Structure from Motion with a neural network

Jiarong Gong

16th June 2023

Abstract

This project delves into the 3D reconstruction of both single and multiple rigid motions, examining the potential of deep learning methods, such as that proposed by Moran et al., to supplant traditional geometry-based approaches. The project is structured into two main parts.

In the first part, we focus on the 3D reconstruction of a single rigid motion, building on the work of Moran et al. In addition to using the dataset they used in their paper, we expand it with a new one named BlendedMVS and evaluate the generalization performance of the network on this enriched dataset. The network, in general, performs commendably in both single-scene optimization and multi-view learning settings. Furthermore, by exploring different architectures and enhancing the network, we manage to slightly improve the success rate of single-scene optimization.

In the second part, our attention shifts to multiple rigid motions. We initially employ YOLOV7 and Deep Sort for motion segmentation, using a portion of the Hopkins 155 dataset. In total, there are nine video files, each corresponding to a segmentation accuracy. The results reveal five instances of 100% accuracy, with the lowest accuracy standing at 99.55%. In terms of single-scene optimization and multi-view learning settings, there are no failed 3D reconstructions, indicating that all the reprojection errors fall below two pixels.

Popular science summary

Transforming 2D Images into 3D Reality: Recovering Objects and Camera Positions

From multiple 2D images, we can recreate a vivid 3D object. Amazing, isn't it? The real challenge lies in the accuracy of this reconstruction. We measure this accuracy using something called 'reprojection errors', which simply put, tells us how closely our 3D reconstruction matches the original 2D images.

The objective of this project is to reconstruct 3D objects from multiple 2D images. This task can be complex, especially when images contain both stationary and moving objects, such as buildings and vehicles. So, the project is divided into two parts based on the object types.

In the first part, we worked with images containing only stationary objects like buildings, parked cars, and trees. We used a new dataset named 'BlendedMVS'. With the help of advanced image processing techniques, we extracted 'point matches' from these images. A 'point match' represents the same 3D point viewed from different images. After compiling these points into an input for our model, we used the network proposed by researchers Moran et al. to automatically generate the 3D objects and camera positions in space. We also add some improvements based on the network to increase the reconstruction quality. The results were promising; we managed to improve the accuracy of the reconstructions, as indicated by lower reprojection errors.

In the second part, we worked with video files, which can be seen as a series of images. This time, the data contained both stationary and moving objects. For this task, we used a dataset called 'Hopkins 155'. In each video, some 2D point matches from both the stationary scene and moving cars were already provided. Since we couldn't reconstruct all of them at once, we segmented them and reconstructed each individually. To do this, we used tools like YOLOV7 and Deep Sort to detect and isolate different objects, such as cars and trucks. By seeing which bounding box a point was located in, we could roughly determine which object it came from. This method gave us highly accurate segmentation results, close to 100%. Using our improved architecture, we achieved reprojection errors below 1 pixel for 'Hopkins 155' dataset, indicating high-quality 3D reconstructions. Figures 1 and 2 are provided here to illustrate the process.

Figure 1: One image from the input image set: multiple angle views of an ox sculpture.

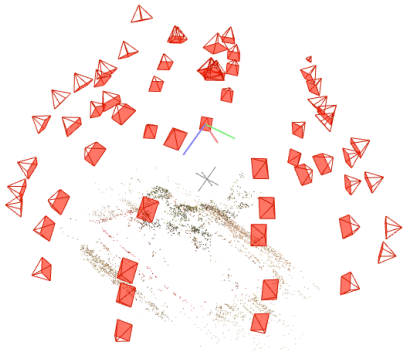

Figure 2: This image illustrates both the 3D point cloud and cameras. The red signs denote camera locations and orientations. The point cloud is surrounded by those cameras. The point cloud is a set of 3D points on the surface of a 3D object.

Acknowledgements

I would like to extend my heartfelt gratitude to Dr. Yaqing and Dr. Carl for their invaluable support throughout my project. Dr. Yaqing, my supervisor, consistently offered insightful guidance on overcoming technical challenges that I could not have managed on my own. His patience and expertise were truly inspiring. Dr. Carl, my assistant supervisor, not only provided me with the opportunity to undertake this project but was also the primary source of its innovative concepts and content. He shared countless innovative ideas that greatly enriched the project, demonstrating his unwavering faith in my abilities.

Additionally, I would like to express my appreciation to the equipment manager, Mr Carl Gustav Werner, from the Mathematical Science Department who granted me access to a computer equipped with a TITAN X GPU, a critical resource for the success of my project.

Finally, I would like to extend my deepest gratitude to my family for their unwavering support, both financially and emotionally, not only during my master's thesis project but throughout my entire journey in pursuit of my Master's degree. Their encouragement, constant reassurance, and belief in me have been instrumental in helping me persevere and reach this milestone.

Contents

1 Introduction

This chapter provides an overview of the thesis, including the research problem, its background, and the purpose of the study. It also outlines the motivation for investigating deep learning-based methods for Structure from Motion (SfM) problems.

1.1 Research Problem

1.1.1 Background

The basic understanding of structure from motion (SfM) problem is the recovery of 3-dimensional structure of objects from their images (projections) taken by several cameras. The fact that the 3-D structure of an unfamiliar object can be recovered via its projections was first proved by Wallach and O'Connell [1] without the need to recognize it.

SfM is a technique for estimating the three-dimensional structure of a scene from a series of two-dimensional images taken at different viewpoints. A common approach to solving SfM problems is using geometry-based methods, which generally consist of two main stages. In the first stage, a starting solution is built by alternately solving triangulation and resection problems, a process known as Incremental Reconstruction. Triangulation is the process of determining the 3D position of a point by measuring its position in multiple images, while resection is the process of estimating the camera's position and orientation based on the known 3D positions of visible points in the scene. After obtaining a starting solution, the second stage involves applying an optimization technique called Bundle Adjustment (BA). BA minimizes the difference between the observed positions of points in the images and the positions predicted by the 3D model and camera parameters. This optimization results in a more accurate reconstruction of the 3D scene, as it refines both the estimated 3D points and camera parameters simultaneously [2].

An alternative way of solving SfM problems is based on neural networks. Recent years have seen more and more such related works [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] and the deep network architectures they proposed seemed to have shown great power on the reconstruction task of 3-D objects. Fu et al. [6], Eigen et al. [7] and Lee et al. [8] focus on the depth map estimation with a single input image using deep neural networks. Although these methods perform well, they suffer from the scale problem, which refers to the difficulty in determining the absolute size and distance of objects in a scene from a single image. Also, these methods are limited by unseen data. To address the scale problem, one useful approach is to utilize at least two different images taken from different viewpoints. Wang et al. [4] proposed a so-called DICL-Flow deep neural network to extract dense matching points between 2

consecutive views instead of SIFT matching as the first step of estimating depth maps. Then they, based on the extracted point matches, followed the traditional geometry-based method to successively estimate the essential matrix E, extract the rotation and translation matrices and recover the 3-D structure via triangulation. Tang and Tan [9], Yao et al. [13], Yu and Gao [14] and Gu et al. [15] utilized the multi-view setting to do 3-D structure recovery. Most of the work is based on the convolutional neural network (CNN). Apart from CNN, some people utilize the recurrent neural network (RNN) to directly estimate the depth maps and camera poses (equivalent to 3-D structure) like Gu et al. [5], who designed the network with 2 sub-networks, called depth GRU optimizer and pose GRU optimizer that are used to iteratively update the depth and camera pose estimations. It showed that such RNN model outperformed several state-of-the-art methods.

Besides direct recovery of 3-D scene structure like [5], people such as Wang et al. [4] employed the deep learning method to replace one of the 2 stages of traditional geometry-based method, giving an initial reconstruction of 3-D objects and BA. Moran et al. [16] employed a deep permutation equivariant neural network to simultaneously recover the camera poses and 3-D points with 2 sub-networks, one for camera pose estimation and the other for 3-D point recovery. With the help of a useful neural network, we can have a good starting solution, which will be sent to BA processing, the second stage. [16] is exactly the basis and starting point of the thesis work.

In addition to those static scenes, the work will explore the deep learning-based method for non-rigid scenes. Non-rigid scenes include those that comprise of static objects like buildings and moving objects such as moving cars and those that consists of moving objects and/or deforming objects. For deformable objects, Novotny et al. [3] solved the structure from motion problem via a deep network by firstly factoring the effects of viewpoint changes and object deformations. For scenes including only static and moving non-deformable objects, Sabzevari and Scaramuzza [17] solved this so-called multi-body structure from motion (MBSfM) problem based on the factorization of multiple-trajectory matrix instead of an initial motion segmentation which is often used in MBSfM problem. Addressing such problem makes up the second part of my thesis work.

1.1.2 Motivation

We have observed that various deep learning methods have been employed to address SfM and MBSfM problems. Researchers have used networks to estimate depth maps solely from input images, extract feature point correspondences between two images, or recover 3-D points only. They have even tackled the same type of problems using different network types (CNN, RNN). All of these approaches have achieved a certain degree of success, which demonstrates the usefulness of deep learning-based methods in this field.

Moran et al. [16] designed a deep neural network euiqvariant to the order of cameras and 3-D points to recover the camera poses and 3-D scene structure simultaneously. They also demonstrated that, with a fine-tuning step, it is possible to successfully reconstruct a novel scene using a pre-trained neural network.

Building on the work of Moran et al. [16], it is meaningful to improve and generalize the method for unseen scenes. In their paper, they tested their neural network architecture on 38 static scenes, achieving impressive performance on par with several state-of-the-art methods. Testing the method on a larger variety of scenes and achieving similar performance as described in the paper would further strengthen the method's validity and increase its persuasiveness.

Furthermore, researchers have studied the reconstruction of deforming and/or multiple moving objects in a scene to some extent. Regarding the MBSfM problem, they often start by segmenting different motions or factoring the multiple-trajectory matrix and then perform clustering for 3-D reconstruction [3, 17]. Testing the generalization performance of the improved network based on [16] on such a database is worth exploring, as well as investigating the deep learning-based method for clustering.

1.2 Purpose

The primary goal of this master thesis is to investigate the potential of deep learning-based methods, such as the one proposed by Moran et al. [16], to replace traditional geometry-based approaches for Structure from Motion problems. Our focus will be on further improving the method and assessing its generalization capabilities.

To achieve this goal, we will carry out the following tasks:

1. Identify and collect new suitable datasets for testing the performance of the proposed model.
2. Improve the existing model as much as possible, focusing on enhancing its accuracy and robustness.
3. Investigate the generalization properties of the improved model, both for familiar and unseen data.

With respect to Multi-Body Structure from Motion (MBSfM), we plan to:

1. Explore the use of deep learning-based methods for segmenting or clustering point tracks of objects in a scene.
2. If possible, further improve upon the proposed method for more accurate and efficient segmentation or clustering.
3. Test the generalization performance of the improved network, based on Moran et al. [16], on the reconstruction of segmented motions. We hope to achieve satisfactory generalization, regardless of whether the model is fine-tuned or not.

By addressing these objectives, we aim to provide valuable insights into the capabilities and limitations of deep learning-based methods for Structure from Motion and contribute to the ongoing research in this field.

2 Theory

In this chapter, we describe some basic and useful theories of knowledge we will use in the thesis project. We begin with an overview of the network proposed in [16], which forms the basis of our project. The network architecture is shown in figure 2.1.

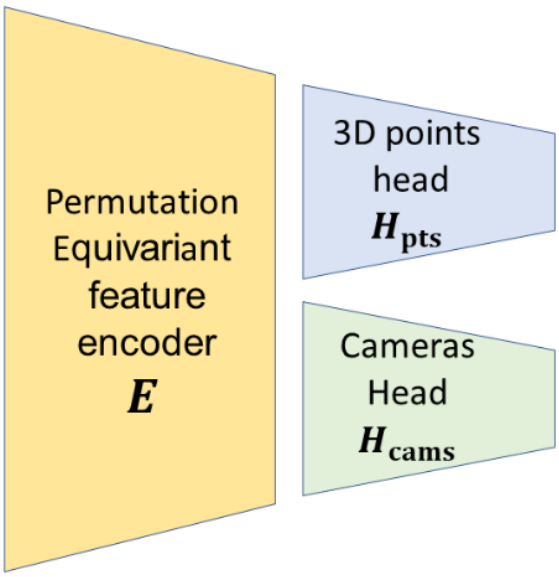

Figure 2.1: The network proposed in [16] is shown here. The first component of the network is the permutation equivariant feature encoder $\boldsymbol{E}$ which is a 3-layered network. Each layer is a linear equivariant layer. The feature encoder is shared by 2 heads. One head outputs the 3D points and the other one outputs the camera matrices.

Figure 2.1 illustrates the neural network architecture that we aim to use and enhance. The feature encoder is concatenated with 2 heads. The essential component of both the feature encoder and the 2 heads is a linear equivariant layer, which imposes a constraint on its parameters. The network takes as input an input tensor and directly outputs a 3D point cloud and all camera parameters. More detailed information is included in subsequent sections and chapters.

2.1 Some notations in 3D reconstruction from multiple images

In this section, we introduce several notations commonly used in 3D reconstruction from multiple images. These notations provide a consistent and concise way to describe the mathematical models and algorithms involved in the reconstruction process. Understanding these notations is essential for comprehending the technical details presented in the following sections and chapters.

Assume a 3-D point $\mathbf{X}$, a camera matrix P, a projection $\mathbf{x}$ and a scale λ. They have the following relationship:

$$\lambda \mathbf{x} = P\mathbf{X} \tag{2.1}$$

In the equation above, $\mathbf{X}$ is represented by a 4-dimensional column vector, i.e. $(X, Y, Z, 1)^T$ and $\mathbf{x}$, i.e. $(x, y, 1)^T$, where X, Y, Z are the 3 coordinates of a point in the world coordinate system and x, y the 2 coordinates in an image. P is a 3×4 matrix, which can be factorized as $P = K \cdot [R|t]$, where K is the camera intrinsic parameter, a 3x3 matrix, R a 3x3 rotation matrix which has a determinant 1 and t a translation vector of length 3. In SfM problem, given at least 2 different $\mathbf{x_1}$ and $\mathbf{x_2}$, we can use triangulation method to compute the unique 3-D point $\mathbf{X}$ if the camera matrices are known. This process is a simple demonstration of recovering 3D structure. Figure 2.2 illustrate the triangulation process.

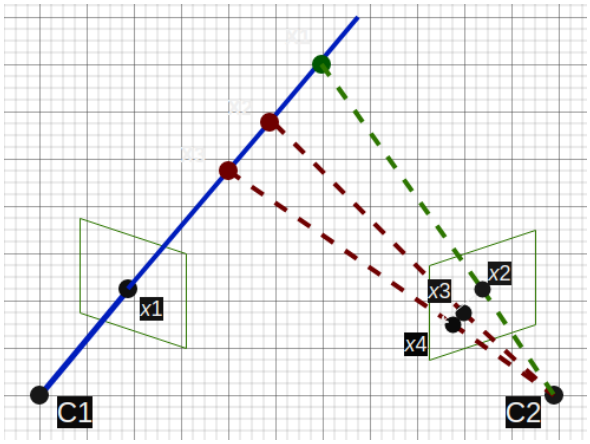

Figure 2.2: The demonstration of the triangulation process. The 2 cameras are known whose camera centers are denoted in C_1 and C_2.

Figure 2.2 illustrates the task of recovering a 3D point. In this figure, X_1, X_2, and X_3 denote three different 3D points, represented by a green point and two red points, respectively. These three points all lie on the blue line. The projection of points on this blue line in camera 1 is x_1, while x_2, x_3, and x_4 are the respective projections in camera 2 of X_1, X_2, and X_3. When relying solely on camera 1, it is impossible to determine the precise location of the 3D point; it could be any point on the blue line, such as X_1, X_2, or X_3. Let's assume X_1 is the true 3D point. When an additional camera (camera 2) is introduced, we draw a green dashed line that includes x_2 and C_2. This green line intersects with the blue line at X_1. This intersection point is unique and represents the exact 3D point we seek.

2.2 Reprojection errors

The reprojection error is a usual, accurate, important and widely-used measure of the accuracy of a 3D reconstruction from multiple images. It is the distance between a 3D point in the world and its corresponding 2D projection in an image, after the

point has been projected back into the image using the estimated camera parameters and 3D model. What we want of 3D reconstruction is to minimize the reprojection error. A smaller reprojection error indicates a better alignment between the 2D image projections and the 3D model, and thus a more accurate reconstruction. It can be computed in the following way:

$$err = ||(\frac{P^1\mathbf{X}}{P^3\mathbf{X}} - x, \frac{P^2\mathbf{X}}{P^3\mathbf{X}} - y)||_2 \tag{2.2}$$

where err is a reprojection error, P^1 the first row of the camera matrix P and so on. $||\cdot||_2$ is 2-norm. The equation means that in order to compute the reprojection error, given projection $\mathbf{x} = (x, y, 1)^T$, we need to compute the reprojection and the distance between the reprojection and projection. In general since there is noise in the projection, the computed camera matrix and the 3-D point are also noisy. Therefore there is often a difference between the projection and reprojection. Figure 2.3 is an example that shows the difference between projection and reprojection. In figure 2.3 we can see the reprojection error clearly. Some reprojected points coincide with measured points while some don't.

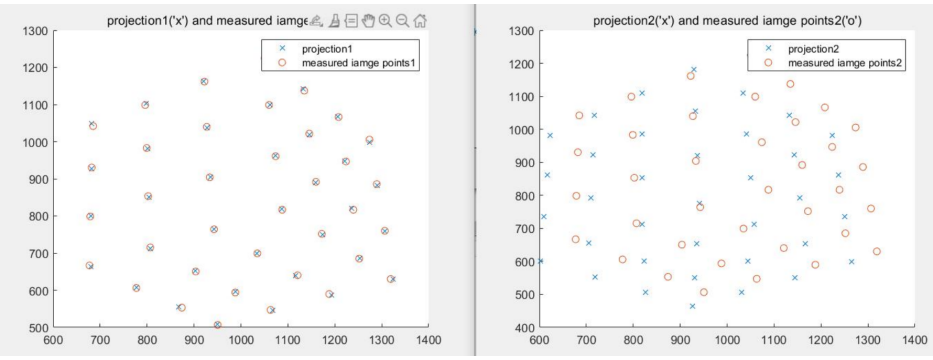

Figure 2.3: A plot of both projection and reprojection. The red circles denote projection(measured by SIFT or other methods), while the blue "x" denotes reprojected 2-D points.

2.3 Deep learning architecture

In [16], they designed the network that comprises of 3 sub-networks, one shared feature encoding and 2 headings for the recovery of camera poses and 3-D structure individually. The neural network architecture is the basis of my project work.

2.3.1 Fully-connected layer

A fully connected layer, also known as a dense layer, is a type of neural network layer in which each neuron in the layer is connected to every neuron in the previous layer. This means that every input feature is connected to every output feature in the layer, and each connection has a weight associated with it.

Fully connected layers are often used in deep learning models for tasks such as classification and regression, where the goal is to predict a label or numerical value based on some input features. These layers can be added to a neural network architecture after a series of convolutional or pooling layers, for example, in order to learn high-level representations of the input data. A figure downloaded from [18] in figure 2.4 demonstrates the fully-connected layers.

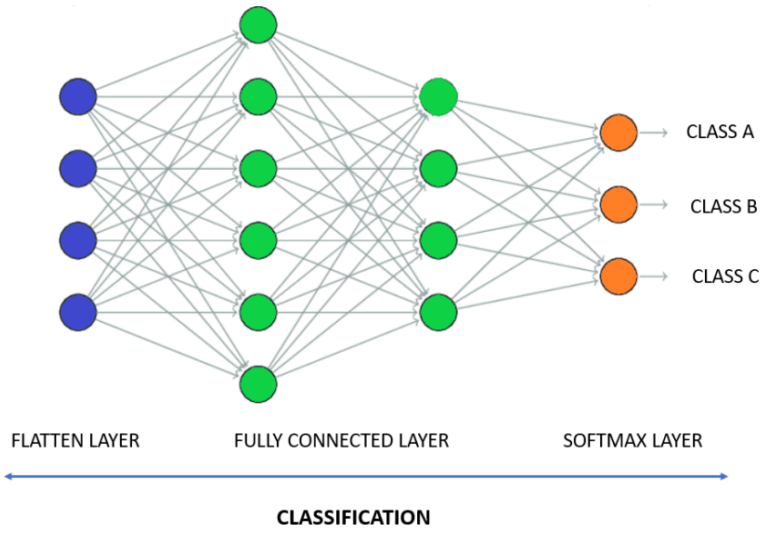

Figure 2.4: A plot of network for classification task that use fully-connected layers

From figure 2.4, we can see that there are an input layer, 2 hidden layers and a output layer (softmax layer) that are fully connected to each other. Therefore each layer is a so-called fully-connected layer.

2.3.2 Permutation equivariant architecture

An equivariant layer is a unique type of fully-connected layer, which employs a parameter-sharing scheme (see [19, 20]) to constrain its parameters. An essential component of constructing equivariant networks is the integration of equivariant layers. In SfM problem, permutation equivariant networks are designed to maintain equivariance with respect to the permutation of the order of images and 3-D points [16]. For example, the order of images changes, the resulting camera matrices should change in the same way. Figure 2.5 that is taken from [16] describes the permutation equivariant property vividly.

In Figure 2.5, the plots at the upper left and upper right illustrate the input measurement tensor M. In these plots, each row represents a distinct image, where points along the row correspond to 2D representations of different 3D points captured in the same image. Conversely, each column represents a specific 3D point, where points along the column correspond to its 2D representations captured in different images. The plots at the bottom left and bottom right illustrate the outputs of the neural network, specifically the camera poses and 3D points respectively. Moving from the upper left plot to the upper right plot illustrates a specific operation, which involves swapping the positions of two rows and two columns. Correspondingly, the output

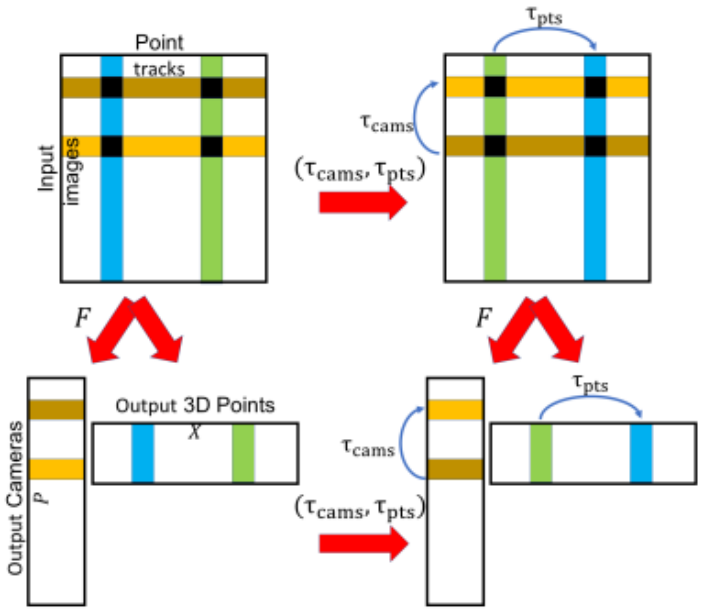

Figure 2.5: Predicting a set of camera positions and 3D points from an input measurement tensor M is equivariant to reordering of the points and the cameras, represented by the pair of permutations $(\tau_{cams}, \tau_{pts})$. The figure is taken from [16]

of the upper left plot also undergoes a similar permutation, resulting in the swapped positions of two rows and two columns as shown in the lower right plot.

To ensure optimal network generalization, it's crucial to incorporate the permutation equivariant property within the layers of the architecture. This property is particularly essential due to the random order of images and 3D points. Therefore, the permutation equivariant architecture, which comprises several permutation equivariant layers, has the distinct advantage of significantly enhancing the model's ability to generalize.

Suppose S_d is the group of permutations on d elements. Then S_m (m corresponds to the number of cameras) can be considered as a specific ordering of cameras. Let n be the number of point tracks and G be the product of S_m and S_n. According to [16], in the case of multiple input and output channels, every G-equivariant affine mapping L can be suitably represented as follows. Assuming the transition from a feature space with d channels denoted as $\mathbb{R}^{m\times n\times d}$ to another feature space with d' channels represented as $\mathbb{R}^{m\times n\times d'}$, the map L can be symbolized as:

$$L(\tilde{M})_{ij} = W_1\tilde{M}_{ij} + W_2\sum_{k=1}^{m} M_{kj} + W_3\sum_{l=1}^{n}\tilde{M}_{il} + W_4\sum_{K=1}^{m}\sum_{l=1}^{n}\tilde{M}_{kl} + \mathbf{b}. \tag{2.3}$$

Here, the input tensor $\tilde{M}_{ij}$ acts as a vector in $\mathbb{R}^d$. The variables W_i are each a member of $\mathbb{R}^{d'\times d}$ for i ranging from 1 through 4, and $\mathbf{b} \in \mathbb{R}^{d'}$ are parameters that can be learned. Then we can see, from equation 2.3, that any G-equivariant affine map L can be factorized as a sum of 4 matrix products, which can be seen as a sum of 4 linear-layered (fully-connected-layered) outputs, greatly reducing the number of parameters

from n^2m^2dd' to $4dd'$.

2.3.3 Activation functions

A fully connected layer can be represented as a matrix multiplication followed by a bias term and an activation function. In general, an activation function is a non-linear mathematical function that is applied to the output of a neuron in a neural network layer. The purpose of an activation function is to introduce non-linearity into the output of the neuron, which allows the neural network to learn more complex patterns and relationships in the input data. Some commonly used activation functions in neural networks include the sigmoid function, the hyperbolic tangent (tanh) function, the rectified linear unit (ReLU) function and the leaky ReLU. The plots of ReLU and leaky ReLU activation functions are in figures 2.6 and 2.7.

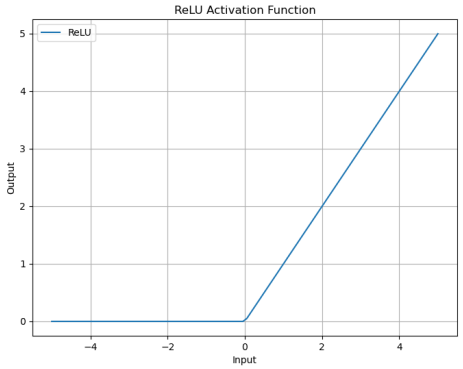

Figure 2.6: The plot of the ReLU activation function

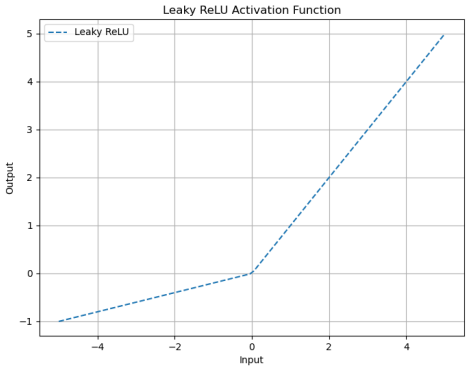

Figure 2.7: The plot of the Leaky ReLU activation function

From figures 2.6 and 2.7, we know that the 2 functions are similar and both non-linear. In their study, Moran et al. [16] utilized ReLU function in the network architecture and argued that any point-wise nonlinearities maintain equivariance to the permutation action. Then it is reasonable to replace ReLU with leaky ReLU if the network gives a better generalization performance when applying leaky ReLU. Leaky RuLU and other functions will be tested later in improving the architecture.

2.4 Residual structures

ResNet, short for Residual Network, is a deep neural network architecture (convolutional neural network, CNN) for image classification, object detection and segmentation, which was first introduced by He et al. [21]. It was designed to overcome the degradation problem in deep neural networks. The degradation problem refers to the phenomenon that increasing the depth of a neural network leads to the accuracy getting saturated and then degrading rapidly. He et al. [21] used such architecture and won the 1st place in the ImageNet classification, detection, and localization competitions in 2015. The key innovation of ResNet is the use of residual blocks, which allow

for very deep neural networks to be trained without the problem of degradation and vanishing/exploding gradient problems. Residual blocks enable the network to learn only the difference between the input and the output of the block, which makes it easier for the network to learn the identity mapping and to maintain the gradient throughout the entire network. ResNet has become a popular and widely used architecture in deep learning research and applications.

The basic building block of ResNet is the residual block, which consists of two convolutional layers with batch normalization and ReLU activation, and a residual connection that adds the input to the output of the second convolutional layer. It is plotted in figure 2.8.

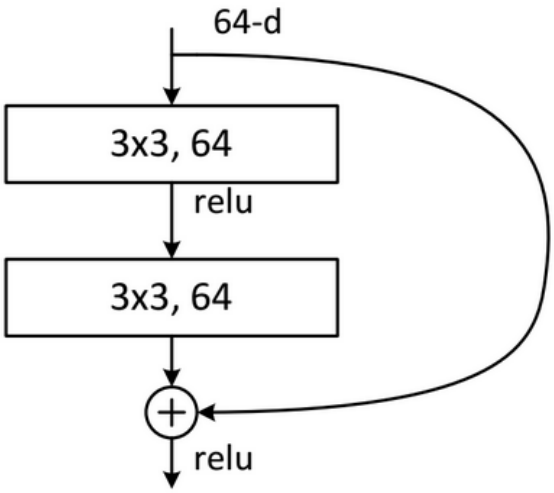

Figure 2.8: A plot of the residual block, the basic building block of Resnet

From figure 2.8 we can see that the difference between the residual block and a normal convolutional block is the shortcut connection (or skip connection) added in the residual block. The skip connection is basically adding the input to the output to extract and learn both global and local features. This idea can be very useful when we design relatively deep architecture.

2.5 The segmentation method

Here we briefly introduce the way and tool of solving the motion segmentation problem. The first tool we use is YOLO (you only look once) version 7 [22]. It is an extremely deep convolutional neural network that is often used to detect and locate objects. YOLOv7 is a more advanced version that gives more accurate location coordinate and classification categories. When it detects an object, it will draw a rectangular box to surround it and assigns a unique identification number to it with the help of a tracking algorithm. An example is shown in figures 2.9 and 2.10.

The second tool is Deep Sort algorithm. Deep SORT (Simple Online and Realtime Tracking with a Deep Association Metric) [23] is an advanced object tracking algorithm that combines both deep learning and Kalman filtering techniques to accurately track

Figure 2.9: The input image with 5 persons in it.

Figure 2.10: The output image of YOLOv7.

moving objects in real-time. It extends the original SORT (Simple Online and Real-time Tracking) algorithm by incorporating a deep association metric to improve the matching of detected objects across video frames. This deep association metric is obtained from a pre-trained neural network that learns to measure the similarity between object appearance features. Consequently, Deep SORT enables more robust and accurate tracking of objects in complex environments and maintains consistent object identities even during occlusions or when objects temporarily leave the frame.

So we use YOLOv7 + Deep Sort to track the object we are interested across a video file. Boxes coordinates are stored. The Hopkins 155 dataset provides us with the unnormalized coordinates (point tracks). To find which motion a specific point track belongs to, we can roughly do it by looking if points are located in boxes. Another way is to set pixels outside bounding boxes to zero and leave the pixels inside boxes unchanged because target objects are inside boxes and we only care about target objects. Then send these cropped images to COLMAP which will be introduced in chapter 3. An example of cropped images is shown in figures 2.11 and 2.12.

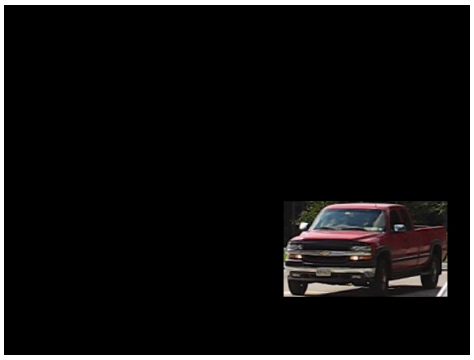

Figure 2.11: The cropped image 1.

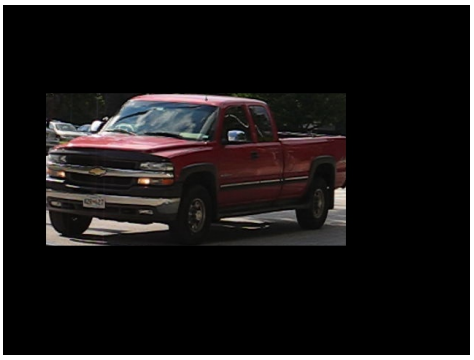

Figure 2.12: The cropped image 2.

Even though we refer to them as cropped images, their sizes remain the same as the original video frames. The only difference is that the pixels we are not interested in are all set to zero.

3 Data and computational resources

In this chapter, we introduce what is the data for the neural network, what it looks like and what new datasets we found and how we did preprocessing, as well as the hardware we have.

3.1 The dataset used in the proposed network

According to [16], the dataset they used are the 39 scans from Olsson and Enqvist [24] and VGG datasets. The datasets consists of both images and point tracks of static scenes. A point track is a set of 2-D points projected from a 3-D point. The 2-D points are the measurements from multiple images. In general, people use SIFT to extract the 2-D feature points and matches and RANSAC to remove outliers [25, 26, 27]. The rest of the points are used to make an input tensor, the exact input for the network.

The tensor size is $m \cdot n \cdot 2$, where m means the number of images or camera matrices, n the number of 3D points detected from a static scene and 2 the 2 dimensions (x and y coordinate). What's more, assuming M is such a tensor, M_{ij} is a 2D point where $0 < i \leq m$ and $0 < j \leq n$. $M_{:j}$ is the j^{th} point track of size $m \cdot 2$ of the j^{th} 3D point. If the j^{th} 3D point is not detected in image i, then set M_{ij} to (0, 0).

Moran et al. [16] used 39 scans which means they made 39 tensors of that kind and input them to a network. The 39 scans Moran provided in his github acount Equivariant-SFM are 39 files for Euclidean Reconstruction and 41 for Projective Reconstruction, both ending with ".npz". Because of the limitation of equipment, we only utilize the files for Projective Reconstruction. In each file, there are 3 individual files which are the tensor named M mentioned above, the matrices named Ns used to normalize 2-D points in tensor M and the camera matrices named Ps_gt denoting the ground truth, respectively. Figure 3.1 shows some images of a scene out of 39.

3.2 New datasets for single and multiple rigid motions

The project is divided into two parts: one focuses on static scenes with a single rigid motion, while the other targets scenes consisting of multiple rigid motions. To enrich the dataset for the proposed network, I have identified two new types of datasets that cater to these distinct scenarios.

Figure 3.1: The plot shows 4 views of a scene from [24], a house on Mårtenstorget. Note that I have merged them into one image.

3.2.1 BlendedMVS dataset: single rigid motion

The new dataset we found are [28] named BlendedMVS. It is a large-scale multi-view-stereo (MVS) dataset for generalized multi-view stereo networks. The dataset contains 17k MVS training samples covering a variety of 113 scenes, including architectures, sculptures and small objects. It seemed to be a bit tricky to download the low-resolution datasets so I downloaded the high-resolution instead. There are no points or point tracks in the dataset, but images, camera parameters and so on.

To prepare the tensor dataset for the neural network, we utilized COLMAP 3.8 installed on an UBUNTU 22.04 system for sparse reconstruction, generating both 2D points and camera parameters. COLMAP was chosen as our primary tool for three key reasons. First, according to [16], the COLMAP algorithm provides a high-quality reconstruction. Second, COLMAP software is user-friendly and capable of exporting reconstruction information in txt file format, facilitating data reading. Lastly, it allows for the comparison of the results with those derived from COLMAP, ensuring a robust and accurate analysis.

The high-resolution images were a bit too large for the laptop to run COLMAP as it always crashed. So we resized all of the images to their half sizes. In this case, we normalized input points in the way described in [29], the same way Moran did for Projective Reconstruction.

To ensure that readers can accurately locate the data we employed, we provide specific details about the dataset utilized. Within the BlendedMVS dataset folder, scenes are not explicitly named. Therefore, we have assigned them names based on their order, such as BlendedMVS_16 (the 16^{th} scene data), BlendedMVS_45 (the 45^{th} scene data),

and so forth. Additionally, considering the considerable number of images in a scene and the constraints of our current GPU, a TITAN X, we selected 21 scene data. The BlendedMVS folder contains three sub-folders, from which we selected 14 scenes from the second sub-folder and 7 scenes from the third one.

3.2.2 Hopkins155 dataset: multiple rigid motions

The second new dataset we use is from [30], called Hopkins 155. We downloaded Part 6, Part 7 and Part 8 from its data page. In these 3 parts, we picked 9 video data in which there are 2 or 3 motions. Along with a video file, there is a ground truth file. The ground truth file contains normalized and unnormalized point tracks x and y respectively, the intrinsic parameter K, motion segmentation ground truth s and so on. In our case, we just utilized the unnormalized point tracks to make input data for the neural network and ground truth to check the segmentation performance of our method. Figure 3.2 shows an example of the scene data.

Figure 3.2: An example of an image containing multiple motions.

In Figure 3.2, the red "+" symbols represent background points, which are associated with one specific rigid motion. Conversely, the points denoted by the green "+" symbols correspond to a different motion. In this particular case, there are two distinct rigid motions in total. It's noteworthy that other datasets may present two or three separate motions.

3.3 Data preprocessing

Upon acquiring the first new dataset, the primarily useful data consisted of images. To process these images to obtain point tracks, we employed COLMAP to extract 2D feature points. One crucial aspect to consider is the tensor size. As we know, the original tensor size is $m \cdot n \cdot 2$. However, when fed into the neural network, the tensor is resized to $2m \cdot n$. To visualize the measurements of an image in Python, you can

extract the coordinates as follows: use "M[0, :]" to obtain the x coordinates of the first image, and "M[1, :]" for the corresponding y coordinates.

3.4 Computational resources and some tools

For implementation and training we used a computer equipped with a TITAN X GPU. With that device, the time cost is around 11 seconds for the code running 100 epochs for a scene data containing 116K 2D points. We preprocessed the scene data and saved them as an npz file on my laptop. When they were done, we transferred them to the remote computer with a TITAN X CPU and edited the code via VI editor.

The basis of deep learning tool I used is the proposed network by Moran et al. [16], which can be downloaded on his github account. The artificial intelligence framework they used is Pytorch. For bundle adjustment, They used the Ceres BA implementation [31].

4 3D reconstruction of BlendedMVS dataset with a single rigid motion

In this chapter, we present the results obtained from testing the network architecture on part of BlendedMVS dataset. Similar to the methodology employed by Moran et al. in their paper, we evaluated the model under two different settings: single-scene optimization and multi-view learning. This rigorous testing approach allows for a comprehensive understanding of the model's performance across a variety of scenarios. Due to hardware limitations and time constraints, in single-scene optimization setting, we ran the code one time for the majority of the datasets. For the datasets that proved to be more challenging, we executed the code two times. The hyper-parameters used in the network training process are as follows:

1. Learning rate: 10^{-4},
2. Network width: 256,
3. Number of layers in the feature encoder (E): 3,
4. Number of training epochs: 10^5.

4.1 Single-scene optimization results

In a typical SfM or 3D reconstruction pipeline, the input is a set of images capturing a single scene from different viewpoints. These images are used to recover the 3D structure of the scene as well as the camera poses. The term "single-scene optimization" refers to the process of optimizing a network, which processes an input scene tensor over many epochs. This reduces the loss and gradually yields accurate 3D point coordinates and camera parameters.

We tested the model on part of the data Yao et al. [28]. The testing result is shown in table 4.1. Note that "B_31" means "BlendedMVS_31" mentioned above.

In table 4.1, the reprojection errors of scenes are shown at epochs 20000, 40000, 70000 and 100000. The time costs are given in seconds and the size of input tensor "M" are also included. "BA_repro" in the table means the reprojection error after the bundle adjustment (BA) and "COL_repro" is the reprojection error given by COLMAP. The boldfaced numbers means the smaller value between "BA_repro" and "COL_repro". The numbers in red here means values exceeding the threshold (2 pixels) and marked as a failure. From table 4.1, we have the following conclusions.

Table 4.1: The testing result on part of BlendedMVS dataset in the uncalibrated setup.

Scan	2e+4	4e+4	7e+4	1e+5	time/s	size of "M"	BA_repro	COL_repro
B_31	10.264	6.382	4.819	5.758	3669	114x6875	0.677	**0.383**
B_32	6.69	3.333	2.319	1.381	6309	134x10243	**0.42**	0.429
B_33	7.565	6.763	6.24	10.081	2633	58x3990	2.324	**0.295**
B_34	111.936	64.863	67.093	93.835	10958	344x20877	6.044	**0.355**
B_35	2.581	1.163	0.608	0.582	3242	58x4772	**0.326**	0.333
B_36	2.493	1.001	0.582	0.548	2370	32x2757	**0.267**	0.272
B_37	3.056	1.26	0.426	0.38	5336	74x9748	**0.281**	0.299
B_38	1.291	0.775	0.602	0.58	6543	66x9760	0.502	**0.46**
B_39	2.53	1.156	0.724	0.7	2488	44x2662	**0.308**	0.314
B_40	11.26	9.47	9.065	9.054	2396	32x3209	3.487	**0.267**
B_42	39.147	4.453	1.728	1.524	4611	146x10252	**0.272**	0.298
B_43	7.794	2.888	1.571	1.44	3652	74x7013	**0.277**	0.285
B_44	23.428	16.316	13.428	13.168	5376	192x12879	2.652	**0.325**
B_45	2.796	1.636	0.889	0.774	11524	132x15031	0.346	**0.339**
B_62	68.689	9.223	3.775	3.134	7720	128x17278	**0.343**	0.357
B_63	24.134	11.218	8.599	8.491	10732	294x23301	**0.38**	0.479
B_64	3.938	2.069	1.477	1.386	2574	68x5324	**0.316**	0.345
B_65	1.231	0.781	0.576	0.563	2368	28x2696	**0.365**	0.415
B_68	10.45	4.943	3.673	3.518	4353	156x9003	0.902	**0.265**
B_69	39.799	22.96	18.511	18.102	6590	182x13087	1.464	**0.36**
B_70	115.135	17.26	15.412	22.314	11188	256x123773	3.132	**0.273**

1. With the epoch increases, the reprojection error often decreases monotonically while in a few cases, it firstly reaches the minimum and then increases or fluctuates.

2. There are 5 failures compared to 0 failure given by COLMAP.

3. In 11(out of 21) cases, the reprojection errors given by the network and BA are smaller than those given by COLMAP.

4. For some data like "B_37", the reprojection error before BA reaches below a pixel, which to some extent means the model is good.

Since the result given by stochastic optimization may vary a bit, we re-ran the code for those failure scans, which are "B_33", "B_34", "B_40", "B_44" and "B_70".

In Table 4.2, "B_34 new" represents the test result obtained during the second run. From the table, it is evident that for the scan "B_34", the outcome varies significantly, with a reprojection error of 1.771, indicating a successful optimization. The results for other scans do not exhibit much variation. Consequently, running the code just once is deemed sufficient.

To conclude, the network, in general, gives a good performance on new data in single-scene optimization in uncalibrated setup.

Table 4.2: The testing result on the failed scans from 4.1 in the uncalibrated setup.

Scan	2e+4	4e+4	7e+4	1e+5	time/s	size of "M"	BA_repro
B_33	7.565	6.763	6.24	10.081	2633	58x3990	2.324
B_33 new	8.845	7.234	6.671	6.665	2509	58x3990	2.842
B_34	111.936	64.863	67.093	93.835	10958	344x20877	6.044
B_34 new	22.466	14.09	10.146	9.473	10971	344x20877	1.771
B_40	11.26	9.47	9.065	9.054	2396	32x3209	3.487
B_40 new	12.723	10.079	9.578	9.478	2396	32x3209	3.487
B_44	23.428	16.316	13.428	13.168	5376	192x12879	2.652
B_44 new	61.792	25.456	19.402	18.897	5377	192x12879	2.309
B_70	115.135	17.26	15.412	22.314	11188	256x123773	3.132
B_70 new	24.986	17.506	16.036	20.341	11566	256x123773	2.845

4.2 Multi-view learning results

Now we test the network in multi-view learning setting on both old and new data. In this setting, Moran et al. selected 23 scans to learn, 10 for training, 3 for validation and 10 for testing. We have 21 new scans and those scans are from 2 groups. Therefore we choose "B_31" and "B_62" for validation, random 9 scans for testing and 10 for training.

Figure 4.1 records the reprojection errors of validation dataset every 5000 epochs when training the network.

Figure 4.1: The reprojection errors of 5 scans from validation dataset when training the network. They are given every 5000 epochs.

In figure 4.1, "mean" denotes the mean reprojection error of the 5 scans. Note that in general the starting reprojection error is over 1000 pixels. From figure 4.1, we can see that the reprojection errors reach hundreds and even dozens of pixels. With the training progresses, the reprojection errors don't decrease much.

Table 4.3 records the evaluation on test set and the result given by Moran is also included as comparison since the output of a network may vary a bit and we used different devices.

Table 4.3: The evaluation on testing dataset and comparison between my result and Moran's.

scan	fine tune	after BA	Moran's after BA
Nijo	44.526	0.39	0.39
Drinking Fountain	88.412	0.284	0.28
Dino 4983	11.661	1.093	1.14
Some Cathedral in Barcelona	110.264	2.441	0.51
Skansen Kronan	36.847	0.411	0.41
Dome	45.173	0.249	1.27
Dino 319	13.649	1.455	1.30
Gustav	76.065	0.157	0.16
Sri Veeramakaliamman Singapore	183.972	8.079	5.45
Alcatraz Water Tower	29.714	1.617	0.47
B_32	203.947	0.45	-
B_33	30.464	2.738	-
B_34	71.478	4.597	-
B_35	123.959	1.517	-
B_36	22.412	0.267	-
B_37	123.694	0.335	-
B_63	175.266	23.394	-
B_64	26.261	1.327	-
B_65	154.916	0.365	-

During the evaluation of test data, we utilized the trained network for inference, fine-tuning (optimizing for an additional 500 epochs), and performed bundle adjustment (BA). Table 4.3 reveals several findings as follows.

1. For the majority of scenes, the learning outcome is satisfactory, with errors below 2 pixels. However, for 5 scenes, the mean reprojection error after BA is slightly higher.

2. Among the 10 shared scenes, the errors exhibit a similar pattern.

3. Interestingly, even if the error after fine-tuning is high (e.g., 203), the error after BA can be significantly reduced (e.g., 0.45).

By defining a success as having an error below 2 pixels and a failure as having an error above 2 pixels, we can observe from Table 4.3 that the failure rate is approximately

26%. The failure rate on the training data is 30%, as shown in Table 4.4, with most failures occurring in the new dataset. In this context, the network's generalization performance is somewhat limited.

Table 4.4: The result of evaluation on training dataset

scan	fine tune	after BA
Corridor	2.53	0.248
De Guerre	12.645	0.219
East Indiaman Goteborg	34.762	0.282
Buddah Tooth	26.941	2.31
Toronto University	7.919	0.503
Sri Thendayuthapani	29.263	0.681
Parg Gate	4.987	0.241
Smolny Cathedral	6.438	0.268
Alcatraz Courtyard	6.033	1.008
Porta San Donato	27.142	0.225
B_38	13.296	0.437
B_39	12.996	1.649
B_40	20.407	2.914
B_42	114.611	3.611
B_43	29.295	4.217
B_44	68.592	0.604
B_45	23.665	4.876
B_68	14.918	0.94
B_69	76.137	6.983
B_70	22.664	0.282

In order to see whether the failure rate results from device and new data, we run the learning code only on Moran's provided data. The corresponding results are included in tables 4.5 and 4.6.

Table 4.5: The result of evaluation on Moran's testing dataset.

scan	fine tune	after BA
Nijo	46.701	0.39
Drinking Fountain	71.053	3.065
Dino 4983	12.959	1.15
Some Cathedral in Barcelona	124.111	2.309
Skansen Kronan	31.616	0.411
Dome	49.535	0.278
Dino 319	17.206	1.458
Gustav	51.437	0.532
Sri Veeramakaliamman Singapore	134.476	7.241
Alcatraz Water Tower	35.399	2.689

From Tables 4.5 and 4.6, we observe that the optimization of "Alcatraz Water Tower" and "Drinking Fountain" failed, which differs from the results in Table 4.3. Nevertheless, the outcomes for most scenes remain similar. In this case, we believe that

Table 4.6: The result of evaluation on Moran's training dataset

scan	fine tune	after BA
Corridor	1.555	0.259
De Guerre	11.741	0.236
East Indiaman Goteborg	2.444	0.346
Buddah Tooth	11.74	2.354
Toronto University	6.034	0.78
Sri Thendayuthapani	30.143	0.276
Parg Gate	8.671	0.698
Smolny Cathedral	5.11	0.295
Alcatraz Courtyard	6.295	0.688
Porta San Donato	12.807	1.089

the failure rate does not stem from the device or new data. Consequently, the network's performance in terms of generalization seems to be somewhat limited within the multi-view learning setup.

When comparing the results produced by our computer with those obtained using Moran's device, a slight difference can be observed. Therefore, it is reasonable to use our results as the baseline for future comparisons with further work in the subsequent chapters.

5 Improvement on neural network structure

In this chapter, we try to improve the network to have a better performance both in single-scene optimization and multi-view learning settings. We firstly add more layers since the original feature encoder layer has only 3 layers. Then we design a architecture similar to ResNet in which people introduce the residual structure to a convolutional neural network that allows for both deeper architecture and better performance.

5.1 Adding 1 and 2 extra layers to the feature encoder layer

The architecture of a network matters a lot. Now we try to improve the proposed network, expecting to get lower reprojection errors. Moran et al. [16] did the hyper-parameter research regarding the learning rate, network width for the encoder E and 2 heads, number of layers in $\{2, 3\}$, the depth threshold h and the std normalization for the layer output. We can see that the depth of the network is not deep. So we firstly add 1 and 2 extra layers to the encoder E. We call the 2 networks "Dense-4" and "Dense-5" because they have 4 and 5 layers in the encoder layer, respectively.

5.1.1 Single-scene optimization results

Firstly we test the new networks in single-scene optimization setting on those scenes that are difficult for the old network to optimize. The result is recorded in table 5.1. And to see if the network has been improved or not, we include the previous result.

In table 5.1, "Dense-3" means the network proposed by [16]; "repro_BA" means the reprojection error after BA. Boldfaced numbers are smaller. "-" denotes missing entries, meaning I just optimize the scene for 40000 epochs. There is a scene named "Buddah Statue" which is hard to optimize but not included in the table because it has many tracks for the computer to allocate enough CUDA memory. "Skansen Lejonet" is also not included in table 5.1. The reprojection errors at epochs 20000, 40000, 70000 and 100000 before BA are assistant metrics of determining whether the network architecture is good or not. The reprojection errors after BA, denoted as "repro_BA" in the table, is the main metric.

From table 5.1, there are 4 successful cases each for 'Dense-4' and 'Dense-5'. With the increase of the number of layers, the time costs increases too. Among these 9 "difficult" scenes, "Dense-5" gives 5 smaller reprojection errors over "Dense-4" and "Dense-3".

Table 5.1: The evaluation result on "difficult" scenes in single-scene optimization setting.

scan	2e+4	4e+4	7e+4	1e+5	time/s	repro_BA	architectures
B_33	9.196	7.234	6.761	6.651	2509	2.842	Dense-3
	10.259	7.047	10.186	9.01	3056	2.994	Dense-4
	7.565	5.978	4.958	4.932	3526	**2.046**	Dense-5
B_40	12.723	10.079	9.578	9.478	2428	2.88	Dense-3
	17.865	9.86	8.463	8.294	2835	3.837	Dense-4
	14.835	5.9	1.795	1.514	3226	**0.253**	Dense-5
B_44	61.792	25.456	19.402	18.897	5377	2.309	Dense-3
	22.86	14.75	10.598	12.855	7131	**1.986**	Dense-4
	23.355	15.651	-	13.261	-	5.018	Dense-5
B_70	24.986	17.506	16.036	20.341	11566	2.845	Dense-3
	19.991	14.28	12.902	12.948	14339	2.983	Dense-4
	18.869	13.365	10.771	11.023	18953	**1.955**	Dense-5
Folke Filbyter	29.063	19.183	-	-	2335	3.964	Dense-3
	17.816	12.298	-	-	2930	1.324	Dense-4
	18.721	15.699	-	-	3689	**0.877**	Dense-5
GustavIIAdlof	19.261	18.582	17.303	17.375	3859	**5.502**	Dense-3
	19.87	18.886	18.697	18.74	5068	5.902	Dense-4
	20.101	19.438	19.178	18.827	6178	5.989	Dense-5
Jonas Ahlstromer	17.572	15.737	15.224	15.225	2423	4.496	Dense-3
	17.118	15.278	14.45	14.644	2834	**0.185**	Dense-4
	17.28	13.933	13.287	12.918	3303	4.855	Dense-5
Urban II	25.935	21.349	19.435	19.518	14682	6.909	Dense-3
	20.15	17.336	16.711	17.846	19909	7.594	Dense-4
	20.502	21.098	20.332	22.046	26542	**6.522**	Dense-5
Tsar Nikolai I	13.502	10.43	-	-	11270	1.877	Dense-3
	13.839	8.097	-	-	14448	**1.655**	Dense-4
	18.068	7.935	-	-	18729	1.665	Dense-5

To conclude, in single-scene optimization setting, adding 1 or 2 extra layers does improve the performance of the network.

5.1.2 Multi-view learning results

Next we test "Dense-4" and "Dense-5" in multi-view learning setting where the network first learns things from the train set and then gives predictions (fine_tune and BA) of test scenes. The result is recorded in table 5.2 and the testing result of previous architecture is also included to be compared with.

Note that the results are based on running the multi-view learning code for only 20000 epochs partly because 21 new scenes were added to the dataset and the running process took much time and partly because from many experiments we found that the reprojection error after learning 10 epochs and after BA was still very small in most cases.

Table 5.2: The evaluation on test set and comparison between the result of the new architectures and of the old architecture.

scan	Dense-3		Dense-4		Dense-5	
	fine tune	after BA	fine tune	after BA	fine tune	after BA
Nijo	**44.526**	**0.39**	53.889	**0.39**	54.043	**0.39**
Drinking Fountain	88.412	0.284	**24.217**	**0.28**	31.409	**0.28**
Dino 4983	11.661	1.093	**9.07**	1.039	9.294	**0.955**
Some Cathedral	110.264	2.441	104.809	**0.761**	**98.663**	4.781
Skansen Kronan	36.847	**0.411**	26.56	**0.411**	**25.766**	**0.411**
Dome	45.173	**0.249**	117.958	0.483	**35.331**	0.325
Dino 319	13.649	1.455	11.845	1.348	**11.46**	**1.293**
Gustav	76.065	0.157	39.78	**0.156**	**24.785**	**0.156**
Sri Veeramakaliamman	183.972	8.079	126.487	9.978	**103.263**	7.163
Alcatraz Water Tower	**29.714**	1.617	30.496	**1.599**	30.006	2.498
B_32	203.947	**0.45**	**86.969**	0.643	89.709	0.504
B_33	30.464	2.738	29.389	2.125	**27.364**	2.154
B_34	71.478	4.597	**64.322**	2.383	68.038	2.323
B_35	123.959	1.517	123.155	**0.399**	**77.096**	3.789
B_36	22.412	**0.267**	17.392	**0.267**	**15.688**	**0.267**
B_37	123.694	0.335	83.608	5.738	**81.229**	**0.281**
B_63	175.266	23.394	**169.686**	19.822	205.49	16.762
B_64	26.261	**1.327**	**22.933**	1.566	23.456	1.353
B_65	154.916	**0.365**	**99.244**	**0.365**	1125.586	**0.365**

From table 5.2, we can see that firstly there are 5, 5 and 7 failed cases for "Dense-3", "Dense-4" and "Dense-5" respectively. Then "Dense-4" and "Dense-5" often give better starting solution, which means the reprojection errors after fine-tuning are smaller. For reprojection errors after BA, there are 7 cases that "Dense-3" gives the smallest numbers and 10, 12 cases for "Dense-4" and "Dense-5" respectively.

To conclude, in multi-view learning setting, adding 1 or 2 layers to the feature encoding layer improves the performance of the proposed network.

5.2 Introducing the residual structure to the feature encoder layer

Since the limitation of time and equipment, we didn't try to find how many layers gives the best performance by adding 1 layer each time to the encoder layer. We know that with the number of layers increases, the performance of a network will eventually fall down, which is the so-called degradation problem (high training and testing errors). The residual structure is exactly one of the solutions. So in this section, we introduce the residual structure to the encoder layer of the network. We design the encoder layer with the residual structure, which is shown in figures 5.1. As we can see, the layer consist of 2 basic blocks that is comprised of 2 linear equivariant layers. Each

layer like "layer 1" in the figure also contains a Batch Normalization layer and a ReLU layer. Since network shown in figures 5.1 consists of 2 basic blocks, we call it "Res-2" for convenience.

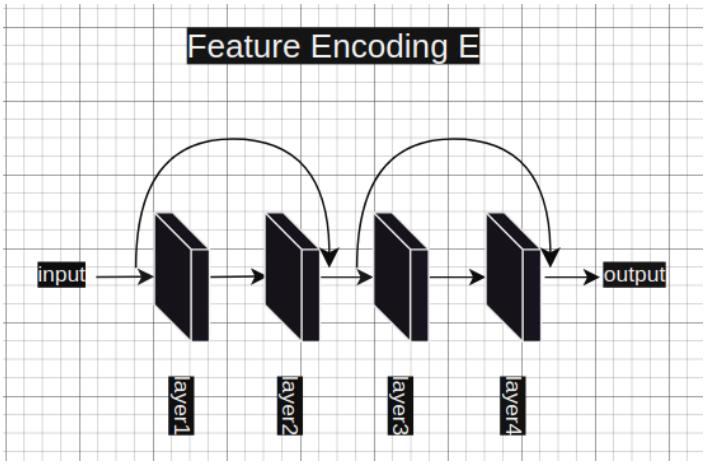

Figure 5.1: The designed architecture of a residual network, adding input to the output of the second layer and adding y_2 to the output of the last layer.

5.2.1 Single-scene optimization results

We test the new network in single-scene setting on those scenes that are difficult for the old network to optimize like before. The result is recorded in table 5.3. For the convenience of making comparison, we include the results of "Dense-4" and "Dense-5". "Dense-3"'s result is not included because the other 2 architectures are better according to **section 5.1**.

From table 5.3 we can see that "Res-2" only gives 1 successful optimization case and 1 smallest reprojection error. The time cost for "Res-2" is larger than that for "Dense-4". So in single-scene optimization setting, "Res-2" is not as good as "Dense-4" or "Dense-5".

5.2.2 Multi-view learning results

Next we test the new network in multi-view learning setting. The result is given in table 5.4. For the result, we only record the reprojection errors after BA because smaller reprrojection error after BA is our ultimate purpose.

In Table 5.4, 'Dense-4' shows somewhat superior performance over 'Res-2' when evaluated from the perspective of failure cases. From the point of smaller errors, there are equal number of cases for "Dense-4" and "Res-2". All in all, "Res-2" performs as good as "Dense-4" in multi-view learning setting.

Table 5.3: The evaluation result on "difficult" scenes in single-scene optimization setting.

scan	2e+4	4e+4	7e+4	1e+5	time/s	repro_BA	architectures
B_33	10.259	7.047	10.186	9.01	3056	2.994	Dense-4
	7.565	5.978	4.958	4.932	3526	2.046	Dense-5
	9.229	43.454	8.001	8.106	3241	**2.029**	Res-2
B_40	17.865	9.86	8.463	8.294	2835	3.837	Dense-4
	14.835	5.9	1.795	1.514	3226	**0.253**	Dense-5
	22.303	10.785	10.751	10.825	2959	1.84	Res-2
B_44	22.86	14.75	10.598	12.855	7131	**1.986**	Dense-4
	23.355	15.651	-	13.261	-	5.018	Dense-5
	18.468	13.134	11.326	10.819	7713	3.114	Res-2
B_70	19.991	14.28	12.902	12.948	14339	2.983	Dense-4
	18.869	13.365	10.771	11.023	18953	**1.955**	Dense-5
	21.71	16.94	18.872	14.004	15951	3.244	Res-2
Folke Filbyter	17.816	12.298	-	-	2930	1.324	Dense-4
	18.721	15.699	-	-	3689	**0.877**	Dense-5
	14.392	12.874	-	-	3109	1.489	Res-2
GustavIIAdlof	19.87	18.886	18.697	18.74	5068	**5.902**	Dense-4
	20.101	19.438	19.178	18.827	6178	5.989	Dense-5
	22.44	19.879	19.283	19.228	5398	5.973	Res-2
Jonas Ahlstromer	17.118	15.278	14.45	14.644	2834	**0.185**	Dense-4
	17.28	13.933	13.287	12.918	3303	4.855	Dense-5
	16.961	15.33	14.194	14.026	2920	3.808	Res-2
Urban II	20.15	17.336	16.711	17.846	19909	7.594	Dense-4
	20.502	21.098	20.332	22.046	26542	**6.522**	Dense-5
	29.888	53.116	34.667	40.768	21816	8.053	Res-2
Tsar Nikolai I	13.839	8.097	-	-	14448	**1.655**	Dense-4
	18.068	7.935	-	-	18729	1.665	Dense-5
	19.209	16.229	-	-	15813	4.699	Res-2

5.3 Test of the best architecture on all scene data

From previous result we can see that "Dense-4" and "Dense-5" are good both in single-scene optimization and multi-view learning setting. Considering the time cost, I think "Dense-4" is better.

In previous experiments we just tested new networks on those "difficult" scenes. Now we test the best network on all data that we found and made from [28]. For the convenience of doing comparison with that of "Dense-3", we copy table 4.1 and paste it here as table 5.5. In table 5.5, we insert a column "Dense-4" after "BA_repro", which is the result of reprojection errors.

From table 5.5, we can see that "Dense-4" is a bit better than "Dense-3" as expected, but not better than "COLMAP" since "COLMAP" always gives good results(errors below 0.5 pixel), even though for some scenes "Dense-3" and "Dense-4" give better results.

Table 5.4: The evaluation on test set and comparison between the results of new architecture and of the old 2 architectures.

scan	Dense-4	Dense-5	Res-2
Nijo	**0.39**	**0.39**	**0.39**
Drinking Fountain	**0.28**	**0.28**	**0.28**
Dino 4983	1.039	**0.955**	1.003
Some Cathedral	**0.761**	4.781	0.95
Skansen Kronan	**0.411**	**0.411**	**0.411**
Dome	0.483	0.325	**0.296**
Dino 319	1.348	**1.293**	1.294
Gustav	**0.156**	**0.156**	**0.156**
Sri Veeramakaliamman	9.978	7.163	5.578
Alcatraz Water Tower	**1.599**	2.498	1.847
B_32	0.643	0.504	**0.435**
B_33	2.125	2.154	3.716
B_34	2.383	2.323	2.535
B_35	0.399	3.789	**0.326**
B_36	**0.267**	**0.267**	**0.267**
B_37	5.738	**0.281**	2.27
B_63	19.822	16.762	36.692
B_64	1.566	**1.353**	1.574
B_65	**0.365**	**0.365**	8.793

Table 5.5: The testing result on part of BlendedMVS dataset in the uncalibrated setup.

Scan	2e+4	4e+4	7e+4	1e+5	time/s	BA_repro	Dense-4	COL_repro
B_31	10.264	6.382	4.819	5.758	3669	0.677	0.385	**0.383**
B_32	6.69	3.333	2.319	1.381	6309	**0.42**	**0.42**	0.429
B_33	7.565	6.763	6.24	10.081	2633	2.324	2.994	**0.295**
B_34	111.936	64.863	67.093	93.835	10958	6.044	2.682	**0.355**
B_35	2.581	1.163	0.608	0.582	3242	**0.326**	**0.326**	0.333
B_36	2.493	1.001	0.582	0.548	2370	**0.267**	**0.267**	0.272
B_37	3.056	1.26	0.426	0.38	5336	**0.281**	**0.281**	0.299
B_38	1.291	0.775	0.602	0.58	6543	0.502	0.502	**0.46**
B_39	2.53	1.156	0.724	0.7	2488	**0.308**	**0.308**	0.314
B_40	11.26	9.47	9.065	9.054	2396	3.487	3.837	**0.267**
B_42	39.147	4.453	1.728	1.524	4611	**0.272**	**0.272**	0.298
B_43	7.794	2.888	1.571	1.44	3652	**0.277**	**0.277**	0.285
B_44	23.428	16.316	13.428	13.168	5376	2.652	1.986	**0.325**
B_45	2.796	1.636	0.889	0.774	11524	0.346	0.346	**0.339**
B_62	68.689	9.223	3.775	3.134	7720	**0.343**	0.35	0.357
B_63	24.134	11.218	8.599	8.491	10732	**0.38**	**0.38**	0.479
B_64	3.938	2.069	1.477	1.386	2574	**0.316**	**0.316**	0.345
B_65	1.231	0.781	0.576	0.563	2368	**0.365**	**0.365**	0.415
B_68	10.45	4.943	3.673	3.518	4353	0.902	**0.253**	0.265
B_69	39.799	22.96	18.511	18.102	6590	1.464	1.338	**0.36**
B_70	115.135	17.26	15.412	22.314	11188	3.132	2.983	**0.273**

6 3D reconstruction of Hopkins 155 dataset with multiple rigid motions

In this chapter, we do 3D reconstruction of the second dataset we found that consists of multiple rigid motions. The content of this chapter can be divided into 2 sections. Firstly, we segment the motions and evaluate the segmentation result. Lastly we reconstruct the object for each motion separately using the best model, "Dense-4".

6.1 Motion segmentation and performance evaluation

The idea of segmentation is that we firstly detect the object (like cars) and locate them with some coordinates. Then based on the 2D point tracks provided by Hopkins 155 dataset, determine if the point is inside some boundaries set up by some coordinates in the first step. For example, if there are 10 points in total are inside a boundary, then the 10 points are considered as being part of the same motion. The other points are attributed to other motions. This method forms the basis of our motion segmentation approach.

In the first step, the tool we used is YOLOV7 [22] (+Deep Sort [23]) which more quickly gives more precise result than previous YOLO versions. When the algorithm detects an object, it will classify it as some category and draw a rectangular box to surround it. The box is mainly based on 4 values, which are the coordinates of 2 points at top left and bottom right. An example illustrating those can be seen in figures 2.9 and 2.10.

However, there are 2 main issues that need to address. We take a video file as an example and give the solutions to corresponding problems. Generally put, the first issue is that the shape of a segmentation ground truth provided by Hopkins 155 dataset is inconsistent with the number of target objects detected by YOLO. The second issue is that there are potential location errors in bounding boxes drawn by YOLO.

The video consists of 25 frames. The motion we would like to reconstruct is a red truck with an id 12. The corresponding point tracks y have a shape of $3 \times n \times 25$. 3 means each point is denoted in homogeneous coordinate like $(x, y, 1)^T$; n represents the number of point tracks and 25 is the number of frames in the video. The first issue is that an object may be classified as different categories. Out of the 25 frames, there are 22 frames in which the red truck is correctly classified. However there is a fact that the id 12 corresponds to 23 objects which means there is an object that is not

classified as a truck but has an id 12. The object we find is the exactly the red truck but classified as a car. The solution to the first issue is that we ignore the category and select those objects with id 12. Then we can have 23 boxes (coordinates). The third dimension of point tracks y is 25, which corresponds to 25 frames in the video file. With some tricks we locate which frame a box is from. After that we remove some data from y and obtain new point tracks y_{new} which has a shape of $3 \times n \times 23$. Note that with YOLO, we can also obtain the coordinates of 23 boxes named $boxes_{12}$ with a shape of 23×4.

What's next is to determine if a point is in a box or not. For example, for $1 \leq i \leq 23$ and $1 \leq j \leq n$, I feed $y_{new}[:, j, i]$ and $boxes_{12}[i, :]$ into a function which measures the distance between the point and the specific box. Algorithm 1 is the pseudo code that illustrates the process in detail.

Algorithm 1 Pseudocode for determining if a point is inside a box

```
counter ← 0
for j = 1 to n do
    for i = 1 to 23 do
        distance ← f(y_new[:, j, i], boxes_12[i, :])
        if distance < threshold then
            counter ← counter + 1
        end if
    end for
    if counter > threshold_count then
        y_new[:, j, i] ← motion1
    end if
end for
```

In algorithm 1, n is the number of points tracks. As can be seen, there are 23 points tracks in the example. For each point track, (1) Looping over the 23 points and 23 boxes, we compute the distance between a point and a corresponding box. If the distance is below some threshold, we consider the point is inside the box. (2) Assume there are 10 points are inside corresponding boxes. we compare 10/23 with another threshold (we call it frequency threshold). If 10/23 is larger than the threshold, the point track is assigned with label 1 or 0 otherwise. Looping over n point tracks, we can assign 1 or 0 to every point track. In the end, those point tracks with label 1 are part of the same motion. Those point tracks with label 0 can be further divided into background and other motions with the same method described above.

After many trials, we set the frequency threshold to 0.95. This high value can solve a problem that in some frames points from other motions are inside the boxes. The next issue is that some points of a car may be outside of the box corresponding to the car because of location errors. To address this, the threshold for the distance between a point and a box is set to 2 pixels. If the distance is below 2, then the point is considered as inside the box. The rules of computing the distance are : (1) If the point is inside the rectangle, the distance is 0. (2) If the point is on the left, right, top, or bottom side of the rectangle, the distance is the horizontal or vertical distance from the point to the rectangle boundary. (3) If the point is outside the rectangle but not on any of the four corners, the distance is the Euclidean distance from the point

to one of the four corners of the rectangle. They can be shown in figures 6.1 - 6.4.

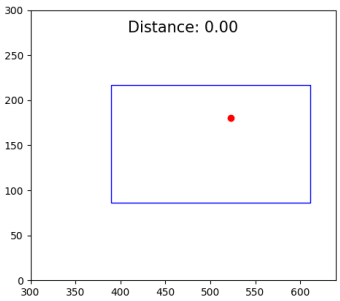

Figure 6.1: A plot shows positions of a point and a box and the distance. The point is inside the box.

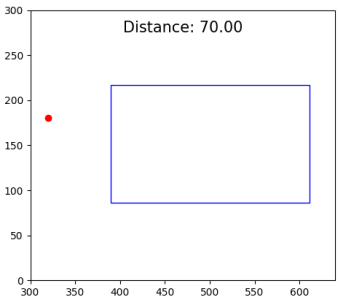

Figure 6.2: A plot shows positions of a point and a box and the distance. The point is on the left side of the box.

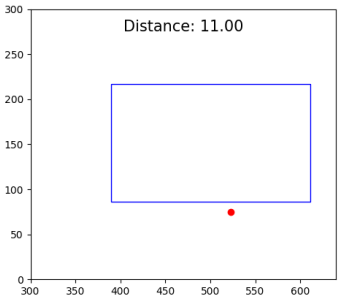

Figure 6.3: A plot shows positions of a point and a box and the distance. The point is under the bottom side of the box.

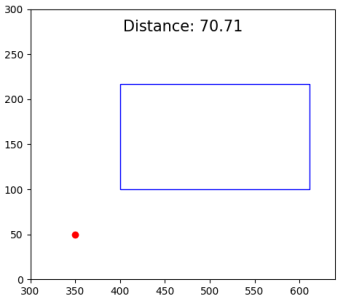

Figure 6.4: A plot shows positions of a point and a box and the distance. The point is on the outside of the box and is closest to the bottom left corner of the box.

After addressing those issues, we obtain a motion segmentation result named s_{ours}. Comparing it to the ground truth s, we have the following levels of accuracy in table 6.1. Note that we selected 9 video files from Hopkins 155 Part6, Part7 and Part8. Each file contains 2 or 3 motions. "Nbr_mo" in table 6.1 means the number of motions.

Table 6.1: The segmentation accuracy results of 9 video files. The first row is the file name.

Name	cars1	cars2B	cars3	cars4	cars5	cars6	cars7	cars8	cars9
Nbr_mo	2	3	3	2	3	2	2	2	3
Accuracy	100%	99.81%	100%	100%	100%	99.78%	99.80%	100%	99.55%

We can see that the smallest value is 99.55% and there are 5 100% accuracy. So this segmentation method is good on these 9 video files. There is an extra advantage for this method that when 2 cars move in the same direction and at the same speed, some

methods may consider them as 1 motion but our method will not make this mistake because 2 cars will be surrounded with 2 boxes and motions are segmented based on their own boxes.

6.2 Reconstruction of multiple motions

Based on the segmentation result in last section, we reconstruct those motions using the best model "Dense-4" described in previous chapters. There are 13 motions (not including background) in total from 9 video files and we pick 6 motions to do 3D reconstruction in both single-scene optimization and multi-view learning settings.

6.2.1 Single scene optimization results

The result is recorded in table 6.2.

Table 6.2: The testing result on part of the segmented motions from Hopkins 155 dataset in the uncalibrated setup.

Scan \ Epochs	2e+4	4e+4	7e+4	1e+5	time/s	after_BA
cars1_motion1	1.106	1.083	1.287	1.226	2662	0.906
cars2B_motion2	0.272	0.277	0.271	0.271	2635	0.223
cars3_motion1	0.358	0.349	0.342	0.342	2565	0.302
cars3_motion2	0.499	0.49	0.48	0.477	2762	0.373
cars5_motion1	0.398	0.397	0.397	0.397	2681	0.365
cars5_motion2	0.345	0.342	0.344	0.344	2679	0.309

From table 6.2, we can see that for all of the 6 motions, the reprojection errors after BA are below 1 pixel, which means single-scene optimization succeeds for all scenes. What's more, the reprojection errors before BA from 20000 epochs to 100000 epeochs are all below 2 pixels. It means that the improved network gives good starting solution for BA for those 6 motions. The reason is mainly because the numbers of 3D points are very small which can be shown by the time cost to some degree. Compared to the time costs in **Chapter 5**, the time costs in table 6.2 are pretty cheap.

6.2.2 Multi-view learning results

In this setting, we add the 6 segmented motions to the dataset and put them all as test dataset, which means the train dataset is the same to that of **Section 5.2**. The result of reprojection errors before and after BA are recorded in table 6.3.

As can be seen, the result for the same test data almost keeps the same to that in **Section 5.2**. As for data from Hopkins 155, the inference result is similar to its optimization result in **subsection 6.2.1**. The reconstruction of data from Hopkins

Table 6.3: The evaluation on test set and comparison between the result of new architecture and of old architecture.

scan	before_BA	after_BA
Nijo	42.765	0.39
Drinking Fountain	30.031	0.28
Dino 4983	10.611	0.978
Some Cathedral	106.743	5.607
Skansen Kronan	27.092	0.411
Dome	42.338	0.562
Dino 319	12.371	1.457
Gustav	49.329	0.157
Sri Veeramakaliamman	113.409	8.845
Alcatraz Water Tower	29.759	1.881
B_32	85.696	0.436
B_33	30.966	2.732
B_34	65.542	2.098
B_35	75.091	1.870
B_36	18.990	0.267
B_37	78.265	3.298
B_63	203.537	25.878
B_64	21.232	1.325
B_65	85.296	1.460
cars1_motion1	1.417	0.904
cars2B_motion2	0.542	0.209
cars3_motion1	0.660	0.290
cars3_motion2	0.676	0.373
cars5_motion1	2.890	0.365
cars5_motion2	0.812	0.309

155 succeeds if error below 2 pixels defines a success. What's more, before BA, the network already gives a good starting solution since the errors are all below 2 pixels.

7 Conclusion

In this chapter, we present our conclusions. We also acknowledge the limitations of our work and suggest potential directions for future research.

7.1 Research objectives

In this section, we revisit the research objectives outlined in the goal document and assess the extent to which they have been achieved during the course of this master's thesis project.

1. **Evaluate the performance of deep learning methods for Structure from Motion (SfM) problems**: The test data was enriched using the BlendedMVS dataset. The project successfully investigated the neural network's ability to provide good starting solutions for bundle adjustment. Comparisons were made with COLMAP, demonstrating that the deep learning method is not as good as COLMAP since it does not always yield a good solution.
2. **Improve network performance and output quality**: Several network architectures were explored, and the network was trained on the enriched dataset, leading to moderate improvements in the network's performance. The quality of the network output was evaluated using reprojection errors, which provided insights into the effectiveness of the proposed method.
3. **Extend the method to moving and deforming objects**: The focus of this part was on moving objects rather than deforming objects. Instead of using a clustering method for motion segmentation, the project employed YOLOv7 and Deep Sort, achieving a high segmentation accuracy of approximately 100%.

In summary, the project made noteworthy progress in addressing the research objectives outlined in the goal document. While some objectives were fully achieved, others warrant further investigation and development. The work presented in this thesis lays a foundation for future research in the field of Structure from Motion, with potential implications for computer vision.

7.2 Limitations and Regrettable Aspects

Despite the successful application and acquisition of a GPU TITAN X, several limitations and regrettable aspects emerged during the course of this 4-month master's thesis project:

1. Single-scene optimization tests in Chapter 5 were performed on a limited number of "difficult" scenes rather than the entire scene dataset.
2. Architectural tests were conducted only in the Projective setting, while Moran's work also explored the Euclidean setting.
3. Additional ideas for network improvement were not explored due to time constraints.
4. State-of-the-art methods from [16] were not tested on any scene data.
5. Multi-view training was conducted for only 20,000 epochs, in contrast to the 100,000 epochs used by Moran.
6. The BlendedMVS dataset, which contains approximately 82 scenes, was underutilized, with only 21 scenes being tested.
7. The majority of test cases were run only once, which may result in variability in the reported results.
8. Deeper residual networks with more than 4 layers were not tested due to computational limitations.

7.3 Future Work

Given the opportunity to continue this research, the following directions could be pursued:

1. Address the limitations and regrettable aspects mentioned above.
2. Perform dense reconstruction and improve the quality of the final model.
3. Design and train a network to directly extract feature points and perform point matching, followed by reconstruction work and potentially replacing bundle adjustment with another network.
4. Investigate traditional projective methods to determine if a network can replace them.
5. Generalize the proposed method for use with deforming objects, expanding the potential applications of the approach.

By addressing these limitations and exploring future research directions, the current work can be further developed and refined, potentially yielding significant advancements in the field.

Bibliography

[1] Hans Wallach and D.N. O'Connell. "The kinetic depth effect". In: *Experimental Psychology* 45.4 (1953), pp. 205–217.

[2] Schonberger Johannes L. and Frahm Jan-Michael. "Structure-from-Motion Revisited". In: *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 4104–4113.

[3] David Novotny, Nikhila Ravi, Benjamin Graham, Natalia Neverova and Andrea Vedaldi. "C3DPO: Canonical 3D Pose Networks for Non-Rigid Structure From Motion". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2019), pp. 7688–7697.

[4] Jianyuan Wang, Yiran Zhong, Yuchao Dai, Stan Birchfield, Kaihao Zhang, Nikolai Smolyanskiy and Hongdong Li. "Deep Two-View Structure-From-Motion Revisited". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2021), pp. 8953–8962.

[5] Xiaodong Gu, Weihao Yuan, Zuozhuo Dai, Chengzhou Tang, Siyu Zhu and Ping Tan. "Dro: Deep recurrent optimizer for structure-from-motion". In: *arXiv preprint arXiv:2103.13201* (2021).

[6] Huan Fu, Mingming Gong, Chaohui Wang, Kayhan Batmanghelich and Dacheng Tao. "Deep ordinal regression network for monocular depth estimation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 2002–2011.

[7] David Eigen, Christian Puhrsch and Rob Fergus. "Depth map prediction from a single image using a multi-scale deep network". In: *Advances in neural information processing systems* 27 (2014).

[8] Jin Han Lee, Myung-Kyu Han, Dong Wook Ko and Il Hong Suh. "From big to small: Multi-scale local planar guidance for monocular depth estimation". In: *arXiv preprint arXiv:1907.10326* (2019).

[9] Chengzhou Tang and Ping Tan. "Ba-net: Dense bundle adjustment network". In: *arXiv preprint arXiv:1806.04807* (2018).

[10] Ziang Cheng, Hongdong Li, Richard Hartley, Yinqiang Zheng and Imari Sato. "One Ring to Rule Them All: a simple solution to multi-view 3D-Reconstruction of shapes with unknown BRDF via a small Recurrent ResNet". In: *arXiv preprint arXiv:2104.05014* (2021).

[11] Benjamin Ummenhofer, Huizhong Zhou, Jonas Uhrig, Nikolaus Mayer, Eddy Ilg, Alexey Dosovitskiy and Thomas Brox. "Demon: Depth and motion network for learning monocular stereo". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 5038–5047.

[12] Xingkui Wei, Yinda Zhang, Zhuwen Li, Yanwei Fu and Xiangyang Xue. "Deepsfm: Structure from motion via deep bundle adjustment". In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16*. Springer. 2020, pp. 230–247.

[13] Yao Yao, Zixin Luo, Shiwei Li, Tian Fang and Long Quan. "Mvsnet: Depth inference for unstructured multi-view stereo". In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 767–783.

[14] Zehao Yu and Shenghua Gao. "Fast-mvsnet: Sparse-to-dense multi-view stereo with learned propagation and gauss-newton refinement". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 1949–1958.

[15] Xiaodong Gu, Zhiwen Fan, Siyu Zhu, Zuozhuo Dai, Feitong Tan and Ping Tan. "Cascade cost volume for high-resolution multi-view stereo and stereo matching". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 2495–2504.

[16] Dror Moran, Hodaya Koslowsky, Yoni Kasten, Haggai Maron, Meirav Galun and Ronen Basri. "Deep permutation equivariant structure from motion". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 5976–5986.

[17] Reza Sabzevari and Davide Scaramuzza. "Multi-body motion estimation from monocular vehicle-mounted cameras". In: *IEEE Transactions on Robotics* 32.3 (2016), pp. 638–651.

[18] IndianTechWarrior. *Fully Connected Layers in Convolutional Neural Networks*. `https://indiantechwarrior.com/fully-connected-layers-in-convolutional-neural-networks/`. Accessed: February 28, 2023. 2023.

[19] Haggai Maron, Heli Ben-Hamu, Nadav Shamir and Yaron Lipman. "Invariant and equivariant graph networks". In: *arXiv preprint arXiv:1812.09902* (2018).

[20] Siamak Ravanbakhsh, Jeff Schneider and Barnabas Poczos. "Equivariance through parameter-sharing". In: *International conference on machine learning*. PMLR. 2017, pp. 2892–2901.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[22] Chien-Yao Wang, Alexey Bochkovskiy and Hong-Yuan Mark Liao. "YOLOv7: Trainable Bag-of-Freebies Sets New State-of-the-Art for Real-Time Object Detectors". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2023, pp. 7464–7475.

[23] Nicolai Wojke, Alex Bewley and Dietrich Paulus. "Simple online and realtime tracking with a deep association metric". In: *2017 IEEE international conference on image processing (ICIP)*. IEEE. 2017, pp. 3645–3649.

[24] Carl Olsson and Olof Enqvist. "Stable structure from motion for unordered image collections". In: *Image Analysis: 17th Scandinavian Conference, SCIA 2011, Ystad, Sweden, May 2011. Proceedings 17*. Springer. 2011, pp. 524–535.

[25] Keju Peng, Xin Chen, Dongxiang Zhou and Yunhui Liu. "3D reconstruction based on SIFT and Harris feature points". In: *2009 IEEE international conference on robotics and biomimetics (ROBIO)*. IEEE. 2009, pp. 960–964.

[26] Zhixin Li and Jie Shan. "RANSAC-based multi primitive building reconstruction from 3D point clouds". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 185 (2022), pp. 247–260.

[27] Mai Xu and Maria Petrou. "Distributed RANSAC for 3D reconstruction". In: *Three-Dimensional Image Capture and Applications 2008*. Vol. 6805. SPIE. 2008, pp. 255–263.

[28] Yao Yao, Zixin Luo, Shiwei Li, Jingyang Zhang, Yufan Ren, Lei Zhou, Tian Fang and Long Quan. "Blendedmvs: A large-scale dataset for generalized multi-view stereo networks". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 1790–1799.

[29] Richard I Hartley. "In defense of the eight-point algorithm". In: *IEEE Transactions on pattern analysis and machine intelligence* 19.6 (1997), pp. 580–593.

[30] Roberto Tron and René Vidal. "A benchmark for the comparison of 3-d motion segmentation algorithms". In: *2007 IEEE conference on computer vision and pattern recognition*. IEEE. 2007, pp. 1–8.

[31] Sameer Agarwal, Keir Mierle and The Ceres Solver Team. *Ceres Solver*. Version 2.1. Mar. 2022. URL: `https://github.com/ceres-solver/ceres-solver`.