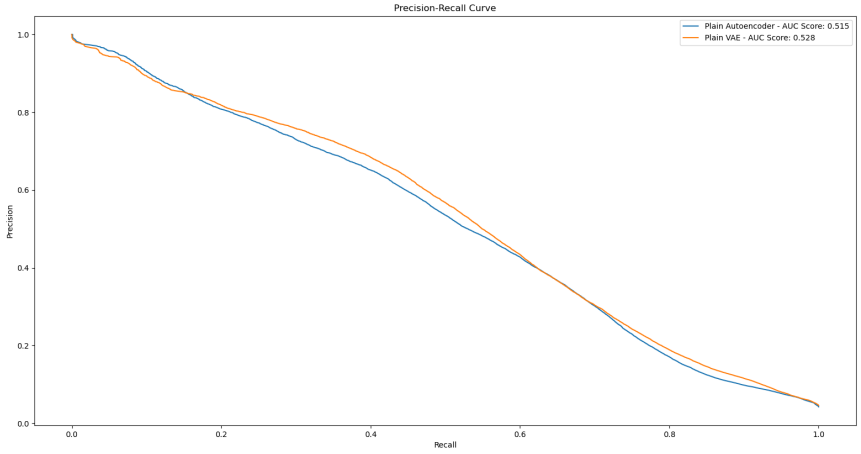

Figure 5.14: Precision-Recall comparison of normal autoencoder versus variational autoencoder.

The problem of the collapsing latent space into purely fitting the prior distribution before it learns proper information about the data because of the strong KL divergence term has been researched by Sønderby et al.[43]. There the authors proposed a few methods to try to reduce this effect and allow for deeper VAE networks that will still be able to learn higher order features of the image. For our use case, we apply 2 of these methods - applying batch normalisation and introducing a warm-up term.

The first thing that we consider is batch normalisation. Batch normalisation was first explored by Ioffe et al.[21]. Batch normalisation works as follows - During the training phase, data is normalised within a batch and this normalised output is then scaled and shifted via additional parameters which are also trained during the back-propogation of the network. The reason scaling and shifting is allowed is so that if the network decides that a scaled normal distribution is a better explanation for the output it could adjust it accordingly. In fact, it is possible for the model to learn an identity mapping such that the normalised values are mapped back to its original value. Batch normalisation for convolutional networks are applied in the way such that elements belonging to the same depth of the network are normalised in the same way.[21] This process for the case of a convolutional network is highlighted in Equation 5.5. Note that this process is done for every depth of the input where the scaling and shifting parameters γ, β learned are shared among every pixel in that depth. Also, $X^{(d)}_{m,p,q}$ refers to the value of a pixel belonging to an input of batch m, height p, width q and depth d and $y^{(d)}_{m,p,q}$ is the output of the batch normalisation

process.

$$\begin{aligned}
\mu_{\mathcal{B}}^{(d)} &= \frac{1}{MPQ}\sum_{m=1}^{M}\sum_{p=1}^{P}\sum_{q=1}^{Q}X_{m,p,q}^{(d)} \\
\sigma_{\mathcal{B}}^{2\,(d)} &= \frac{1}{MPQ}\sum_{m=1}^{M}\sum_{p=1}^{P}\sum_{q=1}^{Q}(X_{m,p,q}^{(d)}-\mu_{\mathcal{B}}^{(d)})^2 \\
\hat{X}_{m,p,q}^{(d)} &= \frac{X_{m,p,q}^{(d)}-\mu_{\mathcal{B}}^{(d)}}{\sqrt{\sigma_{\mathcal{B}}^{2\,(d)}+\epsilon}} \\
y_{m,p,q}^{(d)} &= \gamma^{(d)}\hat{X}_{m,p,q}^{(d)}+\beta^{(d)} \text{ for all } d
\end{aligned} \tag{5.5}$$

During evaluation, batch mean and variance are not calculated from the test batch. This is because the test batch might be potentially small and contain a lot of noise. Therefore, batch normalisation utilises a running mean and running variance calculated during the training phase to do this normalisation instead.

Next, the warm-up term works by starting the weight of the KL-divergence term, β from 0 and linearly increase it to 1 as the training progresses as seen in Equation 5.6. This prevents the collapse of the latent units by allows the VAE to focus first on reducing the reconstruction error to obtain a useful representation before trying to regularise the latent space into a normal distribution.[43]

$$Loss = -\beta KL(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}_i)\,||\,p_{\boldsymbol{\theta}}(\mathbf{z})) + BCELoss(\mathbf{x},\mathbf{x'}) \tag{5.6}$$

We tested the implementation of both methods (i.e. WU - warmup term, BN - batch normalisation) to our VAE structure. Since it is known that batch normalisation acts as a form of regularisation, to assess this we altered the experimental setup by training these various structures on the deer dataset, while testing was done on the full diamond-walk dataset. For the warm-up term, we utilised a linearly increasing β term from 0 to 1 up to the 500 epoch, where it continues to remain at 1 for the remaining epochs. For batch normalisation, similar to SegNet, we applied it to all layers except the output layer. We increased the learning rate from 1e-4 to 1e-3 and the batch size from 20 to 100 as higher values of these hyperparameters tend to work better with batch-normalisation[4]. As we can see from the results in the precision-recall curves in Figure 5.15, each of these enhancements further increases the model's effectiveness with batch normalisation having a much more significant increase.

Analysing the post-PCA latent space in Figure 5.16, we can see how each enhancement loosens the regularisation effect of the prior multivariate normal distribution applied to the latent space, but still reducing the gaps between the various latent representations of samples, compared to the normal autoencoder. It is by balancing the effect of regularisation while still retaining distinguishability among various points that explains this ultimate increase in performance.

5.1.8 Adding Homography Deformations to Inputs

One of the main problems not solved by binary edge images as explained in Section 2.2.1 is the fact that they are not viewpoint invariant. While pooling serves to solve

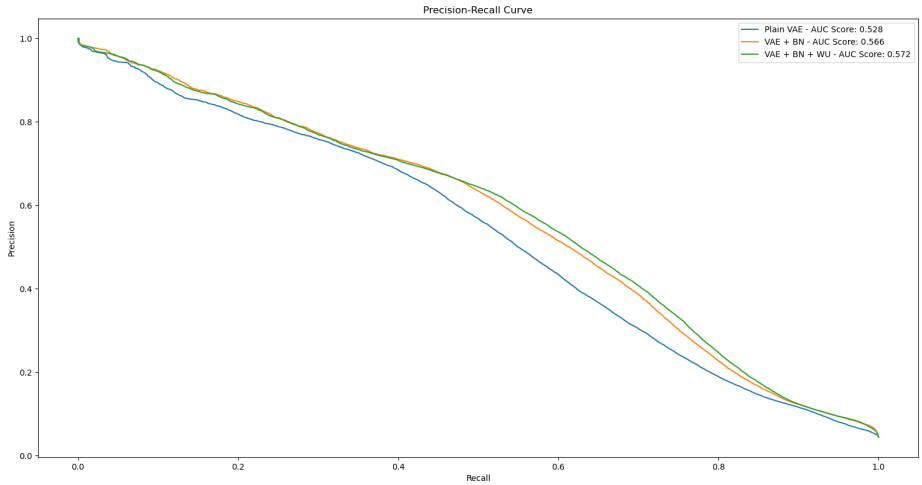

Figure 5.15: Precision-Recall comparison of normal autoencoder, variational autoencoder, variational autoencoder (BN) and variational autoencoder (BN + WU).

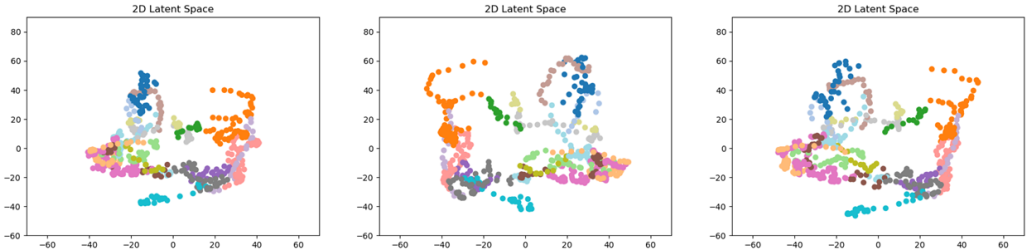

Figure 5.16: Post-PCA 2D Latent Space comparison of left: variational autoencoder, middle: variational autoencoder (BN) and right: variational autoencoder (BN + WU).

this problem, it only does it to a limited extent. Therefore, we adopt the same technique used by Merrill et al.[30] where we apply random 2D projective deformations on our input image and allow the autoencoder to reconstruct the original image. This resembles the denoising autoencoder explained in Section 2.4.4 except that the noise used in this case are random geometrical transformations of the original image.

Doing projective transformations involves tabulating a homographic matrix that transforms an image in the original geometrical plane to match the image in the new geometrical plane. This is shown in Equation 5.7 where a point $\mathbf{x}$ is mapped to another point $\mathbf{y}$ via the homography matrix $\mathbf{P}$ where s is a scaling factor. Such projective transformations are known to be realistic models for actual viewpoint changes.[30]

$$s \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{5.7}$$

Our application of this algorithm is similar to that of Merill et al., except that we apply a different random projective deformation to the images at every epoch, rather than pre-computing these deformations. This is to reduce any systematic bias that might be created within the process. The algorithm proceeds as seen in the pseudocode outlined in Algorithm 2 and an example of the 2D-transformation applied on a binary edge map is shown in Figure 5.17.

Algorithm 2 Pseudocode: Random Projective Deformations

```
procedure ProjDeformation(I)          ▷ I: Image, P: Projective Matrix
    if RandomUniform[0, 1] >= 0.5 then          ▷ Random Coin Flip
        return I                                ▷ No change
    else
        Create 4 sections for I spanning 1/4 of the image at the corners of I
        Pick a random point within each section for a total of 4 points
        Tabulate P using points and the corner coordinates of I
        Transform I using P and scale to the same size as I
        return I
```

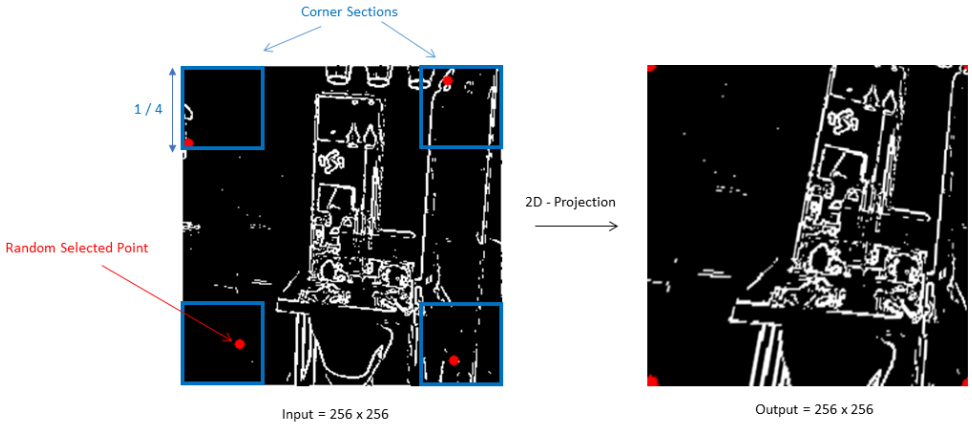

Figure 5.17: Example 2D Projective Transformation of Edge Maps.

We plot the precision-recall curves for models with and without projective transformations in Figure 5.18. Due to the timeline of when this was implemented, the models used were based on a more updated structure that was trained on the entire deer-walk data set and applied on the diamond-walk data set as defined in Section 6.1, the only difference between both models being the presence of such random projective deformations during the training phase. Edge thresholding was also reduced to 200 as this provided better performance for both models since we are also considering similarities near the main diagonal in this case. As we can see, while the initial precision remains the same, there is subsequently a huge decrease in the precision-recall trade-off for the case without random deformations. This is likely cause we are able to identify more image pairs with viewpoint changes through the random deformation method.

Figure 5.19 shows a pair of images and its corresponding binary edge images that

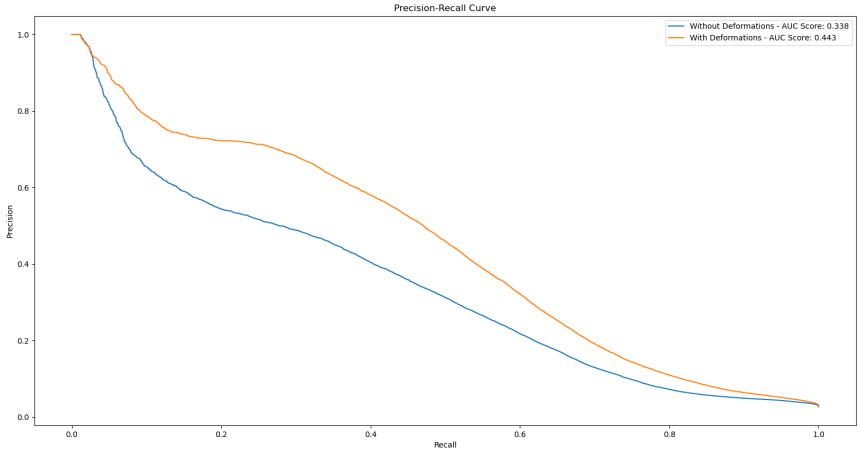

Figure 5.18: Precision-Recall Curve: Presence of Random Deformations.

has a very slight viewpoint change (i.e. the stationary bicycle at the left side has been removed from the scene due to the rotation). As we can see, even slight viewpoint changes was able to degrade the similarity score by quite a large degree in the absence of such random deformations. This is likely because of the sparse and extreme nature of binary edge maps which have little tolerance for rotational changes when doing pixel wise comparison in the latent space. This in turn highlights the importance for us to use such random deformations during the training phase.

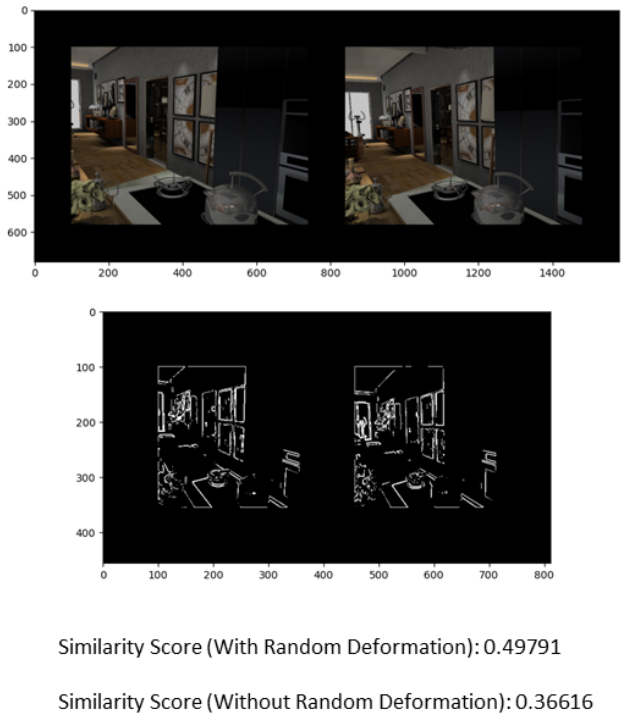

Figure 5.19: Example similarity score of a pair of images with slight viewpoint variation tabulated on models with and without random deformation.

5.2 Determining similarity

Here we outline the methodology for the second phase of our loop closure detection, where we try to determine if an actual loop closure has occurred using the latent feature vector we have obtained from the previous step. Since this phase of the methodology largely involves the elimination of false positive loop closures, testing within this portion is only done for off-diagonal loop closures i.e. frame-lapse of 100 on the full diamond-walk data set. Similar to before, training was done on the deer-walk data set. See Section 4.2.1 on details of frame-lapse.

5.2.1 Distance metric

In order to determine the similarity between 2 latent vectors, we have to select a metric which will allow us to estimate the distance between both these vectors. One popular metric used is to use the euclidean distance which is shown in Equation 5.8 where m is the number of latent variables within each vector. Another popular distance metric is the cosine distance metric as seen in Equation 4.1.

$$\begin{aligned} EuclideanDistance(\mathbf{x}_1, \mathbf{x}_2) &= \sqrt{\sum_{i=1}^{m}(x_1^{(m)} - x_2^{(m)})^2} \\ &= ||\mathbf{x}_1 - \mathbf{x}_2|| \end{aligned} \tag{5.8}$$

A pictorial representation of both these distances within the 2D space can be seen in Figure 5.20. The key difference is while euclidean distance measures the distance between two points, cosine distance measures the angle between the two vectors. Therefore, cosine distance is invariant to the magnitude of these vectors and only considers the orientation difference between both of these vectors.

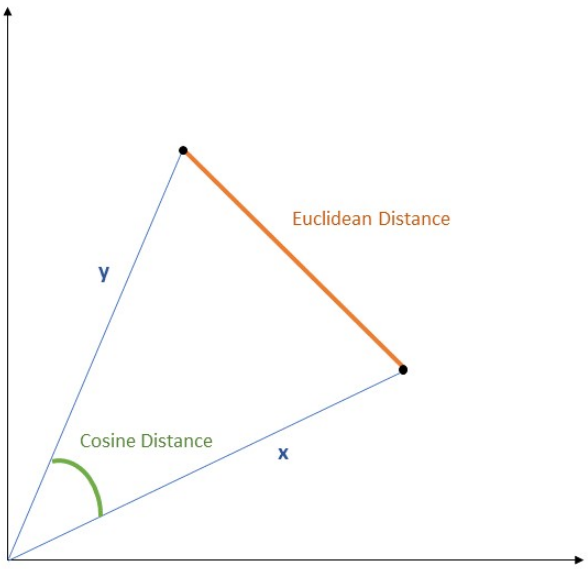

Figure 5.20: Cosine versus euclidean distance.

The similarity metric used in the unsupervised training of Gao et al.[14] is a weighted version of the euclidean distance metric. The weight for this metric is tabulated as follows. Firstly,the mean of all latent variables is taabulated across the test samples as in Equation 5.11, where m is the number of variables and n is the number of test samples.

$$x^{(m)} = \frac{1}{n}\sum_{i=1}^{n} x_i^{(m)} \tag{5.9}$$

This is then fitted to a gaussian function to create the weights as seen in Equation 5.10.

$$\begin{gathered}\delta^{(m)} = exp\left(-\frac{(x^{(m)} - \mu^{(m)})^2}{2\sigma^{(m)^2}}\right) \text{ where} \\ \mu^{(m)} = \frac{1}{n}\sum_{i=1}^{n} x_i^{(m)} \text{ and } \sigma^{(m)} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i^{(m)} - \mu^{(m)})^2}\end{gathered} \tag{5.10}$$

This can then be applied directly to the Euclidean Distance where $\boldsymbol{\delta} = \begin{bmatrix}\delta^{(1)} & \dots & \delta^{(m)}\end{bmatrix}^T$ and $\mathbf{x}_i = \begin{bmatrix}x_i^{(1)} & \dots & x_i^{(m)}\end{bmatrix}^T$

$$WeightedEuclideanDistance(\mathbf{x}_1, \mathbf{x}_2) = ||\boldsymbol{\delta}^T(\mathbf{x}_1 - \mathbf{x}_2)|| \tag{5.11}$$

The intuition behind using this weight vector is that we should value activations in the middle range more than those at the extreme ends. This is because activation units which have high response across samples are usually indicative of common features among the data set, such as edges of foilage or walls which are likely to lead to false positives whereas those with low response are usually a sign of noise.[14]

Since we are interested in defining a similarity index between images rather than defining image difference using distances, we utilise functions which plots an inverse relationship with distances in Equation 5.12, where both the euclidean and weighted euclidean uses the same function. Similar to Gao et al.[14], we use a logarithm function to cushion the impact of large euclidean distances. Cosine distances on the other hand are constrained to be between [-1, 1], therefore a simple subtraction as its similarity function will suffice.

$$\begin{gathered}CosineSimilarity = 1 - CosineDistance(\mathbf{x}_1, \mathbf{x}_2) \\ EuclideanSimilarity = 10 - \log(EuclideanDistance(\mathbf{x}_1, \mathbf{x}_2) + 1)\end{gathered} \tag{5.12}$$

These values are then re-scaled into the range of [0, 1] via the subtraction of the minimum value, followed by a division of the largest resulting value. Finally, metrics are plotted into a similarity matrix S where $S(i, j)$ represents the similarity score between images i and j. The similarity matrix is structured to be the same way as the truth matrix explained in Section 4.1.2 for easy comparison of results. Precision-recall curves can be plotted accordingly by comparing this matrix with the truth matrix as outlined in Section 4.2.1.

To decide on the appropriate metric for our use case, we evaluated each of these 3 metrics by plotting their similarity matrix using the VAE + BN + WU model

defined within Section 5.1.7 and compared it with the truth matrix as in Figure 5.21. We then evaluated their performance using the precision-recall curve as seen in Figure 5.22.

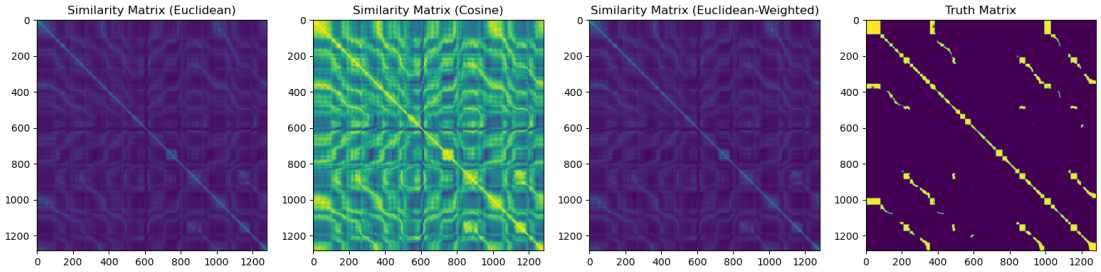

Figure 5.21: Example of plotted similarity matrices against truth matrix.

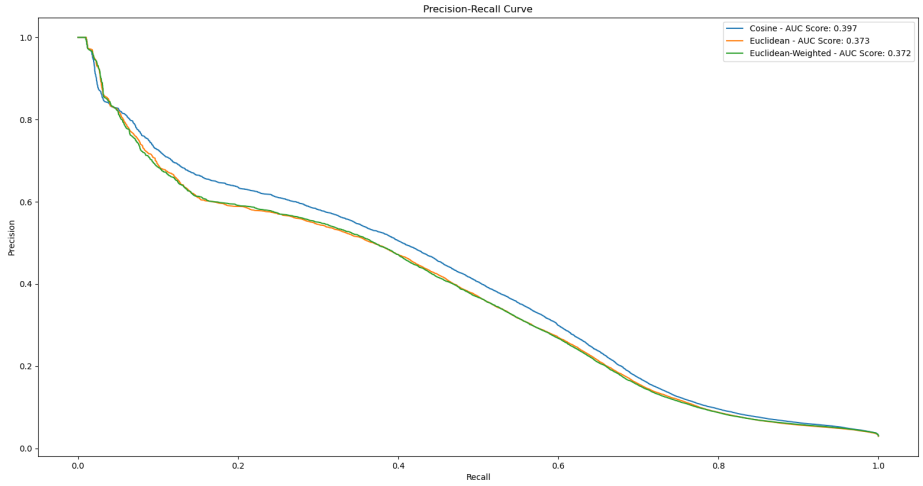

Figure 5.22: Precision-Recall Curve: Similarity metric comparison.

Surprisingly, both weighted euclidean and euclidean performed similarly. This similar performance is largely attributed to the regularisation of the latent space by the VAE and batch normalisation, which constrains the activations across the latent vector to be largely similar. This can be seen in Figure 5.23, where we picked 5 random sample latent vectors and plotted their values against their index. As a result of there having little to no particular regions of high or low activations, the weight vector does minimal change to the euclidean metric. Coupled with the fact that such a weighted metric can be more unstable in small test sizes, we decided to go with the normal euclidean metric instead.

On the other hand, we choose the euclidean metric over the cosine metric, despite the cosine metric performing better because such a metric works better with the rank reduction technique. This will be outlined in Section 5.2.2.

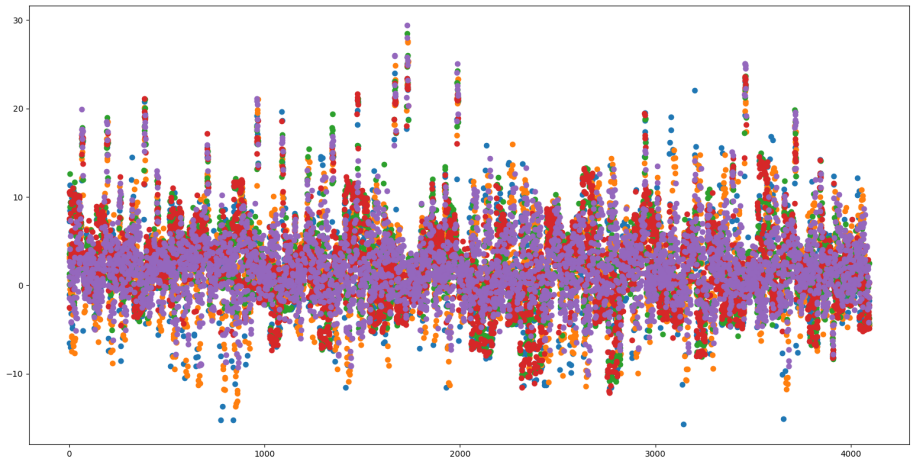

Figure 5.23: Sampled latent vector for VAE structure.

5.2.2 Rank Reduction

As mentioned previously, one of the key issues we are combating when doing loop closure is the problem of perceptual aliasing, where two different scenes may look similar (i.e. false positives). Therefore, we use the technique of rank reduction adopted by Ho et al.[17] to reduce the occurrence of such issues.

Rank reduction is based on the concept of spectral decomposition, where we can express a square matrix M in terms of its eigenvalues and eigenvectors as seen in Equation 5.13. V is a matrix whose columns $\mathbf{v}_i$ are eigenvectors and D is a diagonal matrix whose diagonals are the corresponding eigenvalues, λ_i, where $\lambda_1 > \lambda_2 > \ldots > \lambda_n$. The second equation holds when M is a real symmetric matrix, which is the case for our similarity matrix.

$$\begin{aligned} S &= VDV^{-1} \\ &= VDV^{T} (\text{since M is real and symmetric}) \\ &= \begin{bmatrix} \mathbf{v}_1 & \ldots & \mathbf{v}_n \end{bmatrix} \begin{bmatrix} \lambda_1 & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & \lambda_n \end{bmatrix} \begin{bmatrix} \mathbf{v}_1^T \\ \vdots \\ \mathbf{v}_n^T \end{bmatrix} \\ &= \sum_{i=1}^{n} \mathbf{v}_i \lambda_i \mathbf{v}_i^T \end{aligned} \tag{5.13}$$

Rank reduction works by assuming that each dyad $\mathbf{v}_i \lambda_i \mathbf{v}_i^T$ captures a theme within the images belonging in the test set. The higher the associated eigenvalue of the dyad, the more prominent this theme is among the images. Such dominant themes more often that not are a cause of the perceptual aliasing problem. For example, while moving through a building, a robot may capture a lot of scenes with brick walls which they may associate with each other although they may not be actual loop closures. By removing such dominant scenes, we hope to ultimately remove

the number of false positive loop closures. This is done through eliminating the first k dyads with the largest eigenvalues of the similarity matrix S to obtain the rank reduced matrix R as seen in Equation 5.14. After rank reduction, off-diagonal images were re-scaled into the range of [0, 1]. This will allow the rank reduction matrix to be directly compared with the truth matrix when plotting the precision-recall curve as highlighted in Section 4.2.1.

$$S = \sum_{i=1}^{n} \mathbf{v}_i \lambda_i \mathbf{v}_i^T \text{ where } \lambda_1 > \lambda_2 > \ldots > \lambda_n$$
$$R = \sum_{i=k}^{n} \mathbf{v}_i \lambda_i \mathbf{v}_i^T \tag{5.14}$$

We test the effect of rank reduction on edge maps using the New College data set.[9] This is because such perceptual aliasing occurs much less frequently in the deer-walk data sets and diamond-walk data sets which are of much smaller scale. New College is a data set containing non-simulated real life images captured by a robot traversing through New College in Oxford. More details on testing on this data set will be covered in Section 6.2. We use the same model used as before in Section 5.2. Figure 5.24 shows the top edge maps associated with the top 3 dyads alongside their RGB images, (i.e. the image pairs with the largest change from removing that dyad). As we can see, the first dyad contains image pairs which contain brick walls but are taken at different locations. Since the pattern of edges of walls are quite similar, the algorithm ended up matching them. These are exactly the kind of perceptual aliasing we aim to overcome using rank reduction. Similarly, Figure 5.25 shows the corresponding rank reduced matrices with varying degrees of rank reduction applied. As we can, see the off-diagonal loop closure lines become more defined with higher degrees of rank reduction.

However, dyads 2 and 3 in Figure 5.24 also highlights the shortcoming of rank reduction where we in the process of removing dominant features also end up removing true positive loop closures. This corresponds to the rank reduced images as well where we begin to lose certain off-diagonal loop closures as the rank reduction proceeds. While it is more important to eliminate false loop closures in the case of loop closure detection, we want to simultaneously ensure that rank reduction does not reduce them to the point that key features in the remaining images are removed. This is even more important in our case of binary edge images which in and of itself already contains a relatively small amount of information. Therefore, unlike Ho et al.[17] who used a general heuristic to choose the rank reduction value, we enable the user to choose the value of rank reduction instead as the ideal degree has shown to generally vary for different environments as in the case of our multiple experiments.

The above is also the reason we choose to use euclidean over cosine distances. Figure 5.26 shows the Precision-Recall Curve of the cosine and euclidean distance, with and without a rank reduction of degree 1. This graph shows how the cosine metric interacts poorly with the rank reduction mechanism, with the precision recall curve dropping drastically after only a single reduction. The euclidean curve on the other hand performs comparably better with only a slight dip.

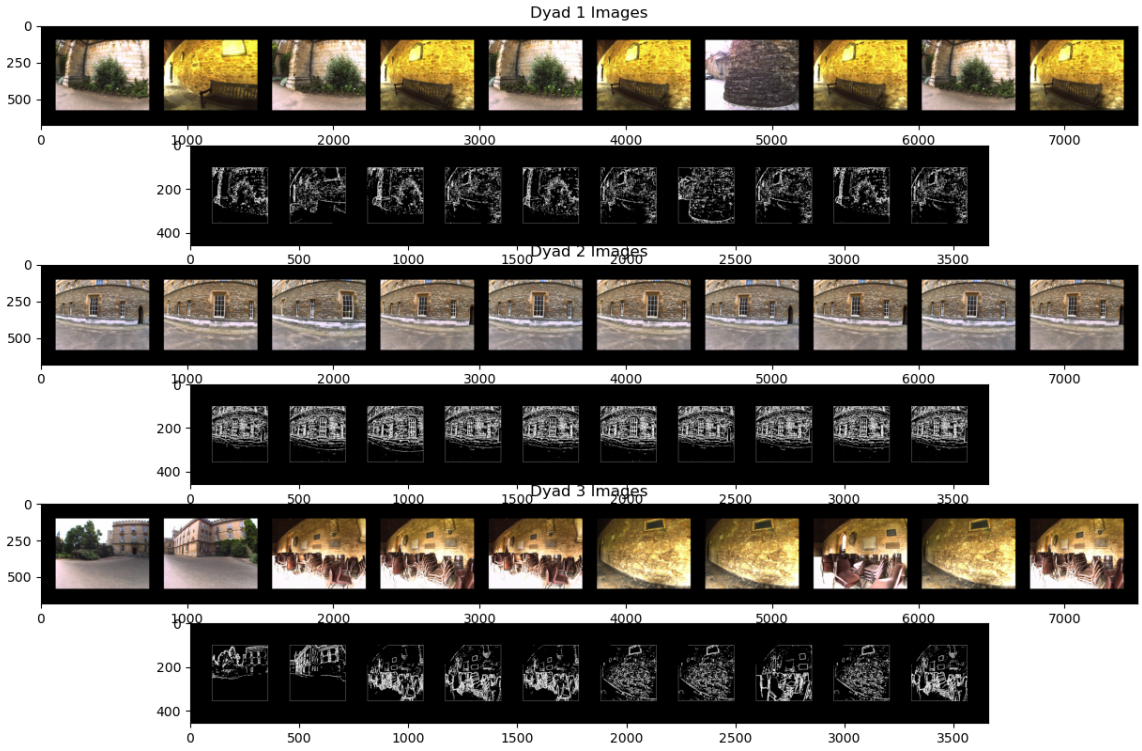

Figure 5.24: Top image pairs associated with largest 3 dyads for New College data set. Pairs are arranged horizontally within the row.

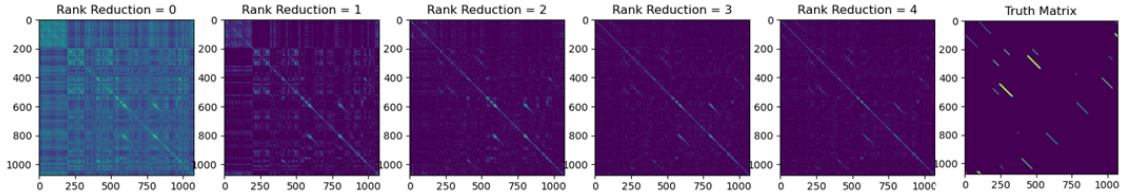

Figure 5.25: Rank Reduced Matrices with varying levels of rank reduction applied compared with truth matrix.

This can be explained via the plotting of the eigenvalues for the dyads in the cosine case and the euclidean case as seen in Figure 5.27. As seen from the graphs, the first dyad of the cosine metric similarity matrix has a significantly higher eigenvalue with respect to the rest of its eigenvalues. In fact, the mean of all eigenvalues in both the case of euclidean and cosine is $\sim$1, but the first eigenvalue of the first dyad for cosine is more than 3 times the value of the eigenvalue of the first euclidean dyad. Therefore, since the first dyad contains majority on the information of the images within the dataset, removing it might have resulted in the elimination of the key image features as well, which severely degraded the model.

Another thing that causes the divergence in eigenvalue distribution is the size of

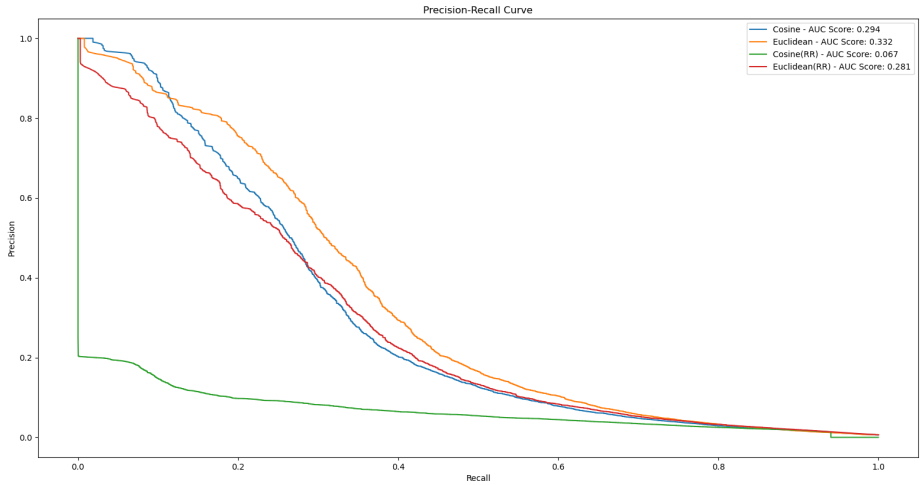

Figure 5.26: Precision Recall Curve: Cosine vs Euclidean (with and without rank reduction).

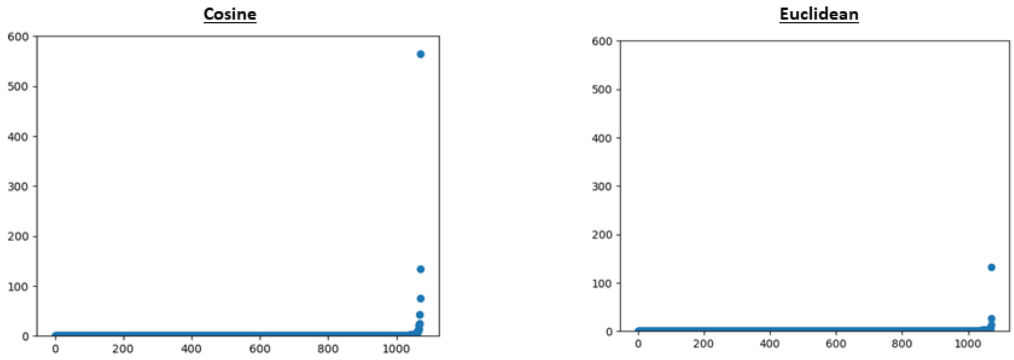

Figure 5.27: Scatterplot for eigenvalues of dyads: cosine (Left), euclidean (Right).

the VAE latent dimension. This is because autoencoders behave very similarly to PCA, with the only difference being that it models non-linear relationships. Therefore since there is still going to be some degree of linear relationship that underlies this non-linearity, a tighter latent bottleneck is going to cause divergence in the eigenvalues of the preceding larger layers by creating a representation that contains a few large eigenvalues among many other smaller ones. Since the outputs we are using for comparison are not directly from the smallest latent dimension, in order to balance the eigenvalue distribution, we experimented in increasing the size of the latent dimension. The caveat to this approach is that too large a latent dimension will create a case of over-fitting where each value can be represented by a unique latent representation. The eventual latent space chosen was a size of 128 - since experimenting with a larger latent space of size 256 results in a 10% decrease in AUC for the non-rank reduced euclidean curve, indicating that overfitting has occurred

and experimenting with a smaller latent space of 64 produced very similar results. To elucidate this, we compare the new model with a latent space of 128 with the original model of latent size 8 and display their results in Figure 5.28 below. As we can see, with a rank reduction of degree 1, the model with a latent space of size 8 suffered a larger decrease in AUC compared to the one of 256. This is once again attributed to the more even eigenvalues of the larger latent space as seen in Figure 5.29.

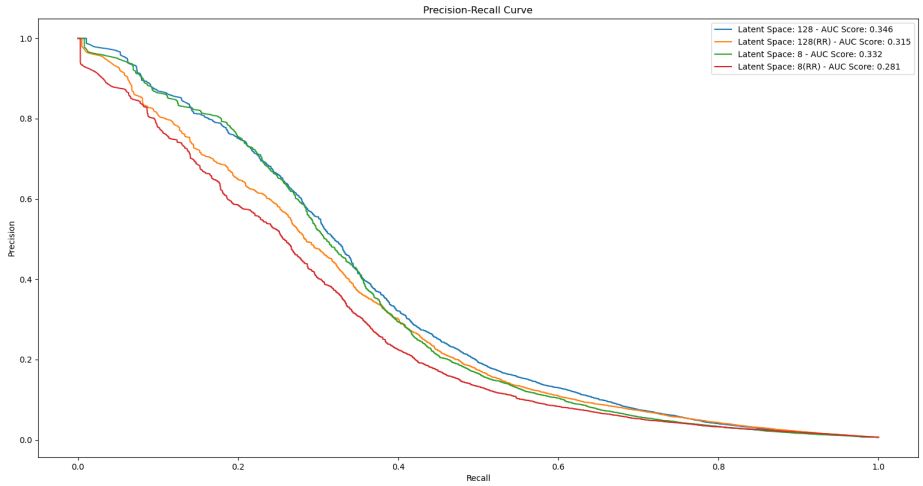

Figure 5.28: Precision Recall Curve: Latent space 8 versus 128 with and without rank reduction of degree 1.

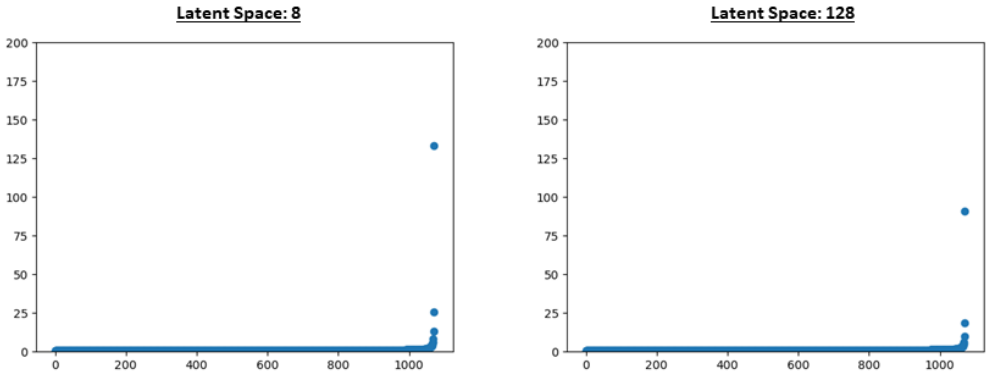

Figure 5.29: Scatterplot for eigenvalues of dyads: latent space of 8 (Left), latent space of 128 (Right).

Lastly, we optimise the Euclidean model with a latent space of 128 through applying varying degrees of rank reduction and highlight the results in Figure 5.30. As we can see, there is a slight increase in precision before the model faces a sharp

drop while trying to increase recall. This is once again because the number of true positives removed by rank reduction is also significant. It is important to note that although overall, the model might have removed more true positives than false positives, such a tradeoff is still considered ideal in the case of loop closure detection. Also, rank reduction is key within the next step of the matching algorithm highlighted in Section 5.2.3 as such an algorithm benefits greatly from operating on a less noisy matrix.

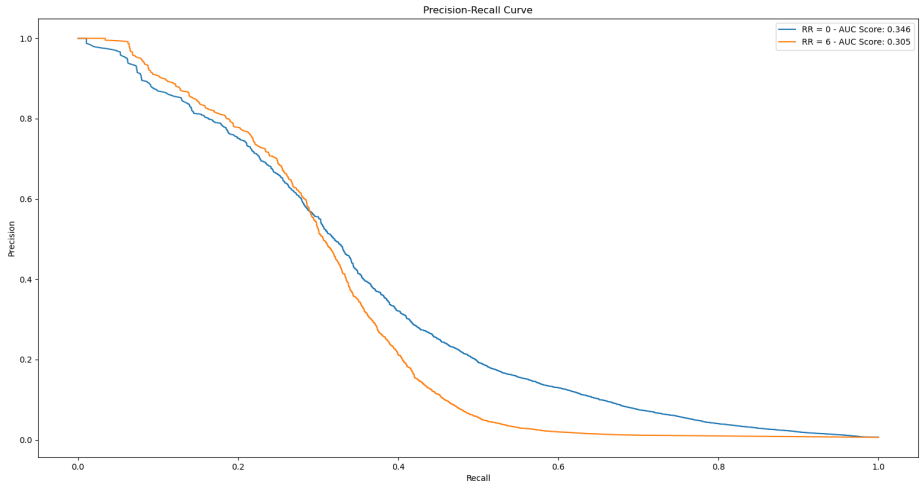

Figure 5.30: Precision Recall Curve: Optimised rank reduction of 6 versus non-rank reduction

5.2.3 Adjusted Smith-Waterman Algorithm

So far, we have only considered matching of one to one matches. However, when images are collected from the camera of a robot, they are inputted in time-sequential order. Therefore, it is better to match a sequence of images over a short time-frame rather than single images and that will drastically reduce the chance of a false loop closure. For this we use an adjusted Smith-Waterman algorithm implemented by Ho et al.[17] in their loop closure sequence detection algorithm. This is a dynamic programming technique that allows us to tabulate the scene sequence within a fixed time-frame that provides the sequence with the highest possible similarity response. Rather than finding just a single highest response loop closure sequence as in the work done by Ho et al.[17], we will adjust the algorithm to tabulate multiple loop closure sequences instead.

The adjusted Smith-Waterman algorithm involves using the values from the previously rank-reduced similarity matrix which we will denote by R, where $R(i, j)$ where i is the row number and j the column number i and j of the similarity matrix. Since R is symmetric, we do not need to consider the main diagonal and every other value above it which we set to 0. Also, since we are not interested in images with a small frame-lapse, we can remove a series of diagonals below the main diago-

nal as well (See Section 4.2.1 for details on frame-lapse). This will give us a matrix with values only at the bottom left triangular corner as seen from the yellow cells in Figure 4.9, which we will refer to as $\hat{R}$.

The adjusted Smith-Waterman algorithm involves tabulating another matrix which we will denote as S. This matrix will tabulate the cumulative similarity across the matrix $\hat{R}$. Starting from the top left corner and moving in a row first, column next fashion we update the matrix $S(i,j)$ using the cases listed in Equation 5.15, where maximal means that it is the biggest of $S(i-1,j-1), S(i-1,j), S(i,j-1)$.

$$S(i,j) = \begin{cases} S(i-1,j-1) + \hat{R}(i,j) & \text{if } S(i-1,j-1) \text{ is non-zero and maximal} \\ S(i,j-1) + \hat{R}(i,j) - \delta & \text{if } S(i,j-1) \text{ is non-zero and maximal} \\ S(i-1,j) + \hat{R}(i,j) - \delta & \text{if } S(i-1,j) \text{ is non-zero and maximal} \\ 0 & \text{if } \hat{R}(i,j) < \text{threshold similarity acceptance} \\ \hat{R}(i,j) & \text{if } S(i-1,j-1) = S(i-1,j) = S(i,j-1) = 0 \end{cases} \tag{5.15}$$

The intuition of the cases are as follows - firstly, we add a penalty term δ for horizontal and vertical frames because we are less interested in image pairs with one to many matching and more interested in one to one matching. Secondly, we reduce the values of S to zero if the value within the similarity matrix does not pass the individual image threshold. This is to indicate that the previous loop closure sequence has ended. The last equation is to restart the cumulative similarity of a new sequence should its adjacent frames not belong to a loop closure sequence.

Note that so far we have only considered sequences in matrix S moving towards the down and right direction, and have not considered sequences moving from the down and left direction i.e. robot traversing through the sequence in the opposite direction. Therefore, we have to invert the rows of matrix $\hat{R}$, i.e. swapping the first and last rows, the second and second last rows, etc. and repeat the process. We will also tabulate another set of sequences from this adjusted matrix.

The main difference within our use case is that it aims to find multiple loop closure sequences instead of the individual highest sequence. We do this by setting a second threshold on the minimum $S(i,j)$ value that we will accept as a loop closure sequence, we can check both of the 2 tabulated matrix and collate all the sequences that passes this minimum which we will deem as loop closures, rather than just taking the highest scoring cell in S. Like the original adjusted Smith-Waterman algorithm, we just find all relevant matches by back-tracking to find the contributing cells to these cumulative scores that passes the threshold. Using this method will essentially help us remove false positives as rather than just checking for the correspondence of single images, i.e. using the single image threshold similarity acceptance as shown in Equation 5.15, we add an additional threshold that requires a series of such single similar images to occur within a time-sequence for us to consider it as a true loop closure. The pseudocode of the full algorithm is summarised in Algorithm 3 below.

Algorithm 3 Pseudocode: Adjusted Smith-Waterman Algorithm

procedure AdjSmithWaterman(R, $seq\_threshold$)
  Convert R to $\hat{R}$ by isolating bottom-left triangle as seen in Figure 4.9
  S = Zero matrix with same dimensions as R
  $SmithMatrix$ = Zero matrix with same dimensions as R
  for each i in S **do** ▷ i: row number
    for each j in S **do** ▷ j: column number
      Iteratively adjust each $S(i, j)$ according to Equation 5.15
  $\mathbf{S(i, j)} = S(i, j) > seq\_threshold$ ▷ **S**: collection of coordinates in S
  for each $S(i, j)$ in $\mathbf{S(i, j)}$ **do**
    for each backtracked (i, j) in $S(i, j)$ **do**
      $SmithMatrix(i, j) = 1$
  $\hat{R}\_flip$ = Flip $\hat{R}$ vertically
  Repeat Step 4-11 for $\hat{R}\_flip$ flipping back row indexes when updating $SmithMatrix$ in Step 11
  return $SmithMatrix$

Figure 5.31 shows an example of applying this adjusted Smith-Waterman Algorithm to the rank reduced matrix obtained from the previous section. As we can see, the algorithm extracts continuous image sequences and those that pass the threshold are shown as the off diagonal lines in the resulting matrix.

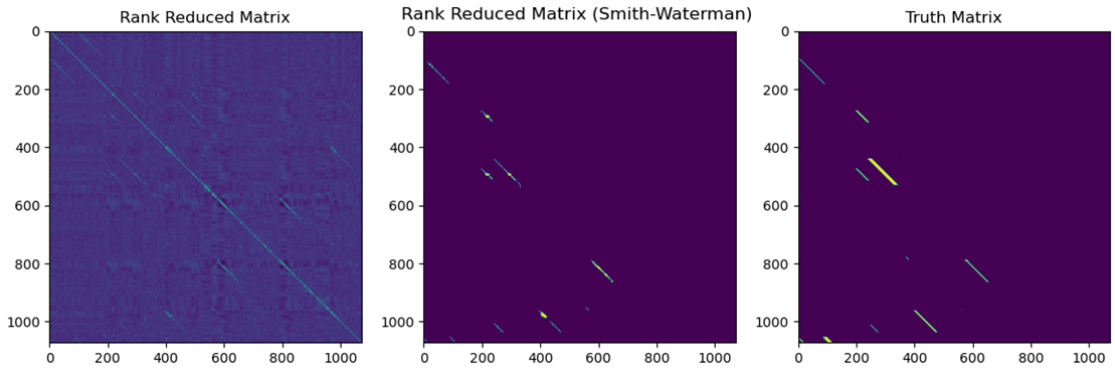

Figure 5.31: Rank Reduced Matrix before and after Smith-Waterman Algorithm.

Lastly, since the Smith-Waterman Matrix just contains values of 1 and 0 with 1 being an identified Smith-Waterman loop closure and 0 being one that is not, for the final step, we can just apply an element-wise multiplication of this matrix with either the rank reduced matrix or the similarity matrix to obtain the final prediction, whichever is the better performing matrix. This will result in the final prediction matrix having similar probability values to either the rank reduced or similarity matrix in locations where the Smith-Waterman Algorithm identified to be positive loops and 0 in the locations where the algorithm determines to be a negative loop, essentially acting as a false positive removal step. Since the values of the post Smith-Waterman matrix are still contained within [0, 1], we can compare it directly with the truth matrix as outlined in Section 4.2.1. One might question why there might

still be a need for rank reduction if it will degrade performance. The main reason is the difficulty of the application of Smith-Waterman on a non-rank reduced matrix. As shown in Figure 5.25, a non-rank reduced matrix has many square like patterns caused by the presence of false positive images and immediately adjacent scenes and this will cause the resulting loop closure sequences identified by the algorithm to occur in clumps of squares rather than neat looking diagonal lines. Therefore, the step of rank reduction is an essential step before the Smith-Waterman algorithm, even if it might degrade performance. We plot a precision recall-curve highlighting the performance at different stages as seen in Figure 5.32. As we can see, the true positive rate is much higher for the post Smith-Waterman algorithm.

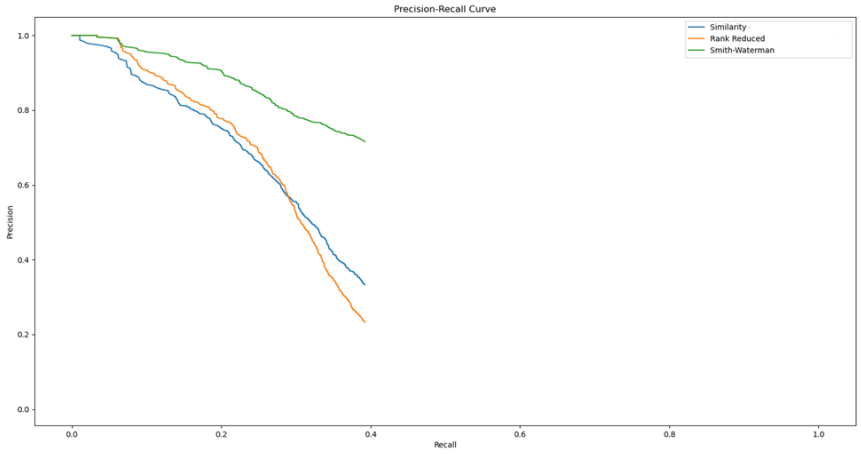

Figure 5.32: Precision Recall Curve: Similarity Matrix, Rank Reduced and Post Smith-Waterman final prediction.

Notice the truncation of the precision-recall curve above. Due to the fact that the Smith-Waterman Algorithm sets the probability of detected invalid loop closure values to 0, the algorithm will naturally hit a limit in terms of its recall performance (i.e. 0.4 in this case) if it does not identify every single positive truth label. This is because adjusting the threshold beyond that point will not allow the method to find any more positive loop closures since all remaining prediction probability values are 0. As a result, the precision-recall curve is truncated and the AUC of the post Smith-Waterman precision-recall curve cannot be directly compared. Instead, we will choose to pay closer attention to the steepness of the precision-recall values before this truncation and the maximum recall the algorithm reaches. As such, many of the Smith-Waterman result curves plotted in 6 will be truncated accordingly.

5.3 Summary of Methodology

5.3.1 Summary of the autoencoder structure

We summarise our final autoencoder structure here in Tables 5.1 and 5.2. For gradient descent, an ADAM optimiser was used.

ENCODER		
Layer	**Input Size**	**Output Size**
CONV11: 8 x 3 x 3, padding = 1 BatchNorm + ELU	1 x 256 x 256(**INPUT**)	8 x 256 x 256
CONV12: 8 x 3 x 3, padding = 1 BatchNorm + ELU	8 x 256 x 256	8 x 256 x 256
MAXPOOL1: 2 x 2 → switch 1	8 x 128 x 128	8 x 128 x 128
CONV21: 16 x 3 x 3, padding = 1 BatchNorm + ELU	8 x 128 x 128	16 x 128 x 128
CONV22: 16 x 3 x 3, padding = 1 BatchNorm + ELU	16 x 128 x 128	16 x 128 x 128
MAXPOOL2: 2 x 2 → switch 2	16 x 128 x 128	16 x 64 x 64
CONV31: 32 x 3 x 3, padding = 1 BatchNorm + ELU	16 x 64 x 64	32 x 64 x 64
CONV32: 32 x 3 x 3, padding = 1 BatchNorm + ELU	32 x 64 x 64	32 x 64 x 64
CONV33: 32 x 3 x 3, padding = 1 BatchNorm + ELU	32 x 64 x 64	32 x 64 x 64
MAXPOOL3: 2 x 2 → switch 3	32 x 64 x 64	32 x 32 x 32
CONV41: 64 x 3 x 3, padding = 1 BatchNorm + ELU	32 x 32 x 32	64 x 32 x 32
CONV42: 64 x 3 x 3, padding = 1 BatchNorm + ELU	64 x 32 x 32	64 x 32 x 32
CONV43: 64 x 3 x 3, padding = 1 BatchNorm + ELU	64 x 32 x 32	64 x 32 x 32
MAXPOOL4: 2 x 2 → switch 4	64 x 32 x 32	64 x 16 x 16
CONV51: 64 x 3 x 3, padding = 1 BatchNorm + ELU	64 x 16 x 16	64 x 16 x 16
CONV52: 64 x 3 x 3, padding = 1 BatchNorm + ELU	64 x 16 x 16	64 x 16 x 16
CONV53: 64 x 3 x 3, padding = 1 BatchNorm + ELU	64 x 16 x 16	64 x 16 x 16
MAXPOOL5: 2 x 2 → switch 5	64 x 16 x 16	64 x 8 x 8 (**LATENT**)
CONV61: 16 x 1 x 1 BatchNorm + ELU	64 x 8 x 8	16 x 8 x 8 → 4096
FC1: 1024-128 (μ)	4096	128 (to KL-Loss)
FC2: 1024-128 (σ^2)	4096	128 (to KL-Loss)
Reparameterise(μ, σ^2)	128, 128	128

Table 5.1: Structure of Encoder

DECODER		
Layer	**Input Size**	**Output Size**
FC3: 128-1024	128	1024
FC4: 1024-4096	1024	4096 $\rightarrow$ 64 x 8 x 8
MAXUNPOOL1: 2 x 2 $\leftarrow$ switch 5	64 x 8 x 8	64 x 16 x 16
CONV71: 64 x 3 x 3, padding = 1 BatchNorm + ELU	64 x 16 x 16	64 x 16 x 16
CONV72: 64 x 3 x 3, padding = 1 BatchNorm + ELU	64 x 16 x 16	64 x 16 x 16
CONV73: 64 x 3 x 3, padding = 1 BatchNorm + ELU	64 x 16 x 16	64 x 16 x 16
MAXUNPOOL2: 2 x 2 $\leftarrow$ switch 4	64 x 16 x 16	64 x 32 x 32
CONV81: 64 x 3 x 3, padding = 1 BatchNorm + ELU	64 x 32 x 32	64 x 32 x 32
CONV82: 64 x 3 x 3, padding = 1 BatchNorm + ELU	64 x 32 x 32	64 x 32 x 32
CONV83: 32 x 3 x 3, padding = 1 BatchNorm + ELU	64 x 32 x 32	32 x 32 x 32
MAXUNPOOL3: 2 x 2 $\leftarrow$ switch 3	32 x 32 x 32	32 x 64 x 64
CONV91: 32 x 3 x 3, padding = 1 BatchNorm + ELU	32 x 64 x 64	32 x 64 x 64
CONV92: 32 x 3 x 3, padding = 1 BatchNorm + ELU	32 x 64 x 64	32 x 64 x 64
CONV93: 16 x 3 x 3, padding = 1 BatchNorm + ELU	32 x 64 x 64	16 x 64 x 64
MAXUNPOOL4: 2 x 2 $\leftarrow$ switch 2	16 x 64 x 64	16 x 128 x 128
CONV101: 16 x 3 x 3, padding = 1 BatchNorm + ELU	16 x 128 x 128	16 x 128 x 128
CONV102: 8 x 3 x 3, padding = 1 BatchNorm + ELU	16 x 128 x 128	8 x 128 x 128
MAXUNPOOL5: 2 x 2 $\leftarrow$ switch 1	8 x 128 x 128	8 x 256 x 256
CONV111: 8 x 3 x 3, padding = 1 BatchNorm + ELU	8 x 256 x 256	8 x 256 x 256
CONV112: 1 x 3 x 3, padding = 1	8 x 256 x 256	1 x 256 x 256 (**RECON**)
Loss = BCELoss(**RECON**, **INPUT**) - βKL_div(μ, σ^2)		

Table 5.2: Structure of Decoder

5.3.2 Summary of training and prediction methodology

Figure 5.33 provides a pictorial summary of the neural network training phase and Figure 5.34 provides the corresponding pictorial summary of the prediction phase. We note that the final batch normalisation and ELU layer is removed from the encoder during the prediction phase. Also, while the main focus is to compare the final resulting matrix with the truth matrix to plot the precision-recall curve, we also plot precision-recall curves with the rank reduction and similarity matrices with the truth matrix in order to assess the reliability of the algorithm at each stage of the process within Section 6.

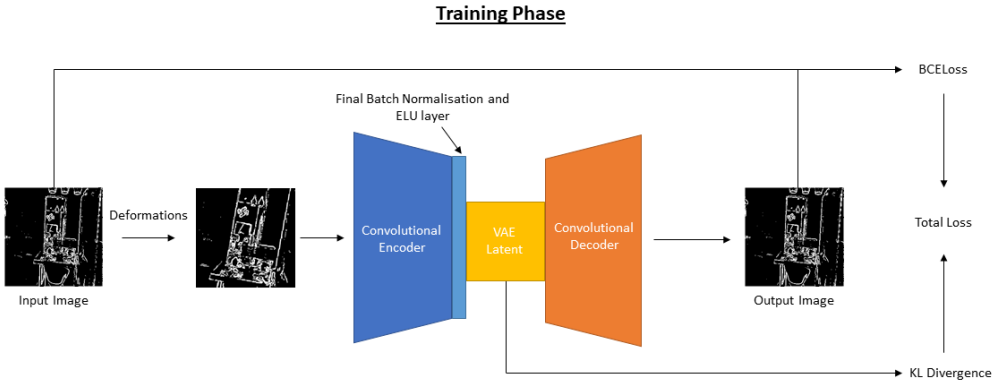

Figure 5.33: Overview of neural network training phase.

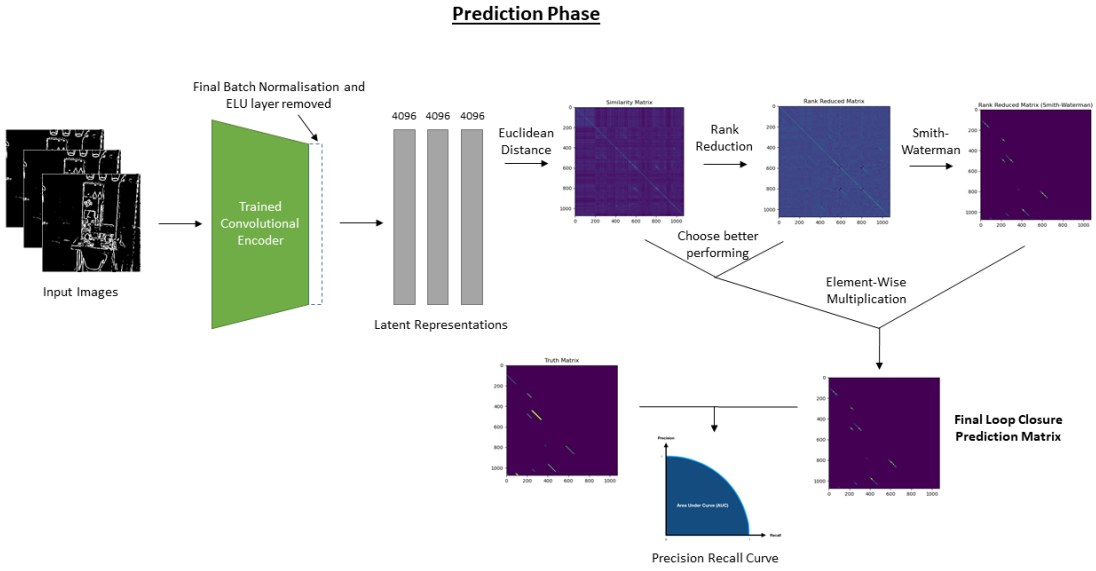

Figure 5.34: Overview of neural network prediction phase.

Chapter 6

Results and Evaluation

Here we highlight our results of testing the final framework on different data sets. For each data set, hyper-parameters were varied using a grid selection method and tested on a validation data set. The model was then retrained with both of these data sets combined before final testing on a blind test set. The varied hyper-parameters included: Batch Size, Epochs, Learning Rate, Weight Decay and the Warm-Up period. Thereafter, the edge threshold, rank reduction and Smith-Waterman threshold parameters were varied to optimise the performance on the test set.

6.1 Testing on Synthetic Datasets

The first test was conducted on the diamond-walk data set, a synthetic data set created by Saeedi et al. from Imperial College London[41]. These images are synthetic in nature because they are artificially rendered photo-realistic scenes and are not actually taken by a camera in real life. Training and validation was also done on a similar synthetic data set called the deer-walk data set. Testing was done with a frame-lapse of 100. The main challenge with the diamond-walk data set is the poor lighting and illumination present, making it difficult to clearly capture edges within the inputs. Note that the truth labels of this test set was approximated using the methodology outlined in Section 4.1.2 and was set to be relatively strict in classifying positives, as having false positive truth labels are not ideal. This resulted in some positive labels being classified as negative.

6.1.1 Results and Evaluation

For the diamond dataset, performing an edge threshold of 100 and a rank reduction of degree 1 gave us the most optimal results. Figure 6.1 shows the matrices through the various matching stages as well as the truth matrix. As we can see, most of the major loop closures have been identified, apart from the shorter loop closures. This is because in general shorter loop closures are harder to detect for our model which uses a overall sequence thresholding through the Adjusted Smith-Waterman algorithm.

We also plotted the post-PCA latent space as seen in Figure 6.2. We see how similar labels have clustered within their own regions which is a positive sign. However, there is significant overlap between some of these labels, especially on the bottom

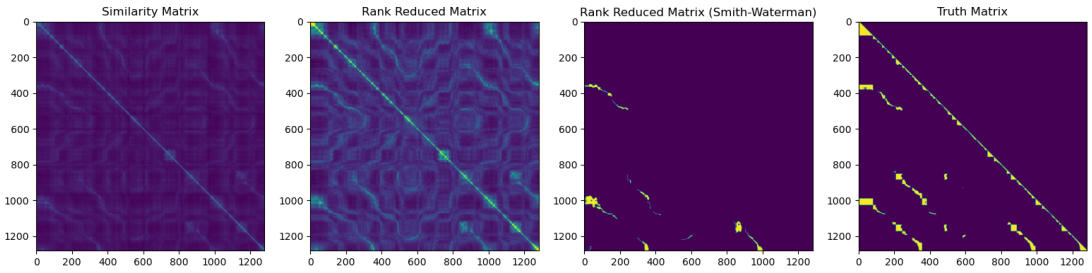

Figure 6.1: Similarity Matrix, RR Matrix and Smith Matrix against Truth for diamond data set.

left side of the image. This is attributed to the fact that the data was classified very tightly by changes in orientation. Therefore, images with slight change in orientation which are actual loop closures might have different labels. As a result, it is more important to see that images with the same labels are clustering well, which the algorithm does fairly well here.

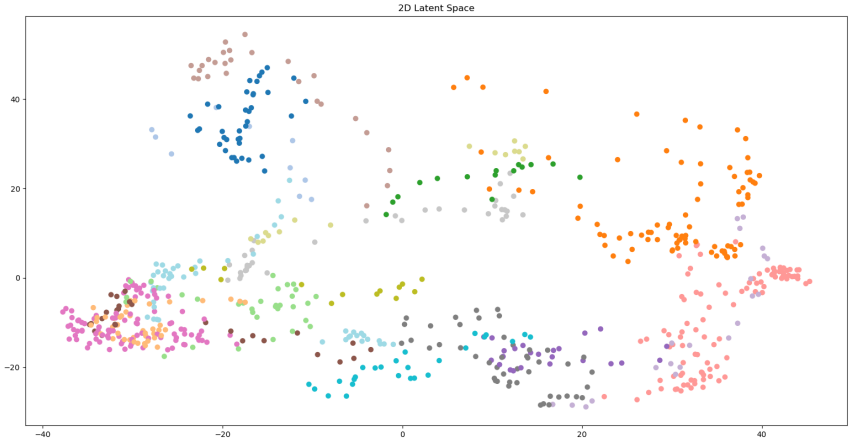

Figure 6.2: Post-PCA latent space distribution.

By examining the strongest sequence identified by the rank matrix in Figure 6.3, we can see that our algorithm has managed to match sequences that have significant viewpoint change, especially the first pair with both rotational and scale changes. This is due to the fact that we use Smith-Waterman which allows us to take lower individual image thresholds without introducing more false positives because of the second sequence threshold and also because of the layers of pooling and the application of random deformations which allowed the algorithm to be relatively tolerant against such changes.
Figure 6.4 shows the plotted Precision-Recall curve that was obtained from different stages of the matching process. As we can see, precision in the case of the the rank reduced matrix remains high, considering the fact that the data set was approximately labelled. There was minimal improvement using Smith-Waterman as the

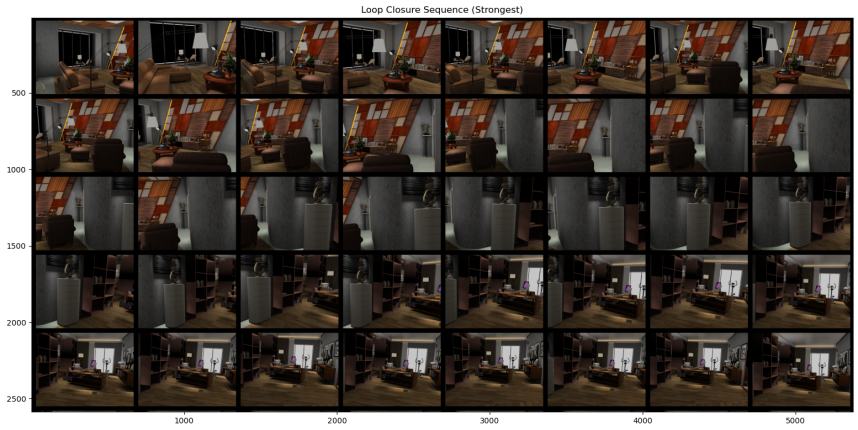

Figure 6.3: Highest Scoring Sequence identified by Smith-Waterman Algorithm. Matching frames are arranged in pairs horizontally within each row.

rank reduced matrix by itself is performing relatively well in identifying individual loop closures. However, we suffer from a slightly sharper drop in precision while trying to increase recall towards the end due to the fact that we do not cover all images for each sequence of loop closure images. This is firstly, because of approximate labelling. Secondly, as seen in the truth matrix, the positive loop closure sequences appear as blocks of squares, meaning there exists many one-to-many matches. This works against rank reduction and Smith-Waterman which aims to create straight defined off diagonal loop closures. Lastly, this is because of the fact that some of the more extreme viewpoint changes were not captured by the algorithm.

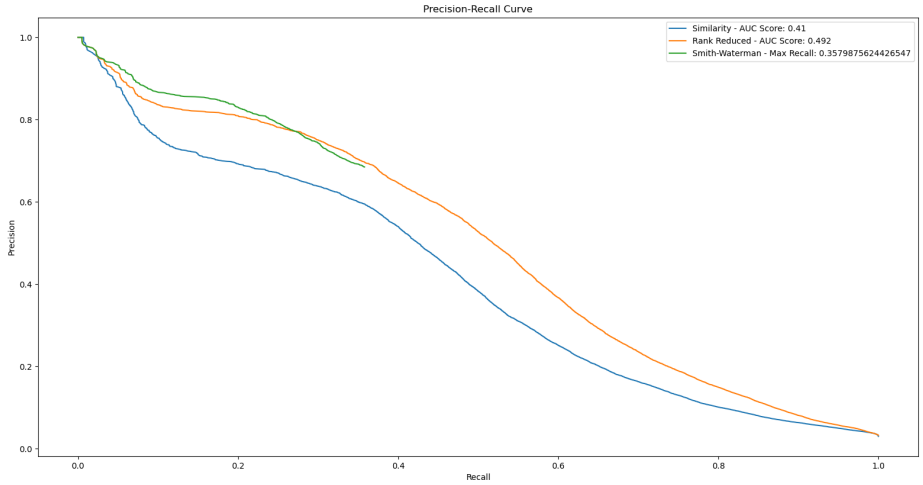

Figure 6.4: Precision-Recall Curve: Diamond Data Set. Smith-Waterman precision recall line is truncated for reasons outlined in Section 5.2.3.

Extreme viewpoint changes are difficult for our algorithm when it causes the focal points to be disrupted by occlusion or if there is too much clutter within the image. Consider Figure 6.5 which is a false negative loop closure. The model likely failed to identify the following image because of the partial occlusion of the curtains, as well as the fact that multiple other focal points exist within these images (i.e. the fridge panel for the right image and the box for the left image). Such issues arises due to the fact that we use global images rather than local key points. The solution to such a problem and why it is difficult to solve within a low illumination data set will be discussed further in Section 6.1.3.

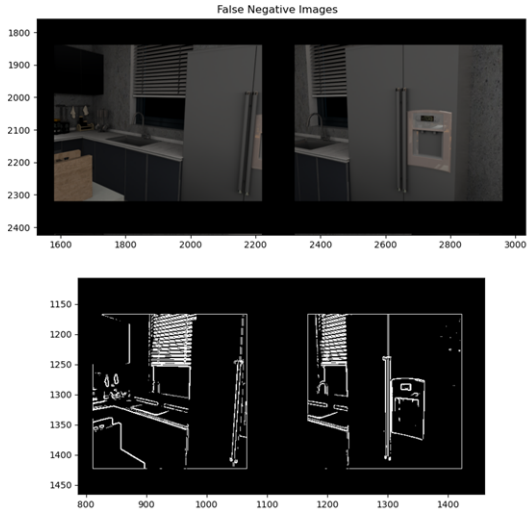

Figure 6.5: Pair of false negative images classified by model.

6.1.2 Rank Reduction in small data sets

Here we discuss rank reduction within a small data set like this. Since the scale of the diamond-walk data set is small, each part of the room is quite unique in terms of its features and hence excessive rank reduction can be very detrimental in such cases. Figure 6.6 shows the Precision-Recall Curves for the varying levels of rank reduction applied.

As we can see, while a rank reduction of degree 1 helped in improving the quality of prediction. Rank reduction with degrees 2 and 3 have removed most of the useful information within the data sets making both precision and recall drop drastically from the model without rank reduction. This is affirmed by inspecting image pairs with the largest decrease of similarity score post a rank reduction of degree 2 as seen in Figure 6.7. As we can see, all of the image pairs removed were actual true positives. Therefore, it is important to exercise precaution with rank reduction especially in small data sets like this.

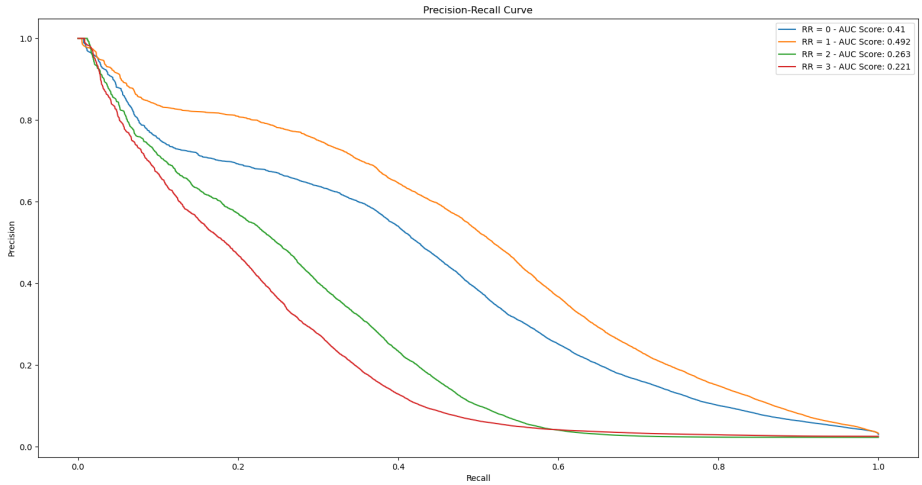

Figure 6.6: Precision-Recall Curve: Diamond Data Set (Rank Reduction).

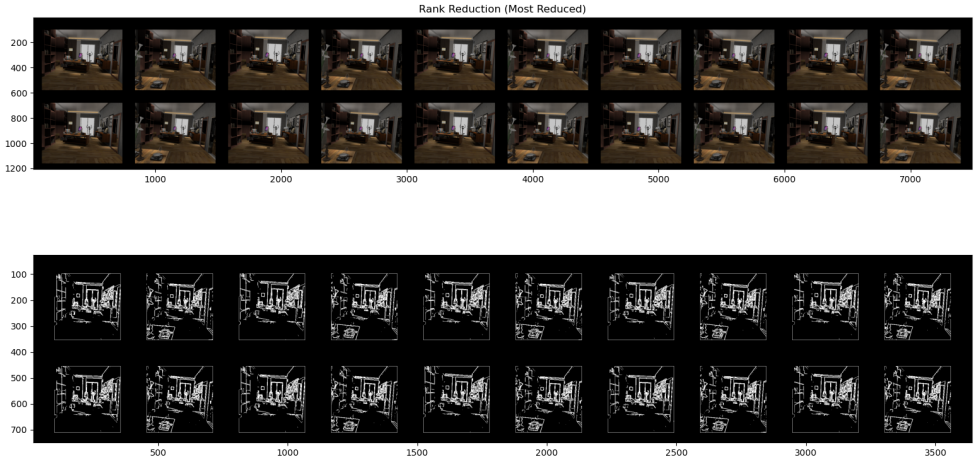

Figure 6.7: Image pairs with largest similarity score reduction from a rank reduction of degree 2. Image pairs are arranged in pairs horizontally within each row.

6.1.3 Limitation of edge thresholding under poor illumination

As mentioned previously, one of the main problems with our algorithm is the fact they use global features instead of local ones. This makes it increasingly susceptible to occlusion and clutter. However, one way we can obtain singular focal points within images is to increase the threshold of edge magnitude within the edge map inputs until we are able to isolate individual focal points with the most significant edges. As we can see from Figure 6.8, by setting a higher edge threshold of 250 we

were better able to capture individual focal points by removing the extraneous noise within the edge images such as the handles of the fridge, majority of the curtains and the oven. This in turn explains the increased similarity score at a a higher threshold.

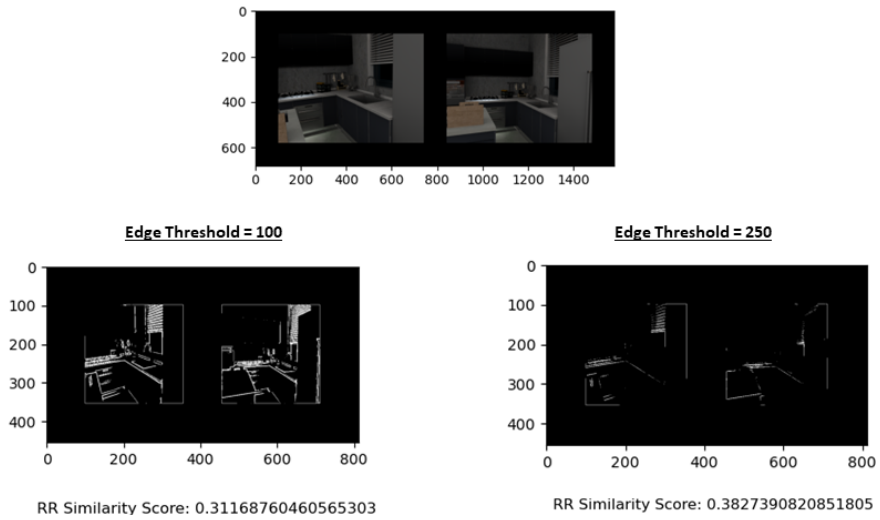

Figure 6.8: Varying degrees of edge thresholding applied on images with clutter.

The issue with this method is the fact that they work poorly in settings of poor or varying illumination as too high a threshold under low illumination settings can potentially erase majority of the features from the image to the point that they become indecipherable as seen in Figure 6.9, where we wipe out the important salient features such as the shape of the chair and most of the wall panels. This is further

Figure 6.9: Sample image and edge map counterpart at a high edge threshold of 250.

affirmed by plotting the Precision-Recall Curves of both different edge thresholds of 100 and 250 in Figure 6.10 with a rank reduction of 1. As we can see, a higher thresholding of 250 has greatly degraded the overall performance of the model as too much salient information has been removed from setting such a high threshold in a poorly illuminated environment.

This trade-off above can be overcome if we choose to apply specific thresholding on each individual image rather than a blanket threshold on the entire data set.

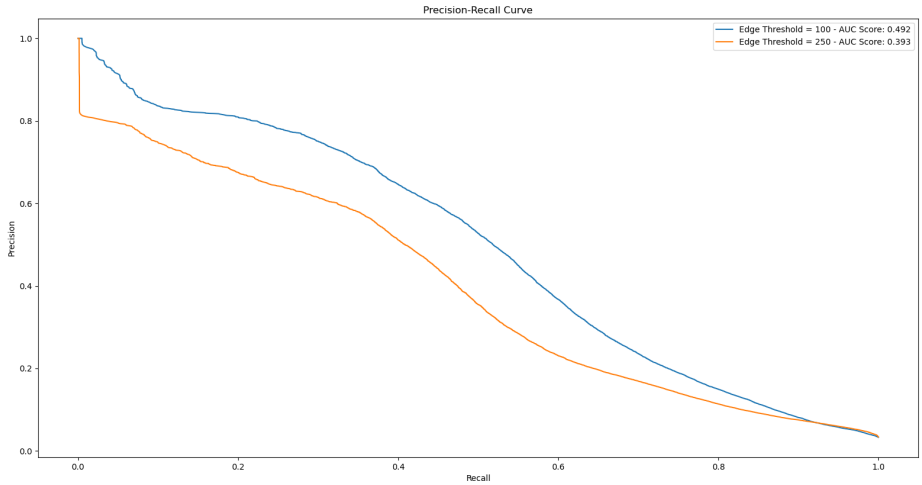

Figure 6.10: Precision recall curve: edge threshold of 100 versus 250.

One possible heuristic is to count the number of edge pixels identified for each image and reduce / increase the edge thresholding for each of these images until they fall within a certain range. This is thus an area of consideration for application into the SCAMP-5 device and will be elaborated in Section 7.2.2.

6.2 Generalising prediction to real-life data sets

Due to the absence of colours and textures and intensity values, binary edge images for synthetic data sets closely resemble the binary edge images of real-life images. As such, we hypothesise that a neural network trained on the binary edge images of synthetic data sets, should be able to generalise well enough to do actual prediction on the real-life data sets. To this end, we trained the model using synthetic images from the deer-walk and diamond-walk data sets, while testing was done on the New College and City Center[9] data sets which were completely excluded from the training process. These datasets consist of images taken by a camera mounted on a robot taken within a campus and park respectively. Both data sets consists of approximately 2400 images per data set. Each of these sequences are taken using a camera on pan-tilt, meaning that the consecutive images captured are from the left and right view of the robots. This means that adjacent input images will not match as they are taken from completely different viewpoints. Since this does not work well with the Smith-Waterman Algorithm which aims to collect sequences of loop closure images, we break each data set into 2 subsets i.e. those taken in the left viewpoint and those taken in the right viewpoint. Both these data sets came with their own set of truth labels. However, such truth labels are also approximated using positional data and are therefore not completely precise in nature.

6.2.1 Performance on New College Data Set

At an edge threshold of 200 and a rank reduction of degree 8 with a frame-lapse of 50 and testing only on the view of one side of the robot of the New College data set, we were able to obtain the following prediction matrices at different stages of the prediction process as seen in Figure 6.11. As we can see from the final Smith-Waterman matrix, we were able to identify all loop closure sequences even the smaller ones that occur in the bottom left corner of the images as well as the shortest one around the (1200, 400) coordinate, with the shortest loop closure identified being across 9 consecutive frames.

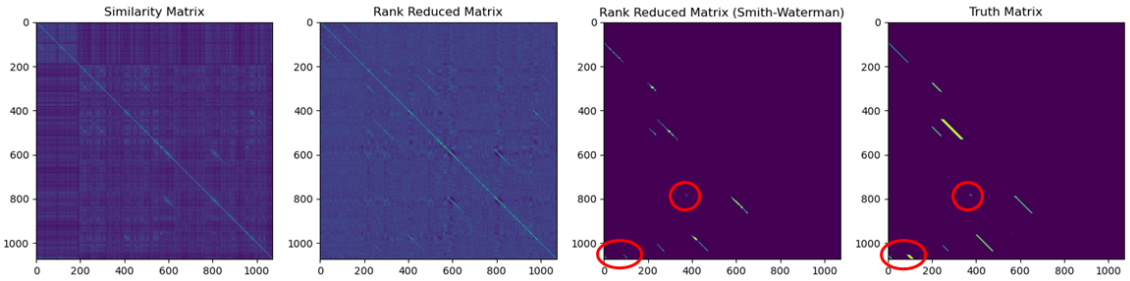

Figure 6.11: Similarity Matrix, RR Matrix and Smith Matrix against Truth for New College data set. Correspondingly found shortest loop closure sequences in the truth and Smith-Waterman matrix is circled in red.

While the truth matrices suggested that there were false positive loop closures identified, visual inspection of these false matches were identified to be due to mislabelling of the approximate truth labels. In order to further verify this, we plot the approximate movement of the robot which was tabulated using the provided x- and y-coordinates where the images were taken in Figure 6.12. Note that approximately 80 frames were not annotated and as such we remove them from the plot. For image pairs that we predict to be a loop closure, we plot red dots at their corresponding x- and y-coordinates and draw a dotted green line between them to represent that match. As we can see, all predicted loop closure matches were within close proximity where the blue lines overlapped, implying that there were indeed no false positives captured within our results.

While we cover every single loop closure sequence within the data set, we were unable to cover every single image pair within those sequences. This can be seen from the different length and thickness of the loop closures identified in both the Smith-Waterman matrix and the Truth Matrix in Figure 6.11. In order to assess the extent of the number of loop closures that were covered by the algorithm, we plot the same approximate movement of the robot but this time using the truth labels that were provided. The result can be seen in Figure 6.13. As we can see, the loops identified around the circular region occurs more densely than those found by our algorithm, although it can also be seen that our algorithm has covered all the major regions where loop closure occurs. This is attributed to the fact that while we were able to identify all the sequences, we do not identify every single image pair within each sequence. This is again attributed to rank reduction and Smith-Waterman

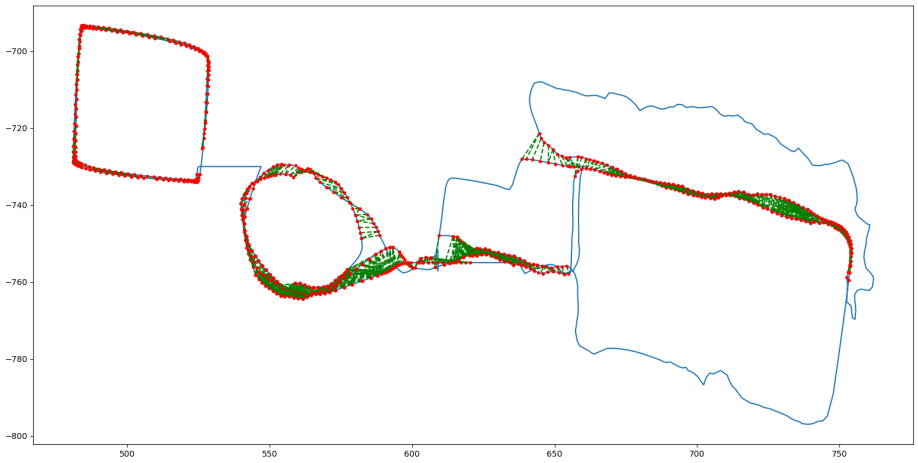

Figure 6.12: Approximate movement of robot annotated with the predicted loop closures on the New College data set.

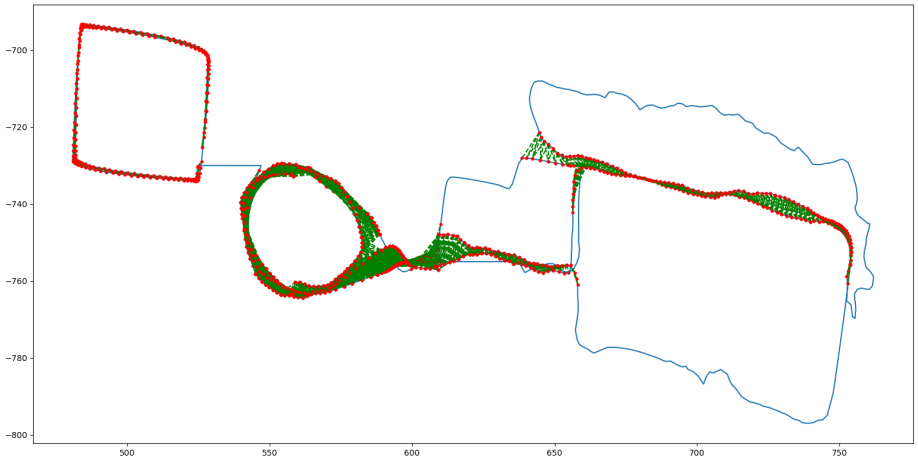

Figure 6.13: Approximate movement of robot annotated with the loop closures generated of truth labels on the New College data set.

which aims to obtain defined thin sequences which as we can see in Figure 6.11, the lines for the loop closure in the Smith-Waterman matrix is much thinner or shorter in some cases compared to the truth matrix counter-part. Yet, we argue that the benefit of identifying every single scene within a sequence especially for the case one-to-many matches i.e. horizontally thick lines, brings about very little additional benefit for revising the robots positional coordinates. Conversely, the removal of such false positives brought about by these algorithms is key to retaining the reliability of the navigational system.

Finally, we examine an image sequence that was identified within the New College Data Set as in Figure 6.14. As we can see image pairs were tolerant to viewpoint and scale changes. It is also tolerant towards the inclusion of additional noise within the image such as the second image pair where one had a person walking in the frame whereas the other one does not.

Figure 6.14: Subset of sequence with the largest total similarity score identified in the New College data set. Image pairs are arranged in pairs horizontally within each row.

Such reliable performance despite the training being conducted on synthetic images is a very useful property as developing loop closure systems with the use of binary edge maps can be done without the additional need of manually capturing a large amount of real-life training photos to train the system on.

6.2.2 Rank Reduction on large data sets

Here we contrast the importance of rank reduction in a large data set like New College with that of rank reduction applied on the diamond-walk data set conducted within Section 6.1.2. Unlike the diamond-walk data set, the New College data set was taken on realistic images outdoors and covered a much larger variety of scenes. There were also many common themes that may lead to the problem of perceptual aliasing such as, brick walls of different buildings or vegetation found in different locations around the gardens. As such, its important to remove such common themes through the use of rank reduction. We varied the degrees of rank reduction and plotted their performance as in Figure 6.15. As we can see, a high degree of rank reduction is required to improve the initial precision of the model. A rank reduction of 2 and 4 did not bring about any increase in the precision whereas a rank reduction of 6 and 8 did, with rank 8 performing better than the rest. Some of these image pairs which were removed by rank reduction can be found in Figure 5.24.

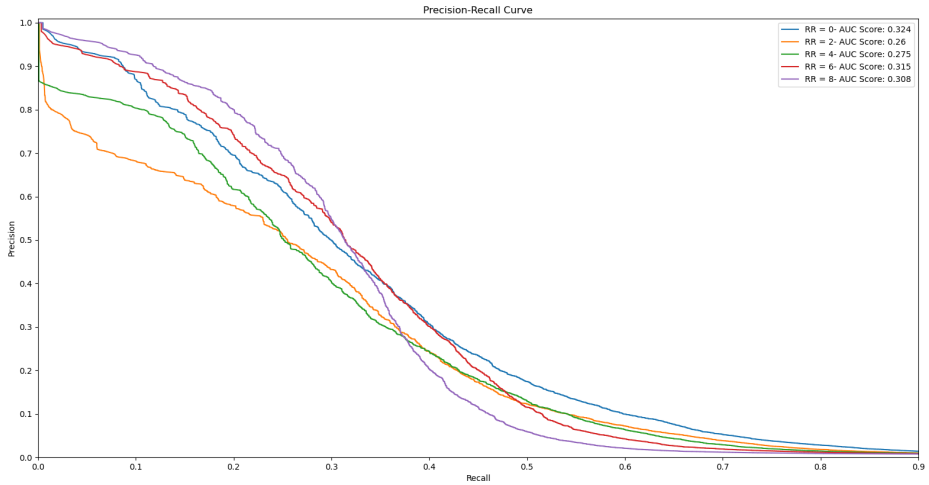

Figure 6.15: Precision-recall curve: Varying rank reduction applied on the New College data set.

Ultimately, the main point that is being driven across here is the importance of allowing the user to tune rank reduction manually due to the fact that binary edge images are quite sensitive to such changes. A general heuristic for rank reduction include setting a higher degree of rank reduction in larger image data sets and a lower degree in smaller image data sets. Generally, outdoor data sets do require a larger degree of rank reduction as well.

6.2.3 Comparison of performance on New College with other systems

Here, we compare our results to the stacked denoising autoencoder (SDA) trained on RGB images conducted by Gao et al. For testing on these data sets, Gao et al. reduced the number of test images by taking an even smaller subset of approximately 240 images from the New College data set. Due to the labelling of this data set being approximated of the robot's position, taking a smaller subset will result in the increased performance for the precision-recall trade-off as there will be less erroneous data. Therefore, in order to make a fair comparison of our algorithm's performance, we took a smaller subset of 269 images from this data set and plot the precision recall curve on this smaller dataset. Frame-lapse was also reduced to 15 frames. Similar to them, we sub-sampled with uniformly from the data set, meaning that we sub-sampled frames with equally spaced intervals. Our model was executed with an edge threshold of 75 as well as a rank reduction of degree 3. Since our similarity matrix performed better than the rank reduction one in this case, we plotted both the performance of the similarity matrix and the post-Smith-Waterman matrix against the against the SDA and FABMAP2.0 algorithms published by Gao et al. in their paper[14] as seen in Figure 6.16.

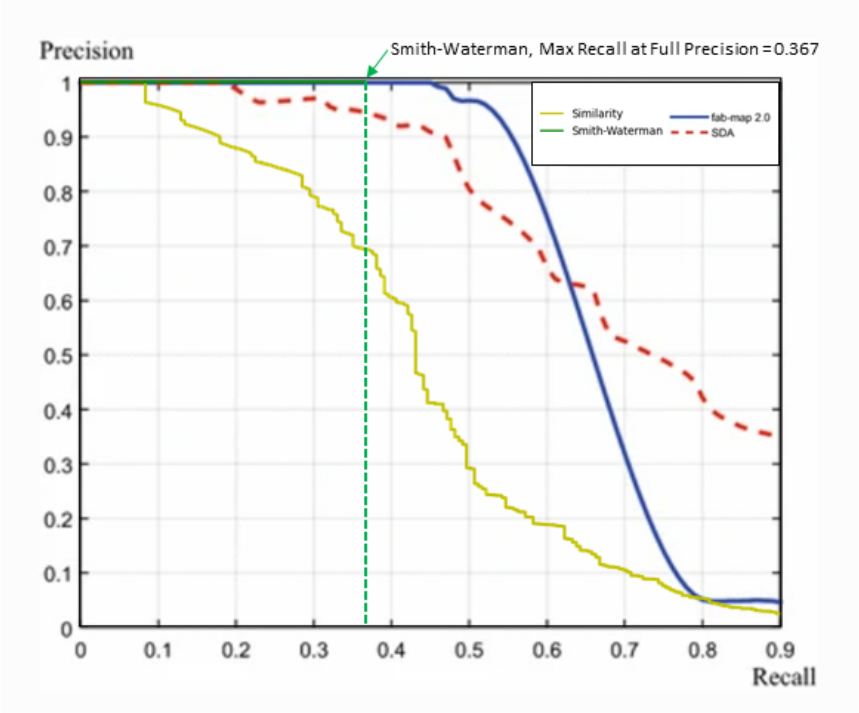

Figure 6.16: Precision Recall Curve: our algorithm versus FABMAP2.0 and denoising stacked auto-encoder (SDA) on the New College data set. Performance of SDA and FABMAP 2.0 were taken directly from [14]. Smith-Waterman PRC was truncated for reasons stated in 5.2.3.

As seen from the curve, our post-Smith-Waterman was able to attain a higher recall of 0.367 than SDA with full precision. This is likely due to the fact that the Smith-Waterman algorithm provides an additional false rejection step unlike the case of the SDA. However, this performance is still lower than the FABMAP2.0 since their approach uses local hand-crafted SURF descriptors which contains more information than binary edge images, for example gradient orientation. We further compare our algorithm performance against other algorithms which uses binary features that were highlighted in Section 3 - the log spectral binary image algorithm reported a maximum recall of 0.209 on the New College data set[48], while BRIEF-GIST reported a maximum recall of 0.79[45]. However, BRIEF-GIST was trained on a curated New College data set that involved the removal of false labelling mentioned above which might make direct comparisons unfair.

On the other hand, comparing the precision-recall performance of the normal similarity matrix without the Smith-Waterman augmentation, we see how performance at the start was quite comparable to the SDA's precision-recall performance, with its performance degrading significantly as the algorithm tries to increase recall. This is not surprising as binary edge images in general have much less content than RGB images. Therefore, it is generally quite hard to increase recall for edge images due to the difficulty in matching such images. We give an example of a False Negative image pair using predictions from the similarity matrix in Figure 6.17. Clearly, without colours delineating the ground and the grass patch, it is generally quite hard to infer from the binary edge map that this is indeed a match.

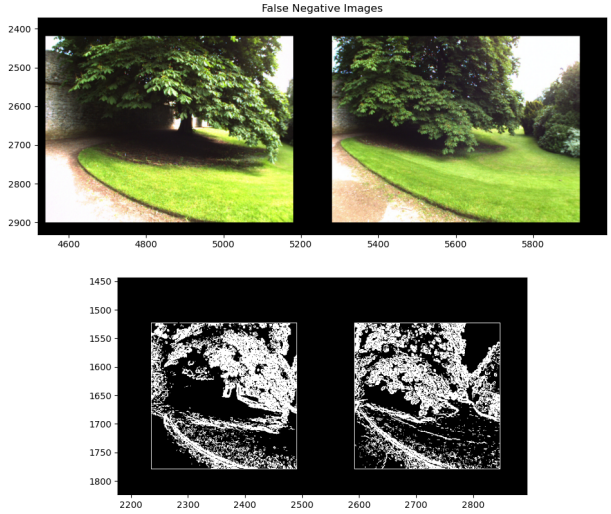

Figure 6.17: Example False Negative Pair obtained via similarity matrix predictions.

However, at the end of the day, it is more important to retain a higher precision within a loop closure detection algorithm and for that our system without any augmentations (i.e. rank reduction and Smith-Waterman) still fared generally well compared to the SDA as shown from the smaller gaps between the yellow and red curves in Figure 6.16 the start of the precision-recall curve.

6.2.4 Results on City Center Dataset

Similar to the New College data set, training was done only the synthetic images of the deer-walk and diamond-walk data set. Optimal results were obtained through an edge thresholding of 100, a rank reduction of degree 5 at a frame-lapse of a 100. Again, we only used half of the data set that corresponds to the view of one side of the robot. The obtained similarity matrix, rank reduced matrix and Smith-Waterman matrix is shown in Figure 6.18.

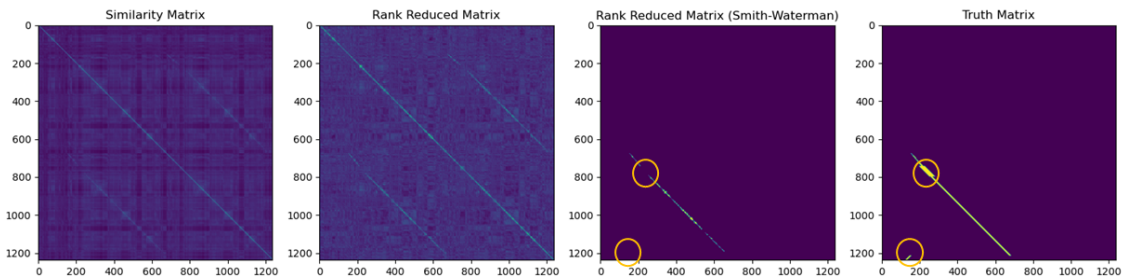

Figure 6.18: Similarity Matrix, RR Matrix and Smith Matrix against Truth for City Center data set. Unidentified loop closure sequence are shown within the orange circles.

By comparing the smith matrix and the truth matrix, we see that we were unable to identify all the loop closures sequences within the City Center data set as seen from the discrepancies annotated by the orange circles. However, upon closer inspection, we realised that majority of both of these sequences contained either extreme viewpoint changes or are mislabelled truth values. We take a few of these examples and plot them in Figure 6.19.

Figure 6.19: Subset of mislabelled image pairs and their corresponding edge maps that resulted in an incorrect False Negative classification. Image pairs are arranged in pairs horizontally within each row.

We see that these False Negative pairs are generally quite different and visually look dissimilar. To further corroborate this, we plot the approximate robot movement with predicted loop closures in Figure 6.20 and those with loop closures identified within the truth labels in Figure 6.21.

As we can see, the only missing loop closure connections between our predicted loop closures and those of the truth labels is the missing portion at the top of the overlapping blue lines. These also happen to be spaced quite far apart relative to the rest of the overlapping blue lines within the image. We believe that as a result there was a mis-classification of these approximated loop sequences. Nevertheless, from these images, we see how we were able to obtain majority of the loop closure sequences without the presence of any false positive loop closures.

Lastly, we once again benchmark our results on the city center data set to the SDA. Similar to before, only a uniformly spaced subset of these images were used. This is to match the number of images Gao et al. used in their experiments.[14] Section 6.2.3 elaborates on the reason on why such a subset was used for testing. Testing here was done on an edge threshold of 100, a rank reduction of 3 and a frame-lapse of 50. We plot both the rank-reduced curve and the post-Smith-Waterman curve in Figure 6.22.

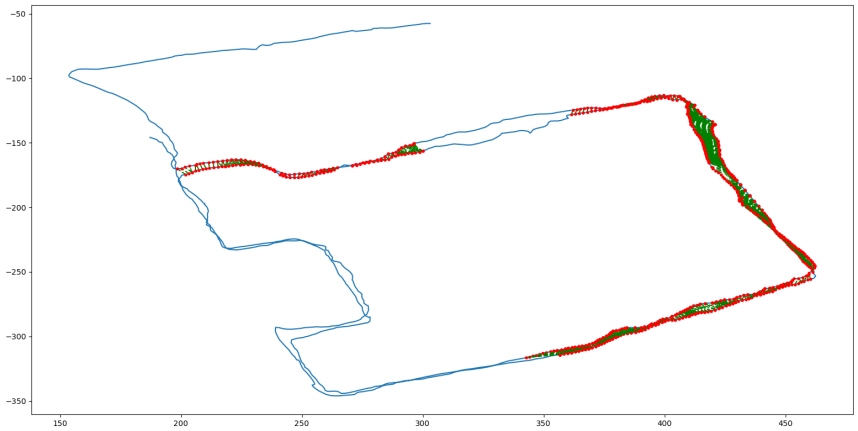

Figure 6.20: Approximate movement of robot annotated with the predicted loop closures on the City Center data set.

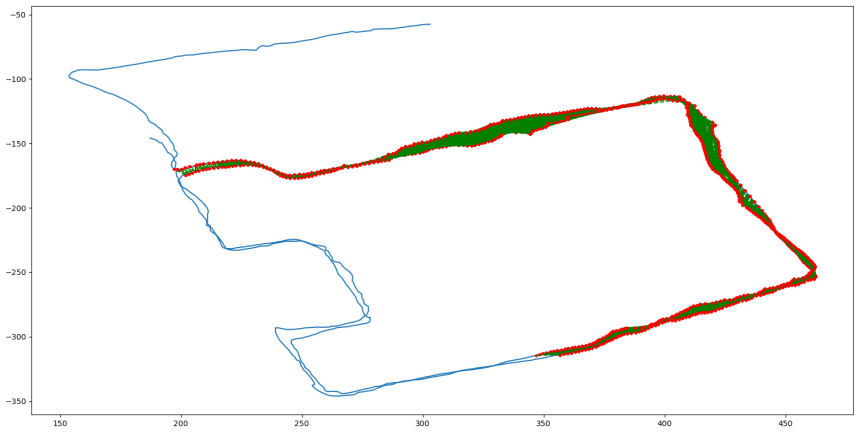

Figure 6.21: Approximate movement of robot annotated with the loop closures generated off truth labels on the City Center data set.

As we can see from the precision-recall curve, we were able to hit a maximum recall of 0.301 at full precision with the additional Smith-Waterman algorithm. This shows better performance than both the FABMAP2.0 as well as the SDA. We also compare this maximum recall value attained by our algorithm performance against other algorithms which uses binary features that were highlighted in Section 3 - the log spectral binary image algorithm reported a maximum recall of 0.277[48], while BRIEF-GIST reported a maximum recall of 0.04 for tiling 1 and 0.32 for tiling 7[45]. Only BRIEF-GIST at tiling 7 was able to beat our post-Smith-Waterman maximum recall value. Similar to us BRIEF-GIST also uses a binary global descriptor. However, they utilised an additional method of tiling here which involves splitting the main image into smaller tiles and calculating descriptors for each of these tiles before recombining these descriptors for comparison purposes. Therefore, their algorithm

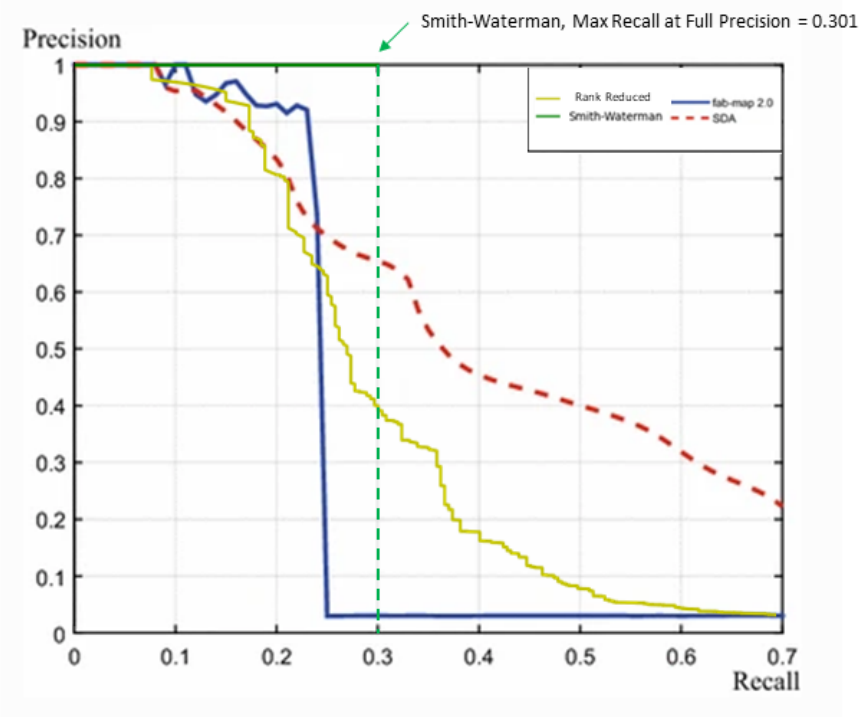

Figure 6.22: Precision Recall Curve: our algorithm versus FABMAP2.0 and denoising stacked auto-encoder (SDA) on the City Center data set. Performance of SDA and FABMAP 2.0 were taken directly from [14]. Smith-Waterman PRC was truncated for reasons stated in 5.2.3.

is only a global descriptor at tiling 1 and becomes more like a local descriptor at tiling 7. This might in turn explain the better performance of their algorithm at higher tiling. However, as mentioned previously in Section 5.1, such local descriptors are not really compatible with binary edge images. Lastly, a comparison of just our rank reduced curve and the SDA shows how our algorithm is quite comparable with the SDA at the start but quickly slides off as it tries to increase recall. This is once again due to the fact that it is much harder to achieve high recall rate using binary edge images as mentioned in Section 6.2.3.

6.3 Identifying loop closures with SCAMP-5 outputs

Next, in order to test if our algorithm is compatible and generalisable to the actual binary edge image outputs of the SCAMP-5 device, we once again trained the model only on the synthetic deer-walk and diamond-walk data set, while testing was done on a subset of 362 binary edge images produced by the SCAMP-5 device. These images were taken in uniform intervals from the original data set which contained over 22000 images. Images were produced at a frame rate of 300 FPS, making the SCAMP-5 images quite noisy. The images were also produced at a very high edge thresholding, which made some scenes very sparse in terms of the amount of information it contains. Since we do not have access to truth labels for this data set,

evaluation was done manually.

We applied a rank reduction of degree 1 to the data set and applied a frame-lapse of 75 frames. We continuously reduced the tightness of the Smith-Waterman threshold and visually inspected each sequence of loop closures until we hit the first case of a false positive. Figure 6.23 shows the similarity matrix, rank reduction matrix and subsequent Smith-Waterman matrix obtained under conditions that did not give any false positive results.

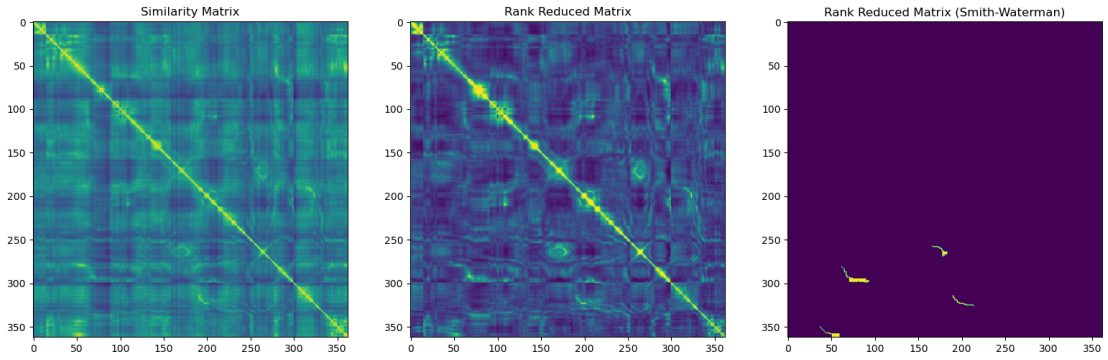

Figure 6.23: Similarity Matrix, RR Matrix and Smith Matrix against Truth for SCAMP-5 data set.

As we can see for the similarity and rank reduced matrices, pixels near the main diagonals are quite bright relative to the rest of the matrix. Since adjacent images are definitely similar, this is a clear sign that the algorithm is working well within this data set. To further verify these results, we visually check each of the 4 major sequences identified by the Smith-Waterman algorithm of which we display the results in Figure 6.24.

As we can see, image pairs within each major sequence are indeed matching and there were no identified false positive sequences under this set of thresholding. The algorithm performs very well on this data set and we highlight some of its strengths within each of these sequences:

1. For images within the first sequence, the algorithm shows that it was able to overcome translation and scale.

2. The second set of images shows how the algorithm was robust towards viewpoint changes with there being a significant amount of viewpoint rotation between pairs.

3. The third set of images shows that the algorithm was able to locate focal points as they were able to match scenes that contained very little information.

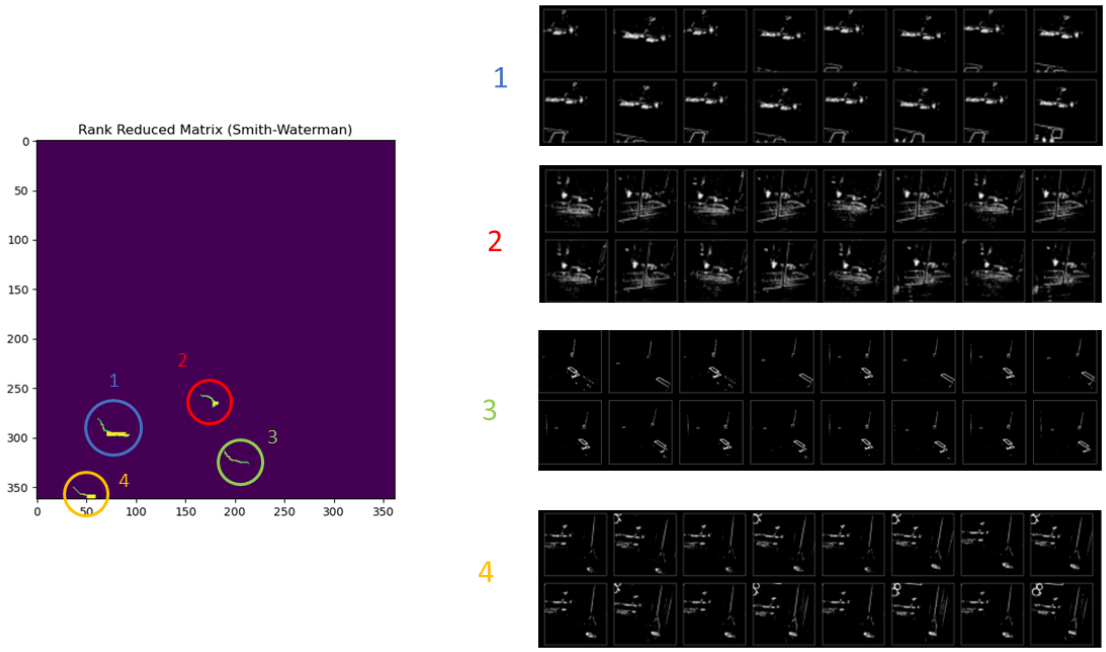

Figure 6.24: Sampled loop closure image pairs from each of the 4 major sequences identified by the Smith-Waterman Algorithm. Image pairs are arranged horizontally within each row i.e. the first two images in row 1 are a pair, the next two images in row 1 are a pair, etc.

4. The last set of images shows that the algorithm was able to filter out noise as it was unperturbed by the additional presence of the circular hanging lights in the top left corner of image as it was able to match two images one with and the other without the hanging lights.

We also tried to identify potential false positive loop closure problems by continuously lowering the acceptance threshold of the Smith-Waterman algorithm. Figure 6.25 shows an example of such a false positive loop closure sequence.

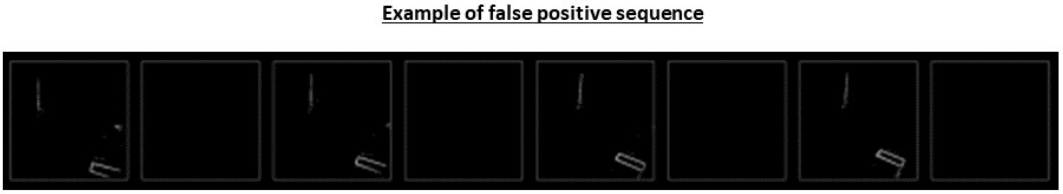

Figure 6.25: Example of false positive sequence identified through a low Smith-Waterman threshold. Image pairs are arranged horizontally within each row.

This example is important because it elucidates a key weakness of high levels of edge thresholding. As we can see,the algorithm is matching an image with very

sparse features with one that is completely black (i.e. devoid of features). Therefore, there needs to exist a mechanism for us to get rid of such sparse images as these can potentially result in many false positive loop closures under such conditions. A simple solution would be to just set a threshold for a minimum number of edge pixels required. However, a more sophisticated method would be to dynamically adjust the thresholding for each image as mentioned in Section 6.1.3 as this will allow us to maximise the amount of possible images we can test for. This will be elaborated on in Section 7.2.2.

6.4 Algorithm Efficiency Summary

Here we highlight the time taken to run each phase of our testing algorithm. Testing was done on an Intel Core i7 CPU equipped with a NVIDIA GEFORCE RTX 2070 GPU. We tested on a subset of the New College data set containing a total of 269 images. Table 6.1 summarises the time taken for each main stage of the process:

Phase	**Total Time Taken (s)**	**Time taken per image (ms)**
Neural Network Prediction	6.81	25.3
Similarity Matrix Tabulation	0.68	2.53
Rank Reduction	2.24	8.33
Smith-Waterman	1.90	7.06
Total Time	11.63	43.23

Table 6.1: Summary of time taken for each step of the prediction process.

Also, in terms of asymptotic efficiency, rank reduction will take a significant amount of time due to the fact that it requires the computation of eigenvalues and eigenvectors which are of complexity $O(n^3 + (n \log^2 n) \log b)$ for a $n \times n$ matrix.[37] Therefore, this will form a bottle-neck in our prediction process when we try to scale the algorithm and hence could be an issue. This will be elaborated on in Section 7.2.2.

6.5 Summary of Results and Evaluation

Firstly, we determined the performance of our algorithm by the training and prediction on the diamond-walk and deer-walk data set and highlighted some difficulties of our algorithm under poor illumination conditions. Next, we show that our algorithm is generalisable enough to test for loop closure on real-life images (i.e. New College and City Center) despite training being restricted to synthetic data sets. Here, we also benchmark the performance against the SDA used by Gao et al.[14] as well as some other algorithms which utilised binary information. The model trained only on synthetic data sets was also shown to perform well on the binary edge image outputs of the SCAMP-5 device. Finally, we rounded off this section with a short summary of the efficiency performance of our algorithm.

Chapter 7

Conclusion and Future Works

7.1 Conclusion

In this piece of research, we have successfully developed a reliable loop closure algorithm implemented on the CPU which utilises a novel set of handcrafted binary edge descriptors designed by Murai et al.[32] as inputs. This algorithm is unique in the sense that it is the first of its kind that combines hand-crafted features with neural networks to perform loop closure detection, whereas most other state of the art methods use either one or the other. Our network is also unsupervised in nature meaning that there is no extra need to annotate our training data sets with labels. This is useful for two reasons - Firstly, such a task is extremely time consuming especially when the train set is large. Secondly, visually identifying loop closure is a subjective process meaning that two individuals can classify the same pair of images differently, depending on their tolerance for both rotational and translational changes. More specifically, we utilised a denoising variational auto-encoder with various augmentations specific to binary edge maps to compress these sparse binary inputs into compact latent descriptors which we can use to represent the image. The VAE regularises the latent space which is particularly useful for binary edge maps which tend to have sparse and extreme image pixel values and the noise we added in this case was in the form of homography deformations which are useful in the context of loop closure detection as we want the descriptor to be invariant to viewpoint changes. Descriptors are then matched using a simple Euclidean distance metric. Since false positive loop closures are detrimental within the context of a SLAM system, we implemented additional mechanisms to control for the occurrence of such false positives such as the rank reduction technique which removes the dominant themes from images and the adjusted Smith-Waterman algorithm which provides an additional temporal restriction on loop closure detection by checking for for loop closure in a sequence rather than using individual images.

The algorithm was demonstrated to have a good performance via its ability to identify all unique loop closure sequences within the New College and City Center data set with no false positives. We also demonstrated that this could be done on a data-set containing outputs produced by an actual SCAMP-5 device. Here, we caveat that while we were able to find all the unique loop closure sequences within the data set, we were not able to cover all matches within these sequences due to the fact that the Smith-Waterman algorithm prefers distinct one-to-one matching within

each sequence rather than one-to-many loop closure matches. Also, our algorithm finds it hard to obtain sequences that are extremely short i.e. 1-3 frames due to the fact that allowing such sequences can more often than not lead to obtaining more false positives. However, we emphasise here that within the context of loop closure detection it is more important that we can identify unique sequences with no false positives rather than trying to cover all the one-to-many loop closure matches which provides very little value-add. This is because false positives can severely corrupt the robot's positional awareness, while the presence of a loop closure sequence is all that is required when we are recovering the robot's positional coordinates regardless of the number of matches exists within those sequences.

Next, we benchmark our algorithms capability by mainly doing a comparison with the state-of-the-art stacked denoising autoencoder (SDA) framework designed by Gao et al.[14] on the New College adn City Center data set. This is also an unsupervised neural network that is used to perform loop closure detection but this solution utilises full-scale RGB images. We demonstrated that our algorithm was able to attain a higher maximum recall rate at full precision with the assistance of the additional false positive detection mechanisms on both data sets. Performance without additional detection mechanisms was also shown to perform at a level close to that of the SDA in terms of precision, considering the fact that we are using inputs that contain much less information than full-scale RGB images.

While using handcrafted binary features within a neural network might seem to be counter-intuitive due to the fact that it already contains much less information compared to just feeding in full-scale RGB images, we discovered that this trade-off in performance brought about an increased flexibility of our neural network's ability to generalise across a multitude of various data sets. This is proven by only training our algorithm using inputs of the synthetically-produced images from the deer-walk and diamond-walk data set but testing on real-life images from the New College / City Center and SCAMP-5 device data sets. We show that in these cases our neural network was still able to successfully and reliably detect loop closures. We emphasise here that there are also distinct differences within these data sets with the New College / City Center data set containing full-scale RGB images that were taken in an outdoor setting, whereas the SCAMP-5 data set contained only binary edge maps taken within the indoor setting with very high thresholding. This increased flexibility is particularly interesting since it provides distinct advantages to the neural network framework. For example, one can have access to a much larger training pool of data as images from different settings can just be simultaneously fed into this network.

In conclusion, we have developed an effective loop closure detection algorithm that uses only binary edge images that is deployed on the CPU. We show that such a detection algorithm is reliable and have capabilities that come close to using actual RGB images. Lastly, we highlighted the increased flexibility of such a system due to its good generalisation properties.

7.2 Future Works

Here we highlight 3 main areas of future works that will be interesting to pursue to further the quality within this research area. The first being the further integration of this loop closure detection system on the SCAMP-5 device, the second being further augmentations to the algorithm to improve efficiency and effectiveness and lastly the need of a more standardised loop closure benchmark data set.

7.2.1 Further integration into the SCAMP-5 device

Since we have shown that our algorithm works well with the outputs of the SCAMP-5 device, one main potential area of future work would be centered around transferring and integrating the current loop detection conducted on the CPU with the BIT-VO system[32] which is an efficient and lightweight visual odometry system which only uses reduced features to do visual odometry. Currently this system lacks the ability to perform loop closure. By integrating our loop closure algorithm which uses the same set of reduced features produced by the device, we will be able to enhance its SLAM capabilities while at the same time, retaining the main advantage of low power consumption and high frame rate.

7.2.2 Improving on effectiveness and efficiency limitations of the current algorithm

There are a few limitations of our algorithm that we have highlighted in Section 6 that can be improved upon. Firstly, would be to avoid the usage of rank reduction as a false positive mechanism check due to its high asymptotic complexity and therefore, will limit the extent to which this loop closure system can be scaled. Also, this removal will lessen the number of hyper-parameters of the algorithm that we have to vary. One possible alternative would be to embed the system with a triplet loss network[18] which aims to remove false positive through the adding of an additional term within the loss function that maximises the difference between an anchor image and a negative image. This will reduce the amount of false positive by moving these false positive images further away in the latent space. However, this solution is a supervised solution and will require truth labels within the training data set. Another solution would be to find a more efficient algorithm that computes eigenvectors and eigenvalues. For example, using a power iteration method of complexity $O(n^2)$[38]. However, this will limit our rank reduction to be of maximum degree 1.

Another limitation we have shown is the difficulty of applying edge thresholding under poor or varied illumination conditions. This is because by doing a blanket edge thresholding across the entire data set under such conditions, we end up with some edge images being too noisy due to insufficient amount of edge pixels removed and others being too sparse due to too many edge pixels being removed. One method to overcome this would be through adapting the SCAMP-5 device to perform dynamic thresholding on each of the images through a heuristic i.e. varying threshold until number of edge pixels identified falls within an acceptable range. Another more interesting solution is the implementation of programmable sensors on such cameras with the ability to adjust individual pixel exposure as in the research done

by Martel et al.[29], where they built neural networks onto sensors that optimise exposure across images.

7.2.3 Crafting a more accurate benchmark data set for loop closure

One of the main issues we faced when conducting evaluation was the lack of accurate ground truth labels within loop closure data sets. Even the most standardised data set used across majority of SLAM for benchmarking i.e. New College and City Center only had approximate ground truth labels which on visual inspection appeared to have quite a significant number of erroneous labelling. Therefore, a possible future endeavour would be through the design of a more defined loop closure data set which allows algorithms to be accurately benchmarked on. Such a data set should try to be objective in its definition of a loop closure by setting limits on the maximum amount of movement and rotation of the camera that entails a loop closure.

Appendix A

Ethics Checklist

	Yes	No
Section 1: HUMAN EMBRYOS/FOETUSES		
Does your project involve Human Embryonic Stem Cells?		✓
Does your project involve the use of human embryos?		✓
Does your project involve the use of human foetal tissues / cells?		✓
Section 2: HUMANS		
Does your project involve human participants?		✓
Section 3: HUMAN CELLS / TISSUES		
Does your project involve human cells or tissues? (Other than from "Human Embryos/Foetuses" i.e. Section 1)?		✓
Section 4: PROTECTION OF PERSONAL DATA		
Does your project involve personal data collection and/or processing?	✓	
Does it involve the collection and/or processing of sensitive personal data (e.g. health, sexual lifestyle, ethnicity, political opinion, religious or philosophical conviction)?		✓
Does it involve processing of genetic information?		✓
Does it involve tracking or observation of participants? It should be noted that this issue is not limited to surveillance or localization data. It also applies to Wan data such as IP address, MACs, cookies etc.		✓
Does your project involve further processing of previously collected personal data (secondary use)? For example Does your project involve merging existing data sets?		✓
Section 5: ANIMALS		
Does your project involve animals?		✓
Section 6: DEVELOPING COUNTRIES		
Does your project involve developing countries?		✓
If your project involves low and/or lower-middle income countries, are any benefit-sharing actions planned?		✓
Could the situation in the country put the individuals taking part in the project at risk?		✓

	Yes	No
Section 7: ENVIRONMENTAL PROTECTION AND SAFETY		
Does your project involve the use of elements that may cause harm to the environment, animals or plants?		✓
Does your project deal with endangered fauna and/or flora /protected areas?		✓
Does your project involve the use of elements that may cause harm to humans, including project staff?		✓
Does your project involve other harmful materials or equipment, e.g. high-powered laser systems?		✓
Section 8: DUAL USE		
Does your project have the potential for military applications?	✓	
Does your project have an exclusive civilian application focus?	✓	
Will your project use or produce goods or information that will require export licenses in accordance with legislation on dual use items?		✓
Does your project affect current standards in military ethics – e.g., global ban on weapons of mass destruction, issues of proportionality, discrimination of combatants and accountability in drone and autonomous robotics developments, incendiary or laser weapons?		✓
Section 9: MISUSE		
Does your project have the potential for malevolent/criminal/terrorist abuse?		✓
Does your project involve information on/or the use of biological-, chemical-, nuclear/radiological-security sensitive materials and explosives, and means of their delivery?		✓
Does your project involve the development of technologies or the creation of information that could have severe negative impacts on human rights standards (e.g. privacy, stigmatization, discrimination), if misapplied?		✓
Does your project have the potential for terrorist or criminal abuse e.g. infrastructural vulnerability studies, cybersecurity related project?		✓
SECTION 10: LEGAL ISSUES		
Will your project use or produce software for which there are copyright licensing implications?		✓
Will your project use or produce goods or information for which there are data protection, or other legal implications?		✓
SECTION 11: OTHER ETHICS ISSUES		
Are there any other ethics issues that should be taken into consideration?		✓

Appendix B

Ethical and Professional Considerations

There are 2 main ethical considerations with regards to the research done within this report. Firstly, is through the use of personal data collection. Due to the fact that the SCAMP-5 camera device was not directly accessible, output images was kindly provided to us by Riku Murai. This involved the capturing of scenes within his home and therefore involves personal data collection. This has been addressed this by firstly, seeking permission to utilise these images for our research and to produce some of these images within the report. Also, since the images produced by the SCAMP-5 are only of edge images, to reduce the amount of information provided within these images, we set an extremely high edge thresholding so that each image only contains a small amount of edges making it hard to identify these objects, hence retaining privacy as much as possible.

The second ethical consideration is the potential military applications of loop closure detection projects since such algorithms could be used within the navigational system of drones and robotics. However, application of our methodology is limited due to the detection being restricted to only visual sensors and the small-scale deployment of the algorithm. Also, no particular contact was made with anyone working within the military with regards to this particular project.

Bibliography

[1] R. Arandjelovic, P. Gronat, A. Torii, T. Pajdla, and J. Sivic. Netvlad: Cnn architecture for weakly supervised place recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5297–5307, 2016.

[2] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[3] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.

[4] J. Bjorck, C. P. Gomes, and B. Selman. Understanding batch normalization. In *NeurIPS*, 2018.

[5] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. Brief: Binary robust independent elementary features. In *European conference on computer vision*, pages 778–792. Springer, 2010.

[6] J. Cardoso, J. Coutinho, and P. Diniz. *Embedded Computing for High Performance: Efficient Mapping of Computations Using Customization, Code Transformations and Compilation*. 01 2017.

[7] S. J. Carey, A. Lopich, D. R. Barr, B. Wang, and P. Dudek. A 100,000 fps vision sensor with embedded 535gops/w 256× 256 simd processor array. In *2013 Symposium on VLSI Circuits*, pages C182–C183. IEEE, 2013.

[8] Z. Chen, O. Lam, A. Jacobson, and M. Milford. Convolutional neural network-based place recognition. *arXiv preprint arXiv:1411.1509*, 2014.

[9] M. Cummins and P. Newman. Fab-map: Probabilistic localization and mapping in the space of appearance. *The International Journal of Robotics Research*, 27(6):647–665, 2008.

[10] M. Cummins and P. Newman. Appearance-only slam at large scale with fab-map 2.0. *The International Journal of Robotics Research*, 30(9):1100–1123, 2011.

[11] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. IEEE, 2005.

[12] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.

[13] D. Gálvez-López and J. D. Tardos. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, 28(5):1188–1197, 2012.

[14] X. Gao and T. Zhang. Unsupervised learning to detect loops using deep neural networks for visual slam system. *Autonomous robots*, 41(1):1–18, 2017.

[15] E. Garcia-Fidalgo and A. Ortiz. *Methods for Appearance-based Loop Closure Detection*. Springer, 2018.

[16] C. G. Harris, M. Stephens, et al. A combined corner and edge detector. Citeseer, 1988.

[17] K. L. Ho and P. Newman. Detecting loop closure with scene sequences. *International journal of computer vision*, 74(3):261–286, 2007.

[18] E. Hoffer and N. Ailon. Deep metric learning using triplet network. In *International Workshop on Similarity-Based Pattern Recognition*, pages 84–92. Springer, 2015.

[19] Y. Hou, H. Zhang, and S. Zhou. Convolutional neural network-based image representation for visual loop closure detection. In *2015 IEEE international conference on information and automation*, pages 2238–2245. IEEE, 2015.

[20] Y. Hou, H. Zhang, and S. Zhou. Bocnf: efficient image matching with bag of convnet features for scalable and robust visual place recognition. *Autonomous Robots*, 42:1169–1185, 2018.

[21] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ArXiv*, abs/1502.03167, 2015.

[22] H. Jégou, M. Douze, C. Schmid, and P. Pérez. Aggregating local descriptors into a compact image representation. In *2010 IEEE computer society conference on computer vision and pattern recognition*, pages 3304–3311. IEEE, 2010.

[23] E. Karami, S. Prasad, and M. Shehata. Image matching using sift, surf, brief and orb: performance comparison for distorted images. *arXiv preprint arXiv:1710.02726*, 2017.

[24] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[25] H. Korrapati and Y. Mezouar. Multi-resolution map building and loop closure with omnidirectional images. *Autonomous Robots*, 41(4):967–987, 2017.

[26] F.-F. Li. Cs231n: Convolutional neural networks for visual recognition, 2020.

[27] D. Liu. A practical guide to relu, Nov 2017.

[28] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[29] J. N. Martel, L. Mueller, S. J. Carey, P. Dudek, and G. Wetzstein. Neural sensors: Learning pixel exposures for hdr imaging and video compressive sensing with programmable sensors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[30] N. Merrill and G. Huang. Lightweight unsupervised deep loop closure. *arXiv preprint arXiv:1805.07703*, 2018.

[31] M. J. Milford and G. F. Wyeth. Seqslam: Visual route-based navigation for sunny summer days and stormy winter nights. In *2012 IEEE International Conference on Robotics and Automation*, pages 1643–1649. IEEE, 2012.

[32] R. Murai, S. Saeedi, and P. H. J. Kelly. BIT-VO: Visual Odometry at 300 FPS using Binary Features from the Focal Plane. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[33] D. Nistér, O. Naroditsky, and J. Bergen. Visual odometry. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 1, pages I–I. Ieee, 2004.

[34] M. S. Nixon and A. S. Aguado. Feature extraction image processing for computer vision, third edition. 2012.

[35] H. Noh, S. Hong, and B. Han. Learning deconvolution network for semantic segmentation. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1520–1528, 2015.

[36] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International journal of computer vision*, 42(3):145–175, 2001.

[37] V. Y. Pan and Z. Q. Chen. The complexity of the matrix eigenproblem. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, STOC '99, page 507–516, New York, NY, USA, 1999. Association for Computing Machinery.

[38] M. Panju. Iterative methods for computing eigenvalues and eigenvectors. *arXiv preprint arXiv:1105.1185*, 2011.

[39] J. Rocca. Understanding variational autoencoders (vaes), Jul 2020.

[40] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. Ieee, 2011.

[41] S. Saeedi, E. D. Carvalho, W. Li, D. Tzoumanikas, S. Leutenegger, P. H. Kelly, and A. J. Davison. Characterizing visual localization and mapping datasets. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6699–6705. IEEE, 2019.

[42] J. Sivic and A. Zisserman. Video google: A text retrieval approach to object matching in videos. In *null*, page 1470. IEEE, 2003.

[43] C. Sønderby, T. Raiko, L. Maaløe, S. K. Sønderby, and O. Winther. Ladder variational autoencoders. In *NIPS*, 2016.

[44] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. A. Riedmiller. Striving for simplicity: The all convolutional net. *CoRR*, abs/1412.6806, 2015.

[45] N. Sünderhauf and P. Protzel. Brief-gist-closing the loop by simple means. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1234–1241. IEEE, 2011.

[46] R. Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[47] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, 2010.

[48] H. Wang, J. Li, M. Ran, and L. Xie. Fast loop closure detection via binary content. In *2019 IEEE 15th International Conference on Control and Automation (ICCA)*, pages 1563–1568. IEEE, 2019.

[49] Y. Wang, T. Mei, S. Gong, and X.-S. Hua. Combining global, regional and contextual features for automatic image annotation. *Pattern Recognition*, 42(2):259–266, 2009.

[50] Á. Zarándy. *Focal-plane sensor-processor chips*. Springer Science & Business Media, 2011.

[51] C. L. Zitnick. Binary coherent edge descriptors. In *ECCV*, 2010.