FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

Lam Huynh

STRUCTURE-FROM-MOTION USING CONVOLUTIONAL NEURAL NETWORKS

Master's Thesis
Degree Programme in Computer Science and Engineering
August 2018

Huynh L. (2018) Structure-from-motion using convolutional neural networks. University of Oulu, Degree Programme in Computer Science and Engineering. Master's thesis, 62 p.

ABSTRACT

There is an increasing interest in the research community to 3D scene reconstruction from monocular RGB cameras. Conventionally, structure from motion or special hardware such as depth sensors or LIDAR systems were used to reconstruct the point clouds of complex scenes. However, structure from motion technique usually fails to create the dense point cloud, while particular sensors are inconvenient and more expensive than RGB cameras. Recent advances in deep learning research have presented remarkable results in many computer vision tasks. Nevertheless, complete solution for large-scale dense 3D point cloud reconstruction still remains untouched.

This thesis introduces a deep-learning-based structure-from-motion pipeline for the dense 3D scene reconstruction problem. Several deep neural networks models were trained to predict the single view depth maps, and relative camera poses from RGB video frames. First, the obtained depth values were sequentially scaled to the first depth map. Next, the iterative closest point algorithm was utilized to further align the estimated camera poses. From these two processed cues, the point clouds of the scene were reconstructed by simple concatenation of 3D points.

Although the final point cloud results are encouraging and in certain aspects preferable to the conventional structure from motion method, the system is just tackling the 3D reconstruction problem to some extent. The prediction outputs still have errors, especially in the camera orientation estimation. This system can be seen as the initial study that opens up lots of research questions and improvements in the future. Besides, the study also signified the positive intimation for using unsupervised deep learning scheme to address the 3D scene reconstruction task.

Keywords: learning-based 3D scene reconstruction, unsupervised depth and camera pose, point cloud from video, deep learning

TABLE OF CONTENTS

FOREWORD

This master's thesis was carried out at the Center for Machine Vision and Signal Analysis (CMVS) at the University of Oulu. First of all, I would like to express my gratitude to Prof. Janne Heikkilä for his supervision and guidance through my studies and thesis. My sincere thank goes to Dr. Markus Ylimäki for reviewing and providing valuable feedback on this work. I would also like to thank everyone at CubiCasa for all the support and discussions during the time of writing my thesis. Finally, I would like to thank my family and friends for their encouragement during my studies.

Oulu, Finland August 24, 2018

Lam Huynh

ABBREVIATIONS

SfM	Structure-from-motion
SLAM	Simultaneous Localization And Mapping
LIDAR	Light Detection And Ranging
SIFT	Scale-Invariant Feature Transform
AI	Artificial Intelligence
DL	Deep Learning
DNN	Deep Neural Networks
CNN	Convolutional Neural Networks
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
RGB	Red-Green-Blue
RGB-D	Red-Green-Blue-Depth
ReLU	Rectified Linear Unit
BN	Batch Normalization
GPU	Graphics Processing Unit
2D	Two Dimensional Space
3D	Three Dimensional Space
GD	Gradient Descent
SGD	Stochastic Gradient Descent
AdaGrad	Adaptive Gradient
Adam	Adaptive Moment Estimation
ϵ	Epsilon
GT	Ground Truth
SVD	Singular Value Decomposition
PMVS2	Patch-based Multi-View Stereo Software Version 2
ResNet	Residual Networks
FCRN	Fully Convolutional Residual Networks
CONV	Convolutional Layer
DOF	Degrees Of Freedom
RANSAC	Random Sample Consensus
ICP	Iterative Closet Point
RTE	Relative Translation Error
ROE	Relative Orientation Error

1. INTRODUCTION

Structure-from-motion (SfM) refers to computer vision methods that exploit the parallax induced by camera motion to infer the 3D scene structure. Conventionally, SfM has been implemented using handcrafted features such as scale-invariant feature transform (SIFT) [1] followed by bundle adjustment based on multiple view geometry. Many software implementations exist such as VisualSFM [2, 3], Bundler [4], and Theia [5] that utilize this kind of conventional approach.

During the last few years, deep neural networks (DNN), and in particular, convolutional neural networks (CNN) have shown to achieve superior performance in many image recognition tasks [6]. Recently, CNNs have also been applied to geometric vision problems including single view depth map prediction [7, 8, 9, 10], camera pose estimation [11, 12], 3D reconstruction [13] and unsupervised learning based on minimization of the reprojection errors from motion cues [14, 15, 16, 17]. These are end-to-end solutions where the networks were trained to produce 3D information directly from the input images without first extracting some predefined or handcrafted features or key-points like in the conventional approaches. The learning-based approaches can exploit richer feature representations, learned from examples, that are characteristic to certain objects or scenes of interest. Therefore, with carefully designed networks and large amount of training data, the learning based approaches can achieve better performance than conventional methods.

To the best of our knowledge, some previous works try to tackle the SfM problem using learning-based techniques. However, a solution for a large-scale dense 3D reconstruction, especially of indoor space, still remains untouched. The primary objective of this work is to utilize Tensorflow [18] to train DNN models that can produce a 3D point cloud from a video representing indoor spaces. The obtained point clouds are the main source of information for many tasks such as 3D scene understanding, 3D semantic segmentation, and building information modeling. These are especially useful in cases where the availability of RGB-D or Light Detection and Ranging (LIDAR) systems are limited, and conventional SfM approaches fail. The main contributions of this thesis include:

1. Experiments with both supervised and unsupervised deep-learning-based structure from motion approaches to reconstruct a dense 3D point cloud of a scene captured with a monocular camera.

2. Introduction of an end-to-end pipeline for generating a 3D point cloud from a sequence of RGB video frames.

2. STRUCTURE-FROM-MOTION

In the imaging process, when capturing an image of the real world with a camera, we lose the depth dimension. Since the beginning day of computer vision, many efforts have been made to recover this geometric information [19, 20], in which the structure-from-motion is the most well-known one. The principle of SfM is to reconstruct the 3D structure of a scene by exploiting the parallax cues in camera motion from a set of overlapping 2D images. The technique was first described by Ullman [21] in 1979 and became gradually popular in the 2000's, due to its ability to produce 3D point clouds easily with relatively low priced cameras instead of an expensive LIDAR system.

Alongside SfM, other research areas, such as simultaneous localization and mapping (SLAM) and photogrammetry, also attempt to tackle the 3D reconstruction problem. In photogrammetry they are interested in using precisely calibrated cameras to generate reliable and highly accurate measurements from photographs. For example, Figure 1 represents a reconstructed model of the exact positions of surface points in satellite photos. On the other hand, in robotics, SLAM system prefers real-time performance to accuracy allowing the robot to navigate using vision. One of the most famous SLAM systems is the self-driving STANLEY car [22] made in Stanford University, which won the DARPA Grand Challenge [23].

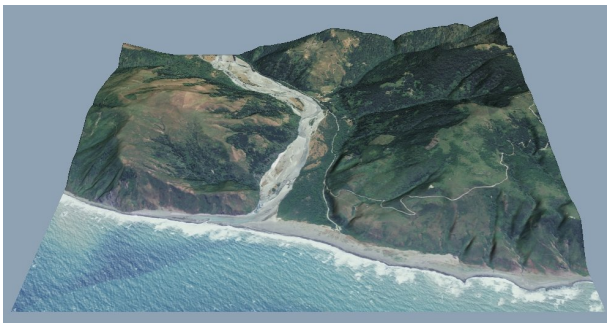

Figure 1. 3D model of the exact positions of surface points from satellite photos. Image from Google with labeled for reuse.

Despite having many variations, depending on the specific application as well as the number and type of the used camera, the generic pipeline of SfM remains the same. As illustrated in Figure 2, the first step is to find corresponding points between pairs of images by matching detected feature points. Then, the relative camera poses between views are obtained by computing and decomposing the fundamental matrices. After that, the 3D triangulation is used to identify the coordinates of the 3D points from corresponding 2D feature points. Finally, nonlinear optimization (bundle adjustment) is applied to minimize the reprojection error between the image locations.

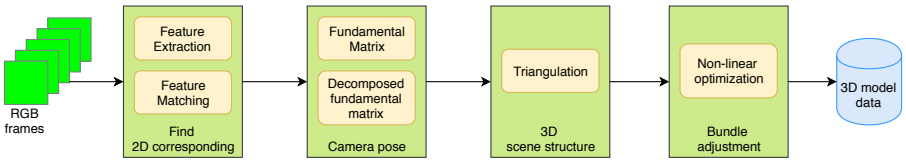

Figure 2. A generic structure from motion pipeline.

There is no doubt that the topic of structure from motion is immense, and detailed explanations are beyond the scope of this thesis. Therefore, this chapter gives an overview of the conventional structure from motion approaches, and discusses their disadvantages which lead to the use of the learning-based technique.

2.1. Finding 2D correspondences

Corresponding points are 2D points that appear in at least two or more views and are the projection of the same point in 3D space. The process usually starts with the detection and description of the interest points or features in the images. The number of existing detectors and descriptors are varied but one can use e.g Harris Corner detector [24] to extract the interest points and SIFT [1] to describe them. Features with similar descriptors will likely match with each other. The shorter the baseline between consecutive frames, the more equal the sets of detected interest points would be. In that case, the detected and described interest points are matched by comparing the pixel intensities in a small square area around the points using normalized cross-correlation, for example.

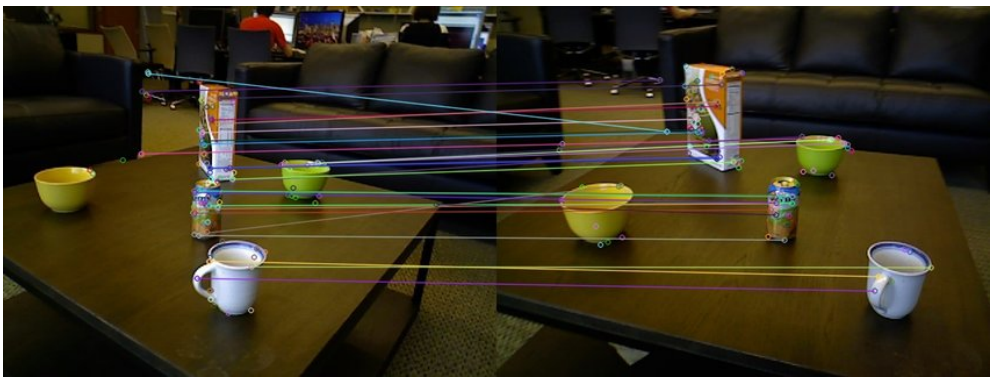

Figure 3. Corresponding points from two images. Image from Quinghua et al. [25].

Figure 3 illustrates a small set of matched feature points between two images using Harris Corner detector [24] and SIFT [1]. However, as one can see in the figure, lots of found correspondences are not correct.

2.2. Calculating relative camera pose

After acquiring the corresponding points, one can estimate the camera pose which contains the rotation and translation of the camera in relation to some known coordinate system. The transformation of a 3D point X′ in the coordinate system C′ to a 3D point X in coordinate system C is formulated as

$$X = RX' + t \tag{1}$$

where R is a 3×3 rotation matrix that contains the camera orientation and t is a 3×1 vector that contains the camera translation.

Multiplying both sides of Equation 1 with $X^T[t]_\times$ yields

$$X^T[t]_\times X = X^T[t]_\times RX' + X^T[t]_\times t$$

$$0 = X^T[t]_\times RX'$$

$$0 = X^T EX'$$

where

$$[t]_\times = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix}$$

is the cross product matrix of t and $E{\sim}[t]_\times R$ is a 3×3 essential matrix. The symbol "$\sim$" denotes the up to scale relation.

Considering corresponding points x and x' from two images in homogeneous normalized image coordinates, the epipolar constraint gives

$$x^T Ex' = 0 \tag{2}$$

Moreover, assume that the camera intrinsic parameters K and K' of the two views are known, substituting $x{\sim}K^{-1}u$ and $x'{\sim}K'^{-1}u'$ into Equation 2 yields

$$(K^{-1}u)^T E(K'^{-1}u') = 0$$

$$u^T(K^{-1T}EK'^{-1})u' = 0$$

$$u^T Fu' = 0 \tag{3}$$

where 1) u and u' are corresponding pixel coordinates in homogeneous format, 2) K and K' are upper triangular matrices that present the camera intrinsic parameters

$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

where f_x and f_y are the focal lengths in pixels, c_x and c_y are the pixel coordinates of the principal point and s is the skewness of pixels. Furthermore, $F{\sim}K^{-1T}EK'^{-1}$ is a 3×3 fundamental matrix. With at least eight corresponding points, an unique solution of the fundamental matrix can be obtained using the least squares method.

Then with the known intrinsic K and K', the essential matrix can be estimated by

$$E{\sim}K'^T FK$$

According to Hartley [26], if the essential matrix has rank 2, it can be decomposed using singular values decomposition (SVD) into

$$E = UZWV^T$$

where U and V matrices are orthogonal, with

$$W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$Z = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

and $ZW = diag([\sigma_1, \sigma_2, \sigma_3])$. One solution is that the first two singular values are equal to one and the third singular value is zero e.g. $ZW = diag([\sigma_1, \sigma_2, \sigma_3]) = diag([1, 1, 0])$. In this case, the rotation and the cross product of the translation vector could be calculated directly by

$$R = UWV^T$$

$$[t]_\times = UZU^T$$

where R is an orthogonal camera rotation matrix and $t_\times$ is a skew-symmetric matrix with an unknown scale of the translation.

2.3. Triangulation

From the obtained transformation matrices, the 3D points can be calculated by triangulation [27]. However, because of the distortion and outliers, instead of convergence, the back-projected rays from the views will not perfectly intersect.

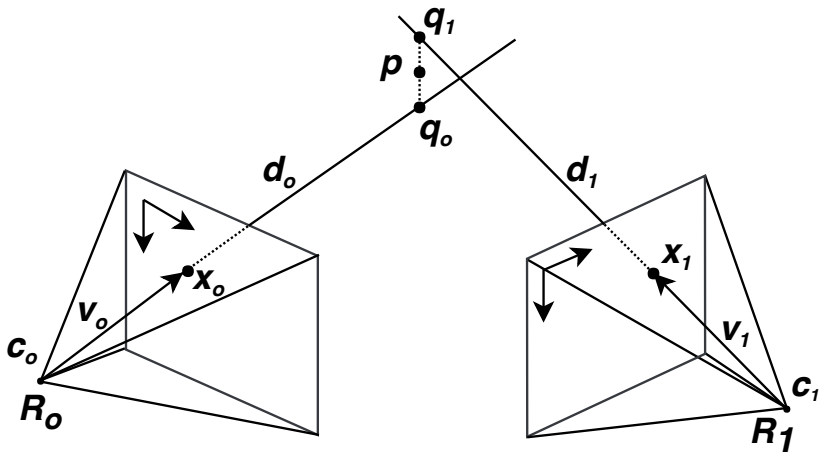

Figure 4. Triangulation.

Therefore, in order to find the best point of intersection, one solution is to calculate the nearest point to all of the back-projected rays by minimizing the distances between the estimated 3D point and the rays

$$||c_j + d_j v_j - p||^2$$

where p is the optimal 3D point, c_j is the origin of camera j, v_j is the normalized optical ray and $c_j + d_j v_j$ is a point on the optical ray in direction v_j. The optimal value for the 3D point p could be determined by finding a least square solution using singular value decomposition.

2.4. Bundle adjustment

The estimated camera poses and the 3D points from the previous parts are not optimal. Possible incorrect and/or inaccurate point matches may lead to an inaccurate 3D scene reconstruction.

To resolve these issues, some non-linear optimizations typically need to be applied to refine the initial estimate of the 3D points and the camera poses. The technique is called bundle adjustment [28], where the main idea is to iteratively tune the optimization parameters in order to minimize the sum of squared reprojection errors. This is an essential part in many SfM software packages, and typically consumes a lot of computation time in a large dataset.

In general, the system tries to minimize the squared Euclidean distance d between the image point x_{ij} and the projected point $\hat{x}_{ij} = P_i X_j$

$$\min_{P_i, X_j} \sum_{i=1}^{m} \sum_{j=1}^{n} D(x_{ij}, P_i X_j)$$

where $P_i = K_i * [R_i t_i]$ is the projection matrix of the i^{th} view, and X_j is the j^{th} 3D point.

This can be solved using the Gauss-Newton, gradient descent or Levenberg-Marquardt method. Detailed explanations can be found in Triggs et al. [28].

2.5. Popular SfM softwares

Bundler [28] is one of the first SfM softwares for unordered image collections. It takes a set of images, features and matches as input and calculates the motion of the camera as well as the 3D structure of the scene. However, the 3D models from Bundler sometimes contain large distortions caused by the accumulation of estimation errors. Nonetheless, the results from this software are usually considered as reference solutions for experimental comparisons.

In 2011, Wu et al. introduced the multi-core bundle adjustment [29] SfM package, called VisualSFM [2, 3], which not only demand less memory usage and dramatically reduce the run time but can also produce very accurate models. This improvement comes from the optimized usage of parallel computational resources. Moreover, VisualSFM has three excellent features including 1) a graphical user interface for the visualization of the SfM process, 2) inbuilt SIFT on GPU feature extraction and 3) possibility to use the patch-based multi-view stereo software (PMVS2) [30] directly from the graphical user interface. With these tools, VisualSFM can produce a point cloud reconstructions directly from a set of images.

Recently announced, fast and scalable structure from motion library, called Theia [5], includes efficient and reliable structure from motion algorithms. Theia library contains many useful implementations for areas such as image manipulation, feature detection, description and matching, robust pose estimation and resectioning, random sample consensus (RANSAC) [31], progressive sample consensus, camera models, and a full SfM pipeline. Furthermore, Theia is an active open source software with well-structured and simple interfaces as well as good online documentation. The re-

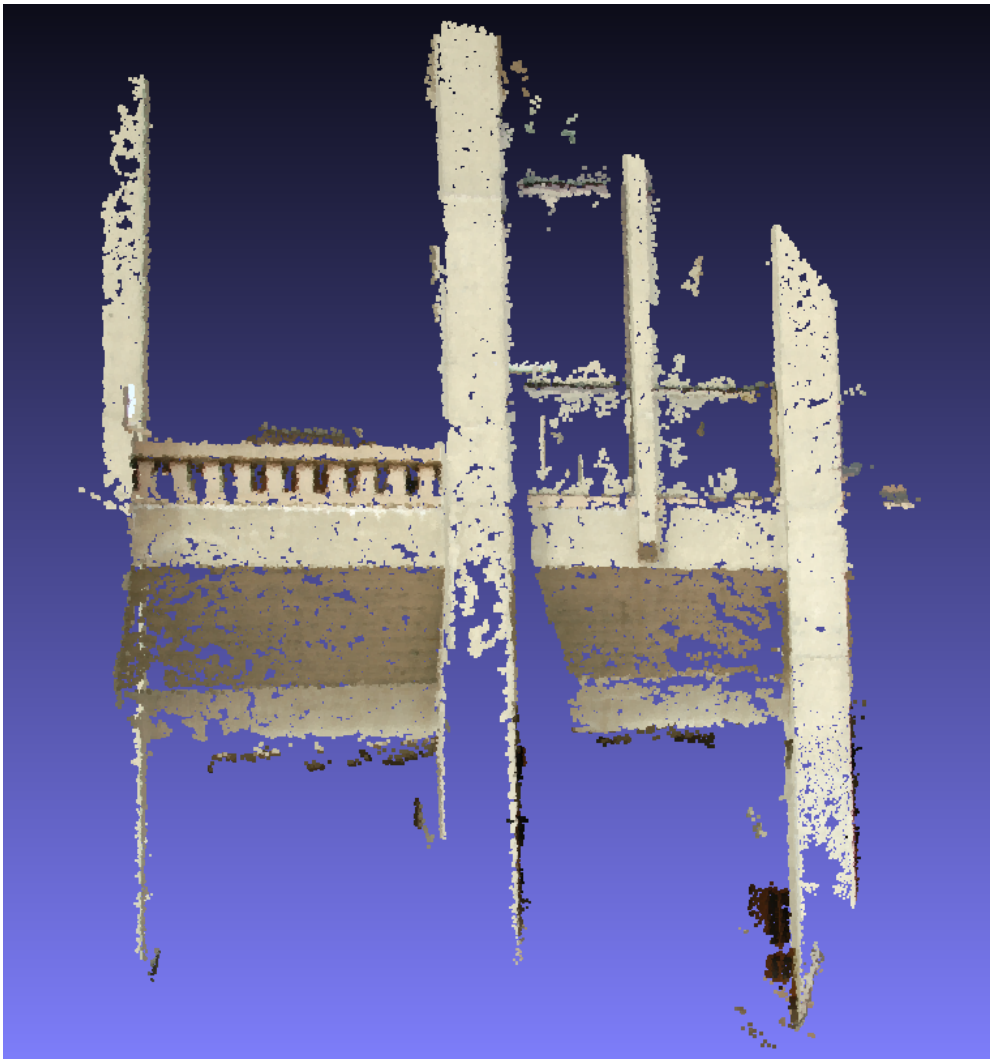

Figure 5. Dense 3D point cloud reconstruction using VisualSFM + PMVS2. The point cloud create from 61 high resolution images (3008×2000 pixels). Source RGB images: the Hall example of PMVS2.

sults made with both VisualSfM and Theia will be later compared with the results from the learning-based approaches.

2.6. Some drawbacks of feature-based structure from motion approach

It is difficult to create a robust, flexible and accurate reconstruction system using feature-based approaches. The scale and complexity of implementation will depend on the required tasks as well as prior assumptions. Along the pipeline, errors may occur, accumulate and immediately affect the accuracy and robustness of the outcome.

For example, the performance of the system will decrease if the chosen features lack essential representations for a generic dataset. The accuracy of the handcrafted features or extracted key-points heavily affects the efficiency of the point correspondences. Unfortunately, these algorithms suffer from the lack of texture in the indoor

environment which usually contains uniformly colored flat surfaces. This issue can be coped by using multiple feature detectors and fusing them together to form better features. However, this will require lots of works in exchange.

Another problem is in the triangulation process. The optical rays could become parallel if the transformation between the cameras is a pure rotation without translation. In this case, the depth of a 3D point cannot be resolved explicitly.

3. LEARNING-BASED STRUCTURE-FROM-MOTION

Since the success of AlexNet [32] for image classification on the ILSVRC-2012 (ImageNet competition) [6], deep learning (DL) has gained a lot of attention from the computer vision research community. DL has quickly been applied to many other problems such as object localization, object detection, semantic segmentation, pose estimation and depth map prediction. The rapid development of the network architectures, computation capacity, and datasets reveal many intriguing new research directions and industry applications.

From the broad domain of DL, this section dedicates to present the core ideas of machine learning as well as deep learning and deep neural networks systems. Some recent studies using deep learning to solve the 3D reconstruction problems are also discussed and analyzed.

3.1. Machine learning and deep neural networks

Artificial Intelligence (AI) is a vast research area which aims to give the computer the ability to perceive, understand, reason, plan and take action at the human-level efficiency. As a subset of AI, machine learning utilizes fundamental statistical models to recognize patterns in the data and make the prediction without the requirement of explicit programming of every rule. That is, computers can make their own decisions based on what they have learned. Deep learning is a powerful machine learning tool. The DNN models are specialized in learning the sophisticated representation from the data. Figure 6 below, illustrates the relationship between AI, machine learning and deep learning.

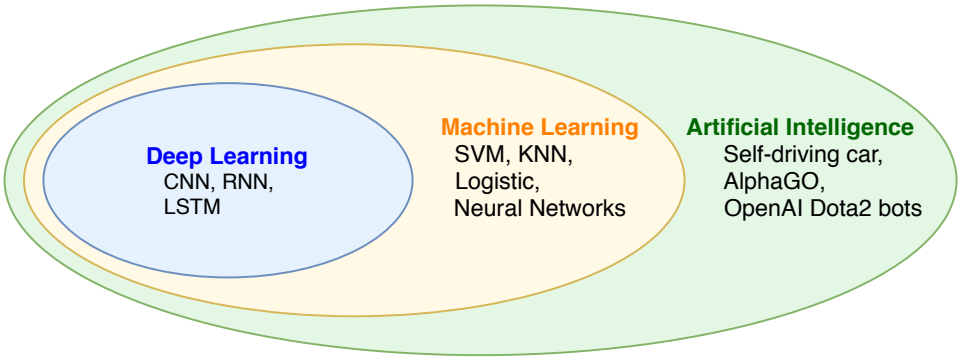

Figure 6. A Venn diagram indicates the relationship between Deep Learning, Machine Learning and Artificial Intelligence.

The elementary component of a neural network is a neuron, also known as unit or node. The magnification part in Figure 7 shows the structure of a single neuron receiving data from three inputs (x_1, x_2, x_3).

The neuron multiplies each of the inputs by the corresponding weight, then sums up and passes the calculated value to an activation function. This process can be formulated as

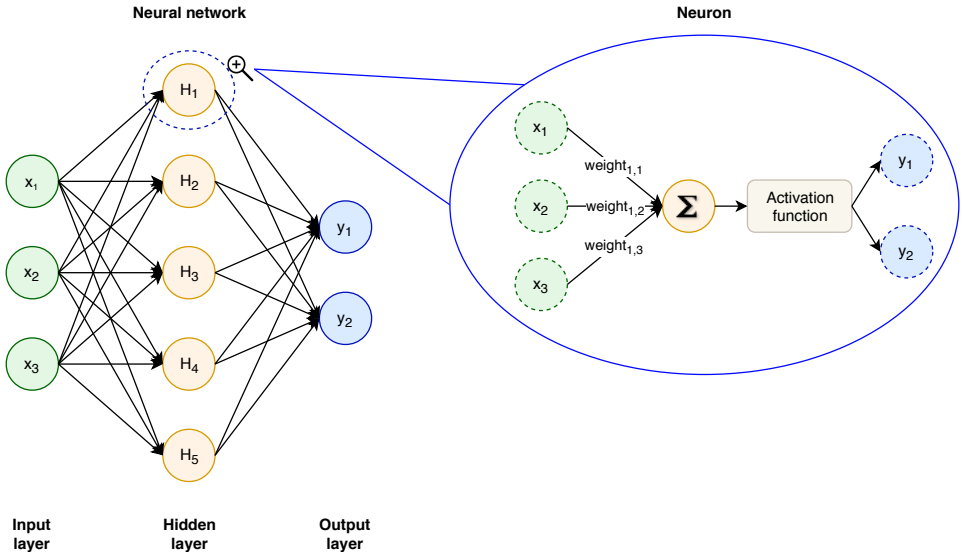

Figure 7. Simple structure of a convolutional layer.

$$y = \theta(b + \sum_{i=1}^{N} x_i w_i) \tag{4}$$

where w_i are the weights, b is the bias, x are the inputs, N is the number of input connections ($N = 3$ in this case) and θ is the activation function.

The neurons with the same depth form a layer and typically there will be thousands of neurons in each layer. Layers with continuous depth then connect to each other to construct the neural network. Training the network is a process of iteratively updating the weights and biases to minimize the loss function. The weights and biases along with the specific network architecture will be used to produce the output from the input data.

3.1.1. Network architecture

In machine learning, the "no free lunch theorem" [33] states that a universal learning algorithm or an ultimate best learning algorithm does not exist. One must create an own implementation to perform well on a specific task. With deep learning, the success of an application depends on the deliberate design of the neural network architecture.

As a consequence of the above fact, DNN's architectures have tremendously evolved over the last decades. There are many variations of the artificial neural networks, in which convolutional neural networks (CNN) have proven theirs competence in many computer vision tasks [34]. Moreover, these networks contain almost similar components, hence the best way to survey the field is thoroughly study of the most important ones.

3.1.2. Convolutional layer

The convolutional layer is the core building block of a CNN, and it is extensively discussed in the literature [35, 35, 36]. For two consecutive convolutional layers, each neuron in a current layer is only connected with a set of local input nodes that lie within a specific spatial region in the previous layer. This spatial region is also called the receptive field. Figure 8 shows a convolutional layer with the filter size of 3×3. The calculation inside a convolutional layer is the same as in an ordinary neural network. However, in this case, the dot product is computed in just the receptive field region.

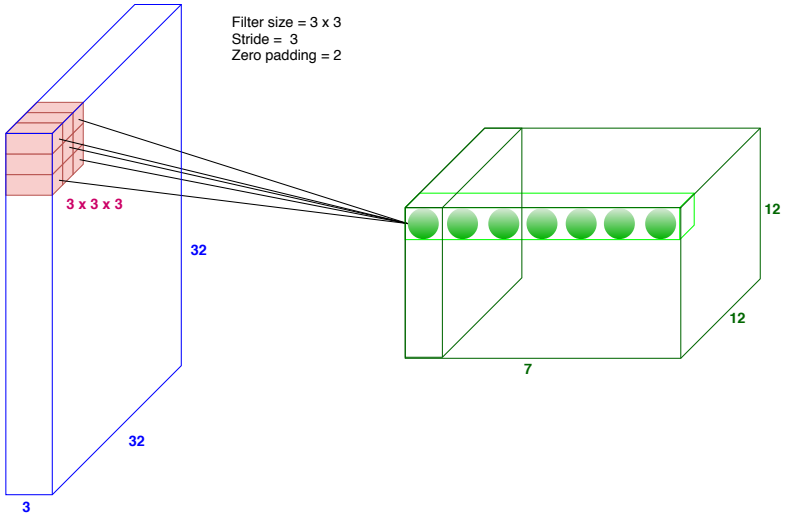

Figure 8. Illustration of a convolutional layer with filter size of 3×3, with strides $(s = 3)$ and zero padding $(p = 2)$.

The size of the output of the convolutional layer is defined by three parameters: 1) the number of filters, which will specify the number of output feature maps, 2) the stride (s) defines the step size of the sliding window and 3) the zero padding (p) which is the width of the frame of zeros added to the input volume before convolution. The spatial size of the output volume after the convolution can be calculated as follows

$$\frac{(w - f + 2p)}{s} + 1$$

where (w) is the size of the input volume and (f) is the size of the receptive field.

The number of parameters can be further diminished using the parameter sharing assumption. Assuming that if an important feature is detected in a specific region, it can be also useful in other regions, hence the network does not need to learn these features twice. For example in Figure 8, instead of storing $12\times12\times7\times(3\times3\times3+1) = 28224$ parameters. The network only needs $7 \times (3 \times 3 \times 3 + 1) = 196$ parameters when using the parameter sharing assumption where 12×12 neurons in the whole volume share the same parameters as shown with the light green box in Figure 8.

3.1.3. Transpose convolutional layer

Transpose convolutional block [37, 38] is also known as deconvolutional layer (TensorFlow 0.1 [18], Caffe [39]), or transpose convolution (Theano [40]).

Transpose convolutional layer is used to upsample the input volume by applying bilinear interpolation. That is, the output is calculated by convolving the input, but instead of scaling down, the output will scale up and still keep the relative representation of the learned features. This is achieved with a convolution using a fractional input stride of $\frac{1}{s}$, where s is the stride. Figure 9 illustrates this process.

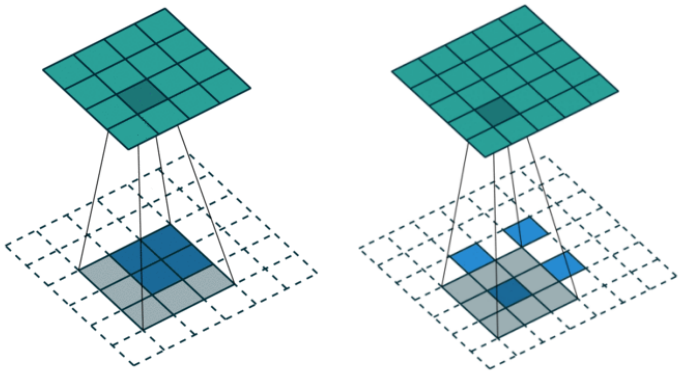

Figure 9. Illustration of the transpose convolutional layer. In the picture, blue maps are the inputs, and green maps indicate the outputs. The left image shows the transpose operation with zero padding and stride $s = 1$. On the other hand, the right image displays the transform with zero padding and stride $s = 2$.

3.1.4. Network in network layer

Network in network refers to an 1×1 convolutional layer, which was introduced by Lin et al. [41] in 2013.

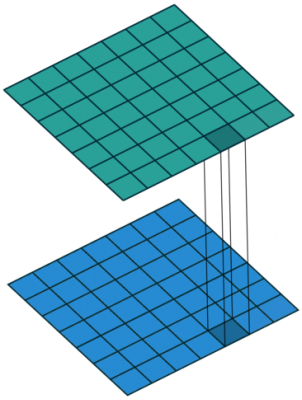

Figure 10. Illustration of an 1x1 convolutional filter.

At the first glance, people may argue about the usefulness of a somewhat flat convolution operator. However, one should remember that the convolutional kernel is, in fact, an $[1 \times 1 \times N]$ 3D tensor, where N is the depth of the input volume. That is, this unique block will perform the dot product in the depth dimension with the input layer. This, in turn, will help the convolved features from the previous layer effortlessly fuse, which is hard to do by using regular convolutional layers. This contemporary concept was later expanded in modern DNN models such as GoogLeNet (Inception V1) [42], Inception V2 [43], Inception V3 [44], ResNet [45], Inception V4 [46], DRNs (Dilated residual networks) [47].

3.1.5. Inception module

Two years after AlexNet, Szegedy et al. won the ILSVRC-2014 with the GoogLeNet [42], which introduced a new inception module. Inspired by the network in network paper [41], inception module applies 1×1 convolutional layer as a feature downsampling operator before feeding the input volume to the parallel blocks. After that, the outputs of the parallel blocks are upsampled again using another 1×1 convolutional filter. This is the fundamental idea of the inception module.

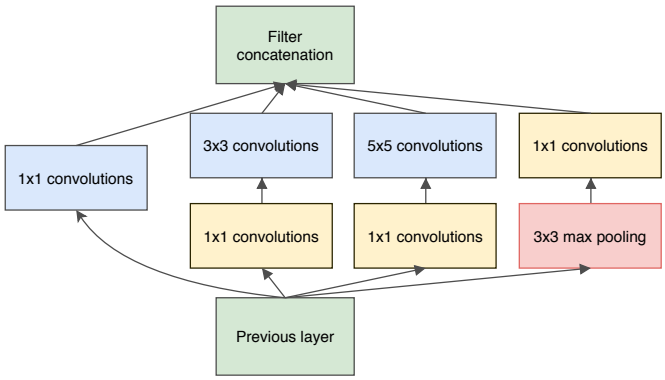

Figure 11. Inception module introduced in GoogLeNet [46].

For example, a convolutional layer in a CNN has the size of $3 \times 3 \times 512$ and its input contains 512 features. Normally, the network needs to perform $512 \times 3 \times 3 \times 512 = 2359296$ convolution operations (ops). However, if the network first convolve with a $1 \times 1 \times 128$ convolutional layer, the number of required ops will be $512 \times 1 \times 1 \times 128 = 65536$.

Then, the convolution with the same 3×3 filter in this smaller feature space will cost $128 \times 3 \times 3 \times 128 = 147456$ ops. Finally, the network will convolve with another $1 \times 1 \times 512$ layer to scale up the final feature space. This time it will also need 65536 ops. In total, the network has to perform 278528 ops, which is $\sim 8{,}5$ times lower than the normal way.

Using this idea, GoogLeNet not only required less parameters but could also effectively increase the network's depth and achieve a decent 6.67% error rate on the ImageNet classification challenge [6]. This study also proved that the neural network could be trained in parallel with convolutional branches instead of sequentially stacking them together.

3.1.6. Residual block

The winners of ILSVRC-2015, i.e. K. He et al., presented one of the most competent network architecture, the residual network (ResNet) [45]. Using ResNet, the authors trained DNN from hundred up to thousand layers without suffering the vanishing gradient problem.

The vanishing gradient problem occurs when the network is built by sequentially stacking up more than 19 convolutional layers. Then, in the backpropagation process (applying the chain rule to calculate the derivatives), after lots of iterative multiplication, the gradient will gradually decay to zero. As a consequence, the accuracy of the trained model will saturate or even decrease dramatically.

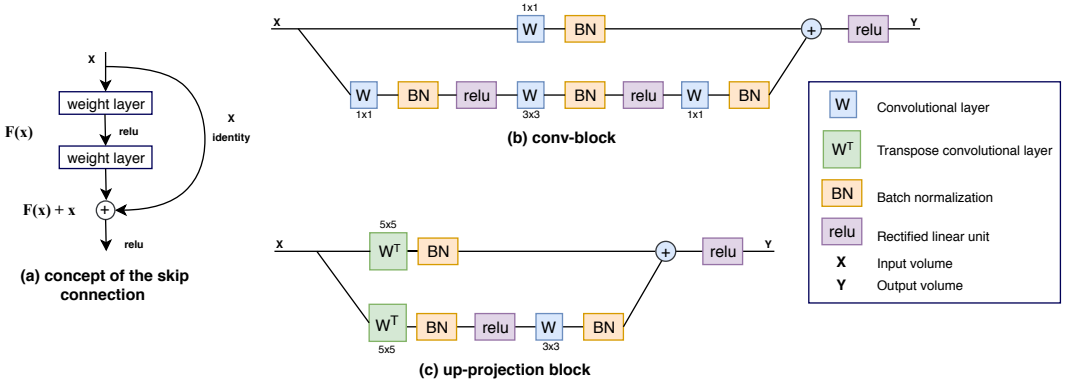

Figure 12. Residual block. The image (a) shows the concept idea, while image (b) and (c) are the combination of skip connection and inception module that are used to train very deep networks [45].

To overcome this problem, K. He et al. use a residual learning block called skip connection (also known as identity shortcut-connection), which is conceptualized in Figure 12 (a). In the skip connection module, the input from the previous layer will be simultaneously fed to two sequential convolutional layers and directly to the next layer. The skip over two layers creates a small network in a network [41], in which the shortcut pass will not decrease the training performance because these layers are only identity mappings of the input values. In other words, the training error of the deeper network cannot be larger than its corresponding shallower version.

Figure 12 (b) and (c) dissect two contemporary building blocks that apply the skip connection and inception module ideas. The conv-block in image (b) is used to train three sequential convolutional layers instead of one. On the other hand, the up-projection block in image (c) employs two transposed convolutional layers substitute for convolution ones to up-sample the input volume. These, in turn, not only increase the model depth, but also significantly reduce the number of features processed by the convolution. These two blocks are also used in this thesis. Detailed explanations for batch normalization (BN) and rectified linear unit (ReLU) will be introduced in the following sections.

3.2. Deep learning techniques

There are a lot of hyper-parameters and processing schemes inside a DNN model that need to be appropriately calibrated to get the best result for a specific task. This section describes the most common deep learning techniques that are used to train deep neural network models.

3.2.1. Weight Initialization

To perform numerical calculations on computers, variable initialization is always required and neural network weights and biases are not exceptions. Moreover, both empirical studies [48] and practice have proved that weight initialization has a big impact on the convergence of the training process. Therefore, this step needs to be carefully handled to avoid the symmetric problem of setting all of weights and biases to zeros in the first place. The network will then update the same gradient during backpropagation and make the training useless.

One common way is to set these values near but not exactly zero and scale them by a $\frac{1}{\sqrt{n}}$ factor to normalize the input variance as follows

$$w_i = \frac{random(n, \mu = 0, \sigma^2 = 1)}{\sqrt{n}}$$

where n is the number of the inputs, μ is the mean, and σ^2 is the variance.

Another favorite initialization method is to use Xavier initializer [48] with the purpose of keeping the same variance between layers. The variance at the layer i could be calculated as follows

$$\sigma^2(w_i) = \frac{2}{{}^{i}n_{in} + {}^{i}n_{out}}$$

where n_{in}, n_{out} are the number of inputs and outputs at layer i, and σ^2 is the variance.

In 2015, K. He et al. [49] presented a standard initialization scheme for ReLU units

$$w_i = random(n, \mu = 0, \sigma^2 = 1) * \sqrt{\frac{2}{n}}$$

where n is the number of the inputs, μ is the mean, and σ^2 is the variance.

Recently, Ioffe and Szegedy introduced a robust technique named batch normalization (BN) [43]. Batch-norm is a generic building block in DNN, where the main idea is to scale and shift the distribution of the training weights. This routine has become very popular because it is not only an efficient initialization method, but also a preprocessing layer that lies right in front of ReLU in ResNet.

In practice, one efficient way to initialize the weight is to use the pre-trained models from related task. Although this initialization scheme is more complex and constrains the network design, applying this can decrease the training time and ease the network convergence in advance.

3.2.2. Activation functions

Activation is used to add non-linearity to the network by crunching the dot product result in a specific function. As said earlier, the non-linearity will help the model to learn complex representations from the training data. There are several types of activation functions, which will be briefly discussed in this section. Illustrations of these functions are presented in Figure 13.

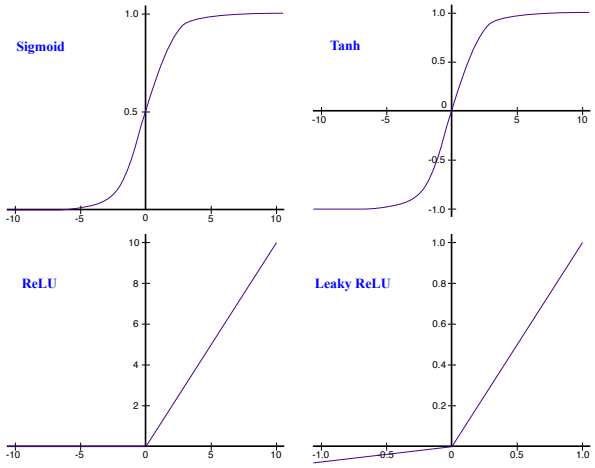

Figure 13. Illustration of well-known activation functions.

The first activation is the sigmoid function, which is formulated as

$$f(x) = \frac{1}{1 + e^{-x}}$$

In practice, this activation operator suffers from saturation to 0 or 1. These neurons will not learn anymore because the gradient is very close to zero. Moreover, sigmoid operator is sensitive to bad initialization and/or data preprocessing, and thus requires special attention when training the network. Another type of activation is $tanh$

$$f(x) = tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

that also suffers from the same saturation problem. Because of this, these traditional activation functions are barely used anymore, in practice.

The next type of activation operation is the rectified linear unit (ReLU) [32], which is calculated with the function

$$f(x) = \max(0, x)$$

Unlike the two other functions, the ReLUs do not suffer from saturation problem and are faster to compute. However, they undergo the dying ReLUs problem which happens when a large gradient is passed through these activation units. In this case, the gradient flow of these units is always zero. In other words, these neurons are dead and useless for training the deep neural networks.

One solution for this problem is to carefully set the learning rate when using ordinary ReLUs. Moreover, some studies managed to tackle this problem by linearly chopping

off the curve (non-linearity) from multiple possible output as illustrates in Figure 14. This idea was later generalized by Goodfellow et al. [50] with the proposed maxout operator. In general, the output after the activation can be further calculated by

$$f_i(x) = \max_{j \in [1,k]} (x w_{ij} + b_{ij})$$

where w_{ij}, b_{ij} are the learned weights and biases, x is the input volume and j is the number of affine feature maps. Figure 14 presents this idea using a quadratic curve. In addition, other studies like Leaky ReLU [51], or PReLU [49] are both special cases of maxout.

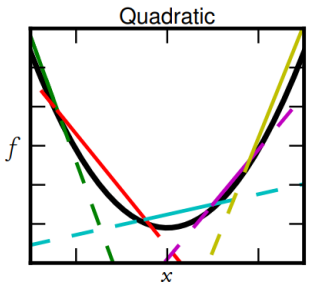

Figure 14. Maxout linearly approximate the quadric activation function [50].

3.2.3. Backpropagation

Backpropagation [52, 35, 53] is an essential algorithm used to calculate the gradient of the loss function. It is calculated using the chain rule, so that the neural network can update the weights and biases in the training process using optimizer such as the gradient descent.

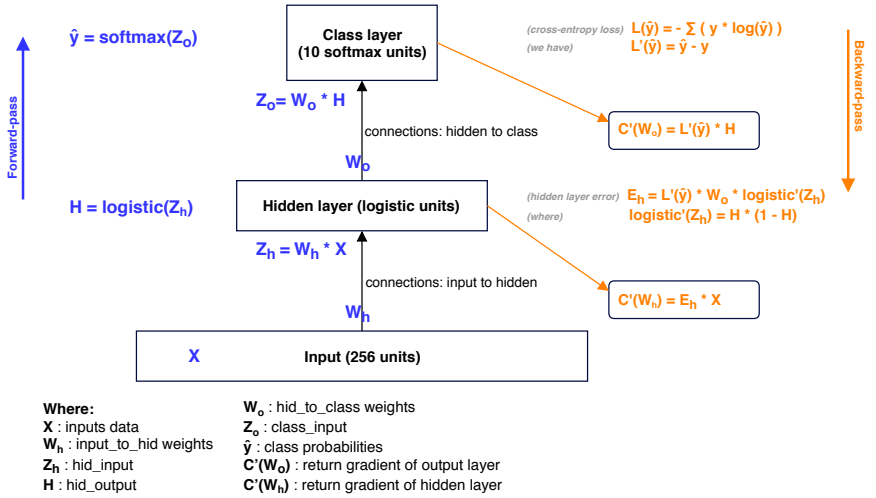

Figure 15. The illustration of the forward and backward pass in a simple model with only one hidden layer and no bias.

This process could be interpreted as the backward propagation of the errors of the network layers. When training, the errors at the current layer will be calculated. Then these error values will be passed backward to the previous layer, and recursively doing this until the errors have travelled back to the first hidden layer. At each layer, the network will update its weights and biases by computing the derivative of the cost for each weight.

Figure 15 shows the basic calculations of the forward and backward pass for a simple network that has the input layer (1-D tensor, with 256 units), only one hidden layer with an arbitrary number of neurons and the output layer that has ten units (representing ten output classes).

3.2.4. Regularization

In machine learning, the term overfitting refers to the problem where the model has learned the training data too well. Because of this, the model has not only learned the essential features but also unwanted noise. This, in turn, causes poor generalization when testing the trained model with real or test datasets. One way to tackle this problem is to increase the size of the training dataset. However, it is not always very straightforward to do this in practice. Instead, there are several regularization techniques that can be utilized to overcome this issue.

One common technique is $L2$ regularization which adds a compensation $\frac{\lambda}{2n}\sum w^2$ to the loss function, where n is the number of training samples, λ is the regularization parameter and w is the current weight. By decaying the weight toward the zero during the training, the network tends to learn relatively smaller weights to avoid overfitting itself.

On the other hand, $L1$ regularization will add the term $\frac{\lambda}{n}\sum |w|$ to the loss function. In practice, $L1$ regularization can make the model invariant to noise, because it concentrates on updating a small number of important weights while pushing other weights close to zero.

In the case where the weight is big, $L2$ will shrink it more than $L1$. In contrary, if the weight is small, $L1$ will shrink that weight more than $L2$. However, as proposed by Zou and Hastie [54], it is possible to use both $L1$ and $L2$.

Another regularization term is dropout [55], presented by Srivastava et al. This technique prefers deactivating some neurons inside the training process by adding a dropout probability p to each neuron. The idea seems counter-intuitive, but by dropping out totally random set of new neurons after every training batch, it helps to control the overfitting very well and is widely used within the research community.

3.2.5. Optimization algorithms and loss functions

In general, the ultimate goal of the training is to update the learnable variables to optimize the cost function by applying some optimization techniques. The definitions of the loss functions and optimization algorithms are probably the most crucial factors in the training process and they depend heavily on the application. In the supervised deep learning, the goal is to minimize the differences between predicted and ground

truth values. On the other hand, in unsupervised learning, the primary objective is to learn the whole probability distribution that can synthesize the wanted representation (or output data). These topics have drawn substantial attention from the research community with fast-evolving pace. This section will discuss about some useful methods, in practice.

The first one is the traditional gradient descent (GD), which calculates the gradient for the whole dataset. The weight is then moved gradually towards the negative gradient direction until it reaches a local minimum. However, when the goal is to find the global minima, the local one is usually not enough. In addition, it is also impractical to calculate the gradient for a big dataset.

One alternative method is the stochastic gradient descent (SGD), which updates the weights and biases by looping through the training dataset and calculates the gradient for each sample. With SGD, the objective function can jump over the local minimum pitfall. Nevertheless, the variance between every sample is usually high which fluctuates the objective function and make it hard to convergent. Therefore, to combine the benefits of GD and SGD, it is common in practice to use the mini-batch SGD to calculate the gradient for a subset of data.

Moreover, to improve the convergence rate in plain gradient descent, a momentum term

$$v_t = \gamma v_{t-1} + \alpha \nabla_w L(w)$$

is usually added into the update scheme by

$$w = w - v_t$$

where v is the velocity, γ is the momentum, α is the learning rate and $\nabla_w L(w)$ is the gradient. This idea was inspired by the momentum term in physics and has become popular because it also works well when applied to the training of DNN.

However, Nesterov found the problem that momentum can keep overshoot the global minima if it moves too fast [56]. To slow down the momentum, he suggested calculating the gradient based on the one-step-ahead rather than the current step before actually moving into that step. This, in turn, makes the update more responsive to the slopes in the training data domain.

Furthermore, it is also possible to adjust the learning rate with a parameter. The adaptive gradient (AdaGrad) [57] is one of these techniques, which uses the dynamic learning rate to make a large update for infrequent parameters and a small update for frequent ones. By using this, there is no need to manually change the learning rate when training.

To push things even further, adaptive moment estimation (Adam) [58] aims to adjust the learning rates and the momentum for every parameter automatically. The Adam process includes 1) calculating the mean moment using

$$\bar{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{5}$$

2) uncentering the variance of the gradient by

$$\bar{v_t} = \frac{v_t}{1 - \beta_2^t} \tag{6}$$

and 3) updating the parameters using

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{\bar{v}_t} + \epsilon} \tag{7}$$

where α is the learning rate, $\beta_1, \beta_2 \in [0, 1)$ are exponential decay rates and ϵ is the smoothing value to avoid division by zero. Normally, $\beta_1 = 0{,}9$ and $\beta_2 = 0{,}999$ whereas $\epsilon = 10^{-8}$.

In practice, Adam [58] is one the best methods due to its fast convergence rates, and it often outperforms other methods. There are many more algorithms which could occupy a whole chapter to explain. For more details, one can check out Ruder's article [59], which gives an excellent overview of these methods.

3.3. Structure from motion using deep neural networks

As discussed in Chapter 2, recovering the 3D structure from a set of images or a video is currently done by using conventional structure from motion methods that are mainly based on feature-based algorithms. Recently, 3D reconstruction using deep learning has received much attention since Eigen et al. [7] introduced a new method for monocular depth prediction. To the best of our knowledge, the compelling task of building an end-to-end 3D scene reconstruction using DNN is still under development. This section will discuss some state-of-the-art works that apply DNNs to tackle problems such as depth map prediction and camera pose estimation, which can be later used for 3D reconstruction.

3.3.1. Related work

One of the foundation studies on the ill-posed monocular depth problem is the paper by Eigen et al. [7] entitled "depth map prediction from a single image using a multi-scale deep network". As claimed by the name, the model consists of two-level CNNs. The first one is the coarse-scaled network, which comprises four convolutional layers with max-pooling followed by two fully connected layers. This network was trained to predict the coarse depth map at a global level. The output then will be feed to a fully convolutional network to refine the previous prediction by aligning it with local details.

One year later, Eigen et al. presented an expanded version [8] from [7] study, which train three groups of CNNs for semantic labeling and surface normal estimation. The first sub-network is used to make the coarse prediction from the input image, and then the second sub-network will predict mid-level resolution. Finally, the third one will refine these outputs to produce the high-resolution prediction.

Another effort to estimate the depth map from the monocular system is the study by Laina et al. [9], where instead of refining from the coarse prediction by using two or three sub-networks, they employ a fully convolutional residual networks (FCRN) to refine and up-sample the images. The authors utilize the residual block in the ResNet-50 [45] to concatenate high-level features from the output of previous convolutional layers. Then these output volumes are up-sampled using transpose convolutional layers

to produce a high-resolution depth map. Their proposed network also removes pooling and fully connected layers, which dramatically reduce both the training time and network parameters. The idea of employing deeper network showed the improvement on the prediction performance compared with Eigen et al. studies [7, 8]. This thesis will later modify the FCRN network for training a supervised DepthNet model.

Recently, Liu et al. proposed a new idea of training DNNs to estimate planar-wise instead of pixel-wise depth maps called PlaneNet [10]. The network was designed based on the dilated residual networks [47], and incorporated with a conditional random field network to predict the segmentation masks for dominant planes in the scene. They created 51k the planar-wise ground truth depth maps from the Scannet dataset [60], and then trained their model on this dataset. This publication currently provides the best result for the single view depth map prediction problem, despite the use of a relatively small training dataset.

On the other hand, using the supervised method, Kendall et al. introduced a CNN model, named PoseNet [11], that can estimate the 3D camera pose from a single image. PoseNet uses the trained weights on ImageNet [6] to perform transfer learning upon the vanilla GoogLeNet [42] for building a camera pose estimation regressor. The PoseNet model was trained on the Cambridge Landmarks (outdoor) [61] and the 7 Scenes (indoor) [62] datasets. The results from this work are promising, which in turn encouraged several later works.

Melekhov et al. [12] proposed a CNN regression model that can predict the relative camera pose between a pair of images. Also utilizing the supervised method, they manipulate $L2$-norm by inserting one parameter to the loss function for balancing the translation and rotation values. Using this objective function, they conduct transfer learning on trained hybrid-CNN weights [63] upon a dataset provided by Wilson et al. [64]. Moreover, to fix the number of spatial bins before feed-forward to the fully connected layers, they added a spatial pyramid pooling [65] layer at the end of the representation learning block. By doing this, the network yields improvement in accuracy evaluation.

Garg et al. [14] were the first to introduce the idea of training a network to predict the depth map from a single image in an unsupervised fashion. They trained a CNN model by optimizing the photometric error when inversely warping the left image to the right image. However, due to the difficult-to-optimize loss function, their model showed some limits on the final result.

In 2017, Godard et al. [15] presented an unsupervised DNN model for producing depth map from the single RGB image. To overcome the problem in Garg et al. [14] study, the authors combined the left-right consistency with the $L1$-norm and the single scale structural similarity [66] to form the loss function. Using this, they trained an auto-encoder by minimizing the image reconstruction cost when warping between pairs of left and right images. In other words, the left image is fed to the network while the right image is used to supervise the training. In the end, the model learned to generate the right from the left image, and the disparity map could be calculated from this stereo pair. The study showed remarkable results. Training the network without any labeled data, the model can still surpass Eigen et al. [7] and Laina et al. [9] (on Make3D dataset) which applied supervised method. Moreover, the inference time is impressive which can process $\sim$28 images per second in a single Titan X GPU.

3.3.2. *DispNet and SfMLearner models*

Mayer et al. [67] proposed an encoder-decoder architecture, called DispNet, that can directly learn to predict the disparity map from RGB images. In the publication, they trained the network on their own synthetic dataset, which was generated based on Flickr [68] images for the background and random Flying Chairs [69] from public 3D chair models as the foreground. This network was then fine-tuned on the large scaled street-view dataset named KITTI [70].

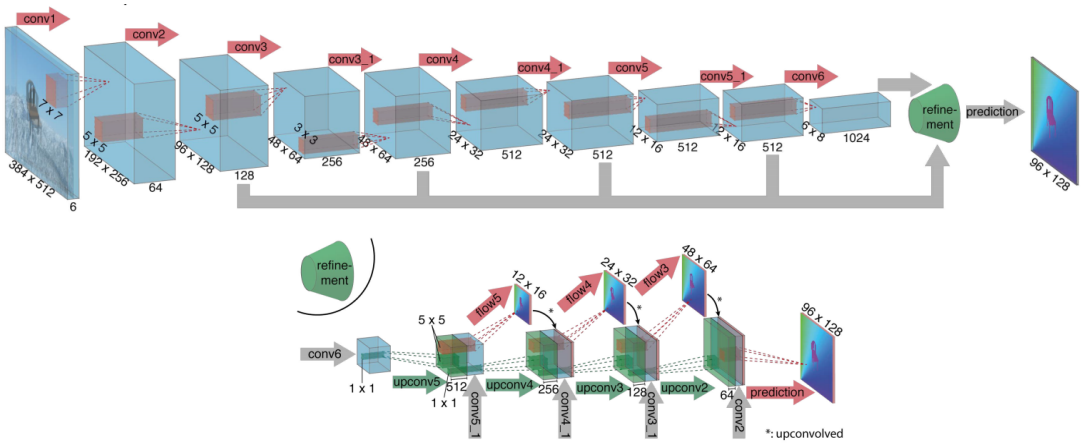

Figure 16. Illustration of the DispNet architecture, based on the FlowNet [71], for disparity map estimation task. The network consists a contractive part to learn the feature maps and an expanding part to later up-sample the prediction using transpose convolution with concatenation of previous feature maps in the contractive part. Image from Mayer et al. [67].

The DispNet's architecture, showed in Figure 16, consists of the contractive and expansion parts. The contractive module has 10 convolutional layers where the first layer uses kernel of size 7×7, and 5×5 for the second and third layer. The rest of the layers use smaller 3×3 kernel size. The numbers of filters are [64, 128, 256, 256, 512, 512, 512, 512, 1024, 1024] from the first to the tenth layer, respectively. These layers learn the coarse feature maps from a stacked pair of $[384 \times 512 \times 6]$ input to be later refined in the following module.

The output from previous part is fed to the DispNet's expansion part, which has four transpose convolutional layers. These layers are used to perform un-pooling and convolution with the concatenated input feature maps from previous feature maps of the contractive module. After four up-convolution processes, the size of the spatial feature maps is expanded from 6×8 to 48×64 before making the final prediction to produce a 96×128 disparity image.

Although the disparity estimation result from the DispNet [67] was not very good, it is still comparable with the state-of-the-art studies and inspired later research such as Zhou et al. [17]. In that paper, they proposed SfMLearner, an unsupervised model that learned to predict the camera poses and depth maps separately from RGB images. The SfMLearner was pre-trained on the Cityscapes [72] and fine-tuned on the KITTI dataset [70].

For the depth map estimation, the SfMLearner employed the DispNet and inserted four convolutional layers to the contractive part. The kernel sizes of the first four convolutional layers are $[(7 \times 7), (7 \times 7), (5 \times 5), (5 \times 5)]$, while the rest of the layers use 3×3 filters. In addition, the numbers of filters are [32, 32, 64, 64, 128, 128, 256, 256, 512, 512, 512, 512, 512, 512] from the first to the fourteenth layer, respectively. In the contractive part of the SfMLearner network, new transpose convolution layers were also added using the same idea as in the DispNet to refine the output depth images. The employment of the additional layers yield improvement on the depth map estimation task.

Furthermore, Zhou et al. introduced the PoseNet [17] to estimate the relative camera poses from a set of consecutive frames. This network has 7 convolutional layers with stride-2, and the numbers of filters are [16, 32, 64, 128, 256, 256, 256], respectively. The first layer uses 7×7 filter kernel, the second employs 5×5 whilst others use 3×3. The final convolutional layer is used to produce the 6-DOFs of the camera motion.

These networks are concurrently trained by minimizing the photometric consistency across monocular video datasets. The Adam optimizer is employed with a default exponential decay rates of 0.9 and 0.999 for the first and second moment estimate values, respectively. The learning rate starts at 2×10^{-4} with the training batch size 4. All of the layers utilize ReLU for activation. In addition, every layer, except the output layer, uses the batch normalization before activation.

The network architectures of this work are mainly based on the design of the FCRN [9], DispNet [67] and SfMLearner [17] models. These approaches seem to perform reasonably well in the single view depth map prediction and camera pose estimation when using indoor datasets. In addition, they provide both supervised and unsupervised deep learning models for training. In this work, the above models were modified and used to create a learning-based 3D reconstruction pipeline. Details of these implementations are described in the next chapter.

4. IMPLEMENTATION

This research focuses on experimenting different learning-based SfM approaches to reconstruct a 3D model of a scene from a set of overlapping images. These DNN models aim at predicting the camera transformation (i.e. camera translation and orientation) and the scene structure (i.e. the depth maps) from the RGB frames. These predicted values are then used to create the 3D point cloud of the captured scene. The point cloud reconstruction consists of three steps: 1) scale estimation of the predicted depth maps, 2) point cloud registration with iterative closest point algorithm [73, 74] to refine the camera transformations and 3) concatenation of overlapping point clouds into a single 3D model. The generic pipeline of this work is illustrated in Figure 17.

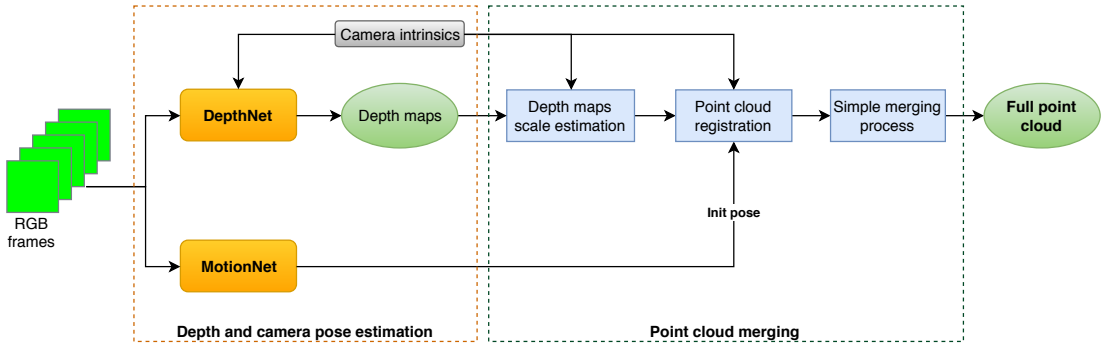

Figure 17. The generic pipeline for point cloud generation from an RGB video.

In this chapter, Sections 4.1 and 4.2 describe the supervised and unsupervised deep-learning SfM approaches, respectively, while Section 4.3 presents a straightforward point cloud merging process.

4.1. Supervised depth map prediction

Approximating the direct mapping from a single RGB image to its corresponding depth map is a difficult problem. However, the depth values are essential information when reconstructing the 3D space. Therefore, to perform the monocular depth estimation, a DNN was trained based on the ResNet [45] and FCRN [9] models.

Moreover, the training of the supervised depth map is two-fold. First, the trained model can directly produce depth maps from RGB images for the dense 3D reconstruction. Second, the weights from the contraction part of the supervised network will be used to initialize the training of the unsupervised model in Section 4.2.

4.1.1. Network architecture

The supervised depth model of this work is based on the FCRN architecture [9], with some changes both in the contraction and extraction modules. The network design is based on the vanilla ResNet [45], which contains one 7×7 convolutional layer (with 64 filters and strides 2) and 19 conv-blocks. The conv-block, as described in Chapter

3, uses the skip connection and identity blocks to train a much deeper network. The contractive part of this network consists of $19 \times 3 + 1 = 58$ convolutional layers.

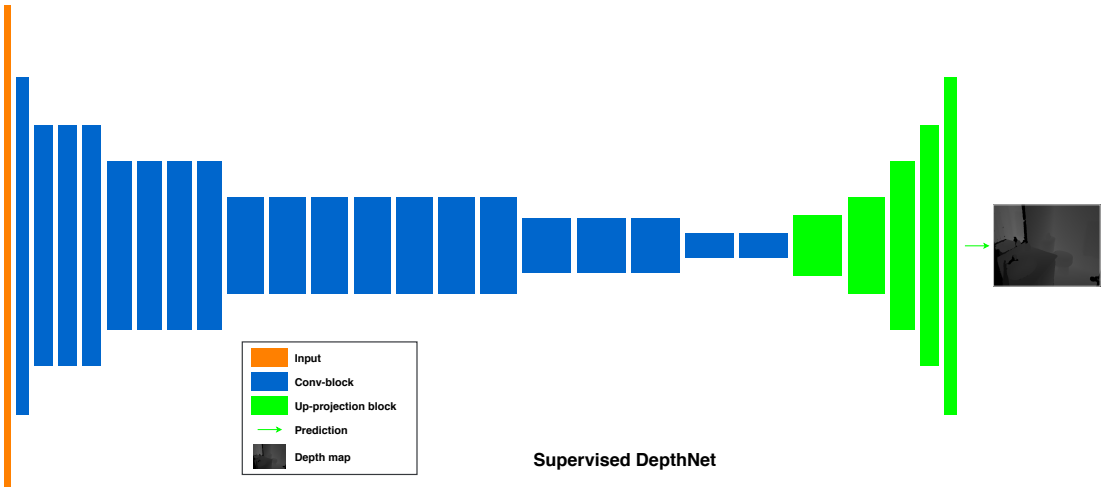

Figure 18. Network architecture for depth map prediction.

Moreover, the conv-blocks help to produce much deeper network for capturing a receptive field up to 497×497 despite the vanishing gradients problem. However, instead of stacking up fully connected layers at the end of the network for inferencing, the up-projection block is used to gradually up-sample the resolution of the predicted depth map.

The network input size is $304 \times 228 \times 3$, and after performing five up-projections, the network can produce a 320×256 pixels depth map. With one additional up-projection block, the network shows small improvement with larger depth maps than the FCRN [9] model.

4.1.2. Data preprocessing and augmentation

The original FCRN [9] was leveraged by transferring learning from the pre-trained ResNet [45] on the ILSVRC dataset [75], which is a very large dataset containing more than 1.2 millions RGB images for scene recognition tasks. After learning the low-levels features from the ImageNet dataset, the pre-trained ResNet model was concatenated with the extraction module (contains up-projection blocks) for the fine-tuning on the NYU-v2 [76]. The FCRN [9] model was trained using the NYU-v2 official train/test split, containing about 95k pairs of RGB-D images after augmentation.

As discussed earlier, the modified network has seven 256-filter conv-blocks and five up-projection blocks for refining larger resolution depth map. However, the new blocks introduce more parameters and make the network slower to converge. Therefore, in addition to the NYU-v2 [76], the model was also fine tuned on the SUNRGB-D [77] dataset. The SUNRGB-D consists of 13215 original image pairs, which, after augmentation, added more than 52k pairs to the combined dataset. Hence, the dataset consists of 147k images in total which in turn is divided into 81k/66k for training and validation, respectively.

The augmentation of the training data consists of several image transformation processes which were applied both on RGB images and depth maps. The transformations were scaling, rotation, coloring and flipping. For the scaling, the RGB images were scaled by a random factor in range $[0.875, 1.25]$. To perform the rotation, the inputs and ground truth depths were rotated in the range of $[-5, 5]$ degrees. The coloring was also applied by multiplying the RGB images with a random value in range $[0.7, 1.25]$. Finally, a subset of pairs were flipped horizontally to produce new images. In the above processes, coloring, flipping and rotation are geometry-invariant while scaling is not. Hence, it is necessary to divide the ground truth depth values by the same scaling factor when perform this type of augmentation.

4.1.3. Training details

During the training phase the weights of the networks are updated using the backpropagation to minimize the objective function

$$L(x) = \begin{cases} |x| & \text{if} |x| \leq c, \\ \frac{x^2+c^2}{2c} & \text{if} |x| > c, \end{cases} \tag{8}$$

where x represents the error at every pixel and the threshold value c is defined by

$$c = \frac{1}{5} \max_i(|\hat{y}_i - y_{gt_i}|) \tag{9}$$

where i is the pixel-indices, $\hat{y}$ is the predicted depth and y_{gt} is the ground truth one.

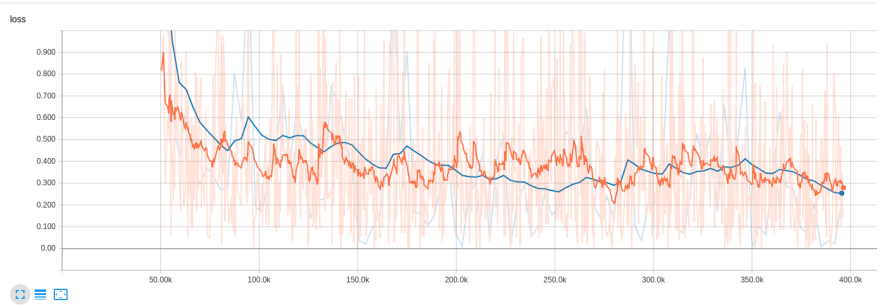

Figure 19. Training of the supervised depth showed by Tensorboard. In the chart above, the orange curve indicates training loss while the blue curve shows the validation one. The decreasing slope of both curves signify the learning progress of the model over training iterations.

However, instead of the common quadratic loss, the reverse Huber function (Eq. 8) is utilized for the regression of the depth values. Equation 9 determines the threshold value c which is 20% of the maximal error of pixels i on each image in the training batch. Then, in Equation 8, the gradients are updated separately at every pixel depending on the size of error x. If the error is larger than the threshold c, the quadratic loss will be calculated, otherwise, the $L1$ norm will be applied. The reverse Huber (Eq. 8), in fact, exploits the best from both worlds because the $L1$ norm is better for a small error while the quadratic loss is more effective on a larger gradient.

The initial learning rate for the Adam optimizer was 10^{-3} and ϵ was set to 10^{-8}. The exponential decay rates for the first and second moment estimate values were 0.9 and 0.999, respectively. In the fine-tuning process, the batch size of 16 was used to train for about 30 epochs, which consumed $\sim$56 hours running on one Telsa K80 GPU. After that, the trained model was saved into the checkpoint files that could be later used to make the inference.

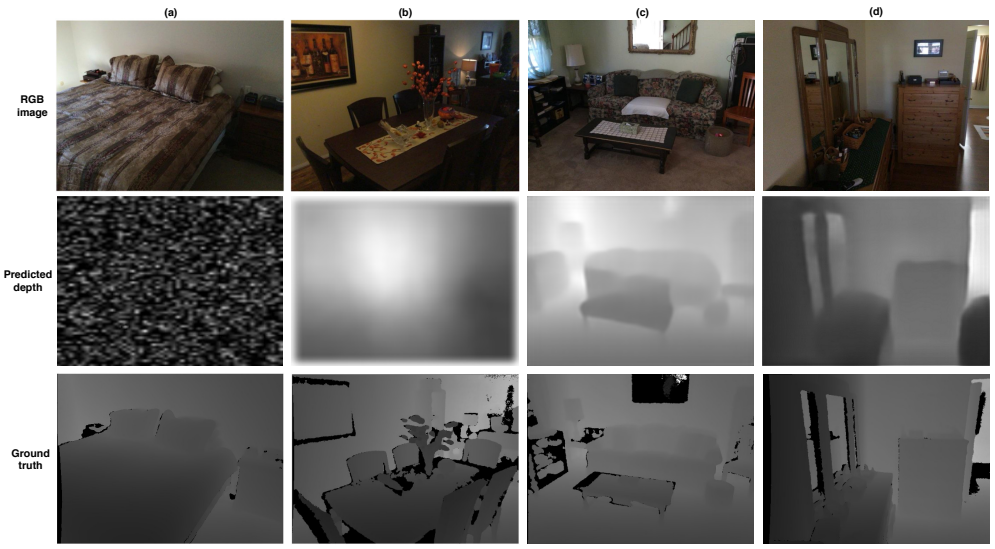

Figure 20. Visualization of the predicted depth maps during training. The estimated depth values gradually became better as seen in the results after about (a) 40k iterations, (b) 75k iterations, (c) 250k iterations and (d) 350k iterations.

As showed in Figures 19 and 20, using the initialized weights from the FCRN [9] eased the training of the supervised network. The network started to converge and produce reasonable depth maps after $\sim$200k iterations.

4.1.4. Error distribution of predicted depth maps from the supervised model

After training, the error distribution of the estimated depth maps was analyzed to spot out systematic errors for further improvement. The errors were calculated with ground truth values from the video sequence number 047 of the SceneNN [78] dataset, which contains 5693 image pairs for this purpose. For every feasible pixel, the median error and the standard deviation were computed with respect to the depth values and the radial distances from the principal point.

In Figure 21, the depth axis shows the range of the predicted values. In case of the supervised depth network, the estimated value is in meters because the model has learned the direct mapping from the ground truth. On the other hand, the radial distance axis indicates how far a certain bin is from the principal point, which is usually at the middle of an image. The vertical axis shows the error value that measures the median error (top) and the standard deviation (bottom).

As one can see, errors tend to be bigger with bigger depth values and near image border regions. The area at depth values from 0.5 to 1.0 meters having large pixel

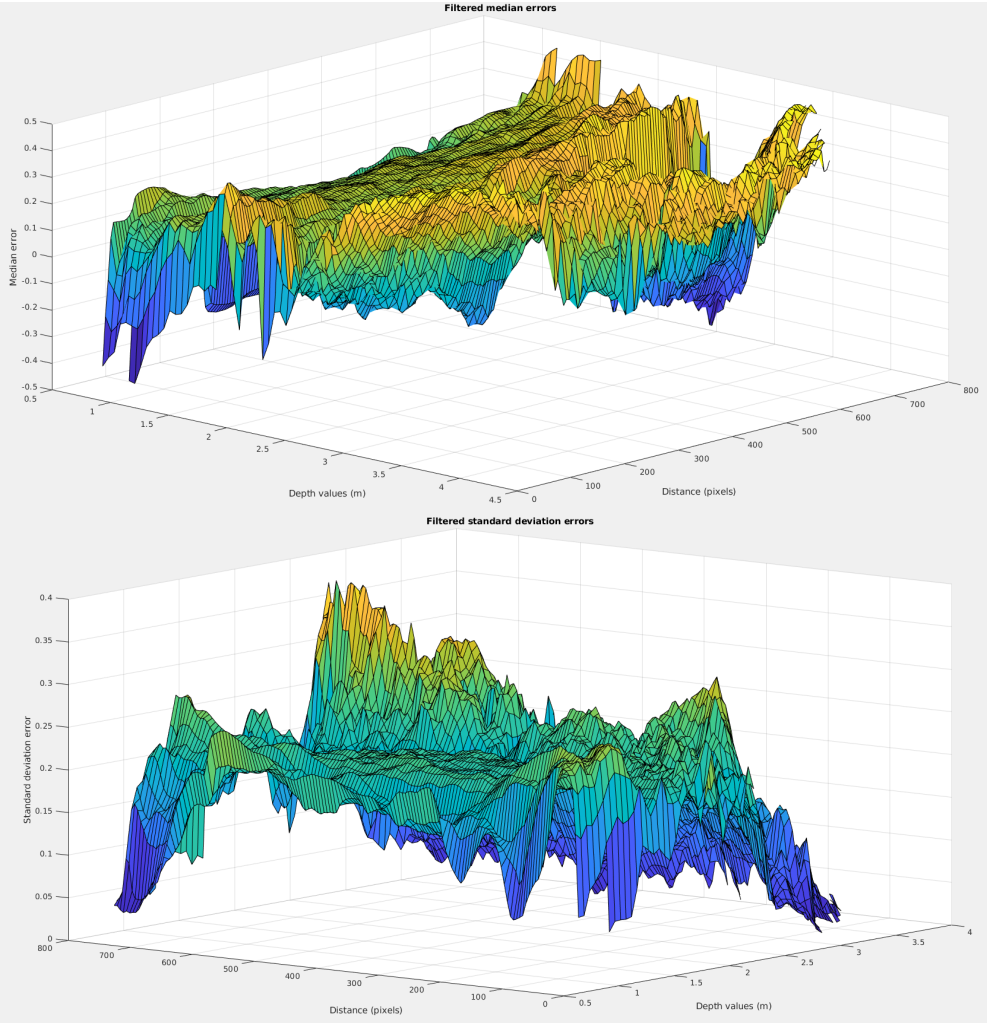

Figure 21. Statistical error distribution of supervised depth map prediction.

distances also contains estimation errors. Therefore, a simple mask was used to filter out the large error region, especially in the border areas.

4.2. Unsupervised depth map and camera pose estimation

In contrast with the supervised deep learning method in the previous section, the second approach trained the neural networks in an unsupervised manner. The DepthNet and MotionNet do not need the depth maps but only the RGB images in the training. Moreover, the trained model can predict both the monocular depths and the relative camera transformations between video frames. Then, along with the semantic segmentation from the SemanticNet, the predicted depth maps are refined using the planar assumption. Finally, these cues are used to reconstruct the 3D point cloud of the captured scene.

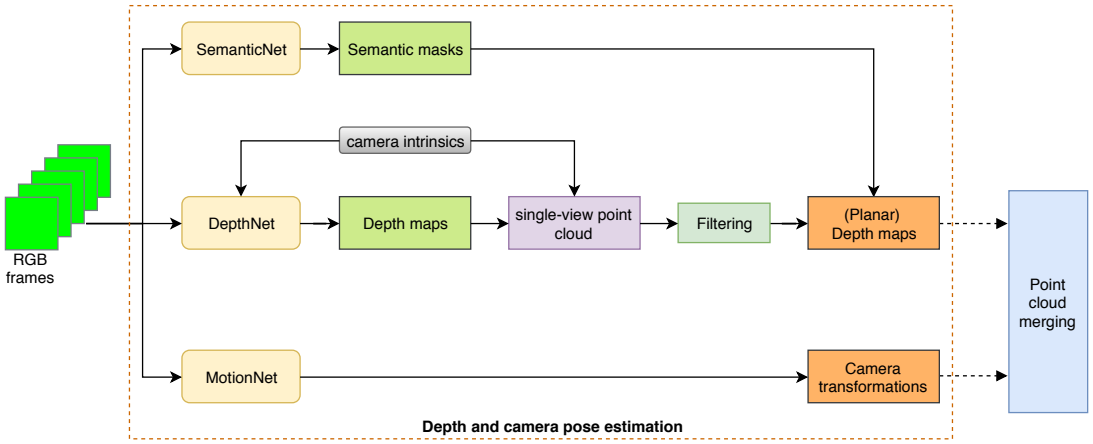

Figure 22. The pipeline used to acquire the depth map and camera transformation by training the unsupervised DepthNet and MotionNet. The SemanticNet was utilized to produce the semantic segmentation masks for applying the planar assumption.

4.2.1. Network architecture

The unsupervised model consists of the DepthNet for depth map prediction and MotionNet for the camera pose estimation. In the learning phase, both networks are trained simultaneously. However, they can infer the depth map and the camera pose separately in the inference stage.

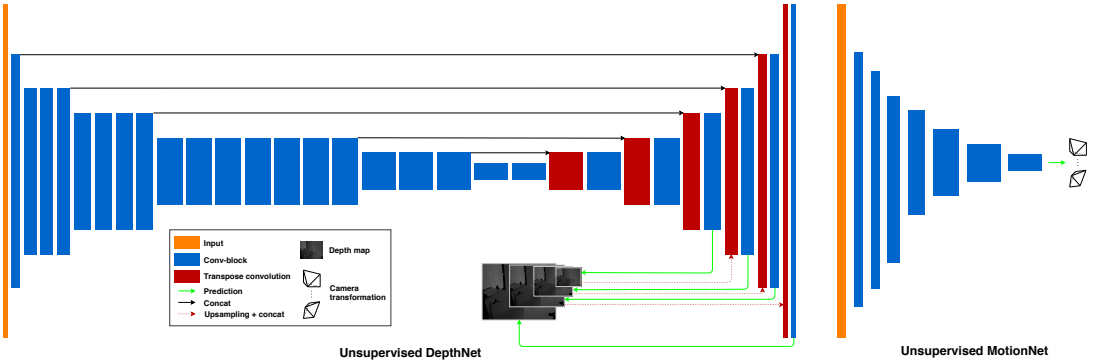

Figure 23. The network architecture of the proposed unsupervised DepthNet and MotionNet.

The DepthNet originates from the DispNet [67] and SfMLearner [17] but uses the network architecture described in Section 4.1.1 for the contractive part to build a better unsupervised model for datasets of indoor scenes. The much deeper contractive part of depth map estimation was utilized because indoor scenes are usually more complex than street-view images from KITTI [70] or Cityscape [72] datasets that were used to train the SfMLearner network. Moreover, the DepthNet was initialized using the pre-trained weights that will help the network to converge faster. The expansion part of the DepthNet employs the same structure as the SfMLearner and produces different-scale depth maps at the last four transpose convolution layers. These scaled outputs not only boost up the training of previous layers but also aid to visualize the training.

The MotionNet consists of eight convolutional layers with stride-2, and the numbers of filters are [16, 32, 64, 128, 256, 256, 256, 512]. The layer also uses 7×7 kernel size filter, the second one employs 5×5 whilst others use 3×3 as in the SfMLearner model. The final layer applies 1×1 convolutional layer to output 3 values for the 3-D translation and 3 values for the Euler angles in degrees.

4.2.2. Data preprocessing

The unsupervised learning technique only need the RGB images alongside with their camera intrinsic for the training. However, the network required a lot more data to train the model. Hence, two big indoor datasets, including the SceneNN [78] and SceneNet [79], were utilized in this task.

Figure 24. Data preprocessing with RGB frames from the SceneNet and SceneNN datasets. Images were stacked in the sequence of $s(s = 3)$ images. The first image is the target view while the others are the source views.

SceneNet [79] is a synthetic dataset containing rendered images from 3D models of indoor scenes. It has 17k 3D scene models and each model consists of 300 rendered images which makes up totally 5.1 million images. The images were rendered by moving the camera randomly inside the textured model having realistic lighting. The only limitation of the rendered dataset is the unrealistic textures of the objects in the images. However, a photorealistic scene was found easier for the network to learn in the early training stage.

SceneNN [78] is a real world dataset, which was captured using the Asus Xtion depth camera. This dataset contains 95 scenes with 52 room scenes, 22 office scenes and 21 main hall and class room scenes. The camera baselines in SceneNN dataset are small (about 10-20 cm), which is helpful for the training at the later stage.

Moreover, the trained explainability model from the SfMLearner [17] was used to remove images that contain occlusions between different views and non-Lambertian surfaces more than 35% of total number of pixels. The consecutive frames were then stacked together with their camera intrinsic to perform the training. Figure 24 shows some training sequences used in the training of the unsupervised networks.

4.2.3. Training details

The objective function for training the unsupervised networks is formulated as

$$L_{total} = \sum_{l} L_{vs}^{l} + \lambda_s L_{smoothness}^{l} \tag{10}$$

where l indicates the scale of the images, $L_{smoothness}$ is the smoothness loss (Eq. 14) and λ_s is the smoothness loss coefficient. The novel view synthesis loss L_{vs} is calculated with

$$L_{vs} = \sum_{s} \sum_{p} |I_t(p) - \hat{I}_s(p)| \tag{11}$$

where $\hat{I}_s$ is the warped source view, I_t is the target view, s presents the source view index and p indicates the pixel index. The network was trained by optimizing the projection error between the source view warped onto the target view and the target view.

To calculate the warped source view image $\hat{I}_s$, the pixel in the target view image was first warped to the source view by

$$p_s \sim K \hat{T}_{t \to s} \hat{D}_t(p_t) K^{-1} p_t \tag{12}$$

where p_t is a point in the target view, p_s is the temporary projected position of p_t in target view warped onto the source view, K is the camera intrinsic parameters, $\hat{T}_{t \to s}$ is the estimated camera transformation from target view to source view and $\hat{D}_t(p_t)$ is the predicted depth value of target view at p_t.

After that, $\hat{I}_s$ was obtained by bilinear sampling the values from four neighboring pixels using

$$\hat{I}_s(p_t) = \sum_{i \in \{t,b\}, j \in \{l,r\}} w^{ij} I_s(p_s^{ij}) \tag{13}$$

where $\hat{I}_s(p_t)$ is the source view warped on the target view, ij indicates the positions of four neighboring pixels at top-left, top-right, bottom-left, bottom-right of the pixel in the source view after warping, (t, b, l, r) implies top, bottom, left, right respectively, and w^{ij} is spatial proportion between the projected position p_s with respect to its 4-pixel neighbors. The sum of four w^{ij} is equal to 1.0.

The aperture problem that occurs in, e.g. textureless areas, creates zero gradient because the projection losses are equivalent in flat uniformly colored surfaces. This problem hinders the learning process of the DepthNet. Therefore, the spatial regularization

$$\begin{aligned} L_{smoothness} = & \frac{1}{m \times (n-2)} \sum_{i=1}^{m} \sum_{j=1}^{n-2} |\frac{\delta^2 f_d}{\delta x^2}|_{ij} + \frac{1}{(m-1) \times (n-1)} \sum_{i=1}^{m-1} \sum_{j=1}^{n-1} |\frac{\delta^2 f_d}{\delta x \delta y}|_{ij} \\ & + \frac{1}{(m-1) \times (n-1)} \sum_{i=1}^{m-1} \sum_{j=1}^{n-1} |\frac{\delta^2 f_d}{\delta y \delta x}|_{ij} + \frac{1}{(m-2) \times n} \sum_{i=1}^{m-2} \sum_{j=1}^{n} |\frac{\delta^2 f_d}{\delta y^2}|_{ij} \end{aligned} \tag{14}$$

where $f_d(x, y)$ is the predicted disparity map with the resolution of $m \times n$, was employed in the training to compensate for this problem. This term try to regularize the projection loss by optimizing the mean of the second-order gradients of the predicted disparity values from the DepthNet. Note that, the predicted depth values were the reciprocal of the disparity map, and were computed by $depth = 1/disp$.

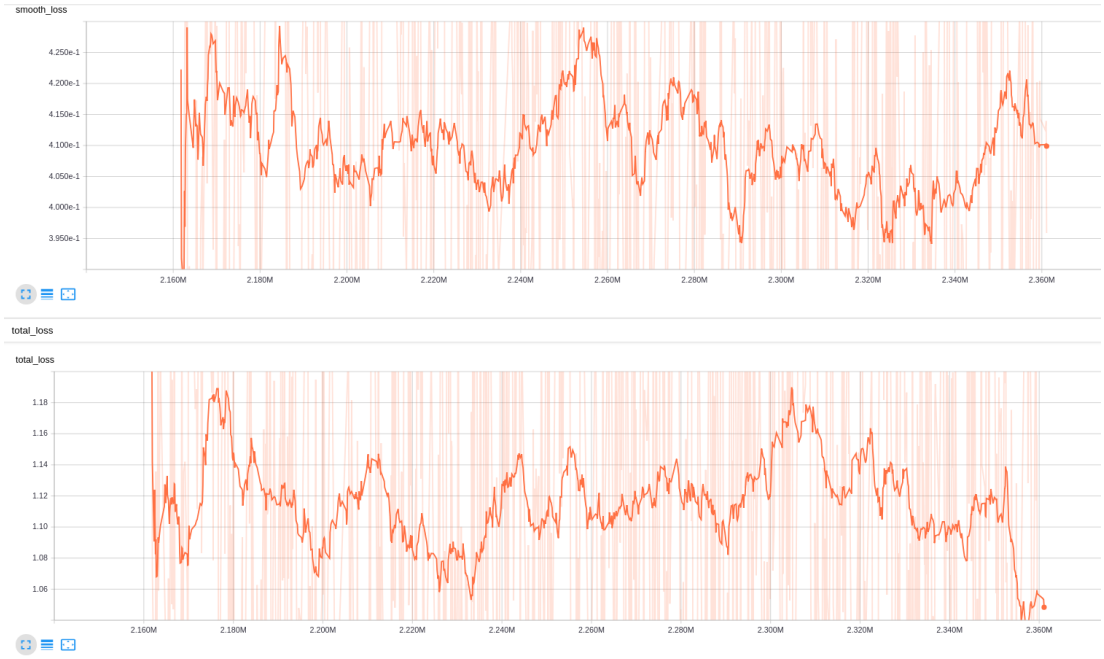

Figure 25. Total training loss and smoothness loss of the unsupervised DepthNet and MotionNet showed by Tensorboard. This illustrates the intermediate training process after ∼2 million iterations. The unsupervised model is more difficult to converge. In the figure, the top chart indicates the smoothness loss while the bottom chart showed the total loss. The total loss fluctuates, but still shows the decreasing slope especially at iteration number 2.35 millions.

Adam optimizer was utilized to train the unsupervised DepthNet and MotionNet, with the initial learning rate 3×10^{-3} and $\epsilon = 10^{-8}$. The smoothness coefficient λ_s was set to $\frac{0,5}{2^l}$, where l is the scale index and $l \in [1 \dots 4]$. The exponential decay rate for the first and second moment estimate values were 0.9 and 0.999, respectively. The contractive part of the DepthNet was initialized using the pre-trained weights from the supervised depth model. Both the convolutional layers in the DepthNet's expansion module and the MotionNet were then initialized using the Xavier method. The training used approximately 750k images and it was trained for about 100 epochs with the batch size of 8. The networks were trained on three Telsa K80 GPUs for around 125 hours.

In contrast to the supervised model, the objective function for training the unsupervised network is much less constrained, which made the network harder and slower to converge. An intermediate training session, illustrated in Figure 25, showed that the training loss took about $185k$ iterations to decrease the total training loss by ∼0.125. Otherwise, the smoothness loss fluctuated between $[3.950 \times 10^{-1} \dots 4.250 \times 10^{-1}]$ preventing the gradient decaying to zero and making the learning meaningful.

The core training of the DepthNet and MotionNet is to minimize the projection error in Equation 11. Figure 26 visualizes this training process in a period of ∼200k

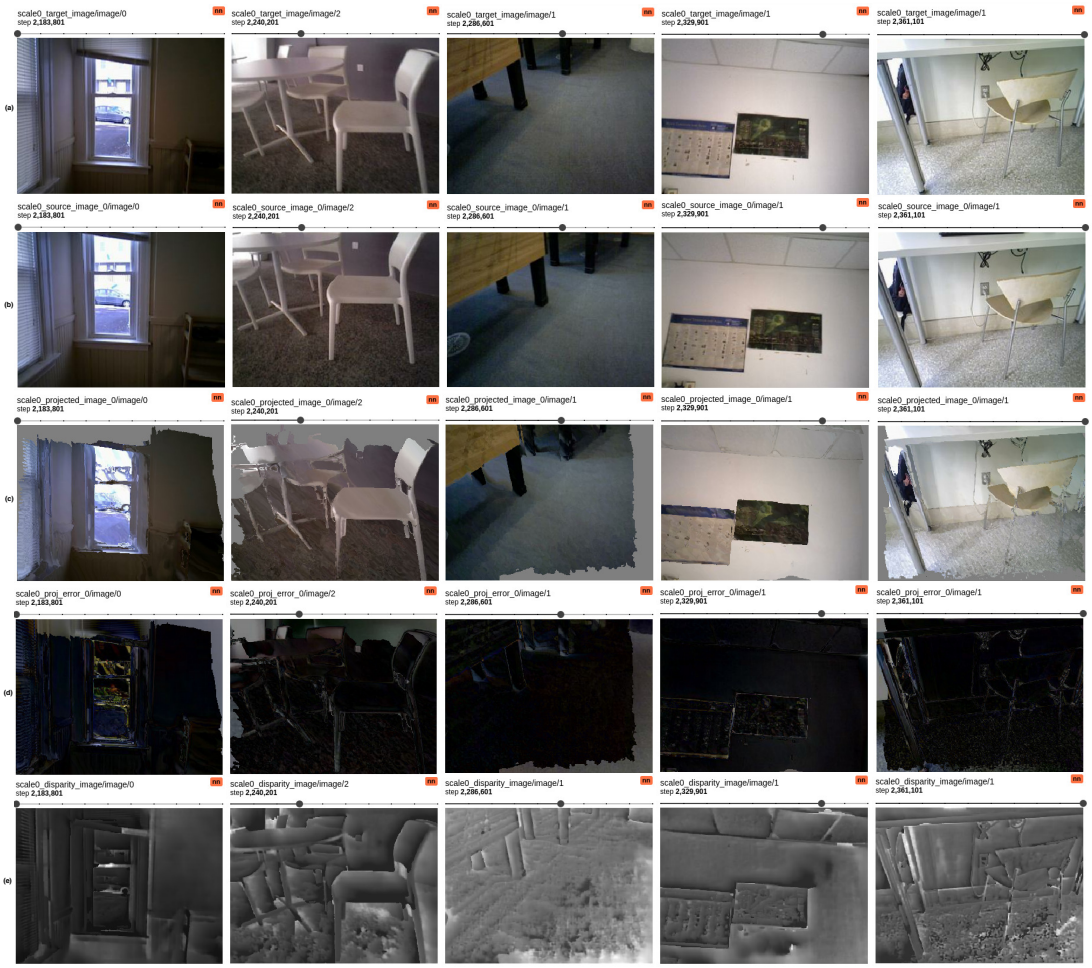

Figure 26. Illustrations of the target image (row a), source image (row b), the warped source view on the target view interpolated using the RGB information from the source view (row c), the projection error between the projected image and the target view (row d), and the predicted disparity map (row e). The predicted depth map is calculated by $depth = 1/disp$.

iterations. It was first started by applying Equation 12 to project pixels from the target view (Fig. 26, row a) to the source view (Fig. 26, row b) using the predicted depth values and camera transformations from the networks. The predicted depth values were calculated by $depth = 1/disp$. These pixels were then interpolated using the 4 neighboring pixels in the source view (Eq. 13) to create the projected image (Fig. 26, row c). The projection errors were calculated and showed in (Fig. 26, row d). From the left to the right, images are presented in an ascending order of the number of training iterations. Although the difference between the showed example images is not very clear, in the course of time, Figures 25 and 26 still present the decreasing trend of the projection error.

4.2.4. Error distribution of predicted depth maps from the unsupervised DepthNet

Using similar analysis as described in Section 4.1.4, the predicted depth values from the unsupervised DepthNet still contain errors around the border areas.

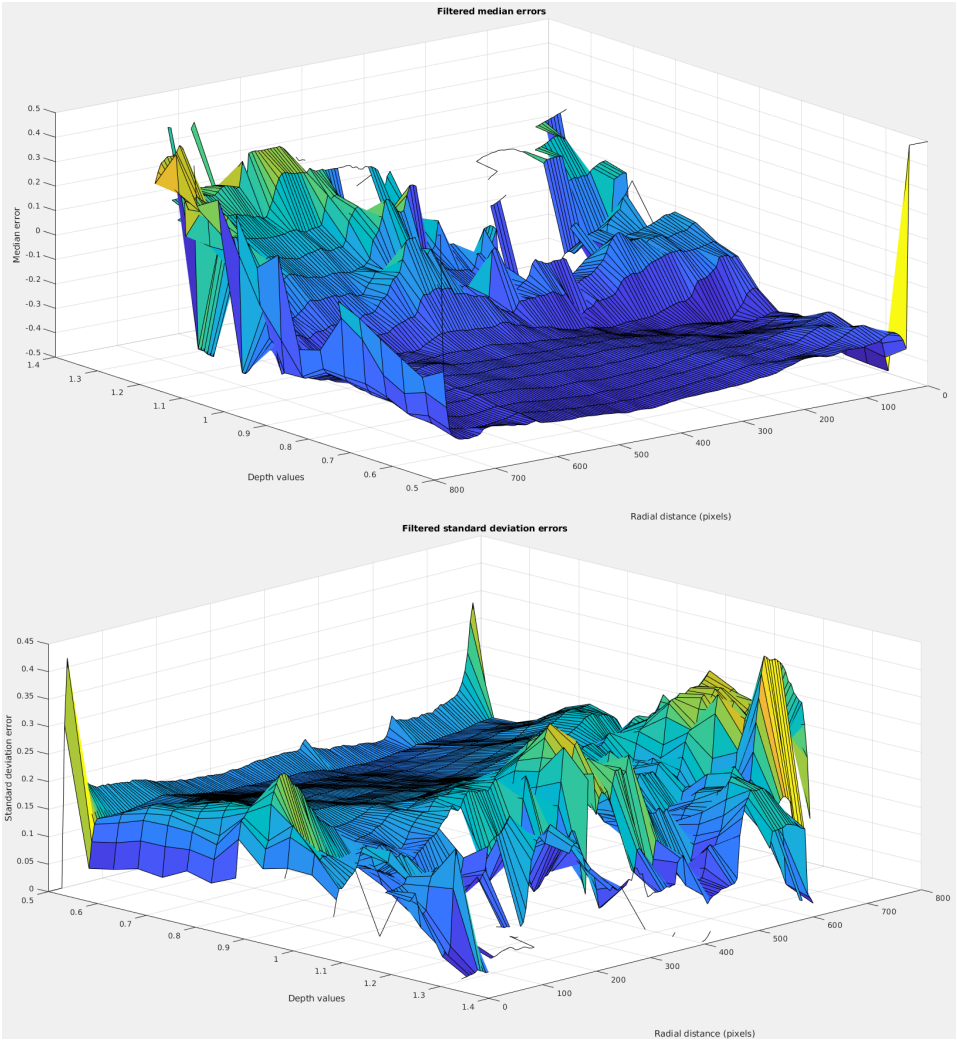

Figure 27. Statistical error distribution of predicted depth maps from the unsupervised DepthNet.

Large errors also appear in the areas of bigger depth values, which in the unsupervised case are not in the metric scale. Although the contractive part of the DepthNet was initialized by the supervised model, the depth prediction values have a rather arbitrary scale. This happens because the network indirectly learns to predict both the depth and the camera pose to minimize the objective function, which is more loosely constrained than the reverse Huber loss.

In general, output depth values seem to shift a little bit from the ground truth, which could push the predicted depth further away from the real values. Therefore, applying small weight λ_d^U for unsupervised depth compensation along with depth map scale estimation could fix this problem.

4.2.5. Semantic network and the planar assumption

Indoor scenes are usually constructed by dominant planar surfaces such as walls, floors, ceilings, and flat objects. The planar-wise understanding is very helpful, because most of the 3D scenes can be reconstructed by exploiting these linear surfaces. Therefore, one can apply a planar assumption to produce a depth map with dominant planes by using semantic segmentation on the predicted depth maps.

As illustrated in Figure 22, the planar assumption starts by predicting the depth and semantic maps. The depth values are filtered before back projecting them into 3D space to create a point cloud. A RANSAC plane fitting function with semantic constraints is then applied to these point clouds to find the dominant planes.

First, the SemanticNet is used to create the segmentation mask for walls, floor, furniture, stairs and others from the input RGB image. It was created based on the ResNet-101 DeepLab [80] model and fine-tuned on the ADE20K dataset [81] for semantic segmentation of indoor scenes. The outputs are pixel-wise 2D semantic labels, where peach color is the wall, green is the floor, gray is furniture and blue is the other classes.

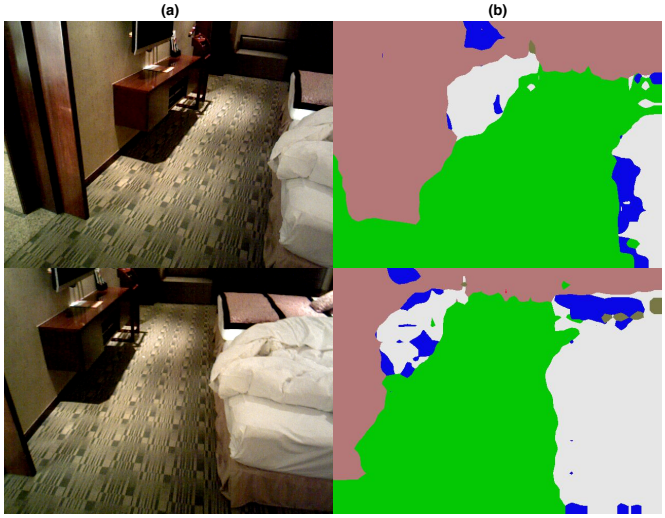

Figure 28. Semantic segmentation of images from the SUNRGB-D dataset. The semantic maps (column b) were created from the RGB images (column a). In the example segmentation images, the peach color presents the wall (labeled 1), green is the floor (labeled 2), gray is furniture (labeled 3) and blue is the other classes (labeled 26).

Next, the predicted depth values having $|z_i - z_{median}| > 3 \times std(z_i)$, where z is the depth value, are filtered out. In addition, the 3D points with less than $N(N = 500)$ neighboring points in a sphere with the diameter $d, \left[d = 0{,}1 \times max(3d\_points\_distances)\right]$ are also removed.

Then, RANSAC plane fitting is applied with specific semantic labels to find dominant wall, floor, furniture and stair planes. RANSAC [31] is a generic robust fitting technique for estimating the model parameters disregarding outliers in the data. Specifically, the plane fitting in our pipeline 1) randomly selects three samples from the point cloud and calculates the plane normal and offset, 2) fits all the samples and counts the number of 3D points $N_{inliers}^{plane}$ if their distances to the plane are smaller than 0.01, and 3) If $N_{inliers}^{plane}$ is greater than 40% of the number of 3D points in the cloud then the

algorithm updates the plane parameters and the set of remaining points. The iteration limit of RANSAC plane fitting is $N_{limit} = 35$, which is chosen by the suggestion of Fischler et al. [31]. For each semantic category, the algorithm tries to find at least 11 dominant planar surfaces.

After that, the obtained planes with the same label are merged together if the normal angles are within 20^o. In this case, the plane merging process determines and fuses the smaller planes to a bigger one. As the results, Figure 29 shows the reconstructed point cloud with (c) and without (b) planar assumption from RGB images (a).

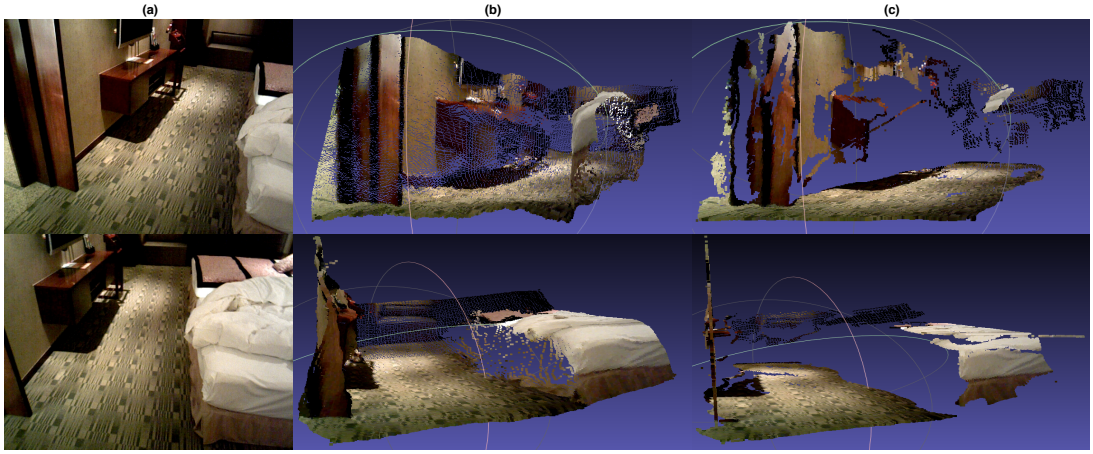

Figure 29. Point cloud reconstruction using the planar assumption. The example images show the reconstructed point cloud without (b) and with (c) the planar assumption from input images (a). Although some of the 3D points are filtered out, the point cloud can still present the most of the scene.

4.3. Point cloud merging using predicted depth maps and camera poses

After getting the depth maps and camera poses, a simple merging routine was implemented to reconstruct the full point cloud of the scene. The predicted depth maps are first rescaled to a uniform scale to fix the misalignment problem. Next, the iterative closest point (ICP) algorithm [73, 74] is used for the registration of the point clouds into a common coordinate frame using the estimated camera poses as an initial guess. Finally, the point clouds are merged together to form the dense 3D point cloud of the scene.

4.3.1. Depth maps scale estimation

In both supervised and unsupervised approaches, the direct mapping of RGB values to corresponding depth information created depth maps having slightly varying scale even when the model was the same. Therefore, the back-projected 3D points of these depth maps from two consecutive frames do not align properly. Figure 30 (a) illustrates this issue, where the level of the floors from two continuous frames are different.

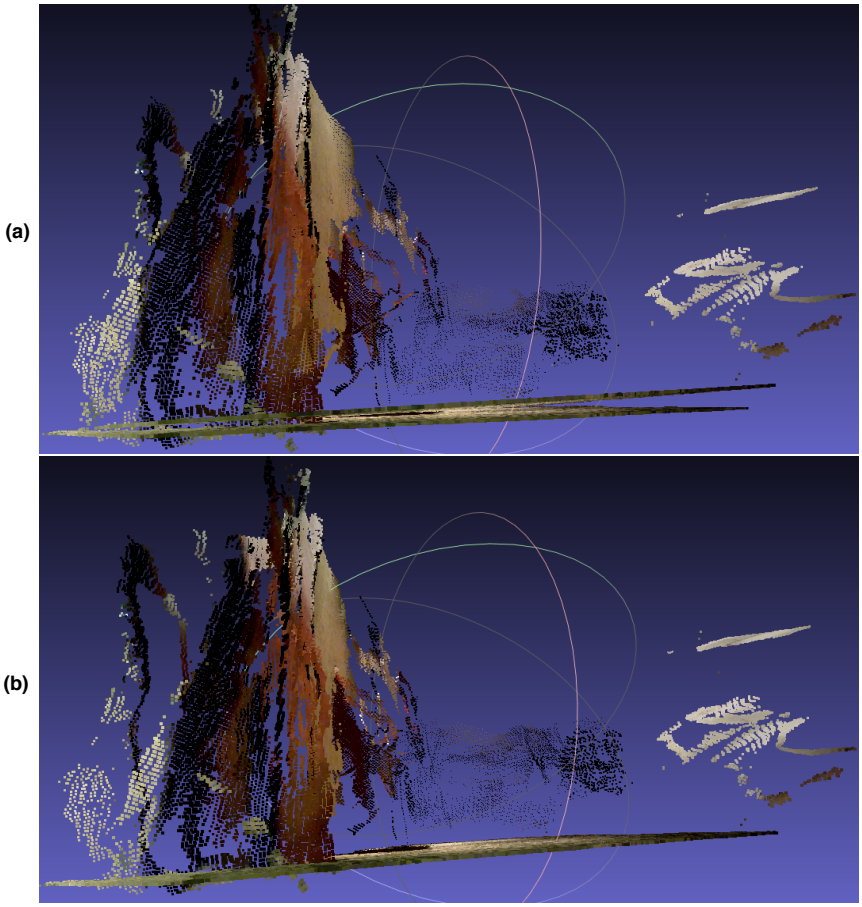

Figure 30. Depth map re-scaling. Point clouds from two consecutive depth maps without (a) and with (b) re-scaling.

To cope with this problem, the depth maps were re-scaled in the following manner. For every pair of consecutive depth maps $frame_t$ and $frame_{t+1}$ 1) re-project the $frame_{t+1}$ onto the $frame_t$ and 2) use RANSAC to find a scale factor which minimizes the error between the $frame_t$ and the re-projection of the $frame_{t+1}$.

After this, the pairwise scale factors were used to re-scale the depth maps into the scale of the first depth map. The back-projected point clouds of two consecutive frames after the re-scaling are presented in Figure 30 (b).

4.3.2. Point cloud registration

All re-scaled depth maps in Section 4.3.1 are back-projected to reconstruct 3D models. However, these overlapping point clouds are acquired from different views. Therefore, point cloud registration process such as the iterative closest point (ICP) [73, 74] is needed to align these point clouds into a single 3D model.

ICP is an algorithm which is used to register two point clouds so that a specific error between the clouds is minimized. This procedure starts by first determining a set of corresponding points between the source and target point cloud. In this case, a subset of floor and wall points from two point clouds was chosen. After that, the estimated

camera poses (e.g. acquired from MotionNet) were applied to point clouds as an initial guess for the transformation.

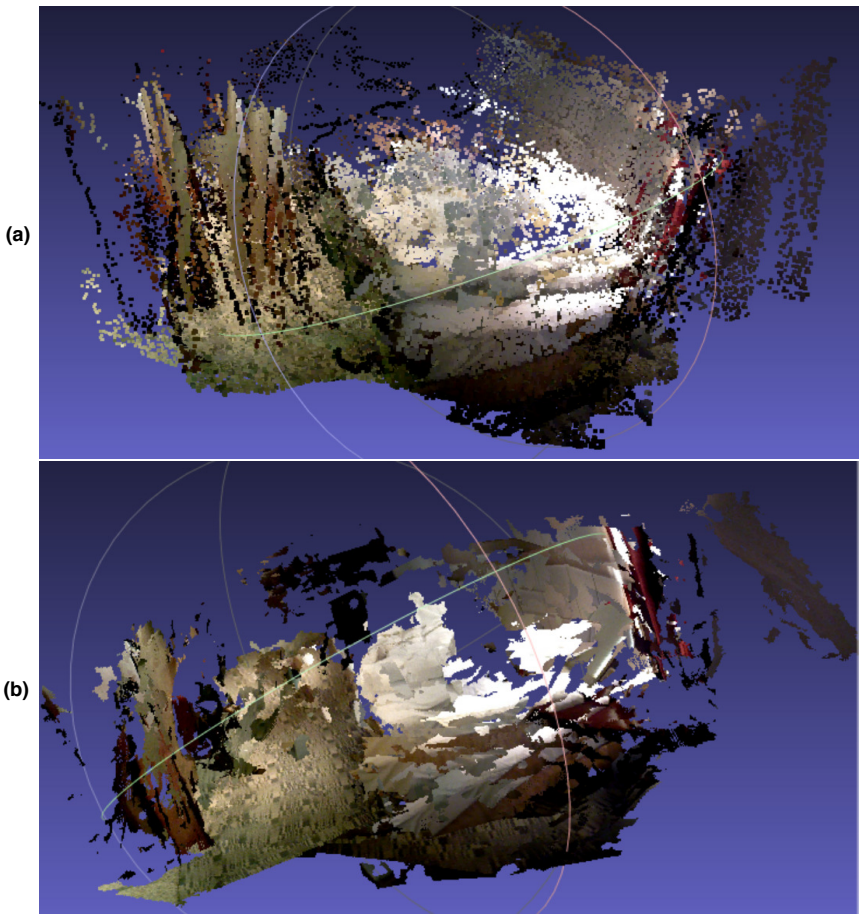

Figure 31. The improvement in the merged point cloud between without (a) and with (b) ICP implementation. As presented in image (a), even though the frames represent the same scene, the locations of floors and especially walls vary without ICP. In contrast using ICP, the merged point cloud in image (b) shows clear refinement having uniform floor and wall surfaces.

From these starting points, ICP iteratively finds the nearest target point for every source point. The obtained point pairs are then used to calculate the best transformation (contains camera rotation and translation) to register the source points to the target points by applying the least-squares method.

The coordinates of the source points are then updated using the computed transformation. The mean error distance between the source and target points is then calculated. If the error decreases iteration by iteration and reaches a pre-defined threshold, the ICP will stop and return the total transformation. Otherwise, the algorithm will repeat until the error is minimized or a maximum number of iterations is reached. Figure 31 shows a set of merged point clouds with and without ICP. The improvement in the point cloud alignment, when ICP is used, is clearly visible.

4.3.3. Simple merging process

As a summary, the main processes of this stage consist of 1) filtering the predicted depth maps based on their statistical errors, 2) re-scaling of the filtered depth maps, 3) ICP registration of the point clouds using the depth values and the estimated camera poses. The refined camera poses from ICP and the processed depth maps are finally concatenated into a single 3D point cloud representing the scene.

Unfortunately, the errors in the predicted depth maps and camera transformations affect the quality of the final merged point cloud at the moment. Therefore, a more sophisticated fusion is needed in the near future. One possible solution would be to apply the depth map fusion by Kyöstilä et al. [82]. The principle of that method is to iteratively fuse the depth maps into a single non-redundant point cloud. At every frame, the predicted depth values are back-projected into the 3D space. The new 3D points are then used to refine the existing points if they are close enough. Otherwise, the new points are inserted into the global point cloud.

Another method is to merge the predicted depth maps using iterative bundle optimization, as suggested by Li et al. [83]. The general idea is first to create depth maps from a set of overlapping images using trained depth model. Then, the inaccurate depth values are improved using a bundled track optimization followed by a position refinement and normal estimation.

5. EVALUATION

The quality of the reconstructed 3D point clouds heavily depends on the accuracy of both the depth map prediction and the camera pose estimation. This chapter first introduces the depth map and camera pose inference results from the proposed model with respect to state-of-the-art studies. Then, the final results are compared with the point clouds generated by conventional SfM methods. The purpose of this evaluation is to find the best learning-based SfM technique. This chapter is divided into five sections. The first section introduces the used evaluation metrics for the depth and camera pose prediction. Then, the next part describes the utilized datasets for the evaluations. Finally, the last three sections present the results from depth map, camera pose, and point cloud reconstruction tasks.

5.1. Metrics

Standard metrics that are extensively used to evaluate monocular depth prediction [7, 8, 9, 14, 17, 15] include the absolute relative error (rel), the squared relative error (sqr-rel), the root mean square error (rms), the root mean square error in log space (log-rms), the scale-invariant error (scale-inv) and the percentage thresholds (δ_i). On the other hand, the camera pose prediction errors are measured using the camera relative orientation error (ROE) and relative translation error (RTE). These metrics are described in the following sections.

5.1.1. Depth estimations

The errors are calculated pixel-wise for each predicted depth map regarding their corresponding ground truth where the depth information is available. In the following equations, the resolution of the predicted and ground truth images are both $m \times n$, $\hat{y}$ indicates the predicted depth values, and y_{gt} is the ground truth depth information.

Absolute relative (rel): $\frac{1}{m*n} \sum \frac{|\hat{y}-y_{gt}|}{y_{gt}}$

Square relative (sqr-rel): $\frac{1}{m*n} \sum \frac{(\hat{y}-y_{gt})^2}{y_{gt}}$

Root mean square error (rms): $\sqrt{\frac{1}{m*n} \sum (\hat{y} - y_{gt})^2}$

Log root mean square error (log-rms): $\sqrt{\frac{1}{m*n} \sum (\log \hat{y} - \log y_{gt})^2}$

Accuracy $\delta_i < 1.25^i$: $\frac{1}{m*n} \sum (|\log \hat{y} - \log y_{gt}| < \log 1.25^i)$

However, all of the previous metrics do not consider the varying scale between the predicted values and the ground truth. To suppress this issue, Eigen et al. [7] proposed the scale-invariant mean squared error (scale-inv), which calculates the pixel-wise mean square error disregarding the scaling factor. This metric is formulated as

$$\frac{1}{2(m*n)} \sum (\log \hat{y} - \log y_{gt} + \alpha(\hat{y}, y_{gt}))^2$$

where $\alpha(\hat{y}, y_{gt}) = \frac{1}{m*n}\sum(\log y_{gt} - \log \hat{y})$ is the regularization factor that is used to minimize the error of a given $(\hat{y}, y_{gt})$.

With the rel, srq-rel, rms, log-rms and scale-inv metrics the lower the values the better the results. Otherwise, in δ_1, δ_2 cases, the higher the better.

5.1.2. Camera pose estimations

The relative camera transformations are evaluated using the relative translation error (RTE)

$$\cos\theta_t = \frac{\vec{\hat{t}} \cdot \vec{t}_{gt}}{||\vec{\hat{t}}||\,||\vec{t}_{gt}||}$$

and the relative orientation error (ROE)

$$\cos\theta_R = \frac{trace\Big(\hat{R} \cdot (R_{gt})^T\Big) - 1}{2}$$

where $\vec{\hat{t}}$ is the predicted camera translation vector, $\vec{t}_{gt}$ is the ground truth translation, $\hat{R}$ indicates the predicted camera rotation matrix and R_{gt} is the ground truth rotation matrix.

The RTE measures the angle between the predicted translation vector $\vec{\hat{t}}$ and the ground truth vector $\vec{t}_{gt}$. RTE instead computes the angle between the estimated rotation matrix $\hat{R}$ and the ground truth rotation matrix R_{gt}. The unit for both RTE and ROE is degrees.

5.2. Datasets

The labeled set of the NYU Depth V2 [76] was used to evaluate the trained depth map prediction models. This part consists of 1449 densely labeled corresponding RGB and depth map pairs. Similar to other works like [7, 9, 10], 654-image pairs are used to test on the supervised and unsupervised depth networks. The quantitative and qualitative reports are presented in Section 5.3.

The camera pose prediction model was tested on a novel benchmark dataset called RGB-D SLAM [84]. The ego-motions in this dataset were captured using eight high-speed tracking cameras (100 Hz), which helped to create a high-fidelity ground-truth trajectory. The proposed MotionNet was compared with the methods in [84, 16] using two video sequences namely freiburg1_360 and freiburg1_desk. Both the RTE and ROE were evaluated. The former sequence contains 2769 images with valid timestamps, while the latter has 544 images. The results are presented in Section 5.4.

Finally, the 3D point clouds were reconstructed from the video number 025 of the SceneNN [78] and the real world datasets. The results are visually compared with the point clouds generated by conventional SfM software like VisualSfM [2] and Theia [5]. Section 5.5 is used to present the obtained outcomes.

5.3. Single view depth map prediction

The quantitative evaluation of the supervised and unsupervised DepthNets are presented in Table 1. The qualitative results comprising of the predicted depth maps and the reconstructed single view point clouds, are shown in Figures 33 and 32, respectively. The acquired results, even not as good as state-of-the-art, are still comparable. This is especially encouraging in the unsupervised DepthNet case, where the network was trained using only the RGB images.

Table 1. Quantitative results of single view depth map prediction. The results were evaluated on 654-images pairs from the labeled test set of the NYU Depth V2 [76]. With $rel, srq - rel, rms, log - rms$ and $scale - inv$ metrics, the lower the values the better the results. For δ_1, δ_2 metrics, the higher the better.

Methods/Metrics	**rel**	**sqr-rel**	**rms**	**log-rms**	**scale-inv**	$\boldsymbol{\delta_1}$	$\boldsymbol{\delta_2}$
Eigen et al. [7]	0.215	0.212	0.907	0.285	0.219	0.611	0.887
Laina et al. [9]	**0.127**	-	0.573	0.195	-	0.811	0.953
Liu et al. [10]	0.142	**0.107**	**0.514**	**0.179**	-	**0.812**	**0.957**
Supervised DepthNet	0.143	0.125	0.592	0.190	**0.208**	0.810	0.938
Unsupervised DepthNet	0.236	0.305	0.601	0.224	0.241	0.702	0.896

Although the evaluation accuracies of the trained DepthNets are lower than others [9, 10], they can still produce reasonable single view point cloud as shown in Figure 32. The dense reconstructions of the view are reasonable when comparing with the ground truth data.

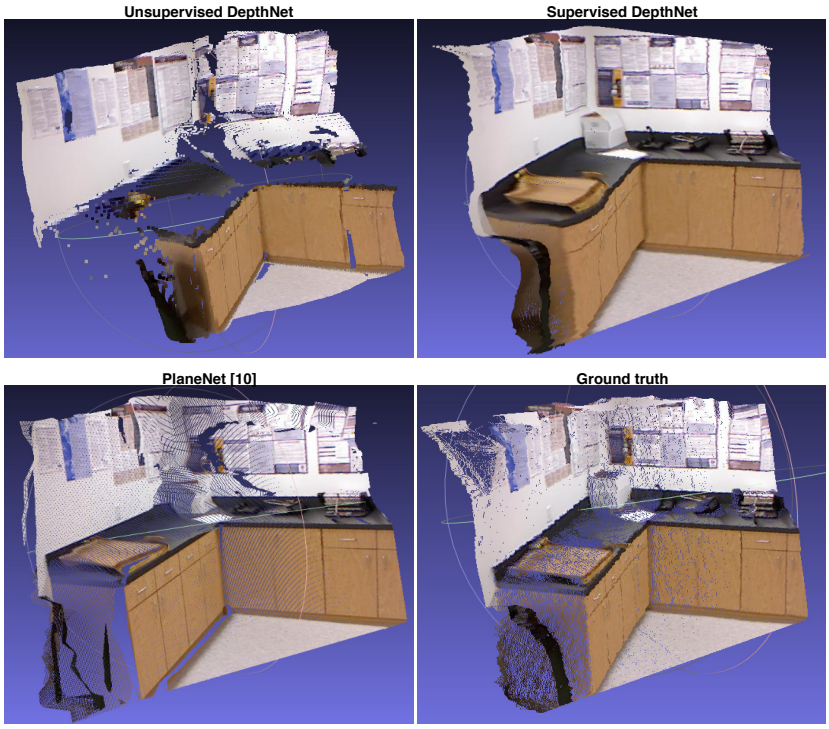

Figure 32. 3D reconstructions of single view point clouds from the predicted depth maps.

Figure 33 shows and compares example predictions from several depth maps estimation models including the supervised DepthNet (column b), the unsupervised DepthNet (column c) and the PlaneNet [10] (column d). While the depth images of the PlaneNet in case number $4, 6$ and 9 look better than the others, the supervised DepthNet result seems more favorable in complex case number 3. In general, the predicted depth maps from these models are quite similar and comparable with the ground truths (column e).

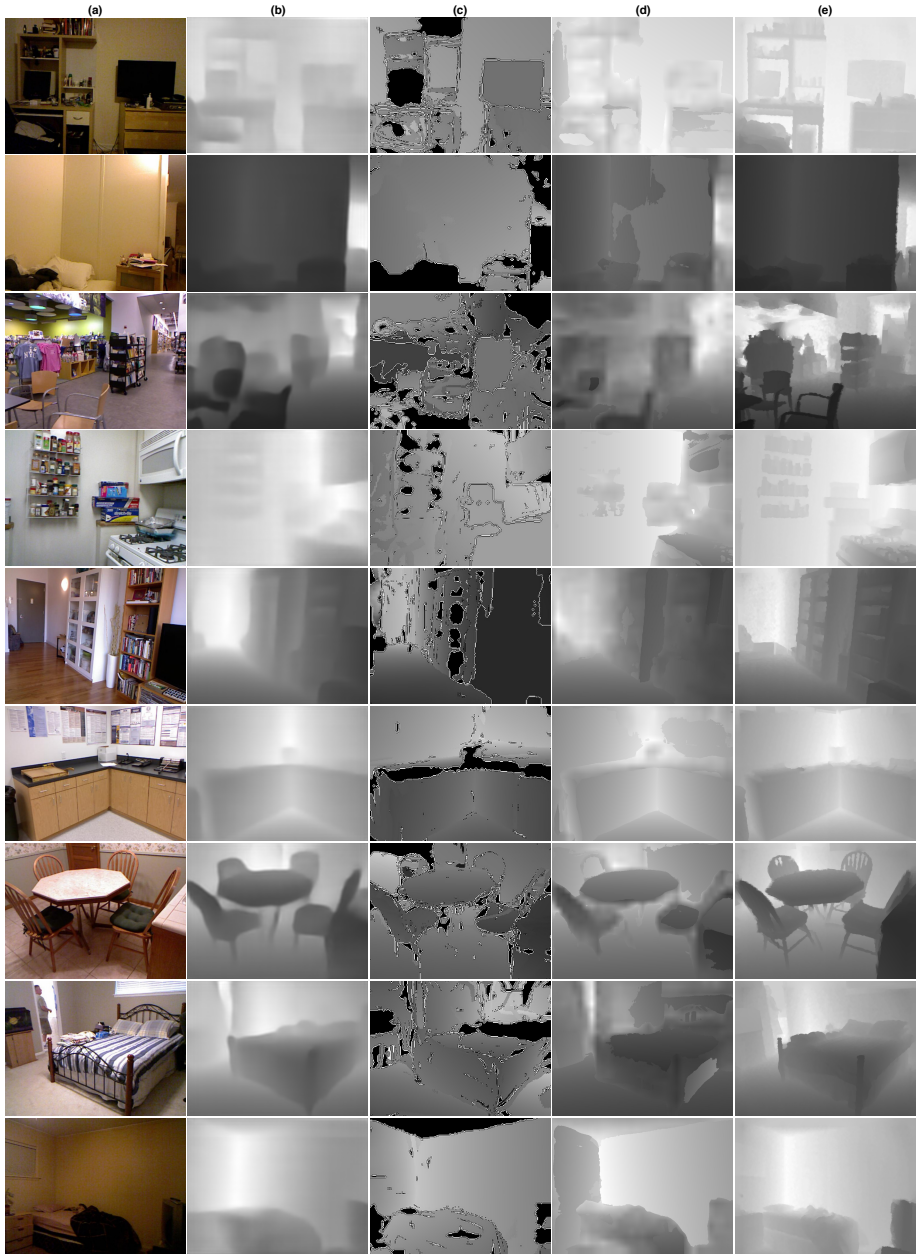

Figure 33. Depth map prediction of the supervised (column b) and unsupervised (column c) DepthNets on the NYU Depth v2 compare with Liu et el. [10] (column d). Column (a) and (e) show the input RGB and ground truth depth images, respectively.

5.4. Camera pose estimation

Quantitative results of the camera pose estimation task are presented in Table 2 and Figure 34. The camera transformations from the unsupervised MotionNet contain relatively big errors in the camera orientations. This issue, in return, caused further problems which are described later in Section 5.5.

Table 2. Camera pose estimation on the freiburg1_360 and freiburg1_desk sequences of the RGB-D SLAM dataset [84].

RGB-D SLAM	360		desk	
	RTE	ROE	RTE	ROE
Sturm et al. [84]	5.672	**27.158**	**0.458**	**28.361**
Vijay et al. [16]	**0.515**	64.343	0.687	48.586
Unsupervised MotionNet	3.054	33.640	2.168	38.287

However, the relative camera pose estimations can be used as initial guesses for the iterative closest point (ICP) [73, 74] algorithm. In addition, the predicted camera transformation and feature-based camera pose estimation can complement each other. If the number of matching feature-points between image pairs are too low (e.g less than five matches), the calculation of the essential matrix will fail. In this case, the system can use the predicted pose to initialize the ICP algorithm instead of the feature-based approach.

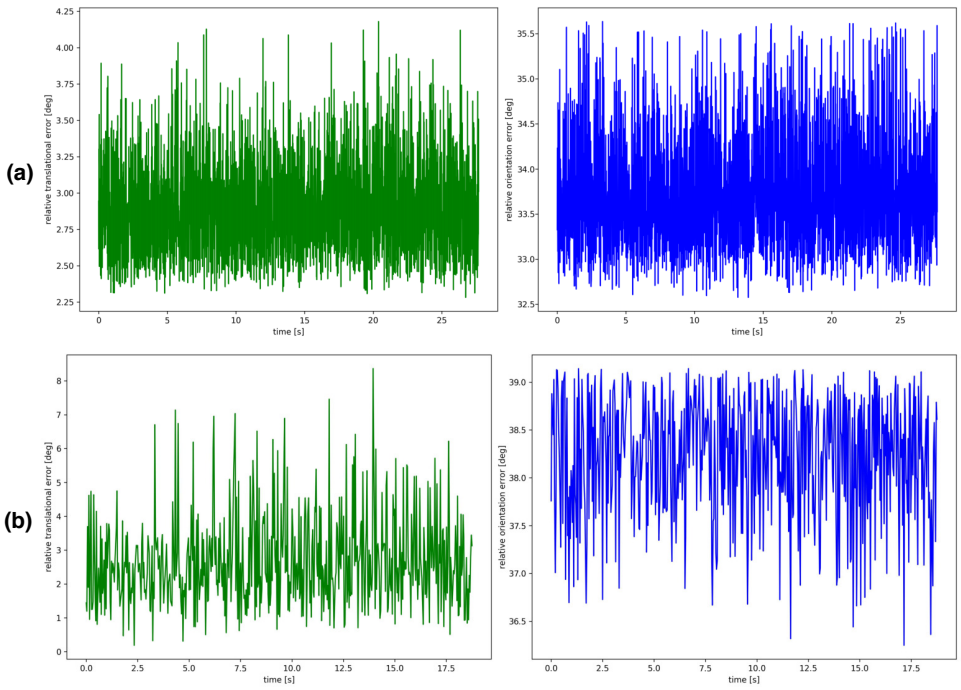

Figure 34. Visualization of the camera translation and orientation error with the respect to time. Row (a) presents the evaluation of the 360^o sequence, whilst row (b) shows the results with the $desk$ sequence. The left column presents the RTE while the right column shows the ROE.

5.5. Point cloud merging

This section discusses the final results of the 3D reconstruction pipeline. As shown in Figures 35, 36 and 37, the 3D point clouds of the scene are relatively incomplete. This is caused by the errors in both the predicted depth maps and camera transformations.

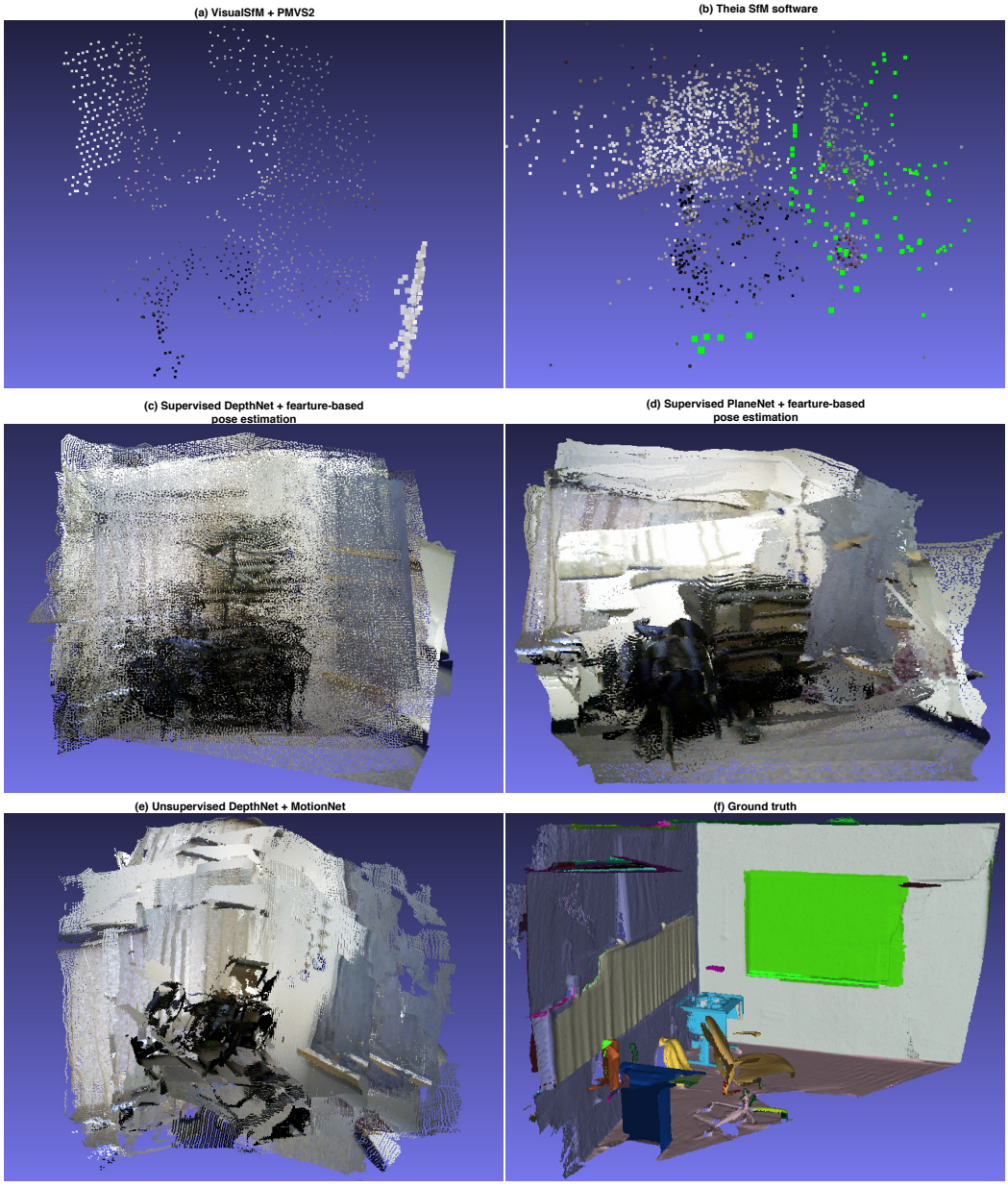

Figure 35. Reconstructed 3D point clouds of the view 025 in SceneNN dataset. Note that the green dots in the point cloud (b) are the estimated camera positions in the world coordinate system.

The simple merging process can produce reasonable point clouds when feeding a small collection of consecutive frames. Figure 35 shows the reconstruction made from about 50 frames of the scene number 025 in the SceneNN dataset [78]. The resolution of the input images was 640×480 pixels. The two point clouds (c) and (d) were created using two different supervised monocular depth models and feature-based pose

estimation, while point cloud (e) was built using the depth maps and camera pose estimation from the unsupervised networks. Even though the quality of the PlaneNet depth maps is better than the others (as shown in Table 1), the result point clouds (c), (d), and (e) are quite similar.

The point clouds in Figure 35 (a) and (b) were created using the structure from motion software VisualSfM [2] and Theia [5], respectively. As it commonly happens, the reconstruction of the scene broke at certain frame when the feature-based tracking failed to estimate the camera pose. This situation became even worse when using low-resolution or lack of texture input images. In these cases, the big scene point cloud was split into many smaller point clouds. These chunks of points are quite hard to recognize and have their own camera coordinate systems. Point clouds (a) and (b) present one of them.

Figure 36 shows the results using VisualSfM (a) and unsupervised deep learning models (b) from the real world dataset. This uses about 60 input images, which were taken with a calibrated mobile device with an image resolution of 1280×720 pixels. The 3D point cloud (b) still contains lots of noise, but nevertheless can reconstruct the shape of the scene especially in the floor area. In contrast, the point cloud (a) is almost empty even with 720p input images.

Figure 36. Reconstructed 3D point clouds from a real world dataset. The top row shows five input frames of the 60 total frames. Point cloud (a) was created using VisualSfM [2], which failed to reconstruct the scene. The learning based approach instead, was able to get a rough structure of the scene as shown in point cloud (b).

So far, the unsupervised DepthNet and MotionNet seem to be the most promising approaches because they can produce sufficient single view point cloud as well as the predicted camera poses. The supervised depth map prediction models and conventional SfM software have troubles when the feature-based technique can not estimate the camera poses.

However, the unsupervised model still has to cope with the errors that occur in the prediction. Figure 37 illustrates a situation where the merged point cloud was reconstructed from a set of 208 consecutive images from the SceneNN dataset [78]. In the merged point cloud, one can see that the left area of the wall behind the chair was wrongly reconstructed because of a large camera pose estimation error. This wall was quite well constructed although it was turned up side down compared with the ground

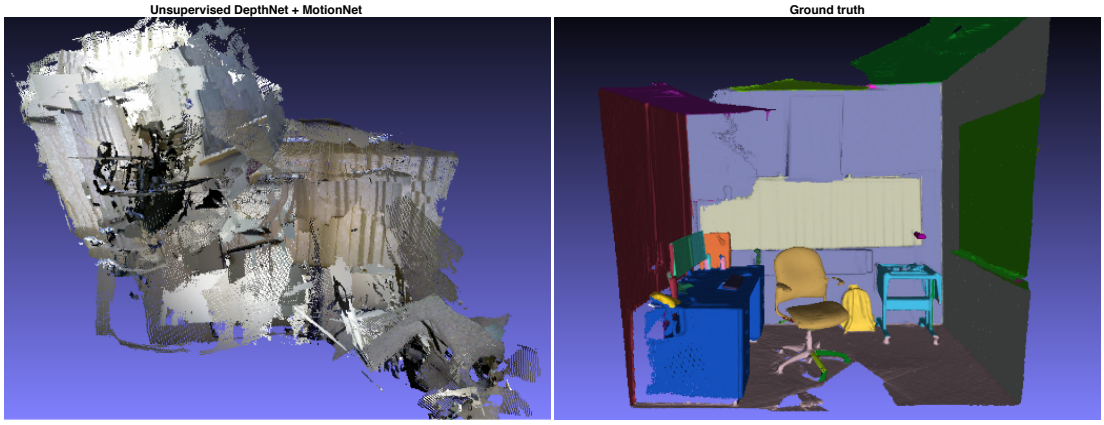

Figure 37. A failure case of the 3D point cloud reconstruction.

truth. This problem can be solved by the improvement of the predicted depth map and camera pose accuracy or implementation of a more sophisticated point cloud merging process.

6. DISCUSSION

This study focused on experiments with deep learning-based structure from motion approaches to reconstruct the dense 3D point cloud of a scene captured with a monocular camera. To achieve this goal, several deep neural networks were trained to predict the depth maps from single view images and relative camera transformations between consecutive frames. The first supervised network was trained to predict the depth values from input images. The second and third unsupervised networks were simultaneously trained to learn the depth and camera pose information of the novel view synthesis in a consistent way from inputting only RGB images. These predicted values were then used to rebuild the dense 3D point cloud of the recorded scene using a simple merging process.

Although the supervised network was easier to train and faster to converge, it always required a lot of carefully labeled data to increase the prediction accuracy. On the contrary, the unsupervised networks were much harder to train and produce relatively lower estimation accuracy. However, it is compelling that they only need the RGB information for the learning process. The trained unsupervised models do not only provide both the depth and camera pose cues, but also overcome the problem in feature-based technique such as feature-mismatch or insufficient matches. The dense 3D point clouds from the unsupervised models are fairly acceptable despite the fact that they just partly solve the reconstruction task. This seems to be a positive intimation for using the unsupervised learning scheme to tackle the 3D scene reconstruction problem.

In addition to increasing the accuracy of the predictions, both the supervised and unsupervised models could be trained with more datasets and initialized using the pre-trained weights. Especially in the unsupervised case, as long as the capturing camera is properly calibrated, it could be easy to collect up to millions of images because the networks will be trained using the pure RGB information. This will help to enhance the precision of monocular depth and camera pose estimation, which will lead to higher reconstruction quality in the near future.

To further improve the final results, more complex merging process should be implemented. The first prominent method is Li et al. [83], which merges the predicted depth maps using the iterative bundle optimization. In their paper, to minimize the errors from these estimated depth values, bundled track optimization is first applied. Next, the position refinement and normal estimation will be used to further optimize these inaccurate predictions. Another possible implementation is depth map fusion scheme from Kyöstilä et al. [82], that incrementally fuse the predicted depth maps into the global point cloud. In this process, every point in each depth map is back-projected into the 3D space and used to refine an existing nearby point if any or added to the point cloud otherwise.

The deep neural network design can be expanded to fully exploit the motion cues as well as integrate the semantic segmentation to better understand the context of the scenes. This can be achieved for example, by training a long short-term memory network (LSTM) [85] for learning the motion not only at the latest previous frames, but also at previous pivot frames where large motion or sudden direction changes appear. This will, in return, provide vital information for the unsupervised MotionNet. Another suggestion is the direct incorporation of semantic segmentation module into the unsupervised DepthNet. This idea originates from the study by Liu et al. [10], which

might boost the performance of the unsupervised models. Furthermore, the comprehension of the context will better solve the aperture problem than using the common smoothness which tends to also smooth out important information of the scene.

One possible future work is based on the study by Izadinia et al. [86], which proposed an interesting idea of rebuilding an indoor scene from a single image without the need to reconstruct everything. In this research, they trained several deep neural networks to perform an object detection and a room layout estimation. The room layout estimation model will fit and build the geometric layout of the room. Then, similar 3D models are simultaneously rendered and aligned with the detected objects to estimate the 3D pose of the objects. Finally, the error between the rendered room and the input image is iteratively minimized to find the best furniture placement for the scene.

Another future direction could be the extension of the unsupervised network to directly learn the 3D representation of a scene from 2D images. This idea is based on the recent work by Tulsiani et al. [87], who proposed a differentiable equation for calculating the gradient of the 3D volume from the 2D image. This novel idea allows backpropagation to learn directly the 3D representation from the 2D observation.

7. CONCLUSION

This thesis presented experiments with both supervised and unsupervised deep learning structure from motion approaches to solve the 3D scene reconstruction problem. Several deep neural network models were trained to predict the monocular depths and relative camera poses from consecutive frames. After the inferences, the predicted depth maps were uniformly scaled to the first depth map. Then, the estimated camera poses were refined using the iterative closest point algorithm. Finally, these scaled depth values and refined camera transformations were utilized to reconstruct the dense 3D point cloud of the scene. Although the final point clouds overcame the conventional structure from motion methods and produced encouraging results, the system is just partly resolving the 3D scene reconstruction problem. Most of the false cases were caused by the accumulated errors of the trained networks, especially in the camera orientation estimation.

Further developments in the future are required to deal with this problem thoroughly. The implementations of more sophisticated point cloud merging processes are necessary to cope with the errors from the predicted depth maps and camera poses. The performance of the deep neural network prediction can be improved by increasing the amount of training data, especially for the unsupervised DepthNet and MotionNet. Modifications in the network architecture could also be applied to understand the context of the scene better. This information in return will provide tremendous help for training the networks and improving the quality of the 3D point clouds. Besides, new 3D reconstruction schemes using deep neural networks can be applied.

8. REFERENCES

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

[2] Wu C. et al. (2011) Visualsfm: A visual structure from motion system .

[3] Wu C. (2013) Towards linear-time incremental structure from motion. In: 3D Vision-3DV 2013, 2013 International conference on, IEEE, pp. 127–134.

[4] Snavely N., Seitz S.M. & Szeliski R. (2006) Photo tourism: exploring photo collections in 3d. In: ACM transactions on graphics (TOG), vol. 25, ACM, vol. 25, pp. 835–846.

[5] Sweeney C., Hollerer T. & Turk M. (2015) Theia: A fast and scalable structure-from-motion library. In: Proceedings of the 23rd ACM international conference on Multimedia, ACM, pp. 693–696.

[6] Russakovsky O., Deng J., Su H., Krause J., Satheesh S., Ma S., Huang Z., Karpathy A., Khosla A., Bernstein M. et al. (2015) Imagenet large scale visual recognition challenge. International Journal of Computer Vision 115, pp. 211–252.

[7] Eigen D., Puhrsch C. & Fergus R. (2014) Depth map prediction from a single image using a multi-scale deep network. In: Advances in neural information processing systems, pp. 2366–2374.

[8] Eigen D. & Fergus R. (2015) Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 2650–2658.

[9] Laina I., Rupprecht C., Belagiannis V., Tombari F. & Navab N. (2016) Deeper depth prediction with fully convolutional residual networks. In: 3D Vision (3DV), 2016 Fourth International Conference on, IEEE, pp. 239–248.

[10] Liu C., Yang J., Ceylan D., Yumer E. & Furukawa Y. (2018) Planenet: Piecewise planar reconstruction from a single rgb image. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2579–2588.

[11] Kendall A., Grimes M. & Cipolla R. (2015) Posenet: A convolutional network for real-time 6-dof camera relocalization. In: Computer Vision (ICCV), 2015 IEEE International Conference on, IEEE, pp. 2938–2946.

[12] Melekhov I., Ylioinas J., Kannala J. & Rahtu E. (2017) Relative camera pose estimation using convolutional neural networks. In: International Conference on Advanced Concepts for Intelligent Vision Systems, Springer, pp. 675–687.

[13] Tateno K., Tombari F., Laina I. & Navab N. (2017) Cnn-slam: Real-time dense monocular slam with learned depth prediction. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), vol. 2, vol. 2.

[14] Garg R., BG V.K., Carneiro G. & Reid I. (2016) Unsupervised cnn for single view depth estimation: Geometry to the rescue. In: European Conference on Computer Vision, Springer, pp. 740–756.

[15] Godard C., Mac Aodha O. & Brostow G.J. (2017) Unsupervised monocular depth estimation with left-right consistency. In: CVPR, vol. 2, vol. 2, p. 7.

[16] Vijayanarasimhan S., Ricco S., Schmid C., Sukthankar R. & Fragkiadaki K. (2017) Sfm-net: Learning of structure and motion from video. arXiv preprint arXiv:1704.07804 .

[17] Zhou T., Brown M., Snavely N. & Lowe D.G. (2017) Unsupervised learning of depth and ego-motion from video. In: CVPR, vol. 2, vol. 2, p. 7.

[18] Abadi M., Barham P., Chen J., Chen Z., Davis A., Dean J., Devin M., Ghemawat S., Irving G., Isard M. et al. (2016) Tensorflow: A system for large-scale machine learning. In: OSDI, vol. 16, vol. 16, pp. 265–283.

[19] Longuet-Higgins H.C. (1981) A computer algorithm for reconstructing a scene from two projections. Nature 293, p. 133.

[20] Hartley R. & Zisserman A. (2003) Multiple view geometry in computer vision. Cambridge university press.

[21] Ullman S. (1979) The interpretation of structure from motion. Proc. R. Soc. Lond. B 203, pp. 405–426.

[22] Thrun S., Montemerlo M., Dahlkamp H., Stavens D., Aron A., Diebel J., Fong P., Gale J., Halpenny M., Hoffmann G. et al. (2006) Stanley: The robot that won the darpa grand challenge. Journal of field Robotics 23, pp. 661–692.

[23] Wikipedia contributors (2018), Darpa grand challenge — Wikipedia, the free encyclopedia. URL: `https://en.wikipedia.org/w/index.php?title=DARPA_Grand_Challenge&oldid=831265487`, [Online; accessed 26-May-2018].

[24] Harris C. & Stephens M. (1988) A combined corner and edge detector. In: Alvey vision conference, vol. 15, Citeseer, vol. 15, pp. 10–5244.

[25] Yu Q., Liang J., Xiao J., Lu H. & Zheng Z. (2018) A novel perspective invariant feature transform for rgb-d images. Computer Vision and Image Understanding 167, pp. 109–120.

[26] Hartley R.I. (1992) Estimation of relative camera positions for uncalibrated cameras. In: European conference on computer vision, Springer, pp. 579–587.

[27] Hartley R.I. & Sturm P. (1997) Triangulation. Computer vision and image understanding 68, pp. 146–157.

[28] Triggs B., McLauchlan P.F., Hartley R.I. & Fitzgibbon A.W. (1999) Bundle adjustment—a modern synthesis. In: International workshop on vision algorithms, Springer, pp. 298–372.

[29] Wu C., Agarwal S., Curless B. & Seitz S.M. (2011) Multicore bundle adjustment. In: Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on, IEEE, pp. 3057–3064.

[30] Furukawa Y. & Ponce J. (2010) Accurate, dense, and robust multiview stereopsis. IEEE transactions on pattern analysis and machine intelligence 32, pp. 1362–1376.

[31] Fischler M.A. & Bolles R.C. (1981) Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM 24, pp. 381–395.

[32] Krizhevsky A., Sutskever I. & Hinton G.E. (2012) Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems, pp. 1097–1105.

[33] Wolpert D.H. (1996) The lack of a priori distinctions between learning algorithms. Neural computation 8, pp. 1341–1390.

[34] LeCun Y., Kavukcuoglu K. & Farabet C. (2010) Convolutional networks and applications in vision. In: Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on, IEEE, pp. 253–256.

[35] Goodfellow I., Bengio Y., Courville A. & Bengio Y. (2016) Deep learning, vol. 1. MIT press Cambridge.

[36] Dumoulin V. & Visin F. (2016) A guide to convolution arithmetic for deep learning. arXiv preprint arXiv:1603.07285 .

[37] Zeiler M.D., Krishnan D., Taylor G.W. & Fergus R. (2010) Deconvolutional networks. In: Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on, IEEE, pp. 2528–2535.

[38] Zeiler M.D., Taylor G.W. & Fergus R. (2011) Adaptive deconvolutional networks for mid and high level feature learning. In: Computer Vision (ICCV), 2011 IEEE International Conference on, IEEE, pp. 2018–2025.

[39] Jia Y., Shelhamer E., Donahue J., Karayev S., Long J., Girshick R., Guadarrama S. & Darrell T. (2014) Caffe: Convolutional architecture for fast feature embedding. In: Proceedings of the 22nd ACM international conference on Multimedia, ACM, pp. 675–678.

[40] Al-Rfou R., Alain G., Almahairi A., Angermueller C., Bahdanau D., Ballas N., Bastien F., Bayer J., Belikov A., Belopolsky A. et al. (2016) Theano: A python framework for fast computation of mathematical expressions. arXiv preprint arXiv:1605.02688 472, p. 473.

[41] Lin M., Chen Q. & Yan S. (2013) Network in network. arXiv preprint arXiv:1312.4400 .

[42] Szegedy C., Liu W., Jia Y., Sermanet P., Reed S., Anguelov D., Erhan D., Vanhoucke V., Rabinovich A. et al. (2015) Going deeper with convolutions. Cvpr.

[43] Ioffe S. & Szegedy C. (2015) Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167 .

[44] Szegedy C., Vanhoucke V., Ioffe S., Shlens J. & Wojna Z. (2016) Rethinking the inception architecture for computer vision. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2818–2826.

[45] He K., Zhang X., Ren S. & Sun J. (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778.

[46] Szegedy C., Ioffe S., Vanhoucke V. & Alemi A.A. (2017) Inception-v4, inception-resnet and the impact of residual connections on learning. In: AAAI, vol. 4, vol. 4, p. 12.

[47] Yu F., Koltun V. & Funkhouser T.A. (2017) Dilated residual networks. In: CVPR, vol. 2, vol. 2, p. 3.

[48] Glorot X. & Bengio Y. (2010) Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics, pp. 249–256.

[49] He K., Zhang X., Ren S. & Sun J. (2015) Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE international conference on computer vision, pp. 1026–1034.

[50] Goodfellow I.J., Warde-Farley D., Mirza M., Courville A. & Bengio Y. (2013) Maxout networks. arXiv preprint arXiv:1302.4389 .

[51] Maas A.L., Hannun A.Y. & Ng A.Y. (2013) Rectifier nonlinearities improve neural network acoustic models. In: Proc. icml, vol. 30, vol. 30, p. 3.

[52] Rumelhart D.E., Hinton G.E. & Williams R.J. (1986) Learning representations by back-propagating errors. nature 323, p. 533.

[53] Yes you should understand backprop. `https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b/`. Accessed: 2018-05-05.

[54] Zou H. & Hastie T. (2005) Regularization and variable selection via the elastic net. Journal of the Royal Statistical Society: Series B (Statistical Methodology) 67, pp. 301–320.

[55] Srivastava N., Hinton G., Krizhevsky A., Sutskever I. & Salakhutdinov R. (2014) Dropout: A simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research 15, pp. 1929–1958.

[56] Nesterov Y. (1983) A method of solving a convex programming problem with convergence rate o (1/k2). In: Soviet Mathematics Doklady, vol. 27, vol. 27, pp. 372–376.

[57] Duchi J., Hazan E. & Singer Y. (2011) Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research 12, pp. 2121–2159.

[58] Kingma D.P. & Ba J. (2014) Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 .

[59] Ruder S. (2016) An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747 .

[60] Dai A., Chang A.X., Savva M., Halber M., Funkhouser T.A. & Nießner M. (2017) Scannet: Richly-annotated 3d reconstructions of indoor scenes. In: CVPR, vol. 2, vol. 2, p. 10.

[61] Cambridge landmarks dataset. `http://mi.eng.cam.ac.uk/projects/relocalisation/#dataset/`. Accessed: 2018-07-05.

[62] Shotton J., Glocker B., Zach C., Izadi S., Criminisi A. & Fitzgibbon A. (2013) Scene coordinate regression forests for camera relocalization in rgb-d images. In: Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on, IEEE, pp. 2930–2937.

[63] Zhou B., Lapedriza A., Xiao J., Torralba A. & Oliva A. (2014) Learning deep features for scene recognition using places database. In: Advances in neural information processing systems, pp. 487–495.

[64] Wilson K. & Snavely N. (2014) Robust global translations with 1dsfm. In: European Conference on Computer Vision, Springer, pp. 61–75.

[65] He K., Zhang X., Ren S. & Sun J. (2014) Spatial pyramid pooling in deep convolutional networks for visual recognition. In: european conference on computer vision, Springer, pp. 346–361.

[66] Wang Z., Bovik A.C., Sheikh H.R. & Simoncelli E.P. (2004) Image quality assessment: from error visibility to structural similarity. IEEE transactions on image processing 13, pp. 600–612.

[67] Mayer N., Ilg E., Hausser P., Fischer P., Cremers D., Dosovitskiy A. & Brox T. (2016) A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4040–4048.

[68] flickr: large sclae image and video hosting service. `https://www.flickr.com/`. Accessed: 2004-02-10.

[69] Aubry M., Maturana D., Efros A.A., Russell B.C. & Sivic J. (2014) Seeing 3d chairs: exemplar part-based 2d-3d alignment using a large dataset of cad models. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 3762–3769.

[70] Menze M. & Geiger A. (2015) Object scene flow for autonomous vehicles. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3061–3070.

[71] Dosovitskiy A., Fischer P., Ilg E., Hausser P., Hazirbas C., Golkov V., Van Der Smagt P., Cremers D. & Brox T. (2015) Flownet: Learning optical flow with convolutional networks. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 2758–2766.

[72] Cordts M., Omran M., Ramos S., Rehfeld T., Enzweiler M., Benenson R., Franke U., Roth S. & Schiele B. (2016) The cityscapes dataset for semantic urban scene understanding. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 3213–3223.

[73] Besl P.J. & McKay N.D. (1992) Method for registration of 3-d shapes. In: Sensor Fusion IV: Control Paradigms and Data Structures, vol. 1611, International Society for Optics and Photonics, vol. 1611, pp. 586–607.

[74] Chen Y. & Medioni G. (1992) Object modelling by registration of multiple range images. Image and vision computing 10, pp. 145–155.

[75] Deng J., Dong W., Socher R., Li L.J., Li K. & Fei-Fei L. (2009) ImageNet: A Large-Scale Hierarchical Image Database. In: CVPR09.

[76] Silberman N., Hoiem D., Kohli P. & Fergus R. (2012) Indoor segmentation and support inference from rgbd images. In: European Conference on Computer Vision, Springer, pp. 746–760.

[77] Song S., Lichtenberg S.P. & Xiao J. (2015) Sun rgb-d: A rgb-d scene understanding benchmark suite. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 567–576.

[78] Hua B.S., Pham Q.H., Nguyen D.T., Tran M.K., Yu L.F. & Yeung S.K. (2016) Scenenn: A scene meshes dataset with annotations. In: 3D Vision (3DV), 2016 Fourth International Conference on, IEEE, pp. 92–101.

[79] McCormac J., Handa A., Leutenegger S. & Davison A.J. (2016) Scenenet rgb-d: 5m photorealistic images of synthetic indoor trajectories with ground truth. arXiv preprint arXiv:1612.05079 .

[80] Chen L.C., Papandreou G., Kokkinos I., Murphy K. & Yuille A.L. (2018) Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. IEEE transactions on pattern analysis and machine intelligence 40, pp. 834–848.

[81] Zhou B., Zhao H., Puig X., Fidler S., Barriuso A. & Torralba A. (2016) Semantic understanding of scenes through the ade20k dataset. arXiv preprint arXiv:1608.05442 .

[82] Kyöstilä T., Herrera D., Kannala J. & Heikkilä J. (2013) Merging overlapping depth maps into a nonredundant point cloud. In: Scandinavian Conference on Image Analysis, Springer, pp. 567–578.

[83] Li J., Li E., Chen Y., Xu L. & Zhang Y. (2010) Bundled depth-map merging for multi-view stereo. In: Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on, IEEE, pp. 2769–2776.

[84] Sturm J., Engelhard N., Endres F., Burgard W. & Cremers D. (2012) A benchmark for the evaluation of rgb-d slam systems. In: Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on, IEEE, pp. 573–580.

[85] Hochreiter S. & Schmidhuber J. (1997) Long short-term memory. Neural computation 9, pp. 1735–1780.

[86] Izadinia H., Shan Q. & Seitz S.M. (2017) Im2cad. In: Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on, IEEE, pp. 2422–2431.

[87] Tulsiani S., Zhou T., Efros A.A. & Malik J. (2017) Multi-view supervision for single-view reconstruction via differentiable ray consistency. In: CVPR, vol. 1, vol. 1, p. 3.