FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

Abdelrahman Mostafa

PATCH-BASED 3D RECONSTRUCTION OF DEFORMING OBJECTS FROM MONOCULAR GREY-SCALE VIDEOS

Master's Thesis
Degree Programme in Computer Science and Engineering
June 2020

Mostafa A. (2020) Patch-Based 3D Reconstruction of Deforming Objects from Monocular Grey-Scale Videos. University of Oulu, Degree Programme in Computer Science and Engineering, 56 p.

ABSTRACT

The ability to reconstruct the spatio-temporal depth map of a non-rigid object surface deforming over time has many applications in many different domains. However, it is a challenging problem in Computer Vision. The reconstruction is ambiguous and not unique as many structures can have the same projection in the camera sensor.

Given the recent advances and success of Deep Learning, it seems promising to use and train a Deep Convolutional Neural Network to recover the spatio-temporal depth map of deforming objects. However, training such networks requires a large-scale dataset. This problem can be tackled by artificially generating a dataset and using it in training the network.

In this thesis, a network architecture is proposed to estimate the spatio-temporal structure of the deforming object from small local patches of a video sequence. An algorithm is presented to combine the spatio-temporal structure of these small patches into a global reconstruction of the scene. We artificially generated a database and used it to train the network. The performance of our proposed solution was tested on both synthetic and real Kinect data. Our method outperformed other conventional non-rigid structure-from-motion methods.

Keywords: NRSFM, deformable objects, spatio-temporal reconstruction, deep learning

TABLE OF CONTENTS

FOREWORD

This thesis was written at the Center of Machine Vision and Signal Analysis (CMVS). I would like to thank Prof. Janne Heikkilä and Matteo Pedone for the guidance I received while doing this work. I am grateful to all who helped me during my study.

Oulu, June 16th, 2020

Abdelrahman Mostafa

LIST OF ABBREVIATIONS AND SYMBOLS

Adam	Adaptive Moment
AF	Activation Function
AI	Artificial Intelligence
ANN	Artificial Neural Network
AR	Augmented Reality
BN	Batch Normalization
CNN	Convolutional Neural Network
CV	Computer Vision
DCNN	Deep Convolutional Neural Network
DCT	Discrete Cosine Transform
DL	Deep Learning
FC	Fully Connected
GBR	Generalized Bas-Relief
GD	Gradient Descent
GPU	Graphical Processing Unit
MAE	Mean Absolute Error
ML	Machine Learning
NRSFM	Non Rigid Structure From Motion
LReLU	Leaky Rectified Linear Unit
PS	Photometric stereo
RAM	Random Access Memory
ReLU	Rectified Linear Unit
RMSD	Root Mean Squared Distance
SGD	Stochastic Gradient Descent
SIFT	Scale-Invariant Feature Transform
SNMAE	Spatially Normalized Mean Absolute Error
SVD	Singular Value Decomposition
SVM	Support Vector Machines

1. INTRODUCTION

Reconstructing the 3D surface of a deformable object is an active area of research in the Computer Vision community. Humans rely on certain cues to construct the shape of an object such as the changes in shading or object texture. A lot of research has been conducted to try to mimic the human capabilities in solving this problem. While it might seem quite an easy task for humans, it is a challenging and an ambiguous problem to be solved by machines. Ambiguities arise when we try to estimate the 3D shape from the projected 2D image in the camera image plane. Hence, this inverse problem is an ill-posed problem.

Recovering the shape of non-rigid objects has many applications such as Augmented Reality, animation movies or video games in entertainment industry. Medical applications can move towards being less invasive. All human organs are deforming and the ability to reconstruct the structure of such organs can be of a great use to surgeons. In industry, this can be beneficial for redesigning some parts or piece of equipment based on the deformations that occur to these parts. For example, plane wings might need to be redesigned if needed.

Using more than one camera will add more information and will make the problem of recovering the shape theoretically easier to be solved. However, from a practical point of view, it is more tedious and demanding to achieve such a setup. For example, the cameras need to be calibrated and synchronised. Hence, the ability to solve this problem using a video from a single camera can be more practical for applications.

Various methods have been proposed to solve this problem. Template-based models use some template as a reference to construct point correspondences between the object and the reference template. However, this limits the practicality of these method as they assume the 3D object shape or shape basis to be known. Non-rigid structure-from-motion (NRSFM) algorithms [1] overcome this limitation by using multiple frames or a video where the object can be observed. Hence, NRSFM methods exploit the availability of a video sequence without the need for the object shape to be known. However, this increases the number of unknowns or degrees of freedom for the problem as the basis shapes need to be calculated. Most NRSFM methods use some variant of the factorization algorithm [2] to solve this problem. The factorization algorithm is a method based on SVD that decomposes a matrix representing a sequence of 2D point clouds into the product of a sequence of camera motion matrices and the corresponding 3D point clouds. This factorization is ambiguous to an affine transformation. In this case, more constraints need to be applied to obtain acceptable results. Photometric stereo (PS) [3] methods use multiple images taken simultaneously under different known lighting conditions to estimate the surface normals. Classical PS methods assume the scenes to be static. For dynamic scenes that may contain rigid or non-rigid objects, dynamic PS methods were proposed to estimate the surface normals of dynamic scenes. However, PS methods assume some lighting or illumination conditions and also assume some reflectance model for the object surface.

Given the limitations and assumptions made by the previous methods and the fact that Deep learning (DL) has proved to be very successful in solving many problems recently, is it possible to exploit some DL techniques to solve this problem by constructing a depth map of the scene? This question motivated our work.

One of the main challenges is obtaining a database for network or model training. The database should contain videos of deforming objects along with the corresponding depth map for each frame to be used for network supervision. The database needs to contain many videos of deforming objects having different textures and observed in different lighting conditions. By doing this, we ensure that we eliminate any assumptions about illumination conditions or object physical properties. Acquiring such a large database using depth-sensing devices such as Microsoft Kinect is not a good idea. It is a laborious and time consuming process. The depth information obtained by these devices is noisy and sometimes some data is missing. This motivated us to generate a synthetic database to use it in training and testing the network.

In this thesis, Chapter 2 introduces the NRSFM approach and some state-of-the-art NRSFM methods. Chapter 3 presents an overview of DL and related concepts. Chapter 4 discusses some of the techniques and network architectures used to solve monocular depth estimation and similar problems. Chapter 5 presents the proposed solution using DL and an algorithm to reconstruct a large scene from small video patches. Chapter 6 shows some experiments and comparison between different methods using synthetic and real Kinect data. The last chapters are for some discussion and conclusion.

2. NRSFM CONVENTIONAL METHODS

This chapter presents the non-rigid structure-from-motion (NRSFM) approach used to construct the 3D structure of deforming objects. The first section describes the NRSFM approach. The second section discusses some of the ambiguities related to 3D reconstruction. The third section introduces some conventional state-of-the-art NRSFM methods.

2.1. Non Rigid Structure from Motion

Unlike template-based approach, NRSFM is template-free i.e no template or reference mesh for the shape of the object is needed [1]. In practice, this reference mesh can not always be available. However, NRSFM requires frame-to-frame 2D point correspondences for the video sequence [4]. The key points or points of interest in frames can be detected using features detector such as Scale-Invariant Feature Transform (SIFT) detector [5] or Harris corner detector [6]. The features then need to be matched to get correspondences throughout the frames of the video. Hence, NRSFM methods exploit the availability of a video sequence without the need for the shape of the object to be known. NRSFM methods rely on this information to construct the shape of the object in addition to the motion of the camera usually using a variant of the factorization algorithm [2]. It is a method that decomposes a matrix representing a sequence of 2D point clouds into the product of a sequence of camera motion matrices and the corresponding 3D point clouds. Hence, NRSFM algorithms compute the 3D locations of these key points to construct the shape of the non-rigid object. Figure 1 shows an example of a deforming object and the corresponding 3D structure obtained by NRSFM methods. Note that most NRSFM methods do not produce dense construction but only 3D representation for the key tracked points. The dense construction can then be obtained by scattered interpolation or other approaches like the one used in [7].

2.1.1. Non-Rigid Factorization under Orthographic Projection

Orthographic projection is a parallel projection where all projection lines are orthogonal to the projection or image plane.

Given n image points on an object, the transformation from world coordinates to the image plane for an orthographic camera can be represented as:

$$\begin{bmatrix} u_1 & \dots & u_n \\ v_1 & \dots & v_n \end{bmatrix} = \boldsymbol{RP} + \boldsymbol{T} \tag{1}$$

where $\boldsymbol{R}$ is a 2×3 matrix that has the first 2 rows of a rotation matrix, $\boldsymbol{P}$ is a $3 \times n$ matrix representing the 3D points tracked on the object and $\boldsymbol{T}$ is a $2 \times n$ translation matrix.

For simplicity, we can assume that the origin of the world coordinates is the centroid of the object and center the 2D points in the image plane to have zero mean as:

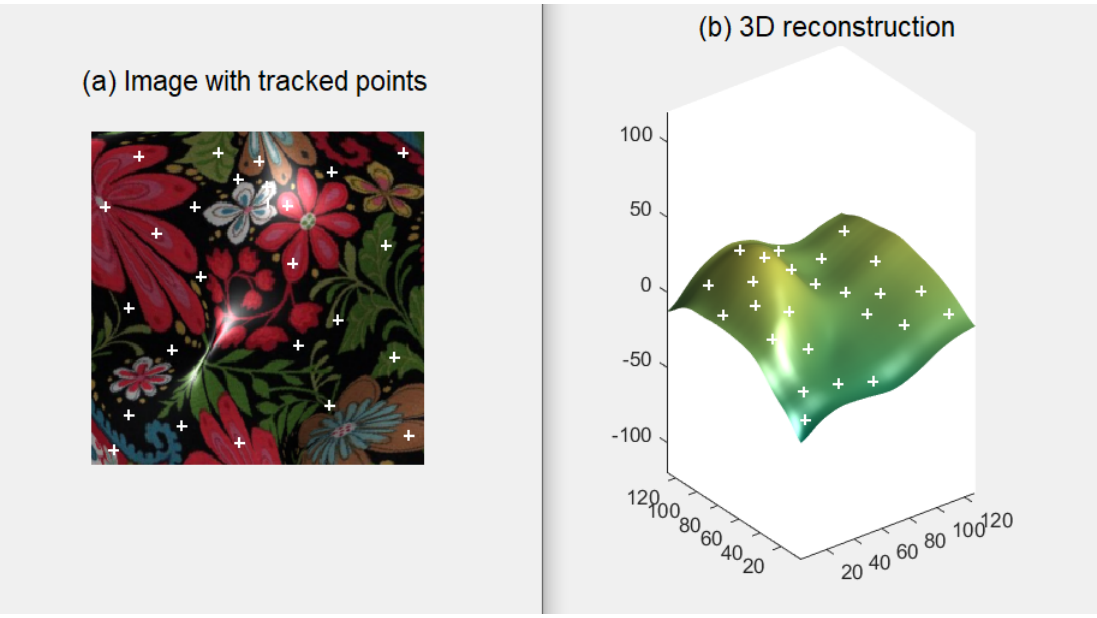


Figure 1. Left: single frame of a video capturing a deforming object with the tracked 2D points represented by the center of white plus signs. Right: the corresponding 3D representation for the tracked points represented by the center of white plus signs that is obtained using NRSFM methods. A dense representation for the structure of the deforming object is obtained by interpolating these scattered 3D key points.

$$\hat{u}_i = u_i - \frac{1}{n}\sum_{i=1}^{n} u_i, \qquad \hat{v}_i = v_i - \frac{1}{n}\sum_{i=1}^{n} v_i \tag{2}$$

Hence, we can omit the translation vector $\boldsymbol{T}$. Equation (1) can then be rewritten as:

$$\begin{bmatrix} \hat{u}_1 & \dots & \hat{u}_n \\ \hat{v}_1 & \dots & \hat{v}_n \end{bmatrix} = \boldsymbol{RP} \tag{3}$$

We can assume that the shape can be represented by n_s linear combination of basis shapes. So, Equation (3) can be rewritten as:

$$\begin{bmatrix} \hat{u}_1 & \dots & \hat{u}_n \\ \hat{v}_1 & \dots & \hat{v}_n \end{bmatrix} = \boldsymbol{R}\sum_{i=1}^{n_s} c_i \boldsymbol{S_i} \tag{4}$$

where $\boldsymbol{S_i}$ is a $3 \times n$ matrix of the i^{th} basis shape and c_i is the corresponding coefficient of the i^{th} basis shape.

Given n_f frames of a video with n tracked 2D points, we can combine the points from all frames as:

$$\boldsymbol{W} = \begin{bmatrix} \hat{u}_1^1 & \dots & \hat{u}_n^1 \\ \hat{v}_1^1 & \dots & \hat{v}_n^1 \\ \vdots & \ddots & \vdots \\ \hat{u}_1^{n_f} & \dots & \hat{u}_n^{n_f} \\ \hat{v}_1^{n_f} & \dots & \hat{v}_n^{n_f} \end{bmatrix} = \begin{bmatrix} c_1^1 \boldsymbol{R^1} & \dots & c_{n_s}^1 \boldsymbol{R^1} \\ \vdots & \ddots & \vdots \\ c_1^{n_f} \boldsymbol{R^{n_f}} & \dots & c_{n_s}^{n_f} \boldsymbol{R^{n_f}} \end{bmatrix} \begin{bmatrix} \boldsymbol{S_1} \\ \vdots \\ \boldsymbol{S_{n_s}} \end{bmatrix} = \boldsymbol{MS} \tag{5}$$

where $\boldsymbol{W}$ is the measurement matrix, $\boldsymbol{M}$ is the motion matrix and $\boldsymbol{S}$ is the basis shapes matrix.

We can get $\boldsymbol{M}$ and $\boldsymbol{S}$ matrices using Singular Value Decomposition (SVD) [8]. However, the solution is not unique and there is an affine ambiguity. Hence, an affine corrective matrix $\boldsymbol{Q}$ is needed that has an inverse as can be seen in following equation:

$$\boldsymbol{W} = \boldsymbol{MS} = (\boldsymbol{MQ^{-1}})(\boldsymbol{QS}) \tag{6}$$

The factorization algorithm or some of its variants are used in most NRSFM algorithms. The next section discusses some ambiguities that are related to the problem of 3D reconstruction.

2.2. Ambiguities Related to 3D Reconstruction

Ambiguities do exist for this problem. That is basically because we lose information after projection and moving from the 3D world to the 2D image plane. As the space of 3D world is much bigger than the 2D space of the image plane, we can think of it as many or infinitely many states in the 3D world space map to the same state in the 2D space. That means that the inverse problem is ambiguous when we need to get back to the 3D world from the 2D space. So, one state in the 2D plane can map to infinitely many states in the 3D world. Hence, which state is the real one that represents the original state of the object? Impossible to know as we already have lost much information after projection.

2.2.1. Affine Ambiguity

From Equation (6), we can see that an affine corrective matrix Q is needed in order to remove the affine ambiguity. So, NRSFM methods suffer from this ambiguity which can be seen in the factorization algorithm.

2.2.2. GBR Ambiguity

For orthographic camera model and Lambertian surfaces illuminated by directional light sources, the ambiguities are known as Generalized Bas-Relief (GBR) [9] transformations in addition to a translation ambiguity. The GBR consists of scale and shear transformations along the z-axis which is the optical axis. This ambiguity appears in Photometric Stereo (PS) reconstruction. The next part describes each

transformation or ambiguity (translation along z-axis, scale along z-axis and shear along z-axis). For each case, the transformation is applied to the original structure of object shown in Figure 1 (b). The mathematical representation for these ambiguities is presented in Chapter 5 (see Section 5.3).

Translation along z-axis Ambiguity

For orthographic projection, any translated version of the object along the z-axis would yield exactly the same image in the image plane (after applying the corresponding transformation to the directional light source). That is because the projection lines are parallel to the optical axis. Hence, we will get the same projection in the image plane. Figure 2 (a) illustrates the translation along z-axis ambiguity.

Scale along z-axis Ambiguity

For orthographic projection, any scaled version of the object along the z-axis would yield exactly the same image in the image plane. Figure 2 (b) illustrates the scale along z-axis ambiguity.

Shear along z-axis Ambiguity

For orthographic projection, any sheared version of the object along the z-axis would yield exactly the same image in the image plane. Figure 2 (c) illustrates the shear along z-axis ambiguity.

All Ambiguities

For orthographic projection, any object that undergoes a transformation that include translation along the z-axis operation, scale along z-axis operation or shear along z-axis operation would yield exactly the same image in the image plane. Figure 2 (d) illustrates all previous ambiguities combined. Note again that all these structures would have exactly the same image in the image plane as in Figure 1 (a) after accounting for illumination.

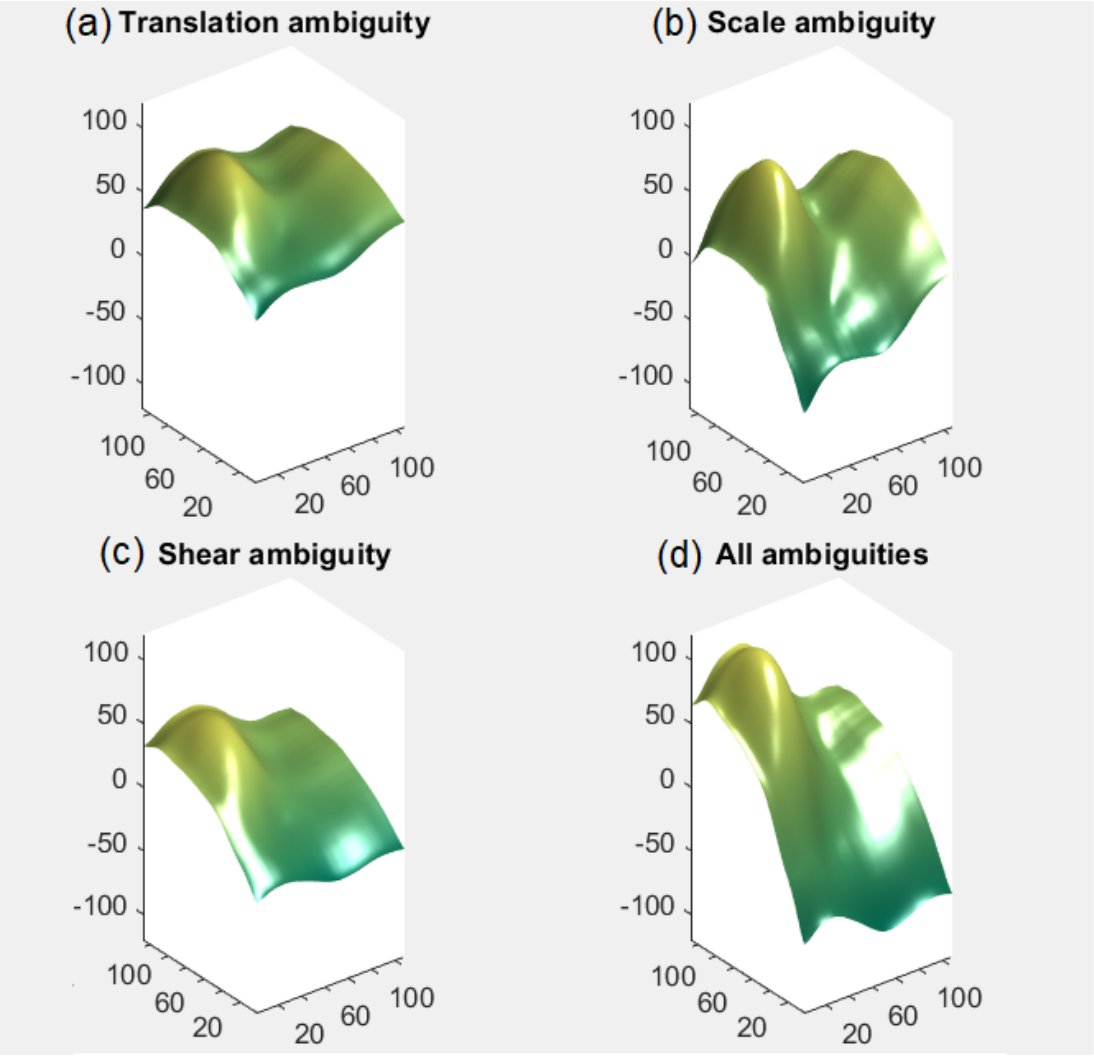


Figure 2. Top left: the object can be translated to any distance along the optical axis or z-axis from an orthographic camera and it would have exactly the same appearance in the image plane as in the left figure in Figure 1. Top right: the object can be scaled along the z-axis and it would have exactly the same appearance in the image plane. Bottom right: the object can be sheared along the z-axis and it would have exactly the same appearance in the image plane. Bottom right: the object can undergo a general transformation that includes translation along z-axis, scale along z-axis or shear along z-axis and it would have exactly the same appearance in the image plane.

When we discuss the proposed solution in Chapter 5, we will see that these ambiguities, which appear when we use the synthetic data in training the network, will force us to follow some approach in order to achieve better performance.

2.3. State-Of-The-Art NRSFM Methods

This section presents some of the state-of-the-art NRSFM methods.

2.3.1. Kernel Shape Trajectory Approach (KSTA)

In [10], the authors introduced the kernel trick [11] used in Support Vector Machines (SVM) to the traditional NRSFM method. As was previously illustrated, NRSFM methods use a linear combination of basis shapes to represent the deformations of the object. However, the linearity assumption is not always true and most of the time these methods need many basis shapes to try to capture the shape deformations. By introducing the kernel trick, the authors were able to capture non-linear relationships in the basis shapes coefficients. The authors were able to obtain better results by using this trick.

2.3.2. CSF2

The authors in [12] developed a novel NRSFM algorithm that uses the standard factorization method in addition to modeling the trajectories of the 3D points. The trajectories of the 3D points are modeled using basis vectors in the Discrete Cosine Transform (DCT) [13] domain. The authors claimed that this new algorithm improved the results and could capture high frequency deformations as well.

2.3.3. Dense Depth Estimation without Explicit 3D Motion Estimation (DDE)

In [14], the authors proposed an algorithm to recover a dense representation or a depth map of a dynamic scene without solving for the 3D motion parameters. They assumed the scene to be of piece-wise planar model and a global as-rigid-as-possible transformation between consecutive frames. Given two frames, the per-pixel optical flow and a sparse depth representation for the first reference frame, the algorithm recovers the dense representation for the second frame. The algorithm can also be applied to multiple frames or a video sequence.

The next two chapters give an overview of Deep Learning (DL) and related concepts and discuss some solutions that were used to solve the problem of monocular depth estimation and similar problems.

3. DEEP LEARNING OVERVIEW AND NETWORK ARCHITECTURES

Deep Learning (DL) proved to be very successful in solving many problems related to computer vision in recent years. Hence, it captured the attention of many researchers in the community. In this chapter, the first section gives a brief introduction or an overview of DL and other related important concepts. The second section discusses U-Net which will be used in the proposed solution introduced in Chapter 5.

3.1. Deep Learning Overview and Important Concepts

DL is a branch of Machine Learning (ML) that uses Deep Neural Networks (DNNs) in the learning process. In contrast to Artificial Neural Networks (ANNs), DNNs have many hidden layers - hence the name "Deep Neural Networks". ML is a subset of Artificial Intelligence (AI). AI tries to make machines intelligent or smart. This can be done by explicitly programming the machines to perform some tasks. However, no learning is achieved by the machines. ML tries to make machines learn by providing examples or data to learn from. Learning can be achieved using labeled data which means the output for each input is provided i.e data have labels. This is called supervised learning as the network has both the input and the output. The network tries to learn how to map the input to produce the output. Whereas unsupervised learning uses unlabeled data and the network has only the input. Here, the network tries to categorize different inputs or produce some output by achieving some consistencies. The following subsections discuss some important concepts and building blocks of neural networks.

3.1.1. Perceptron

Perceptron is the building block of ANNs or DNNs. A perceptron is also called an artificial neuron as it mimics the neurons in human brain. A neuron in a human brain produces a signal which is the output of this neuron based on the signals it receives from different neurons which this neuron is connected to. These signals that the neuron receive are the input to this neuron. A perceptron is the mathematical realization or model of a human neuron. Figure 3 shows the mathematical model of an artificial neuron which can be written as:

$$y = \varphi(\sum_{i=1}^{n} x_i w_i + w_0) = \varphi(\boldsymbol{W}\boldsymbol{X}), \tag{7}$$

where w_0 is the bias and φ is the activation function. $\boldsymbol{X} = [1\ x_1\ \cdots\ x_n]^T$ is the inputs vector and $\boldsymbol{W} = [w_0\ w_1\ \cdots\ w_n]$ is the weights vector using vector notation.

For the classical Perceptron, the activation function φ is a step function which outputs $+1$ or -1. These outputs represent two classes and the Perceptron was used to solve classification problems in ML. More details on activation functions are discussed

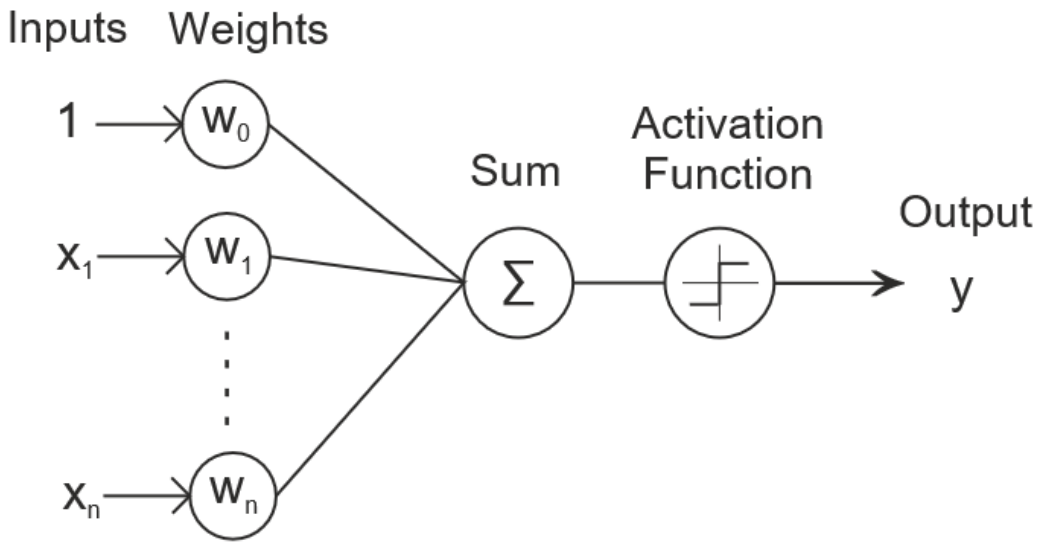


Figure 3. The mathematical model of a perceptron.

later in Subsection 3.1.3. For ANNs, the activation function introduces non-linearity to the model.

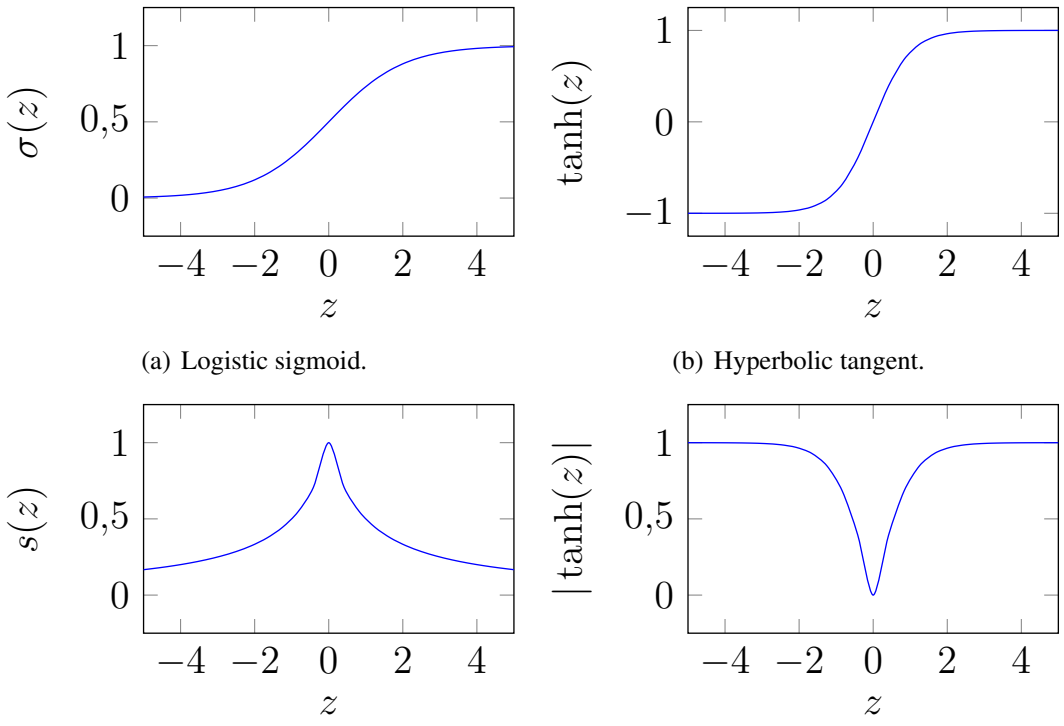
3.1.2. ANN

The building block of (ANNs) is the perceptron. The perceptron mimics the neuron in the brain and the ANN mimics the brain itself. An ANN is composed of many perceptrons organised in a specific structure. Introducing more artificial neurons to the network makes the network able to realize more complex functions.

3.1.3. Activation Function

The purpose of the activation function is to introduce non-linearity to ANNs. By introducing this function, the ANN will not be limited to learning lines and planes but it will be able to learn more complex functions and curves. Of course, this is of great benefit as the problems are not so simple to be solved using linear models. Figure 4 shows some of the activation functions known in the literature especially the logistic sigmoid $\sigma(z)$ and the hyperbolic tangent $tanh(z)$. The logistic sigmoid and the hyperbolic tangent functions are on the form:

$$\sigma(z) = \frac{1}{1+e^{-z}} \qquad \tanh(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}, \tag{8}$$



(a) Logistic sigmoid.

(b) Hyperbolic tangent.

(c) Softsign.

(d) Rectified hyperbolic tangent.

Figure 4. Activation functions that are commonly used. The logistic sigmoid $\sigma(z)$ and the hyperbolic tangent $tanh(z)$ are pretty known in the literature. More recently used activation functions are the softsign and the rectified hyperbolic tangent.

A rectified hyperbolic tangent is a hyperbolic tangent that is rectified to produce only positive values i.e the absolute of a hyperbolic tangent function. The softsign function is on the form:

$$s(z) = \frac{1}{1 + |z|}, \tag{9}$$

The slope of these functions is close to zero for most of the function domain. This causes the vanishing gradient problem [15] especially for deeper networks. Weights are updated during back propagation. If the gradient vanishes, these weights will not be updated (see Equation (12)) and the network will not be able to learn. Another problem is the exploding gradient which is the opposite of the vanishing gradient. This happens if the gradient is very large. These problems hinder the learning process of the network. They especially arise when training very deep networks. This forced the researchers to try to find more suitable activation functions. Now, Rectified Linear Unit (ReLU) [16] and its variants are widely used by the community researchers. Another advantage of ReLU is its simplicity. ReLU activation function and its derivative are on the form:

$$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases} \qquad f'(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases} \tag{10}$$

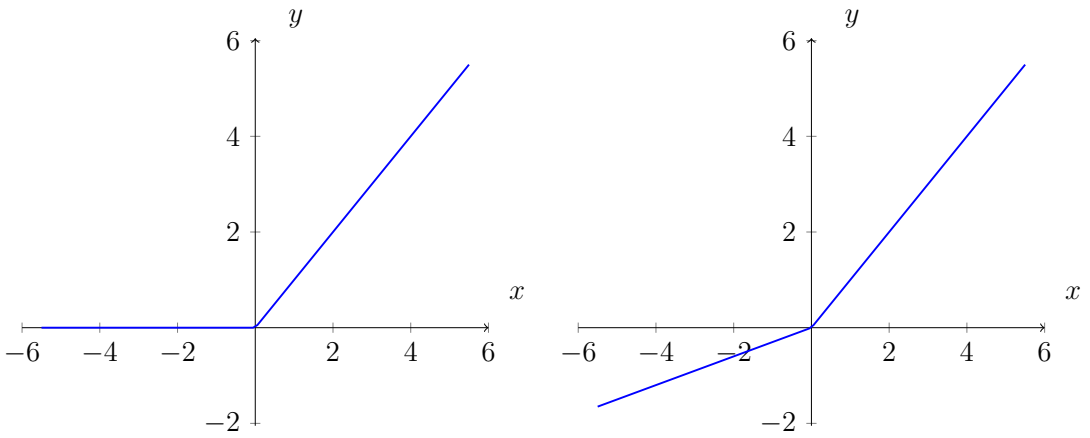


Figure 5. Left: ReLU activation function. Right: LReLU activation function.

For non-positive values, the function and its derivative is zero leading to shutting off these neurons which means such neurons will not be able to learn. This can be good as the model will not overfit the training data but can hinder model learning in some cases especially if the number of such neurons is huge. Leaky ReLU (LRelu) is a variant or a modification of ReLU. It introduces some small positive slope α for non-positive values - hence the name "Leaky". Usually α would be less than one. LReLU activation function and its derivative are on the form:

$$f(x) = \begin{cases} \alpha x & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases} \qquad f'(x) = \begin{cases} \alpha & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases} \tag{11}$$

All neurons are able to learn not like the case for ReLU activation function. The proposed solution discussed in Chapter 5 uses Leaky Rectified Linear Unit (LReLU) which is a variant of ReLU. Figure 5 illustrates both functions side by side.

3.1.4. Loss Function

We can think of the learning problem as an optimization problem where the goal is to minimize some objective function or loss function. This function quantifies the error or the loss between the original ground truth data and the prediction made by the network. L1 norm or L2 norm are examples for representing the loss functions. The variables we need to optimize upon are the weights of the neurons. The goal is to get the optimum weights that minimize the loss function and get the global minimum or actually the nearest local minimum if the function has more than one minimum.

3.1.5. Optimizer

Optimizer is used to optimize the defined loss function. Gradient Descent (GD) is well-known in the literature. It takes the steepest descent towards the nearest local minimum. Other variants of GD include Stochastic Gradient Descent (SGD) which updates the weights using a single training sample unlike GD that uses the whole

training data. The SGD is faster but convergence to the minimum is not guaranteed. The weights are updated as can be seen in the following equation:

$$w_i = w_i - \eta(\frac{\partial J(\boldsymbol{W})}{\partial w_i}), \tag{12}$$

where w_i is the weight updated, η is the learning rate, $\boldsymbol{W}$ is the weights and J is the objective or loss function. The learning rate η is a hyperparameter that controls the change in weight updates.

During network training, the forward pass computes the output of the network by propagating from the input until reaching the output. Weights are updated during the backward pass while back-propagating the gradients from the output back to the input.

Other optimizers came to exist as the neural networks became more complex and deeper. Adaptive Moment (Adam) [17] optimizer is now widely used by the community as it updates weights in an adaptive manner using first and second moment or momentum. First moment is the mean of the data and second moment is the uncentered variance. Adam uses exponentially moving averages to estimate these moments. In addition to the learning rate η hyperparameter, Adam uses β_1 and β_2 as additional hyperparameters for the exponential decay rate for the first and second moment estimates, respectively. Recommended values suggested by the authors for β_1 and β_2 are $0{,}9$ and $0{,}999$, respectively. Adam optimizer is used to train the model proposed in Chapter 5.

Figure 6 shows the steps taken by the optimizer to reach the nearest local minimum of the loss function J. For visualization purposes, only two weights are considered in the figure.

The next section discusses Convolutional Neural Network (CNN) which was introduced to deal with images as input.

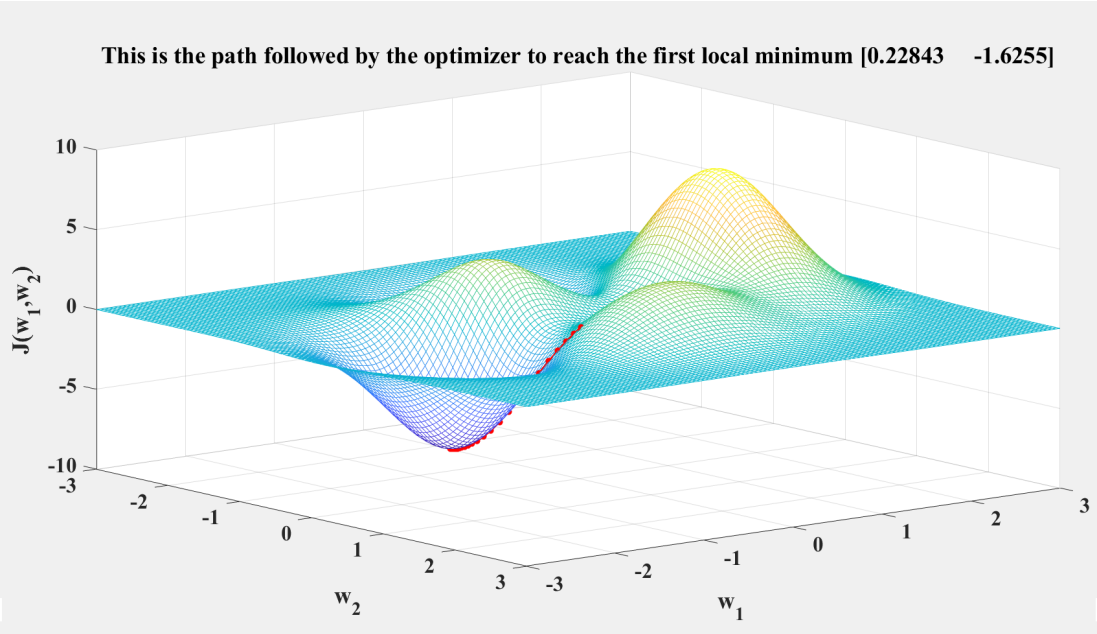


Figure 6. Optimization of a loss function J.

3.2. CNN and U-Net

The following subsections discuss Convolutional Neural Network (CNN) and U-Net which is an encoder decoder network. Both had proved to be successful in many applications.

3.2.1. CNN

Convolutional Neural Network (CNN) was first introduced by LeCun [18] in 1989. It did not receive much attention at that time. However, about a decade ago, it had much more attention and proved to be very useful and successful in solving many problems. The structure of a CNN makes it suitable to deal with images or even higher dimensional data. At the heart of CNN is the convolution operation hence, the name "Convolutional Neural Network". 2D convolution with an input image can be written on the form:

$$G[m,n] = (f \star k)[m,n] = \sum_j \sum_k k[j,k] f[m-j, n-k], \tag{13}$$

where G is the output activation map with indexes m and n for the height and width of the activation map, respectively. f is the input, k is the kernel. j and k are the indexes for kernel height and width, respectively.

Convolution between input and kernel is actually the same operation as was the case for perceptron discussed in Subsection 3.1.1, multiplying the input by weights of the kernel element-wise and then summing.

Converting an image to a 1D vector and then feeding it as an input for an ANN will make us lose spatial information which should be useful to be exploited. That is why CNN uses the image as input to make use of spatial information. This also applies to higher dimensional data such as 3D images, videos or even 3D videos. However, increasing the dimensionality of input and layers needs more computational power. Also, inspired by the Visual Cortex, each neuron in CNN would have a receptive field to receive information from. That means the neuron gets information from a specific region in the input. Another useful aspect of CNN is the use of shared kernels. The kernel is a 2D (or of higher dimensions depending on the input) matrix which is the weights learned by the network just like the perceptron. Sharing kernels between neurons in the same layer is useful as the number of learnable parameters is greatly reduced. Also, this ensures location invariance which means capturing same features irrespective of the location in the input layer or previous convolutional layer. Figure 7 illustrates the output produced by convolving the input with a 3×3 kernel.

The neurons in a convolutional layer or activation map will also have activation function as was the case for perceptron. Pooling layers are used for feature size reduction and to extract prominent features from the previous layer. Typically, Max Pooling or Average Pooling is used. As the name suggests, Max Pooling returns maximum value covered by the kernel and Average Pooling returns the average of all values within the window or kernel. After a number of convolutional layers, the last convolutional layer is flattened and then followed by Fully Connected (FC) layers

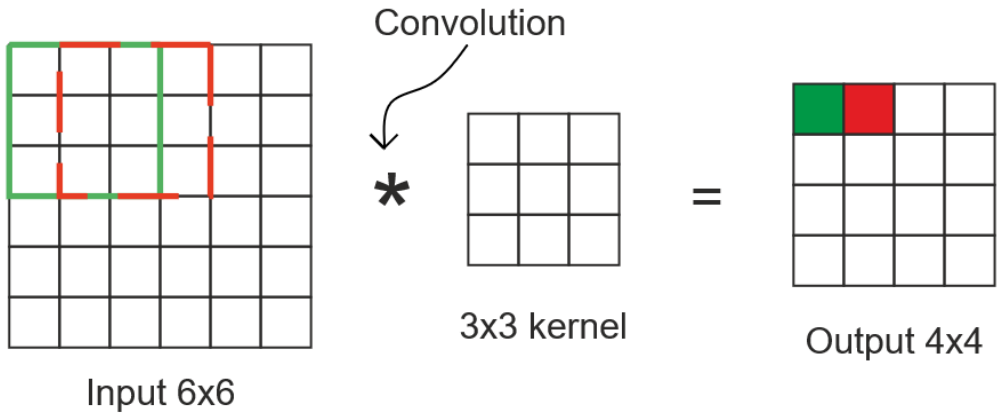


Figure 7. Convolution between input and filter. Note that each neuron in the output has a corresponding receptive field in the input. For example, neurons represented by green and red have green and red receptive field, respectively. The kernel is shared among all neurons.

just like a typical ANN to produce the output of the network. The output is typically categorical in this case and the network predicts the class of the input for classification problems. A typical CNN can be seen in Figure 8. The first layers in a CNN typically capture low level features such as edges. Deeper layers or activation maps capture higher level features.

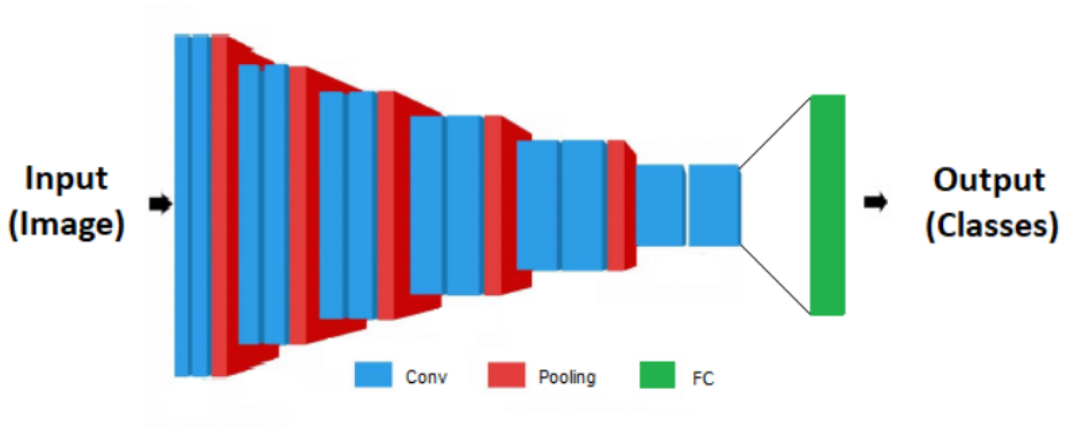


Figure 8. A typical CNN. The input is an image. Convolutional layers are followed by pooling layers. The last convolutional layer is flattened and then followed by a fully connected layer. The final output is the class which the image belongs to.

3.2.2. U-Net

The U-Net was first introduced in [19] for biomedical image segmentation. The U-Net is similar to CNN but the output is an image instead of the class which the image belongs to. This is extremely useful in applications where the goal is to obtain the class of each pixel in the image (semantic segmentation or instance segmentation) rather than determining only one class for the whole image. To produce an image as an output, the features obtained during the encoding phase are decoded or upsampled. Figure 9 shows how a typical U-Net looks like. The Skip connections are used to copy the features from the encoder to the decoder in order to preserve locality.

As illustrated before, feature encoding causes size reduction of feature maps. To get back to the original image resolution, upsampling is needed. Upsampling can be achieved using interpolation such as bi-linear interpolation, bi-cubic interpolation or nearest neighbor interpolation. However, to use learnable parameters instead of a predetermined method for upsampling, we can use transposed convolution or deconvolution. It is achieved by using a transposed convolution matrix. A convolution matrix is basically a matrix representation for the convolution operation. U-net has proved to be very useful especially in semantic segmentation or instance segmentation. The proposed solution is based on the U-net architecture.

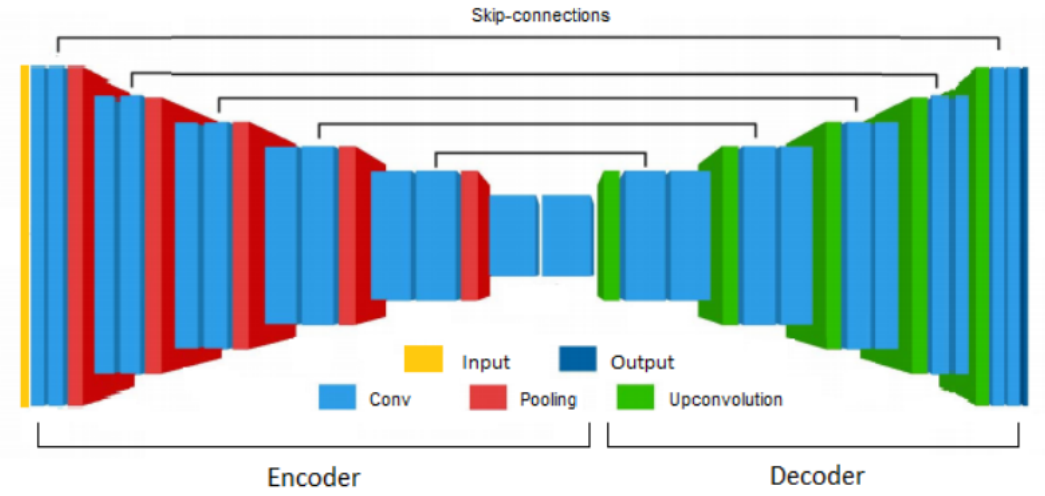


Figure 9. A typical U-Net or an encoder-decoder network. The input is an image and the output is an image.

4. DEPTH ESTIMATION USING DEEP LEARNING

In this chapter, we present some Deep Learning (DL) solutions to the depth estimation problem. The first section illustrates some approaches that were used to estimate depth from monocular images and similar problems. The second section discusses some approaches that were used to estimate the depth of non-rigid objects.

4.1. Estimating Depth Using Deep Learning

Estimating depth from single monocular images is an important problem that received the attention of many researchers. It has many applications in real life especially related to Augmented Reality (AR). Some research has also been devoted to estimating depth from stereo images or multiple images of the same scene in addition to estimating depth from videos. This is extremely useful for applications such as autonomous cars. Time constraint is also very crucial for such applications. Hence, the need for training networks that can predict output quickly within a specific time window. This section introduces some of the proposed solutions found in the literature to solve this problem using DL.

The authors in [20] have taken an interesting approach for estimating depth and motion from monocular stereo images. To estimate the motion and structure, they used optical flow which is the apparent motion between two consecutive frames. That is why they used stereo images. Initially, they used an encoder-decoder network or a U-net to estimate the optical flow between two stereo images and a confidence map using only the image pair. Then using these information alongside the image pair, they trained another encoder-decoder network to predict the depth map and the normal map for the first image. They also used a fully connected branch that shares the same encoder to predict the egomotion (camera motion) between the two frames. Figure 10 shows the encoder-decoder pair (Bootstrap) network that they used in the Depth and Motion Network (DeMoN).

The authors then followed an iterative approach to obtain finer depth maps. They used an iterative network that has three pairs similar to the one used in Bootstrap network but with more inputs. The estimates produced by the Bootstrap network are used as additional inputs to get finer details. In particular, they used the depth and normal maps produced by the Bootstrap network to compute the optical flow between the image pair and a warped second image in the image pair. They fed all this

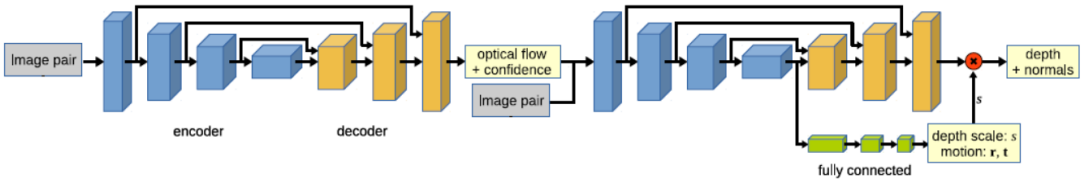


Figure 10. The Bootstrap network used in DeMoN architecture.

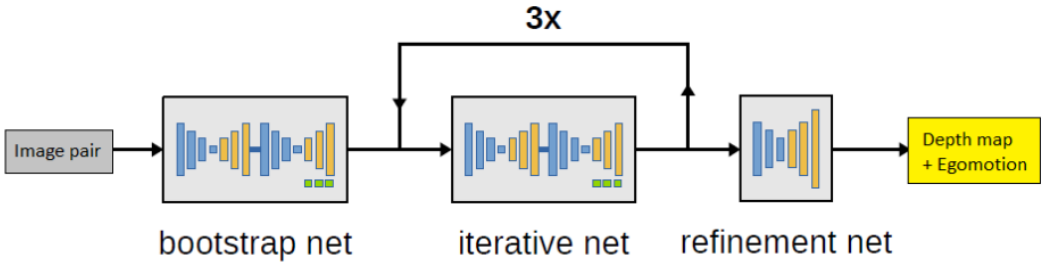


Figure 11. The full DeMoN architecture.

information in addition to the image pair to the first encoder-decoder in the iterative network. For the second encoder-decoder, they used a depth map computed using the estimated optical flow produced by the first encoder-decoder network and the motion parameters estimated from the Bootstrap network. All this information is then fed to the second encoder-decoder in addition to the ground truth image pair, a warped second image, optical flow and confidence map estimated by the first encoder-decoder. Finally, a refinement network which is an encoder-decoder network is used to get higher resolution images. The schematic representation of the full network architecture can be seen in Figure 11. We must note that this approach of driving the optical flow from estimated depth and normal map and driving the depth map using the optical flow and camera motion is only valid for rigid objects. Hence, the authors assumed that objects detected in the scene are rigid objects.

In [21], the authors used a Fully Connected (FC) Convolutional Neural Network (CNN) to predict a coarse global depth map for the scene followed by another FC CNN to refine the predicted depth map. What is interesting is that they used a scale-invariant loss function. So, objects at different scales will have the same loss or penalty. The scale-invariant loss function that the authors used is:

$$L(z, z^*) = \frac{1}{n}\sum_i (\log z_i - \log z_i^* + \frac{1}{n}\sum_j (\log z_j^* - \log z_j))^2 \tag{14}$$

where z_i is the predicted depth value for pixel i and z_i^* is the ground truth depth value for that pixel.

The invariance is achieved by using the inner sum by subtracting the mean in the logarithmic space from the corresponding depth map. Any scalar multiplication in linear space will be transformed as an addition operation in logarithmic space. Subtracting the mean in the logarithmic space from the depth values will cancel out the scalar multiplier.

Extending their work in another paper [22], the authors trained a three-scale FC CNN instead of two-scales only. They added another term to the loss function to ensure that predicted shapes have similar local structures as the ground truth. This can be done by comparing gradients for predicted and ground truth depth maps.

The loss function used in this work is on the form:

$$L(z, z^*) = \frac{1}{n}\sum_i (\log z_i - \log z_i^* + \frac{1}{n}\sum_j (\log z_j^* - \log z_j))^2 + \frac{1}{n}\sum_i [(\nabla_x d_i)^2 + (\nabla_y d_i)^2] \tag{15}$$

where $d_i = \log z_i - \log z_i^*$, $\nabla_x d_i$ and $\nabla_y d_i$ are the horizontal and the vertical gradient of the difference in logarithmic space, respectively.

This inspired us to use invariant loss functions to train our model which will be illustrated in the next chapter.

In [23], the authors considered the problem of determining the normal maps from monocular single images. The normal map represents the structure of the scene without considering the global depth of the scene. They used a global network to predict a coarse normal map representation and a local network to predict more finer details. The patches extracted based on a sliding-window fashion. The outputs from both networks are then fused to generate the final normal map output.

In [24], the authors used pre-trained networks to get an initial depth map and an initial normal map for a single monocular image. Then, they trained a network to produce a refined depth map using the initial depth and normal maps. Another network is also trained to produce a refined normal map given the initial depth and normal maps. They claimed that following this procedure gave finer and better results.

Authors in [25] tried to enhance depth estimation by embedding focal length information in the learning process. A branch is added for this purpose to an encoder-decoder network. They used a pre-trained VGG model [26].

Many attempts have been done to estimate depth from videos using unsupervised learning by using some consistency metrics as a loss function during training. The network architectures used are generally some variations of an encoder-decoder network or a U-net as in [27].

Other DL approaches for solving the depth estimation problem can be found in [28, 29, 30, 14, 31, 32] using supervised or unsupervised learning.

These encoder-decoder network architectures can be used to solve other problems such as obtaining the optical flow between two images [33, 34], semantic segmentation [35] ,medical image segmentation [36, 37, 38, 39], volumetric segmentation [40, 41] or image deblurring [42, 43, 44].

4.2. Deep Learning with Non Rigid Objects

Most of the research focused on constructing the shape of rigid objects. In fact, many solutions were designed specifically to exploit the geometric features of rigid objects. Constructing the shape of non-rigid objects did not receive much of attention as it is more challenging. This section introduces some of the solutions found in the literature to reconstruct the shape of non-rigid or deforming objects.

The authors in [45] used a 2D U-net to reconstruct the 3D shape of deforming objects. They used identity connections or shortcut connections to overcome the gradient vanishing problem as in residual networks (resnet) [46]. They used a loss

function that has three components or three types of loss functions. Let $\boldsymbol{S} = \{\boldsymbol{S_f}\}, f \in \{1, \ldots, F\}$ denote predicted 3D points, and $\boldsymbol{S^*} = \{\boldsymbol{S_f^*}\}$ is the ground truth. The 3D error is represented by:

$$L_{3D}(\boldsymbol{S}, \boldsymbol{S^*}) = \frac{1}{F} \sum_{f=1}^{F} \|\boldsymbol{S_f^*} - \boldsymbol{S_f}\|_F^2, \tag{16}$$

where $\| \cdot \|_F$ denotes the Frobenius norm.

To force neighboring points to be closely located. An isometry prior is used as:

$$L_{iso}(\boldsymbol{S}) = \frac{1}{F} \sum_{f=1}^{F} \|\hat{\boldsymbol{S}}_{\boldsymbol{f}} - \boldsymbol{S_f}\|_F, \tag{17}$$

where $\hat{\boldsymbol{S}}_{\boldsymbol{f}}$ is Gaussian smoothed version of $\boldsymbol{S_f}$. This forces the estimated surface to be smooth.

Finally, a contour loss $L_{cont,}(\boldsymbol{S}, \boldsymbol{S^*})$ is added by comparing the 2D projection of the predicted point cloud and ground truth point cloud on a 2D plane. If the camera parameters are known, prospective projection is used otherwise orthographic projection is used.

The total loss is represented by:

$$L(\boldsymbol{S}, \boldsymbol{S^*}) = L_{3D}(\boldsymbol{S}, \boldsymbol{S^*}) + L_{iso}(\boldsymbol{S}) + L_{cont,}(\boldsymbol{S}, \boldsymbol{S^*}), \tag{18}$$

In [47], authors train a U-net to predict depth map of a deforming object from a single image. The network learns the deformations from shading as a cue which is similar to Shape-from-Shading (SfS).

The authors in [48] took a similar approach to the previous one. However, they extracted patches from the scene. To combine or stitch the depth maps from different patches, they translated reconstructed patches along the viewing direction to have the same reconstruction in the overlapping areas. This is an optimization problem which is solved in the least square sense. They used the Levenberg Marquardt algorithm for optimizing this problem.

In [49], the authors used a Deep Convolutional Neural Network (DCNN) to estimate the 3D shape of garments. The input to the network is a masked image which contains the garment and the output is the corresponding 3D mesh that represent the garment. The network architecture is based on the SqueezeNet [50] architecture which achieved the performance of AlexNet [51] using fewer parameters (50x fewer parameters). Hence, a faster training can be achieved.

To construct a 3D mesh of a non-rigid object [52], the authors devised a network that has three branches. The first two branches are used in parallel to get the 2D trajectories of the vertices in the 3D mesh and the depth estimation for these points. The 2D trajectories are predicted using belief maps [53]. The third branch is then used to fuse the 2D trajectories and depth values to obtain the 3D mesh of the shape.

5. PROPOSED SOLUTION USING 3D U-NET

In this chapter, we present the proposed solution to the problem of estimating the depth maps for a video of a deforming object. We assume that the scene is captured by an orthographic static camera. So, we do not need to solve for the camera motion as is the case for NRSFM methods. Motivated by the success obtained by using the U-net architecture to solve many similar problems as was shown in the last chapter, we estimate the depth maps for small local video patches using an encoder-decoder network. Since it is not efficient to obtain a large-scale real data for training the network, we generate a synthetic data for training, validation and testing purposes. The motion of the object surface in these video patches is modeled by a parametric model using small number of parameters. The small local depth maps for the video patches are then combined and stitched together (in space and time domain) to construct the global video depth map. It is assumed that the global shape deformations can be modeled by combining the deformations in local patches which are realized by the parametric model we used to generate the data.

The first section describes the database that was generated and used to train the model. The second section illustrates the network architecture. The third section presents the various invariant loss functions used to train the network. Section four illustrates the metrics used for evaluation. Section five shows how to combine video patches to globally reconstruct the observed object. This requires some post-processing in space and time domains. Section six sheds the light on some key implementation details.

5.1. Database Used in Training the Network

Video sequences are abundant as they only require simple cameras to capture deforming objects. However, obtaining the ground truth depth maps is not that easy. That requires using more sophisticated devices like Microsoft Kinect to acquire depth information. Collecting data using Kinect devices and using it for network supervision is not practical for the following reasons:

1. depth information from these devices is not accurate,
2. depth information is sometimes missing,
3. laborious and time consuming,
4. difficult to produce various deformation variations that an object might have under different lighting conditions, and
5. very hard to produce enough number of samples for network training, validation and testing.

For the previous reasons, it is quite reasonable that using artificially generated database is the better option to produce samples for network supervision. Also, the local shape of a deforming object can be approximated by a simple parametric model using a small number of parameters. The advantages of using artificially generated database are exactly the opposite of the disadvantage of using real data captured using Kinect devices. However, deformations realized by artificially generated data needs

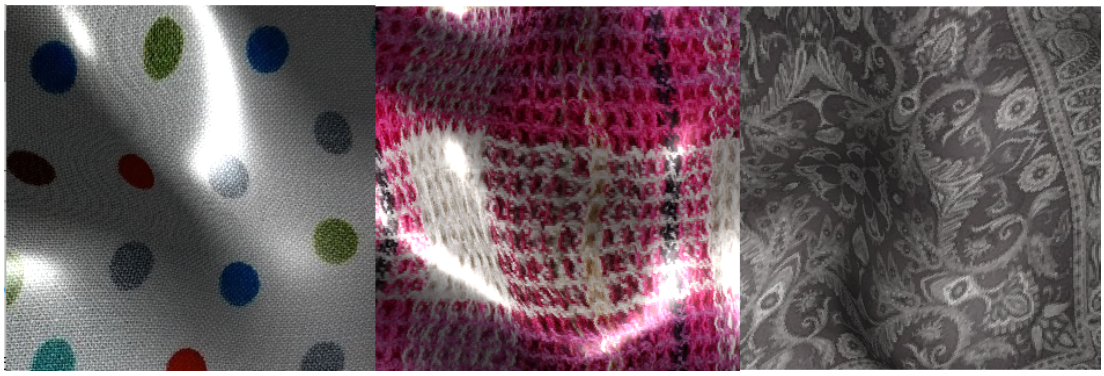


Figure 12. Some frames of artificially generated samples.

to be close to those found in real data. This ensures the applicability of using this model in real life scenarios for many relevant applications. More details about the implementation of database generation process are provided in Subsection 5.6.1. It is also worth mentioning that the output of some NRSFM algorithm can be used as a supervision for the network training but this approach is not useful as the model accuracy would be dependant on the accuracy of these algorithms.

The input to the network is a video of 16 frames. Each frame or image is 64×64 pixels. The output of the network is the corresponding depth map for each frame. However, the output depth map has a resolution of 32×32 pixels. Using lower resolution is more computationally efficient especially when the number of data samples used for training is huge. Less resources such as storage, RAM and GPU are needed in this case. For example, storage space is roughly (depends on disk parameters such as block size and sector size) reduced by a factor of 4 when using 32×32 resolution instead of the full resolution of 64×64. A total of 150k artificial samples were generated by a Matlab script. The videos generated are in grey scale as color information seemed not to be that important to learn motion. Also, storage space is roughly reduced by a factor of 3 by using grey images instead of color images. For the data generated, 80% was used for training, 10% for validation and 10% for testing. Figure 12 shows some artificially generated animations that were used to train the network.

5.2. Network Architecture

The structure of the network is pretty much similar to a 3D U-net (overview of regular 2D U-net was discussed in Section 3.2) with some key modification. The 3D U-net is similar to the regular U-net [54] or 2D U-net but uses 3D operations instead of 2D operations. That means that kernels or filters for convolution and all layers are 3D. The 3D U-net was first used in [40] to segment volumetric images for medical applications.

The proposed network structure can be seen in Figure 13. Bottlenecks in the network architecture are avoided by doubling the number of channels before the max pooling layers as suggested in [55]. Experiments also have shown that this modification improves model performance. For all model convolutions, $3 \times 3 \times 3$ kernels are used. For max pooling layers, $2 \times 2 \times 2$ kernels are used with a stride of 2 which

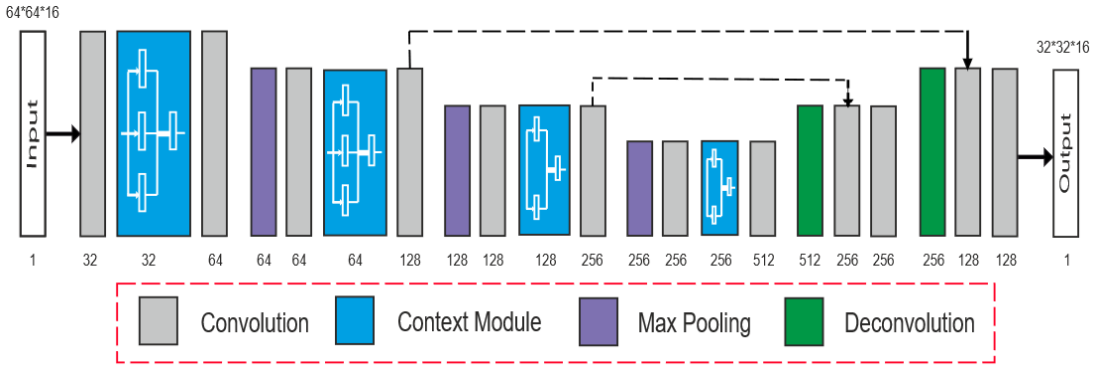


Figure 13. Network architecture. The input is a grey-scale video of a deforming object and the output is the corresponding depth map sequence. Below each layer is the number of channels for that layer. Dashed lines represent feature maps forwarding.

effectively halves all dimensions of a layer. Batch Normalization (BN) is used to make the model converges faster. The number of learn-able parameters that the network has is $37,719,073$. LReLU activation function is used in model training. Using LReLU gave better performance than using ReLU. It was also more reasonable to use LReLU as the final output depth map values are real in the range [-1,+1] as depth maps are normalized and centered around the origin. During training, the value of α (see Equation (11)) was set to be $0{,}3$ (the default value used by Keras). In the contracting path, a context module (more details in Subsection 5.2.1) is introduced. In the expanding path, deconvolution or transposed convolution is used rather than simply upsampling which improved performance. Context modules and network parameters used are discussed in the following subsections.

It is also worth mentioning that using 2D U-net did not produce good results. Convolving along time dimension alongside spatial dimensions gave better results than simply using time dimension as channels or features. These results seem reasonable and plausible as features obtained along time dimension are important for understanding the motion.

5.2.1. Context Module

In order to enlarge the receptive field and get richer features, a context module was introduced in the contracting path [56]. It performs parallel dilated convolutions with different dilation rates. Dilated convolution covers larger receptive field with a smaller kernel by skipping some neurons. Figure 14 shows 2D dilated convolution using dilation rates of 1,2 and 3. If we use the standard convolution (Dilation rate = 1) to cover larger receptive field, larger kernels with larger number of parameters would be needed. For example, to cover a receptive volume of $7 \times 7 \times 7$, using a standard convolution would require a kernel of $7 \times 7 \times 7$ which has 343 parameters. Using a dilated convolution with dilation rate of 3 would require a kernel of $3 \times 3 \times 3$ which only has 27 parameters. This is a huge reduction by a factor of more than 12 for each input and output channel.

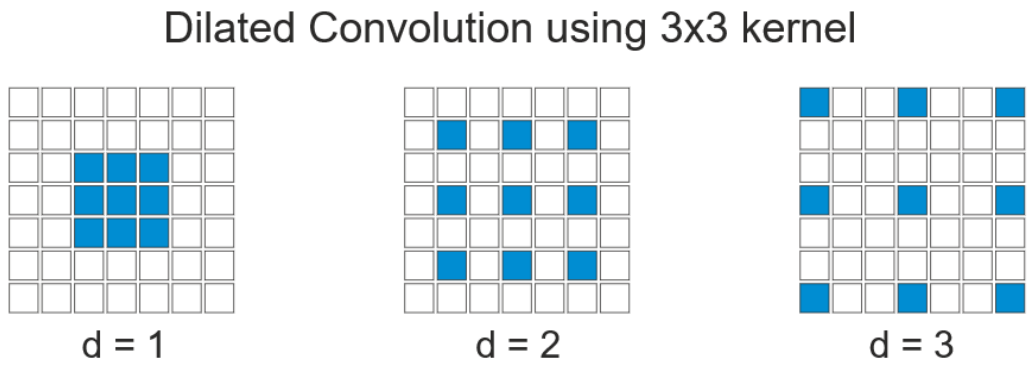


Figure 14. Dilated convolution with different dilation rates. Left: dilation rate = 1 also known as standard convolution. Center: dilation rate = 2 and covers 5×5 area. Right: dilation rate = 3 and covers 7×7 area.

The feature maps generated by these parallel dilated convolutions are concatenated. The output of this module is the convolution of these concatenated feature maps. For the first two stages in the network, 3 parallel dilated convolutions are used with dilation rates (1,2,3). For the following deeper stages, 2 parallel dilated convolutions are used with dilation rates (1,2) as the feature maps size gets smaller. Figure 15 illustrates the context modules used in the network architecture. The model performance was greatly enhanced after adopting such modules.

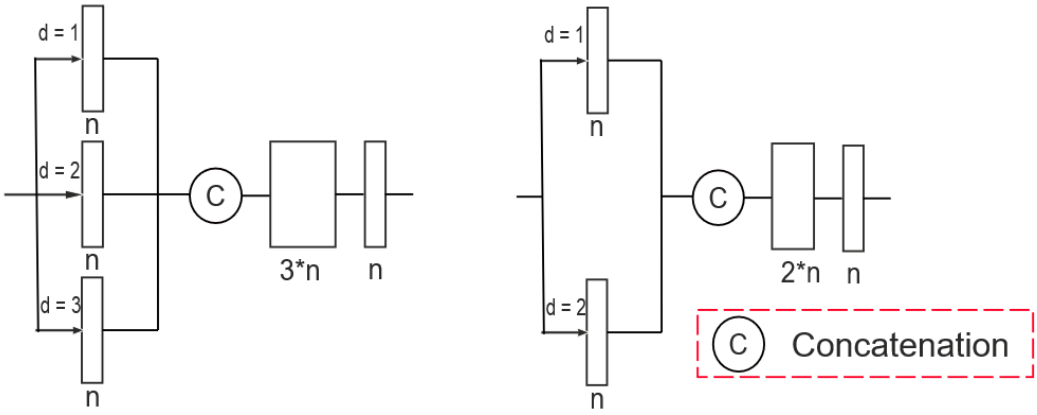


Figure 15. Left: context module with 3 parallel dilated convolutions (dilation rates = 1,2,3). Right: context module with 2 parallel dilated convolutions (dilation rates = 1,2).

5.2.2. Network Parameters

In order to prevent overfitting, a validation set is used to achieve this purpose. The learning rate is halved if the validation loss does not improve for 3 consecutive epochs. The initial learning rate is set to $0{,}01$. The network stops training if the validation loss does not improve for 5 consecutive epochs. This is called early stopping. The batch size used for training is 16. The entire training dataset is normalized before training. Adam [17] is the optimizer used to train the model. The exponential decay rates for the first and second moment estimates are $0{,}9$ and $0{,}999$, respectively.

5.3. Loss Functions

This section describes the loss functions used to train the previous proposed network. As was discussed in Subsection 2.2.2, translation, scale and shear along z-axis ambiguities exist. Hence, we need loss functions that are invariant to such ambiguities. Using loss functions that are not invariant to these ambiguities causes network confusion. Hence, the network will not be able to learn properly. That is because infinitely many structures in world coordinates map to the same projected image in the image plane. That means performing translation, scale or shear along z-axis operations causes the structure in world coordinate to change. However, the projected image would remain the same as lighting and shading conditions can be adjusted accordingly to produce the same projected image. We can represent the depth map as a discrete function in x and y coordinates as:

$$z = f(x, y), \tag{19}$$

To introduce translation component to the shape along z direction, a scalar is simply added to this function which is a zero order term as:

$$z_t = f(x, y) + t, \tag{20}$$

To scale the shape along z direction, a scalar is multiplied by this function as:

$$z_{sc} = \alpha f(x, y), \tag{21}$$

Finally, to make the shape shear along z direction, first order terms in x and/or y can be added to the function as:

$$z_{sh} = f(x, y) + s_1 x + s_2 y, \tag{22}$$

where s_1 and s_2 are shear parameters which are simply a scale for x and y, respectively. That is the depth values are pushed along the z direction by s_1 and s_2 scales of x and y, respectively. A general representation which include the three operations is:

$$z_{all} = \alpha f(x, y) + s_1 x + s_2 y + t, \tag{23}$$

The next two subsections discuss differential and point cloud invariant loss functions, respectively. A differential invariant is derived using the derivatives of the object surface whereas a point cloud invariant is derived using the 3D structure of the surface directly.

5.3.1. Differential Invariant Loss Functions

The loss functions introduced in this subsection are derived using the derivatives of the surface. The loss functions introduced here are invariant to some or all of the operations discussed before that cause ambiguities. Hence the name "differential invariant loss functions". All these differential invariants can be derived using the theory of moving frames [57].

Translation along z-axis Invariant

We need to use a loss function that produces the same quantity for a shape or a translated version along the z-axis of this shape. For orthographic projection, the shape or infinitely many translated versions of that shape would map to the same projected image in the image plane. Hence, we need a function that is invariant to such an operation and produces the same loss value for all these translated versions. In other words, we need a loss function that gives the same value when applied to Equation (19) and Equation (20).

To get rid of the zero order translation term, we can use the first derivative of the shape surface function as a loss function. The first derivative removes the zero order term i.e the first derivative of Equation (19) and Equation (20) gives exactly the same value. The translation invariant loss function is represented by:

$$L_{t\_inv}(z, z^*) = \frac{\sum_p [(\frac{\partial z}{\partial x} - \frac{\partial z^*}{\partial x})^2 + (\frac{\partial z}{\partial y} - \frac{\partial z^*}{\partial y})^2] M_p}{\sum_p M_p} \tag{24}$$

where z and z^* are the predicted and ground truth depth maps, respectively. M_p is a mask for pixels that have valid depth values i.e missing or invalid depth values are masked out. It is quite obvious that any translated version of the depth map would have the same loss or error value. The error or loss is calculated as the squared Euclidean distance between the first derivatives of the ground truth and predicted depth map.

Scale along z-axis Invariant

As the case for translation invariant, we need to use a loss function that produces the same quantity for a shape or a scaled version along the z-axis of this shape. For orthographic projection, the shape or infinitely many scaled along z-axis versions of that shape would map to the same projected image in the image plane. Hence, we need a function that is invariant to such an operation and computes the same loss value for

all these scaled versions. In other words, we need a loss function that gives the same value for Equation (19) and Equation (21).

We can simply achieve this by transforming the function to the logarithmic space. By switching to the logarithmic space, the scalar multiplier is transformed to an additive scalar term. This problem is now similar to the previous translation case. We can then compute the first derivative in that space to obtain a scale invariant measure. The scale invariant loss function is:

$$L_{s\_inv}(z, z^*) = \frac{\sum_p [(\frac{\partial \log z}{\partial x} - \frac{\partial \log z^*}{\partial x})^2 + (\frac{\partial \log z}{\partial y} - \frac{\partial \log z^*}{\partial y})^2] M_p}{\sum_p M_p} \tag{25}$$

It is quite obvious that any scaled version along z-axis of the depth map would have the same loss or error value.

Translation and Scale along z-axis Invariant

The first derivative eliminates the translation component. However, the scale multiplier remains. Normalizing the first derivative would eliminate the scale multiplier as it would appear in the denominator when normalizing and would be canceled out accordingly. The translation and scale invariant loss function can be represented by:

$$L_{t-sc\_inv}(z, z^*) = \frac{\sum_p [(\frac{\frac{\partial z}{\partial x}}{|\nabla z|} - \frac{\frac{\partial z^*}{\partial x}}{|\nabla z^*|})^2 + (\frac{\frac{\partial z}{\partial y}}{|\nabla z|} - \frac{\frac{\partial z^*}{\partial y}}{|\nabla z^*|})^2] M_p}{\sum_p M_p}, \tag{26}$$
$$where \qquad |\nabla z| = \sqrt{(\frac{\partial z}{\partial x})^2 + (\frac{\partial z}{\partial y})^2}, \qquad |\nabla z^*| = \sqrt{(\frac{\partial z^*}{\partial x})^2 + (\frac{\partial z^*}{\partial y})^2},$$

Again, any translated and/or scaled version along z-axis of the depth map would have the same loss or error value.

Translation, Scale and Shear along z-axis Invariant

Shear introduces first order terms. In order to remove these terms, second derivatives can be used. This also takes care of the translation component. To eliminate the scale factor, we can normalize the second derivative of the function. Hence, normalized hessian matrix is translation, scale and shear along z-axis invariant. The corresponding loss function to achieve this invariance is:

$$L_{all\_inv}(z, z^*) = \frac{\sum_p [(\frac{\frac{\partial^2 z}{\partial x^2}}{|\nabla^2 z|} - \frac{\frac{\partial^2 z^*}{\partial x^2}}{|\nabla^2 z^*|})^2 + w(\frac{\frac{\partial^2 z}{\partial x \partial y}}{|\nabla^2 z|} - \frac{\frac{\partial^2 z^*}{\partial x \partial y}}{|\nabla^2 z^*|})^2 + (\frac{\frac{\partial^2 z}{\partial y^2}}{|\nabla^2 z|} - \frac{\frac{\partial^2 z^*}{\partial y^2}}{|\nabla^2 z^*|})^2] M_p}{\sum_p M_p},$$
$$where \ |\nabla^2 z| = \sqrt{(\frac{\partial^2 z}{\partial x^2})^2 + 2(\frac{\partial^2 z}{\partial x \partial y})^2 + (\frac{\partial^2 z}{\partial y^2})^2}, \quad |\nabla^2 z^*| = \sqrt{(\frac{\partial^2 z^*}{\partial x^2})^2 + 2(\frac{\partial^2 z^*}{\partial x \partial y})^2 + (\frac{\partial^2 z^*}{\partial y^2})^2}, \tag{27}$$

Here, w is a weight for the mixed partial derivative term. Any version of the shape that is translated, scaled or sheared along z-axis would have the same loss value.

5.3.2. Point Cloud Invariant Loss Functions

We can obtain loss functions that are invariant to translation, scale and shear along z-axis using the point clouds as well as using differential calculus as was described in Subsection 5.3.1.

Translation Invariant

Subtracting the mean of the point cloud from each point in the point cloud will ensure that the origin is the center of gravity for the shape. Hence, any translated version would be translated to have the origin as center of gravity. The translation along z-axis invariant loss function is on the form:

$$L_{t\_inv}(z, z^*) = \frac{\sum_p [(z - (z^* - z_m^*))^2] M_p}{\sum_p M_p} \tag{28}$$

where z_m^* are the mean value for the ground truth depth map. Note that the network is trained to produce a shape centered at the origin. For any upcoming invariant loss function, the network is trained to produce the transformed shape. Again, these transformations are done to account for the ambiguities discussed before.

Scale along z-axis Invariant

To make the function scale along z-axis invariant, we can divide the depth map by its standard deviation. This means all depth maps would have a standard deviation of one. Hence, all depth maps would have the same shape size. The scale along z-axis invariant loss function can be represented by:

$$L_{sc\_inv}(z, z^*) = \frac{\sum_p [(z - \frac{z^*}{\sqrt{z_v^*}})^2] M_p}{\sum_p M_p} \tag{29}$$

where z_v^* is the variance for the ground truth depth map. Here, the network is trained to produce a scaled shape with a standard deviation of one as an output.

Translation and Scale along z-axis Invariant

We can combine the previous two steps to get a translation and scale along z-axis invariant loss function. That is subtracting the mean and dividing by the standard deviation. This is also known as z-score normalization. It is quite obvious that this indeed translation and scale along z-axis invariant as all translated and scaled versions would be centered at the origin and scaled to have the same shape size. The corresponding invariant loss function is on the form:

$$L_{t-sc\_inv}(z, z^*) = \frac{\sum_p [(z - \frac{z^* - z_m^*}{\sqrt{z_v^*}})^2] M_p}{\sum_p M_p} \tag{30}$$

The network is trained to output a z-score normalized shape.

Shear along z-axis Invariant

Given a point cloud P with n points as follows:

$$\boldsymbol{P} = \begin{bmatrix} x_1 & y_1 & z_1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & z_n \end{bmatrix} = \begin{bmatrix} \boldsymbol{X} & \boldsymbol{Y} & \boldsymbol{Z} \end{bmatrix} \tag{31}$$

We can then get the following matrices:

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{X^T X} & \boldsymbol{X^T Y} \\ \boldsymbol{X^T Y} & \boldsymbol{Y^T Y} \end{bmatrix}, \qquad \boldsymbol{B} = \begin{bmatrix} \boldsymbol{X^T Z} \\ \boldsymbol{Y^T Z} \end{bmatrix} \tag{32}$$

The shear parameters that will make the new depth map invariant to shear are:

$$\begin{bmatrix} s_1^* \\ s_2^* \end{bmatrix} = -\boldsymbol{A}^{-1}\boldsymbol{B} = -\frac{1}{|\boldsymbol{A}|}\begin{bmatrix} (\boldsymbol{Y^T Y})(\boldsymbol{X^T Z}) - (\boldsymbol{X^T Y})(\boldsymbol{Y^T Z}) \\ (\boldsymbol{X^T X})(\boldsymbol{Y^T Z}) - (\boldsymbol{X^T Y})(\boldsymbol{X^T Z}) \end{bmatrix} \tag{33}$$

where $|\boldsymbol{A}|$ is the determinant of $\boldsymbol{A}$. Hence, the depth map that is invariant to shear along z-axis for a depth map z is given by:

$$z_{sh\_inv} = z + s_1^* x + s_2^* y \tag{34}$$

To show that this quantity is indeed shear invariant, We need to show that this quantity is the same for Equation (19) and Equation (22) i.e the quantity is the same for $\boldsymbol{Z}$ and $\boldsymbol{Z} + s_1\boldsymbol{X} + s_2\boldsymbol{Y}$. So, for $\boldsymbol{Z} + s_1\boldsymbol{X} + s_2\boldsymbol{Y}$, to get the new shear parameters, we substitute $\boldsymbol{Z} + s_1\boldsymbol{X} + s_2\boldsymbol{Y}$ for $\boldsymbol{Z}$ in Equation (33):

$$\begin{bmatrix} \hat{s}_1 \\ \hat{s}_2 \end{bmatrix} = \begin{bmatrix} s_1^* \\ s_2^* \end{bmatrix} - \frac{1}{|\boldsymbol{A}|}\begin{bmatrix} s_1[(\boldsymbol{Y^T Y})(\boldsymbol{X^T X})] + s_2[(\boldsymbol{Y^T Y})(\boldsymbol{X^T Y})] \\ s_1[(\boldsymbol{X^T X})(\boldsymbol{Y^T X})] + s_2[(\boldsymbol{X^T X})(\boldsymbol{Y^T Y})] \\ \qquad -s_1[(\boldsymbol{X^T Y})(\boldsymbol{Y^T X})] - s_2[(\boldsymbol{X^T Y})(\boldsymbol{Y^T Y})] \\ \qquad -s_1[(\boldsymbol{X^T Y})(\boldsymbol{X^T X})] - s_2[(\boldsymbol{X^T Y})(\boldsymbol{X^T Y})] \end{bmatrix} \tag{35}$$

After simplification and taking into account that $|\boldsymbol{A}| = (\boldsymbol{X^T X})(\boldsymbol{Y^T Y}) - (\boldsymbol{X^T Y})^2$, we get:

$$\begin{bmatrix} \hat{s}_1 \\ \hat{s}_2 \end{bmatrix} = \begin{bmatrix} s_1^* - s_1 \\ s_2^* - s_2 \end{bmatrix} \tag{36}$$

Substituting $\hat{s}_1$, $\hat{s}_2$ and $z + s_1 x + s_2 y$ for s_1^*, s_2^* and z, respectively in Equation (34), we get:

$$\hat{z} = (s_1^* - s_1)x + (s_2^* - s_2)y + z + s_1 x + s_2 y = z + s_1^* x + s_2^* y = z_{sh\_inv} \tag{37}$$

Hence, the quantity is the same for any sheared along the z direction version of the original depth map. So, this quantity is invariant to shear along z-axis. From Equation

(36), we can see that any shearing along z-direction will be accounted for by shearing in the opposite direction. We can now obtain the loss function that is invariant to shear along z-axis as:

$$L_{sh\_inv}(z, z^*) = \frac{\sum_p [(z - (z^* + s_1^* x^* + s_2^* y^*))^2] M_p}{\sum_p M_p} \tag{38}$$

The network is trained to output a shape with zero shear parameters i.e a shape with no shear.

Translation, Scale and Shear along z-axis Invariant

To get a loss function that is invariant to all previous operations. We can transform the shape as follows:

1. translate the ground truth shape to the origin that is $z_t^* = z^* - z_m^*$, $x_t^* = x^* - x_m^*$, $y_t^* = y^* - y_m^*$ where z_m^*,x_m^* and y_m^* are the means for the 3 axes,
2. shear the centered shape to have zero shear parameters by using Equation (34) that is $z_{sh}^* = z_t^* + s_1^* x_t^* + s_2^* y_t^*$,
3. scale the transformed shape from previous step to have a standard deviation of one that is $z_{transformed}^* = \frac{z_{sh}^*}{\sqrt{z_{shv}^*}}$,

The invariant loss function is then simply the squared Euclidean distance between the ground truth transformed shape and the predicted one. The invariant loss function is on the form:

$$L_{all\_inv}(z, z^*) = \frac{\sum_p [(z - z_{transformed}^*)^2] M_p}{\sum_p M_p} \tag{39}$$

Here, the network is also trained to output a transformed shape with zero mean, zero shear parameters and a shape size (standard deviation in this case) of one.

5.4. Metrics Used for Evaluation

The metric used for evaluation is the mean Euclidean distance between the spatially normalized (have standard deviation of one) ground truth depth maps and the normalized predicted depth maps after alignment [58]. The spatially normalized mean absolute error (SNMAE) for one frame can be represented by:

$$SNMAE = \frac{\sum_p |z_{aligned} - z^*| M_p}{\sqrt{z_v^*} \sum_p M_p} \tag{40}$$

where $z_{aligned}$ is the aligned predicted depth map, z^* is the ground truth depth map, M_p is a mask for valid ground truth depth values and z_v^* is the variance of the ground truth depth map.

The alignment is needed due to the global inherent ambiguities as in [59, 60]. These ambiguities have been accounted for during model training using appropriate loss functions that are invariant to such ambiguities as was discussed in Section 5.3. Hence, appropriate alignment is needed in order to account for these ambiguities for evaluation purposes also.

5.4.1. Translation, Scale and Shear along Z-Axis Alignment

The alignment is done as follows:

1. the two shapes are translated so that the origin is the center of gravity for both shapes,
2. both shapes are scaled to have a Root Mean Squared Distance (RMSD) of 1, and
3. shearing along z-axis to align both shapes.

For step 2, RMSD can be used as a shape size metric. In fact, any positive shape size metric S(x) that fulfills the property

$$S(\alpha x) = \alpha S(x), \tag{41}$$

can be used. The variance or standard deviation can be used as a metric for shape size also since both are positive and satisfy the property.

The procedure used to make the point cloud loss function invariant to shear along z-axis as discussed in Subsection 5.3.2 (shear along z-axis invariant loss function) can be used again. After determining shear parameters as in Equation (33) for both the predicted and ground truth depth maps, they can be used to account for shear effect as:

$$Z + s_1 X^* + s_2 Y^* = \hat{Z} + s_1^* X^* + s_2^* Y^*, \tag{42}$$

where X^*, Y^* are the X and Y vectors of the ground truth point cloud. s_1, s_2 are the shear parameters of the predicted depth map. s_1^*, s_2^* are the shear parameters of the ground truth depth map. Z is the predicted depth map and $\hat{Z}$ is the aligned predicted depth map.

The previous equation can be rearranged as:

$$\hat{Z} = Z + (s_1 - s_1^*)X^* + (s_2 - s_2^*)Y^*, \tag{43}$$

to obtain the aligned predicted depth map. The shear operation causes the shape size of the aligned predicted depth map to change slightly so it is scaled again to have shape size of 1.

Figure 16 shows an example of two structures that need to be aligned and the results obtained after performing each step in the aligning procedure.

5.5. Large Scene Reconstruction

To reconstruct a large scene, we need to take into consideration both spatial and temporal dimensions as we are dealing with sequence of images or a video. Spatial

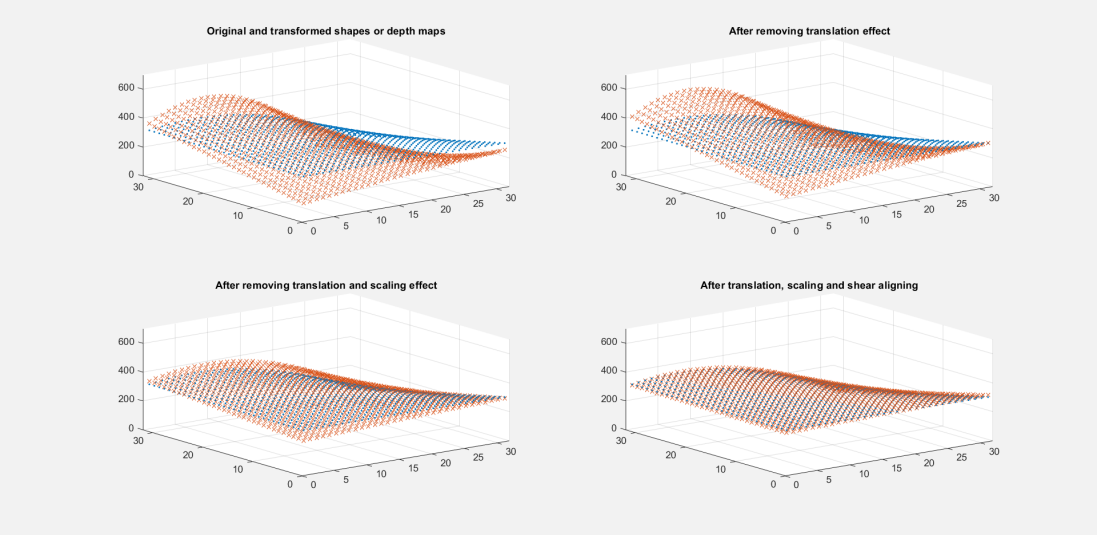


Figure 16. Top left: two shapes before aligning. The orange shape needs to be aligned to the blue original one. Top right: after compensating for the translation along z-axis effect. Bottom left: after compensating for the translation and scale along z-axis effects. Bottom right: after compensating for the translation, scale and shear along z-axis effects.

reconstruction of a large scene using a multiscale algorithm is discussed in Subsection 5.5.1 whereas temporal reconstruction of a large scene is introduced in Subsection 5.5.2.

5.5.1. Spatial Reconstruction of a Large Scene Using Multiscale Algorithm

For spatial dimensions, the input of the network is 64x64 as was illustrated previously. So, we need to deal with input of different sizes especially of larger sizes to reconstruct a large scene. Put in other words, we need to scale this solution. To reconstruct a scene with different spatial dimensions, the following can be done for an input of size h*w:

1. Upsample or downsample the original scene to 2^mx2^n where m and n are the powers of the closest power of two for h and w, respectively. Note that m and n must be greater than or equal to 6 as the network input is 64x64. Let N to be the minimum of m and n where $N \geq 6$,
2. We have $N - 5$ levels to be used for construction where level 0 is the finest resolution and level $N - 6$ is the coarsest one. Each level is obtained by downsampling or halving the resolution of the previous level. This is pretty much the same as Laplacian or Gaussian pyramids which are image pyramids [61, 62]. Figure 17 shows the image pyramid for an image,
3. Extract 64x64 patches from each level with 50% overlapping between neighboring patches,
4. Run the network to get the output for each patch which is the predicted depth map for that patch,

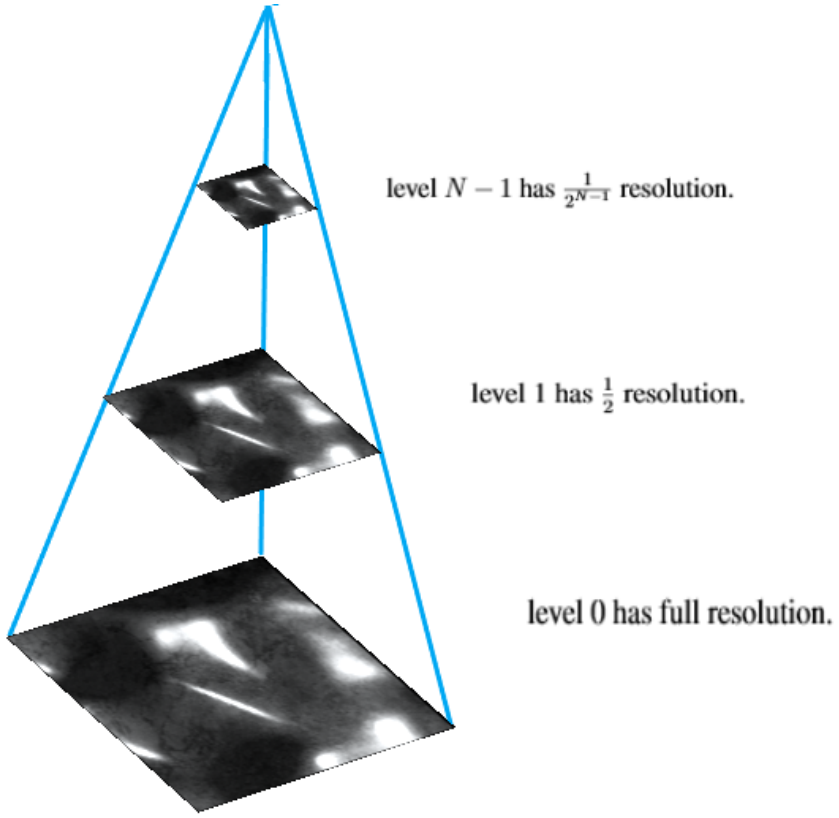


Figure 17. Image pyramid which has different scales of the original resolution.

5. To align patches as discussed in Subsection 5.4.1, each patch in a level i is aligned to the corresponding one in the higher level $i+1$ after upsampling the higher level to have the same resolution of the finer level. This is done starting from the last to the coarsest level that is level $N-2$. Figure 18 illustrates this process.

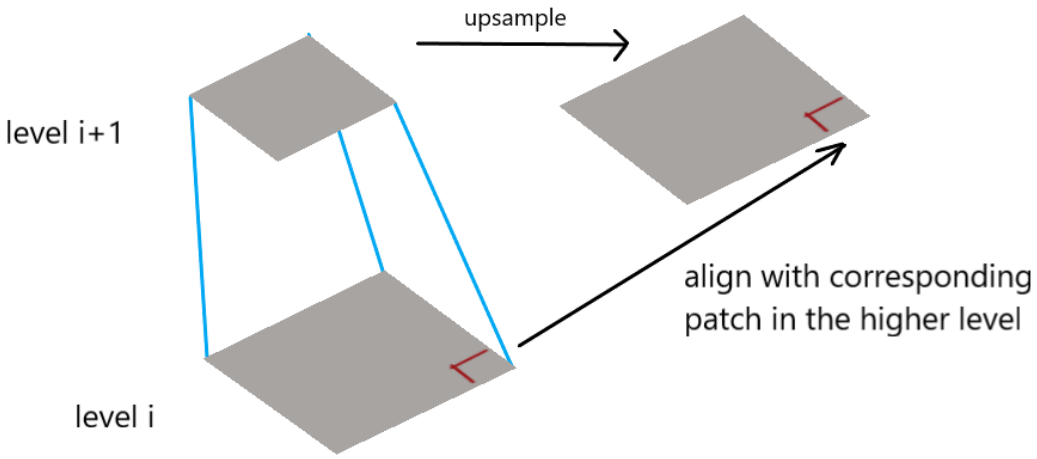


Figure 18. Aligning patches with the corresponding ones in the higher level.

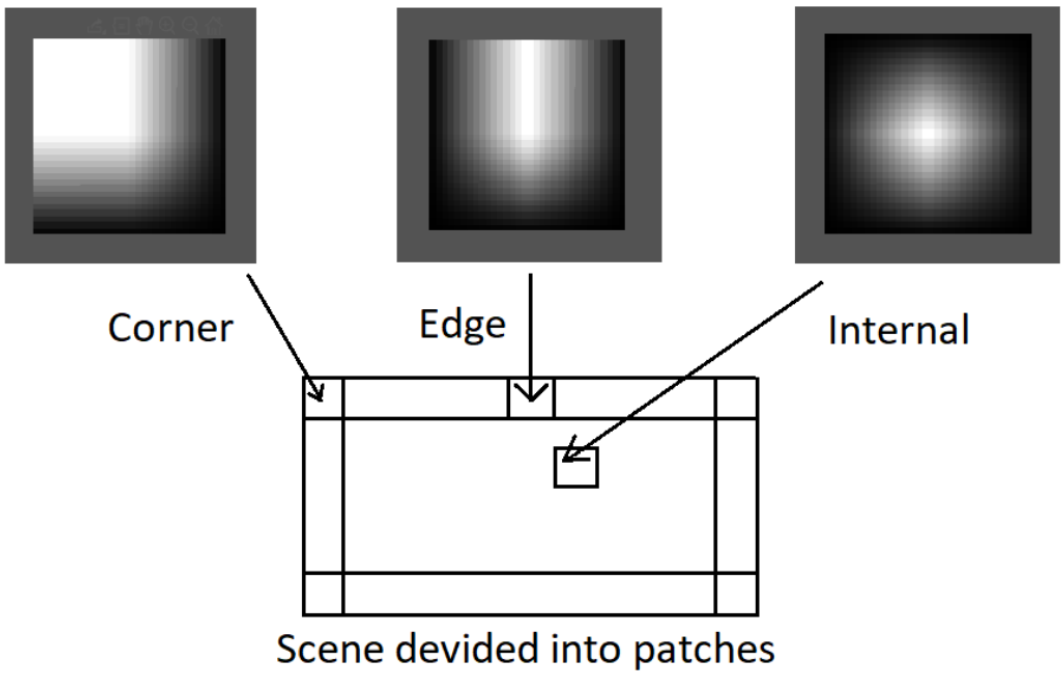


Figure 19. Top: Filters used to smooth the boundaries between patches. Corner and edge filters are rotated depending on the corner or edge position, respectively. A white pixel corresponds to one. A black pixel has a value of zero. Bottom: a large scene divided into patches. Note: overlapping between patches is not shown in the figure for simplicity but the idea applies for any percentage of overlapping anyways.

6. To smooth boundaries between overlapping patches, different filters are used depending on the position of the patch. Figure 19 illustrates the filters used to smooth the results. Three filters are used depending on the position (corner, edge and internal) of the patch within the large scene. Filters for upper left corner and upper edge are shown in the figure. Corner and edge filters are rotated depending on the corner or edge position in the scene, respectively. For example, to get the filter for the upper right corner, we rotate the corner filter for the upper left corner 90 degrees clockwise. Each level in the pyramid from the top is constructed in that way until reaching the bottom of the pyramid. Note that the weight of a pixel depth value contributed by a patch depends on the pixel location within that patch. The closer the pixel to the center, the more weight this patch contributes to the overall pixel depth value. Note also that the sum of weight contributions from different patches to a pixel depth value is one. These filters are used for 50% overlapping patches but similar approach can be used to obtain filters for any percentage of overlap between patches.
7. The final output is a weighted sum of the different levels in the pyramid where the sum of the weights is one. This is of course is done after upsampling each level to have the final resolution of the finest level. The final output is also scaled back to the original resolution of the scene which is h*w. That output can then be aligned to the ground truth depth map. We need this alignment also due to the inherit ambiguities as was discussed before.

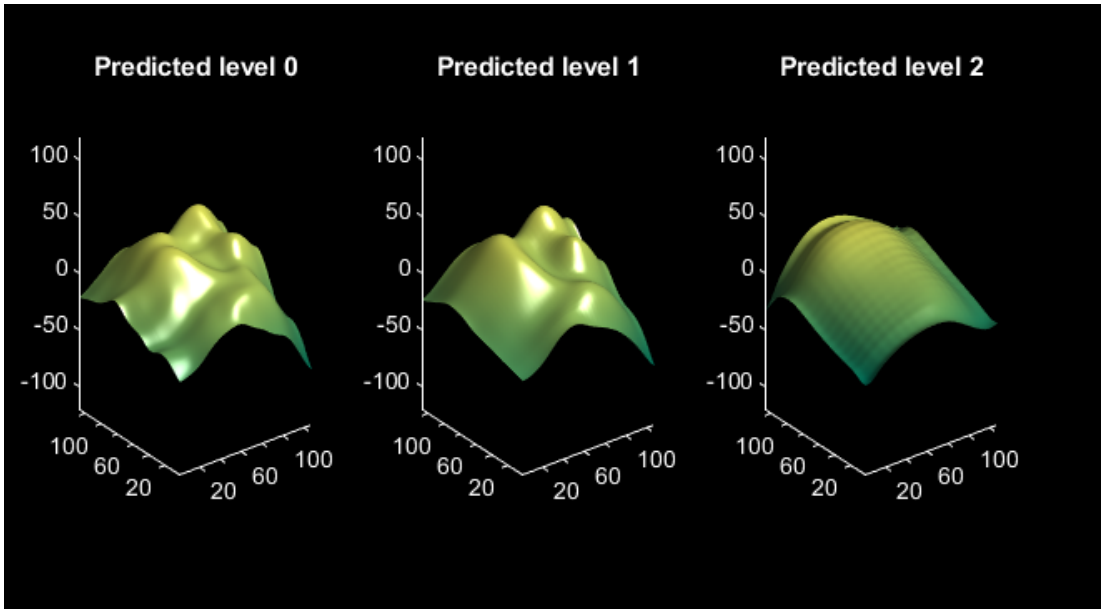


Figure 20. An example of predicted depth maps at different levels. Each level captures different frequency bands of folds or wrinkles. The coarsest level captures the lowest frequency whereas the finest level captures the highest ones with high details.

Figure 20 shows an example of a scene of size 128x128. So, we have 3 levels for that scene as the output depth map of a patch has a size of 32x32. It can be seen that each level captures specific frequency band of folds or wrinkles. The coarsest level captures the lowest frequencies and that is why it appears to be smooth with no much of details. The finer levels capture more details and higher frequencies of folds.

It is worth mentioning also that we tried to solve this problem using an optimization algorithm to solve for transformation parameters (Equation (23)) in the intersection area (the 50% overlap) between neighboring patches in order to stitch them. The first patch which is in the upper left corner did not undergo any transformation and all other patches were transformed to best align each other. This was done to get a single solution for these system of equations. The results were not bad but the parameter α which is the scale parameter had a small value for some patches which resulted in producing a slanted flat surface. This slanted flat surface was produced as a result of trying to best align the patches in the overlapping or intersection areas. That is why the previous method gave quite better results. Also, using this optimization algorithm (Levenberg-Marquardt-Fletcher) was very computationally expensive especially as we have many patches to be stitched especially in the lower levels of the pyramid.

5.5.2. Temporal Reconstruction of a Large Scene

As the case for spatial dimensions, we need to take care for inputs of large size for time dimension. However, the case for temporal dimension is much easier than the previous one. The network input and output is 16 frames as illustrated before. To construct a larger input in time domain, again we split the input into samples of 50% overlapping in time dimension. The final output is constructed using filters that give more weight to

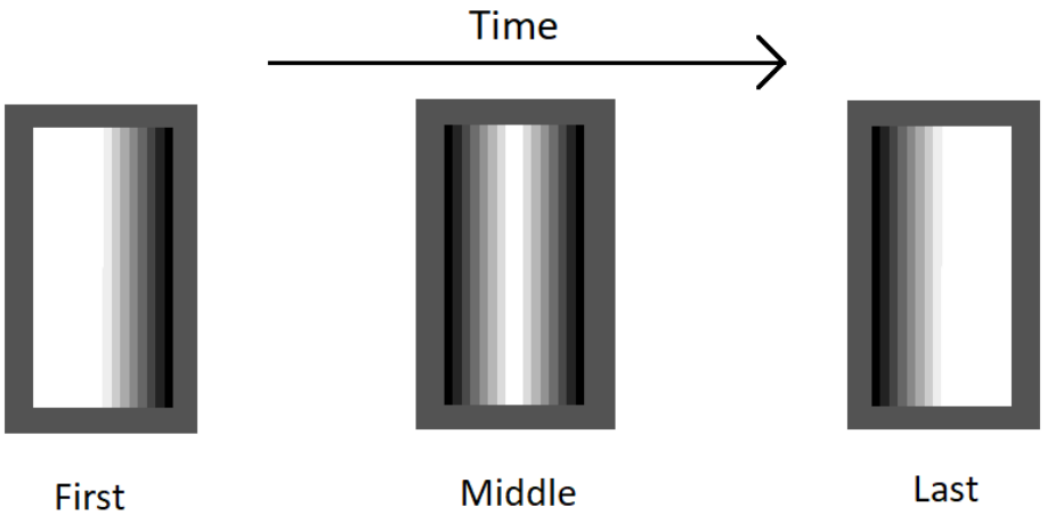


Figure 21. Filters used to smooth the boundaries between samples with 50% overlap. A white pixel corresponds to one. A black pixel has a value of zero. Left: the filter used for the first sample that is the first 16 frames of input. Right: the filter used for the last sample that is the last 16 frames of input. Middle: the filter used for the middle samples.

the frames in the middle of the sample. Figure 21 shows the filters used in this process. The filters are projected here and hence, only one dimension of space is shown but the filters are 3D of course. The filter size is 32x32x16 where the first 2 dimensions are space dimensions and the last one is a time dimension.

So, for a large scene, the patches are constructed across the time dimension first and then the spatial dimensions to get the final depth map of the large scene input.

5.6. Key Implementation Details

5.6.1. Dataset Generation

We used Matlab to artificially generate deforming samples. These samples were used to train, validate and test the network proposed in this chapter. The textures used in this dataset (5645 textures) are from the DTD database [63]. A parametric model was used to control the motion of the deforming object. A total of 6 parameters were used to control the motion: intensity controls the intensity or displacement along the z-axis, flexibility controls the flexibility or hardness/softness of the surface, folding controls the degree of folding seen in the surface, directionality controls the directionality or orientation of the displacement along the z-axis , constraint controls the degree of anchoring that the frame border has, speed controls the speed of the motion between consecutive frames. Figure 22 shows the effect of increasing the values of these parameters. A total of 153600 samples were generated. Each video sample has 16 frames of size 64×64 pixels. The ground truth depth maps were obtained using ray casting [64].

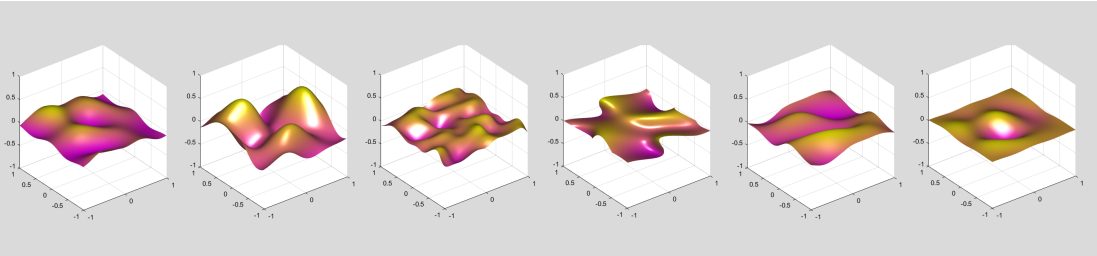


Figure 22. The effect of increasing the value of one parameter at a time. Left to right: a frame generated using small values for the motion parameters. increasing z-intensity. increasing flexibility. increasing folding. increasing directionality. increasing constraint.

5.6.2. Implementation of Differential Loss Functions

For differential invariant loss functions, Sobel operators were used for depth maps differentiation. Sobel operators can be seen in Figure 23. To get the derivative of the depth map with respect to x or y, Sobel operators or kernels for x-direction or y-direction are used, respectively. To get the second partial derivatives, two Sobel kernels are used. The first kernel is used to get the first derivative and then a second kernel is used to get the second derivative.

-

+1	0	-1
+2	0	-2
+1	0	-1

x-direction kernel

+1	+2	+1
0	0	0
-1	-2	-1

y-direction kernel

Figure 23. Sobel operators used for depth maps differentiation. Left: Sobel operator used to get the derivative with respect to x-direction. Right: Sobel operator used to get the derivative with respect to y-direction.

6. EXPERIMENTS AND RESULTS

In this chapter, we evaluate our method and compare it with other non-rigid structure-from-motion (NRSFM) state-of-the-art methods introduced in Chapter 2 (see Section 2.3). We compare our work with KSTA [10] and CSF2 [12] methods. For DDE [14] method, results could not be produced using our experiments due to source code unavailability. As was mentioned in the previous chapter, the metric used for evaluation is the spatially normalized mean absolute error (SNMAE) between the spatially normalized (have standard deviation of one) ground truth depth maps and the normalized predicted depth maps after aligning (see Section 5.4). This metric was used for artificial data. For real data, we used the mean absolute error (MAE) metric in millimeters. The first section compares the performance of our proposed solution using different loss functions (see Section 5.3). The second section compares our method and NRSFM state-of-the-art methods using synthetic data of large scenes. The third section compares our method and NRSFM state-of-the-art methods using real Kinect data.

6.1. Comparison of Model Performance Using Various Loss Functions

We trained the proposed network discussed in Chapter 4 using different differential and point cloud invariant loss functions. Out of the 153600 samples artificially generated, 80% of samples were used for training the network, 10% were used for validation to prevent overfitting while training and 10% were used for testing for this experiment. Each video sample has 16 frames of size 64×64 pixels. Table 1 shows the errors obtained using different invariant loss functions. The error is in the form of mean $\pm$ standard deviation. Each frame in the video sample is aligned with the corresponding ground truth frame. L_{t-sc_inv} represents the translation and scale along z-axis invariant loss function whereas L_{all_inv} represents the translation, scale and shear along z-axis invariant loss function. For the differential invariant loss function L_{all_inv}, different values for w were used (see Equation (27)).

From the table, we can see that the best performance of the model was obtained using the translation, scale and shear along z-axis differential invariant loss function using $w = 2$. Table 2 shows the errors obtained using different invariant loss functions but only the first predicted frame is aligned with the corresponding ground truth frame. All remaining frames in the sample are aligned using the same aligning parameters

Table 1. Performance of the model using different invariant loss functions. Each predicted frame is aligned with the corresponding ground truth frame.

Loss Function	Differential Invariant				Point Cloud Invariant	
	L_{t-sc_inv}	L_{all_inv}			L_{t-sc_inv}	L_{all_inv}
		$w = 1$	$w = 2$	$w = 4$		
Error (SNMAE)	0,4258 $\pm$ 0,1384	0,4413 $\pm$ 0,1165	**0,3832 $\pm$ 0,1305**	0,4049 $\pm$ 0,1231	0,6277 $\pm$ 0,1687	0,5226 $\pm$ 0,1123

Table 2. Performance of the model using different invariant loss functions. Only the first predicted frame is aligned with the corresponding ground truth frame. The alignment parameters used for aligning the first frame are used for aligning the remaining frames in the sample.

Loss Function	Differential Invariant				Point Cloud Invariant	
	L_{t-sc_inv}	L_{all_inv}			L_{t-sc_inv}	L_{all_inv}
		$w=1$	$w=2$	$w=4$		
Error (SNMAE)	0,6657 ± 0,2718	0,6573 ± 0,2552	**0,6133 ± 0,2591**	0,6200 ± 0,2493	0,8809 ± 0,2838	0,7090 ± 0,1971

used to align the first frame. In real time applications, it is not practical or might even be possible to align each frame to a ground truth frame as we do not have the ground truth data.

From the table, we can see again that the best performance of the model was obtained using the translation, scale and shear along z-axis differential invariant loss function using $w = 2$. The errors obtained are quite higher than in Table 1 which is anticipated. In Table 2, we used only the first frame for alignment purposes. However, the loss functions used are invariant to ambiguities. Hence, each frame would need different alignment parameters. Using the alignment parameters for the first frame to align all frames will not produce good results theoretically. However, as mentioned we needed this experiment for real applications and practicality purposes. We can see that the errors are increased by 50% roughly.

For the next experiments, we compare the performance of our proposed solution to the NRSFM state-of-the-art methods introduced in Chapter 2 (KSTA and CSF2). We used the model trained using the translation, scale and shear along z-axis differential invariant loss function using $w = 2$ as the best results for the previous experiments were achieved using this loss function.

6.2. Performance Comparison to NRSFM Conventional Methods Using Artificial Data

In this experiment, we compared our solution to NRSFM state-of-the-art methods KSTA and CSF2 using artificially generated samples. The samples produced using motion parameters that are different from the ones used to generate the training dataset. We generated a 1000 samples and each sample has 16 frames of size 256×256 pixels. So, we used the large scene reconstruction algorithm introduced in Chapter 5 (see Section 5.5) to combine patches and construct the full large scene. For the state-of-the-art NRSFM methods, we needed tracked points across the video frames to use them as an input to these algorithm. Since we generated the sequences artificially, we used the 2D locations of the 3D meshes vertices as input to these algorithms. The input to these algorithms is the optimal input as no 2D tracking algorithm is used. For all video samples, at least 1089 points were tracked in a video sample. The 2D tracked points were fed as an input to NRSFM algorithms to get the points in the 3D space.

Table 3. Comparison of the performance of state-of-the-art NRSFM methods and our method using artificially generated dataset.

Method	KSTA	CSF2	Ours
Error (SNMAE)	$0{,}8738 \pm 0{,}6369$	$0{,}8746 \pm 0{,}6372$	$\mathbf{0{,}5907 \pm 0{,}4536}$

To obtain a dense reconstruction so we can compare these methods with ours, we used scattered interpolation for each frame. Table 3 shows the errors produced by different methods. For all methods, all frames were aligned with the corresponding ground truth frames before calculating the errors.

From the table, we can see that our method outperformed other state-of-the-art NRSFM methods. Figure 24 shows the reconstruction of one frame produced by different methods. More examples can be found in Appendix 1.

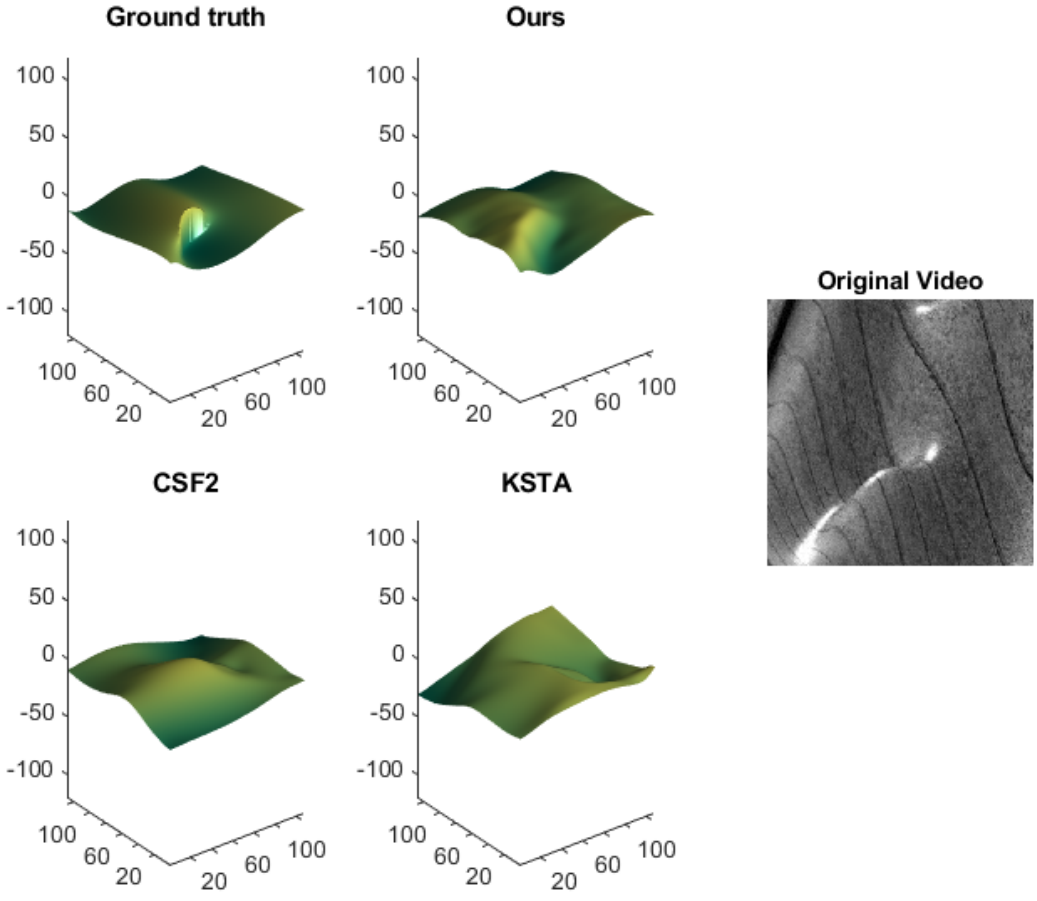


Figure 24. One frame reconstruction of an artificial sample using different methods.

Table 4. Comparison of the performance of state-of-the-art NRSFM methods and our method using Kinect data.

Method	KSTA	CSF2	Ours
Error (MAE) in mm	4,3	4,6	**3,7**

6.3. Performance Comparison to NRSFM Conventional Methods Using Real Kinect Data

A video of a person moving was captured using a Kinect 1 device. A fixed 64×64 region was cropped manually from the video which only contained deformations in the shirt caused by the person movement. The video sequence have 24 frames. To get the 2D tracked points for real Kinect data, we used the minimum eigenvalue algorithm [65]. A total of 31 points were fully tracked across the video using this algorithm. The 3D points were obtained using NRSFM methods and scattered interpolation was used to obtain a dense representation as was the case for synthetic data. For all methods, all frames were aligned with the corresponding ground truth frames obtained from the Kinect 1 device before calculating the errors. Gaussian filtering was used to smooth the noisy ground truth depth data. Table 4 shows the errors in millimeters produced by different methods. Figure 25 shows the reconstruction of one frame produced by different methods.

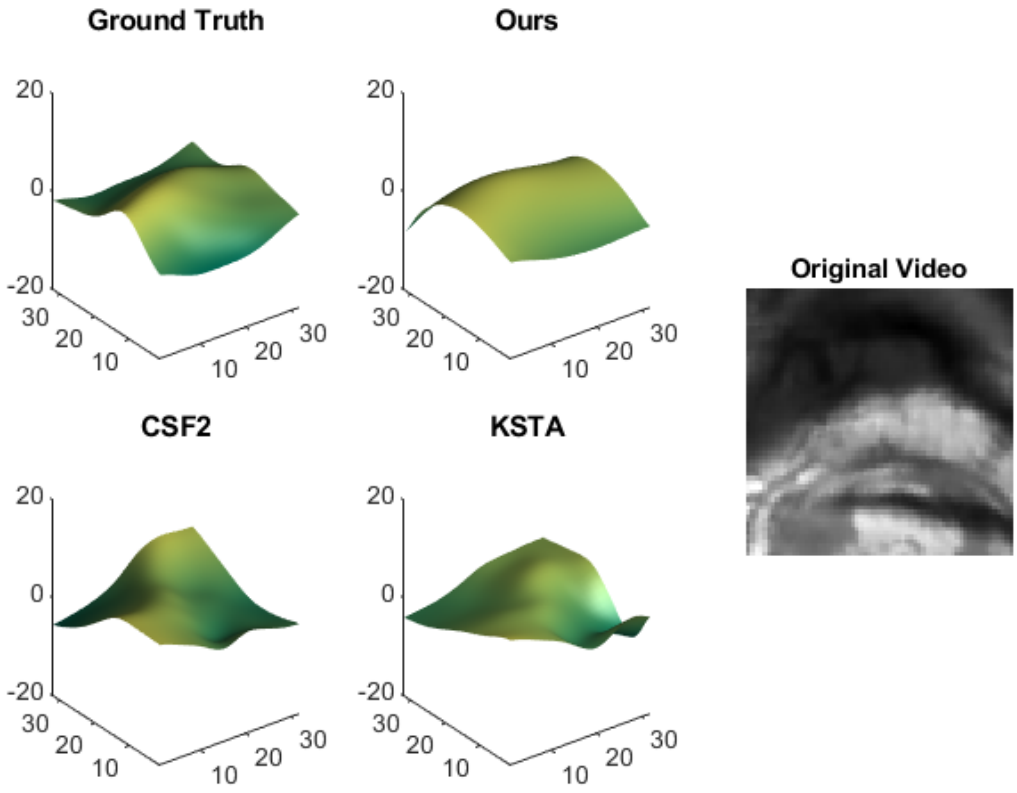


Figure 25. One frame reconstruction of a real Kinect sample using different methods.

7. DISCUSSION

In this thesis, we presented a Deep Learning (DL) approach to estimate spatio-temporal depth maps directly from grey-scale videos. We compared our work with other state-of-the-art non-rigid structure-from-motion (NRSFM) methods. The first two sections discuss the results obtained using both synthetic and real Kinect data. The last section discusses some of the limitations and possible future work.

7.0.1. Results Using Synthetic Data

The network was trained using synthetic dataset that was generated using a simple parametric model with few parameters. In order to make the comparison fair, we generated a large scene dataset using values for motion parameters that are different than the ones used in training. For NRSFM methods, the optimal input was provided since the dataset was generated artificially using rendered 3D meshes and we can obtain the 2D locations of meshes vertices easily. Our method significantly outperformed other state-of-the-art NRSFM methods (see Table 3).

7.0.2. Results Using Real Kinect Data

Testing a method using real data is very important to see the applicability of using this method in real life applications. Our method outperformed other state-of-the-art NRSFM methods (see Table 4). However, this experiment was performed on only one video sequence. More testing is needed to ensure that the method works for real life scenarios.

7.0.3. Limitations and Possible Future Work

We trained the network using data that were captured from a static camera and we did not solve for camera motion. However, this is not the case that we have for real life scenarios. So, we might investigate the feasibility of solving the camera motion when the camera is moving. However, there is an ambiguity concerning that it is impossible to know whether the non-rigid object is expanding while the camera is static or the camera is moving while capturing a slightly expanding or a constant-size object [1]. So, more investigation regarding this problem is needed.

Another assumption is that the global shape of a scene can be modeled by combining the small local patches that are assumed to have the deformations that can be realized by the parametric model that we used to generate the data. In this case, the shape deformations that can be constructed might be limited. We might need to train the network on a dataset using a wide range of values for the motion parameters.

For real data, more testing is needed to ensure the applicability of this method. This might in turn guides us to use more realistic model to generate the training dataset if needed.

Another limitation is the case with dynamic scenes with more than one object. The scene might have rigid or non-rigid objects with background. Our method only assumes that the camera is capturing one object and no background exist in the scene. This limitation needs to be addressed.

The global scene reconstruction algorithm might need to be improved in order to make sure that this is the optimal way to construct the global scene from small local patches.

Another future work includes investigating the feasibility of using optical flow to get better performance using this DL-based method.

8. CONCLUSION

Estimating the spatio-temporal depth map of non-rigid deforming objects directly from video sequences is not an easy task due to the ill-posedness of the problem. Researchers have used many techniques to address this problem. Given the recent success of Deep Learning (DL), we addressed this problem by using a DL-based approach.

Training a Deep Convolutional Neural Network requires a large-scale dataset. We addressed this problem by artificially generating a dataset and using it in training the network. Although the network was not trained using real data, our experiments have shown that our method performs well on real Kinect data. Hence, using synthetic data to train a network is more practical and promising.

In this thesis, a DL-based approach was presented to estimate the spatio-temporal depth map of non-rigid deforming objects directly from small local patches of video sequences. In order to obtain a large-scale dataset to train the network, we artificially generated a dataset using a parametric model that has few parameters. We presented an algorithm to construct the global scene from small local video patches. We compared our method against other state-of-the-art non-rigid structure-from-motion (NRSFM) methods and found that our method outperformed these methods on both synthetic and real Kinect data.

9. REFERENCES

[1] Salzmann M. & Fua P. (2010) Deformable surface 3d reconstruction from monocular images. Synthesis Lectures on Computer Vision 2, pp. 1–113.

[2] Tomasi C. & Kanade T. (1992) Shape and motion from image streams: a factorization method: full report on the orthographic case. Tech. rep., Cornell University.

[3] Woodham R.J. (1980) Photometric method for determining surface orientation from multiple images. Optical engineering 19, p. 191139.

[4] Hartley R. & Zisserman A. (2004) Multiple View Geometry in Computer Vision. Cambridge University Press, 2 ed.

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

[6] Harris C.G., Stephens M. et al. (1988) A combined corner and edge detector. In: Alvey vision conference, 50, Citeseer, pp. 10–5244.

[7] Garg R., Roussos A. & Agapito L. (2013) Dense variational reconstruction of non-rigid surfaces from monocular video. In: 2013 IEEE Conference on Computer Vision and Pattern Recognition, pp. 1272–1279.

[8] Computations M. (1996) Johns hopkins university press. Baltimore, 199.

[9] Belhumeur P.N., Kriegman D.J. & Yuille A.L. (1999) The bas-relief ambiguity. Int. J. Comput. Vision 35, p. 33–44. URL: https://doi.org/10.1023/A:1008154927611.

[10] Gotardo P.F.U. & Martinez A.M. (2011) Kernel non-rigid structure from motion. In: 2011 International Conference on Computer Vision, pp. 802–809.

[11] Schölkopf B., Smola A.J., Bach F. et al. (2002) Learning with kernels: support vector machines, regularization, optimization, and beyond. MIT press.

[12] Gotardo P.F.U. & Martinez A.M. (2011) Non-rigid structure from motion with complementary rank-3 spaces. In: CVPR 2011, pp. 3065–3072.

[13] Chen W.H., Smith C. & Fralick S. (1977) A fast computational algorithm for the discrete cosine transform. IEEE Transactions on communications 25, pp. 1004–1009.

[14] Kumar S., Ghorakavi R.S., Dai Y. & Li H. (2019), Dense depth estimation of a complex dynamic scene without explicit 3d motion estimation.

[15] Hochreiter S. (1998) The vanishing gradient problem during learning recurrent neural nets and problem solutions. Int. J. Uncertain. Fuzziness Knowl.-Based Syst. 6, p. 107–116. URL: https://doi.org/10.1142/S0218488598000094.

[16] Nair V. & Hinton G.E. (2010) Rectified linear units improve restricted boltzmann machines. In: Proceedings of the 27th international conference on machine learning (ICML-10), pp. 807–814.

[17] Kingma D.P. & Ba J. (2014), Adam: A method for stochastic optimization.

[18] LeCun Y., Boser B., Denker J.S., Henderson D., Howard R.E., Hubbard W. & Jackel L.D. (1989) Backpropagation applied to handwritten zip code recognition. Neural computation 1, pp. 541–551.

[19] Ronneberger O., Fischer P. & Brox T. (2015), U-net: Convolutional networks for biomedical image segmentation.

[20] Ummenhofer B., Zhou H., Uhrig J., Mayer N., Ilg E., Dosovitskiy A. & Brox T. (2017) Demon: Depth and motion network for learning monocular stereo. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) URL: http://dx.doi.org/10.1109/CVPR.2017.596.

[21] Eigen D., Puhrsch C. & Fergus R. (2014), Depth map prediction from a single image using a multi-scale deep network.

[22] Eigen D. & Fergus R. (2014), Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture.

[23] Wang X., Fouhey D.F. & Gupta A. (2014), Designing deep networks for surface normal estimation.

[24] Qi X., Liao R., Liu Z., Urtasun R. & Jia J. (2018) Geonet: Geometric neural network for joint depth and surface normal estimation. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 283–291.

[25] He L., Wang G. & Hu Z. (2018) Learning depth from single images with deep neural network embedding focal length. IEEE Transactions on Image Processing 27, p. 4676–4689. URL: http://dx.doi.org/10.1109/TIP.2018.2832296.

[26] Simonyan K. & Zisserman A. (2014), Very deep convolutional networks for large-scale image recognition.

[27] Gordon A., Li H., Jonschkowski R. & Angelova A. (2019), Depth from videos in the wild: Unsupervised monocular depth learning from unknown cameras.

[28] Liu F., Shen C., Lin G. & Reid I. (2016) Learning depth from single monocular images using deep convolutional neural fields. IEEE Transactions on Pattern Analysis and Machine Intelligence 38, p. 2024–2039. URL: http://dx.doi.org/10.1109/TPAMI.2015.2505283.

[29] Xu Q., Wang W., Ceylan D., Mech R. & Neumann U. (2019), Disn: Deep implicit surface network for high-quality single-view 3d reconstruction.

[30] Kurenkov A., Ji J., Garg A., Mehta V., Gwak J., Choy C. & Savarese S. (2017), Deformnet: Free-form deformation network for 3d shape reconstruction from a single image.

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

[32] Wang Y., Wang P., Yang Z., Luo C., Yang Y. & Xu W. (2019) Unos: Unified unsupervised optical-flow and stereo-depth estimation by watching videos. In: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 8063–8073.

[33] Fischer P., Dosovitskiy A., Ilg E., Häusser P., Hazırbaş C., Golkov V., van der Smagt P., Cremers D. & Brox T. (2015), Flownet: Learning optical flow with convolutional networks.

[34] Ilg E., Mayer N., Saikia T., Keuper M., Dosovitskiy A. & Brox T. (2016), Flownet 2.0: Evolution of optical flow estimation with deep networks.

[35] Chen L.C., Papandreou G., Kokkinos I., Murphy K. & Yuille A.L. (2016), Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs.

[36] Milletari F., Ahmadi S.A., Kroll C., Plate A., Rozanski V., Maiostre J., Levin J., Dietrich O., Ertl-Wagner B., Bötzel K. & Navab N. (2016), Hough-cnn: Deep learning for segmentation of deep brain regions in mri and ultrasound.

[37] Zhou X.Y., Zheng J.Q., Li P. & Yang G.Z. (2019), Acnn: a full resolution dcnn for medical image segmentation.

[38] Juarez A.G.U., Selvan R., Saghir Z. & de Bruijne M. (2019), A joint 3d unet-graph neural network-based method for airway segmentation from chest cts.

[39] Shan S., Yan W., Guo X., Chang E.I.C., Fan Y. & Xu Y. (2017), Unsupervised end-to-end learning for deformable medical image registration.

[40] Çiçek Ö., Abdulkadir A., Lienkamp S.S., Brox T. & Ronneberger O. (2016) 3d u-net: Learning dense volumetric segmentation from sparse annotation. CoRR abs/1606.06650. URL: `http://arxiv.org/abs/1606.06650`.

[41] Milletari F., Navab N. & Ahmadi S.A. (2016), V-net: Fully convolutional neural networks for volumetric medical image segmentation.

[42] Zhou S., Zhang J., Zuo W., Xie H., Pan J. & Ren J. (2019) DAVANet: Stereo deblurring with view aggregation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.

[43] Su S., Delbracio M., Wang J., Sapiro G., Heidrich W. & Wang O. (2016), Deep video deblurring.

[44] Nah S., Kim T.H. & Lee K.M. (2016), Deep multi-scale convolutional neural network for dynamic scene deblurring.

[45] Golyanik V., Shimada S., Varanasi K. & Stricker D. (2018), Hdm-net: Monocular non-rigid 3d reconstruction with learned deformation model.

[46] He K., Zhang X., Ren S. & Sun J. (2015), Deep residual learning for image recognition.

[47] Bednařík J., Fua P. & Salzmann M. (2018), Learning to reconstruct texture-less deformable surfaces from a single view.

[48] Tsoli A. & Argyros A.A. (2019) Patch-based reconstruction of a textureless deformable 3d surface from a single rgb image. In: 2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW), pp. 4034–4043.

[49] Danundefinedřek R., Dibra E., Öztireli C., Ziegler R. & Gross M. (2017) Deepgarment: 3d garment shape estimation from a single image. Comput. Graph. Forum 36, p. 269–280. URL: https://doi.org/10.1111/cgf.13125.

[50] Iandola F.N., Han S., Moskewicz M.W., Ashraf K., Dally W.J. & Keutzer K. (2016), Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size.

[51] Krizhevsky A., Sutskever I. & Hinton G.E. (2012) Imagenet classification with deep convolutional neural networks. In: F. Pereira, C.J.C. Burges, L. Bottou & K.Q. Weinberger (eds.) Advances in Neural Information Processing Systems 25, Curran Associates, Inc., pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[52] Pumarola A., Agudo A., Porzi L., Sanfeliu A., Lepetit V. & Moreno-Noguer F. (2018), Geometry-aware network for non-rigid shape prediction from a single view.

[53] Wei S.E., Ramakrishna V., Kanade T. & Sheikh Y. (2016), Convolutional pose machines.

[54] Ronneberger O., Fischer P. & Brox T. (2015) U-net: Convolutional networks for biomedical image segmentation. CoRR abs/1505.04597. URL: http://arxiv.org/abs/1505.04597.

[55] Szegedy C., Vanhoucke V., Ioffe S., Shlens J. & Wojna Z. (2016) Rethinking the inception architecture for computer vision. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition,. URL: http://arxiv.org/abs/1512.00567.

[56] Zhou S., Zhang J., Zuo W., Xie H., Pan J. & Ren J.S.J. (2019) Davanet: Stereo deblurring with view aggregation. CoRR abs/1904.05065. URL: http://arxiv.org/abs/1904.05065.

[57] Olver P.J. (1999) Classical Invariant Theory. London Mathematical Society Student Texts, Cambridge University Press.

[58] Kendall D.G. (1989) A survey of the statistical theory of shape. Statistical Science 4, pp. 87–99. URL: http://www.jstor.org/stable/2245331.

[59] Bednarík J., Fua P. & Salzmann M. (2018) Learning shape-from-shading for deformable surfaces. CoRR abs/1803.08908. URL: http://arxiv.org/abs/1803.08908.

[60] Tsoli A. & Argyros A.A. (2019) Patch-based reconstruction of a textureless deformable 3d surface from a single rgb image. 2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW) , pp. 4034–4043.

[61] Burt P. & Adelson E. (1983) The laplacian pyramid as a compact image code. IEEE Transactions on communications 31, pp. 532–540.

[62] Adelson E.H., Anderson C.H., Bergen J.R., Burt P.J. & Ogden J.M. (1984) Pyramid methods in image processing. RCA engineer 29, pp. 33–41.

[63] Cimpoi M., Maji S., Kokkinos I., Mohamed S. & Vedaldi A. (2013), Describing textures in the wild.

[64] Roth S.D. (1982) Ray casting for modeling solids. Computer Graphics and Image Processing 18, pp. 109 – 144. URL: http://www.sciencedirect.com/science/article/pii/0146664X82901691.

[65] Jianbo Shi & Tomasi (1994) Good features to track. In: 1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 593–600.

10. APPENDICES

Appendix 1 One frame reconstruction of two artificial samples using different methods.