6-DoF Relative Camera Motion Estimation from Stereo Views

Using Deep Convolutional Neural Network

Author:	Zihan Dong
Programme:	MSc Robotics&Computation
Supervisor:	Danail Stoyanov
Submitted:	10/09/2018

Abstract

Pose estimation has been an essential problem in machine vision problem for decades. The conventional approaches has achieved remarkable progress but also suffer the limitation on feature extraction, computation efficiency and other dilemmas. The proposed pose estimation CNN is inspired by some early attempts on integrating the deep CNN into real-time visual systems, where in our network, we build an regression net based on the GoogLeNet for feature extraction. The model achieved an ordinary result comparing to some state-of-art models, but still shows some promising potentials on interpreting and recovering geometry relationship.

Contents

Chapter 1 – Introduction

Pose estimation has been an essential problem in machine vision problem for decades. A rich literature could be found studying on the geometry solvers for this problem and some has achieved remarkable progress [1]–[3]. However, most of the conventional methods are highly rely on the correspondence feature points, which has been struggling on the aspect of efficiency [4] and robustness. With the growth of the applications in real-time system, the demand on fast, robust and memory-ease localization or pose recovery algorithm has become even higher. On the other side, in recent decade, every year there are increasing number of neural network based models has been adapted and used to tackle the problems in a wide range. In the field of machine vision, CNN and related network structures successfully cracked many classification[5], [6], detection[7] and segmentation tasks with an impressive accuracy. Attempts has been made since then to accommodate the neural network into real time visual system for fast feature extraction and pose regression[8]–[11], where most of them decide to build their model based on an existing CNN that was originally for classification method. By means of that, they could save massive time by using such CNN as an feature extractor and start training based on the pre-trained and fine-tuned weights.

Our proposed model for relative pose estimation is inspired by [8] and [12]. In our model, we select the GoogLeNet [6] as the main frame for feature selector, where we appended extra structure to process on the intermediate feature map output and perform the regression for rotation and translation separately. We trained our model on the DTU-MVS [13] dataset and evaluate our model with the error in the form of Euclidean distance and angular error. The proposed model achieved a fairly high accuracy on estimating the translation while failed on the rotation tasks. Further improvements could be applied on the model structure and feature map concatenation.

Chapter 2 – Related Works

The estimation of the camera pose is an essential problem in machine vision and has been studied for many years. Extensive applications of pose estimation techniques could be found serving as a key component in visual based systems like Structure from Motion (SfM) and related Simultaneous Localization and Mapping (SLAM) or Visual Odometry (VO)[14] systems. Most of the early studies on this topic is focused on the geometry interpretation of the visual clues. There exists rich literature regarding such traditional approaches and as addressed in [15] most of them tackle the problem by employing various features and forming different optimization problems based on the correspondences found in multi-view images. Nowadays, arising of the neural network concept offers us the opportunity to solve this problem from an inspiring perspective, as the convolutional neural network (CNN) gains its reputation in cracking many machine vision challenges. In this chapter, we will briefly go through the traditional method for pose estimation and those well-known models. Later we will introduce the neural-network based models from the early stage to recent years.

2.1 Traditional Method

Camera pose is the key information to triangulate 3D points and restore geometry relationships between different views in classic machine vision. When this problem was firstly addressed by Longuet-Higgins[16], the proposed algorithm is aimed to deduce relative pose from camera motion. This is basically a 2D – 2D matching and could be done by obtain the Essential Matrix $\boldsymbol{E}$ while applying the *epipolar constrain* to form the relation between views and $\boldsymbol{E}$.

The original estimation problem in [16] could be expressed as a least-square minimization problem shown as

$$\min_{E} \sum_{j=1} {x_2^j}^T E x_1^j$$

then we project onto the essential manifold[17] with the SVD of $\boldsymbol{E}$, such a process is a crucial step as it decomposes $\boldsymbol{E}$ into rotational and translational components and thus recover the relative pose from the view. After been introduced by Philip [5], the 5-point algorithm for Essential Matrix estimation is considered a robust scheme to solve the above equation. Nistér[6] managed to lower the polynomial to 10 degree, since then, many efforts has been made to avoid the high degree of polynomials in 5-point algorithm[7,8]. However, more recent

studies are focusing on forming the unit sphere for parameterizing the image plane[20], while there are also attempts on iteratively solving the Essential Matrix[10,11]. For uncalibrated cameras, the 8-point algorithm is needed to estimate the Fundamental Matrix, it is further improved by Richard I. Hartley [23] with a normalized version against noisy conditions. While point-based algorithm has the advantages in simplicity as stated in [23], it also suffers from the limitation of the number of feature points, under the circumstances when more points are available, a traditional solver is required to solving the least-square estimation.

With the arising of SfM and SLAM system, 3D points are registered globally in the system and this yields the necessity to estimate absolute pose under global coordinates rather than the relative pose between multiple views. Thus, the Features to Point pose estimation technique, which is based on 3D-2D correspondences, has been revised in many feature-based visual systems. The most common algorithm regarding this problem is the Perspective-n-Point (PnP) formulation that was initially defined by Hartley and Zisserman [24], with the homogeneous world-point $p_w = [x\ y\ z\ 1]^T$, corresponding image point $p_c = [u\ v\ 1]^T$ and intrinsic $\boldsymbol{K}$, we can estimate the parameters in $\boldsymbol{R}$ and $\boldsymbol{T}$ by constructing the perspective projection model as

$$sp_c = \boldsymbol{K}[\boldsymbol{R} \mid \boldsymbol{T}]p_w$$

For most of the cases the Random Sample Consensus (RANSAC)[25] scheme will be embedded to improve robustness by rejecting the outliers. With no information of intrinsic, the recovery of pose could be formulated as the P-6-P with 6 correspondences and could be solved using Direct Linear Transformation in [24]. Under certain circumstances when the intrinsic is accessible, this could be further reduced to a P-3-P problem as described in [26].

In more recent studies, Bundle Adjustment(BA) has been introduced for the refinement of camera pose as well as 3D points. This could be seen in various SfM and SLAM systems [27]–[30]. Conventionally, the problem is defined as a least-square problem based on a particularly designed geometry error function, where the Gauss-Newton method, Levenberg–Marquardt method or other general least-square iterative solvers could be applied to obtain the best estimations regarding the point positions and pose parameters. However, this also brought up the problem on computation limitation, as for most SLAM system, the computation cost growth with the system running when new measurement were made every frame [31].

Among all the traditional methods we mentioned above, features and the related correspondences are regarded as an indispensable step to form up the primary conditions for the pose estimation problems. Enormous works have been done to achieve more comprehensive, distinguishable and robust feature extraction. For local feature generation, detectors and descriptors are needed to identify the salient region and characterizing the associations to neighbourhood respectively. A widely accepted extractor consists of Hessian-affine detector[22](perform well on locating scale and affine invariant interest point) combined with the SFIT detector[32] that embeds these evidences based on the local image gradients.

Following a similar perspective, Herbert Bay[33] introduced an efficient feature called "SURF" to enable to enable real-time operation. The Hessian matrix-based detector and distribution-based descriptor in SURF makes it much faster to compute and compare. Other outstanding works includes ORB features [34] that integrates the FAST algorithm for corner detection and oriented BRIEF detector[35], which is faster than SIFT on a orders of magnitude and was proven to be fairly reliable when applied in real-time ORB-SLAM[30] system.

The feature-based visual system is indeed reliable under certain circumstances, the limitations for feature-based method are also exist. As stated by Nathan Piasco in [15], some feature extractors were struggling identify key-points in repetitive structure or textures surfaces. Visual changes for small baseline motions are also quite challenging to distinguish in current SfM method. Besides, the application in real-time system yield the low-cost and high-efficiency algorithm for feature extraction, which could be another weakness for current methods, as some algorithms are considerable computation costly [22] and need extra memory to store previous reference frame and points to perform key-points searching. That is one of the major motivations for us to investigate the Neural Network based method, as addressed in the following section.

2.2 Neural Network based Methods

Studies on camera motion estimation and 3D reconstruction were dominated by the conventional geometry methods for decades, until the recent years, there was a rapid growth in neural network based approaches as a result of the increase in computation capability and emerge of the support to implement parallel computing on Graphical Process Unit and distributed process. As mentioned in previous section, those specifically designed feature extraction methods still suffer from limitations subject to the flaws on model complexity, memory requirement and non-ideal pattern, illuminations and texture. Meanwhile, various deep network models build with Convolutional Neural Network (CNN) architecture have already made some remarkable achievements in image classification tasks [5], [6], [36], their capability of the instinctive feature learning of the convolutional operator between layers has been proven to be very effective over a wide range of visual themes. A year after GoogLeNet [28] made its success on ImageNet Visual Recognition Challenge[37], Alex Kendall adapted it into a camera pose regression network and performed transfer learning using the pretrained weights on majority of the network links. The obtained model is accommodated to several environments and still maintains rather high accuracy. Such an attempt shows the strength on robustness of CNN based model when dealing with dissimilar environment. Later in [38], Alex Kendall has further improved this network by introducing a set of novel loss functions that could leverage the geometry interpretation along with the ability to learn the optimal weights automatically. A similar design of the loss function could be found in [39], where they

attempted to perform image-based registration with the assistant of Riemannian geodesic distance under the Euclidean group **SE(3)**.

At the same period, researchers also found their interests in applying the neural network in real-time system like SLAM and VO for integrated pose estimation and feature extraction. In Keisuke Tateno's work[34], the CNN architecture was embedded into the LSD-SLAM[40] which they embraced as an paradigm to enable the direct semi-dense approach in [40]. The CNN here served as an effective feature extractor (for each input frame) as well as a depth predictor (for each keyframe). Others like Sen Wang[36] and Fei Guo[41] achieved the consensus on adopting the Recurrent Neural Network(RNN) for sequential input frames. Model buildings like such is a innovated but also reasonable approach to dealing with time-sequential data, as the RNN is famous for handling and discovering the relationship over an array of ordered contexts information. They both selected LSTM cells to temporarily register and process the output from previous CNN layers. However, while Sen Wang[36] adopted the VGG-net[36] as the main architecture for CNN branch, Fei Guo[41] decided to implement the PoseNet[38] as the CNN branch, which could be traced back to GoogLeNet[28] as we mentioned earlier.

Currently there does not exists a convincing cross-evaluation for the neural network models mentioned above, in most of their studies the comparison was made only to the conventional geometry model, as there is still a fairly performance gap between the two categories. Nevertheless, from our perspective, the state-of-art model could be the DeMoN model published in 2017 by Benjamin Ummenhofer and Huizhong Zhou [39]. They chose the well-known FlowNet[42] as an starting point to build their network, then they implemented a serial 3-block network architecture with each block an encoder-decoder unit. While the first block is responsible for generating optical flow maps, a novel iterative structure is adapted into the second (middle) block to enable repetitive output refinement. Another refinement block is connected at end to enhance the resolution and accuracy of the raw prediction from first two block. The overall network gives a high precision prediction on both the depth map and pose result. The observable might be the computation cost, as the network has a complex structure with iterative unit, such behaviour somehow levels out the advantage of efficiency of the CNN w.r.t the conventional feature extraction and pose estimation method.

For our network, the proposed network is closely related to the PoseNet[38] as we adopted the GoogLeNet-like structure to serve as the main branches for feature extractor. Additionally, a light-weighted network structure is appended and designed to interpret the intermediate feature map output into relative poses as been done in [41]. The dual-branching architecture was encouraged by [43] and [44], where our model out-performed the model in [43] on some tasks (more detail in section 6.4 Results). We also tried out the Spatial Pyramid Pooling technics that firstly introduced by Kaiming He[45] for compatible input size, but we discarded this design later as this is unnecessary and will slow done the model.

Chapter 3 – Background

In this chapter we will introduce the background theory that was involved in this experiment including the notation we choose for data representation, the camera projection model we used to generate ground truth and also some basis to build the neural network.

3.1 Data Representation

The approaches we have chosen for data representation is closely related to our model design and will have an effect on various aspect like model efficiency, robustness and performance. The input for a typical pose estimation system w.r.t stereo views is an RGB image pair with a small-baseline camera motion. This suggests that we need a 3-channel matrix to store each image where the value in it ranges from 0 to 255. For the output of the system, the pose difference between cameras describes both the translation and rotation occurred during the camera motion. The translation is widely accepted to be addressed in $\mathbf{T} = [t_x,\ t_y,\ t_z]$, which is a straightforward representation of displacement along three axes. However, a range of representations are qualified for the task to describe rotation. We have compared four mostly used representations: Axis-angle, Euler-angle, Rotation Matrix and Rotation Quaternion and decide to use Quaternion under a comprehensive consideration.

Quaternion Such representations for rotation operations has numerous advantages comparing to other representations. It yields 4 parameters to form a valid rotation, which could be expressed as the form $\mathbf{q} = [w|\mathbf{v}]$ with $w = q_w$ for the scale part and $\mathbf{v} = \left[q_x\mathbf{i},\ q_y\mathbf{j},\ q_z\mathbf{k}\right]^T$. Quaternion has the Unlike the 3-parameter representation such as Axis-angle and Euler-angle representation, quaternion is continuous on its domain and thus is crucial when applying for regression tasks. It also yields much less parameters comparing to the continuous Rotation Matrix representation that need 9 constrained parameters to form a valid rotation. In deep network models, a massive number of links need to be constructed within the network along with the constrains for every new output parameters we added. Thus, we decide to use quaternion as a trade-off between complexity and practicability.

Transformation The quaternion representation could be transformed from or back to any rotation forms. Here we only interested in the transformation from orthogonal rotation matrix to the quaternion presentation, which will be needed while extracting information from dataset. For a 9-parameter rotation matrix $\mathbf{R}$ with $\mathrm{r}_{mn} \in \{r_{11} \dots r_{33}\}$, the quaternion could be calculated as shown in equation (2). Here we only present one of the four formulas to form $[q_w, q_x, q_y, q_z]$,

as extra works are needed to resolve the ambiguity problem regarding quaternion representations.

$$\begin{cases} q_w = \frac{\sqrt{1+r_{11}+r_{22}+r_{33}}}{2} \\ q_x = \frac{r_{32}-r_{23}}{4q_w} \\ q_y = \frac{r_{13}-r_{31}}{4q_w} \\ q_z = \frac{r_{21}-r_{12}}{4q_w} \end{cases} \tag{2}$$

Distance between the quaternion is an indispensable concept for the evaluation purpose of model. It is considered the measurement of the closeness between ground truth pose and the predicted poses. However, unlike the norm distance used in translation, we need to obtain an angular value for a direct sense of the error value. This yields the calculation of pose difference in the angle-axis. From the definition of quaternion, we know the properties of quaternion as $\boldsymbol{q^{-1}} = \boldsymbol{q}^*$ where $\boldsymbol{q}^* = [q_w, -q_x, -q_y, -q_z]$ is the conjugate of quaternion $\boldsymbol{q}$. From the definition of rotation, we know that the distance between two rotations matrix could be obtained by $\mathbf{R}_d = \mathbf{R_1}/\mathbf{R_2}$, thus the difference in quaternion follows the similar formation as $\boldsymbol{r} = \boldsymbol{pq}^*$. As the quaternion could be expressed as $(cos\left(\frac{\theta}{2}\right), \boldsymbol{v}\sin\left(\frac{\theta}{2}\right))$, we can expand the $\boldsymbol{pq}^*$ and obtain the following equation:

$$cos\left(\frac{\theta}{2}\right) = p_w q_w + p_1 q_1 + p_2 q_2 + p_3 q_3 \tag{3}$$

Then the retrieve of angular value is a straight forward process by $\theta = 2\arccos(|\langle \boldsymbol{p}, \boldsymbol{q} \rangle|)$.

3.2 Camera Projection

The conversion of camera pose is compulsory when forming the ground truth for relative camera motions. When the camera poses were given in the form of extrinsic rotation and translation alongside the camera intrinsic parameters, the relationship between pixel coordinates and world coordinates could be expressed as:

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

where the f here is the focal length for the camera. The rotation and translation could then be extracted from transformation matrix that was computed from the intrinsic and extrinsic matrix.

3.3 Neural Network Basics

Neuron is the basic structure unit for neural network, each neuron are connected to a range of neighbours where it can process on the input signals received from a set of neurons, then passing it on while the process is finished. Such a process could be expressed in the form of

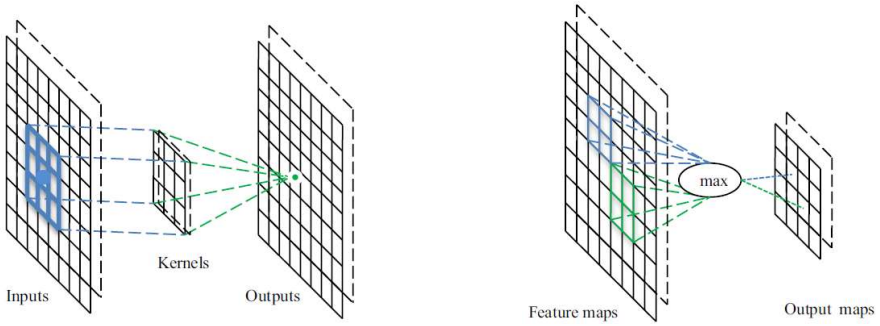


Figure 1 Basic CNN structure (a) Convolution Operator (b) Max-pooling

$$\mathrm{p}_j(t) = \sum_i o_i(t) w_{ij} + w_{0j}$$

where the $input\ p_j(t)$ to the current neuron j is computed from the output of predecessor neuron set $\mathrm{o}_i(t)$. $bias\ w_{0j}$ will be applied to the equation in general cases to prevent saturation and inactivation. This suggests it is possible to describe the network as functions, a widely used composition is the nonlinear weighted sum and usually the different form of activation functions were applied w.r.t different objectives.

However, in CNN the network architecture is slightly more complicated, as we introduced the concept of convolutional kernels to sample the input with convolutional operators, along with the pooling operations for refinement and size reduction. As shown in Figure 1 (a), the kernel first selects window with pre-defined size, then perform the convolution operation with window f and kernel g in the form of

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m] g[n-m]$$

where the kernel g is a set of parameter-controlled 2-d filters based on certain distribution functions. By setting several kernel sizes and arrange the network in different structures, features in the raw image could be learnt in a range of varying scales and distinct perspectives.

Pooling layer is another commonly used structure in CNN. Similar to convolutional operation, it is transversely operated on the input map, but the pooling operator has no parameters and thus only can perform a simple selection based on certain criterion (eg. Max, Min or Mean).

With these basic layers, more complex modules and structure could then be designed serving for more specific purpose. More detailed description of CNN's impressive capability on feature selection could be found in section (4.1 CNN as Feature Extractor).

Chapter 4 – Network Architecture

This chapter describes a comprehensive process to build the proposed model as well as the motivations for all the choices on structure organizing, parameter settings and loss function design. As we also mentioned the model building in section 6.2 Model Implementation, this chapter focus more on the design of the model at the concept level.

4.1 CNN as Feature Extractor

All the achievements the CNN model made on Machine Vision task [5], [6], [36], [46] have demonstrated CNN's impressive capability on feature extraction as we quickly mentioned in section 2.2 Neural Network based Methodsand 3.3 Neural Network Basics. Studies in [47] shows an competitive performance of CNN when comparing to SIFT features. The Flownet[42], however, proved that CNN based could predict the optical flow evidence as accurate as conventional methods. For a more comprehensive comparison, Ali Sharif Razavian[48] studied feature extraction functionality of multiple CNN and tested them on few tasks and discovered that CNN has an astonishing compatibility while being applied on the task that is differ from what it was trained on (this could be done by remove several task-specified layers at the end and replacing them with specific-designed new output layers for different tasks, more about this on section 4.4 PoseCNN). The overview structure of the modified model integrated with baseline method for model comparison in [48] could be found in Figure 2 . A more valuable discovery in [48] that actually benefit our experiment is that, according to the result produced in Ali's experiment, models originally designed for general purpose object attribute detection(labelling) has the best multi-task performance with high robustness, while predictably, those networks trained on categorial specified task has an relatively lower score.

Such results inspired us with the thought of start building our network based on existing CNN models that were that were initially designed for general purpose classification, which is exactly what others were doing [34, 36, 37, 39, 40, 46]. Since the feature is the essential evidence for the pose estimation and camera re-localization problem, researchers already started to investigate the CNN applications in related domains. In the literatures mentioned above, there are several choices on basis network selection (AlexNet[29], VGGNet serials[36], Flownet[42] and GoogLeNet[28]). We then concisely assessed these models, where more comprehensive analysis could be found in this book [50].

AlexNet was firstly introduced by Alex Krizhevsky [29] on LSVRC-2012 competition and is widely accepted as the first deep network that achieved an performance burst on a very large

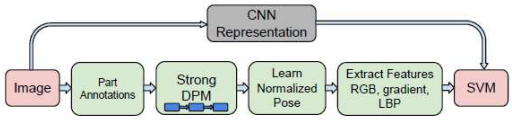
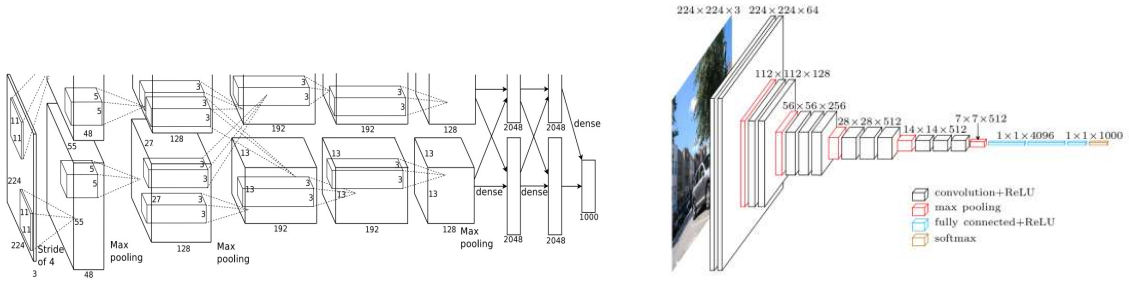


Figure 2 CNN representation replaces pipelines[48]



(a) AlexNet (b) VGG-16

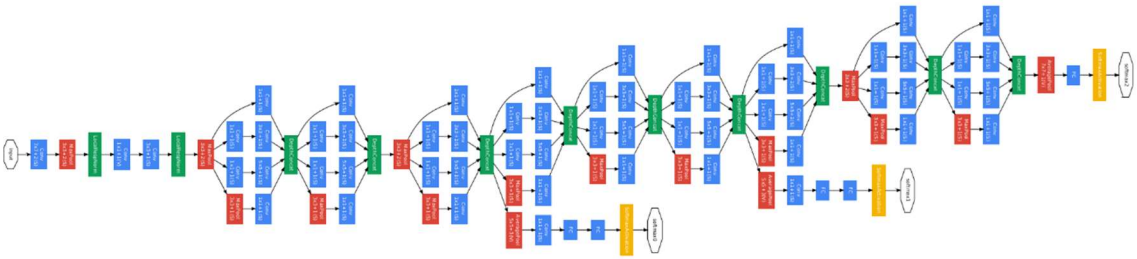


Figure 3 Architecture for three basis network candidates c) GoogLeNet

dataset with various image topics. As shown in Figure 3(a), AlexNet is a 7 hidden layer network with varying kernel size. There are also cross-sampling between the top and bottom branch between second and third layer following with a depth concatenation on the output feature map.

VGG-Net [36] is another CNN that goes even deeper than the AlexNet. The increase of depth has contributed to an improvement on classification accuracy. With a single branch 16 layers architect (19 layers architecture is also available), the proposed model out-performed AlexNet on the ImageNet contest. However, from our perspective, its straightforward structure and less flexible kernel size made it not as beneficial as the following GoogLeNet.

GoogLeNet brought us a drastically different perspective on the designing of CNN model when a group of researchers from Google Inc. [28] published this network and evaluated it on the ImageNet dataset. The novel concept of "Inception Module" and the design for auxiliary output to assist training draws a lot of immediate attention, where peoples were surprised by the ability of GoogLeNet on capturing features from a spanning scale. We finally decided to use this model as our basis network. More details about GoogLeNet could be found in section 4.2 Network Overview and 4.3 GoogLeNet.

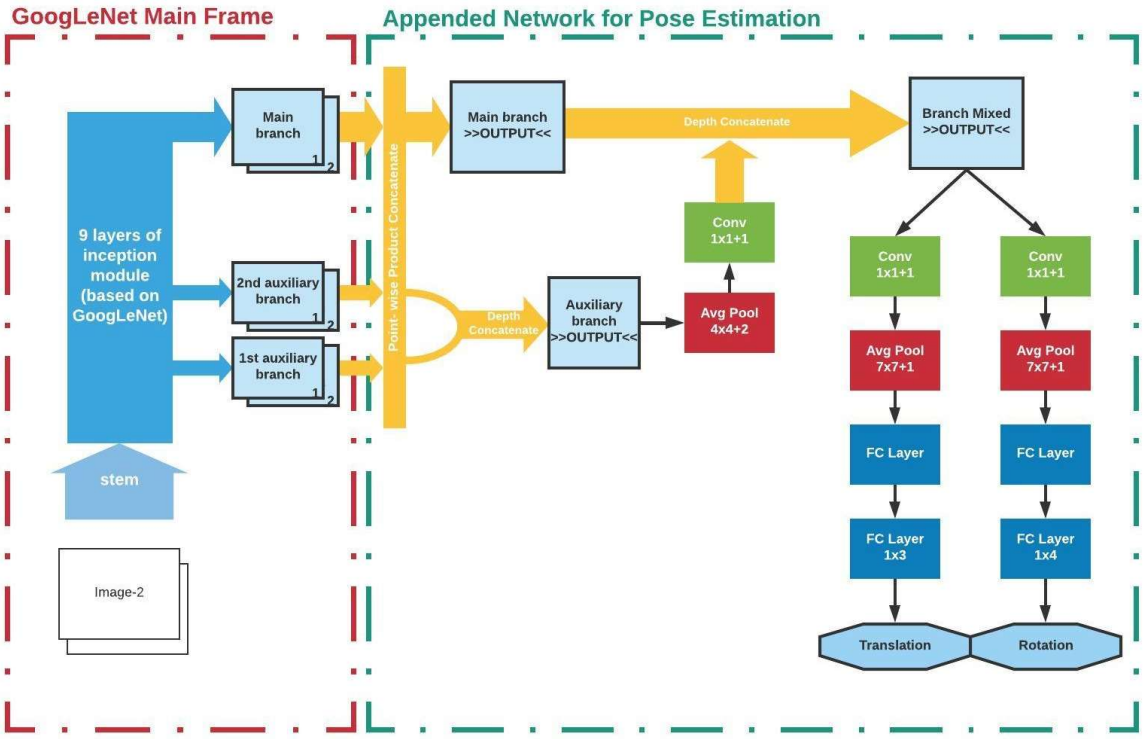


Figure 4 Overview Structure of Proposed Model (baseline version)

GoogLeNet Main Frame									Pose Estimation Network			
Layer-Type	**Patch /stride**	**output map size**	**params num**	**# 1x1**	**# 3x3**	**# 5x5**	**pool size**		**Layer-Type**	**Patch /stride**	**output map size**	**params num**
conv	7x7/2	112x112x64	**2.7K**							depth concatenate		
max	3x3/2	56x56x64						Element-Wise Product Of Stereo	avg pool	3x3/2	7x7x256	
conv	3x3/1	56x56x192	**112K**		192				conv	2x2/1	7x7x256	**321K**
max	3x3/2	28x28x192								depth concatenate		
incep-3a		28x28x256	**159KK**	64	128	32	32		conv	2x2/2	7x7x1024	**788K**
incep-3b		28x28x480	**380K**	128	192	96	64		branch			
max	3x3/2	14x14x480							conv	1x1/1	7x7x1024	**513K**
incep-4a		14x14x512	**354K**	192	208	48	64		avg pool	7x7/1	1x1x1024	
	----------} Aux output 1 (14 x 14 x 128)							↑↑↑	FC		1x1024	
incep-4b		14x14x512	**437K**	160	224	64	64	Element-Wise Product Of Stereo		**------} Rotation (1 x 4)**		
incep-4c		14x14x512	**463K**	128	256	64	64		branch			
incep-4d		14x14x528	**580K**	112	288	64	64		conv	1x1/1	7x7x1024	**513K**
	----------} Aux output 2 (14 x 14 x 128)							↑↑↑	avg pool	7x7/1	1x1x1024	
incep-4e		14x14x832	**840K**	256	320	128	128		FC		1x1024	
max	3x3/2	7x7x832							**------} Translation (1 x 3)**			
incep-5a		7x7x832	**1072K**	256	320	128	128					
incep-5b		7x7x1024	**1388K**	384	384	128	128					
	----------} Main branch output 2 (7 x 7 x 1024)							↑↑↑				
	Total Params:		**5787K**							**Total Params:**		**2135K**

Table 1 Model Parameters Settings

4.2 Network Overview

By adapting and truncating the GoogLeNet as the feature extractor for our network, a set of intermediate output could be produced which representing features from shallow to deep level. Our target was then focused on interpreting these features with the intention of estimating relative pose. Such intention yields the construction of an appended structure that could receive output from predecessor layers and perform regression tasks to generate the poses, which could be seen as shown in Figure 4 . We first perform the depth concatenation on the feature outputs with a proper dimension reduction to blend the three output branches. Then, a dual branch is used for rotation and translation estimation.

This is a baseline model that we initially build by examining similar models and based on our intuition and understanding on proposed hypothesis. The precise parameter settings for this model could be found in Table 1 . In the following section, details will be provided on how we adapted the GoogLeNet and construct the appended network to form a complete CNN for relative pose estimation.

4.3 GoogLeNet

GoogLeNet serves as a good basis network as we addressed in section 4.1 CNN as Feature Extractor. The design of inception module showed great potential on capturing features over a wide span. For each inception module, the features were collected on the scale of $1 \times 1, 3 \times 3$ and 5×5 scale. This was achieved by splitting the main branch into 4 branches where in each branch a specified convolutional layer is placed with different kernel size settings. Note that in one branch among the total four, there is only a max-pooling layer instead of convolution layers. Such an arrangement is explained in [28] as to "take the additional beneficial effect" of the parallel pooling layers, since most of the state-of-art works in early years were benefit from adding pooling layers to the network. After splitted up, results from 4 branches are then concatenated together to form up the final output. The demonstration of singe inception module could be found in Figure 5 (a).

Another innovation design in GoogLeNet is the auxiliary output structures that are aimed to assist the training process. Since the two auxiliary outputs are branched out much earlier than the main branch output, theoretically, the features obtained from these two branches should be shallower or more concrete (comparing to the more abstract final output from main branch). Such design could endow the model with a boosting in understanding and descripting features from different level, and thus increase the robustness when encountered images from drastically differed themes.

Considering that we need GoogLeNet to serve as the feature extractor, we had to truncate the original network by removing all the structure for classification purpose while keeping the rest of the network. More specifically, we discarded the layers after the average-pool layers in main and auxiliary branches as shown in Figure 5 (b) and (c). By doing this we end up with three set of feature maps $F_{feature} \in \{\mathcal{F}_{main}, \mathcal{F}_{aux1}, \mathcal{F}_{aux2}\}$ as shown in Table 1 Model Parameters Settings.

4.4 PoseCNN

Passing the stereo images into the truncated GoogLeNet and performing forward propagation, this will give us two collections of sets of feature maps. We denote the output generated from first image as F_{feat_1} and output from second image as F_{feat_2}, where each output contains $\{\mathcal{F}_{main}, \mathcal{F}_{aux1}, \mathcal{F}_{aux2}\}$ as described above. The intention is to compare as well as combining F_{feat_1} and F_{feat_2} to form the intermediate input for the subsequent of the network. As F_{feat_1} and F_{feat_2} were produced with shared-weight networks, the information embedded in the feature map was expected to be relatively similar. We firstly proposed the "absolute difference" method for map concatenation where the diversity is evaluated on the element-wise level. This could be simply described as:

$$\mathcal{F}_{intermediate\_input} = |\mathcal{F}_{feat\_1} - \mathcal{F}_{feat\_2}|$$

Since the two feature maps have the identical size and depth, the subtraction could be done easily while the produced map will remain the same size. In later experiments, we inspired by [44] where they introduced the element-wise product approach. Ideally, this could amplify those feature pairs that are spatially similar, meanwhile it can also suppress those who are significantly dissimilar. A reasonable explain for such property can be traced back to where we defined the ReLU as our activation function, the inactive area in a feature map is more likely to be close to zero and thus still remain inactive after the element-wise production. We visualized the feature map from a sample output as shown in Figure 6 , we can easily distinguish the dark-blue area in the map that indicates the inactive area with a value close to zero.

The next step is to concatenate the blended set of feature map $\{\mathcal{F}_{main}, \mathcal{F}_{aux1}, \mathcal{F}_{aux2}\}$ into a uniformly sized representation. Many CNN model with branches [37, 48] performed the depth concatenation considering these maps are representing different level of features, where their importance could be learnt by the parameter controlled filters. As shown in Figure 5 (a), the GoogLeNet also uses the depth concatenation between branches in each inception module.

To finally generate the relative poses, we designed a dual branch architecture as shown in the right side of Figure 4 . We have referred to the deduction that concluded by Mohamed

Elhoseiny and Tarek El-Gaaly [52], where in their studies, the experiments conducted on Cross- Product Model(CPM), Late Branching Model(LBM) and Early Branching Model(EBM)

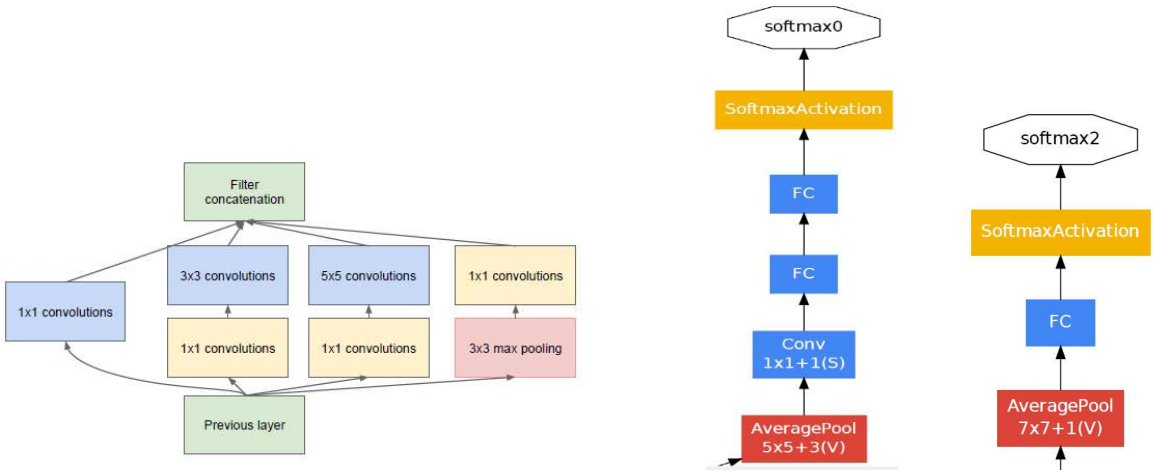
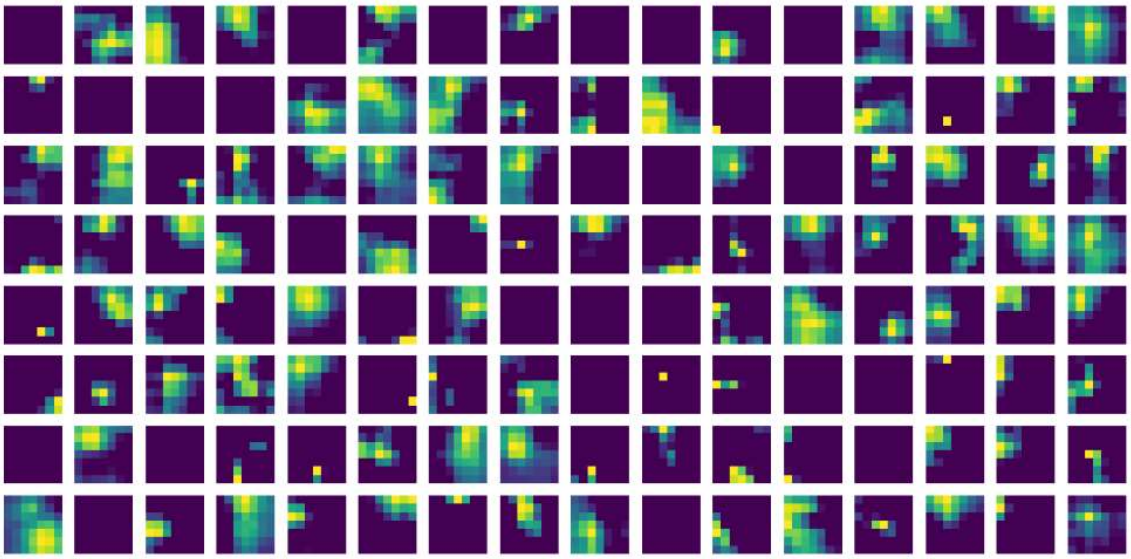
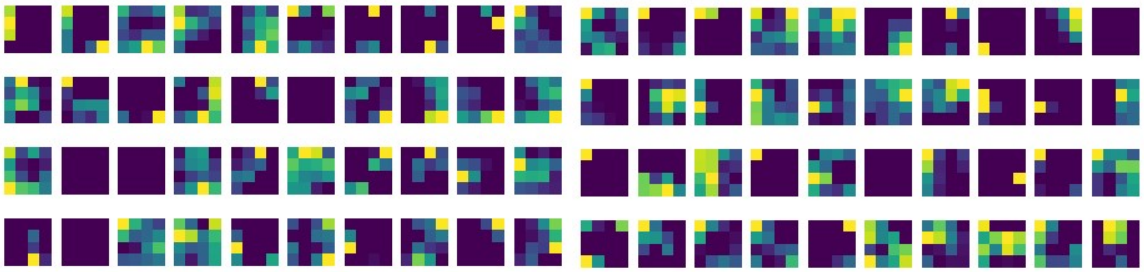


Figure 5 Units in GoogLeNet (a) Single Inception module with Dimension Reduction. (b)Original structure in GoogLeNet for auxiliary output branch. (c) Original structure in GoogLeNet for main-branch output branch



(a) 128 random selected output feature maps from main branch



(b) 40 output feature maps from aux_1 (c) 40 output feature maps from aux_1

Figure 6 Visualized Feature Map of intermediate output from 3 branches.

shows that the EBM perform slightly better than the other two models on a set of datasets. We have took the advantage of this conclusion and designed an early-branching network structure for rotation and translation estimation. Even though the above deduction in [52] was made on category-pose CNN, we thought it is still significant considering the rotation and translation estimation in our model were supposed to rely on distinct features just like in [52]. After applying a convolution layer followed with an average pooling, we first serialized the output with an 1×1024 fully connected (FC) layer, then we applied another 1×3 FC (1×4 for rotation branch) and disable the ReLU to perform pose regression.

4.5 Loss Function

The formation of the loss function will have a direct effect on the performance of the model. A commonly used loss function is the Least-Square loss where it can be seen in [36, 40]. For our cases, we computed the errors on translation and rotation separately and then add the two parts together with a weight control parameter β. To express this mathematically:

$$Loss(I_1, I_2) = \|\hat{x} - x\|_2 + \beta\|\hat{q} - q\|_2$$

Here β is rather an experience-based parameter, as it adjusts the weighting between translation and rotation component. For different uses (eg. indoor/outdoor, city/countryside, slow motion /fast motion), the balance between the two components will also changes. Another problem is related to the quaternion presentation, since by definition, the norm of quaternion $\|q\|$ should always be equal to 1, however, without constrains added (could be regulariser or other form), the output from regressor is very likely to fail this criterion, such issue forces us to adopt the normalized quaternion rotation as described in [38]. The normalized rotation error follows the form of:

$$\text{Error}_{\text{rot}} = \hat{q} - \frac{q}{\|q\|}$$

There are also many other geometric loss functions trying to better interpret the error in 3D space. One prominent loss is the reprojection error that is widely adopted in Visual based system where the map points are feasible in global coordinates. More reasonable loss could be achieved in quaternion domain to enable rotation operations and utilize the properties of quaternion representations, but one intractable for such design is the non-analytic computation as we need to work out quaternion derivatives for back-propagation. Remarkable works has already been done in [50–52] to derive the quaternion gradient, quaternion Gauss-Newton and further Levenberg-Marquardt for Quaternion Neural Network (QNN). However, for current phase, the implementation of QNN is still considerably difficult.

Chapter 5 – Datasets

In this chapter, we will introduce the visual dataset employed for this experiment. The scale of the dataset along with the data format and indexing rules will be precisely addressed in the following section.

5.1 Datasets Overview

There exists vast number of dataset for the evaluation of computer vision method. For camera pose estimation tasks, prevailing choices including stereo set in KITTI benchmark [56], RGB-D SLAM Dataset and Multiview Dataset from TUM [57], DTU Multi-view Stereo Dataset [57] and many others. We have compared the state-of-art datasets mentioned above and put the information in Table 2 Comparison of Datasets. As we can see that, similar to KITTI and TUM, many datasets were generated on a moving vehicle, this usually lead to a sequence with constantly moving camera pose (ground truth). The DTU dataset, however, were prepared for the 3D reconstruction purpose and all the images were taken on a collection of fixed camera poses. As we were doing relative pose estimation, we only need to compute the relative poses between different camera position combination once, and then we can apply the ground truth to all image themes. Additionally, considering we want to implement a robust estimator for an extensive range of topics, the vast categorise in DTU will also benefit our training process despite its lack in image numbers. Thus we have choose to training our model mainly on DTU dataset.

5.2 DTU

The DTU Robot Image Data Sets collection [58] consists three datasets for the purpose of evaluating Computer Vision Methods. In the following experiments, we exploit the "MVS Data Set – 2014" [57] to generate training and testing data and also for further evaluation. In the total 124 categories (each category contains image about one object from a varying views), images for 80 objects were captured from 49 camera positions, while the rest 44 objects were captured with an extra 15 camera positions. All raw images has 1600x1200 pixels.

Dataset	Sequences/Categories	resolution	Frames/Images	Ground Truth
KITTI-stereo	22	0.5M	41K	Yes
TUM RGB-D	22	0.3M	65K	Yes
DTU MVS	124	1.9M	50K	64 Fixed Pose

Table 2 Comparison of Datasets

Chapter 6 – Experiments

The step-by-step procedure for implementing and evaluating the proposed CNN model will be described in this chapter. All the implementations involved in this chapter could be found on our Github repository[1], a brief ReadMe to reproduce the experiment result was attached in the APPENDIX – 1 and also on Github repository.

6.1 Data Pre-process

The raw data in DTU datasets consist of RGB images along with the camera pose as described in 5.2 DTU. Considering that we intend to utilize the Neural Network architecture to estimate the relative camera motion from the image pairs, pre-process of the data is thus necessary to rectify the raw image data and generating ground-truth for the camera motion.

6.1.1 Image pre-processing

The original RGB images from DTU dataset are captured with the size of $1600 \times 1200 \times 3$, for this experiment we need to down-sample the image to $224 \times 224 \times 3$ to fit the GoogLeNet model and for better efficiency. The image is firstly centre-cropped into square size to prevent any distortion problem, then we use the OpenCV library [59] for fast batch image interpolation, where for image shrinking problem, we adopt the "INTER_AREA" interpolation algorithm in OpenCV to produce moire'-free results [60].

After resizing the images, we also perform the data centralization by subtracting the mean from each RGB image to make the data zero-centred. Most of the machine learning algorithms will benefit from the zero-centred input as ReLU and many other activation functions has perceptible nonlinear response near zero point.

6.1.2 Generate Ground Truth

In the DTU-MVS dataset, the ground truths were provided as the extrinsic rotation and translation with camera focal length, principal point offset and the axis skew (here skew=0). We could use the equation in section 3.2 Camera Projection to form the intrinsic and extrinsic matrix and thus obtain the rotation component and translation vector from camera to world coordinates. Since the DTU-MVS dataset has 49 (some has 64) camera positions, this allow us to form different image pairs within a category by selecting two camera positions among the 49 (or 64) registered camera positions. Note that we intended to estimate mainly on the small baseline motion, such

[1] https://github.com/ZihanDong/poseCNN

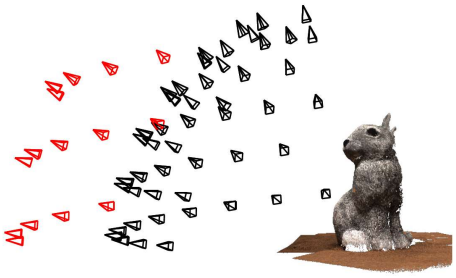


Figure 7 Camera Positions in DTU-MVS dataset

an objective yields an extra constrains while forming stereo image pairs as we need to restrict the difference in rotation and translation within the threshold value. We have tried different settings for threshold, where for translation we simply define the baseline as the normalized Euclidean distance $d = \|x_1 - x_2\|$, for rotation, as the rotation on each axis will have an distinct effect on image, thus we define the baseline by restricting the angular difference on all 3 axes to be lower than the threshold angle respectively. We found that about 1/5 of the camera position pairs in the dataset satisfied such criterion as all the camera positions for DTU dataset were clustered at on side of the object in 3D space as shown in Figure 7 .

6.1.3 Dataset Splitting

As mentioned in section 6.1.2 Generate Ground Truth, we successfully managed to generate the relative pose from the camera poses between two image pairs. To split the dataset into training and testing, what we are essentially doing is generating two set of relative image pairs that containing different camera position combinations. For 49-position categories, we have $C_{49}^2 = 4707$ combinations available, where for 64-position settings, there exists in total $C_{49}^2 = 8064$ possible combinations. Overall, the dataset could produce $80 \times 4707 + 44 \times 8064 \approx 731\text{K}$ ground truth pairs for training and testing (further filtered by small-baseline criterion mentioned above). Practically, we formed around 60K ground truth pairs that were actually used in model implementation and we split them half-half to produce the Train and Testing set. We have deliberately kept a few categories absent from the training set, so the model will have a chance to be tested on unseen image themes.

6.2 Model Implementation

Instead of implementing the network from scratch, we built our network by utilizing the layer wrapper implemented in the tensorflow-posenet repository[2]. Two classes named "GoogLeNet" and "PoseCNN" were inherited from the "network" class and are able to construct two connected computation graph to representing the network structure shown in Figure 4 . Apart from the layer wrapper, we also performed transfer learning by importing the

[2] https://github.com/tensorflow/tfjs-models/tree/master/posenet

pre-trained weights from the repository mentioned above to initialize the GoogLeNet-part of the network. For current version of our model, these weights will be fixed during training as they were already been trained on an much larger and general-purpose dataset.

During the later stage of the experiment, we tried to improve our model by sequentially adding extra 2 set of inception-like unit between concatenation operation and the rotation and translation branches. Such modification has further increased the depth of network as well as enhance the ability on interpreting feature maps theoretically. Contrast of our unit with the naïve-inception module could found in Figure 8 .

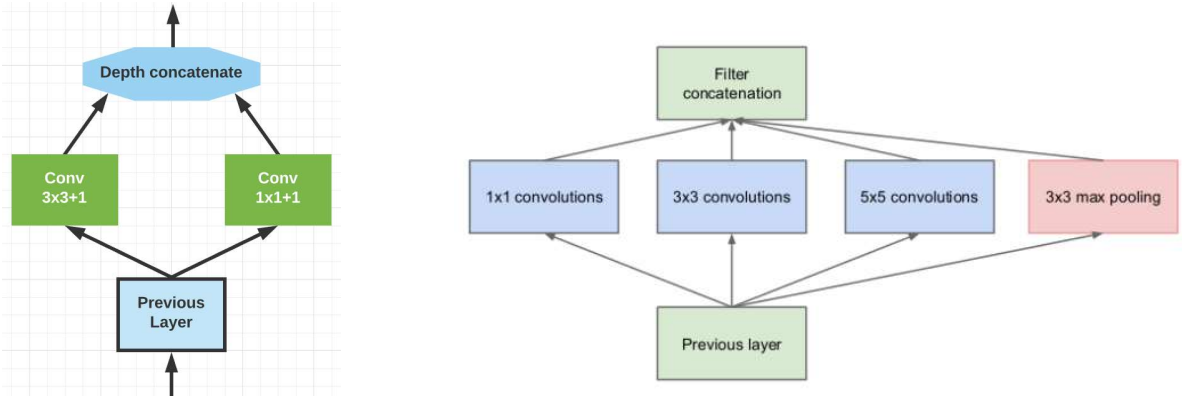


Figure 8 Contrast of our inception-like unit and naive inception

6.3 Training

To load our model and feed the data for training and testing, we implemented a group of python scripts with different training / testing settings. We firstly initialize our network as two connected computation graphs and load all of their parameters into two tensorflow sessions. Then the index of data samples will be fetched along with the ground truth. The data batch will be generated and pre-processed on the fly each time when the model calls for the next batch. A small subset of test set will be passed to the model every 20 iterations to perform validation during the training, where the entire model with all the parameters will be saved every 500 iterations.

We trained the proposed model on our local device using Tensorflow CUDA GPU acceleration on a single graphical card with video memory=8GB and Compute Capability = 6.1 according to the Nvidia CUDA GPU list[3]. Besides, our device was also equipped with a 8 threads 3.4GHz CPU, 6GB of free memory and secondary storage disk with sequential read speed ≈ 170MB/s. We tried 11 different training settings and end up with 12 models as shown in Table 3 (baseline model marked in shallow yellow). Depending on the specific settings, the training time for 5000 iterations varied from 1 hour to 2.5 hours. More analysis on the time efficiency could be found in the section6.4.

[3] https://developer.nvidia.com/cuda-gpus

	"- -" stand for the default setting: beta=10, batch size=100, loss function=norm (q) with β, iterations = 5K, concatenate = elementwise-product						
No.	**Settings**	**beta**	**Batch size**	**Loss function**	**Iterations**	**Concatenate**	**Pose-CNN Model**
1	beta_1/	1	- -	without β		- -	origin model
0	beta_10/	10	- -	without β	10K	- -	origin model
2	beta_100/	100	- -	without β	- -	- -	origin model
3	absolute_diff/	- -	- -	no norm, with β	- -	$\lvert F_{feat\_1} - F_{feat\_2} \rvert$	origin model
4	new_loss/	- -	- -	- -	- -	- -	origin model
5	new_CNN/	- -	- -	no norm, with β	- -	- -	main brach pool - > conv
6	loss_CNN/	- -	- -	- -	- -	- -	main brach pool - > conv
7	deep_CNN/	- -	- -	- -	- -	- -	Add 1 incep-like unit
8	deep_CNN_iter300/	- -	- -	- -	0.3K	- -	Add 1 incep-like unit
9	deep_CNN_iter20K/	- -	- -	- -	20K	- -	Add 1 incep-like unit
10	deep_CNN_batch20/	- -	20	- -	- -	- -	Add 1 incep-like unit
11	deep_deeper/	- -	200	- -	- -	- -	Add 3 incep-like unit

Table 3 Training settings in summary

6.4 Results and Evaluations

In this section we introduces the methods used for model evaluation and the corresponding results after applying to our model. Referring to the training settings shown in Table 3 Training settings in summary, we have trained and tested our model on all 12 different settings using the test dataset, evaluation is performed by group under the control-variable criterion.

6.4.1 Train/Validate Loss

The changings of the defined loss over iterations have been recorded and plotted when training is finished. We selected the result plots from the group of varying β (setting No. 0 - 2) and the varying iteration number (setting No. 7, 8) and presented in Figure 9.

There are 2 motivations for varying the value of beta. The first purpose is to find the best beta that suits the current environment, where the second is to balancing the punishment on rotation term and translation term. As we can see from Figure 9, the translation loss is rather stable in all 3 settings, where the train rotation loss has been suppressed by increasing the beta to 100 as shown in in the bottom right plot. However, the validation loss remains at the same level which suggests that increasing beta will not benefit the learning of rotation parameters.

While training the model with a slightly deeper network (setting No.7), we observed an bounce of the validation loss as shown in Figure 10 , which alerts us on the possibilities of model over-fitting. Thus we have performed the another training on same setting but with max iteration limited at 300 iteration. Evaluations are in the following section.

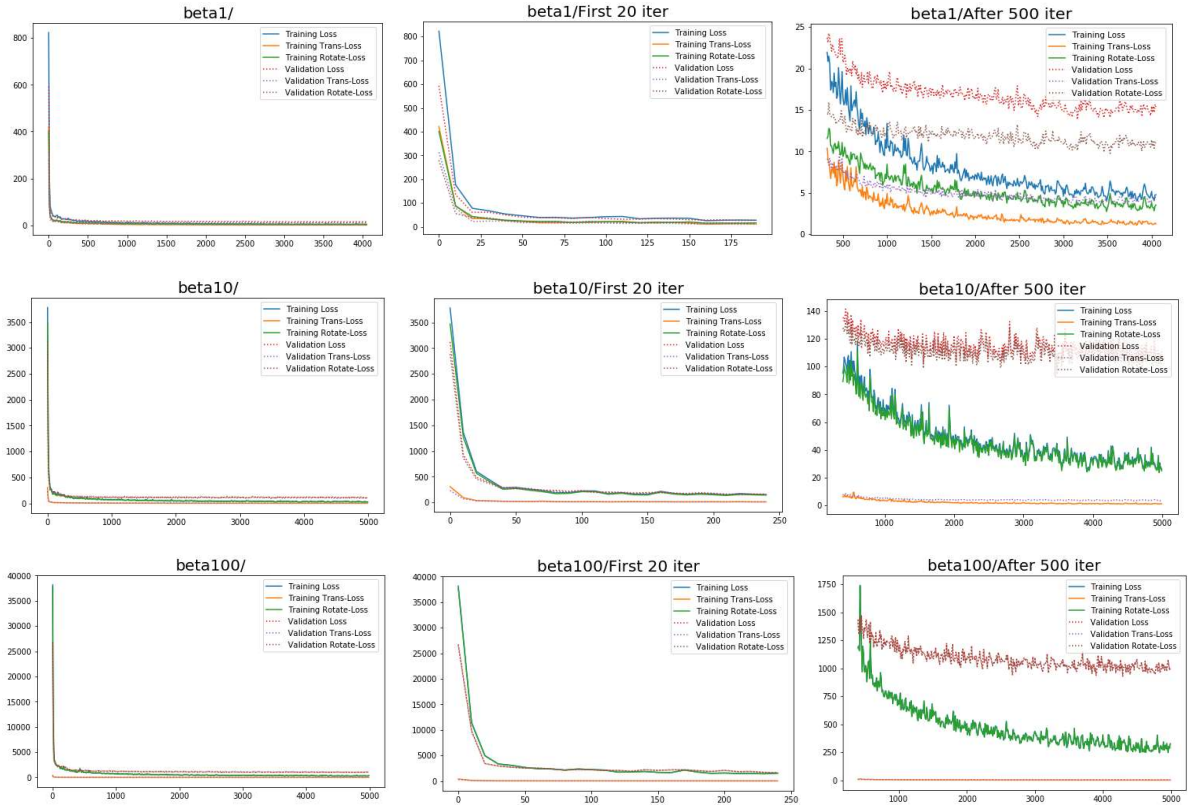


Figure 9 Loss Over Iteration for different beta settings. **top**) beta=1 **middle**) beta=10 **bottom**) beta=100

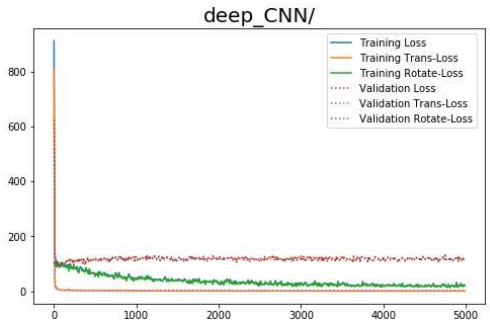


Figure 10 Validation loss bouncing at iter=250

6.4.2 Translation and Rotation Error

We employed the Euclidean distance to evaluate the translation error between ground truth translation and predicted translation. For rotation, we used the angular distance between two quaternions as described in section 3.1 Data Representation. We stored the all the predictions made by the network while we fed the 30K test samples. The error was presented using the Cumulative Graphs so we can obtain a direct sense of the magnitude of the error as well as its distribution. The results were grouped according to different training settings.

Comparison with the baseline model

In Figure 11 , we compared our baseline model with several modified model under different training settings. We found that beta=10 works pretty well when interpreting the relative translation, but none of the varying-beta baseline model has an satisfied result on rotation estimation. Such a result also meets the evidence we found in Figure 9 where we thought that large beta value has no effect on boosting the learning in the rotation components.

An important clues in Figure 11 is that the new loss function with normalized quaternion $q_{norm} = \frac{q}{\|q\|}$ has improved the performance on rotation estimation by a large margin, meanwhile, the model still maintains a similar accuracy on translation estimation. Therefore, we replaces the normalized-q loss for all the experiments afterwards.

Going deeper with extra Inception modules

Candidates in this group all used the same training settings with normalized loss function, where the only difference is on their model structure. The results shown in Figure 12 is quite confusing, we tried to discover some valuable patterns in the plot and made several conclusions:

- Deeper network with convolutional modules will benefit the learning for rotation components, but it will also lead to an accuracy drop in translation estimation, especially on the interval between [0.1m, 0.3m] for our test scene.
- Proposed regression model still struggling on properly descripting the rotation operation (slightly better than the accuracy of random guessing) even after we added 3 extra inception modules. Some other models, however, has achieved much better results with model at similar depth, this suggests there are still some flaws for our model either in feature utilization or the design in network architecture.

6.4.3 Error (loss) by Category

We have classified all the error made by our model into categories that corresponding to the image theme of input images. The results shown in the heat map and the histogram Figure 13 suggests that samples from some of the categories are harder to predict than others. We have manually inspected part of those low accuracy categories and found many images have problems like insufficient illuminations, incomplete patterns or drastically changed views, which have made the prediction much more challenging.

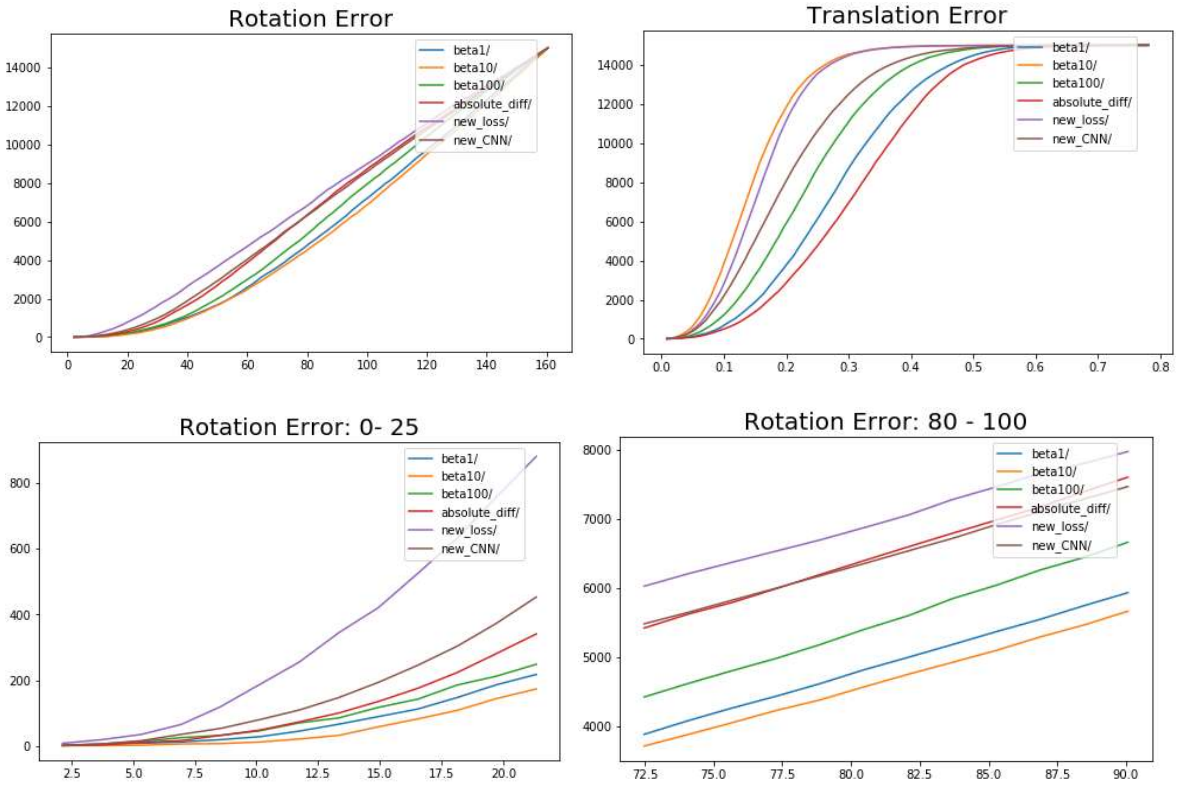


Figure 11 Evaluations on Baseline and related models

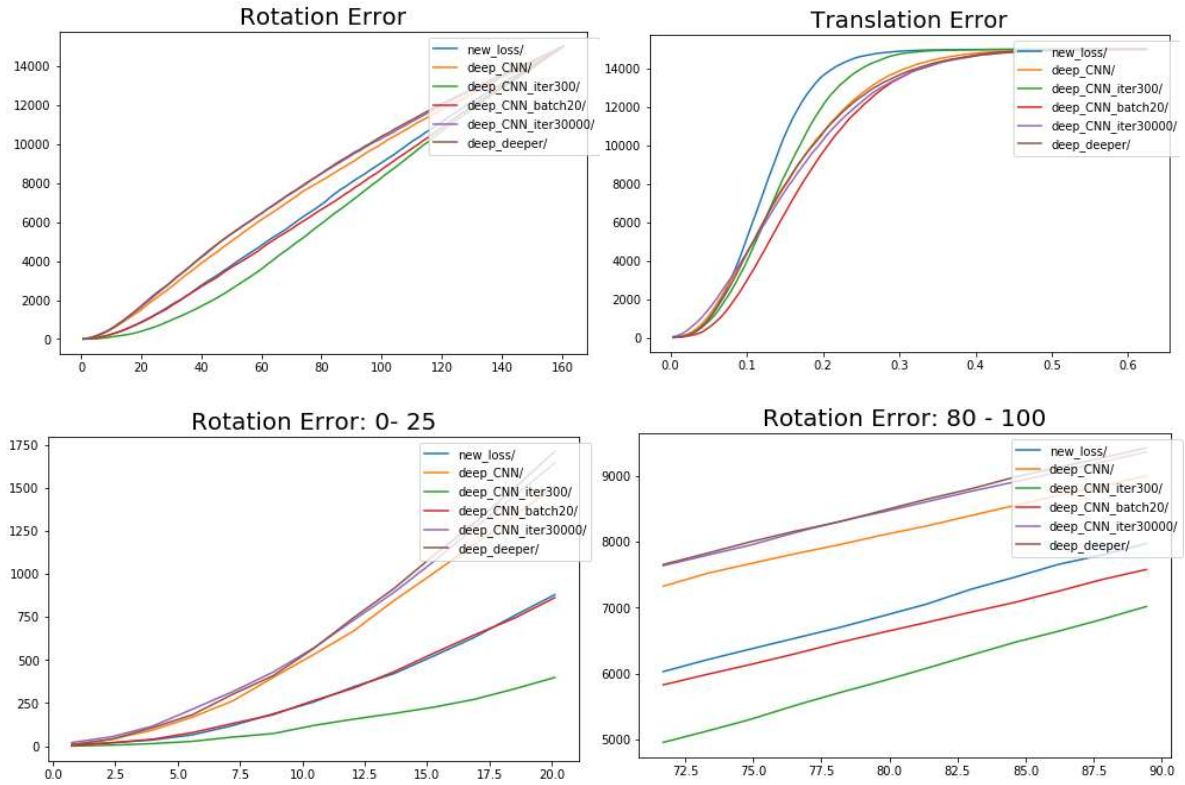


Figure 12 Evaluations on models with deeper architecture

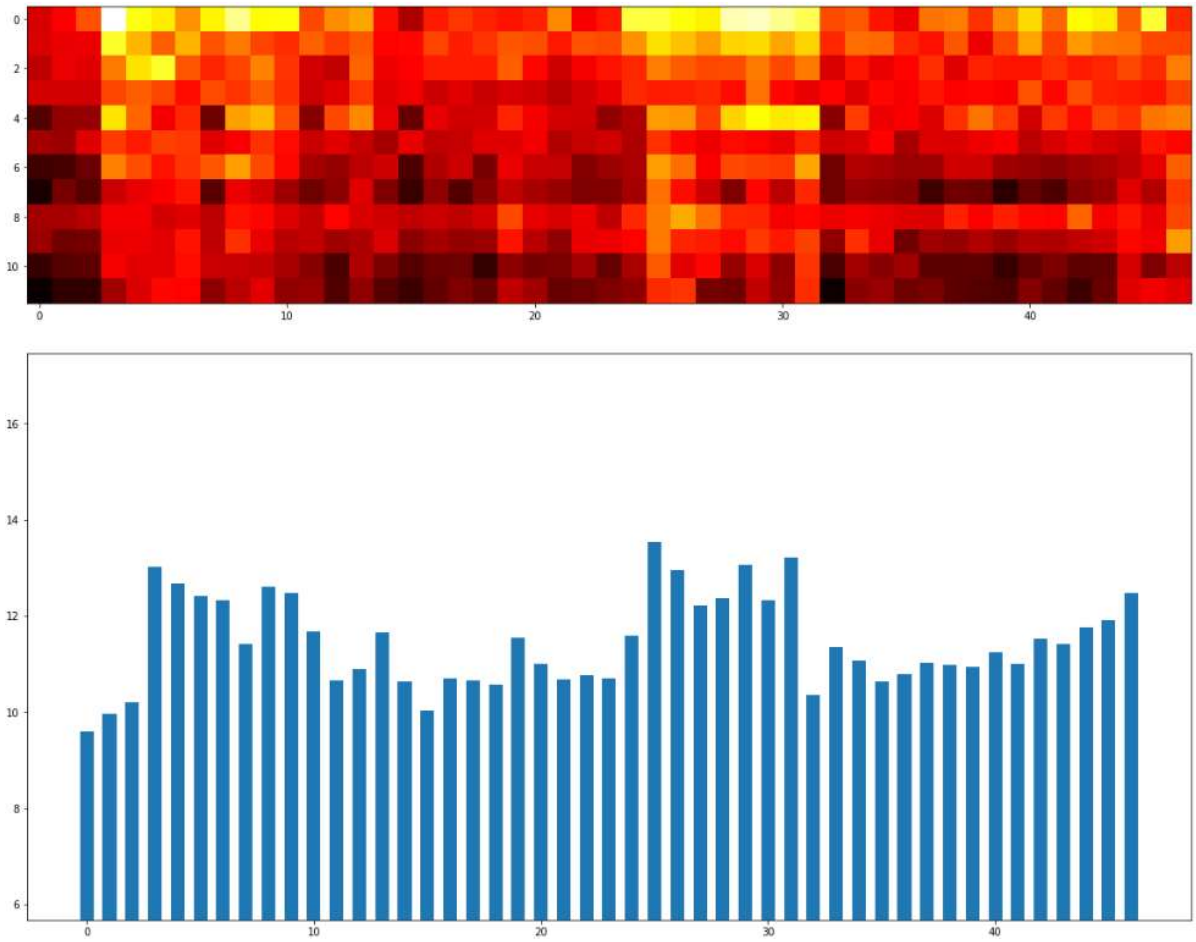


Figure 13 Error (loss) by category. The x-axis is for different categories; y-axis in heat map is for setting numbers; y-axis in histogram is the avg-error made per prediction.

Chapter 7 –Future Work and Conclusion

In this experiment we implemented the GoogleNet based CNN for fast relative pose estimation. We trained our model on the DTU-MVS [13] dataset and evaluate our model with the error in the form of Euclidean distance and angular error. The proposed model achieved a fairly high accuracy on estimating the translation while failed on the rotation tasks. Inspired by the evaluation made in section 6.4.3 and 6.4.2, future works for improve the performance of this network could be focused on truncating the translation branch while extending the rotation branch, as deeper network contributes to rotation accuracy but hazard the translation

interpretation. Multi-thread system could also be attempted as the process for reading images and pre-processing is considerably slow and could be run parallelly. To conclude, this network only achieved part of the functions for what it was expected to do. However, the idea of applying deep learning for visual recognition and geometry interpretation is an innovation and promising perspective to tackle the up coming machine-vision problem.

References

[1] J. Philip, 'A Non-Iterative Algorithm for Determining All Essential Matrices Corresponding to Five Point Pairs', *The Photogrammetric Record*, vol. 15, no. 88, pp. 589–599, Oct. 1996.

[2] Hongdong Li and R. Hartley, 'Five-Point Motion Estimation Made Easy', in *18th International Conference on Pattern Recognition (ICPR'06)*, Hong Kong, China, 2006, pp. 630–633.

[3] D. Nistér, 'An efficient solution to the five-point relative pose problem', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 6, pp. 756–770, 2004.

[4] K. Mikolajczyk and K. Mikolajczyk, 'Scale & Affine Invariant Interest Point Detectors', *International Journal of Computer Vision*, vol. 60, no. 1, pp. 63–86, Oct. 2004.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, 'ImageNet classification with deep convolutional neural networks', *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, May 2017.

[6] C. Szegedy *et al.*, 'Going Deeper with Convolutions', *Google AI*, 2015. [Online]. Available: https://ai.google/research/pubs/pub43022. [Accessed: 24-Aug-2018].

[7] W. Kehl, F. Milletari, F. Tombari, S. Ilic, and N. Navab, 'Deep Learning of Local RGB-D Patches for 3D Object Detection and 6D Pose Estimation', *arXiv:1607.06038 [cs]*, Jul. 2016.

[8] A. Kendall, M. Grimes, and R. Cipolla, 'PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization', *arXiv:1505.07427 [cs]*, May 2015.

[9] B. Ummenhofer *et al.*, 'DeMoN: Depth and Motion Network for Learning Monocular Stereo', *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5622–5631, Jul. 2017.

[10] S. Wang, R. Clark, H. Wen, and N. Trigoni, 'DeepVO: Towards end-to-end visual odometry with deep Recurrent Convolutional Neural Networks', in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 2043–2050.

[11] K. Tateno, F. Tombari, I. Laina, and N. Navab, 'CNN-SLAM: Real-Time Dense Monocular SLAM With Learned Depth Prediction', presented at the Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 6243–6252.

[12] M. S. Müller, S. Urban, and B. Jutzi, 'SQUEEZEPOSENET: Image based pose regression with small convolutional neural networks for real time uas navigation', *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences - International Conference on Unmanned Aerial Vehicles in Geomatics, Bonn, Germany, 4–7 September 2017. Vol.: IV-2/W3*, 2017.

[13] R. Jensen, A. Dahl, G. Vogiatzis, E. Tola, and H. Aanaes, 'Large Scale Multi-view Stereopsis Evaluation', in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Columbus, OH, USA, 2014, pp. 406–413.

[14] D. Nister, 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.*, 2004, vol. 1, pp. I–I.

[15] N. Piasco, D. Sidibé, C. Demonceaux, and V. Gouet-Brunet, 'A survey on Visual-Based Localization: On the benefit of heterogeneous data', *Pattern Recognition*, vol. 74, pp. 90–109, Feb. 2018.

[16] H. C. Longuet-Higgins, 'A computer algorithm for reconstructing a scene from two projections', *Nature*, vol. 293, no. 5828, pp. 133–135, Sep. 1981.

[17] M. Gromov, 'Filling Riemannian manifolds', *J. Differential Geom.*, vol. 18, no. 1, pp. 1–147, 1983.

[18] Z. Kukelova, M. Bujnak, and T. Pajdla, 'Polynomial Eigenvalue Solutions to the 5-pt and 6-pt Relative Pose Problems', in *Procedings of the British Machine Vision Conference 2008*, Leeds, 2008, pp. 56.1-56.10.

[19] H. Stewénius, C. Engels, and D. Nistér, 'Recent developments on direct relative orientation', *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 60, no. 4, pp. 284–294, Jun. 2006.

[20] M. Arie-Nachimson, S. Z. Kovalsky, I. Kemelmacher-Shlizerman, A. Singer, and R. Basri, 'Global Motion Estimation from Point Matches', in *Proceedings of the 2012 Second International Conference on 3D Imaging, Modeling, Processing, Visualization & Transmission*, Washington, DC, USA, 2012, pp. 81–88.

[21] U. Helmke, K. Hüper, P. Y. Lee, and J. Moore, 'Essential Matrix Estimation Using Gauss-Newton Iterations on a Manifold', *Int J Comput Vision*, vol. 74, no. 2, pp. 117–136, Aug. 2007.

[22] V. Lui and T. Drummond, 'An Iterative 5-pt Algorithm for Fast and Robust Essential Matrix Estimation', in *Procedings of the British Machine Vision Conference 2013*, Bristol, 2013, pp. 116.1-116.11.

[23] R. I. Hartley, 'In defence of the 8-point algorithm', in *Proceedings of IEEE International Conference on Computer Vision*, Cambridge, MA, USA, 1995, pp. 1064–1070.

[24] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003.

[25] M. A. Fischler and R. C. Bolles, 'Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography', *Commun. ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1981.

[26] J. A. GRUNERT, 'Das pothenotische problem in erweiterter gestalt nebst bber seine anwendungen in der geodasie', *Grunerts Archiv fur Mathematik und Physik*, pp. 238–248, 1841.

[27] J. Zhang, M. Boutin, and D. G. Aliaga, 'Robust Bundle Adjustment for Structure from Motion', in *2006 International Conference on Image Processing*, 2006, pp. 2185–2188.

[28] R. Kummerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, 'G^2o: A general framework for graph optimization', in *2011 IEEE International Conference on Robotics and Automation*, Shanghai, China, 2011, pp. 3607–3613.

[29] F. Dellaert, 'Factor Graphs and GTSAM: A Hands-on Introduction', p. 27.

[30] R. Mur-Artal, *ORB_SLAM2: Real-Time SLAM for Monocular, Stereo and RGB-D Cameras, with Loop Detection and Relocalization Capabilities*. 2018.

[31] C. Cadena *et al.*, 'Past, Present, and Future of Simultaneous Localization And Mapping: Towards the Robust-Perception Age', *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, Dec. 2016.

[32] D. G. Lowe, 'Distinctive Image Features from Scale-Invariant Keypoints', *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004.

[33] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, 'Speeded-Up Robust Features (SURF)', *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, Jun. 2008.

[34] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, 'ORB: An efficient alternative to SIFT or SURF', in *2011 International Conference on Computer Vision*, 2011, pp. 2564–2571.

[35] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, 'BRIEF: Binary Robust Independent Elementary Features', in *Computer Vision – ECCV 2010*, 2010, pp. 778–792.

[36] K. Simonyan and A. Zisserman, 'Very Deep Convolutional Networks for Large-Scale Image Recognition', *arXiv:1409.1556 [cs]*, Sep. 2014.

[37] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, 'ImageNet: A Large-Scale Hierarchical Image Database', p. 8.

[38] A. Kendall and R. Cipolla, 'Geometric Loss Functions for Camera Pose Regression with Deep Learning', *arXiv:1704.00390 [cs]*, Apr. 2017.

[39] B. Hou *et al.*, 'Deep Pose Estimation for Image-Based Registration', Apr. 2018.

[40] J. Engel, T. Schöps, and D. Cremers, 'LSD-SLAM: Large-Scale Direct Monocular SLAM', in *Computer Vision – ECCV 2014*, 2014, pp. 834–849.

[41] F. Guo, Y. He, and L. Guan, 'RGB-D camera pose estimation using deep neural network', in *2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, 2017, pp. 408–412.

[42] P. Fischer *et al.*, ‘FlowNet: Learning Optical Flow with Convolutional Networks’, *arXiv:1504.06852 [cs]*, Apr. 2015.

[43] I. Melekhov, J. Ylioinas, J. Kannala, and E. Rahtu, ‘Relative Camera Pose Estimation Using Convolutional Neural Networks’, Feb. 2017.

[44] K. Konda, *Learning visual odometry with a convolutional network.* .

[45] K. He, X. Zhang, S. Ren, and J. Sun, ‘Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition’, *arXiv:1406.4729 [cs]*, vol. 8691, pp. 346–361, 2014.

[46] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, ‘OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks’, *arXiv:1312.6229 [cs]*, Dec. 2013.

[47] V. Devi, J. Baber, M. Bakhtyar, I. Ullah, W. Noor, and A. Basit, ‘Performance Evaluation of SIFT and Convolutional Neural Network for Image Retrieval’, *International Journal of Advanced Computer Science and Applications*, vol. 8, no. 12, 2017.

[48] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, ‘CNN Features off-the-shelf: an Astounding Baseline for Recognition’, Mar. 2014.

[49] M. Braun, Q. Rao, Y. Wang, and F. Flohr, ‘Pose-RCNN: Joint object detection and pose estimation using 3D object proposals’, in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, 2016, pp. 1546–1551.

[50] R. Shanmugamani, *Deep Learning for Computer Vision: Expert Techniques to Train Advanced Neural Networks Using TensorFlow and Keras*. Packt Publishing, 2018.

[51] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, ‘PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes’, *arXiv:1711.00199 [cs]*, Nov. 2017.

[52] M. Elhoseiny, T. El-Gaaly, A. Bakry, and A. Elgammal, ‘A Comparative Analysis and Study of Multiview CNN Models for Joint Object Categorization and Pose Estimation’, p. 10.

[53] P. Arena, L. Fortuna, G. Muscato, and M. G. Xibilia, ‘Multilayer Perceptrons to Approximate Quaternion Valued Functions’, *Neural Networks*, vol. 10, no. 2, pp. 335–342, Mar. 1997.

[54] N. Matsui, T. Isokawa, H. Kusamichi, F. Peper, and H. Nishimura, ‘Quaternion Neural Network with Geometrical Operators’, *J. Intell. Fuzzy Syst.*, vol. 15, no. 3,4, pp. 149–164, Dec. 2004.

[55] D. Xu, L. Zhang, and H. Zhang, ‘Learning Algorithms in Quaternion Neural Networks Using GHR Calculus’, *Neural Network World*, vol. 27, no. 3, pp. 271–282, 2017.

[56] A. Geiger, P. Lenz, and R. Urtasun, ‘Are we ready for autonomous driving? The KITTI vision benchmark suite’, in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, Providence, RI, 2012, pp. 3354–3361.

[57] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, ‘A benchmark for the evaluation of RGB-D SLAM systems’, in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 573–580.

[58] ‘DTU Robot Image Data Sets | Data for Evaluating Computer Vision Methods etc.’ .

[59] ‘OpenCV library’. [Online]. Available: https://opencv.org/. [Accessed: 02-Sep-2018].

[60] ‘OpenCV: Geometric Image Transformations’. [Online]. Available: https://docs.opencv.org/3.1.0/da/d54/group__imgproc__transform.html#gga5bb5a1fea74ea38e1a5445ca803ff121acf959dca2480cc694ca016b81b442ceb. [Accessed: 02-Sep-2018].

APPENDIX – 1

1. To run the Training and Testing Code, Please first download the DTU-MVS dataset to your local driver, and change the dataset path in the training and testing scripts.

2. Please go to the github repository to download the sorce code

3. (optional) Run pre-process.py to pre-process the dataset, skip this step may lead to high memory occupation and slow training speed.

4. Select desired Traning settings to train the model

5. Test desired trained model by pass the corresponds argument to Test.py