3-D SCENE RECONSTRUCTION FOR PASSIVE RANGING USING DEPTH FROM DEFOCUS AND DEEP LEARNING

A Dissertation

Submitted to the Faculty

of

Purdue University

by

David R. Emerson

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

Purdue University

Indianapolis, Indiana

August 2019

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

Dr. Lauren A Christopher, Chair

Department of Electrical Engineering

Dr. Zina Ben Miled

Department of Electrical Engineering

Dr. Brian King

Department of Electrical Engineering

Dr. Paul Salama

Department of Electrical Engineering

Approved by:

Dr. Brian King

Head of the Graduate Program

This dissertation is dedicated to my wife, Meggan, and my two wonderful kids, Mackenzie and Braelyn, who encouraged me to pursue my dreams and put up with all of the stress and late nights.

ACKNOWLEDGMENTS

Foremost, I would like to thank my committee chair, Dr. Lauren Christopher, for her support and expert guidance in my studies. I would also like to thank my committee members Dr. Brian King, Dr. Zina Ben Miled and Dr. Paul Salama for their support in these endeavors.

For the use of Indiana University's super computer cluster Big Red II, I would like to acknowledge the supported in part by Lilly Endowment, Inc., through its support for the Indiana University Pervasive Technology Institute, and in part by the Indiana METACyt Initiative. The Indiana METACyt Initiative at IU was also supported in part by Lilly Endowment, Inc.

I would like to thank NVIDIA® for their donation of a Titan Xp GPU through their Academic GPU Grant Program (`https://developer.nvidia.com/academic_gpu_seeding`). Without their generous donation we would not have been able to complete this research in a practical amount of time.

I would also like to thank the Science, Mathematics And Research for Transformation (SMART) Scholarship for Service Program for the opportunity to continue my higher education in pursuit of my degree.

I would like to thank the NSWC Crane for allowing me to participate in the Crane PhD Fellowship to complete my degree.

Finally I would like to thank Sherrie Tucker for all of her support in getting this document prepared and ready for publishing.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Figure Page

Figure Page

Figure Page

Figure Page

LIST OF ALGORITHMS

ABBREVIATIONS

2-D	2-dimensional
3-D	3-dimensional
4-D	4-dimensional
abbr	Abbreviation
CCD	Charge-Coupled Device
CMOS	Complementary Metal Oxide Semiconductor
CNN	Convolutional Neural Network
CoC	Circle of Confusion
CV	Computer Vision
DAG	Directed Acyclic Graph
DCM	Differential Contrast Method
DfD	Depth from Defocus
DfF	Depth from Focus
DfS	Depth from Stereo
DoF	Depth of Field
DNN	Deep Neural Network
ELU	Exponential Linear Unit
FPGA	Field Programmable Gate Arrays
GA	Genetic Algorithm
GC	Graph Cuts
HVS	Human Visual System
ICM	Intensity Contrast Method
MRF	Markov Random Field
MSE	Mean Square Error

NMAE	Normalized Mean Absolute Error
NRMSE	Normalized Root Mean square Error
OLS	Ordinary Least Squares
PC	Personal Computer
PSO	Particle Swarm Optimization
pReLU	Parametric Rectified Linear Unit
ReLU	Rectified Linear Unit
SBC	Single Board Computer
SGD	Stochastic Gradient Descent
SGM	Semi-Global Method
SOM	Self-Organizing Map
sReLU	s-Shaped Rectified Linear Unit
SSIM	Structural Similarity Index Measure
Std	Standard Deviation
UDP	User Datagram Protocol
USB	Universal Serial Bus
vs.	Versus

ABSTRACT

Emerson, David R. Ph.D., Purdue University, August 2019. 3-D Scene Reconstruction for Passive Ranging Using Depth from Defocus and Deep Learning. Major Professor: Lauren A. Christopher.

Depth estimation is increasingly becoming more important in computer vision. The requirement for autonomous systems to gauge their surroundings is of the utmost importance in order to avoid obstacles, preventing damage to itself and/or other systems or people. Depth measuring/estimation systems that use multiple cameras from multiple views can be expensive and extremely complex. And as these autonomous systems decrease in size and available power, the supporting sensors required to estimate depth must also shrink in size and power consumption.

This research will concentrate on a single passive method known as Depth from Defocus (DfD), which uses an in-focus and out-of-focus image to infer the depth of objects in a scene. The major contribution of this research is the introduction of a new Deep Learning (DL) architecture to process the the in-focus and out-of-focus images to produce a depth map for the scene improving both speed and performance over a range of lighting conditions. Compared to the previous state-of-the-art multi-label graph cuts algorithms applied to the synthetically blurred dataset the DfD-Net produced a 34.30% improvement in the average Normalized Root Mean Square Error (NRMSE). Similarly the DfD-Net architecture produced a 76.69% improvement in the average Normalized Mean Absolute Error (NMAE). Only the Structural Similarity Index (SSIM) had a small average decrease of 2.68% when compared to the graph cuts algorithm. This slight reduction in the SSIM value is a result of the SSIM metric penalizing images that appear to be noisy. In some instances the DfD-Net output is mottled, which is interpreted as noise by the SSIM metric.

This research introduces two methods of deep learning architecture optimization. The first method employs the use of a variant of the Particle Swarm Optimization (PSO) algorithm to improve the performance of the DfD-Net architecture. The PSO algorithm was able to find a combination of the number of convolutional filters, the size of the filters, the activation layers used, the use of a batch normalization layer between filters and the size of the input image used during training to produce a network architecture that resulted in an average NRMSE that was approximately 6.25% better than the baseline DfD-Net average NRMSE. This optimized architecture also resulted in an average NMAE that was 5.25% better than the baseline DfD-Net average NMAE. Only the SSIM metric did not see a gain in performance, dropping by 0.26% when compared to the baseline DfD-Net average SSIM value.

The second method illustrates the use of a Self Organizing Map clustering method to reduce the number convolutional filters in the DfD-Net to reduce the overall run time of the architecture while still retaining the network performance exhibited prior to the reduction. This method produces a reduced DfD-Net architecture that has a run time decrease of between 14.91% and 44.85% depending on the hardware architecture that is running the network. The final reduced DfD-Net resulted in a network architecture that had an overall decrease in the average NRMSE value of approximately 3.4% when compared to the baseline, unaltered DfD-Net, mean NRMSE value. The NMAE and the SSIM results for the reduced architecture were 0.65% and 0.13% below the baseline results respectively. This illustrates that reducing the network architecture complexity does not necessarily reduce the reduction in performance.

Finally, this research introduced a new, real world dataset that was captured using a camera and a voltage controlled microfluidic lens to capture the visual data and a 2-D scanning LIDAR to capture the ground truth data. The visual data consists of images captured at seven different exposure times and 17 discrete voltage steps per exposure time. The objects in this dataset were divided into four repeating scene patterns in which the same surfaces were used. These scenes were located between 1.5 and 2.5 meters from the camera and LIDAR. This was done so any of the deep

learning algorithms tested would see the same texture at multiple depths and multiple blurs. The DfD-Net architecture was employed in two separate tests using the real world dataset.

The first test was the synthetic blurring of the real world dataset and assessing the performance of the DfD-Net trained on the Middlebury dataset. The results of the real world dataset for the scenes that were between 1.5 and 2.2 meters from the camera the DfD-Net trained on the Middlebury dataset produced an average NRMSE, NMAE and SSIM value that exceeded the test results of the DfD-Net tested on the Middlebury test set. The second test conducted was the training and testing solely on the real world dataset. Analysis of the camera and lens behavior led to an optimal lens voltage step configuration of 141 and 129. Using this configuration, training the DfD-Net resulted in an average NRMSE, NMAE and SSIM of 0.0660, 0.0517 and 0.8028 with a standard deviation of 0.0173, 0.0186 and 0.0641 respectively.

1. INTRODUCTION

Depth estimation is considered to be one of the most challenging problems in computer vision (CV) today. The human visual system (HVS) takes advantage of several cues to determine distances of objects, the most important being the binocular cue which results "from the fact that both eyes see an object under a different angle" [1]. With the ever increasing advancements in robotics and autonomous systems the need for improved (accurate and fast) depth estimation algorithms and techniques is essential. The integration of self-driving cars and other autonomous systems into main-stream society means that these systems can no longer rely on a simple 2-dimensional (2-D) image to completely understand the world in which these systems are operating.

Depth information can be determined in many different ways. These methods can generally be broken down into two broad categories; 1) Active Methods and 2) Passive Methods. While the majority of this research will concentrate on one particular method of passive depth inference, it is important to understand the engineering trade-offs between active and passive depth estimation methods and the ultimately the trade-offs between the passive methods themselves.

1.1 Active Depth Estimation Methods

Active ranging methods are a very diverse set. Systems can use ultrasonic ranging, they can use visible light in the form of structured light patterns, non-visible laser light like that used in laser range finders and LIDAR systems, or electromagnetic emissions in the radio frequency (RF) spectrum like that of radar. Whatever the system employed, they all have at least two things in common. They each require a transmitter which emits a signal in the band of interest and they need a matched

receiver which will receive and decode the returned signal. These methods can be highly accurate because they are directly measuring the distance to a point of interest in 3-D space. However, their size and expense can be prohibitive when smaller autonomous systems require accurate depth information. For example imagine placing an $8,000 LIDAR system on a $1,000 quad copter. In some applications, like covert surveillance, the system needs to be passive so as not to draw attention to the depth estimation system. Therefore, there is a need to invest in passive, small, lightweight and power efficient technologies.

1.2 Passive Depth Estimation Methods

Passive methods have no active transmission, so the depth estimation occurs through what is termed depth inference, which is to say the depth is estimated by using specific cues that indicate the potential depth of an object. These methods have several advantages over the active methods. The largest is that they do not require a matched transmitter, which can add additional cost, weight and complexity to a depth estimation system. A drawback to the passive methods is that the algorithms used to infer the depth can be computationally expensive and may not be as accurate as the active methods. The passive depth estimation methods discussed here are 1) Depth from Stereo, 2) Depth from Focus and 3) Depth from Defocus.

When considering the various methods to infer depth from a series 2-D images, one has to keep in mind that there are two interdependent systems at work. The first being the algorithmic system in which computer vision and statistical techniques are used to develop depth maps based on the required algorithm inputs. The second system is the physical system which includes the digital imaging sensor, and a lens which has properties described in Chapter 2. Only when the two systems are correctly paired are the results accurate.

1.2.1 Depth From Stereo

Depth from Stereo (DfS), also known as Stereo Vision, is a technique for inferring depth by triangulation from two cameras that are a set distance apart from each other [2]. This is very similar to the way human vision works. The relative depths of objects in a given scene are obtained by comparing the two images and performing a matching correspondence between the same objects in the scene. This creates a disparity map, which is inversely proportional to the scene depth at the corresponding pixel location [3]. The term disparity in the realm of stereo vision refers to the difference in distance that an object projects onto the image plane of each camera.

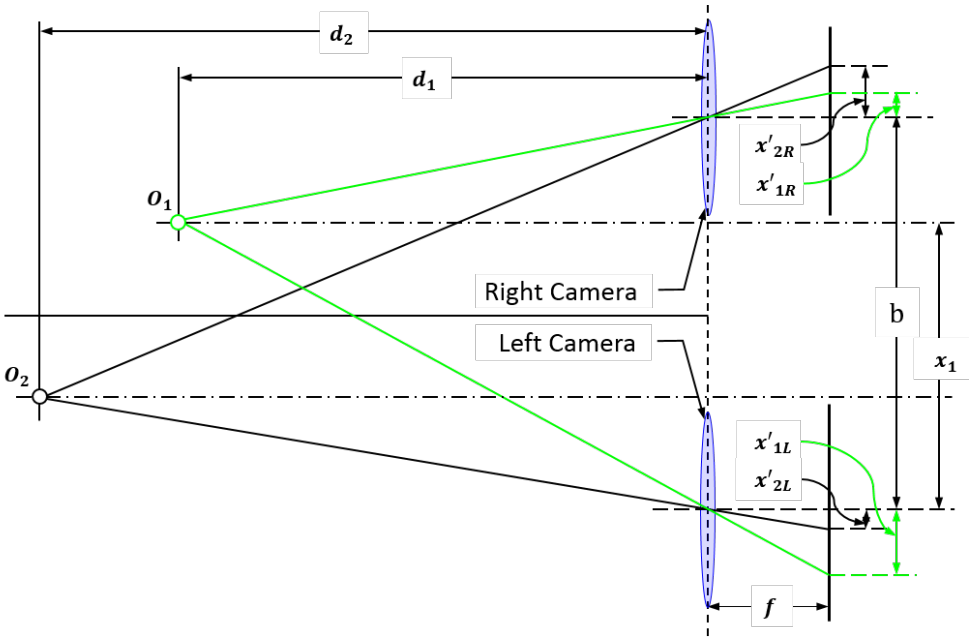


Fig. 1.1. Geometry setup of a typical Depth from Stereo setup. Image adapted from [3, 4]

DfS methods were once considered to be more sensitive (i.e. have higher resolution) than either Depth from Focus or Depth from Defocus methods. However, it has been shown that mathematically the disparity produced by DfS methods and the Depth from Defocus method are identical [5]. In fact, the only reason that DfS methods are considered to have higher resolution is a function of the physical configu-

ration of the two cameras. This can be seen in Figure 1.1. Effectively the two camera system forms a synthetic aperture of a larger system [5] which can be several times larger than a standard camera lens. Where as the Depth from Focus and the Depth from Defocus methods use a single camera, limiting the aperture size to a single lens system.

In order to determine the depth of an object, O_1 for example, the method of similar triangles is used. A set of traingles are considered similar if they meet the following definition: "Triangles ΔABC and $\Delta A'B'C'$ are *similar* iff corresponding angles are congruent and the lengths of corresponding sides are proportional." [6]

$$\frac{x_1}{d_1} = \frac{x'_{1L}}{f} \tag{1.1}$$

$$\frac{x_1 - b}{d_1} = \frac{x'_{1R}}{f} \tag{1.2}$$

Starting from a reference origin of the left lens, the ratios are shown in Equations 1.1 and 1.2. Solving each equation for x_1, setting them equal to each other and then solving for the depth of the object d_1 results in the following equation:

$$d_1 = \frac{fb}{x'_{1L} - x'_{1R}} \tag{1.3}$$

It should be noted that the accuracy of the depth is also limited by the resolution of the digital imaging system, since the pixel location is the unit of measure for x'_{1L} and x'_{1R}. There are however special algorithms that can compute the disparities at the sub-pixel level [4]. One of the drawbacks to the DfS configuration is that only a horizontal separation is assumed [4], which means that in order for the algorithms to work correctly the cameras must be perfectly aligned in the vertical dimension.

In addition to a perfect vertical alignment, it is preferred to have the camera centerlines parallel. However, the modern algorithms used in DfS can accommodate for some angular rotation in the cameras. That being said, too much rotational

difference and the same object will no longer be in both the left and right image image planes. Another limitation to the DfS setup is object occlusion. This is where one object masks another object(s) in one of the cameras, but not the other camera. This prevents the algorithms from correctly corresponding objects in the left/right image plane.

Many different algorithmic approaches have been proposed to solve the challenge of developing depth maps from stereo image pairs. In fact [2, 7] have proposed using Markov Random Fields (MRF) to develop the depth map. In addition to MRFs, Hirschmüller and Scharstein have proposed several other methods including a correlation-based method [8], the use of the semi-global method (SGM) developed by Hirschmüller [9] and the use of a global method using graph cuts (GC) was introduced by Boykov, et al. [10]. In addition to traditional stochastic models, the use of deep learning models employing convolutional neural networks (CNNs) for DfS have also been proposed by Žbontar and LeCun [11] and Lou, et al. [12].

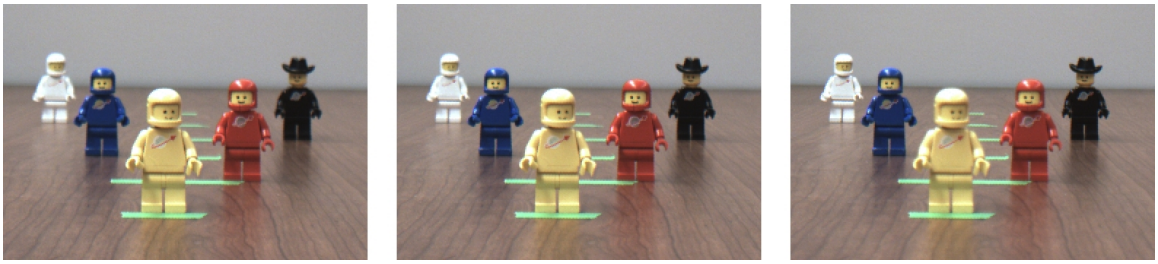
1.2.2 Depth From Focus

As with Depth from Stereo, the purpose of Depth from Focus (DfF), also known as shape from focus, is to estimate the depth of objects in a scene. This method is considered an ill-posed problem [13], which means that it does not meet the Hadamard criteria [14]. This method differs from the DfS method, in that instead of using two cameras that are separated by a given distance, a single camera is used to capture the scene. The required data consists of a focal stack which is a series of images where each image has its focus distance increased (or decreased depending on starting location) so that objects in a scene gradually come in and out of focus.

The central idea is then to assume that for every pixel located at (x, y, z), where x and y are the pixel locations in an image and z is the index of the focal stack, there is one pixel that is maximally sharp. The index of the image in the focal stack with the maximally sharp pixel can then be related to the focus distance, which is the distance

from the imaging system to a plane where everything in that plane is maximally in focus [15]. Unlike the DfS method the DfF method does not have to rely on a precise two camera configuration, instead only a single camera is required.

Depth estimation in a DfF system is performed by searching for a combination of lens/camera parameters that result in an object being in focus at a given distance from the camera/lens. This may be achieved by changing either the lens to sensor distance, the focal length or the object distance, or any combination thereof [5]. Figure 1.2 shows an example subset of images in a focal stack at various focal distances.



(a) Foreground Focus (b) Midground Focus (c) Background Focus

Fig. 1.2. Lego® man focal stack example. (a) The yellow space man is in focus; (b) the blue space man is in focus and (c) the black space man is in focus.

This method has several drawbacks, one of which is the requirement to generate multiple images to create the focal stack. This means that there is an increased requirement for image storage since each scene will require several images to determine the depth of an object in a scene. And the depth resolution is directly related to the number of images taken. For example if only 25 images are taken then the maximum number of depth levels that can be differentiated is 25. If the images in the focal stack are taken at equidistant focal distances then the depth resolution is the focal distance step size. However, there is nothing to limit the focal stack creation to a uniform step. In fact the step can be of arbitrary distance only limited by the imaging system physical properties. Depending on scene complexity equidistant or even varied focal distance step sizes may not provide enough depth levels to accurately represent the scene.

Similarly to the DfS methods, Gaganov and Ignatenko have successfully applied MRFs to develop the depth maps from a focal stack [16]. A variational approach has been proposed by [15] in which an energy term consisting of a combination of a data fidelity term and a regularization term are minimized to create the desired depth map. In addition to more traditional statistical methods, deep learning and CNN's have been used for DfF by Hazirbas, et al. [13] showing a lot of promise. In fact their work takes advantage of Ronneberger, et al. [17] whose work with deep neural networks and semantic segmentation led to the development of the U-Net, which is also the basis for this this research.

1.2.3 Depth From Defocus

Depth from Defocus (DfD) is inspired by Pentland's research into the imperfections of biological lens systems [18], in this case the HVS. This research led to the first significant advancement in depth estimation by using the blur of the scene. Much like DfF, the DfD method is again an ill-posed problem [15]. The DfD method also relies only on a single camera imaging system. DfD differs from the DfF method, in that instead of taking several images at various focal distances the DfD method uses only very few images [15]. In fact, only a minimum of two images per scene are required to accurately generate a depth map. The DfD method takes one image considered to be in-focus and a second image considered to be out-of-focus image. In place of creating a focal stack as in the DfF method, the DfD method instead creates a synthetic stack by using the in-focus image and blurring it using N different blur kernels, effectively creating an N-level stack. This has the distinct advantage over the DfF method by not needing to take more image data if the focus stack was not large enough to accurately model the depth of the objects in the scene. Since control of this synthetic stack is entirely dependent on the blur kernel we have a lot of control over the depth estimation resolution. In the case of a Gaussian blur kernel, changing the σ value in a linear fashion would generate a stack that would be equivalent to

the DfF method with an equidistant focal distance step. However, a non-linear step in σ values would also be a potential method to create a suite of blur kernels. DfD also does not suffer from the same correspondence issues as does the DfS methods, because it is a single camera.

Schechner and Kiryati [5] introduced a point that, while DfS does suffer from the problem of occlusions, the DfD method also suffers from the same problem, only to a lesser degree. If the occluded part is small compared to the support of the blur-kernel, and its depth is close to that of the occluding object, the resulting error will be small [5].

As with the DfS and DfF methods, traditional and some more novel statistical methods have been applied to solving the ill-posed DfD problem. Watanbe and Nayar developed a novel algorithm that used a class of broadband operators that when combined produces accurate depth maps of a given scene [19]. Crofts developed an interesting approach of depth estimation by developing a 4-D lookup table that could be applied to planar surfaces in the images to infer the depth [20]. Liu experimented with MRFs and graph cuts methods to develop accurate depth maps [21]. And recently Pasinetti, et al. used the intensity contrast method (ICM) and the differential contrast method (DCM) to generate depth maps [22]. The DfD problem is just now being pushed in the realm of deep learning. Zhang, et al. have proposed a hybrid deep learning network architecture that uses both the in-focus/out-of-focus image pairs and the left/right stereo image pairs to generate the depth map [23].

This thesis presents a new DfD depth inference method that uses only an in-focus and out-of-focus image pair in conjunction with a new deep learning architecture. The novel aspects of this research are; 1) the development of a deep learning architecture that improves overall performance and inference speed as compared to the current stat-of-the-art methods and 2) the application of this deep learning architecture to data collected with a camera/microfluidic lens combination with ground truth data collected with a 2-D LIDAR.

Chapter 2 provides the theoretical background for the geometric optics which governs the physics behind the depth estimations methods, including the DfD method. Chapter 3 describes the datasets used in this research and discusses the error metrics that will be used to evaluate the new DfD deep learning architecture against the current state of the art graph cuts algorithm. Chapter 4 describes the current state of the art method using a graph cuts algorithm and the algorithm's performance on the datasets outlined in Chapter 3. Chapter 5 describes the new improvements to the state of the art using the DfD deep learning architecture, and presents the experimental results. Chapter 6 introduces two methods geared towards optimizing the performance of the DfD-Net, one method designed to improve the results of the performance metrics and the other method designed to reduce the complexity/increase in processing speed of the network. Chapter 7 discusses the application of the DfD-Net to images that were blurred using a microfluidic lens. Finally Chapter 8 concludes the discussion and Chapter 9 contains the recommendation for future research.

2. GEOMETRICAL OPTICS

To understand the underlying principles of depth estimation methods, we first need to gain an understanding of the physical nature of a lens. For the purposes of this analysis we will only consider a single lens system that employs a thin convex lens. What is meant by thin is that the refraction at each of the lens interfaces is neglected. Figure 2.1 shows the path of the light rays through a thick lens as they pass through various thicknesses of the lens. Figure 2.2 shows the thin lens assumption, in which the light rays only refract once at the center of the lens. In essence the thickness of the lens is neglected and the light ray refractions is assumed to take place at the vertical axis (center) of the lens [24].

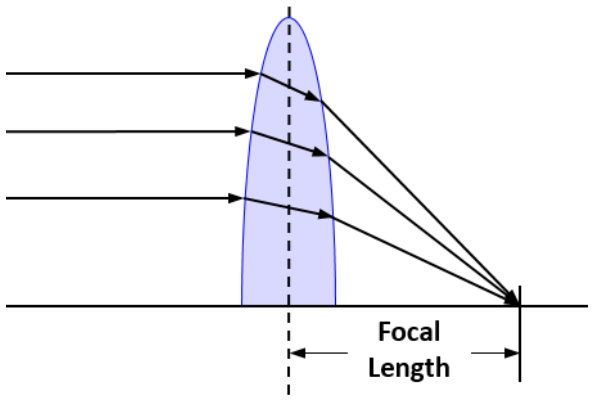


Fig. 2.1. Representative Thick Lens Light Ray Travel

A secondary assumption about the optical system is that the lens is symmetric, both axially along the vertical axis and rotationally symmetric about the principle axis.

2.1 Thin Lens

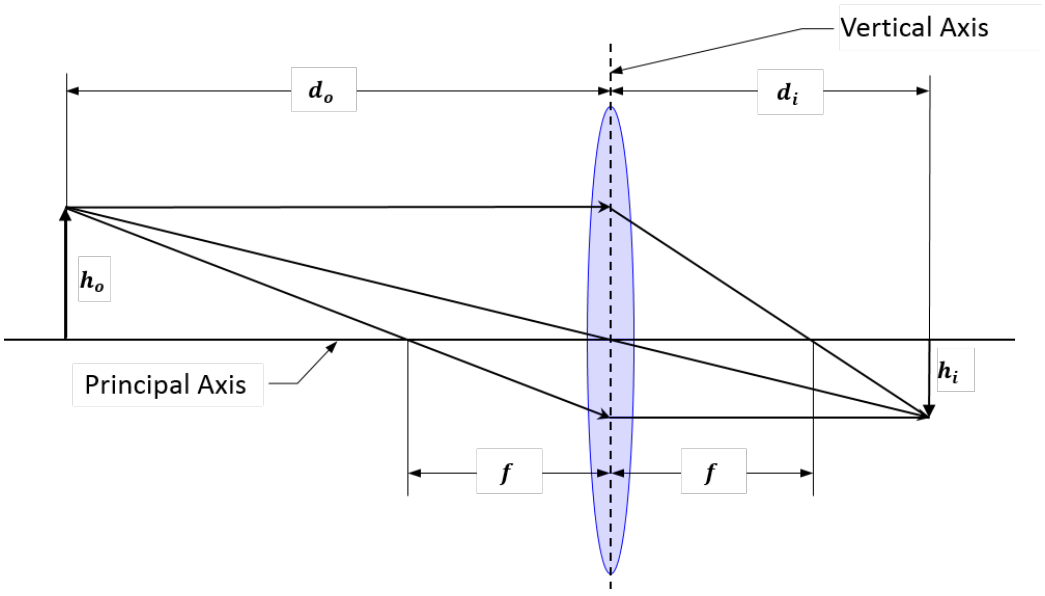


Fig. 2.2. Representative Thin Lens Geometry Setup [24, 25]

To develop the governing equations for a thin lens model, Figure 2.2 is used. This figure sets up the basic geometries to develop the equations which govern the interactions between the focal length (f), the distance an object is away from the lens (d_o) and the distance of the image of the object from the lens (d_i). h_o represents the natural height of the object and h_i represents the height of the image of the object. It should be noted that the image of the object is shown on the opposite side of the lens, however, the image can also be on the same side of the lens as the object. The image is then considered to be imaginary, but the governing mathematical equations still hold. Without loss of generality, only the case where the object image is real is considered for this analysis.

Figure 2.3 expands upon the geometries to emphasize the similar triangles that are used to develop the thin lens equation. The triangle which contains the lines d_o and A is similar to the triangle which contains the lines d_i and B. Using the properties of similar triangles the ratios of the line segment d_o and d_i can be related to the

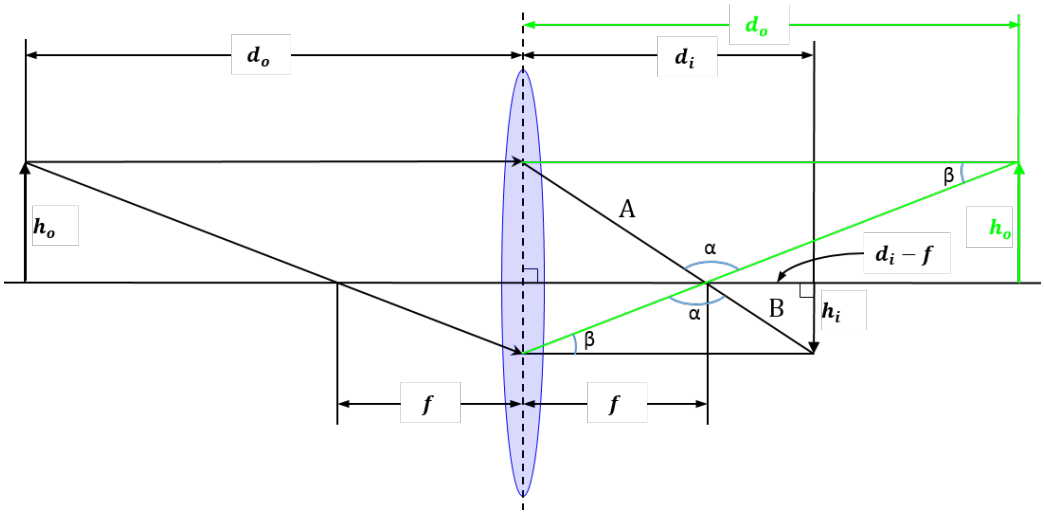


Fig. 2.3. Extended View of the Thin Lens Geometries with Additional Similar Triangles Added [24, 25]

line segments A and B. In addition the line segments f and $d_i - f$ are also similarly related to the ratio of A and B. Equation 2.1 defines these relationships.

$$\frac{d_o}{d_i} = \frac{A}{B} = \frac{f}{(d_i - f)} \tag{2.1}$$

Rearranging Equation 2.1 yields the following:

$$d_o(d_i - f) = d_o d_i - d_o f = f d_i$$

$$d_o d_i = f(d_o + d_i)$$

Finally gathering like terms the thin lens equation is shown in Equation 2.2.

$$\frac{1}{f} = \frac{d_o + d_i}{d_o d_i} = \frac{1}{d_i} + \frac{1}{d_o} \tag{2.2}$$

Using the similar triangle relationship, a formula for the ratio of the object height to the image height can be related to the object distance from the lens and the image distance from the lens (Equation 2.3).

$$\frac{d_o}{d_i} = \frac{A}{B} = \frac{h_o}{h_i} \tag{2.3}$$

Rearranging the equation in terms of the image height results in Equation 2.4. This equation is also known as the magnification equation for the lens.

$$h_i = h_o \frac{d_i}{d_o} \tag{2.4}$$

Equations 2.2 and 2.4 will be the base equations used for further derivations in this chapter.

2.2 Circle of Confusion

The circle of confusion (CoC), also known as the blur diameter or point spread function of a lens, can be determined in a similar manner as the thin lens equation itself. Figure 2.4 shows a similar geometric configuration as Figure 2.2. However, instead of an object with a given height, a single point placed on the principal axis of the lens is used for the analysis and equation development. Here, d_o represents the distance at which the point is perfectly in focus, (CoC is equal to zero) and d_f is the far distance at which the image of the point is blurred with some diameter c. Similarly d_n is the near distance at which the image of the same point produces the same diameter c blur. As before, d_i is the distance from the lens to the image plane.

Two separate equations must be developed to understand the circle of confusion. First, using the far point denoted by d_f and using the similar triangle relationship between the triangle with a base length of D and a height of d_f and the triangle with a base of C and a height of $d_f - d_o$ Equation 2.5 can be established. For the purposes

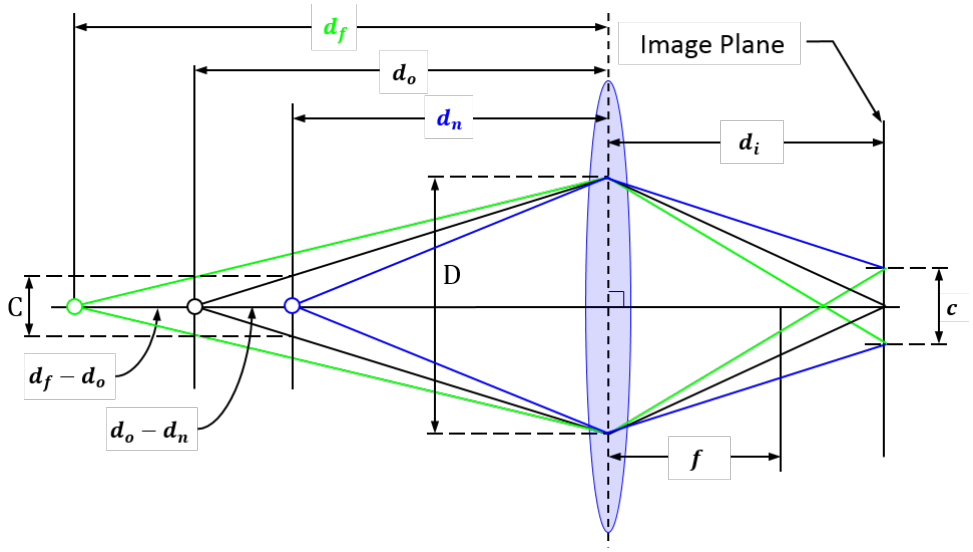


Fig. 2.4. Circle of Confusion Geometry Configuration [25]

of this analysis it is assumed that d_f will always be larger than d_o, i.e. the out of focus object will always be further away from the lens as compared to the in focus object.

$$\frac{D}{C} = \frac{d_f}{(d_f - d_o)} \tag{2.5}$$

Rearranging Equation 2.5 and solving for C yields the following relationship:

$$C = D\frac{d_f - d_o}{d_f} \tag{2.6}$$

It should be noted that this equation would also be the same equation if we were to use the radius (D/2) and (C/2) since we are using the ratio of the two. For this reason the term blur radius will be used instead of blur diameter. Using the relationship developed in Equation 2.4 and substituting in the appropriate variables Equation 2.4 now becomes:

$$c = C\frac{d_i}{d_o} \tag{2.7}$$

Substituting the results of Equation 2.6 into Equation 2.7 results in the following:

$$c_{far} = D\frac{(d_f - d_o)}{d_f}\frac{d_i}{d_o} = Dd_i\left(\frac{1}{d_o} - \frac{1}{d_f}\right) \tag{2.8}$$

What is left to solve for is d_i. Rearranging the thin lens equation (Equation 2.2) to solve for d_i produces the following equation:

$$\frac{1}{f} = \frac{1}{d_o} - \frac{1}{d_f} \Longrightarrow d_i = \frac{fd_o}{(d_o - f)} \tag{2.9}$$

Substituting Equation 2.9 into Equation 2.8 generates the following relationship between the circle of confusion radius and the given lens parameters.

$$c_{far} = D\left(\frac{d_f - d_o}{d_f}\right)\frac{d_i}{d_o} = D\frac{fd_o}{d_o - f}\left(\frac{1}{d_o} - \frac{1}{d_f}\right) \tag{2.10}$$

To further simplify this equation, the lens' f-number (n) which is the ratio of the lens focal length f and the lens aperture (in this case D) can be substituted into Equation 2.10. This results in the final equation for the circle of confusion radius:

$$c_{far} = \frac{d_o f^2}{n(d_o - f)}\left(\frac{1}{d_o} - \frac{1}{d_f}\right) \tag{2.11}$$

The maximum circle of confusion radius for a given lens can be determined by setting d_f equal to inf. This reduces Equation 2.11 to the following:

$$c_{max} = \frac{f^2}{n(d_o - f)} \tag{2.12}$$

The same process can be used to develop an equation for the CoC for the near point denoted by d_n. Using the similar triangle relationship between, C, D, d_o and d_n Equation 2.13 is created.

$$\frac{D}{C} = \frac{d_o}{d_o - d_n} \tag{2.13}$$

The relationship between c and C for the near point is defined as:

$$c = C\frac{d_i}{d_n} \tag{2.14}$$

Rearranging Equation 2.13 in terms of C and substituting into Equation 2.14 yields Equation 2.15.

$$c_{near} = D\left(\frac{d_o - d_n}{d_o}\right)\frac{d_i}{d_n} = Dd_i\left(\frac{1}{d_n} - \frac{1}{d_o}\right) \tag{2.15}$$

Once again using the rearranged version of the thin lens Equation (2.9), substituting in for d_i and using the relationship between the f-number and the lens aperture (D), Equation 2.15 now becomes the following:

$$c_{near} = \frac{d_o f^2}{n(d_o - f)}\left(\frac{1}{d_n} - \frac{1}{d_o}\right) \tag{2.16}$$

While an equation for the maximum CoC can be derived from the definition of c_{far}, the same cannot be said for the derivation of a similar formula using the definition of c_{near}. This is because as the point gets closer to the lens the $1/d_n$ term grows and is unbounded. Therefore the maximum circle of confusion for the near point is limited by the size of the image plane or digital sensor used to capture the image plane. Using Equations 2.11, 2.12, 2.16 and the parameters outlined in Table 2.1 a range of values for the blur radius and the quantized blur radius can be calculated.

Table 2.1.
Example Lens/Camera Imager Parameters

	Parameter	**Value**
Lens	F-Number (n)	3.7
	Focal Length (f)	9.6 mm
	Focus Distance (d_o)	0.5 m
Imager	Pixel Size	4.8 x 4.8 μm

Figure 2.5 shows the plot of the blur radius versus the distance from the lens for each of the piecewise functions. The blue line represents the blur radius based on the functions defining c_{near} and c_{far}. It can be seen that the c_{far} curve is monotonically increasing, asymptotically approaching c_{max} (green line) for objects in the far region and the c_{near} curve is also monotonically increasing and unbounded.

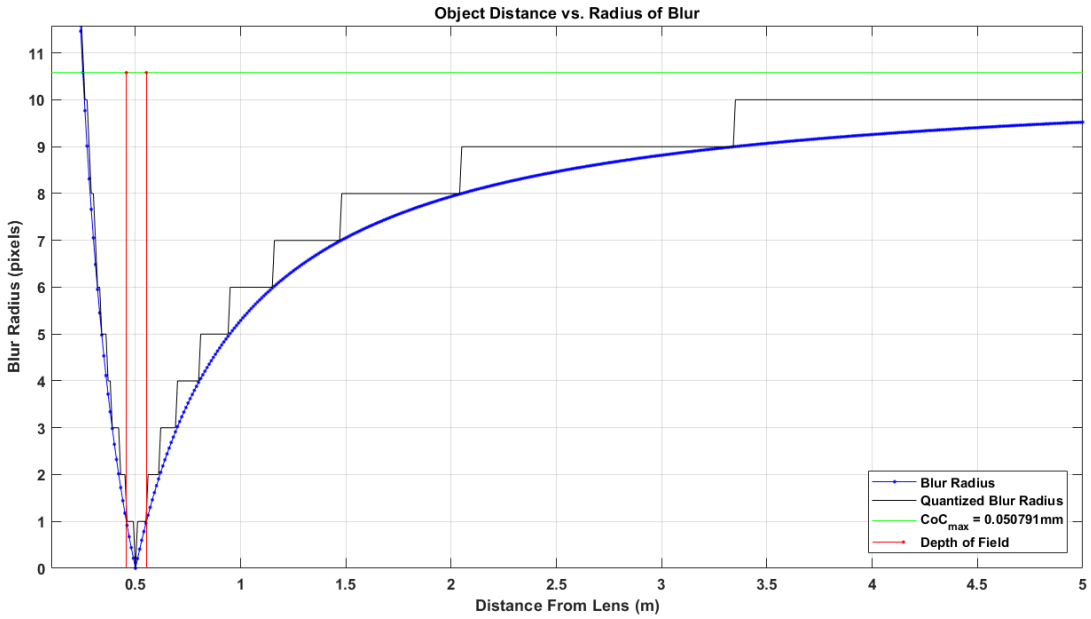


Fig. 2.5. Object Distance vs. Blur Radius Chart

In a digital system light is captured by an imaging sensor, typically a Complementary Metal Oxide Semiconductor (CMOS) or a Charge-Coupled Device (CCD) sensor. These sensors quantize the amount of light received and this quantizing unit is known as a pixel. The stepped black line in Figure 2.5 shows the quantized version of blur radius using the same parameters listed in Table 2.1. It can be seen that the further away from the lens an object gets the more difficult it becomes to distinguish a distance from the lens based solely on the blur radius. Because digital imaging sensors are not binary quantizers, each pixel offers some additional blur radius discrimination by the way of changing intensity values. However, this is highly sensor dependent and could vary based on a number of factors, including sensor pixel size, pixel sensitivity, etc.

2.3 Depth of Field

The depth of field (DoF) is a term that denotes the range of distances both in front of the focus distance and behind the focus distance where the CoC does not change. The relationship between the DoF and the CoC is based on the lens and image plane geometries. In addition, the DoF is also directly dependent on the sensor (i.e. 35mm film, CCD imager) and what is termed an acceptable level of blur. In a digital camera system this acceptable level is typically when the circle of confusion radius is less than or equal to the physical size of one pixel on the imager. To determine the farthest distance an object can be while still meeting the acceptable blur criteria Equation 2.11 is rearranged to solve for d_f.

$$c_{far} = \frac{d_o f^2}{n(d_o - f)} \left(\frac{1}{d_o} - \frac{1}{d_f} \right) \Longrightarrow d_f = \frac{d_o f^2}{f^2 - nc(d_o - f)} \tag{2.17}$$

When the term $nc(d_o - f) \geq f^2$ the far distance is considered to be ∞. The same rearranging process can be done for Equation 2.16 to solve for d_n.

$$c_{near} = \frac{d_o f^2}{n(d_o - f)} \left(\frac{1}{d_n} - \frac{1}{d_o} \right) \Longrightarrow d_n = \frac{d_o f^2}{f^2 + nc(d_o - f)} \tag{2.18}$$

Both the near and far discriminators have been dropped from the equations because when determining the DoF the CoC radius is the same for both the near and far distances. The depth of field is then simply the difference between the far distance d_f and the near distance d_n:

$$DoF = d_f - d_n \tag{2.19}$$

2.4 Depth Resolution

Using Figure 2.5 as a reference it can be quickly seen that the quantizing nature of the pixel and the non-linear curve describing the far object blur radius leads to zones where the depth of an object will be indistinguishable based on the blur. This leads to three distinct scenarios that need to be considered when attempting to differentiate the distances of objects in a scene using only the blur information.

1. A change in focus distance does not cause a change in the object's blur radius.
2. A change in focus distance causes an object's blur radius to change and the object remains on the same side of the inflection point.
3. A change in focus distance causes an object's blur radius to change but the object moves from one side of the inflection point to the other.

The first scenario simply means that the object was either too far from the initial focus point or the change in focus distance was so small that the change was not enough to register in another pixel. In this case it will not be possible to determine the depth of the object from the blur radius change. The second scenario is the optimal scenario. The change in blur radius will allow algorithms to determine the depth based on the increase or decrease in blur radius. The third scenario is the most

difficult because in the real world we may not know if this situation has happened. Algorithms will be able to determine the depth, but depending on the algorithm it may not accurately determine the correct depth.

In order to be able to differentiate objects at various distances based on their change in blur, the change must be large enough to be detectable in an imaging sensor. In the digital realm this means that the blur difference between two objects must be greater than or equal to one quantizing unit (pixel). Using Equation 2.11 a relationship between the CoC for two separate objects can be developed in terms of a pixel.

$$|c_1 - c_2| = \frac{d_o f^2}{n(d_o - f)} \left| \left(\frac{1}{d_2} - \frac{1}{d_1} \right) \right| \tag{2.20}$$

Equation 2.20 is the absolute difference in CoC between two objects located in the far field, where c_1 is the CoC for one object and c_2 is the CoC for a second object. This does not provide enough information in order to determine if the two objects will be in different blur zones. In order to garner this last bit of information the pixel size itself must be used and, as stated before, the absolute difference between the CoC for each object must be greater than or equal to one.

$$\frac{|c_1 - c_2|}{pixel\_size} = \frac{d_o f^2}{n(d_o - f)} \left| \left(\frac{1}{d_2} - \frac{1}{d_1} \right) \right| \left(\frac{1}{pixel\_size} \right) \geq 1 \tag{2.21}$$

2.5 Summary

This chapter introduced the physical geometric configuration of a thin, single, convex lens system and the underlying mathematical equations that govern its behavior. This chapter also discussed the circle of confusion and how the blur radius can be determined based on the object distance from the focus distance. In addition the depth of field and depth resolution were discussed. A description and analysis of the datasets that were used in this research will be discussed in the next chapter.

3. DATASETS

This chapter will introduce the datasets used and the methodology used to create a blurred version of the dataset. This chapter will also discuss the metrics used to evaluate the performance of the depth inference methods describe in Chapters 4 and 5 and to ultimately provide a means of comparing the two methods.

3.1 Synthetically Blurred Dataset

The synthetically blurred dataset consists of the third size images from the Middlebury College Stereo Datasets from 2005 and 2006 [7, 8]. Originally these datasets were intended to be used to develop and test DfS algorithms, however based on previous depth from defocus work [21] this dataset was re-purposed to be used in this research.

The images in this dataset consist of various scenes using common household objects with various coloring and textures. The images were taken at three different illumination levels and three different exposure times per illumination level. Each scene was captured from two separate camera viewpoints (left and right). Figure 3.1 shows an example of the illumination and exposure levels for the aloe plant dataset. These lighting conditions allow the machine learning algorithms to be exposed to a variety of possible lighting conditions that could be seen in the wild. This wide range of illumination and exposure levels allows the models to generalize to work in various lighting conditions.

Figure 3.2 shows the exposure times for each of the three illuminations. The red bars represent the exposure level 0 times, the green bars represent the exposure level 1 times and the blue bars represent the exposure level 2 times. From these graphs it can be seen that the exposure times were not uniform across the dataset for any of

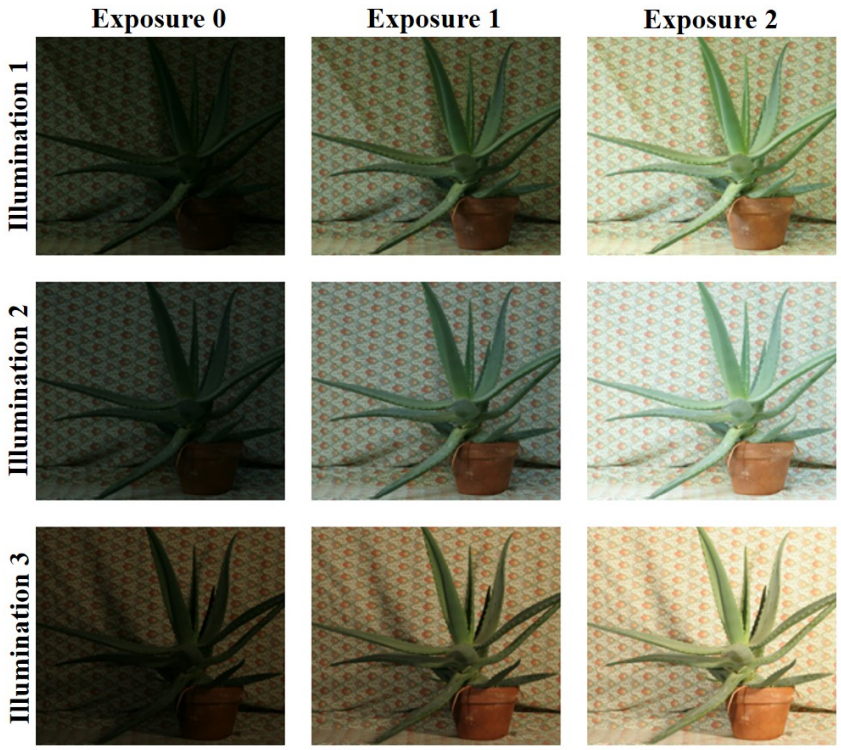


Fig. 3.1. Middlebury College Stereo Aloe Plant Illumination and Exposure Level Example

the illumination settings. It will be shown in Chapter 4 that these illumination and exposure levels have a direct effect on the performance of the current state of the art algorithm and that the proposed deep learning architecture presented in Chapter 5 is superior.

In addition to the visual color views of the scenes, these datasets also contain the ground truth depth map for each scene. The ground truth data was collected using the structured light method. Structured light is the "active illumination of the scene with a specially designed 2-D spatially varying intensity pattern" [26]. The pattern is generated in a horizontal orientation and then again in a vertical orientation.

Structured light works by taking advantage of the fact that if the surface of an object is flat there will be no distortions of the light pattern. However if the object surface is not flat, then the geometry of the surface will produce distortions in the

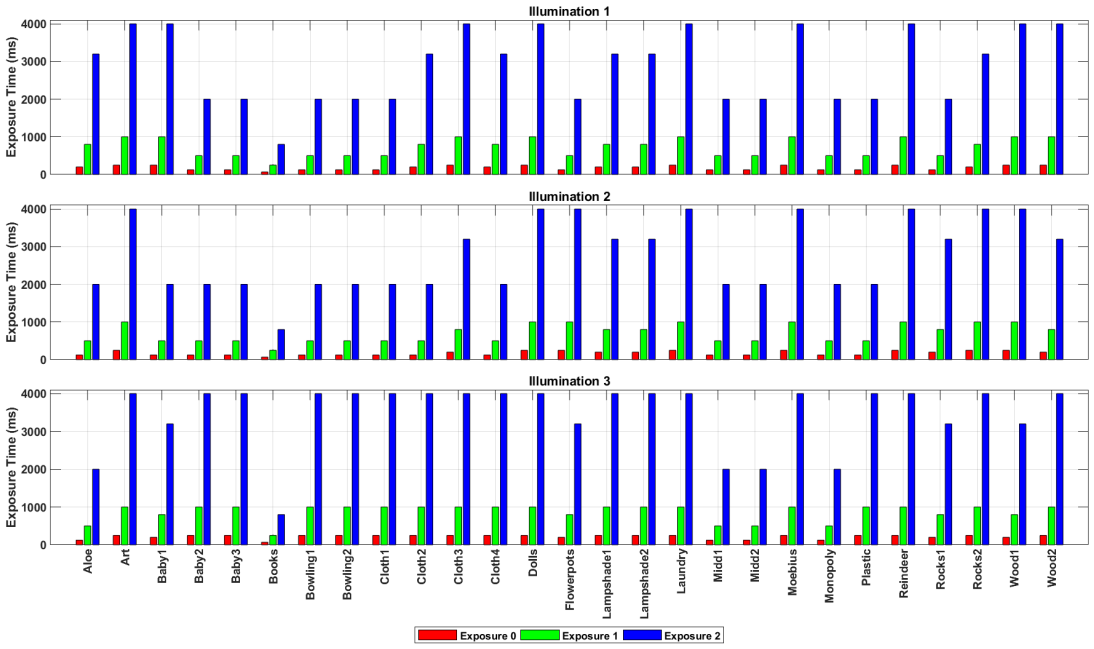


Fig. 3.2. Middlebury College Stereo Vision Dataset Exposure Time Charts

light pattern. The depth can then be determined based on the relationship between the original structured light pattern and the observed light pattern. There are many different possible patterns that can be used to determine the depth. Scharstein, et al. used a 10-bit Gray-code sequence and its inverse [27] as the desired pattern. The Middlebury researchers used a liquid crystal display (LCD) projector to project the Gray-code, both horizontal and vertical, onto the objects in the scene. They were then able to produce the required depth maps.

In order to create the synthetically blurred images, the corresponding depth map is used in the following manner: First a series of blur kernels based on a Gaussian blur filter is created using Equation 3.1. Where m is an odd number and is the size of the blur kernel, and x and y are the position in the kernel.

$$k_i(x,y) = \frac{1}{\sqrt{2\pi}\sigma_i} e^{\frac{-(x-\lfloor \frac{m}{2} \rfloor)^2 (y-\lfloor \frac{m}{2} \rfloor)^2}{2\sigma_i^2}} \quad x, y \, \epsilon \, \{0 \ldots m-1\} \tag{3.1}$$

For each depth class (i) a new blur kernel is generated based on a desired sigma value. The sigma value is determined based on the number of classes in the problem and the range of maximum and minimum sigma values. The sigma value is calculated using the following equations:

$$\sigma_{step} = \frac{\sigma_r}{N} \tag{3.2}$$

$$\sigma_i = \begin{cases} \sigma_{min} + i\sigma_{step} & \sigma_{min} \neq 0 \\ \sigma_{min}(i+1) & \sigma_{min} = 0 \end{cases} \quad i \,\epsilon\, \{0 \ldots N-1\} \tag{3.3}$$

Where σ_r is the maximum desired change in sigma value, σ_{min} is the minimum allowable sigma value and the maximum sigma value is related to σ_{min} and σ_r in the following manner: $\sigma_{max} = \sigma_r + \sigma_{min}$. Initially, Liu [21] had used a minimum sigma value of zero, however it was discovered that the difference in blur kernels was not changing until sigma reached a value of approximately 0.24.

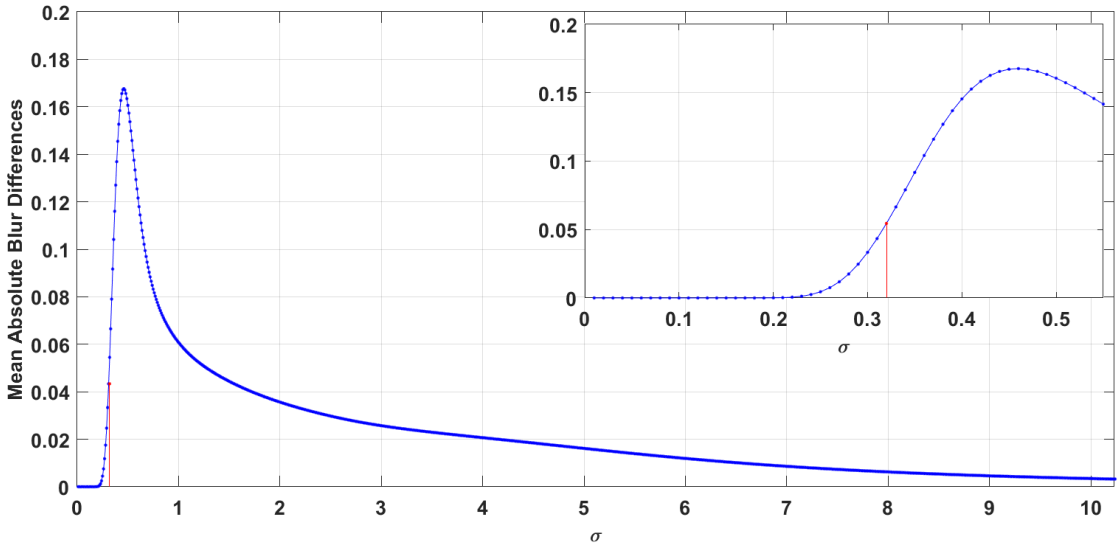
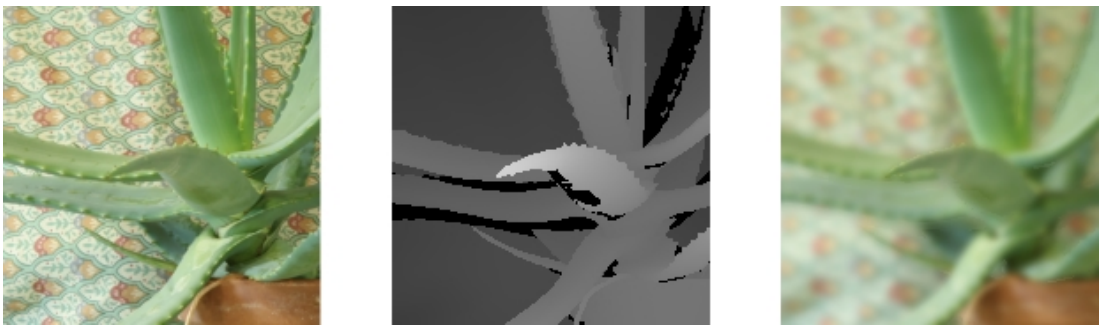


Fig. 3.3. Sum of the Absolute Differences Between Blur Kernels

Figure 3.3 shows the sum of the absolute difference of the blur kernels for each sigma value ranging from zero to ten. It can be seen that there is a minimal difference between the kernels with the smaller values of sigma. Small σ_r values and a large number of classes mean that there will be several class levels that result in a blur kernel that will produce the same results when convolved with the in-focus image. Which means that several classes could become indistinguishable from each other in terms of the error between the synthetically blurred images and the out of focus image. For this reason, a σ_{min} value of 0.32 was chosen to ensure that each blur kernel would be unique. To produce the synthetic stack, the in-focus image, f, in Equation 3.4 is convolved with the kernel from Equation 3.1.

$$b_i = k_i * f \tag{3.4}$$

The depth map value at a given pixel location is used as the index of the blurred set of images, and the resulting blurred pixel value is used in the synthetically blurred image. For example if there were 256 possible depth map levels then there would be 256 synthetically blurred versions of the original image and if a depth map value of 200 was at a given pixel location (x, y) then the 200th blurred image would be selected and the value of the pixel at (x, y) would be copied into the synthetically blurred image at pixel location (x, y).



(a) Original Image (b) Depth Map (c) Blurred Image

Fig. 3.4. Synthetic Blurring Example on Middlebury College Aloe Image [7, 8]

Figure 3.4 shows a representative example of the in focus image, the depth map and the final synthetically blurred image as applied to the Middlebury College Aloe image [7, 8]. It can be seen that the higher valued (brighter) depth map values result in less blurring, while the lower valued (darker) depth map values produce more blurring as expected.

As with any natural scene the distribution of the depth map values is not uniformly distributed across the range of [0 ... 255]. Figure 3.5 shows the combined distribution for the entire Middlebury Stereo Vision dataset [7, 8]. This dataset is lacking depth values at each end of the distribution. While the current state of the art graph cuts method does not have a dependence on the distribution of the depth map values, when considering some deep learning algorithms, the distribution of the depth map values can become very important. This is true when the training set does not have a particular depth map value, but the test set does. It would be unreasonable to expect the algorithm to perform well in the absence of particular training examples.

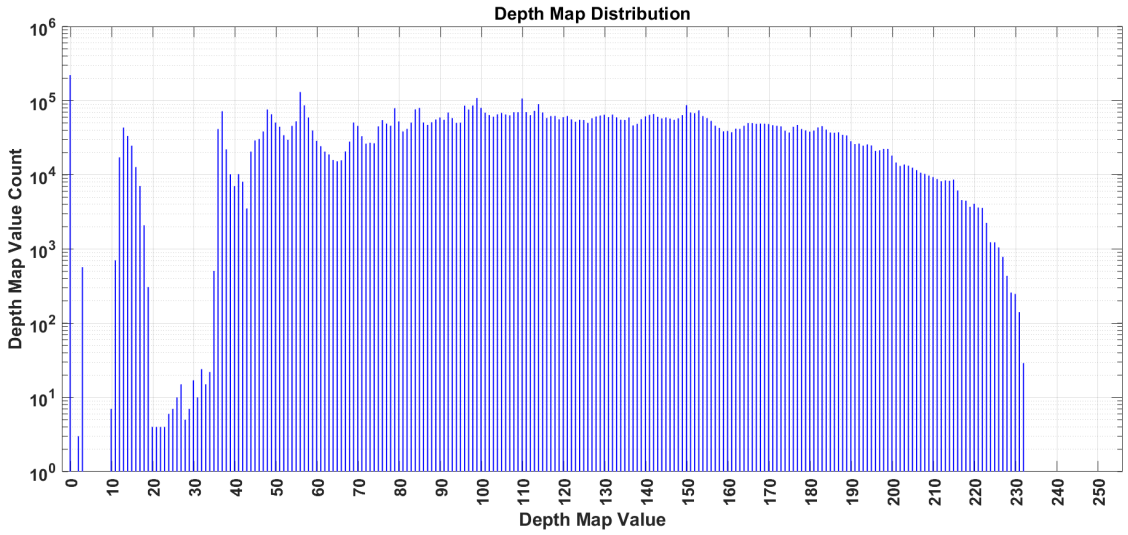


Fig. 3.5. Middlebury College Dataset Overall Depth Map Distribution

3.2 Real World Dataset

The following section describes the hardware and the procedures used to collect the real world camera dataset. This section also outlines the data processing steps used to match the imagery data with the depth ground truth data.

3.2.1 Camera & Microfludic Lens

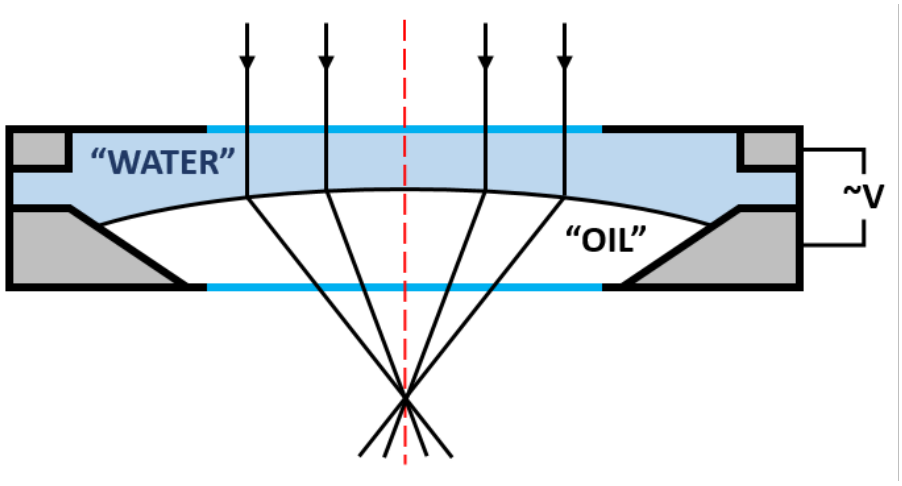


Fig. 3.6. Example Microfluidic Lens Cross-Section

The camera that was used for this data collect was a FLIR® (formerly Point Grey Research) Chameleon3 1.3 MP Color USB3 vision camera (CM3-U3-13Y3C-CS) coupled with a Varioptic® Caspian M12-316-9.6 liquid lens. Figure 3.6 shows an example cross-section of a typical microfluidic lens. The liquid lens uses a property known as electrowetting, which occurs when "a drop of insulating liquid (e.g. oil drop) is deposited on a flat surface, made of a conductive material covered with an insulating and hydrophobic layer, and then both the drop and surface are immersed in a conductive liquid (e.g. electrolyte). Voltage is then applied between the conductive substrate and the conductive liquid causing the liquid drop to change shape" [28]. The liquid lens requires a high voltage driver that was designed in house. The voltage driver produces a voltage that ranges from 9.8V to 62.075V in 256 discrete steps,

where each step represents a voltage change of approximately 0.205V. The lens driver is operated in an open loop control manner which means that there is no feedback to the voltage driver to ensure that the lens focus point remains stable. The lens was configured according to the Varioptic® documentation with the focus point set to 1 meter. The camera and lens properties are outlined in Table 3.1.

Table 3.1.
Microfluidic Lens/Camera Imager Specifications

Parameter	**Value**
Lens	
Effective Focal Length	9.6 mm
F-Number	3.7
Image Circle Diameter	9.1 mm
Camera	
Resolution (h x w)	1024 x 1280, 1.3 MP
Frame Rate	149 FPS
Sensor	On Semi P1300, CMOS 1/2”
Readout Method	Global Shutter
Pixel Size	4.8 x 4.8 μm
ADC	10-bit

3.2.2 LIDAR

Like the Middlebury College dataset, ground truth labels are required in order to train and evaluate the performance of the depth estimation algorithms on real world datasets. The ground truth data was collected using an Ouster OS-1 64-beam 2-D LIDAR. The LIDAR was configured in the high resolution mode with a scan time of 10 Hz and produces a panoramic 64 x 2048 point depth map. Table 3.2 lists the LIDAR specifications. The LIDAR sends the data back to a PC via a User Datagram Protocol

(UDP) Ethernet connection. Each data packet consists of 16 azimuth measurements with 64 range values per azimuth measurement. To get a complete 360 degree scan of the environment 128 UDP data packets are required.

Table 3.2.
OS-1 LIDAR Specifications

Parameter	**Value**
Beams	64
Resolution (h x w)	64 x 2048
Vertical Resolution	0.52 deg
Azimuth Resolution	0.18 deg
Rotation Rate	10 Hz
Range	0.5m - 120m

3.2.3 Real World Data Collection

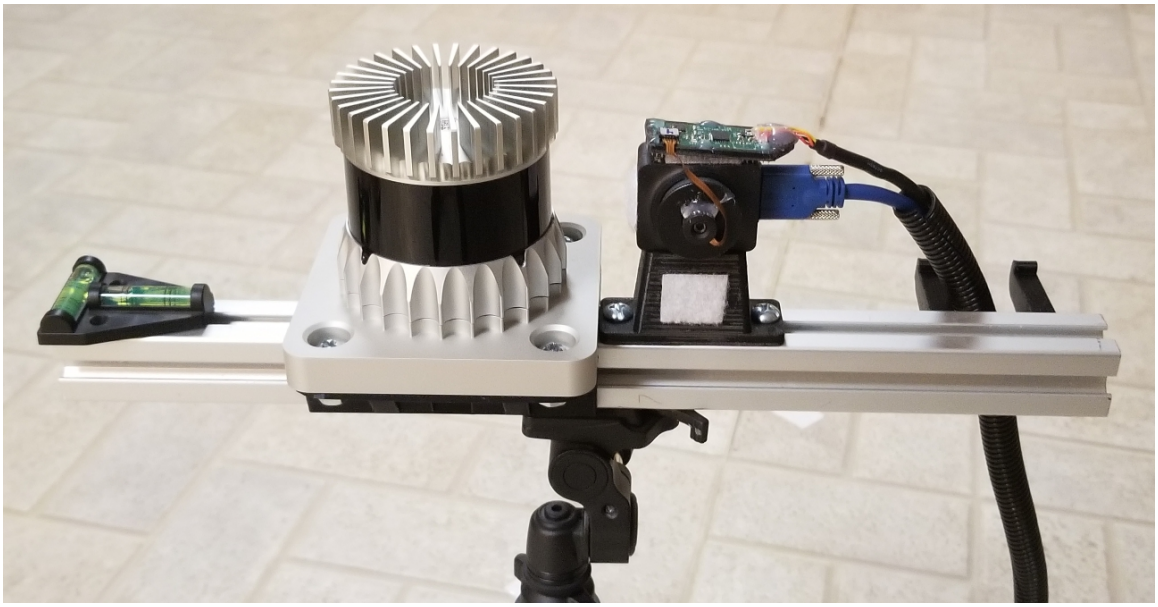


Fig. 3.7. Camera/LIDAR Capture Rig

The camera and LIDAR were mounted on a 30mm x 30mm piece of 80/20 extruded aluminum T-slot framing system using custom 3-D printed mounting brackets. The vertical optical axis of the camera was aligned as best as possible with the vertical centerline of the LIDAR. The horizontal optical axis of the camera and the horizontal axis of the LIDAR are separated by approximately 93mm. Figure 3.7 shows the configuration and placement of the camera and LIDAR mounted on a standard tripod.

The data was collected on various textured objects. These objects were textured using various patterns and materials of varying color. The scene was setup with various objects and surfaces that ranged between 1.5 and 2.5 meters from the camera. The camera was configured with an exposure time that was varied from 70 ms to 10 ms in 10 ms increments. This varied exposure time allows for the generation of images with varied lighting conditions that range from slightly under exposed to slightly over exposed. These lighting conditions expose the machine learning algorithms to a variety of possible lighting conditions that could be seen in the wild. For each exposure time the microfluidic lens was set to 17 different voltage steps ranging from 127 to 143 with a voltage step of 135 resulting in the most in-focus image. Table 3.3 lists the camera settings that were used to capture the data. For each combination of lens voltage step and exposure time, four images were taken and averaged on a per pixel basis to form the final output image. This was done to alleviate some of the noise which was inherent in the camera. A total of 64 scenes were collected.

For each scene a ground truth depth map was also captured using the Ouster OS-1 LIDAR. Five scans were taken of the scene and then averaged to attenuate the noise from the LIDAR. Figure 3.8 shows an example of the raw panoramic data collected from the LIDAR. The area of pixels in the top center third are what the camera is capturing. Figure 3.9 shows the combined depth distribution for the entire real world dataset.

Table 3.3.
Camera Data Capture Settings

Parameter	Value
Image Capture Size (h x w)	728 x 736
Offset (x, y)	272, 148
Brightness	4.00
Gain (dB)	8.00
Sharpness	2500
Frame Rate	5 FPS

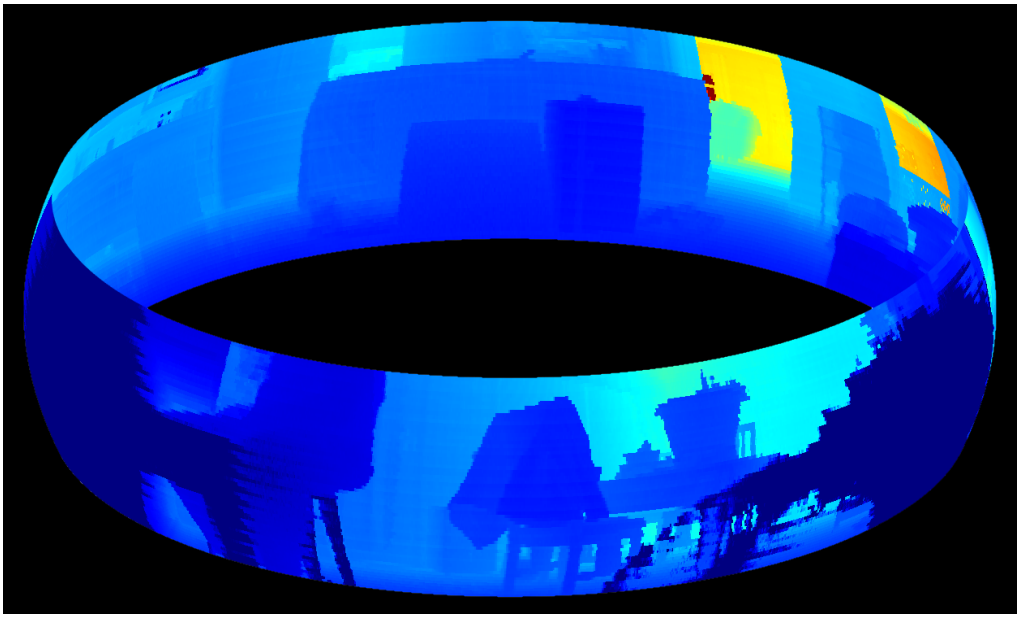


Fig. 3.8. Example LIDAR Panoramic Scene

3.2.4 Real World Scene Configuration

The following section will describe how each scene was setup for the real world data collect. Figure 3.10 shows the configuration of the scene for each of the first four datasets. The blue line represents a surface in which various textures and patterns were attached. The field of view represents what the camera and LIDAR will see

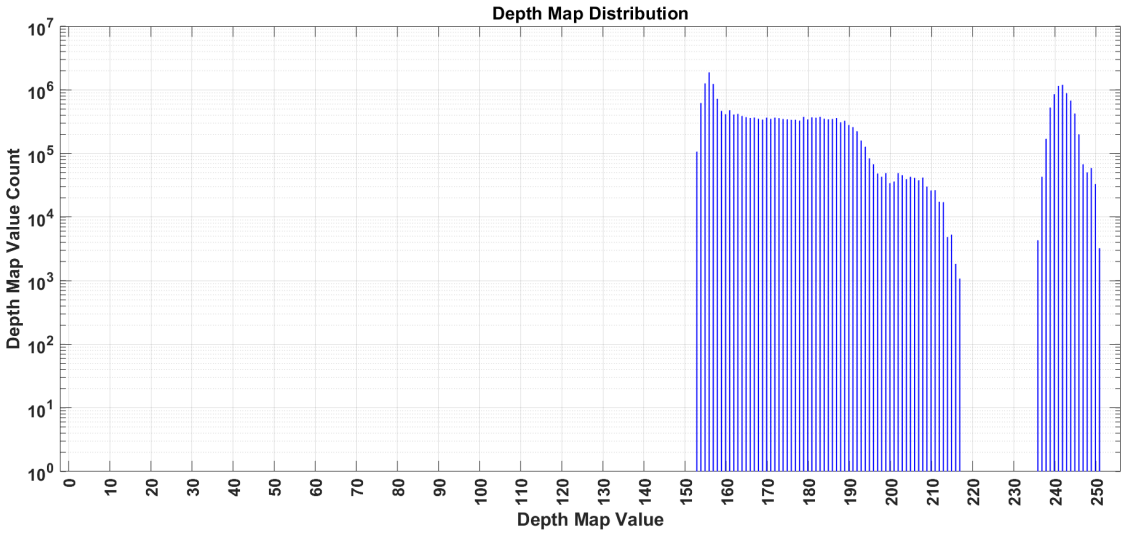


Fig. 3.9. Real World Dataset Depth Map Distribution

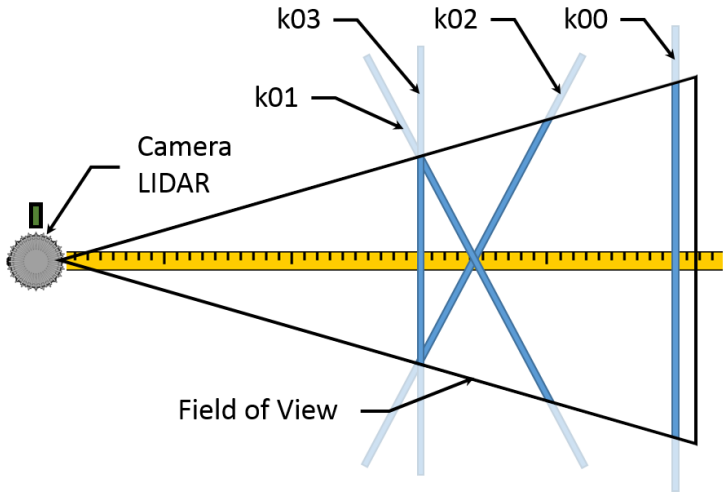


Fig. 3.10. Real World Dataset Scene Configuration Example

after the data has been post processed following the steps outlined in Section 3.2.5. For each of the four scenes the same pattern/texture was used and only one surface was used at a time. For example the k00 scene was a flat surface perpendicular to the camera and LIDAR and was approximately 2.5 meters from the capturing equipment. The k01 scene, took the same texture and rotated it so that the surface was oblique to the camera/LIDAR centerline with the closet side of the surface on the left and the

farthest side of the surface on the right. The k02 scene was identical to the k01 scene except the closet side to the camera and LIDAR was on the right and the farthest side was on the left. Based on the field of view the surface ranged between 1.5 and 2.2 meters from the camera and LIDAR. The k03 scene was setup similar to the k00 scene except the surface was set to 1.5m from the camera and LIDAR. The entire 4-tuple scene was constructed in such a way that the deep learning algorithm would see the same texture at multiple depths. Every fourth scene in the dataset is a duplicate in its geometry of the first four scenes, for example the k04 scene has the same setup as the k00 scene except the texture was changed. Similarly, the k05 scene is the same configuration as the k01 scene with the same texture as the k04 scene. This pattern was repeated for 16 different textures to give a total of 64 scenes for the dataset.

3.2.5 Real World Data Processing

Since the image data and the LIDAR data are of different scales/resolutions and spectrums a method of scaling and registering the two datasets was developed. The procedure listed below outlines the process to create a final dataset.

1. The first step in the process was to determine the sharpest image for a given exposure time. This was done by taking the 2-D Discrete Fourier Transform (using the 2-D FFT) of each image/voltage step within a given exposure time. The sum of the magnitude of the FFT was then taken. The sum of the energy is used as a measure of the amount of high frequency content within the image. An image with high frequency content indicates that edge transitions in the image are sharper which means the image is more in-focus. Similarly, a lower energy sum indicates that there is less high frequency content which means that the image is less in-focus. Once the in-focus image is found for each exposure time all of the images are cropped to 630 x 630 pixels (height x width).

2. The LIDAR data was converted to an XYZ Cartesian coordinate system from the native polar coordinate system. Figure 3.11 shows the X and Y axes for the

Cartesian coordinate reference frame. The Z axis is coming out of the image, creating a right-handed coordinate system. This conversion is accomplished using Equations 3.5-3.9. Where $meas\_ID$ (measurement ID) is an index to the LIDAR data. This index ranges between 0 and 2047 and is part of the raw data that is provided by the LIDAR system. The $beam\_az\_index$ is a correction factor to the measurement ID and is part of the LIDAR calibration data provided by the unit and is specific to a particular LIDAR system. For the LIDAR unit used to collect the data this quantity ranged from -19 to +19. The $beam\_alt\_angle$ is the altitude angle for each of the 64 laser beams emitted from the LIDAR. This quantity is also part of the calibration data and is specific to the LIDAR. These values ranged from -16.8294 to +16.4486. Finally, r is the raw LIDAR range data. Once the data was converted the X values were then used as the range from the LIDAR to surfaces in the scene. The data at this point is 64 x 2048 and is then cropped at +/- 150 points on the horizontal axis, centered at point 1024 (0 degrees in the LIDAR coordinate system).

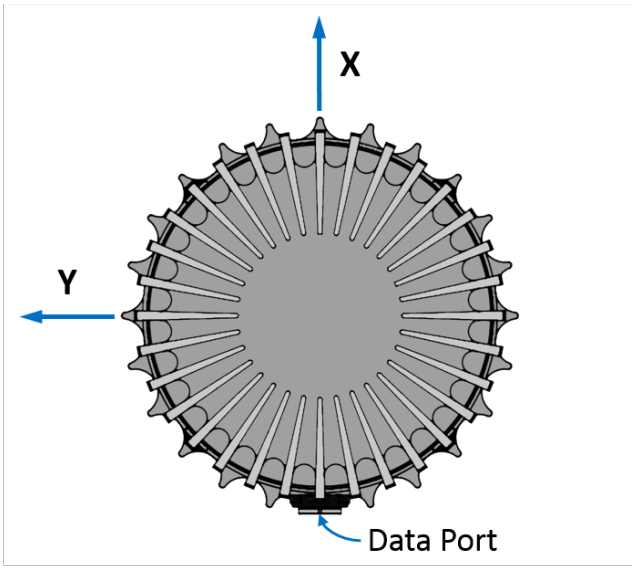


Fig. 3.11. LIDAR Coordinate Reference Frame

$$\theta = 2\pi \left(\frac{mod(meas\_ID + beam\_az\_index[i], 2048) - 1023.5}{2048} \right) \tag{3.5}$$

$$\phi = 2\pi \left(beam\_alt\_angle[i] \right) \tag{3.6}$$

$$X = rcos(\theta)cos(\phi) \tag{3.7}$$

$$Y = -rsin(\theta)cos(\phi) \tag{3.8}$$

$$Z = rsin(\phi) \tag{3.9}$$

3. Once the LIDAR data has been cropped it is filtered with a 7x1 and then a 1x7 median filter. A traditional 7x7 median filter was not used because, while in general a median filter will preserve edges while removing noise, the corners of objects in the data will tend to round with such a large filter.

4. The next step was to scale the LIDAR data values themselves. Because the LIDAR data is in units of millimeters and the scene was set to a maximum distance of 2.5 meters from the camera/LIDAR. The LIDAR values were divided by 10 and floored to bring the distance measurements into the range of 0 to 255.

5. Next, because the LIDAR data is 64x300 it has to be scaled up to match the image size. Because the scales in each dimension are vastly different, the LIDAR data was scaled up in a recursive manner using nearest neighbor interpolation. After each time the ground truth data was upsampled, the data went through a 13x1 and a 1x13 median filter. Figure 3.12 shows an example of the recursive scaling for each of the six steps used in the upsampling process starting with the input size of 64x300.

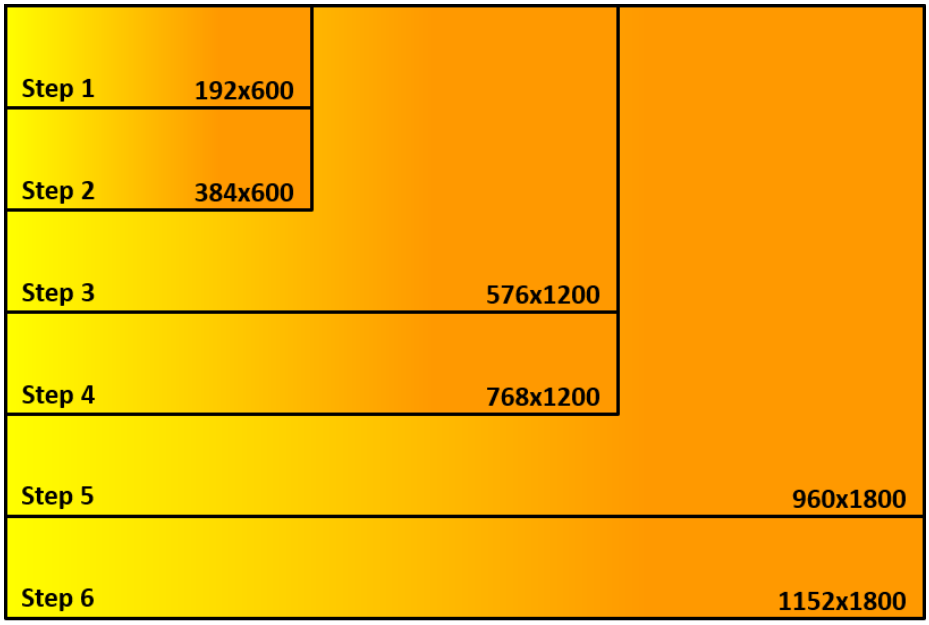
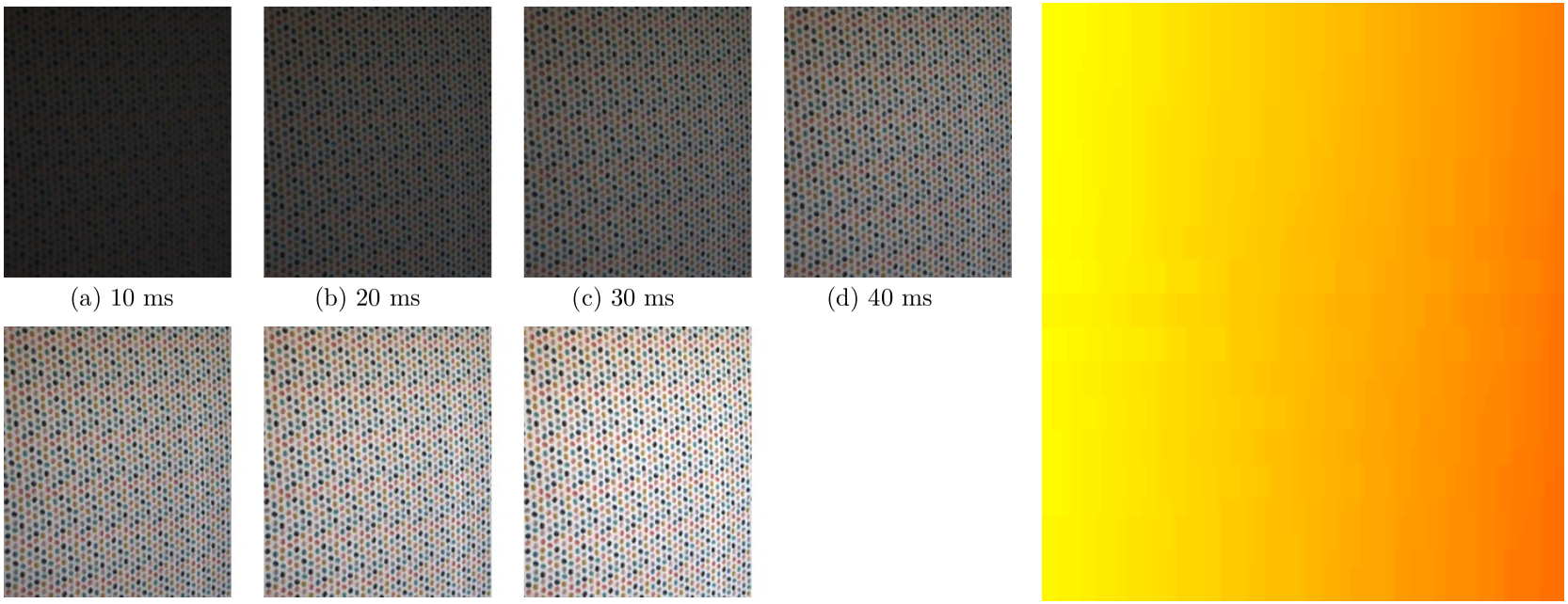


Fig. 3.12. LIDAR Recursive Upsampling Example

6. The next step in the process was registering the LIDAR data to the camera images. For this step a tool was developed that creates a sliding window of 630x630 pixels for the LIDAR data and the image data then overlays the two datasets. The window for each can be moved within the bounds of the original data to ensure a proper alignment. The tool allows the user to introduce translation shifts, rotations and skewing to he LIDAR data to allow alignment of the two datasets. Once the proper alignment was found a transformation matrix was created and applied to the LIDAR data.

Figure 3.13 shows an example of one of the scenes that was captured for this dataset after all of the data processing steps have been completed. Figures 3.13a - 3.13g show the cropped image scene taken with the seven different exposure times and Figure 3.13h shows the cropped, scaled and filtered ground truth LIDAR data. The LIDAR data has been colorized to accentuate the minute changes in depth for the scene.



(a) 10 ms (b) 20 ms (c) 30 ms (d) 40 ms

(e) 50 ms (f) 60 ms (g) 70 ms (h) LIDAR Ground Truth

Fig. 3.13. Example Image Scene Exposure Levels and Corresponding Ground Truth LIDAR Data

3.3 Error Metrics

In order to accurately compare results across multiple algorithms and multiple datasets an error metric is needed. Since the error metric performs a dimensionality reduction on a large dataset to a single number there is no one-size-fits-all metric that can accurately distinguish between various algorithms and datasets. For this reason three separate error metrics were selected based on their strengths and offsetting weaknesses.

3.3.1 Normalized Root Mean Square Error

The Normalized Root Mean Square Error (NRMSE) is a normalized version of the Root Mean Square Error (RMSE) which is defined in Equation 3.10, where X_i is the ground truth depth map and Y_i is the estimated version of the ground truth depth map and N is the total number of samples.

$$RMSE(X,Y) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(X_i - Y_i)^2} \tag{3.10}$$

In this case the smaller RMSE values indicate that the estimated depth map is in closer agreement with the ground truth depth map. A by-product of squaring the difference between the ground truth and estimated depth maps is that the RMSE metric gives errors with larger differences more weight than errors with smaller differences. This can be misleading in terms of a total error metric since a few high variance outliers can lead to a higher error [29]. However, due to its nature the RMSE is well suited to representing error distributions that behave in a more Gaussian manner.

The act of normalization produces a dimensionless statistic which helps with the performance evaluation between different datasets and/or different algorithms which are used to generate the estimated depth maps. The normalized version of the RMSE is defined in Equation 3.11.

$$NRMSE(X,Y) = \frac{RMSE}{max(X) - min(X)} \tag{3.11}$$

A nice property of the RMSE and consequently the NRMSE is that it is symmetric, i.e $NRMSE(X,Y) = NRMSE(Y,X)$.

3.3.2 Normalized Mean Absolute Error

The Normalized Mean Absolute Error (NMAE) was chosen as the second performance metric. The MAE is defined in Equation 3.12. Similarly to the RMSE, smaller MAE values indicate a closer agreement between the ground truth depth map and the estimated depth map.

$$MAE(X,Y) = \frac{1}{N}\sum_{i=1}^{N}|X_i - Y_i| \tag{3.12}$$

Table 3.4.
Five hypothetical sets (cases) of 4 errors, and their corresponding totals, MAEs and RMSEs [30]

Variable	**Case 1**	**Case 2**	**Case 3**	**Case 4**	**Case 5**
e_1	2	1	1	0	0
e_2	2	1	1	0	0
e_3	2	3	1	1	0
e_4	2	3	5	7	8
MAE	2.00	2.00	2.00	2.00	2.00
RMSE	2.00	2.24	2.65	3.55	4.00

Chai and Drexler have determined that the MAE is suitable to describe uniformly distributed errors [29]. Because MAE is an average error metric it has the distinct advantage over RMSE as it does not suffer from the same fate produced by outliers.

In fact Willmott and Matsuura produce an excellent comparison table on 5 separate error cases [30]. In that table they show that the MAE remains constant while the RMSE error continues to grow based on an increase in the variance of the error-magnitude. Table 3.4 is recreated for ease of comparison between the various cases and to illustrate the point.

It is also interesting to note that the RMSE lower bound is governed by the MAE, in fact MAE $\leq$ RMSE. Similarly the NMAE and NRMSE follow the same trend. The normalized version of the MAE metric is defined in Equation 3.13. Following the same properties of the RMSE the MAE and consequently the NMAE is a symmetric metric, i.e $NMAE(X,Y) = NMAE(Y,X)$.

$$NMAE(X,Y) = \frac{MAE}{max(X) - min(X)} \tag{3.13}$$

3.3.3 Structural Similarity Index

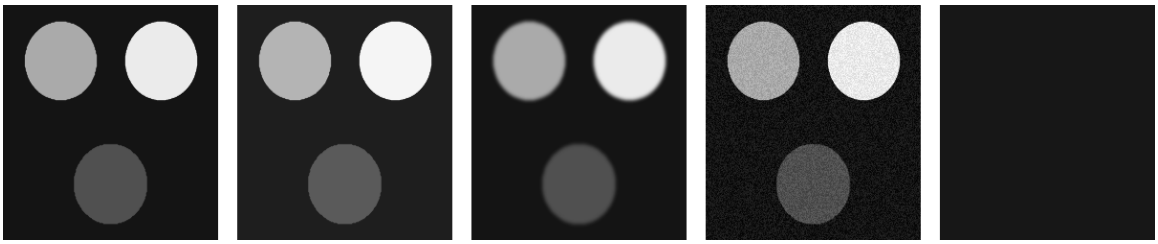
In addition to the above error metrics an image quality metric is used to assess how well the resulting depth map resembles the ground truth depth map. The Structural Similarity Index (SSIM) which was developed by Wang, et al. [31] as an image/video quality metric was chosen as the third metric for algorithm and dataset comparison. It was designed to more accurately represent the HVS and humans ability to quantify image quality. SSIM uses a combination of three separate and relatively independent quantities (luminance, contrast and structure) to make the comparison [31]. The SSIM metric is defined in Equation 3.14, where X and Y are the input images, μ_x and μ_y are the respective means of the input images, σ_x^2 and σ_y^2 are the respective variances for the input images and σ_{xy} is the covariance between the two input images. The constants c_1 and c_2 are defined as $c_1 = (k_1 L)^2$ and $c_2 = (k_2 L)^2$ where k_1 and k_2 are typically 0.01 and 0.03, respectively. The value for L is $2^{(\#\_of\_bits\_representing\_a\_pixel)} - 1$. For all of the depth maps used in this research the value of L was set to 255. The c_1

and c_2 constants were added to the equation to provide numeric stability when one or more of the denominator terms containing the constants are close to zero [31].

$$SSIM(X,Y) = \frac{(2\mu_X\mu_Y + c_1)(2\sigma_{XY} + c_2)}{(\mu_X^2 + \mu_Y^2 + c_1)(\sigma_X^2 + \sigma_Y^2 + c_2)} \tag{3.14}$$

According to Wang, et al. [31], the SSIM has several nice properties, including the symmetry property where $SSIM(X,Y) = SSIM(Y,X)$. The SSIM is also bounded on the upper end by 1 ($SSIM(X,Y) \leq 1$). And finally the SSIM has a unique maximum: $SSIM(X,Y) = 1 \iff X_i = Y_i, \forall i \in x, y$ [31]. This last property indicates that only when the two images are identical is the SSIM equal to one, otherwise it is always less than one.

Using this 3-tuple metric evaluation scheme provides insight into the source and cause of potential errors. As an example the images in Figure 3.14 represent a sample ground truth depth map (Figure 3.14a) and the sample depth map with various distortion errors added to the image (Figure 3.14b-3.14d).



(a) (b) (c) (d) (e)

Fig. 3.14. Example Depth Map Error Conditions: (a) Original Image, (b) Mean Shift Image, (c) Blurred Image, (d) Noised Image and (e) Solid Image

Table 3.5 shows the results of the three metrics for each of the four images in Figure 3.14. The mean shifted image shown in Figure 3.14b is the original image with the value of 10 added to each pixel. Here it can be seen that the NRMSE and the NMAE are the same value, which is to be expected since the errors here are

exactly as outlined in Table 3.4. The SSIM shows a value of 0.9453 which indicates that the mean shift image is very similar to the original image, which is as expected since structurally the images are indeed very similar with only a difference of 10 between pixels.

Table 3.5.
Sample Error Calculations for Figure 3.14

Image	**Mean**	**NMAE**	**NRMSE**	**SSIM**
Original Image	57.7543	-	-	-
Mean Shift Image	67.7543	0.0465	0.0465	0.9453
Blurred Image	57.7543	0.0114	0.0466	0.9541
Noised Image	57.7141	0.0407	0.0470	0.4384
Solid Image	23.0000	0.1819	0.3643	0.7443

The blurred image in Figure 3.14c is a Gaussian blurred version of the original image. The NRMSE is almost identical to the previous image, but when compared to the NMAE value it can be seen that the NRMSE is suffering from the outlier syndrome in which these outliers are amplified by the squaring process in the NRMSE. Both the NRMSE and the NMAE show relatively low error rates, which is due to the fact that the error between the original image and the blurred image is only occurring as a ring around the edge of the circles.

The noised image in Figure 3.14d has a uniformly distributed noise in the range of [-17,17] added to the original image. This again produces very similar NRMSE values to the other two distorted images. The NMAE is also the same order of magnitude as the mean shifted image. The SSIM, on the other hand is extremely low, with a value of 0.4384, which indicates that the two images have little structural similarity. While the HVS may be able to cut through the noise to discern the depth map value the SSIM metric clearly indicates that this image is not an accurate estimate of the depth map in general.

The last image, Figure 3.14e has all of its pixels set to 23. The background pixel value for the original depth map example is 20. The NRMSE and NMAE show a very large error difference, however if the non-background depth map values were closer to the background the solid image errors would become much smaller indicating a potentially good results which is obviously not the case. The SSIM value is very high for the fact that the solid image in no way represents a good match to the original image. Without the other two metrics the SSIM could provide a misleading conclusion that the resulting depth map is a better representation of the original depth map than the noised image.

3.4 Summary

This chapter introduced the synthetically blurred dataset based on the images from the Middlebury College Stereo Vision Dataset [7, 8]. The method of creating the blurred versions of the Middlebury images was also discussed. This chapter also discussed the error metrics, NMAE, NRMSE and SSIM that will be used to provide a comparison of the performance of the various algorithms and datasets used in this research.

This chapter also introduced the real world dataset that was created using a FLIR® Chameleon3 1.3 MP Color USB3 Vision camera coupled with a Varioptic® microfluidic lens. The ground truth data was collected using an Ouster OS-1 64-beam 2-D scanning LIDAR. The data collection and processing methods were also described in this chapter. The next chapter will present an analysis of the current state of the art DfD algorithm on the datasets out lined in this chapter.

4. DEPTH FROM DEFOCUS USING THE GRAPH CUTS ALGORITHM

In this chapter the current state-of-the-art DfD algorithm and graph cuts energy minimization algorithm will be introduced. A detailed discussion will be presented on each of the major components of the algorithm and finally the results will be presented for each of the datasets discussed previously in Chapter 3.

4.1 Algorithm Overview

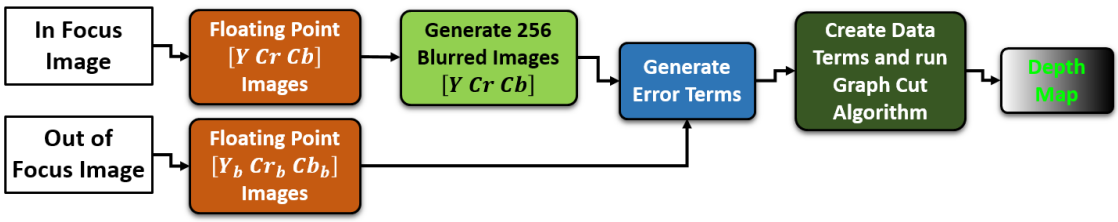


Fig. 4.1. Depth from Defocus Block Diagram

Figure 4.1 outlines the overall algorithmic approach taken. The first step in the process is to take two images of a scene. One image is considered to be in-focus and the other image is considered to be defocused, or out-of-focus. The out-of-focus image is taken by adjusting the focus distance either closer to or farther from the camera. The images are then converted to 32-bit floating point images and then transformed into the YCrCb color space. Once the images are converted, the in-focus image is then blurred using the same Gaussian kernel which was defined in Equation 3.1. For each class (i) a new blur kernel is generated based on a desired sigma value (σ_i). The sigma value is determined based on the number of classes in the problem, in this case 256, and the range of maximum and minimum sigma values, σ_{max} and σ_{min}

respectively. For the format of the depth maps that are in the current datasets the number of depth map levels is equal to 256 [0 ... 255] which corresponds to an 8-bit grayscale depth map image. The sigma value is again calculated in the same manner as discussed in Section 3.1 using equations 3.2 and 3.3.

Initially, Liu [21] had used a σ_{min} value of zero, however it was discovered that the difference in blur kernels was not changing until sigma reached a value of approximately 0.24. For this reason a σ_{min} value of 0.32 was chosen to ensure that each blur kernel would be unique.

After the N blurred versions of the in focus image have been created the next step is to generate the error terms. The error terms are created using Equation 4.1. Where X_Y, X_{Cr} and X_{Cb} represent each color component of the out of focus image. $\hat{X}_Y^{(i)}$, $\hat{X}_{Cr}^{(i)}$ and $\hat{X}_{Cb}^{(i)}$ are the color components for each of the synthetically blurred versions of the in focus image and c is a constant. The term er_i represents the sum of the squared errors of each color channel for each of the classes. This error term is what is to be minimized in order to find the best (most accurate) depth map for a given scene. This minimization occurs in the form of a graph cuts algorithm. As a baseline for comparison against the deep learning algorithms explored in this research the graph cuts method used by Liu [21] will be used as the standard to compare against.

$$er_i = c\left[\left(X_Y - \hat{X}_Y^{(i)}\right)^2 + \left(X_{Cr} - \hat{X}_{Cr}^{(i)}\right)^2 + \left(X_{Cb} - \hat{X}_{Cb}^{(i)}\right)^2\right] \; i \,\epsilon\, \{0... N-1\} \quad (4.1)$$

The first variants of the graph-cuts algorithm were designed for solving binary problems, for example foreground/background region segmentation in which only two classes exist. In this context the algorithm is guaranteed to find the global minimum [32]. When more than two classes exist, such as in the problem of depth estimation where there are 256 possible classes, the algorithm cannot be guaranteed to obtain the global minimum. Recent advancements by Boykov, Kolmogorov, et al. [10,33,34] have led to the theory of multi-label graph cuts algorithms which handle multiple classes. While the algorithm may no longer be guaranteed to find a global

minimum, it does have certain nice properties, like convergence in a finite number of iterations and that the algorithm is known to approximate the global minimum cost by a factor of two. This particular variant of the graph cuts algorithm that can handle more than two classes is known as the $\alpha - expansion$ method and was developed by Boykov, Kolmogorov, Veksler and Zabih [10, 33, 34].

The graph cuts algorithm is part of a family of min cut/max flow energy optimization algorithms. The goal in this particular case is to optimize a Gibbs Energy equation (4.2).

$$E(L) = \sum_{i \epsilon \mathcal{V}} E^i_{data}\left(L(i)\right) + \sum_{i \epsilon \varepsilon} \sum_{j \epsilon \varepsilon} E^i_{smoothness}\left(L(i), L(j)\right) \tag{4.2}$$

Where the data term, $E^i_{data}\left(L(i)\right)$, is the term er_i defined in Equation 4.1. For the smoothness term there are many possible choices, however it must meet two important criteria. Firstly the smoothing term must be a convex function [35] and second the smoothing term must also be a metric. Steen and Seebach outline a set of criteria that a function must meet in order to be deemed a metric [36]:

$$\begin{aligned} &d(x, y) \geq 0 \\ &d(x, y) = 0 \Leftrightarrow x = y \\ &d(x, y) = d(y, x) \\ &d(x, z) \leq d(x, y) + d(y, z) \end{aligned} \tag{4.3}$$

Where $d(x, y)$ is a distance between x and y [36]. There are many choices for the smoothing function that meet the definition of a metric. The goal is to pick a function that can handle potential abrupt changes in the depth map values. Table 4.1 outlines several potential smoothing functions that avoid over-penalizing sharp changes in the disparity between neighboring pixel, while generally favoring disparity maps in regions that have similar labels [32]. Table 4.1 is reproduced from [32] and illustrates several popular smoothing functions.

Table 4.1.
Discontinuity Preserving Smoothness Functions [32]

Name	$\boldsymbol{E_{smoothness}(L(i), L(j))}$
Truncated Quadratic	$\beta \cdot min\left(K, (L(i) - L(j))^2\right)$
Truncated Absolute	$\beta \cdot min(K, \lvert L(i) - L(j)\rvert)$
Potts Model	$\begin{cases} K, & if L(i) \neq L(j) \\ 0, & otherwise \end{cases}$
Intensity-Adaptive Potts Model	$\begin{cases} 2K, & if \lvert I_1(i) - I_2(j)\rvert \leq \beta \, and \, L(i) \neq L(j) \\ K, & if \lvert I_1(i) - I_2(j)\rvert > \beta \, and \, L(i) \neq L(j) \\ 0, & otherwise \end{cases}$

Boykov, et al. proposed the use of the truncated absolute smoothness term with $\beta = 1$ and $K = 4$ [10]. However, because of the potential for a very large disparity change in the depth map, (e.g. transitioning from a value of 0 to a value of 255 or vice versa), this function cannot handle such large jumps. Liu proposed a modification to the recommended smoothness function in which $K = 255$ which is the maximum disparity value for the given datasets [21].

The graph cuts algorithm is based on the mathematical principals of graph theory. For example, let $\mathcal{G}$ be a weighted graph defined by Equation 4.4.

$$\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle \tag{4.4}$$

Where $\mathcal{V}$ is a set of vertices or nodes and $\mathcal{E}$ is a set of edges in the graph. In the graph cuts method each graph also contain two special vertices, known as the source terminal (S) and the sink terminal (T). In the application of the graph cuts

optimization to the DfD problem, the vertices are related to the squared error differences between the pixels of the defocused image and the pixels of the synthetically blurred images. The terminals are the set of labels, in this case the depth map values, typically the source represents the α or label that will replace the current label and the sink is the label that will be kept. Initially each terminal is connected to each node by an edge.

The $\alpha - expansion$ method can consider any combination of neighboring pixels, in this instance the method only takes into account the neighboring pixels that are north, south, east and west of the current pixel, or data point, of interest. This is also known as 4-adjacency or 4-neighbor. The neighboring pixels are connected to each other by links called N-links and the terminals are connected to each data term links called T-links.

Figure 4.2 shows a simplified graph connection. The solid black double arrow lines represent the N-links, the T-links between the source (S) and each pixel are represented by the blue dashed lines and the T-links between the sink (T) and each pixel are represented by the red dashed line. The example shows the $\alpha - expansion$ for $\alpha = 0$. The sample input is the cut by the green dashed line resulting in the expansion of the α label which can be seen in the resulting output.

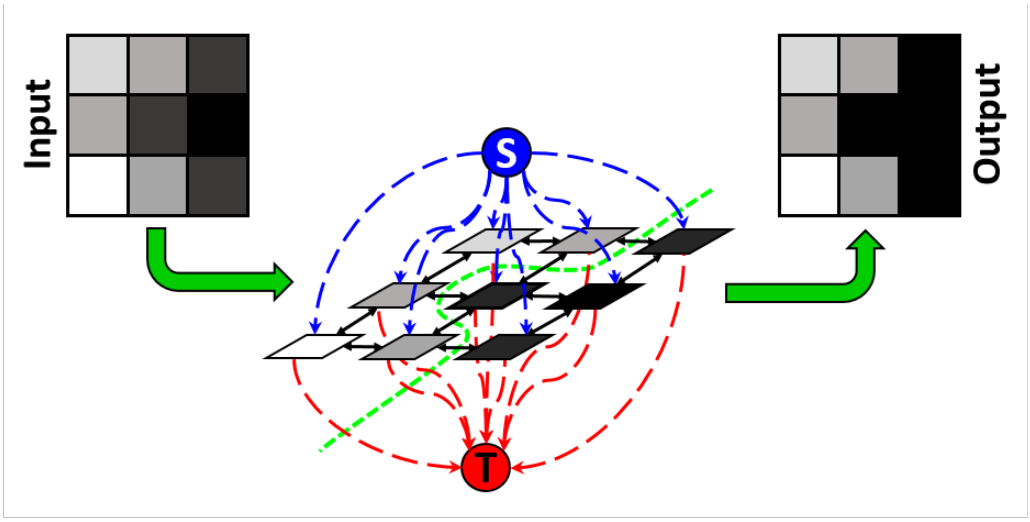


Fig. 4.2. Graph-Cuts Example Node

Each edge $\mathcal{E}$ has a non-negative weight associated with it, and this weight is used to determine the optimal cut in the graph. These edges can be thought of more simply as a transport mechanism such as a tube carrying water, hence the term max flow. The algorithm looks at the weights of all of the edges to determine maximum flow, or in other words the highest total weight value for each path from the source terminal to the sink terminal. Once the flows are computed along each edge the maximum flows from each source to each sink represent the minimum cut. A cut ($\mathcal{C}$) is defined as a subset of the edges in the graph ($\mathcal{C} \subset \mathcal{E}$) such that the terminals are separated in the induced graph [10]. Once the minimum cuts have been determined the nodes that have been separated from the source are given the label of the α depth map value and the nodes that were not separated keep their existing depth map labels [32].

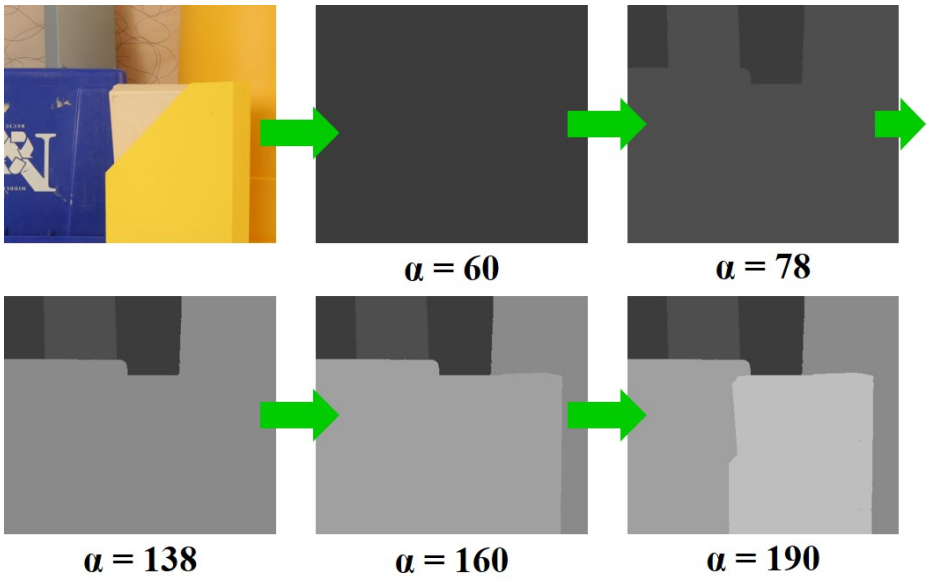


Fig. 4.3. Graph-Cuts Example Node

One iteration consists of expanding each of the potential depth labels. For example at the start of the algorithm the α value would be zero. The algorithm would expand all pixels that have a minimum cut with respect to the current α label. After the completion of the expansion the next label is assigned to α and the process of finding the minimum cuts for the new α is performed. This process is then repeated for each

of the potential label classes. Figure 4.3 shows the expansion process for various α values. At each iteration through the labels many pixels are allowed to change their values simultaneously.

After completion of all of the required cuts the final product is the estimated depth map. This algorithm is very robust to input image size and the number of label classes. It also does not need any $a - priori$ information about the scene or the number of classes. Its major drawback are the computational requirements to iterate through each pixel and each class label. It must traverse the whole image space and select from one of the available classes for each of the pixels. This also means that the algorithm run time is completely dependent on the input image size and the number of possible classes. Figure 4.4 shows the algorithm run time versus the number of pixels in an image presented to the algorithm. The results are averaged across 30 runs for each image size, and it can be seen that as the images increase in size the algorithm run time also increases which is to be expected.

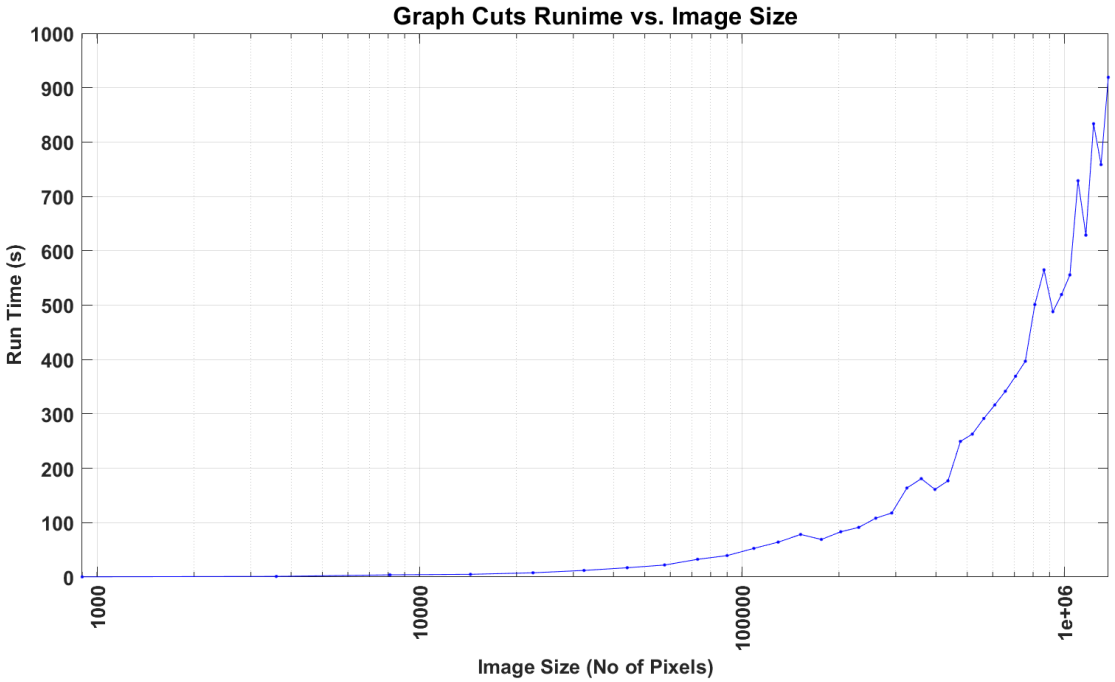


Fig. 4.4. Graph-Cuts Runtime Chart

4.2 Synthetically Blurred Dataset Results

The results presented here for the synthetically blurred Middlebury College datasets are broken down by illumination and exposure levels. Figure 4.5 shows the NMAE, NRMSE and the SSIM results for the graph cuts method for each of the left and right scenes for the images in the Illumination 1 category. The images that are in the exposure level 0 category produce the worst results for each of the three metrics. The other two exposure levels (exposure level 1 and exposure level 2) produce very similar performance results for each of the three metrics. The poor results of the exposure level 0 images indicate that the lighting conditions can affect the ability of the graph cuts algorithm to differentiate the various depth levels.

Figure 4.6 shows an example of the Flowerpots images in the illumination 1 category. It can be seen that the exposure level 0 image has lost a lot of the object definition and boundaries. The lower exposure level also prevented capturing a lot of the textures that can be seen in the other two exposure level images. This lack of texture makes it very difficult to determine the blur differences between the in-focus and the out-of-focus image.

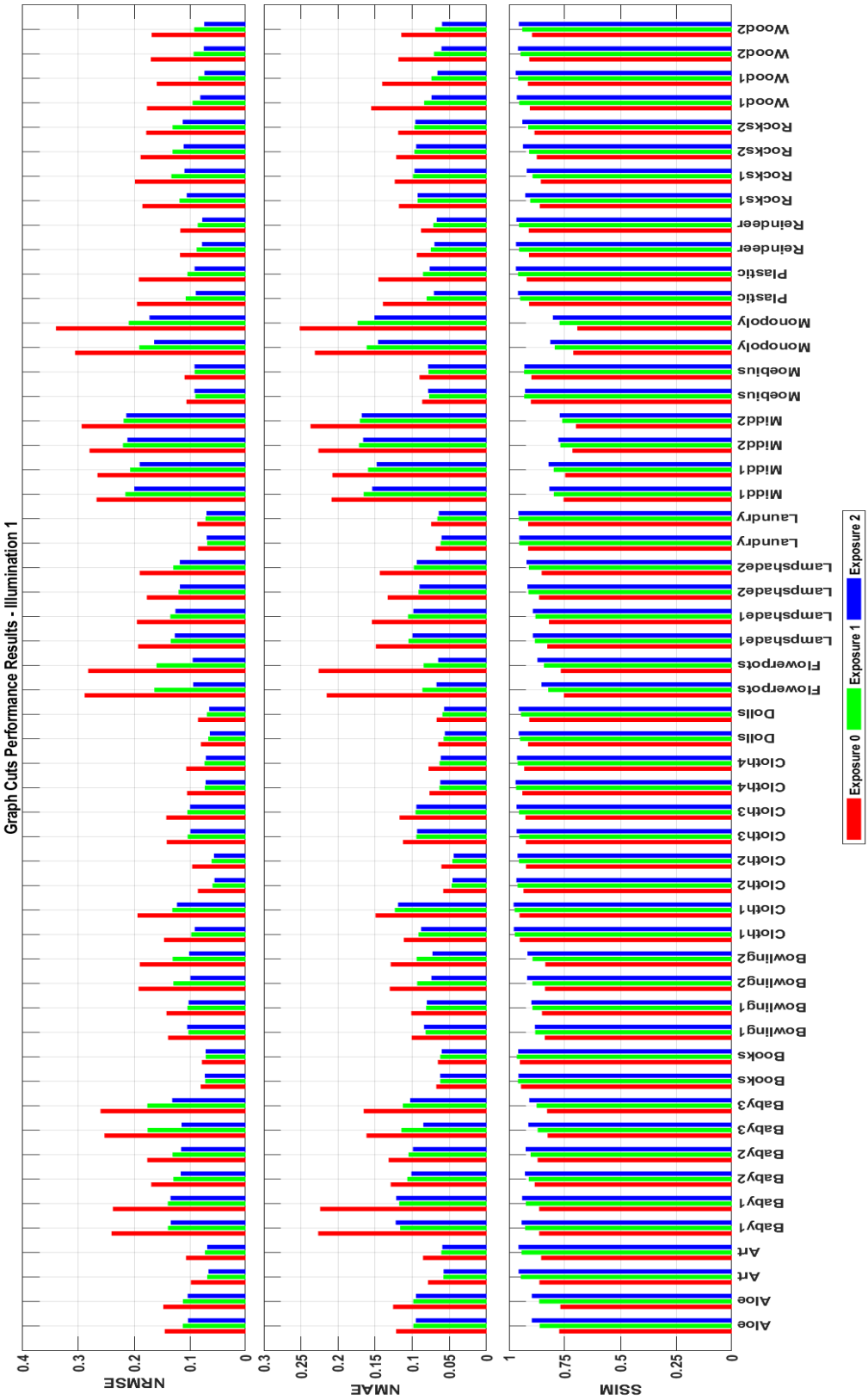


Fig. 4.5. Graph Cuts Performance Results for Middlebury College Dataset - Illumination 1

(a) Exposure Level 0 (b) Exposure Level 1 (c) Exposure Level 2

Fig. 4.6. Flowerpots Images Under Illumination 1 Conditions

Image processing methods can improve the underexposure problem, the most basic process being an increase in the image mean by adding a constant value to each pixel. This has the drawback of also amplifying the noise in the image. A more advanced method developed by Park, et al. is an optimization-based method to enhance low-light images using the spatially adaptive Retinex model [37]. Recently there has been some work towards deep learning to improve the performance of image restoration techniques. Chen, et al. have developed a deep learning method that can process extreme low-light images "with severe noise and color distortion that is beyond the operating conditions of existing enhancement pipelines" [38]. These methods add additional steps in preprocessing the input images before they can be run through the graph cuts algorithm.

Figure 4.7 shows the NMAE, NRMSE and the SSIM results of the graph cuts algorithm for the images in the illumination 2 category. As with the previous illumination 1 results the exposure level 0 results are worse than the other two exposure levels. Figure 4.8 show similar performance results for the NMAE, NRMSE and the SSIM results of the graph cuts algorithm for the images in the illumination 3 category. The missing plot points in each chart indicate that there was no data taken for a given image and illumination/exposure level combination. This is mainly due to the fact that the data was not usable. In all cases the images with the shortest exposure time

have the poorest results. This is again due to the fact that the texture of the objects in the scene is lost and therefore the difference in blurs is no longer discernible making it very difficult for the graph cuts algorithm to determine the depth map levels.

Table 4.2 lists the top 5 image pairs and the bottom 5 image pairs where the graph cuts algorithm performed the best and worst respectively. The primary metric used for ranking the results is based on the NRMSE metric with the NMAE metric used as a tie breaker. It should be noted that there is a large disparity in the performance results between the top and bottom results for each of the three metrics. The image pair where the algorithm performed the best was approximate 6.1 times better by the NRMSE metric, approximately 5.5 times better by the NMAE metric and had an SSIM improvement of 0.2725 over the image pair where the algorithm performed the worst. For this reason an average NRMSE is not reported across the entire dataset, but instead is reported for each exposure level. The same is also true for the NMAE and the SSIM metrics.

Table 4.3 outlines the mean and standard deviation for each illumination and exposure level combination for each of the evaluation metrics. The table also shows the overall combined mean and standard deviation for each exposure level in the dataset. The table shows that the exposure level 0 images for each illumination level have the largest mean and standard deviation for the NRMSE and NMAE metrics. Similarly the mean SSIM metric for the exposure level 0 images is lower as compared to the other exposure levels. This again points to a weakness in the graph cuts algorithm's ability to process images in low lighting conditions.

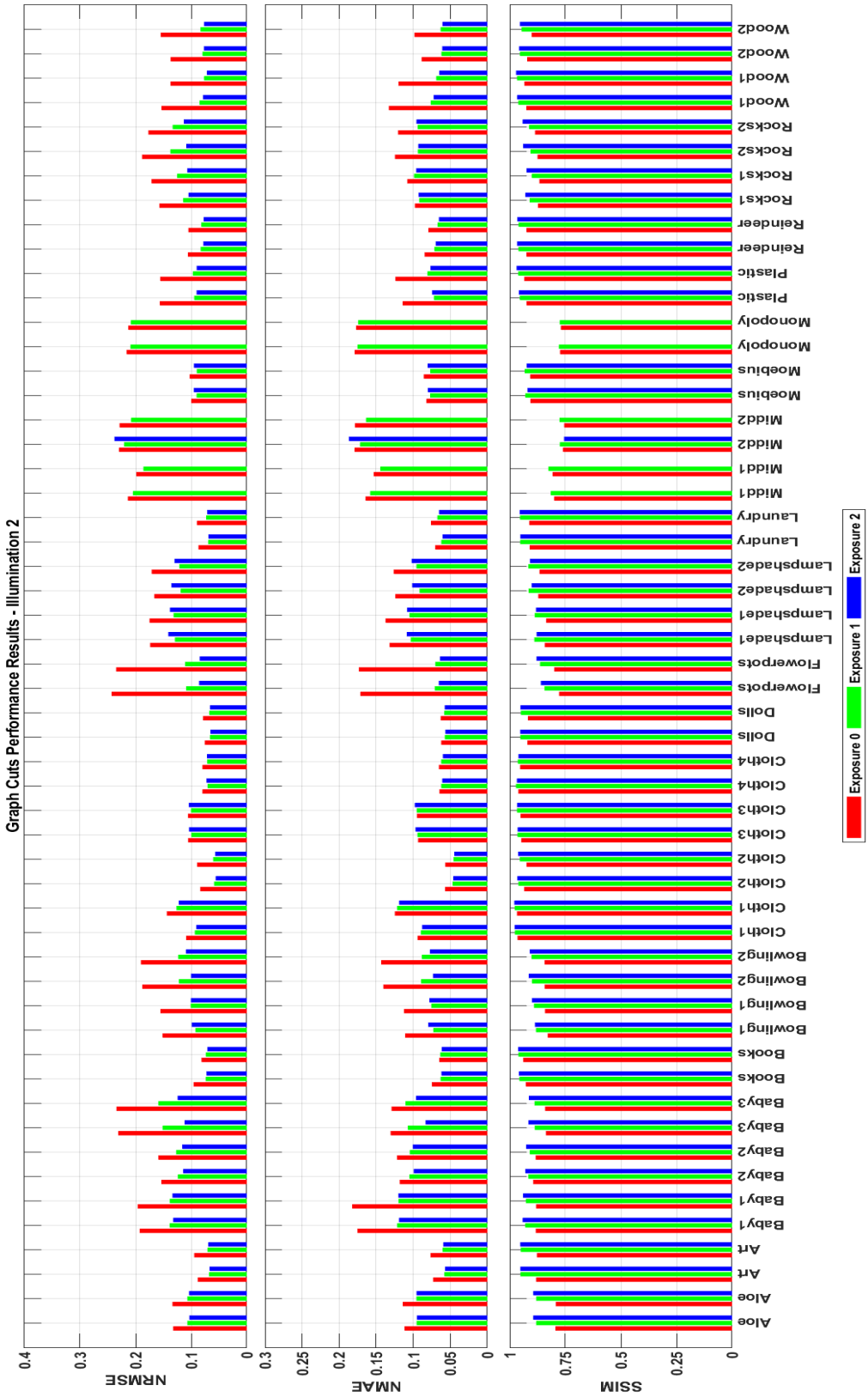


Fig. 4.7. Graph Cuts Performance Results for Middlebury College Dataset - Illumination 2

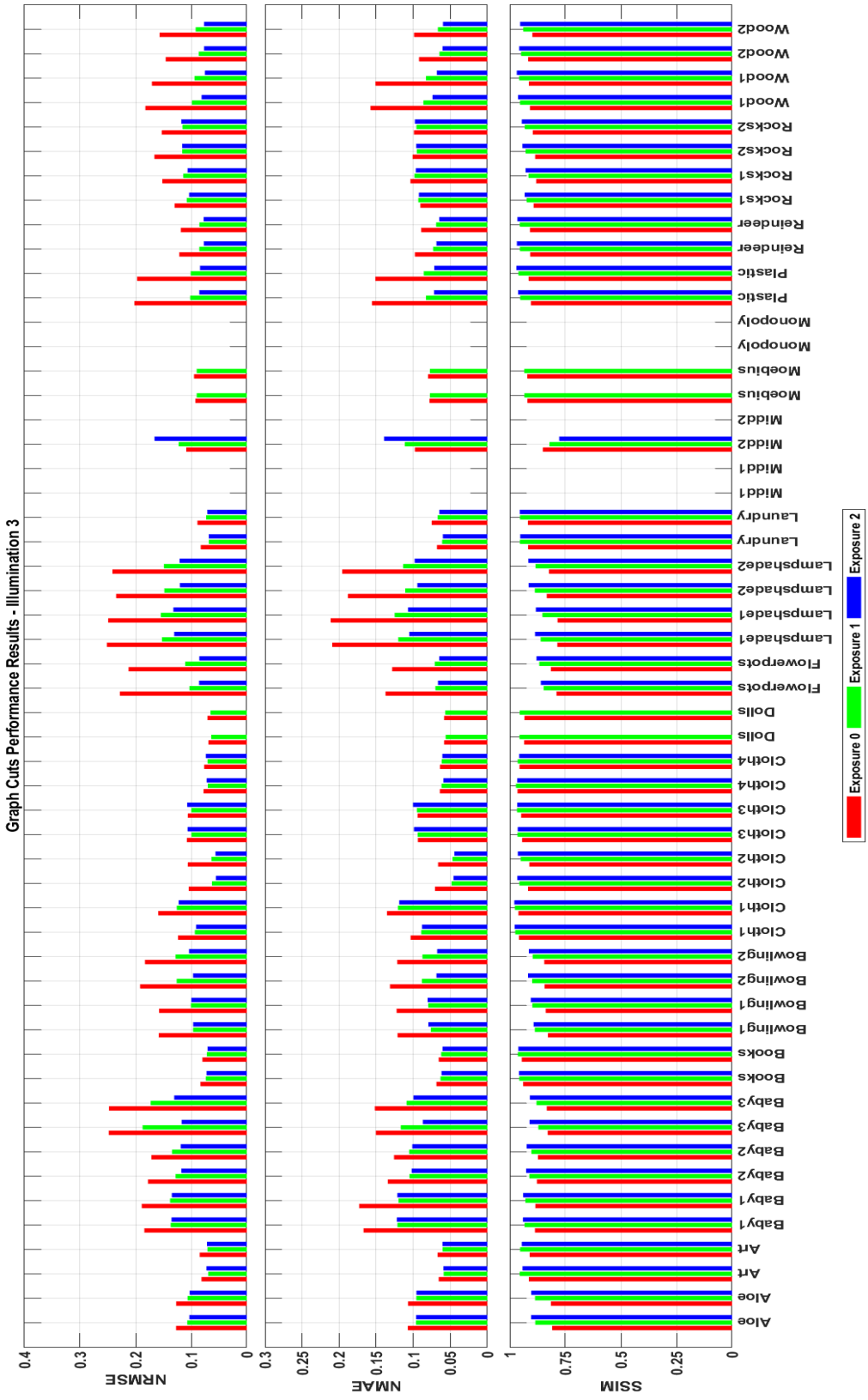


Fig. 4.8. Graph Cuts Performance Results for Middlebury College Dataset - Illumination 3

Table 4.2.
Top 5 and Bottom 5 Graph Cuts Performance Results for the Middlebury College Stereo Vision Dataset

Name	**View**	**Illumination**	**Exposure**	**NRMSE**	**NMAE**	**SSIM**
Top 5						
Cloth2	Left	3	2	0.05564	0.04564	0.96617
Cloth2	Left	1	2	0.05580	0.04583	0.96685
Cloth2	Left	2	2	0.05600	0.04605	0.96606
Cloth2	Right	3	2	0.05635	0.04449	0.96364
Cloth2	Right	1	2	0.05665	0.04442	0.96179
Bottom 5						
Flowerpots	Right	1	0	0.28269	0.22632	0.76750
Flowerpots	Left	1	0	0.28892	0.21530	0.75375
Midd2	Right	1	0	0.29444	0.23722	0.69981
Monopoly	Left	1	0	0.30607	0.23134	0.71187
Monopoly	Right	1	0	0.34040	0.25169	0.69369

Figure 4.9 shows the resulting depth maps. The best results are shown on the left side of the figure and the worst results are shown on the right side of the figure. While the results on the left are very impressive, the results on the right are not so impressive. The algorithm effectively loses objects in the darker regions of the images. The wooden box in front of the Monopoly® board is completely missing, the lampshade and teddy bear are completely missing and the flower pots on the right hand side of the image are missing as well. A further look into some of the other results shows a similar effect, in that objects are lost or the algorithm smears the depth map values across objects that are not at the same depth.

Fig. 4.9. Top 5 and Bottom 5 Performance Results for the Middlebury College Stereo Vision Dataset. (a) & (e) Infocus Image, (b) & (f) Defocused Image, (c) & (g) Ground Truth Depth Map, (d) & (h) Graph-Cuts Computed Depth Map.

Table 4.3.
Graph Cuts Performance Mean & Standard Deviation

		NRMSE		**NMAE**		**SSIM**	
Lighting		**Mean**	**Std**	**Mean**	**Std**	**Mean**	**Std**
Exp 0	Illum 1	0.1733	0.0679	0.1321	0.0533	0.8627	0.0701
	Illum 2	0.1496	0.0505	0.1141	0.0372	0.8783	0.0584
	Illum 3	0.1486	0.0556	0.1132	0.0415	0.8877	0.0503
	Overall	0.1574	0.0593	0.1200	0.0452	0.8759	0.0609
Exp 1	Illum 1	0.1186	0.0439	0.0938	0.0320	0.9092	0.0574
	Illum 2	0.1124	0.0419	0.0910	0.0325	0.9146	0.0551
	Illum 3	0.1046	0.0302	0.0852	0.0217	0.9259	0.0391
	Overall	0.1121	0.0396	0.0902	0.0294	0.9163	0.0517
Exp 2	Illum 1	0.1051	0.0379	0.0881	0.0303	0.9231	0.0524
	Illum 2	0.0979	0.0309	0.0823	0.0249	0.9329	0.0401
	Illum 3	0.0975	0.0247	0.0823	0.0221	0.9338	0.0382
	Overall	0.1004	0.0320	0.0844	0.0262	0.9296	0.0444

Table 4.4 shows the average run times for the graph cuts algorithm. Based on the average run times the algorithm is extremely dependent on the exposure level. In fact, the graph cuts algorithm takes longer to process the image pairs with lower illumination and exposure levels versus the image pairs with higher illumination and exposure levels.

Table 4.4.
Average Graph Cuts Run Time for the Middlebury College Stereo Vision Dataset

	Run Time (s)		
Lighting	**Exposure 0**	**Exposure 1**	**Exposure 2**
Illumination 1	301.756	190.978	162.000
Illumination 2	268.622	180.911	166.371
Illumination 3	254.951	190.873	170.427

4.3 Summary

In this chapter the overall DfD algorithm which employed the graph cuts energy minimization algorithm was discussed. The results for the synthetically blurred datasets were presented. The algorithm produces a wide range of results based on the input image pair illumination and exposure levels, primarily performing better when the lighting conditions are optimal. The data also shows that the graph cuts algorithm run time is very dependent on the illumination and exposure levels, as the exposure level increased the run time decreased. In the next chapter the proposed DFD-Net, a deep learning approach to solving the DfD problem will be presented.

5. DEPTH FROM DEFOCUS USING A DEEP LEARNING ALGORITHM

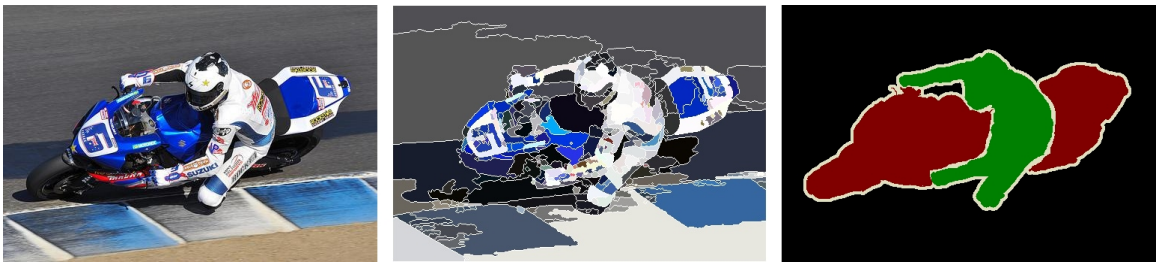
With the recent advancements in machine learning techniques that take advantage of fully connected neural networks and convolutional neural networks (CNN) there are several potential algorithms and concepts to explore. These new methods can be thought of more as architectures instead of algorithms due to their building block nature. This chapter will explore the application of these deep learning architectures to the solution of the DfD depth estimation task. In order to narrow down the search from such a wide scope, the candidate architecture was taken from a semantic segmentation solution.

5.1 Algorithm Overview

The output of a semantic segmentation task is very similar to the output of a depth map task and for this reason the deep learning architectures that perform this segmentation task are considered as the starting point for the application of deep learning to the DfD task. Because the candidate architecture is based on semantic segmentation a little background on semantic segmentation is warranted. Semantic segmentation is the process of segmenting an image based on the objects in a given scene versus the more traditional segmentation process where similarly valued pixels that are within a given threshold are lumped together regardless of the object to which the pixels belong.

Figure 5.1 shows an example of the difference between the traditional segmentation and semantic segmentation. Figure 5.1a and 5.1c are provided by the Visual Object Classes Challenge 2012 (VOC2012) [39]. It can be seen that the traditional segmentation method, Figure 5.1b, simply groups like valued pixels, and in this case

the pixels are assigned an average value from a given segmentation patch. The semantic segmentation method groups pixels together for the same object (motorcycle, rider and background).



(a) Original Image (b) Traditional Segmentation (c) Semantic Segmentation

Fig. 5.1. Example Segmentation (Traditional vs. Semantic) [39]

The basis for this work is rooted in the semantic segmentation architecture known as the U-Net developed by Ronneberger, et al. [17]. The U-Net was originally designed to perform a binary classification at the pixel level of images of cellular structures into either cells, "1", or background, "0". The success of the U-Net has led to its use across many other tasks. Figure 5.2 shows the graphical representation of the U-Net. The U-Net can be thought of as an encoder/decoder style network where on the left-hand side of the network the input image is encoded into distinct feature maps and on the right-hand side of the network the feature maps are then decoded into a semantic segmentation map.

The U-Net, itself, was based on a fully convolutional network designed for semantic segmentation [40]. This network took in a full size image and gradually contracted the image to a very small feature map (height and width), but with a large number of these feature maps (4096). The network then performs a single large expansion at the end to produce a segmentation map similar in size to the original input.

The U-Net removes this single large expansion in favor of several smaller, gradual expansions. The design of the U-Net is such that it is symmetric with respect to the contraction and expansion that occurs to the input image. The U-Net also adds skip connections that concatenate the output of a particular layer prior to a contraction

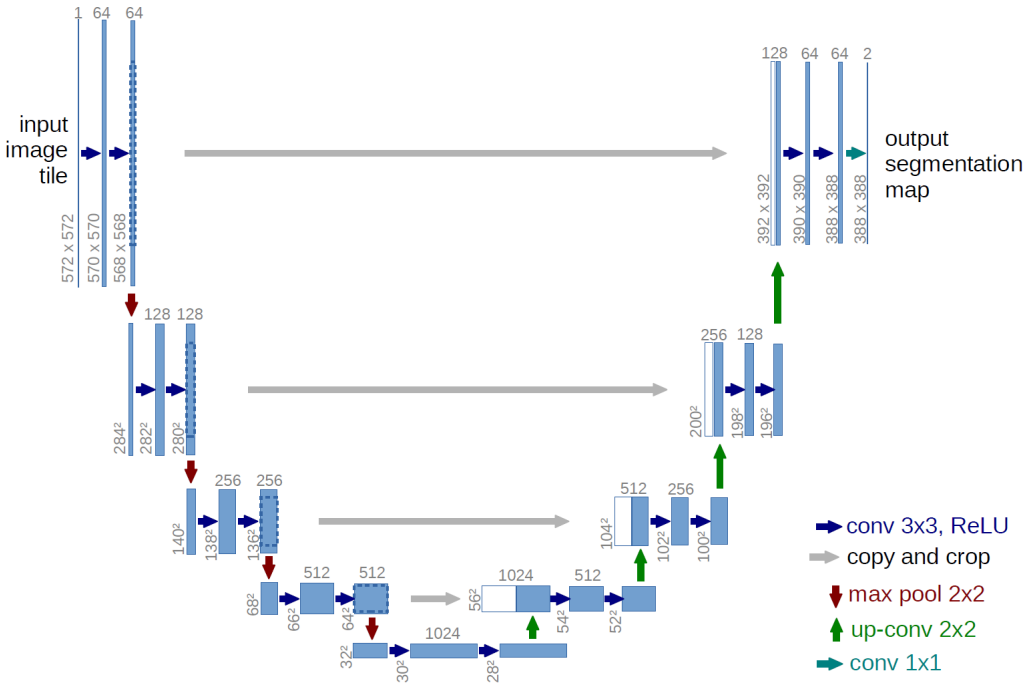


Fig. 5.2. U-Net Semantic Segmentation Deep Learning Architecture [17]

to the output of an expansion layer. This concatenation of tensors allows the U-Net to perform improved localization of features within the image. This is because the expansion, by nature, produces a coarse representation of the decoded feature map and the concatenation of the encoded feature map prior to a contraction allows fine grain details to be reintroduced into the decoding chain, which is what allows the network to produce fine grain feature localization [17]. The arrangement of the U-Net in Figure 5.2 is designed to show where each contraction and expansion occur, with each level indicating when the size of the tensor image has been changed.

The U-Net is essentially a mapping function that maps a grayscale input image of cells to a binary map of the location of the cells. This mapping is shown in Equation 5.1, where $\mathcal{G}$ is a function that maps the input image (X) to the label space (L). It is assumed that this mapping is a surjective mapping, as denoted by the "$\twoheadrightarrow$". This simply means that each label will be mapped to by a given input.

$$\mathcal{G} : X \twoheadrightarrow L \tag{5.1}$$

Where $X \epsilon \mathbb{R}^{48^2}$, $x_i \epsilon [0, ..., 255]$ and $L \epsilon [0, 1]$. The dimensionality of X is determined by the size of the input. When considering the input space for the U-Net the sizes of the receptive fields at each level contribute to the minimum allowed input size. The receptive field is essentially the area of the image that is used to make a decision. Since the largest convolutional filter size for the U-Net is 3x3 the receptive field is 3x3, however this is not the input receptive field, but the smallest receptive field at the lowest level of the U-Net architecture. Because the U-Net gradually contracts the input, the minimum input size that results in a 3x3 receptive field at the lowest level is 48x48 pixels.

The U-Net is constructed of pairs of convolutional layers and Rectified Linear Unit (ReLU) layers. Figure 5.3 shows the arrangement of these layers to create the U-Net convolutional block. The numbers N_1 and N_2 are the number of convolutional filters in each convolutional layer and the 3x3 represents the size of each of the filters.

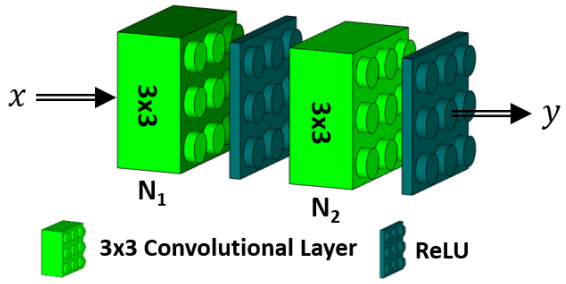


Fig. 5.3. Example U-Net Convolution Block

Equation 5.2 represents the overall equation that governs the behavior of this block, where x is the input to the block and y is the output.

$$y = \mathcal{F}(x) \tag{5.2}$$

The function $\mathcal{F}$ is defined in Equation 5.3, where W_i is the set of convolutional filters in layer i, b_i is the bias term within the convolutional layer and θ_r is the ReLU activation layer.

$$\mathcal{F} = \theta_r(W_2 \cdot \theta_r(W_1 x + b_1) + b_2) \tag{5.3}$$

The ReLU is a non-linear activation layer which is used to constrain the output of the previous layer. The ReLU layer was developed by Hahnloser and Seung [41] and is designed to only pass the positive outputs from the previous layer. The mathematical behavior describing the ReLU is shown in Equation 5.4.

$$\theta_r(x) = \begin{cases} x, & x > 0 \\ 0, & otherwise \end{cases} \tag{5.4}$$

The Depth from Defocus Network (DfD-Net) is based on this U-Net structure where blocks of convolutional filters and activation layers transform the input data and then pass the transformation on to downsampling or upsampling layers as well as bypassing lower levels. The major modification that was made to the U-Net to create the DfD-Net was replacing the basic convolutional/ReLU layer pairs, shown in Figure 5.3, with a "residual" block. This block was originally developed by He, et al. as a means to reduce model complexity and improve overall performance [42]. The original residual block is defined by Equation 5.5, where x is the input to the block, W_i are the weights of the convolutional filters to be learned and y is the output of the residual block.

$$y = \mathcal{F}(x, \{W_i\}) + x \tag{5.5}$$

The residual block can be broken down into two separate paths. The first is the residual mapping path described by the function $\mathcal{F}$ in Equation 5.6 and is the learned portion of the path [42]. The second path is the "identity" path in which the input is simply added to the output of the residual path.

$$\mathcal{F} = W_2 \cdot \theta_r(W_1 x + b_1) + b_2 \tag{5.6}$$

Both the U-Net and the original residual block use the ReLU activation function, but the ReLU has a drawback. It can enter a state where the neuron effectively dies, preventing information from being passed forward to the next layer. The gradient is also diminished, which means that the gradients are prevented from flowing backwards through the network during the back propagation steps in the training [43]. To combat this problem He, et al. developed the the parametric ReLU (pReLU) [44], which adds an additional learned parameter (α) that allows input values less than zero to pass through the function. By allowing these negative input values to pass, the pReLU avoids the vanishing gradients problem [44]. The governing equation for the pReLU has the following form:

$$\theta_p(x) = \begin{cases} x, & x > 0 \\ \alpha x, & otherwise \end{cases} \tag{5.7}$$

Within the DfD-Net residual block the ReLU has been replaced by the pReLU activation function. Another change that was made to the original residual block was the introduction of the batch normalization process. This process, developed by Ioffe and Szegedy, performs a transformation that normalizes the features within a network, "by making it have the mean of zero and the variance of one" [45].

The batch normalization process has the advantage of ensuring that the distribution of the training data remains consistent across the entire training process, which allows the network to train faster [45]. This normalization occurs independently across each dimension, i.e. individual convolutional filter outputs, within a training mini-batch $\mathcal{B} = \{x_1^{(k)}, x_2^{(k)}, ..., x_m^{(k)}\}$, where k represents the dimensional output of the previous layer, or the number of feature maps, $x_i^{(k)}$ is a particular input within the mini-batch and m represents the size of the mini-batch. Equations 5.8 - 5.10 outline the normalization process across the entire mini-batch [45]. In Equation 5.10 the constant, ϵ, is added in order to maintain numerical stability in the event the variance, $\sigma_{\mathcal{B}}^2$, goes to zero.

$$\mu_{\mathcal{B}}^{(k)} = \frac{1}{m}\sum_{i=1}^{m} x_i^{(k)} \tag{5.8}$$

$$\sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m}\left(x_i^{(k)} - \mu_{\mathcal{B}}^{(k)}\right)^2 \tag{5.9}$$

$$\widehat{x}_i^{(k)} = \frac{x_i^{(k)} - \mu_{\mathcal{B}}^{(k)}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \tag{5.10}$$

Once the normalized input has been calculated an additional scale and shift operation is performed on the normalized input. This step ensures that the batch normalization transform can represent the "identity" transform [45]. Equation 5.11 shows the final step in the batch normalization process, where $\gamma^{(k)}$ is the scaling factor and $\beta^{(k)}$ is the shifting factor. These parameters are learned during the training process.

$$BN_{\gamma^{(k)},\beta^{(k)}}(x_i^{(k)}) = \gamma^{(k)}\widehat{x}_i^{(k)} + \beta^{(k)} \tag{5.11}$$

Figure 5.4 shows the modified residual block used in the DfD-Net. The numbers N_1 and N_2 are the number of convolutional filters in each convolutional layer and the 3x3 represents the size of each of the filters. The number N_0 represents the number of inputs into the residual block from the previous layer(s). It is important to note that in order to avoid a tensor addition imbalance N_2 must equal N_0.

Based on the modifications made to the original residual block the residual function, $\mathcal{F}$, is now defined in Equation 5.12. The bias term, b_1, associated with the weights, W_1, is now subsumed in the β term within the batch normalization function.

$$\mathcal{F} = W_2 \cdot \theta_r(BN(W_1 x)) + b_2 \tag{5.12}$$

The residual function, by way of the convolutional filters, performs a localized transformation of the input. This is because the convolutional filters are at most 3x3. Therefore, only the values directly neighboring the anchoring input influence

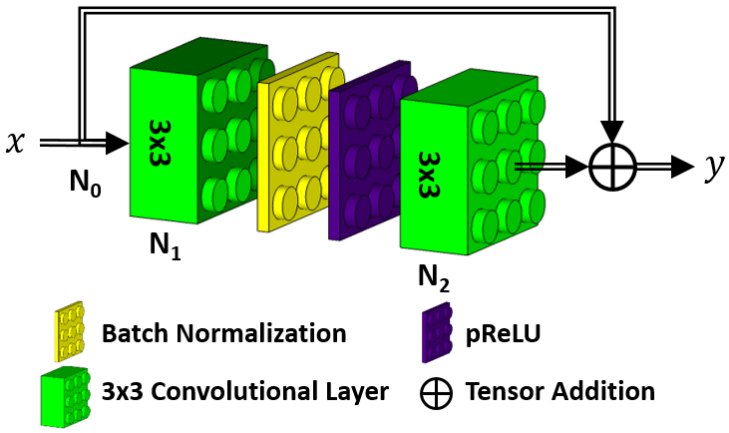


Fig. 5.4. Example DfD-Net Residual Block

the output of the convolutional filters. However, the identity path adds the global features of the input back to the locally transformed input. It is the merger of local and global features that give the residual block its improved performance, especially in a deep learning task that requires accurate spatial localization of features, like generating depth maps.

The second major difference between the DfD-Net and the U-Net is the use of a convolutional block to perform the downsampling. The original U-Net used the traditional max pooling strategy to perform the downsampling, in this case a 2x2 block with a stride of 2 in the horizontal and a stride of 2 in the vertical directions [17]. The max pooling operator works by taking the max pooling window and overlaying it on top of the output of the convolutional filter and extracting the maximum value within that block. The window is then moved exactly like a 2-D convolutional operator. This effectively becomes a feature downsampling operation. In contrast, the DfD-Net uses a 2x2 convolutional layer with a stride of 2x2 to perform the downsampling action. Compared to the max pooling downsampling layer, these convolutional downsamplers have the added benefit of learnable parameters in the form of the convolutional filter weights that perform the downsampling operations. This increases the model flexibility by including the number of filters used to perform the downsampling as a tunable

hyper-parameter. The upsampling method used in the U-Net architecture "consists of an upsampling of the feature map followed by a 2x2 up-convolution that halves the number of feature channels" [17]. However, in the DfD-Net, the upsampling is again solely performed by the use of a fractional 2x2 convolutional layer with a stride of 2x2. This has the same pros and cons of the convolutional downsampler.

The third difference between the two architectures is the number of levels. The original U-Net used 5 levels (4 of which perform image tensor downsampling by a factor of 2 in each dimension). The DfD-Net only uses 3 levels (2 of which perform image tensor downsampling by a factor of 2 in each dimension) to perform the depth map generation. Figure 5.5 shows the graphical representation of the DfD-Net architecture. The offset structure of the network diagram is designed to show that for each level the image tensors are the same dimensions due to the downsampling and upsampling operations. The numbers inside the convolutional blocks represent the number of filters for each layer. The smaller numbers under the blocks indicate the layer number as built in the Dlib framework [46].

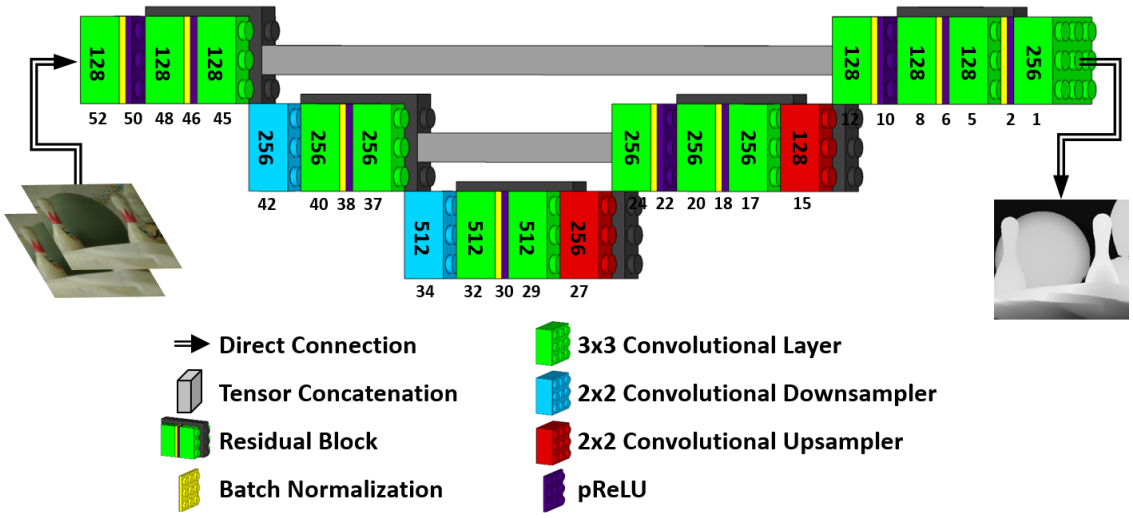


Fig. 5.5. Graphical Representation of the DfD-Net Network Architecture

The final difference between the U-Net and the DfD-Net is the number of input channels. The input for the U-Net, is a single grayscale image. However, the DfD

algorithm requires, at a minimum, two images as the input to the algorithm. Therefore, the DfD-Net also requires a minimum of two images, which is the equivalent to an input image with six channels. These two input images provide the contextual blur information for each of the three color channels, red (R), green (G) and blue (B) at the pixel level. The DfD-Net operates on the differences in the blur levels between the same color channels to develop the final depth map output.

Compared to the graph cuts method, where the RGB input images were converted to the YCrCb color space, YCrCb images were not used in the DfD-Net. This is because the YCrCb pixel is simply a linear combination of the RGB pixel and it is expected that the convolutional layers within the DfD-Net will learn a mapping that would convert either an RGB image or a YCrCB image to the same space. To illustrate this point, take the set of functions $\mathcal{H}(x)$ that convert an RGB pixel, (P_{RGB}), to a YCrCb pixel, (P_{YCrCb}) in Equation 5.13.

$$\mathcal{H}(P_{RGB}) = P_{YCrCb} \tag{5.13}$$

Applying a second function, $\mathcal{G}(x)$, to both sides of Equation 5.13 results in Equation 5.14, where $\mathcal{F}(x) = \mathcal{G}(\mathcal{H}(x))$.

$$\mathcal{F}(P_{RGB}) = \mathcal{G}(P_{YCrCb}) \tag{5.14}$$

This shows that the output of a function operating on a YCrCb pixel can be equal to the output of a set of functions operating on an RGB pixel. For this reason the YCrCb color space was not considered as an input to the DfD-Net.

Much like the U-Net, the DfD-Net can be thought of as a mapping function that maps the in-focus and out-of-focus image pairs, X to the label space L. The same high level mapping function that describes the U-Net mapping (Equation 5.1) can be used to describe the DfD-Net mapping. The only difference is the dimension of the input space, $X \epsilon \mathbb{R}^{6 \cdot 12^2}$, $x_i \, \epsilon \, [0, ..., 255]$. The input size is again determined by the receptive field size of the DfD-Net and the number of downsampling stages within

the network architecture. Because the DfD-Net gradually contracts the input, the minimum input size that results in a 3x3 receptive field at the lowest level, is 12x12 pixels and the multiplication by six represents the 6 channel input.

The output label space has also been expanded, whereas, the output of the U-Net was a single channel image representing $N = 2$ classes, the output of the DfD-Net is a single channel image containing $N = 256$ classes, therefore $L \epsilon [0, ..., 255]$. It is the combination of these changes that allow the DfD-Net to produce accurate depth maps for a given pair of input images.

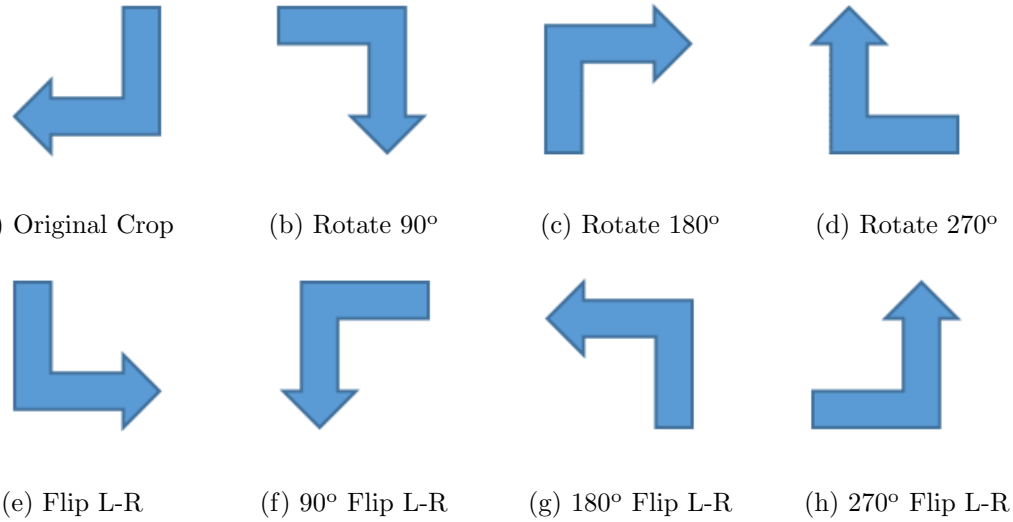
5.2 DfD-Net Training

The DfD-Net architecture was built using the Dlib - Machine Learning Toolkit [46]. The training of the network is performed much like the training for traditional deep learning architectures. The input into the network consists of the 3-channel RGB in-focus image and the 3-channel RGB out-of-focus image. The architecture works directly with the RGB color images, which is different from the prior graph cuts method which performed the best using the YCrCb color space. The ground truth for the training is the single channel grayscale depth map with values that range from 0 to 255.

5.2.1 Training Data Augmentation

In order to increase the amount of available training data to the training algorithm and to increase the generality of the network, a data expansion technique is required. Originally the technique used was a cropping mechanism that randomly picks images patches from the 6-channel input and then mirrors the image in the left-right direction (L-R flip) with a probability of 50% and then flips the resulting image in the up-down direction (U-D flip) with a probability of 50%. However, after further research it was found that this scheme was limited. For this reason a new enhanced cropping mechanism was developed. The new cropping method begins by randomly selecting

an image from the training set and then randomly selecting a 32x32 pixel image patch from that image. Once the image patch is selected the patch is rotated by 0, 90, 180 and 270 degrees. Then each rotation is mirrored in a left-right (L-R) flip. Figure 5.6 shows an example of this data expansion where one input image crop is expanded to eight training examples.



(a) Original Crop (b) Rotate 90° (c) Rotate 180° (d) Rotate 270°

(e) Flip L-R (f) 90° Flip L-R (g) 180° Flip L-R (h) 270° Flip L-R

Fig. 5.6. Enhanced Image Crop Data Expansion Example

A comparison of the two different data expansion methods is presented in Figure 5.7. For each cropping method the training step batch size was set so that each method provided the same number of examples per step. Aside from the data expansion method the training parameters for each expansion method are identical. The basic expansion method took over 48 hours to train, while the enhanced expansion method took just over 21 hours to train. The NRMSE, NMAE and SSIM test results for the basic expansion method was 0.0500, 0.1057 and 0.6590 respectively. For this training run, the enhanced expansion method produced a better resulting NRMSE, NMAE and SSIM of 0.0197, 0.0687 and 0.9007, respectively. What is interesting to note is that for the basic cropping method the model is over fitting, as seen by the divergence of the test and training results for each of the three metrics as training

progressed over time. Seen in Figure 5.7 with the solid lines. The enhanced method substantially eliminates overfitting, as seen in the dotted lines, and also converges in half the number of training steps.

Figure 5.8a shows the distribution of the random image selection process during an entire training event. Each image was selected on average 8753.74 times. The distribution is very uniform which is exactly what it should be to ensure that each scene is used equally during the training. Figure 5.8b and 5.8c shows the distribution of the pixels that were selected for the 127th image (least selected image) and the 42nd image (most selected image) in the training set. The heatmap images show the individual pixel usage in a given image. The more a particular pixel was used in the training the redder that pixel is, and the less the pixel was used the bluer the pixel.

In general this enhanced cropping method allows the algorithm to see more than is typically available during the training. Figure 5.9 shows the comparison between the distribution of the Middlebury ground truth data (blue) and the total number of times a particular depth map values was used during training (red). The average number of times the DfD-Net was trained with a given depth map value was approximately 422 times more than the number of times that depth map value is in the dataset.

5.2.2 Training Function

The output of the last layer of the network is a softmax layer which performs the classification. The error function used in the training is the cross-entropy loss function. The training is performed using the Adam optimization method developed by Kingma and Ba [47]. The Adam optimizer is a stochastic optimization method that has been proven to converge in less time and require less memory resources than the traditional stochastic gradient descent (SGD) method [47]. The optimizer has three hyper-parameters that can be defined at training time: 1) α, the weight decay step size, 2) β_1, an exponential decay rate factor and 3) β_2, a second exponential decay rate factor. The authors in [47] explain that "The algorithm updates exponential

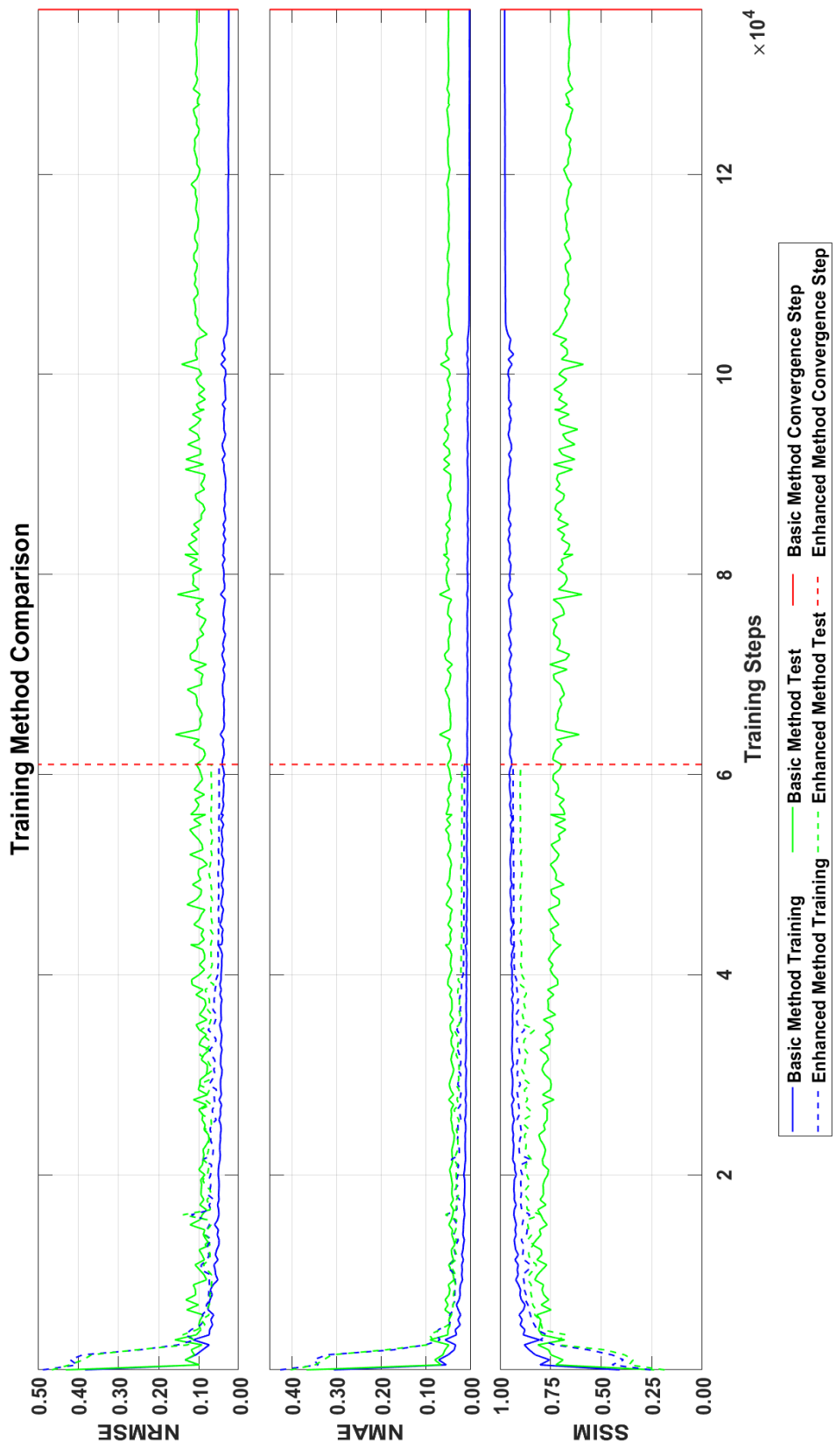
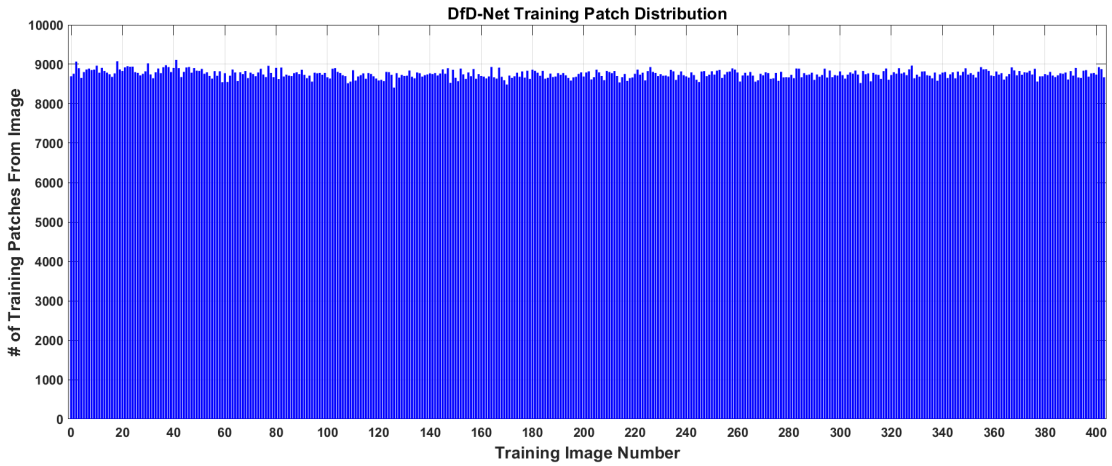
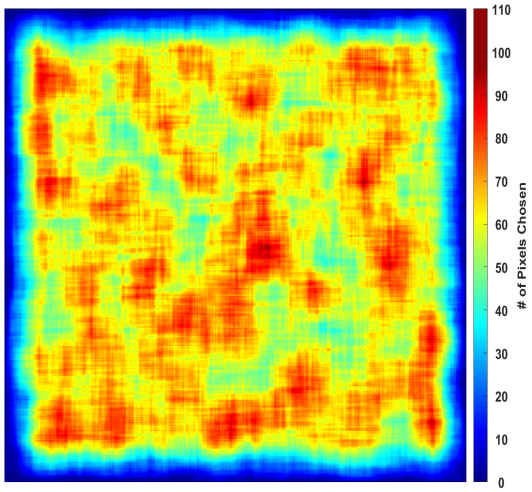


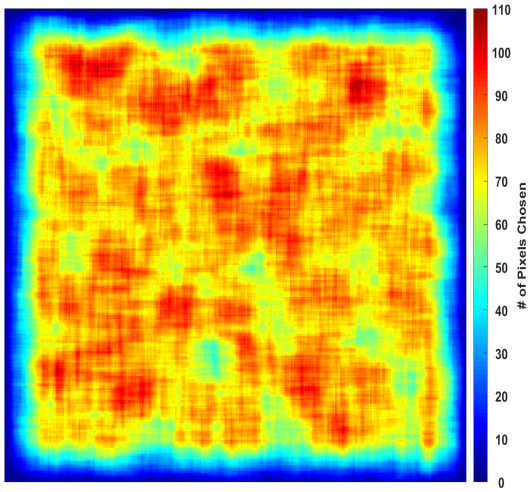
Fig. 5.7. Basic vs. Enhanced Training Method Comparison



(a) Random Image Selection Distribution



(b) Image 127 Pixel Selection Distribution



(c) Image 042 Pixel Selection Distribution

Fig. 5.8. Enhanced Cropper Distributions

moving averages of the gradient (m_t) and the squared gradient (v_t) where the hyper-parameters $\beta_1, \beta_1 \epsilon [0, 1)$ control the exponential decay rates of these moving averages. The moving averages themselves are estimates of the 1st moment (the mean) and the 2nd raw moment (the un-centered variance) of the gradient". These parameters were set to the following: $\alpha = 0.0005$, $\beta_1 = 0.5$ and $\beta_2 = 0.99$ for the DfD-Net

The initial learning rate was set to 0.0001 and the learning rate schedule is governed by Dlib's internal learning rate solver. The Dlib approach is better than the

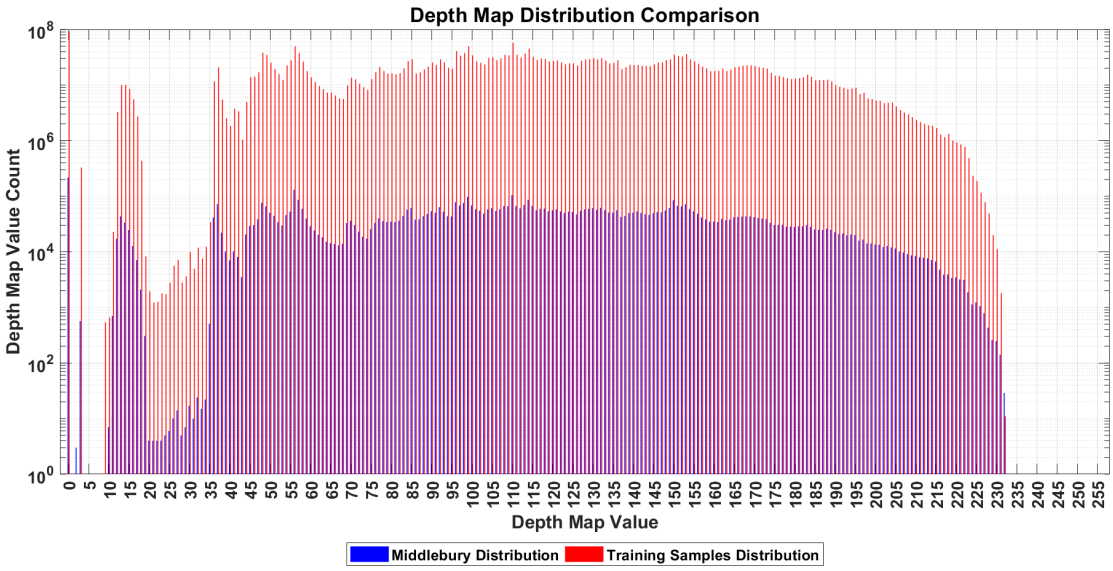


Fig. 5.9. Ground Truth vs. Training Depth Map Value Distribution Comparison

traditional approach because the learning rate does not need user intervention to check when the loss stops improving to move to a lower learning rate. Other methods just blindly pick an arbitrary number of iterations to train and then stop. The Dlib solver works by assuming that the loss per training iteration can be modeled as a time series dataset corrupted by Gaussian noise with a mean of 0 and a variance of σ^2 [48]. The ordinary least squares (OLS) function is used to estimate the slope of the loss values. Learning progress is made when the slope of the loss function is less than zero. The Dlib learning rate scheduler uses this approach with two user definable settings. The first is the number of training iterations where no learning progress is made, i.e. the slope of the loss is greater than zero, and the second is the learning rate reduction factor which is used to reduce the learning rate when the number of training iterations without progress is met. For the training performed in this research the number of training iterations without progress was set to 2500 and the learning rate reduction factor was set to 0.1. Which means that when there were 2500 consecutive training iterations where the loss did not decrease the current learning rate was multiplied by the reduction factor of 0.1. The training was stopped

when the learning rate reached a value of 1e-10. The average training time can range anywhere from 19 hours up to 160 hours depending on the architecture parameters and the image crop size.

5.3 Synthetic Blur Dataset Results

Just as important as the architecture is the data that is used to train the network. The dataset split between training and testing for the Middlebury College dataset [7,8] was done in a very deliberate manner. The guiding principle behind the placement into training or testing was based on the distribution of the depth maps. After analyzing the distribution of the depth map values, the images that were selected for the test set were such that their depth map value distribution was contained within the distribution of the training images. Figure 5.10 shows the depth map value distribution of the training images (blue) and the test images (red).

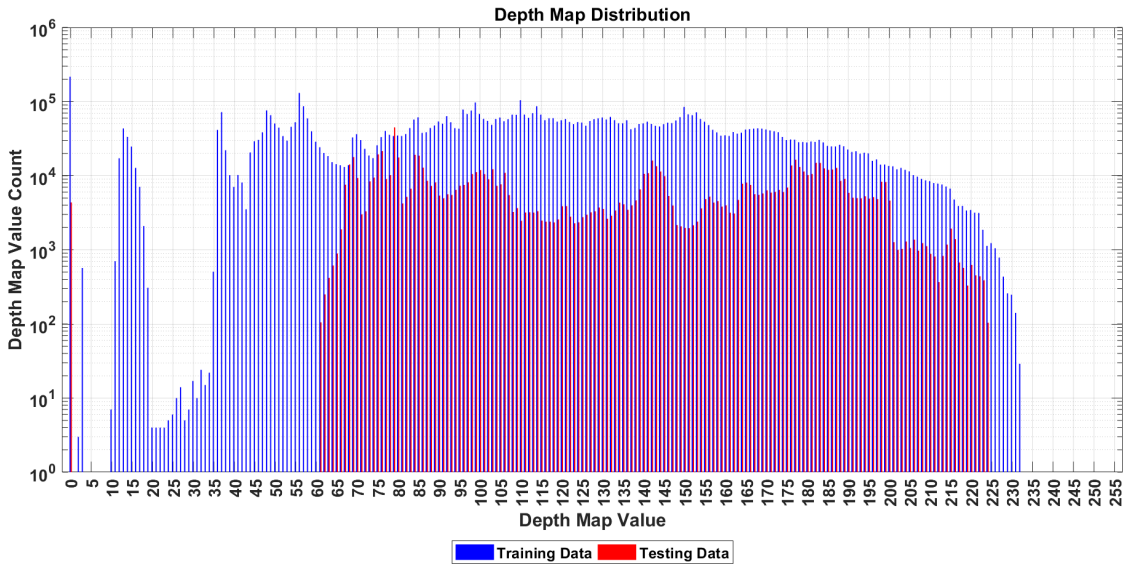


Fig. 5.10. Depth Map Value Distribution for Middlebury College Training and Testing Datasets

Table 5.1 lists the Middlebury College Stereo Vision [7,8] images that are used as the training examples for the DfD-Net and the testing examples. In total there were

404 image pairs in the training set and there were 54 image pairs in the test set. This represents an 88.2%/11.8% split for the training and testing data.

Table 5.1.
Middlebury Training and Testing Dataset Images

Training Dataset Images			
• Aloe	• Cloth 1	• Lampshade 1	• Monopoly
• Baby 1	• Cloth 2	• Lampshade 2	• Plastic
• Baby 2	• Cloth 2	• Laundry	• Rocks 1
• Baby 3	• Cloth 4	• Midd 1	• Rocks 2
• Bowling 1	• Dolls	• Midd 2	• Wood 1
• Bowling 2	• Flowerpots	• Moebius	• Wood 2
Testing Dataset Images			
• Art	• Books	• Reindeer	

Once the training has completed, the test dataset is run through the network architecture to determine its overall performance and the results are presented here. Figures 5.11, 5.12 and 5.13 shows the NMAE, NRMSE and the SSIM performance results for the images in the illumination 1, illumination 2 and illumination 3 categories respectively. From the graphs it can be seen that the DfD-Net performance across each image is very consistent for each of the evaluation metrics. This indicates that the DfD-Net's performance is not dependent on the illumination or exposure levels as compared to what was seen in the prior graph cuts algorithm results.

Table 5.2 lists the top 5 image pairs where the DfD-Net method performed the best and the bottom 5 image pairs where the DfD-Net method performed the worst. Once again the primary metric used for ranking the results is based on the NRMSE metric and the NMAE metric is used as a tie breaker. Unlike the performance deviation of the graph cuts method the DfD-Net performance deviation is much tighter.

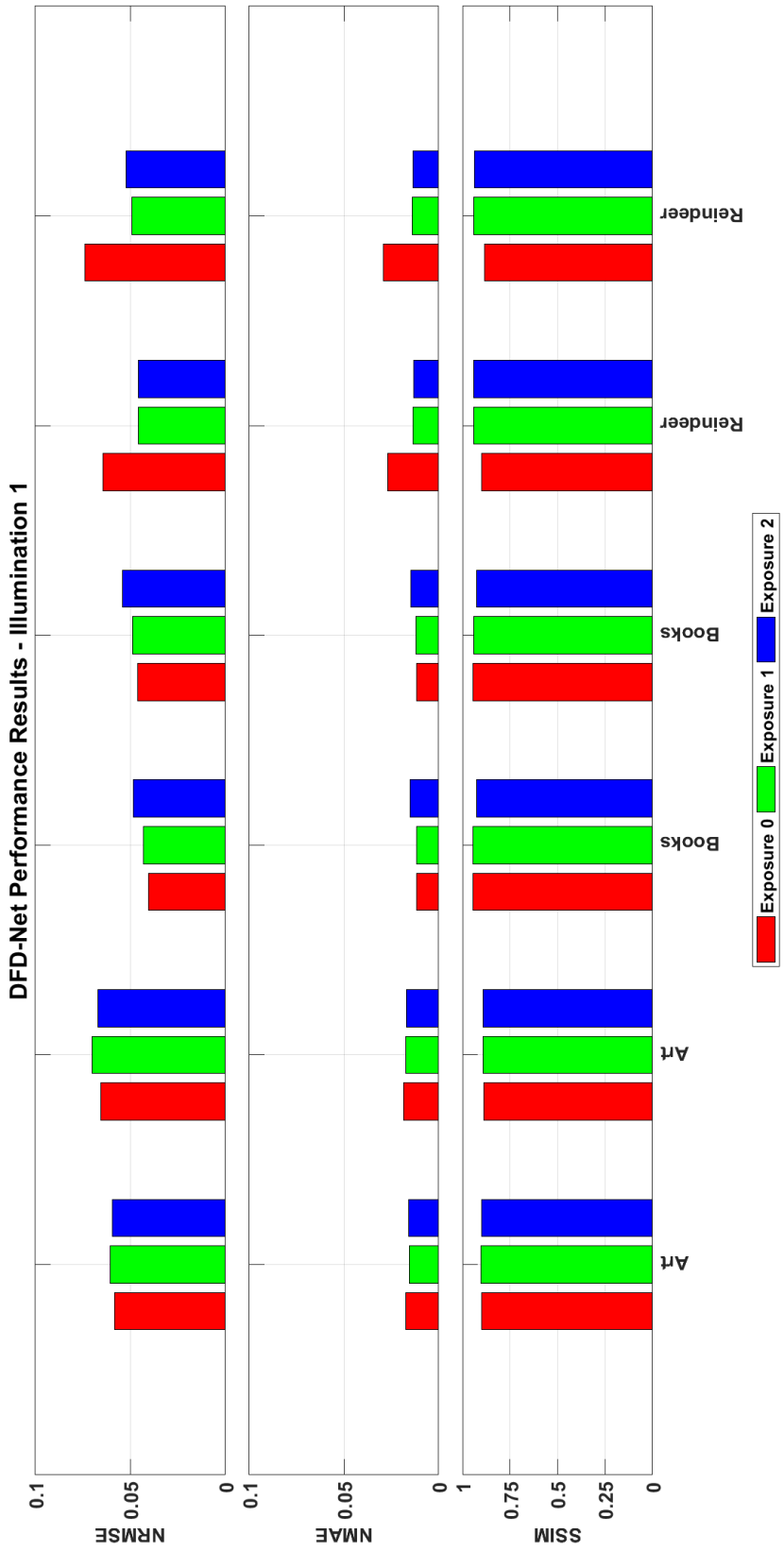


Fig. 5.11. DfD-Net Performance Results for Middlebury College Dataset - Illumination 1

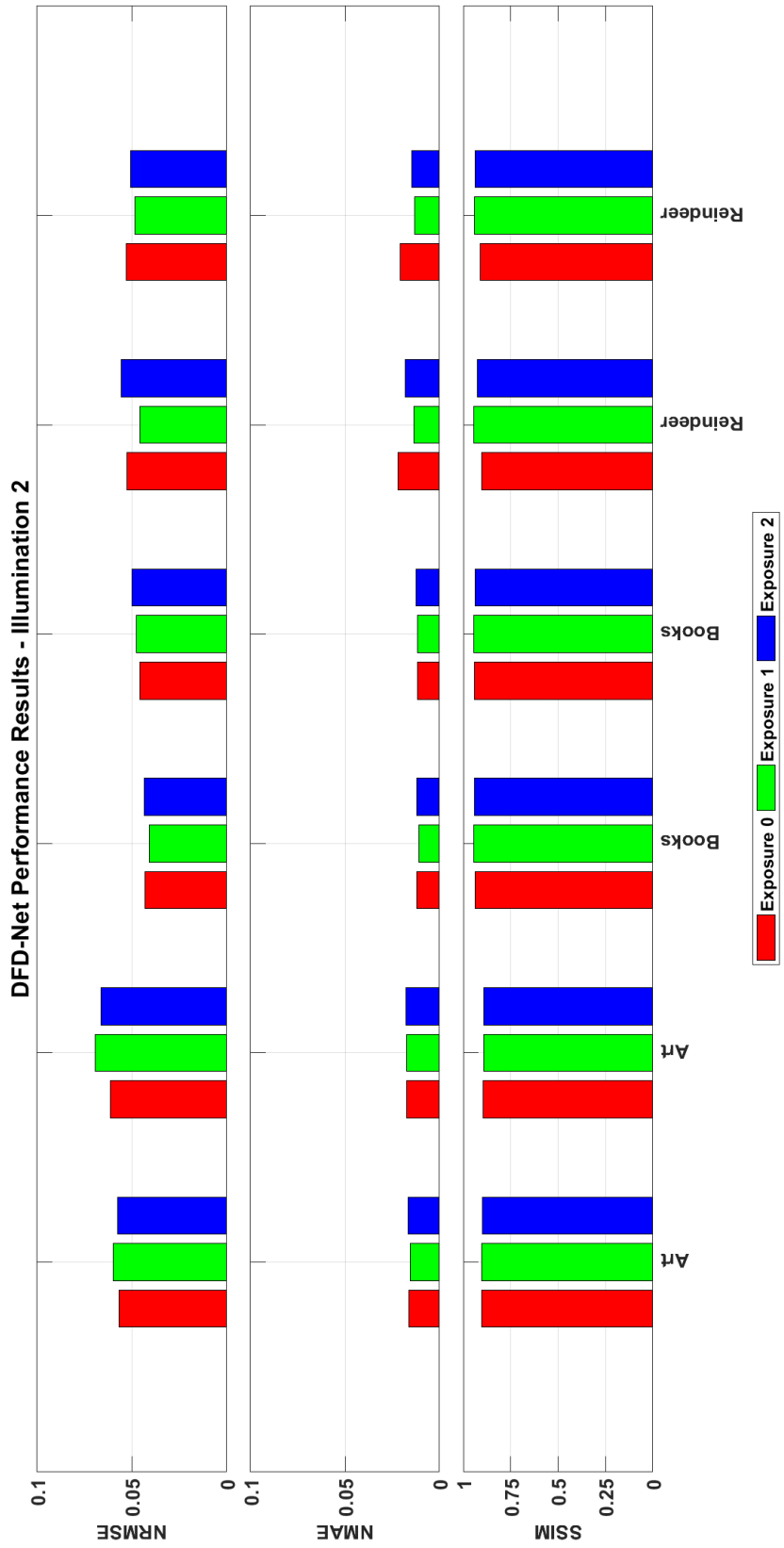


Fig. 5.12. DfD-Net Performance Results for Middlebury College Dataset - Illumination 2

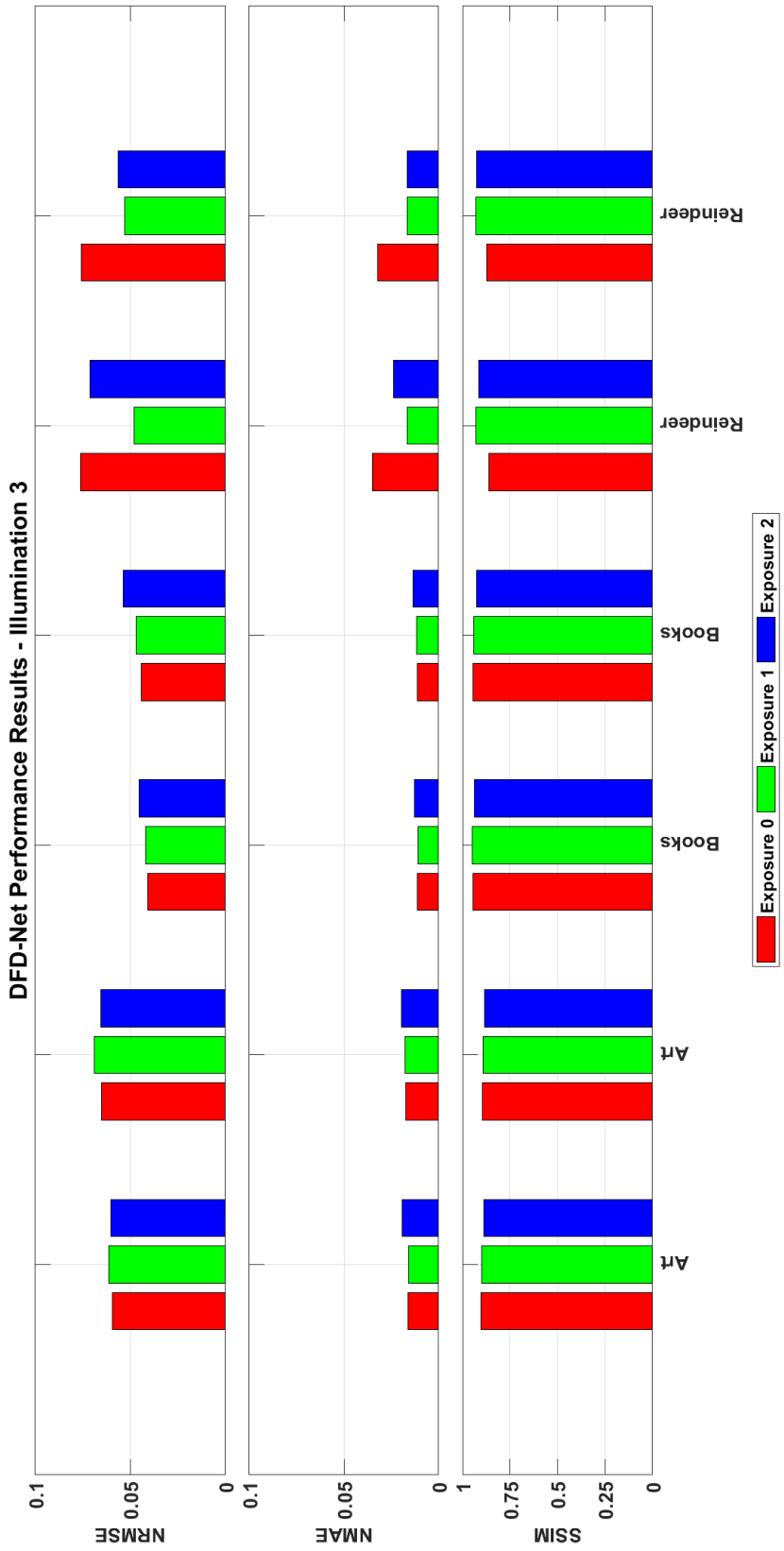


Fig. 5.13. DfD-Net Performance Results for Middlebury College Dataset - Illumination 3

Table 5.2.
Top 5 and Bottom 5 DfD-Net Performance Results for the Middlebury College Stereo Vision Dataset

Name	**View**	**Illumination**	**Exposure**	**NRMSE**	**NMAE**	**SSIM**
Top 5						
Books	Left	1	0	0.0406	0.0118	0.9447
Books	Left	3	0	0.0409	0.0111	0.9458
Books	Left	2	1	0.0410	0.0109	0.9452
Books	Left	3	1	0.0420	0.0107	0.9475
Books	Left	2	0	0.0430	0.0122	0.9391
Bottom 5						
Art	Right	1	1	0.0701	0.0174	0.8915
Reindeer	Left	3	2	0.0712	0.0240	0.9157
Reindeer	Right	1	0	0.0738	0.0293	0.8845
Reindeer	Right	3	0	0.0758	0.0324	0.8736
Reindeer	Left	3	0	0.0763	0.0349	0.8608

Table 5.3 outlines the mean and standard deviation for each illumination and exposure level combination for each of the evaluation metrics. The table also shows the overall combined mean and standard deviation for each exposure level in the dataset. The small standard deviations for each metric indicate that the results are holding very close to the means of each metric. The consistency in the illumination level mean values between exposure levels indicates that the DfD-Net does not have performance problems with low lighting conditions.

Figure 5.14 shows the top 5 and bottom 5 performance results for the DfD-Net. The four images on the upper left represent the highest performing results with an NRMSE, NMAE and SSIM of 0.0406, 0.0118 and 0.9447, respectively. While the four images on the lower right indicate the worst performance results with an NRMSE, NMAE and SSIM of 0.0763, 0.0349 and 0.8608 respectively. With an average score of

0.0552, 0.0163 and 0.9180 for the NRMSE, NMAE and SSIM, respectively. It can be seen that the resulting depth maps are very blotchy for the worst depth map results.

Table 5.3.
DfD-Net Performance Mean & Standard Deviation for the Middlebury College Stereo Vision Dataset

		NRMSE		**NMAE**		**SSIM**	
Lighting		**Mean**	**Std**	**Mean**	**Std**	**Mean**	**Std**
Exp 0	Illum 1	0.0582	0.0126	0.0192	0.0075	0.9095	0.0276
	Illum 2	0.0522	0.0068	0.0166	0.0042	0.9160	0.0191
	Illum 3	0.0603	0.0152	0.0205	0.0105	0.9040	0.0356
	Overall	0.0569	0.0119	0.0188	0.0076	0.9098	0.0270
Exp 1	Illum 1	0.0529	0.0103	0.0140	0.0022	0.9274	0.0238
	Illum 2	0.0520	0.0105	0.0136	0.0024	0.9280	0.0243
	Illum 3	0.0534	0.0101	0.0149	0.0030	0.9227	0.0231
	Overall	0.0528	0.0097	0.0142	0.0024	0.9260	0.0224
Exp 2	Illum 1	0.0546	0.0078	0.0148	0.0014	0.9204	0.0205
	Illum 2	0.0541	0.0078	0.0152	0.0027	0.9215	0.0224
	Illum 3	0.0588	0.0091	0.0176	0.0042	0.9128	0.0222
	Overall	0.0558	0.0081	0.0159	0.0031	0.9182	0.0208

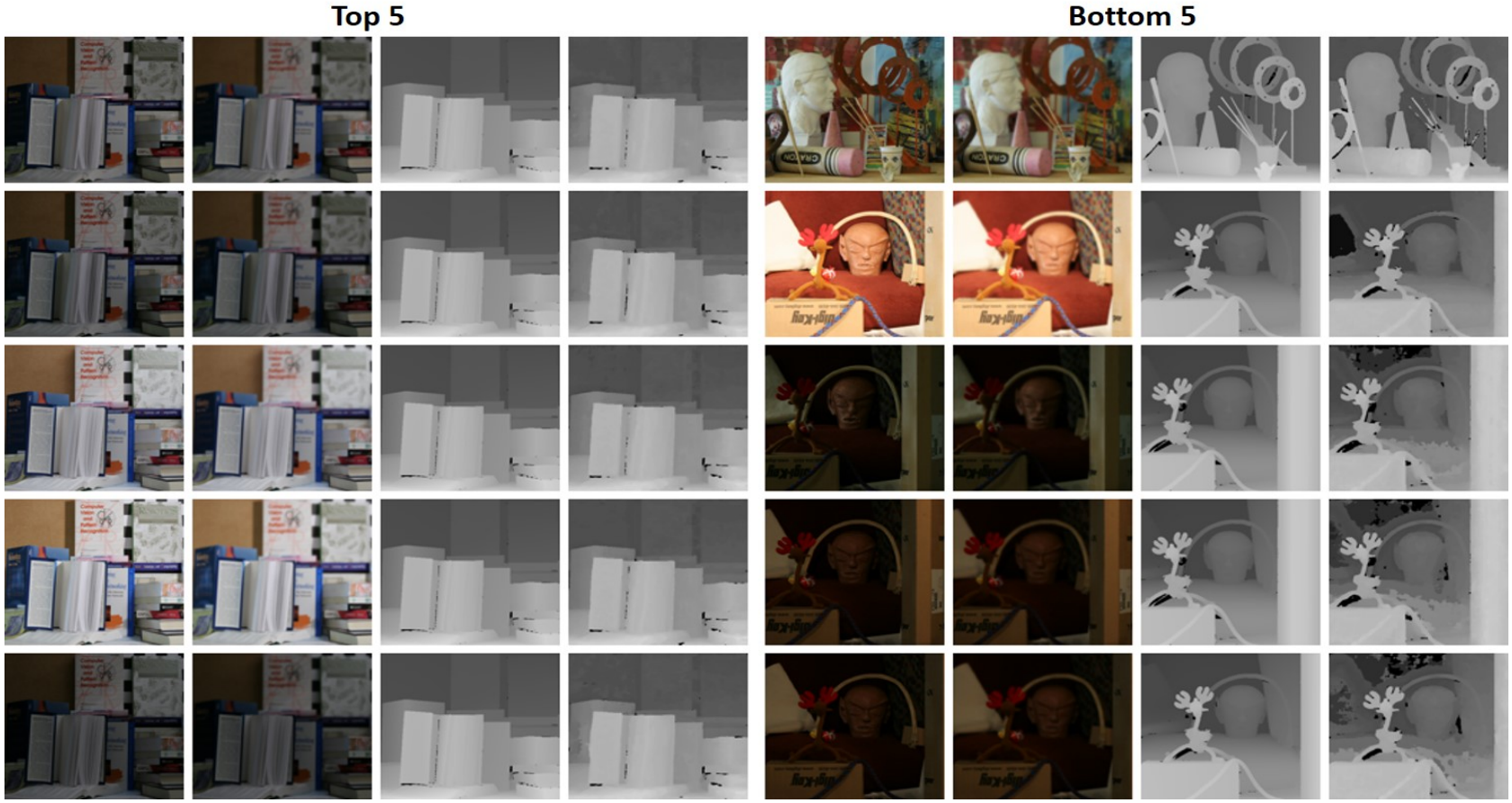


Fig. 5.14. Top 5 and Bottom 5 Performance Results for the Middlebury College Stereo Vision Dataset. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Ground Truth Depth Map, (d) & (h) DfD-Net Computed Depth Map.

Table 5.4 shows the average run times for each combination of illumination and exposure level. The timing analysis was done on both a single CPU and an NVIDIA® GTX-1080 GPU. The hardware used for both the CPU on and GPU test was the Desktop 2 configuration outlined in Table B.1. The DfD-Net run time does not depend on the illumination levels or exposure levels, which is in contrast to the graph cuts algorithm, Table 4.4, which was extremely dependent on the illumination and exposure levels.

Table 5.4.
Average DfD-Net Run Time for the Middlebury College Dataset

	CPU Run Time (s)			**GPU Run Time (s)**		
Lighting	**Exp 0**	**Exp 1**	**Exp 2**	**Exp 0**	**Exp 1**	**Exp 2**
Illumination 1	9.961	10.155	10.061	0.396	0.396	0.396
Illumination 2	10.000	10.050	10.079	0.396	0.395	0.396
Illumination 3	9.981	9.996	9.981	0.397	0.396	0.396

Since the DfD-Net is based on the U-Net architecture, a training run was performed using the U-Net along with all of the same training parameters used in the DfD-Net training. Table 5.5 shows the comparison of the mean performance numbers between the graph cuts method, the U-Net architecture and the DfD-Net architecture for each of the metrics and each exposure/illumination level. The DfD-Net has such a limited test set, so the comparison is only made between the image sets listed in the Test Image section of Table 5.1. For each exposure and illumination level the DfD-Net outperforms the graph cuts method in both the NRMSE and NMAE metrics. The graph cuts method, however, does produce better SSIM results for exposure level 1 and 2 and exposure level 0 - illumination level 3. However, referring back to Table 3.5 and Figure 3.14 the example depth map with noise produced worse SSIM results as compared to the solid image. The DfD-Net produces depth maps that have more noise, which can drive the SSIM metric lower versus the graph cuts method

that produces depth maps that can be more singular in value. Figures 5.15, 5.16 and 5.17 show the performance comparisons between the DfD-Net architecture and the graph cuts algorithm for each of the three illumination levels. The original U-Net also performed better than the graph cuts method when comparing the NMAE metric, but did not out perform the graph cuts or DfD-Net in the NRMSE or SSIM metrics.

Table 5.5.
Graph Cuts, U-Net & DfD-Net Average Performance Comparison

Lighting		**Graph Cuts**			**U-Net**			**DfD-Net**		
		NRMSE	**NMAE**	**SSIM**	**NRMSE**	**NMAE**	**SSIM**	**NRMSE**	**NMAE**	**SSIM**
Exp 0	Illum 1	0.0998	0.0800	0.9066	0.1224	0.0615	0.7616	**0.0582**	**0.0192**	**0.9095**
	Illum 2	0.0952	0.0756	0.9127	0.1191	0.0587	0.7725	**0.0522**	**0.0166**	**0.9160**
	Illum 3	0.1080	0.0860	**0.9222**	0.1125	0.0560	0.7676	**0.0603**	**0.0205**	0.9040
	Overall	0.1010	0.0806	**0.9138**	0.1180	0.0588	0.7673	**0.0569**	**0.0188**	0.9098
Exp 1	Illum 1	0.0765	0.0652	**0.9543**	0.1098	0.0543	0.7732	**0.0529**	**0.0140**	0.9274
	Illum 2	0.0752	0.0640	**0.9568**	0.1072	0.0517	0.7749	**0.0520**	**0.0136**	0.9280
	Illum 3	0.0787	0.0673	**0.9574**	0.1107	0.0532	0.7760	**0.0534**	**0.0149**	0.9227
	Overall	0.0768	0.0655	**0.9562**	0.1092	0.0531	0.7747	**0.0528**	**0.0142**	0.9260
Exp 2	Illum 1	0.0727	0.0630	**0.9610**	0.0920	0.0496	0.7852	**0.0546**	**0.0148**	0.9204
	Illum 2	0.0747	0.0644	**0.9603**	0.0956	0.0486	0.7861	**0.0541**	**0.0152**	0.9215
	Illum 3	0.0747	0.0637	**0.9576**	0.0972	0.0511	0.7817	**0.0588**	**0.0176**	0.9128
	Overall	0.0741	0.0637	**0.9597**	0.0949	0.0498	0.7843	**0.0558**	**0.0159**	0.9182

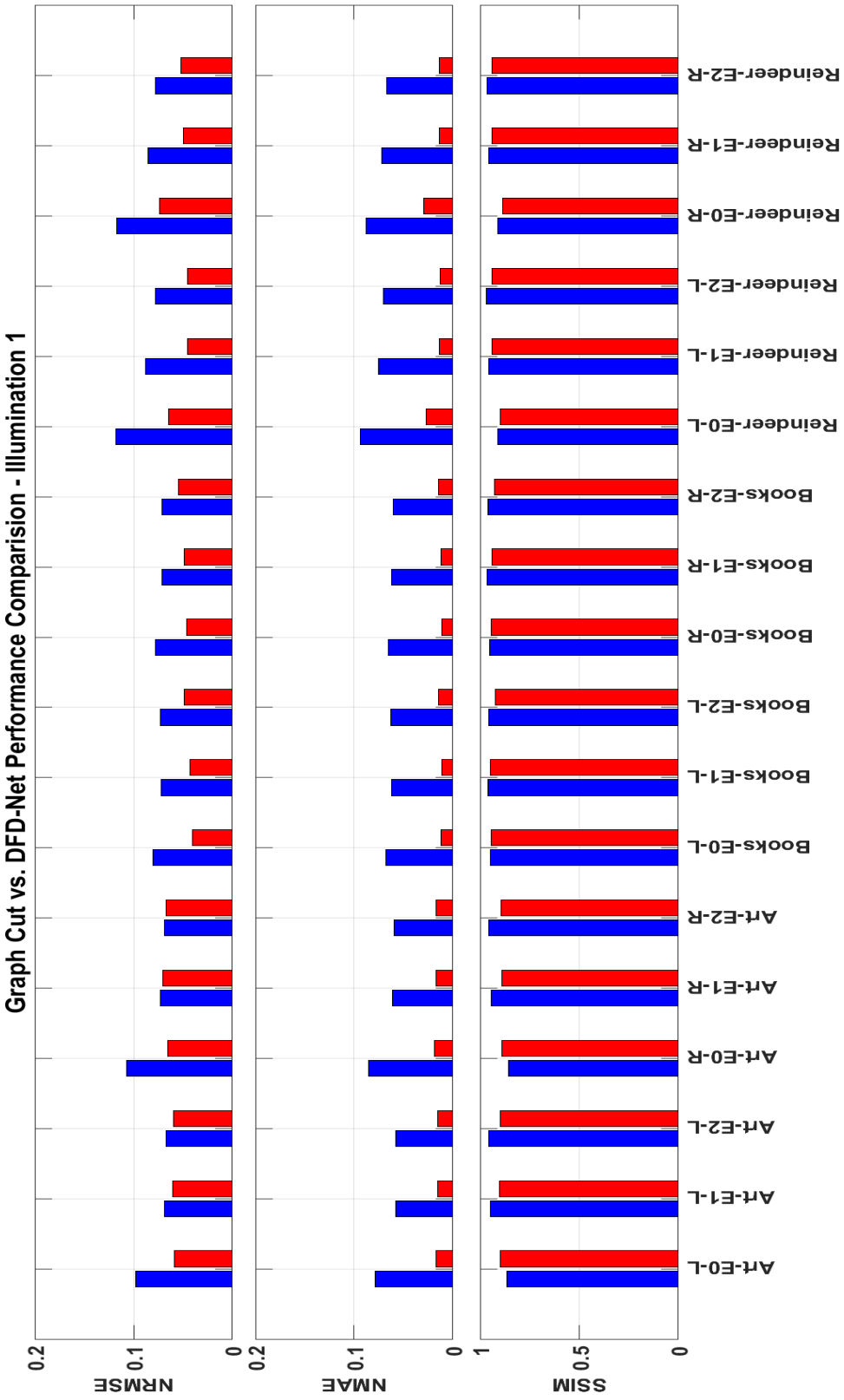


Fig. 5.15. Illumination Level 1: DfD-Net & Graph Cuts Performance Comparison

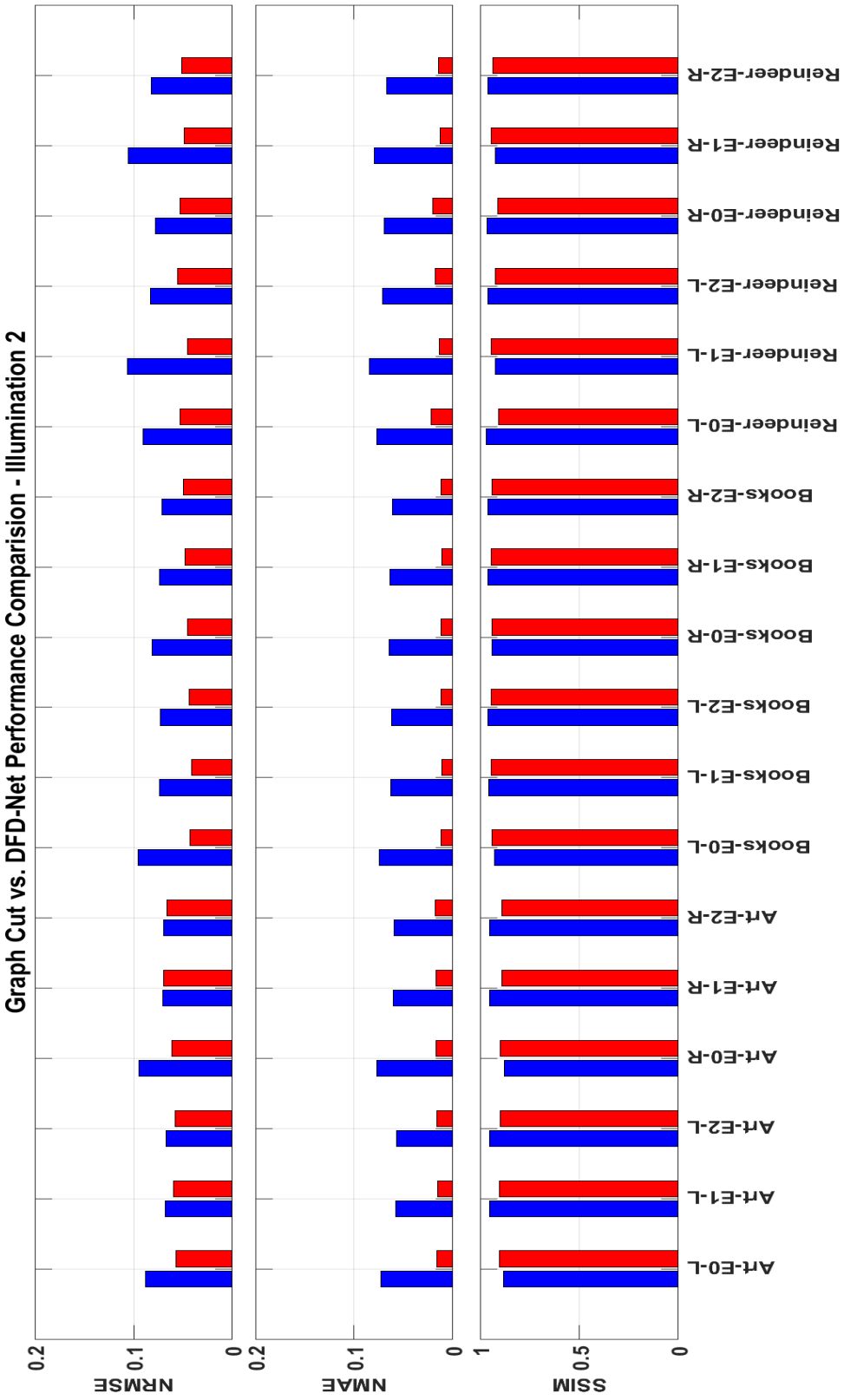


Fig. 5.16. Illumination Level 2: DfD-Net & Graph Cut Performance Comparison

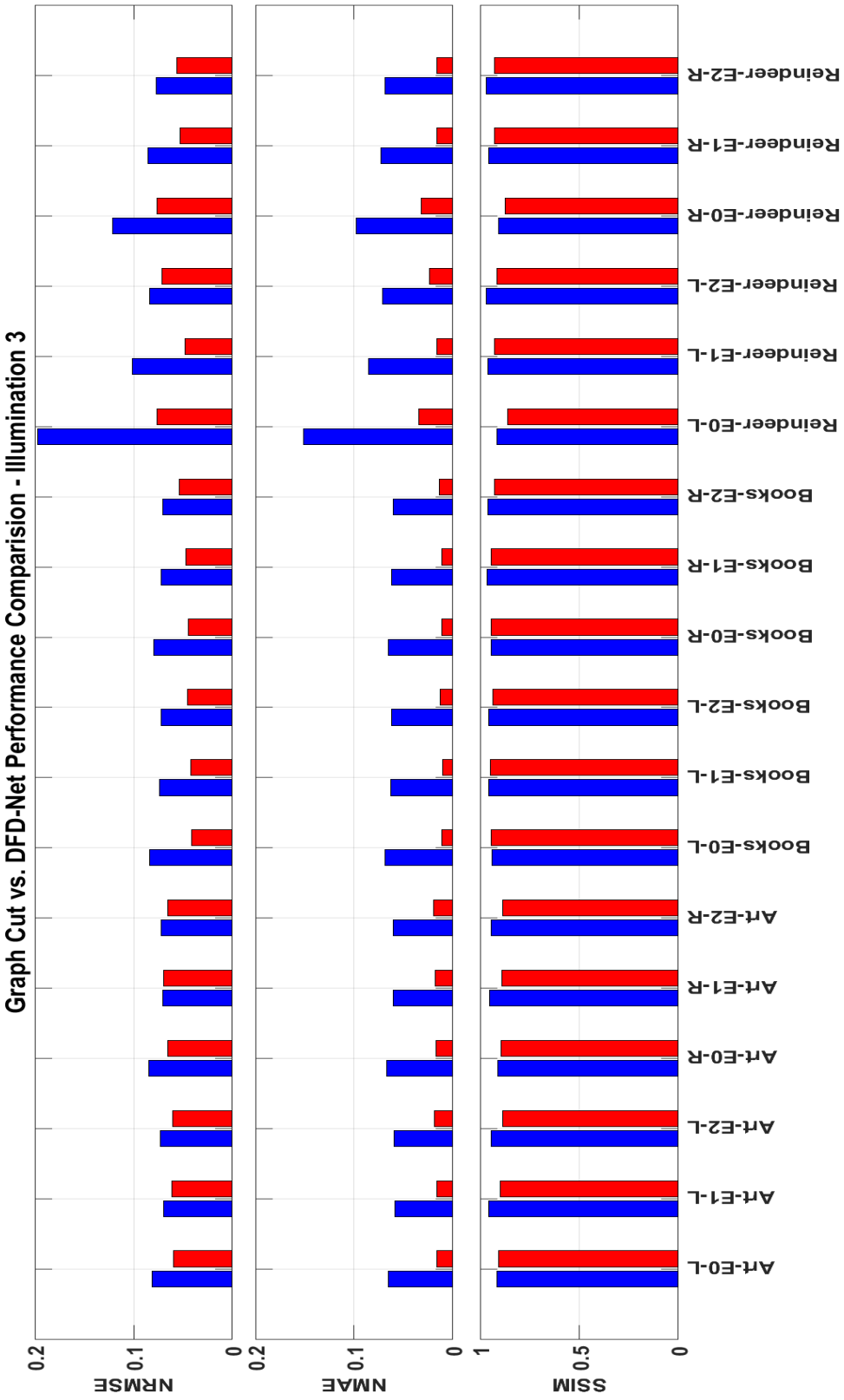


Fig. 5.17. Illumination Level 3: DfD-Net & Graph Cut Performance Comparison

5.4 Architecture Confidence Testing

Due to the stochastic nature of the weight initialization process, the training input image patch selection and the Adam optimizer, a series of tests were conducted to determine if the initial results are truly indicative of the network architecture's capability and not because the random processes perfectly aligned. The two tests that were performed are: a training repeatability test and a k-fold cross validation test.

5.4.1 Training Repeatability

The training repeatability test was performed to assess how repeatable the training process is. This test repeatedly trained the DfD-Net from scratch 30 times. Figure 5.18 shows the distribution for each of the metrics. The light blue area represents $\pm 2\sigma$ from the mean (dark blue dashed line) of the test results and the black dots are the individual test results from each of the training events. Table 5.6 shows the overall minimum, mean, maximum and standard deviation of the test results for the 30 training events. The only test that was not within the bounds was trial 18. This test produced performance results that exceeded the 2σ criteria.

Table 5.6.
Repeatability Trials Test Performance Results

	NRMSE	**NMAE**	**SSIM**
Minimum	0.0552	0.0163	0.9000
Mean	0.0632	0.0183	0.9075
Maximum	0.0699	0.0200	0.9180
Std Deviation	0.0039	0.0009	0.0045

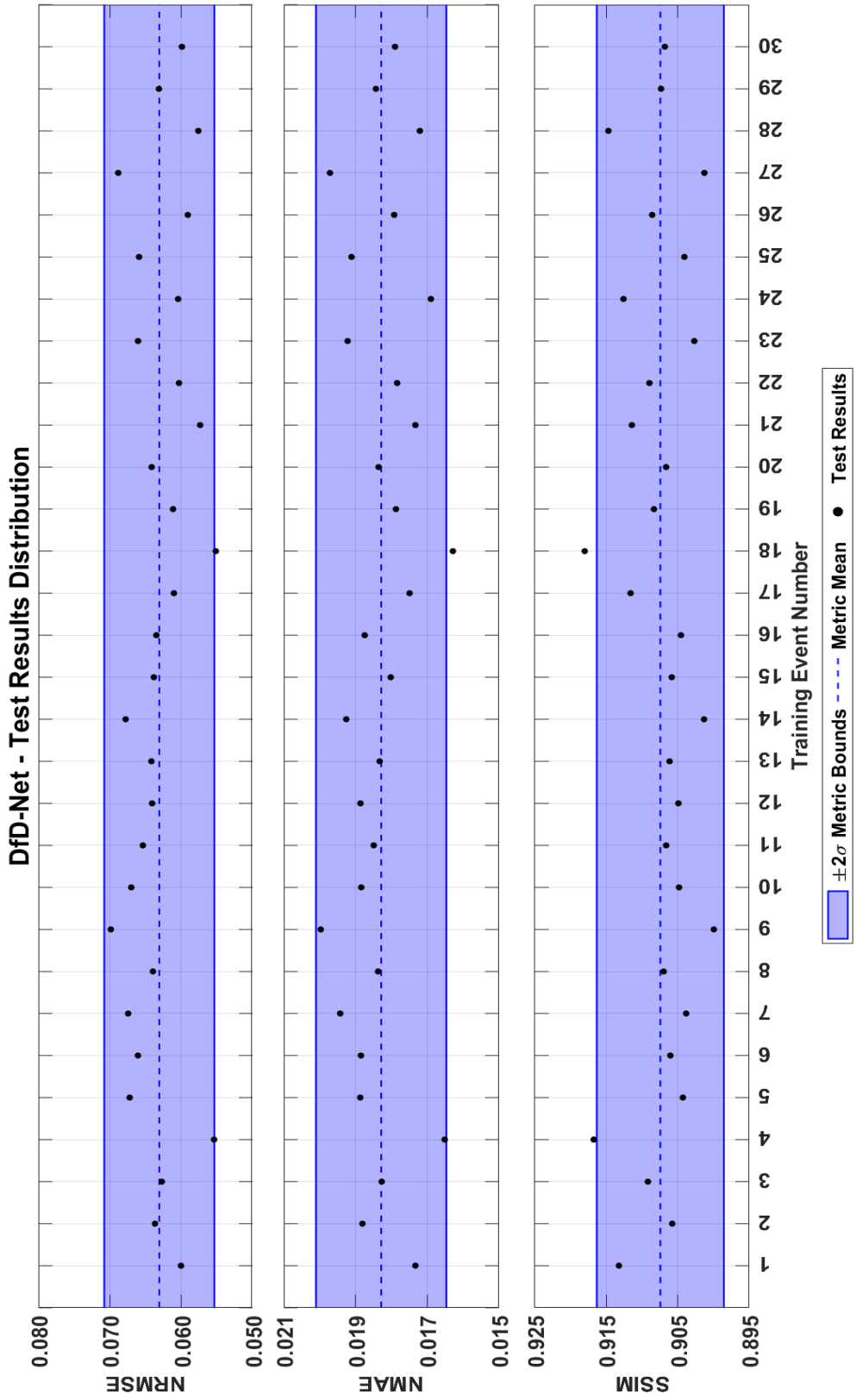


Fig. 5.18. DfD-Net Multi-Training Event Test Results Distribution

5.4.2 9-Fold Cross Validation

The k-fold cross validation test breaks the datasets into k groups where $k-1$ groups are used as the training dataset and one group is used as the validation test set. For each "fold" one group is selected as the validation set and the other $k-1$ groups are the training set. For the Middlebury College dataset k was set to nine. Table 5.7 shows the combined results of each trial. The "Test Data Scene" column indicates which scenes were used as the validation test set. When comparing the results of each of the folds, the test set that exhibits a significant reduction in performance is the K03 set. To better understand why this particular set performed far worse than the other sets an examination of the depth map distributions for both the training and test sets is required. Figure 5.19 shows the overall training and testing depth map distribution. The inset shows the areas where the number of depth map values in the testing distribution exceeds the training distribution. The box labeled "1" is the under represented distribution for the Midd2 scene.

Table 5.7.
DfD-Net Middlebury Synthetic Dataset 9-Fold Cross Validation Performance Results

Test	**Test Data Scene**	**NRMSE**	**NMAE**	**SSIM**
K01	Reindeer, Wood1, Wood2	0.04854	0.01878	0.9205
K02	Monopoly, Rocks1, Rocks2	0.06844	0.02304	0.9120
K03	Midd1, Midd2, Plastic	0.17032	0.11322	0.7204
K04	Lampshade1, Lampshade2, Moebius	0.07570	0.03451	0.8809
K05	Dolls, Flowerpots, Laundry	0.06131	0.01880	0.9002
K06	Cloth2, Cloth3, Cloth4	0.04298	0.01335	0.9459
K07	Bowling1, Bowling2, Cloth1	0.04755	0.02066	0.9007
K08	Baby1, Baby2, Baby3	0.08727	0.03220	0.9002
K09	Aloe, Art, Books	0.06106	0.01785	0.9018

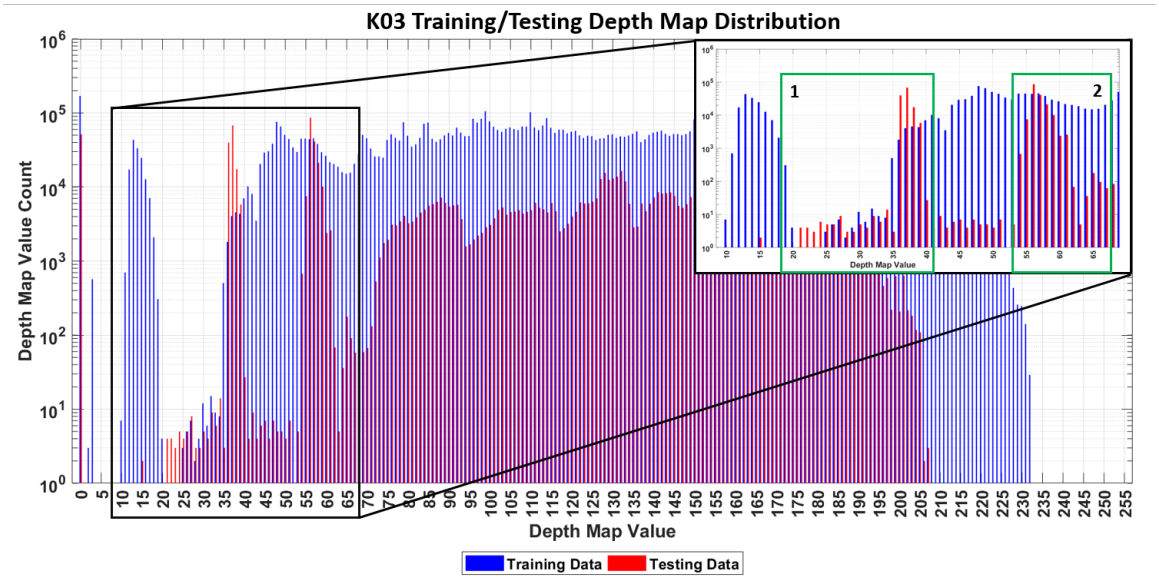
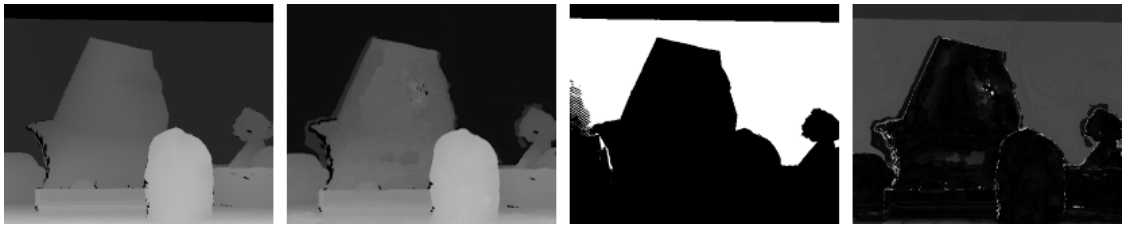


Fig. 5.19. K03 Cross Validation Training & Testing Depth Map Distribution

Figure 5.20 shows an example of the test results for the Midd2 dataset. Figure 5.20a is the ground truth depth map for the scene, Figure 5.20b is the resulting depth map determined by the trained DfD-Net. Figure 5.20c is a mask highlighting the location of the depth map values 35-38 that are underrepresented in the training set, and Figure 5.20d shows the resulting absolute error difference between the ground truth and the DfD-Net depth maps. The brighter areas indicate where the largest error differences are located. Comparing the mask and the error it can be seen that as expected, the largest depth map errors occur where the underrepresented depth map samples are located.

The 9-fold cross validation test results show that the architecture is robust across various training/test set combinations, even with the stochastic nature of the training process. The results of the 9-fold cross validation show that all but one fold produces similar results to the initial training described in Section 5.4.1. The one fold that produced a poorly trained network was due to the fact that the training samples for the most heavily concentrated depth map values were under represented in the training process which means that the network did not see enough examples to accurately



(a) (b) (c) (d)

Fig. 5.20. K03 Cross Validation Midd2 Testing Depth Map Comparison. (a) Ground truth, (b) DfD-Net depth map, (c) Mask of under represented values and (d) depth map error

recreate the given depth values for the test dataset. However, the training samples that were well represented in the dataset were accurately recreated as evident in the depth map error shown in Figure 5.20d.

5.4.3 Training Patch Size Analysis

Throughout various phases of testing to determine the best architecture, it was discovered that the training image patch size affected how well the final trained network performed on the test dataset. Once this trend was discovered a secondary exploration into the effect of patch size vs. network performance was conducted. Figure 5.21 outlines the DfD-Net performance on the test data for various patch sizes used during training. Similar to Figure 5.18, the light blue area represents the DfD-Net test distribution. From the figure it can be seen that the 32x32 patch size produces the best results for each of the three metrics as compared to the other patch sizes. Although based on the test result distribution discussed in Section 5.4.1 it may also be possible, with more training vignettes, for the 24x24, 28x28, 36x36 and/or the 40x40 patch sizes to produce similar results to the 32x32 image patch size.

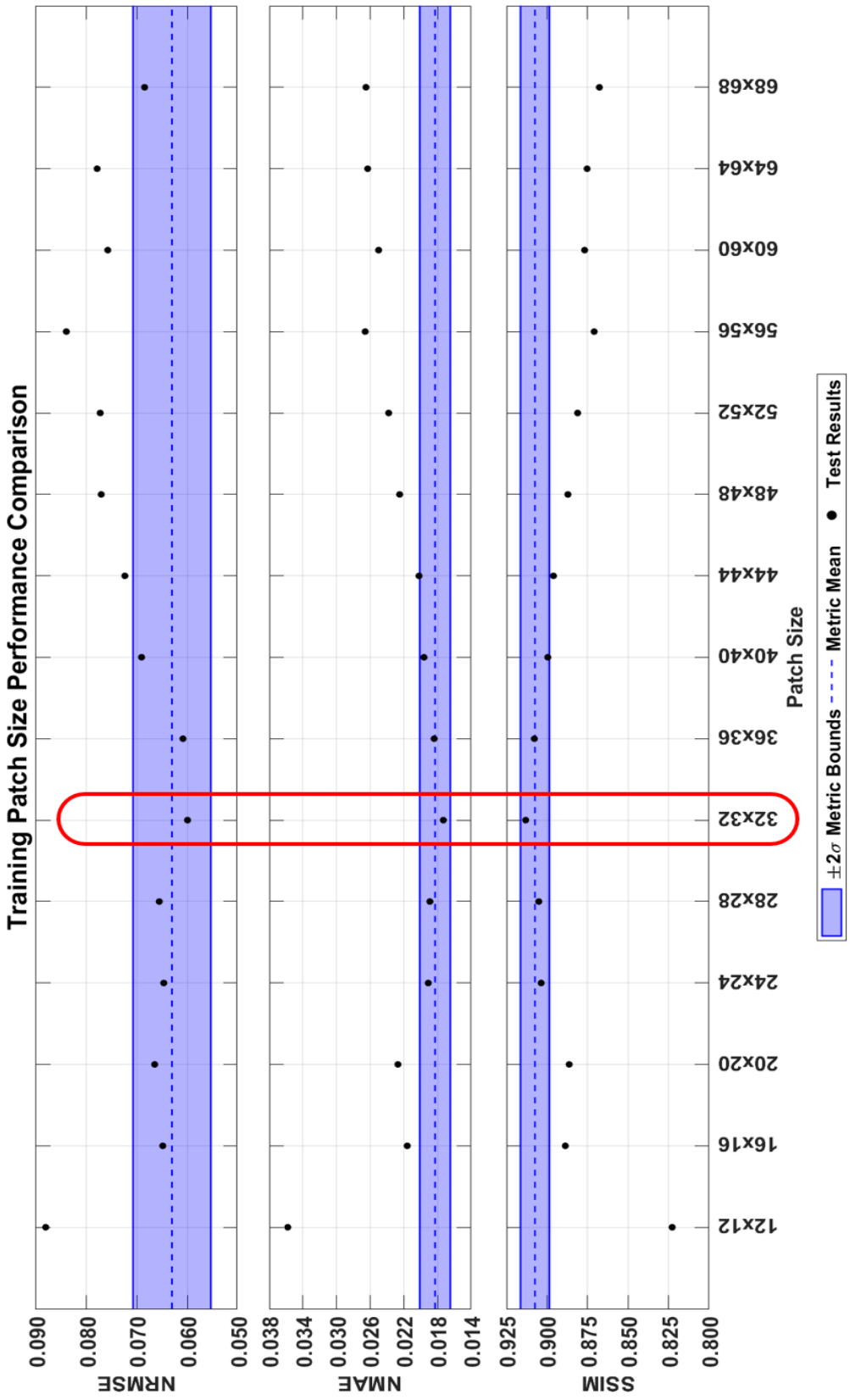


Fig. 5.21. DfD-Net Training Patch Size Performance Results

5.4.4 Training With Noise Analysis

To determine the robustness of the DfD-Net with respect to noisy images the DfD-Net was trained with various degrees of noise. Gaussian noise with a mean of zero, and three separate standard deviations, ($\sigma = \{1, 2, 3\}$) was added to both focus images simulating camera noise. The noise was added during the patch selection process outlined in Section 5.2.1. Once the image patch was selected noise was added to each pixel on each of the six input channels. Each noise scenario was trained independently 10 times to get the distribution of the test accuracy for a given noise input.

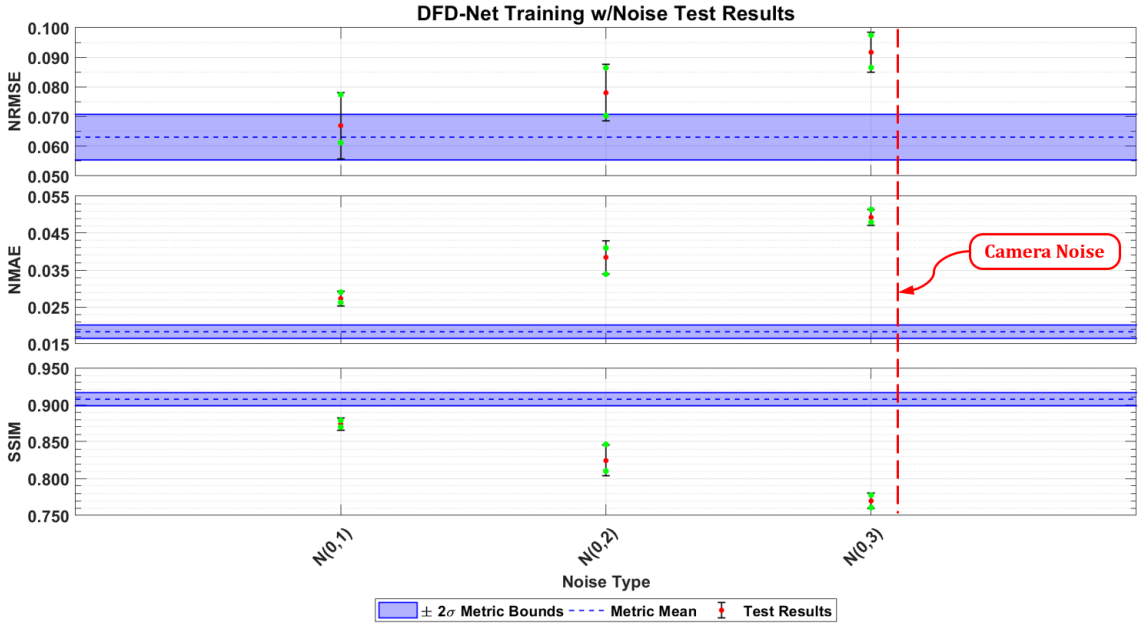


Fig. 5.22. DfD-Net Test Results with Various Noise Added During Training

Figure 5.22 shows the results of training with the three noise levels. Similar to Figure 5.18, the light blue area represents the DfD-Net test distribution when trained with no noise. The red dots indicate the mean test accuracy and the bars represent $\pm 2\sigma$ from the mean. The green dots represent the minimum and maximum test accuracy for the 10 training vignettes. This figure shows that the performance of

the DfD-Net degrades with increasing noise used during training. To compare, the Chameleon3 real world data collection camera configured according to Table 3.3 an image with an average pixel value of approximately 127 has a noise level that can be approximated by a normal distribution, $\mathcal{N}(0, 3.1)$.

5.4.5 Up/Down Sampling Filter Size

In a traditional image processing spatial aliasing is a concern when an image is either downsampled or upsampled. In the case of downsampling high frequency content within the image may appear to be at a lower frequency. To alleviate this aliasing a low pass filter should be applied to the image prior to downsampling. Furthermore, when upsampling an image the results should be passed through a low pass filter.

There is a concern that aliasing may occur within the DfD-Net as a result of the downsampling and upsampling convolutional filter size at each level in the architecture. To determine if the sampling filter size impacts the performance of the DfD-Net several different filter sizes were tested. For each filter size scenario, the DfD-Net was trained independently 10 times to get the distribution of the test accuracy. Figure 5.23 shows the results of testing various convolutional filter sizes. The same size filter was used for both downsampling and upsampling the tensors within the network. The light blue area represents the DfD-Net test distribution for the 2x2 filter sizes.

From these results it can be seen that deviating from the 2x2 filter size does not produce a significant improvement. For the NRMSE and NMAE the average results were inline with the baseline average results for each filter size except the 6x6 filter size which had a larger average error for each metric. The results of the SSIM metric follow the same trend as the other two metrics with the exception that the 1x1 filter was slightly worse than the 3x3, 4x4, and 5x5 filter sizes. This indicates that if any aliasing occurs during the downsampling and upsampling it does not degrade the depth map generation process.

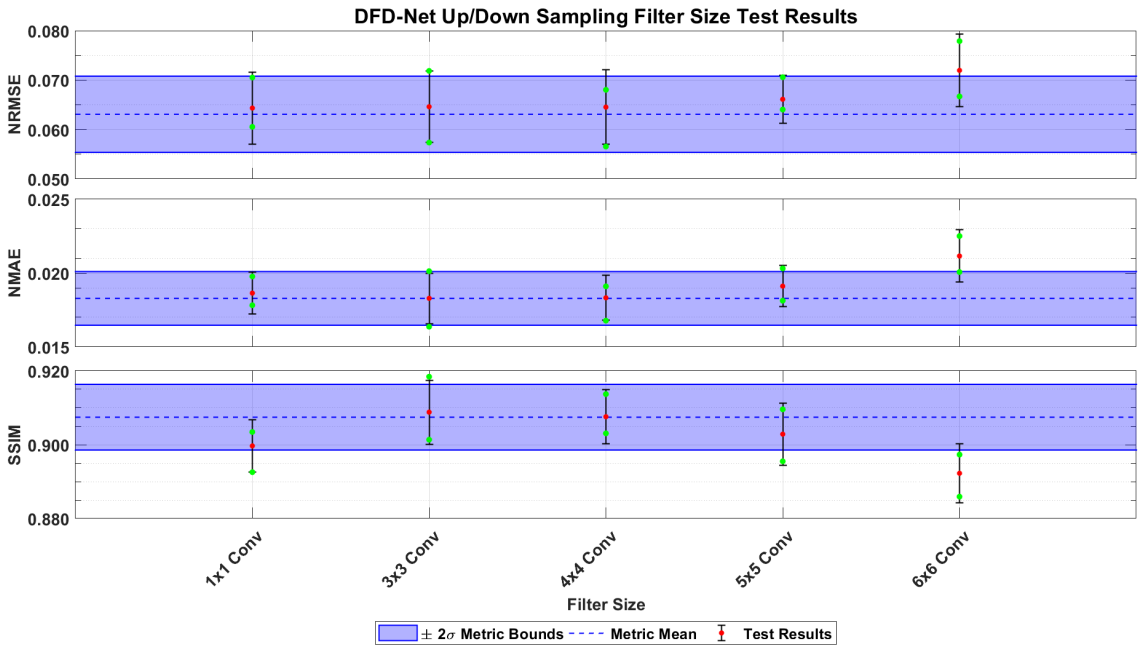


Fig. 5.23. DfD-Net Test Results with Various Convolutional Filter Up/Down Sampling Filter Sizes

5.5 Summary

In this chapter the DfD-Net deep learning architecture was discussed. The results for the synthetically blurred datasets were presented. In general the DfD-Net outperformed the graph cuts algorithm when comparing the NRMSE and NMAE metrics. However the DfD-Net did not outperform the graphs cuts algorithm when comparing the SSIM metric results.

While the process of training sample selection and the Adam optimizer introduce variability in the final results, the standard deviation for each metric is very tight. In fact the distribution for each metric were within $\pm 2\sigma$ from the mean, with the exception one test that produced results outside the 2σ bounds (most accurate of the 30 trials).

The DfD-Net single CPU run time was an average of 10.0293 seconds per image. The run time for the CPU/GPU configuration was an average of 0.3977 seconds per

image. The run time of the DfD-Net is significantly improved compared to the run time of the graph cuts algorithm, running at between 16 and 30 times faster than the graph cuts method depending on illumination and exposure level. More importantly the algorithm runtime is not dependent on exposure of illumination level.

The next chapter will discuss optimizing the performance of the DfD-Net using a variant of the Particle Swarm Optimization (PSO) algorithm and a new method of clustering filter outputs using the self-organizing map (SOM) to determine a minimum number of required filters per layer.

6. DFD-NET PARAMETER OPTIMIZATION

As deep learning systems continue to increase in size and complexity there will be a need to optimize the parameters that drive the network's performance. The goal is to determine an optimal set of parameters that produce a fully trained network whose performance is better than that of the network prior to optimization.

There is ongoing research to optimize the hyperparameters in a deep learning architecture using techniques such as Bayesian optimization [49–51]. These techniques use the results of prior hyperparameter function evaluations to generate a predictive distribution to guide the algorithm to find a set of hyperparameters that results in a minimized objective function, in this case the error of the output of a given DNN. Others are using evolutionary algorithms like Genetic Algorithms (GA) [52] to determine the optimal set of hyperparameters.

In this chapter two separate optimization methods will be discussed. The first is the use of the Particle Swarm Optimization (PSO) algorithm to improve the performance of the DfD-Net. The second method is one that determines the minimum number of convolutional filters and/or fully connected layers required by the network architecture to perform its given task. This method is intended to decrease the overall runtime of the network architecture while still maintaining or exceeding the level of performance prior to the optimization.

6.1 DfD-Net Performance Optimization Using The Particle Swarm Optimization Algorithm

PSO has begun to gain attention from works by Ye [53] and Lorenzo, et al. [54] in the application of DNN hyperparameter optimization. PSO is an iterative randomized search optimization algorithm that was originally developed by Eberhart and

Kennedy [55] and was modeled after social interactions between that of swarming insects or that of flocking birds. This research uses a variation developed by Clerc and Kennedy [56] in which a constriction factor was used to speed up the rate of convergence. The swarm consists of candidate solutions, called particles. Each particle is a vector which contains a potential solution to the problem that requires optimizing. In the case of Lorenzo, et al. their research concentrated on optimizing only the convolutional layers (size of the filters and number of filters per layer) and the maxpooling layers (size and stride) [54] in a given network architecture, while Ye expanded the particle to include the Stochastic Gradient Descent parameters (learning rate, decay and momentum) and the dropout rate used in a given network architecture [53].

In this research, the PSO algorithm is used to improve the performance of the DfD-Net, by finding a network configuration whose results exceed the baseline performance metrics for the Middlebury College [7, 8] synthetically blurred dataset presented in Chapter 5.3. Compared to previous works, the PSO particle in this research increases the number and type of hyperparameters to be optimized. The new PSO particle includes the parameters that affect the convolutional filter layers, the batch normalization layers, the activation layers and the size of the image crop used during training. The PSO algorithm used in this research is the global best algorithm also known as the G-best algorithm and is intended to find the globally optimal solution for a given objective function. One of the advantages of the PSO algorithm over other optimization algorithms is that it does not need to differentiate the objective function. This is particularly useful when the objective function to be minimized is not easily differentiable or not continuously differentiable. This makes the use of PSO to optimize the parameters within a deep learning architecture particularly appealing.

The objective function that the PSO algorithm will minimize is defined in Equation 6.1, where the "tr" subscript represents the results for a given metric for the training dataset and the "te" subscript represents the results for a given metric for the test dataset. The γ is a weighting factor that determines the importance of the training and test results, and for this research $\gamma = 0.3$.

$$f(x) = \gamma(NMAE_{tr} + NRMSE_{tr} + (1 - SSIM_{tr})) + (1 - \gamma)(NMAE_{te} + NRMSE_{te} + (1 - SSIM_{te})) \tag{6.1}$$

Figure 6.1 shows the general structure of the particle along with the quantities that will be optimized by the PSO algorithm. The parameters are: 1) the number of convolutional filters and their height and width, 2) the non-linear activation functions, 3) the use of a batch normalization layer between the convolutional layers and the activation layers and 4) the image crop size used during training. Traditionally the PSO algorithm is used to optimize real number parameters. Since, all of the parameters for the DfD-Net are either integers or not numbers at all and are also not necessarily contiguous, e.g. the convolutional filter height and width are odd, mapping functions were used to convert the resulting PSO particle into valid parameters for the DfD-Net architecture. The parameters and limits for each of the particles is outlined in Appendix C.

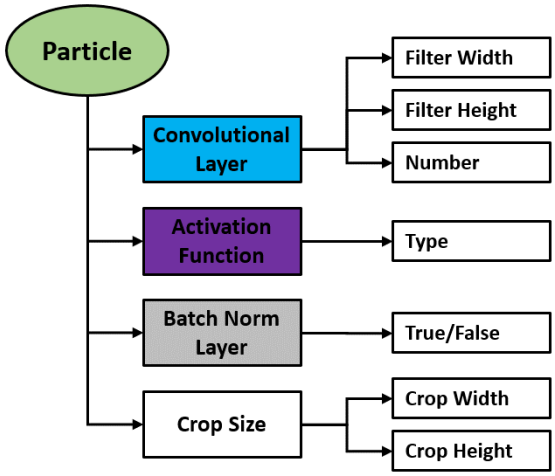


Fig. 6.1. Graphical Representation of the DfD-Net PSO Particle

The entire PSO process is outlined in Algorithm C.1. In this research the number of particles used in each iteration was set to 20, and the maximum number of iterations was set to 50. For each iteration 20 versions of the DfD-Net were generated and each

network was trained independently on the synthetically blurred dataset. At the end of each iteration the objective function for each particle/network architecture was calculated. Using the objective function results the personal best for each particle and the global best particle were determined (See Section C.1 for complete details). This was repeated until the maximum number of iterations was reached.

Figure 6.2 shows the graphical representation of the DfD-Net architecture that has been optimized by the PSO algorithm. The PSO algorithm determined that, in all but one case, the pReLU was the best activation function for the network. A sigmoid layer was selected as the first activation layer after the input. The PSO algorithm also determined that the batch normalization layer should be used in exactly the same locations as they were in the original DfD-Net architecture. And of course the largest difference between the original DfD-Net and the PSO DfD-Net are the changes to the number of filters in each of the convolutional layers. The PSO algorithm also determined that a training crop size of 32x32 pixels produced the best results.

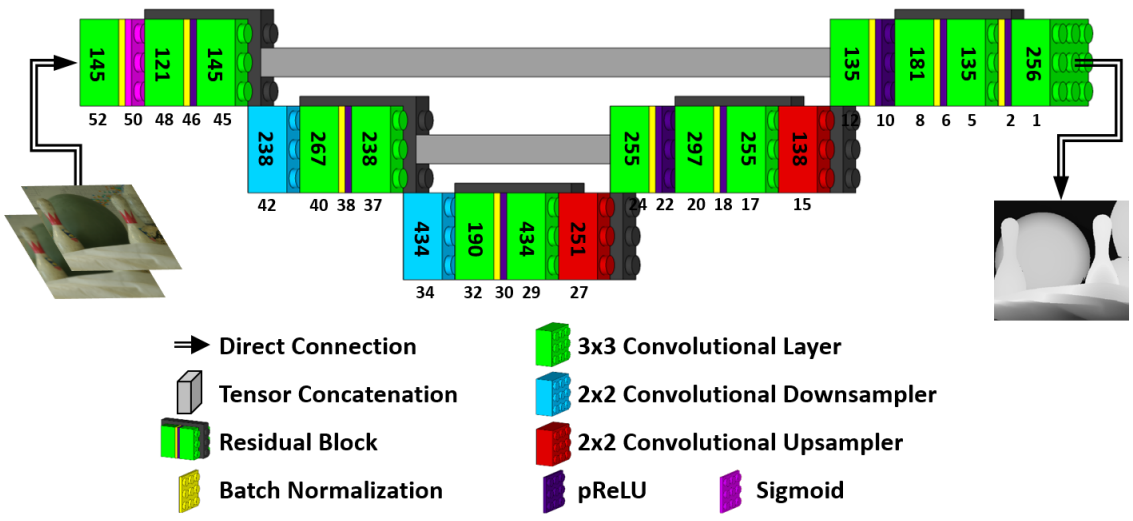


Fig. 6.2. Graphical Representation of the DfD-Net PSO Network Architecture

Figure 6.3 shows the top 5 and bottom 5 performance results for the DfD-Net. The four images on the upper left represent the highest performing results with an NRMSE, NMAE and SSIM of 0.0340, 0.0093 and 0.9477, respectively. While the four

images on the lower right indicate the worst performance results with an NRMSE, NMAE and SSIM of 0.0873, 0.0264 and 0.8457, respectively. With an average score of 0.0517, 0.0154 and 0.9156 for the NRMSE, NMAE and SSIM, respectively.

Table 6.1 shows the mean performance numbers for each of the metrics for each exposure/illumination level for the PSO DfD-Net. In addition the best results for the DfD-Net are also shown as a comparison. The PSO DfD-Net produced better results for both the NRMSE and NMAE for all of the exposure level 0 and 1 images. The PSO DfD-Net also produced an overall better SSIM result for the exposure level 0 images versus the original DfD-Net, however, the original DfD-Net produces a higher SSIM value for the exposure level 1 and 2 images.

Table 6.1.
DfD-Net & PSO DfD-Net Average Performance Comparison

Lighting		**DFD-Net**			**PSO DFD-Net**		
		NRMSE	**NMAE**	**SSIM**	**NRMSE**	**NMAE**	**SSIM**
Exp 0	Illum 1	0.0582	0.0192	**0.9095**	**0.0524**	**0.0178**	0.9092
	Illum 2	0.0522	0.0166	**0.9160**	**0.0499**	**0.0157**	0.9128
	Illum 3	0.0603	0.0205	0.9040	**0.0515**	**0.0176**	**0.9092**
	Overall	0.0569	0.0188	0.9098	**0.0512**	**0.0171**	**0.9104**
Exp 1	Illum 1	0.0529	0.0140	**0.9274**	**0.0469**	**0.0132**	0.9247
	Illum 2	0.0520	0.0136	**0.9280**	**0.0476**	**0.0126**	0.9277
	Illum 3	0.0534	0.0149	**0.9227**	**0.0482**	**0.0142**	0.9187
	Overall	0.0528	0.0142	**0.9260**	**0.0476**	**0.0133**	0.9237
Exp 2	Illum 1	0.0546	**0.0148**	**0.9204**	**0.0526**	0.0149	0.9173
	Illum 2	0.0541	0.0152	**0.9215**	**0.0529**	**0.0143**	0.9189
	Illum 3	**0.0588**	**0.0176**	**0.9128**	0.0634	0.0187	0.9022
	Overall	**0.0558**	**0.0159**	**0.9182**	0.0564	0.0159	0.9128

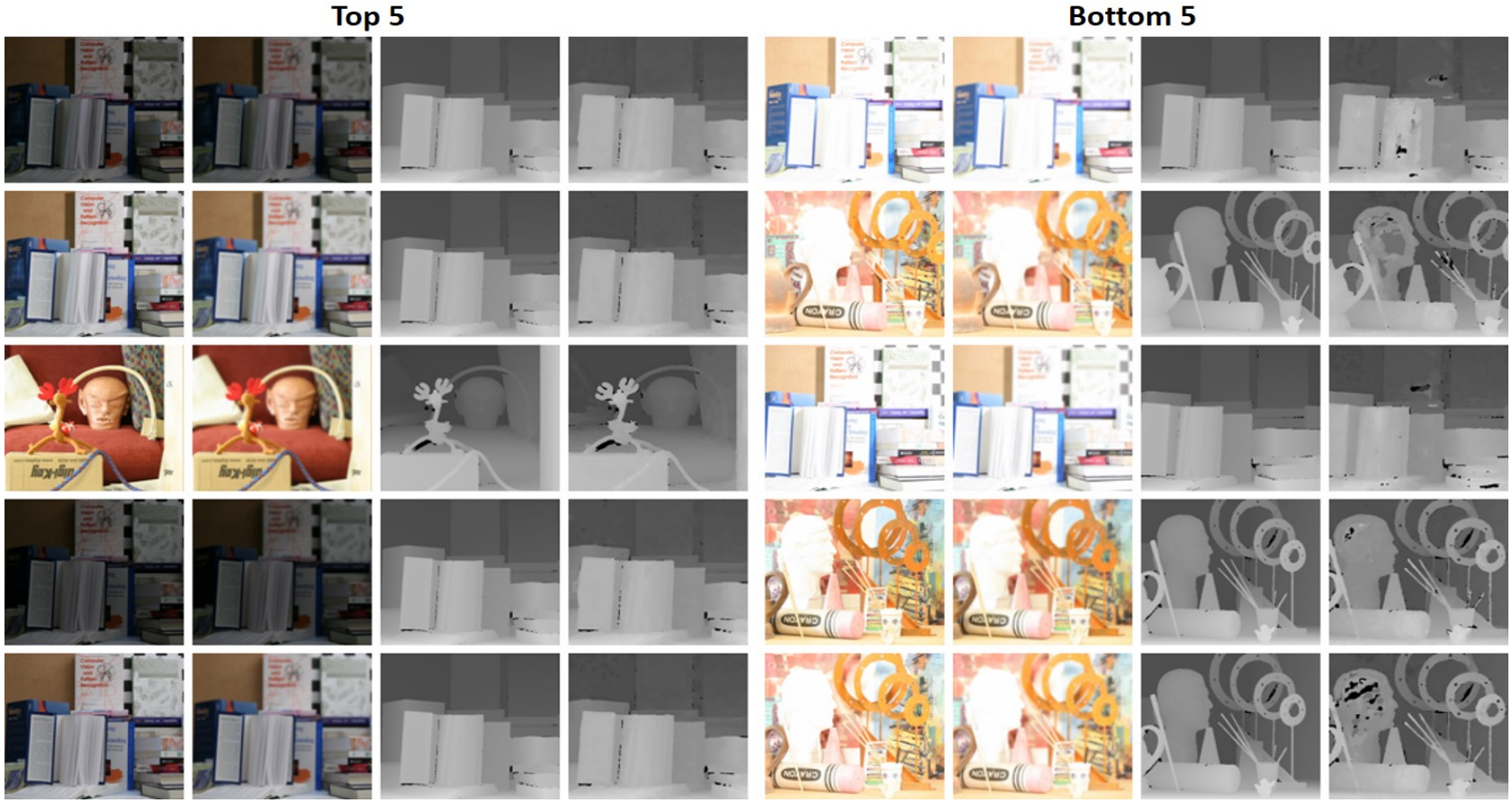


Fig. 6.3. Top 5 and Bottom 5 PSO Performance Results for the Middlebury College Stereo Vision Dataset. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Ground Truth Depth Map, (d) & (h) PSO DFD-Net Computed Depth Map.

6.2 DfD-Net Complexity Reduction Using An Unsupervised Clustering Algorithm

Edge node computing allows one to process data at the edge of a distributed network, sending only the pertinent data back (image classification, bounding box locations, etc...) to a centralized location versus sending back millions of bytes of data to have the central node perform the processing. Processing at the edge alleviates the network congestion that would occur otherwise, and allows the central node to spend less time processing the data and more time on other important tasks. However, these edge nodes have low computing capability, they generally are small single board computers. The introduction of new single board computers dedicated to deep learning architectures [57] and the embedded hardware industry pushing R&D efforts to implement DNNs on FPGAs [58] has greatly increased the potential of edge node computing. Even with these advancements, the embedded systems still have finite resources. By working to reduce the number of parameters, e.g. the number of neurons in a fully connected layer or the number of filters in a convolutional layer, the overall computational requirements are reduced and potentially increasing the processing speed of the network.

There has been some prior research dedicated to complexity reduction. Chu and Krzyżak proposed a method of filter reduction by analyzing the size of the convolutional filter [59]. Their research suggests for a single grayscale input image the upper bound on the number of non-redundant filters would be 256^{r^2} where r is the radius of the filter. This has limitations in that they assume that the difference between any grayscale value is negligible and collapse the problem down to representing the feature maps in a binary fashion which reduces the number of non-redundant filters to 2^{r^2}. Based on their proposed equation, the number of non-redundant filters begins to grow quickly if you consider color images and more than one convolutional layer.

RoyChowdhury, et al. proposed a method that analyzes the filters to determine their redundancy. They used the "cosine similarity, $\left\langle \frac{w_i}{\|w_i\|}, \frac{w_j}{\|w_j\|} \right\rangle$ with $\langle ... \rangle$ denoting

the inner product between the two filters" [60] as a measure of how similar the filters are. They would then take the similar filter means and replace all of the similar filters with a single filter. This requires that they manually rewire the network to account for the fact that there are m less filters. Their method was also not able to discern differences in their CNN test architecture which was a variant of the LeNet-5 CNN architecture [61].

Another reduction method developed by Yang, et al. [62] uses the amount of energy consumed by the architecture on a given platform as a measure of reducing the number of filters in a DNN. Their method removes filters from a trained network, much like the work done by RoyChowdhury, et al, however they add an additional fine tuning step that utilizes a closed-form least-squares method to improve the accuracy. The drawback is that this has to be done on the individual platform to realize the performance gains.

In this research, the application of an unsupervised clustering algorithm on the outputs of a DNN layer is used to determine the minimum number of convolutional filters and/or fully connected layers in a DNN architecture. This reduction in model complexity is performed while still maintaining a similar level of test accuracy that was achieved prior to the reduction.

The datasets used are the synthetically blurred Middlebury College Stereo Datasets from 2005 and 2006 [7, 8] as described in Sections 3.1. The DfD-Net described in Section 5.1 will serve as the baseline architecture for comparison of the reduction efforts. The training repeatability analysis conducted in Section 5.4.1 and shown in Figure 5.18 will serve as the baseline performance measure for each metric. These results will be used to compare against the reduction results in order to assess the performance gain or loss of the reduction candidates. Any result that exceeds the mean for each metric is considered to be a successful reduction candidate.

A representative sample from each of the test scenes was run through the network and the outputs of each layer for each scene was recorded. Figure 6.4 shows an example of the output for the first pReLU layer (layer 50) for one of the Art test

input images. The three images are examples of the output from a pReLU layer where they are nearly identical, with only small variances between the images.

(a) (b) (c)

Fig. 6.4. DfD-Net Layer 50 Filter Output: (a) Filter #070, (b) Filter #084 and (c) Filter #108

The non-linear layers were chosen as the analysis points because they have the potential to bring less similar outputs from the previous layer closer together. For example, the pReLU layer has the potential to align dissimilar inputs to produce a similar output. Referring back to the equation for a pReLU (Equation 5.7), if $\alpha = -1$ then $f(1) = f(-1) = 1$. In addition to the pReLU layers the points of tensor addition for the residual blocks were also analyzed. For the upsampling and downsampling layers there were no pReLU layers following them, therefore these layers were analyzed directly. Normally the points of tensor concatenation should also be considered as analysis points, however buffer convolutional layers (layers 24 and 12) were inserted into the design to help manage tensor imbalance on the upsampling side of the network. For this reason the tensor concatenation points were not considered.

In order to determine the minimum number of convolutional filters per layer an unsupervised clustering method called the self-organizing map (SOM) algorithm originally developed by Kohonen [63] was chosen. It has several advantages over other popular clustering algorithms. The main advantage is that the SOM uses competitive neurons that fight to match their own weights to the values of a particular input. This competition also allows for the possibility that a particular neuron may never activate for the given set of input vectors. This means that if a neuron fails to activate for any

input then another neuron will activate for multiple inputs, which results in clustering N inputs into C classes. This has the advantage over other clustering algorithms like k-means in that we don't necessarily know the number of clusters in the output of a particular layer, only the upper bounds. For each of the representative samples, the outputs were clustered using the SOM algorithm. This resulted in a different number of clusters for each layer/residual block analyzed. For this reason the cluster minimum, mean and maximum cluster values were used as the basis for the filter reduction.

The first reduction approach was an ensemble method where the number of filters for all of the layers were modified at one time. The number of filters for each layer were taken directly from the SOM algorithm results outlined in Table D.1. This resulted in nine different combinations of filter values. Each combination was trained for 15 independent training events to ascertain the test distribution for each of the three metrics.

The second method is an iterative reduction approach. Instead of reducing all of the layers simultaneously, the residual blocks were reduced one at a time. The decision of which block to begin with is now the question. To understand better we can simply look at the number of multiplies occurring within a given convolutional filter. The logical choice would be to try and reduce the largest number of filters first, however due to the DfD-Net's architecture and the means by which the inputs get downsampled and then upsampled the network is symmetric and each level, in general, has the same number of multiplications. Since there is no "one good" residual block to begin the reduction analysis, the strategy taken was to start at the input to the network and work across the same level. Once the level was completely analyzed and the number of filters was determined, the network was retrained from scratch using the reduced filter numbers. After training was completed the second level was analyzed to determine the number of filters for each layer. This process was repeated for each level until the last level was reached. At each level the filter combinations that resulted in the best performance metrics were selected. Once at the final level, the

minimum, mean and maximum number of filters were determined. Each combination was trained for 15 independent training events to ascertain the test distribution for each of the three metrics.

Table D.2 shows the final combinations of convolutional filters for each layer. The layer numbers are color coded to match the layer type as depicted in Figure 5.5. The numbers in each column represent the number of convolutional filters for a given layer as determined by the SOM algorithm and selected permutations of the minimum, mean and maximum values from Table D.1. The network configurations designated with 'B' are the configurations that were determined by performing the ensemble reduction analysis and the network configurations designated with 'F' are the results of the iterative reduction analysis. This is not an exhaustive search of the space. For this particular architecture the upper bounds on the number of possible combinations is defined in Equation 6.2, where n is a function of the number of residual blocks and independent convolutional filters. Based on the DfD-Net architecture there are $4^{13} = 67,108,864$ possible filter reduction combinations.

$$Reduction\,Combinations = 4^n \tag{6.2}$$

Figure 6.5 shows the results of each of the network configurations listed in Table D.2. The red dots indicate the mean test accuracy and the bars represent $\pm 2\sigma$ from the mean. The green dots represent the minimum and maximum test accuracy for the 15 training vignettes. The labels on the x-axis represent the network configuration. The F-02-02 network produced the best results with a minimum NRMSE and NMAE of 0.06095 and 0.0184 respectively and a maximum SSIM value of 0.9062. The F-02-02H and F-02-02L network configurations are specialized versions of the F-02-02 network configuration in which the number of filters was rounded up to a multiple of eight (F-02-02H) or rounded down to a multiple of eight (F-02-02L). This was done to see if the GPU architectures gave a performance advantage to filters that were a multiple of eight.

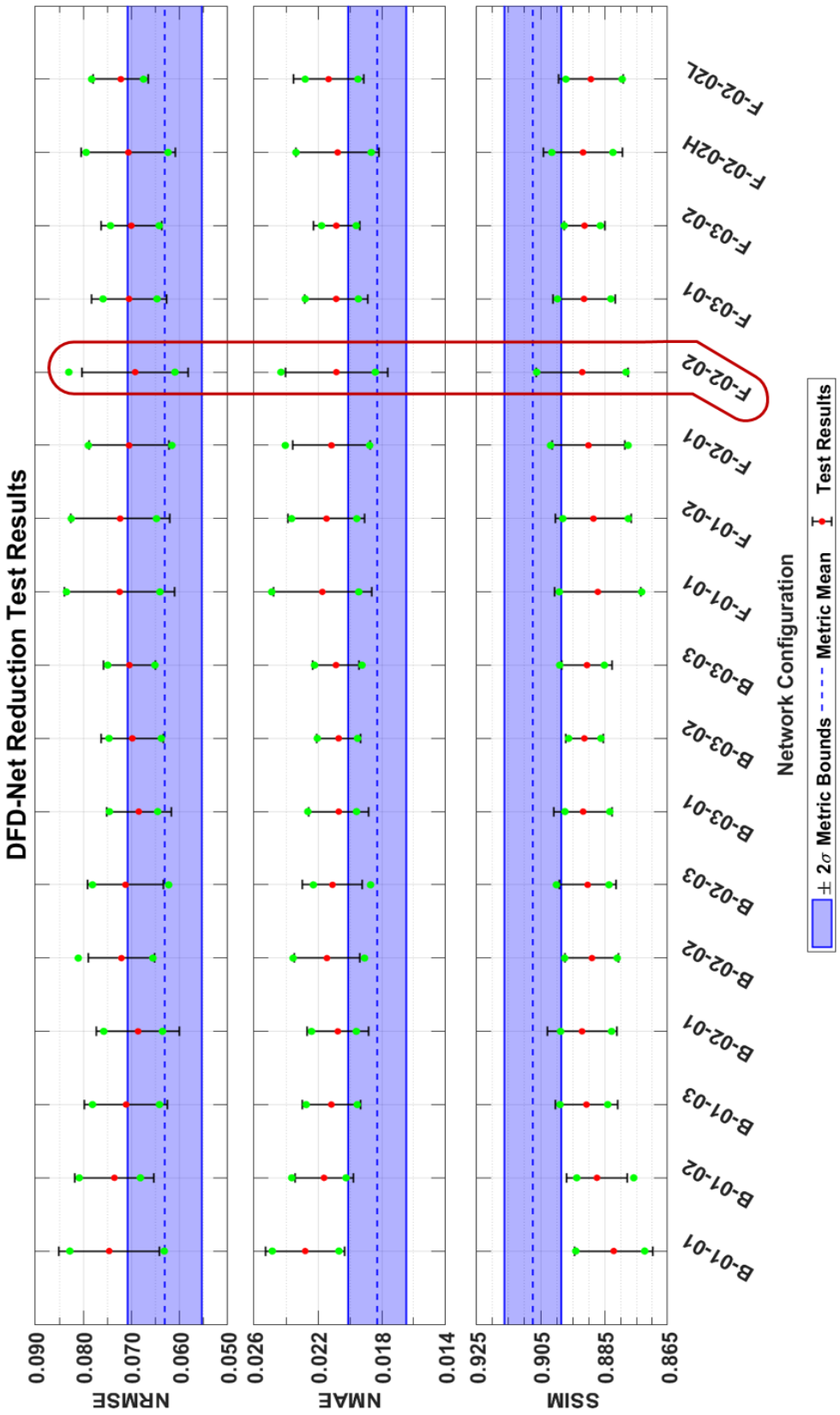


Fig. 6.5. DfD-Net Reduction Test Results

A timing analysis was performed on each of the network configurations outlined in Table D.2. Table D.3 shows the results of this analysis for each of the four hardware platforms listed in Table B.1 in Appendix B. The B-01-01 network configuration had the fastest performance for the Laptop (57.5% speed increase), Jetson TX2 (52.44% speed increase) and Desktop 2 (34.26% speed increase) while the F-01-02 network configuration has the fastest performance for the Desktop 2 hardware (23.62% speed increase). From The results shown in Figure 6.5 network configuration F-02-02 produced the best performance results for each of the three metrics. This network configuration produced an average speed increase of 44.85%, 40.04%, 14.91% and 26.70% for the Laptop, Jetson TX2, Desktop 1 and Desktop 2 respectively. As a side note the F-02-02L and F-02-02H network configurations did not provide any significant speed increases above the root F-02-02 network configuration.

6.3 Summary

6.3.1 PSO Summary

It was expected that the variance observed in Section 5.4.1 and Figure 5.18 between identical network configurations would make it difficult for the PSO algorithm to converge to a solution. However, the PSO algorithm did determine an improved solution that produced an average NRMSE that was approximately 6.25% below the DfD-Net average NRMSE. Similarly, the PSO DfD-Net produced an average NMAE that was 5.25% below the DfD-Net average NMAE. However the SSIM value for the PSO DfD-Net was 0.26% lower, but substantially similar to the baseline DfD-Net average SSIM value.

6.3.2 Complexity Reduction Summary

The method of clustering convolutional filters to reduce the number of required filters produces network architectures that run faster than the baseline network ar-

chitecture. This method also demonstrates that the network performance does not have to suffer as a result of the reduction process. With low complexity a combination of convolutional filters that produced performance results that exceeded the baseline mean by 3.4% for the NRMSE was determined. The NMAE and the SSIM were only slightly worse than the baseline means with a decrease of only 0.65% and 0.13% respectively. In addition to keeping within the bounds of the metrics for the original network configuration, the new configuration also significantly reduced the average runtime per image for each of the four test platforms.

While the reduction strategy and/or the analysis points will vary depending on the particular network architecture, the iterative reduction approach yielded better performance results when compared to the more aggressive ensemble approach. However, the iterative approach will require at most 4^n reduction iterations to reach the final network configuration.

The next chapter will discuss the application of the DfD-Net architecture to the real word dataset.

7. DEPTH FROM DEFOCUS WITH A MICROFLUIDIC LENS

This chapter will discuss the applications of deep learning and the DfD-Net to the real world dataset introduced in Section 3.2. This research expands upon the research that was previous conducted by Liu, et al. [21,64] in which the graph cuts method was used on images captured with a microfluidic lens and Pasinetti, et al. [22] in which the ICM and DCM methods were used to infer depth from a microfluidic lens using the image contrast. The research shows the potential of using a microfluidic lens as a means to capture in-focus and out-of-focus images.

The first section introduces the application of the synthetic blurring process to the real world dataset and assessing the performance of the DfD-Net trained on the Middlebury College dataset. In the second section, an analysis of the data collection hardware was performed and some issues are discussed that were discovered during this research. Additionally, this chapter will also discuss the training of the DfD-Net entirely from scratch on the real world dataset and reporting those results.

7.1 Real World Dataset Synthetic Blur Results

An experiment was created to test the performance of the real world dataset using synthetically blurred camera data. To compare the camera and LIDAR data the in-focus image for each scene and exposure time was selected, then synthetically blurred. However, in order to use the LIDAR data, it is inverted compared to the Middlebury College ground truth data, where larger depth map values in the Middlebury College dataset represent surfaces that are closer to the camera, and the larger LIDAR values represent surfaces that are farther from the camera. Since the LIDAR data was scaled to a range between 0 and 255 the LIDAR data can be inverted using Equation 7.1.

$$DepthMap_{inv} = 255 - DepthMap \tag{7.1}$$

Once the inverted depth maps were generated for each scene the in-focus image was blurred using the same process (and the same σ values) outlined in Section 3.1 using Equations 3.1 through 3.4. The synthetically blurred real world dataset was then run through the original (baseline) DfD-Net trained on the Middlebury College dataset described in Section 5.1. The top 5 and bottom 5 performers were determined based on the NRMSE score with the NMAE used in the event of a tie. Table 7.1 shows the top 5 and bottom 5 performance results for the real world synthetically blurred dataset. Figure 7.1 shows the comparison between the inverted ground truth LIDAR data and the depth map inferred by the DfD-Net for the top 5 and bottom 5 results.

For the top 5 performers the results are exceedingly good, surpassing the top 5 performance numbers of the Middlebury test dataset (Table 5.2). However, the results for the bottom 5 are far worse than those of the DfD-Net bottom 5 results (Table 5.2) or the graph cuts bottom 5 results (Table 4.2) when comparing the NRMSE numbers.

Figure 7.2 shows the average results for each scene for each of the three metrics (NRMSE, NMAE and SSIM). What immediately stands out is where the largest errors occur for each of the three metrics. The errors are occurring every fourth scene starting with the first scene, k00. From Section 3.2.4 the 4-tuple scene configuration starts with a surface that is 2.5m from the camera and is repeated every fourth scene. It is this scene configuration that is failing to produce good results.

To better understand why these particular scenes did not produce results that were on par with the other scenes, an analysis of the training and test depth maps values was performed. Figure 7.3 shows the distribution of the Middlebury College training data (blue) based on the enhanced cropping method, detailed in Section 5.2.1, used in the training of the DfD-Net, and the real world test data (red). The depth map values of 4 through 8 are not in the training dataset, and the values of 9 through

Fig. 7.1. Top 5 and Bottom 5 Performance Results for the Synthetically Blurred Real World Dataset. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) LIDAR Ground Truth Depth Map, (d) & (h) DfD-Net Computed Depth Map.

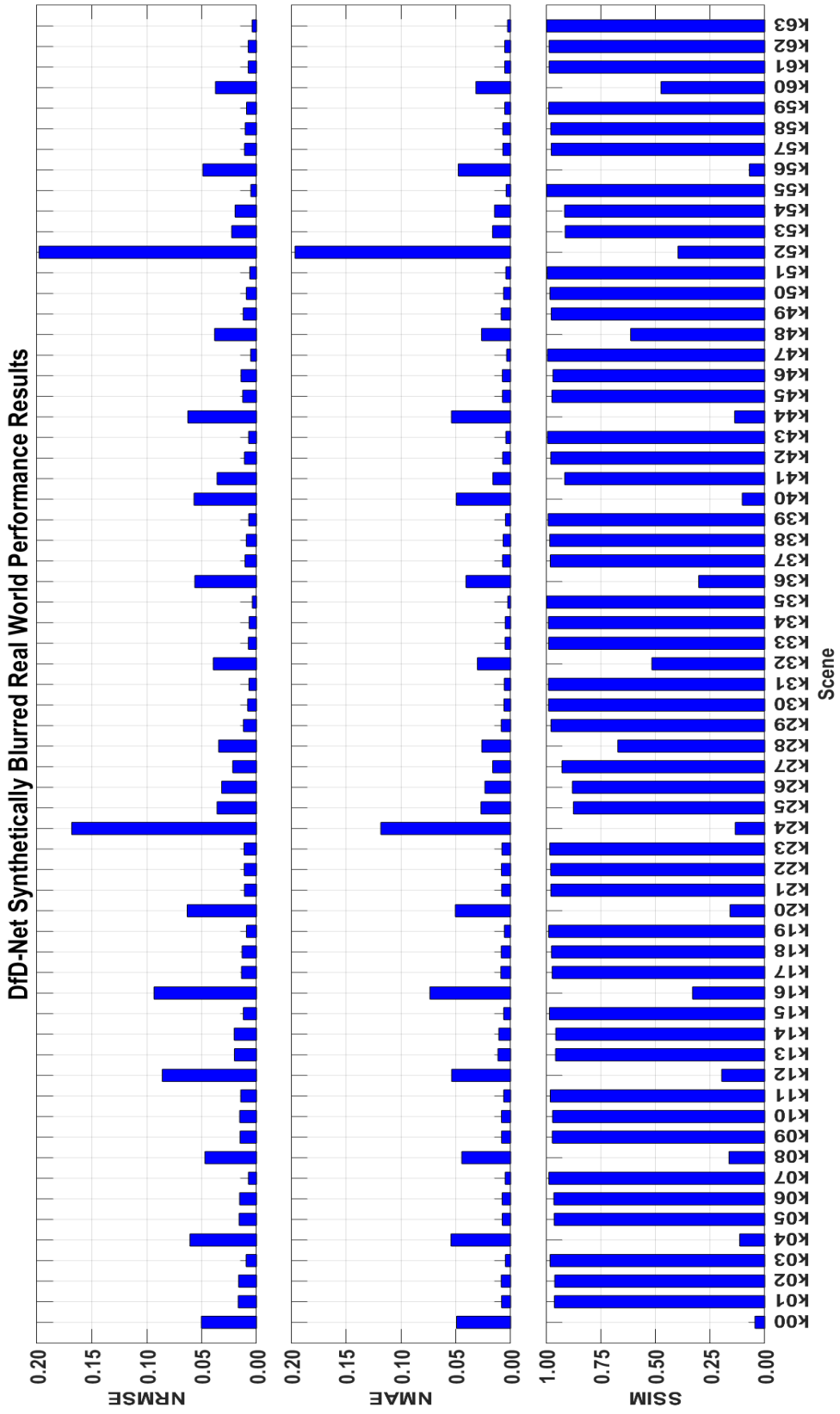


Fig. 7.2. DfD-Net Synthetically Blurred Real World Performance Results

Table 7.1.
Top 5 and Bottom 5 DfD-Net Performance Results for the Synthetically Blurred Real World Dataset

Name	**Exposure Time (ms)**	**NRMSE**	**NMAE**	**SSIM**
Top 5				
k35	30	0.00358	0.00244	0.99717
k35	40	0.00362	0.00238	0.99692
k35	20	0.00365	0.00257	0.99748
k63	50	0.00377	0.00275	0.99639
k63	40	0.00377	0.00271	0.99611
Bottom 5				
k52	50	0.19443	0.19307	0.41257
k52	60	0.19558	0.19393	0.40980
k52	70	0.19583	0.19411	0.40810
k52	20	0.19858	0.19716	0.39257
k52	10	0.21352	0.20993	0.32512

11 for the real world dataset are under represented in the training dataset. The two scenes that produced the highest average NRMSE value, (the worst performers), are the k52 and k24 scenes.

Figure 7.4a shows the comparison between the training depth map distribution and the distribution of the k52 scene, which produced the worst NRMSE score. Similarly, Figure 7.4b shows the distribution for the k24 scene that produced the second worst NRMSE score. The k52 scene has its depth map value completely covered by the training set and the k24 scene only has the depth map values of 9 through 11 under represented.

Taking the ground truth depth map, input image pair and the resulting inferred depth map from the k24 scene at an exposure time of 50ms, (highest NRMSE of all

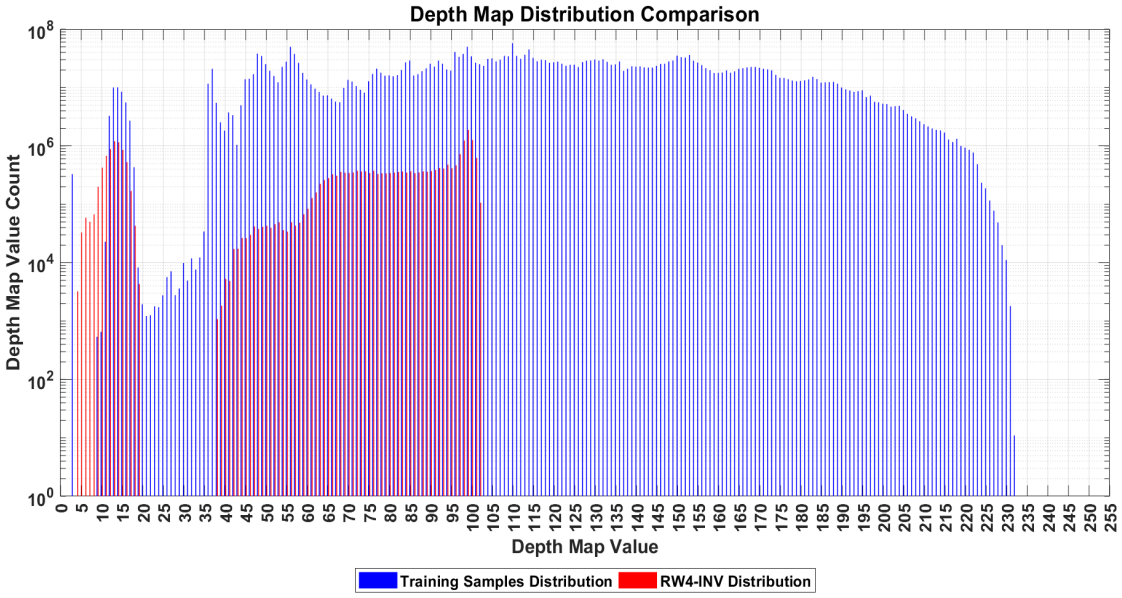
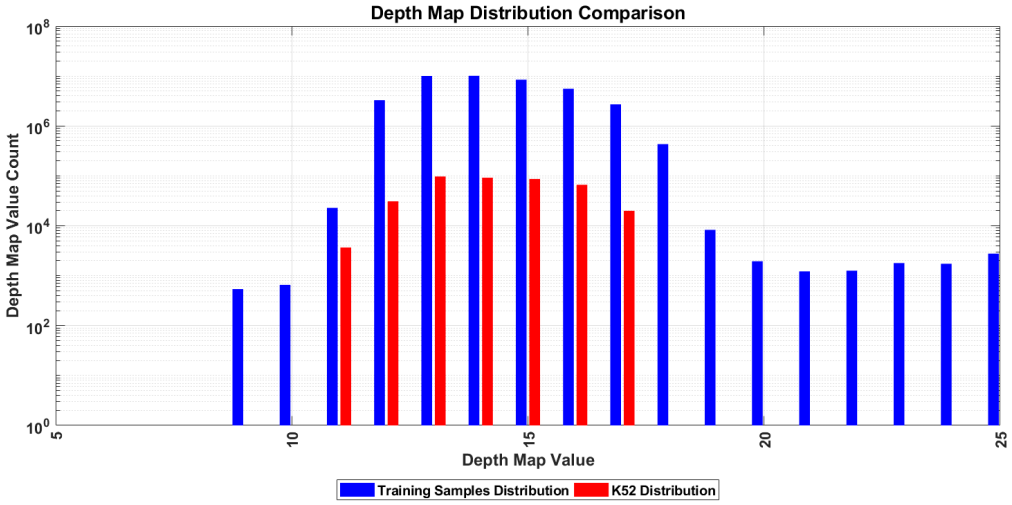


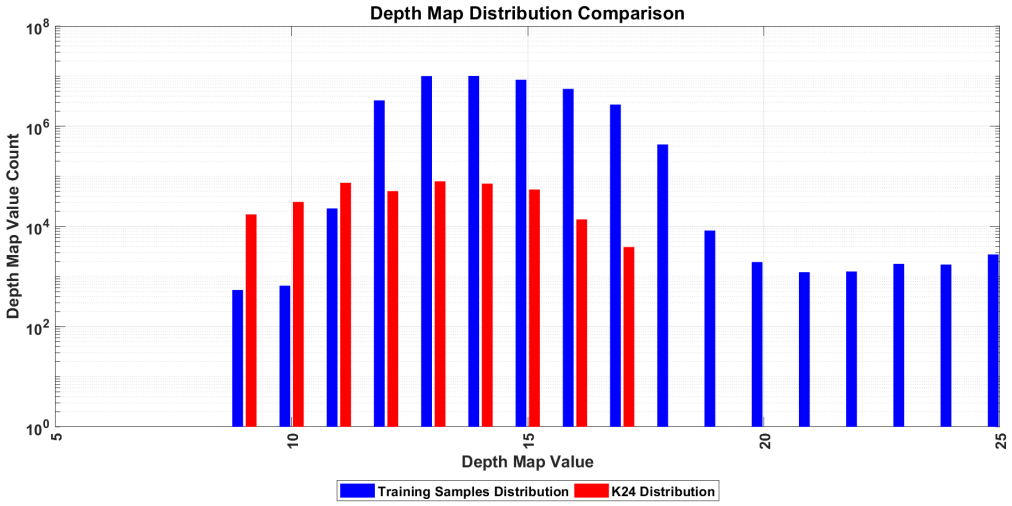
Fig. 7.3. Depth Map Value Distribution for the Middlebury College Training Dataset and the Real World Test Datasets

exposure times) a comparison is made to show that while some of the depth map values may have been under represented in the training set, the error in the depth map inference is not due to this fact. Figures 7.5a and 7.5b depict the in-focus and out-of-focus image pair. Figures 7.5c and 7.5d show the ground truth, and the depth map determined by the DfD-net, respectively. The depth maps have been colorized in order to accentuate small differences in depth values that would otherwise be difficult to visually detect in a grayscale image. Figure 7.5e shows the absolute error between the two depth maps (DM) as determined by Equation 7.2. Figure 7.5f shows the location of the depth map values of 9 through 11 map, in white, while Figure 7.5g overlays the location of these values over top of the error map. Figures 7.6a - 7.6g show the same analysis for the k52 scene at an exposure time of 10ms. Clearly it can be seen that the errors are not localized to the under represented values, but are more systemic. For the k24 scene the larger errors are occurring at the transitions between strips in the darker blue areas.

$$\epsilon = |DM_{gt} - DM_{DfD-Net}| \tag{7.2}$$



(a) Scene k52, Exposure: 10ms Distribution



(b) Scene k24, Exposure: 50ms Distribution

Fig. 7.4. K52 & K24 Depth Map Value Distribution

For the k52 scene the source of the error becomes a little more clear. Figure 7.7a shows a small, unmagnified portion of the in-focus image from the dataset and Figure 7.7b shows the synthetically blurred version of the same image. To contrast, Figure 7.7c shows the same pattern taken at the same distance with a higher resolution Samsung Galaxy S8 cellphone camera with the parameters outlined in Table

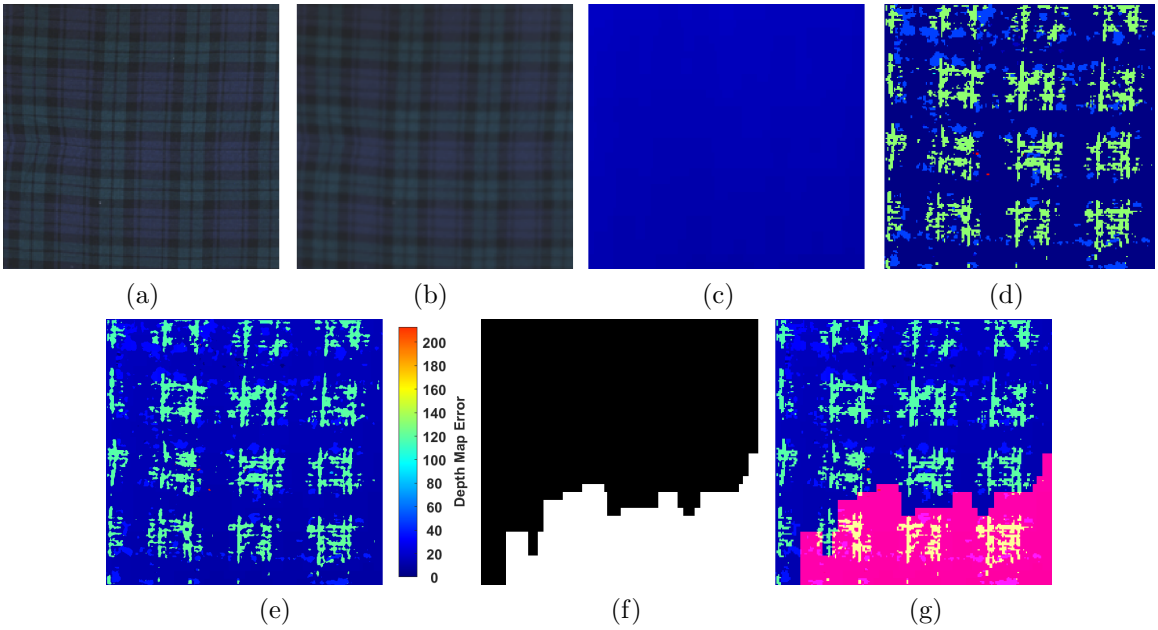


Fig. 7.5. Depth Map Error Comparison for the k24/50ms Exposure scene. (a) In-Focus Image (b) Out-of-Focus Image (c) Ground Truth, (d) DfD-Net Depth Map, (e) Error Map, (f) Pixel Mask and (g) Under Represented Pixel Mask an Error Map Overlay

7.2. It should be noted that the pixel size of the cellphone camera is approximately 3.4 times smaller than the Chameleon3 camera used to collect the real world dataset. The camera resolution is too coarse to resolve the finer details of some of the patterns/textures within the dataset, especially at a distance of 2.5m from the camera. The distinguishing features and edges are lost in the in-focus image and the problem only compounds when the synthetic blur is applied. Even some of the color is lost, because from Figure 7.7c it can be seen that the surface being imaged is a green and white hounds tooth pattern and not a dark gray pattern.

Because the camera and lens combination are not capable of resolving details to the level required for the scenes located at 2.5m the synthetically blurred results were only analyzed for the scenes that ranged between 1.5 and 2.2 meters from the camera. Figure 7.8 shows the performance results for each scene and each exposure time. There are three prevalent trends in the results. The first is that for a particular scene the

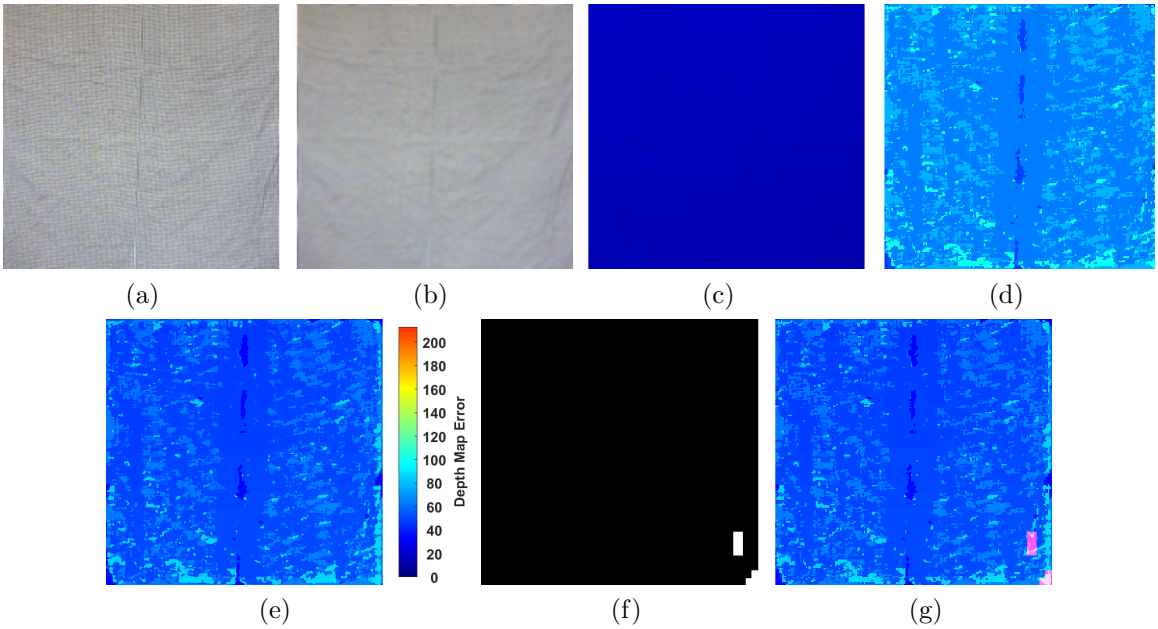


Fig. 7.6. Depth Map Error Comparison for the k52/10ms Exposure scene. (a) In-Focus Image (b) Out-of-Focus Image (c) Ground Truth, (d) DfD-Net Depth Map, (e) Error Map, (f) Under Represented Pixel Mask and (g) Pixel Mask and Error Map Overlay

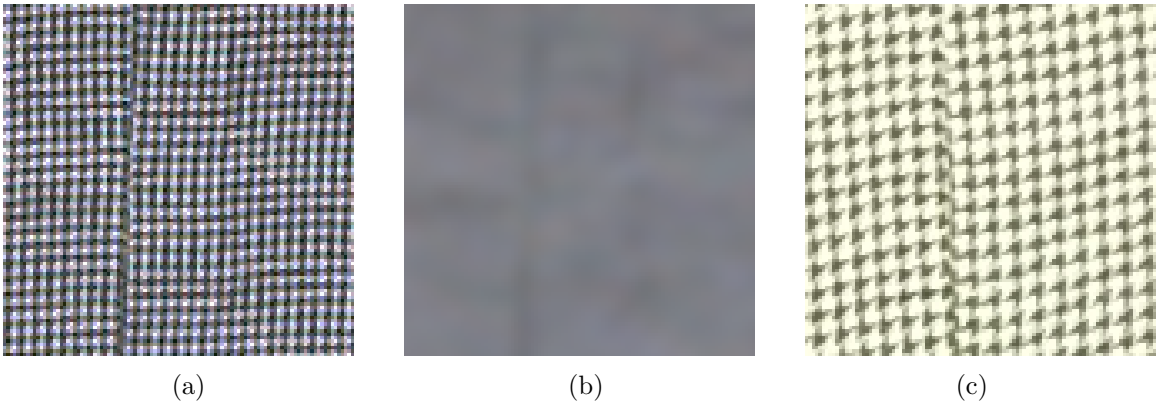


Fig. 7.7. Scene k52 In-Focus and Out-of-Focus Image Comparison. (a) In-Focus Image, (b) Synthetically Blurred Image and (c) High Resolution In-Focus Image

results are consistent across all of the exposure times. The second trend is that the shortest exposure times result in performance numbers that are far worse than the

Table 7.2.
Samsung Galaxy S8 Rear Camera Specifications

Parameter	**Value**
F-Number	1.7
Resolution (h x w)	3024 x 4032, 12 MP
Frame Rate	30+ FPS
Sensor	Sony IMX333, CMOS 1/2.55”
Pixel Size	1.4 x 1.4 μm

longer exposure times. This is most obvious in scenes k25, k26, k27 and k41. This is because the overall texture that was used was very dark and the shorter exposure times resulted in images that lost a lot of the texture information as compared to the longer exposure times for the same scene. The last trend is one where the performance drops as the exposure times increase. The k05, k06, k55 and k56 scenes exhibit this behavior. Here the particular textures are very bright and as the exposure time increased the image brightness increased to the point where the images were slightly over saturated. Table 7.3 presents the overall performance for each of the three metrics.

Table 7.3.
DfD-Net Real World Synthetically Blurred Overall Test Results

	NRMSE	**NMAE**	**SSIM**
Minimum	0.0036	0.0024	0.6475
Mean	0.0129	0.0082	0.9702
Maximum	0.0835	0.0703	0.9984
Std Deviation	0.0105	0.0069	0.0418

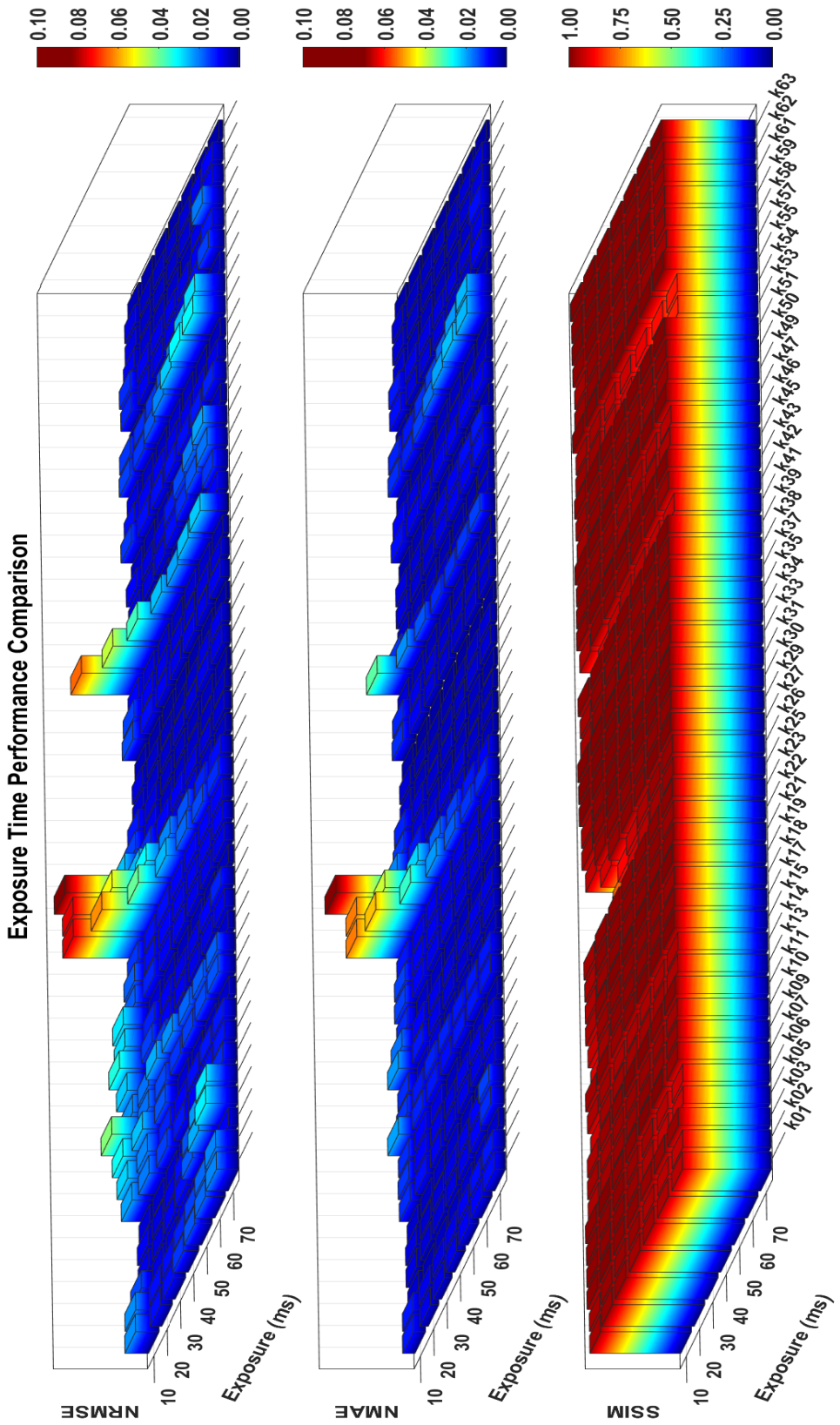


Fig. 7.8. DfD-Net Synthetically Blurred Real World Exposure Time Performance Comparison

7.2 Real World Dataset

7.2.1 Microfluidic Lens Issues

In Chapter 2 the derivation of the geometric optics equations was introduced. These equations described the behavior of a thin lens systems for a given f-number and focal length, and provided a theoretical model for predicting the blur radius and quantized blur radius based on the physical size of a pixel in an imaging sensor (refer to Figure 2.5 for an example based on the supplied specifications of the lens and camera used in the real world data collect). While the manufacturer of the lens provided a single set of specifications for the microfluidic lens, it was discovered that the lens changes optical characteristics based on the voltage step applied.

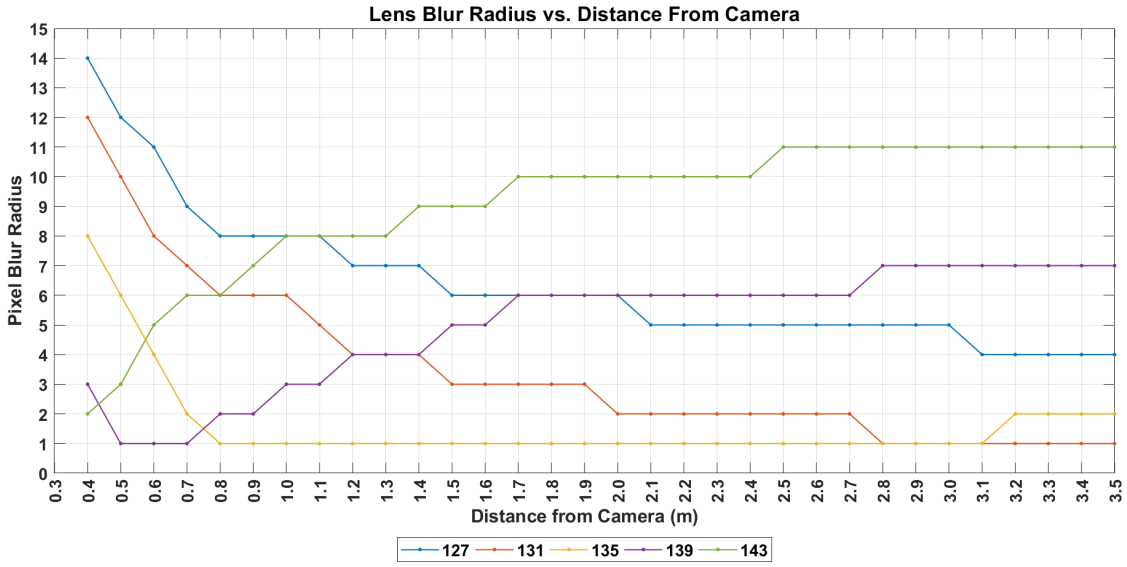


Fig. 7.9. Quantized Pixel Blur Radius for the Chameleon 3 Camera and Microfluidic Lens

Figure 7.9 shows representative samples of the quantized blur radius of the microfluidic lens based on the voltage step and distance from the camera. These values were measured using a black and white target similar to the one shown in Figure 7.10a with the camera centered at the intersection of the two black rectangles. Measurements for each voltage step were taken at 0.1m intervals between 0.3m and 4.0m.

As the voltage step increased from 127 to 143 the focus distance (d_o) moved closer to the camera. What is also important to note is that one of the lens parameters, the f-number (n), that governs the shape of the near and far focus curves changes with the voltage step as well.

(a) Blur Radius Target (b) Point Source Target

Fig. 7.10. Lens Test Targets

After getting familiar with the camera and microfluidic lens, several undesirable properties of the combined system were identified. The first was that the temperature of the lens can affect the sharpness of the image taken. To compound the problem the camera's operating temperature changes over time, this temperature change is imparted to the fluid in the lens. In order to assess the effect of the camera's temperature on the lens and the dataset a test was conducted to capture and understand the affects. The data was collected using a test image similar to the one shown in Figure 7.10b, with the camera positioned 1m from the target image. The voltage step was set to 135 and the exposure time was set to 40 ms. Images were captured as the camera was started up at an initial operating temperature of 27.25 °C and were continued to be captured as the camera warmed up to a temperature of approximately 54.75 °C. When the camera is completely warmed up it operates at an average temperature of approximately 54 °C. Using the Discrete Cosine Sharpness Measure (DCSM) developed by De and Masilamani [65] the sharpness of each image was calculated. The sharpness of the noise was also calculated by removing the white dot from each image and replacing it with the average background pixel value. The results of the DCSM

for each image are shown in Figure 7.11 which indicates that the image sharpness increases with temperature. Figure 7.11 also shows that the noise of the background also increases with temperature. However, the increase in noise does not completely account for the increase in sharpness of the image.

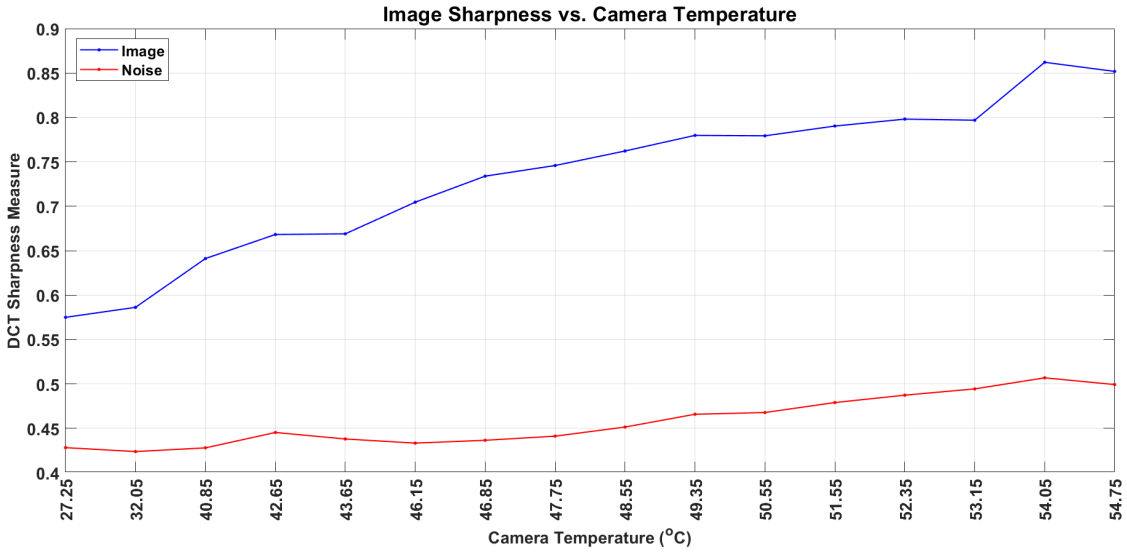


Fig. 7.11. Image Sharpness vs. Camera Temperature

The second undesirable property is a hysteresis that occurs when transitioning from one voltage step to another. It was discovered that the lens focal point didn't always return to the same point as set by the voltage step setting. To capture this behavior an experiment was conducted with the camera configured to the same settings outlined in Table 3.3, with the exposure time set to a single value of 30 ms and the voltage steps were varied across the range used in the data collection. The image used in this test is a single white dot 5mm in diameter on a field of black, similar to the image shown in Figure 7.10b, with the camera positioned 1m from the target image. At each voltage step an image of the target was captured and the DCSM was calculated [65]. Figure 7.12 shows the behavior of the lens as the voltage step was changed. The blue line represents starting at a voltage step of 126 and then capturing an image at each voltage step from 127 to 143. The red line represents setting the

voltage step to 144 and then capturing the same image moving from voltage step 143 to 127. The images were all taken at a temperature of approximately 51 °C to ensure that the camera temperature did not affect the results. The chart shows that the sharpness of the image at a particular voltage step will vary depending on the voltage step that the lens was previously configured.

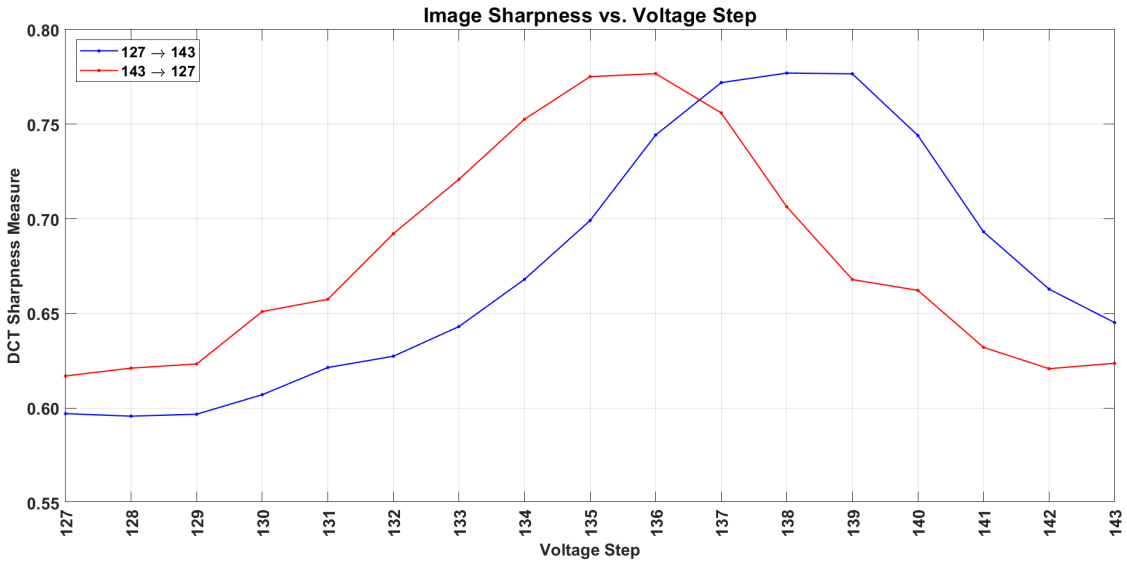


Fig. 7.12. Image Sharpness vs. Voltage Step

7.2.2 Architecture Overview and Training

Several variants of the original DfD-Net presented in Section 5.1 and shown in Figure 5.5 were explored. These include adding additional convolutional filters to the architecture and increasing the number of levels in the architecture. However, these modifications did not improve the overall performance of the network, and in some cases the architectures began to overfit to the training data. The training method employed was the same method as described in Section 5.2. The only change in the training was an increase in the number of training iterations without progress from 2500 to 3000.

Table 7.4.
Real World Training and Testing Dataset Scenes

Training Dataset Scenes							
• k00	• k01	• k02	• k03	• k04	• k05	• k06	• k07
• k08	• k09	• k10	• k11	• k12	• k13	• k14	• k15
• k16	• k17	• k18	• k19	• k20	• k21	• k22	• k23
• k24	• k25	• k26	• k27	• k28	• k29	• k30	• k31
• k36	• k37	• k38	• k39	• k40	• k41	• k42	• k43
• k48	• k49	• k50	• k51	• k52	• k53	• k54	• k55
• k56	• k57	• k58	• k59	• k60	• k61	• k62	• k63
Testing Dataset Scenes							
• k32	• k33	• k34	• k35	• k44	• k45	• k46	• k47

Table 7.4 lists the real world dataset scenes that are used as the training sets for the DfD-Net and the testing sets. In total there were 392 image pairs in the training set and 56 image pairs in the test set, which represents an 87.5%/ 12.5% split for the training and testing data. The dataset split between training and testing for the real world dataset was determined in much the same way as the training and testing sets were chosen for the Middlebury College synthetically blurred dataset. The real world dataset training and test set selection was based on the distribution of the depth maps. In addition, the test image scenes were selected based on the size and pattern of the texture used to create the scene. The k32 - k35 test scenes had a very small repeating pattern with very little area where there was no texture and the k44 - k47 test scenes had a pattern where there were areas in which there was no texture change, i.e. a solid background. Figure 7.13 shows the depth map value distribution for the training set (blue) and the test set (red).

Based on the results of the synthetically blurred dataset and the realization of the fact that the camera could not capture fine enough details to discriminate the

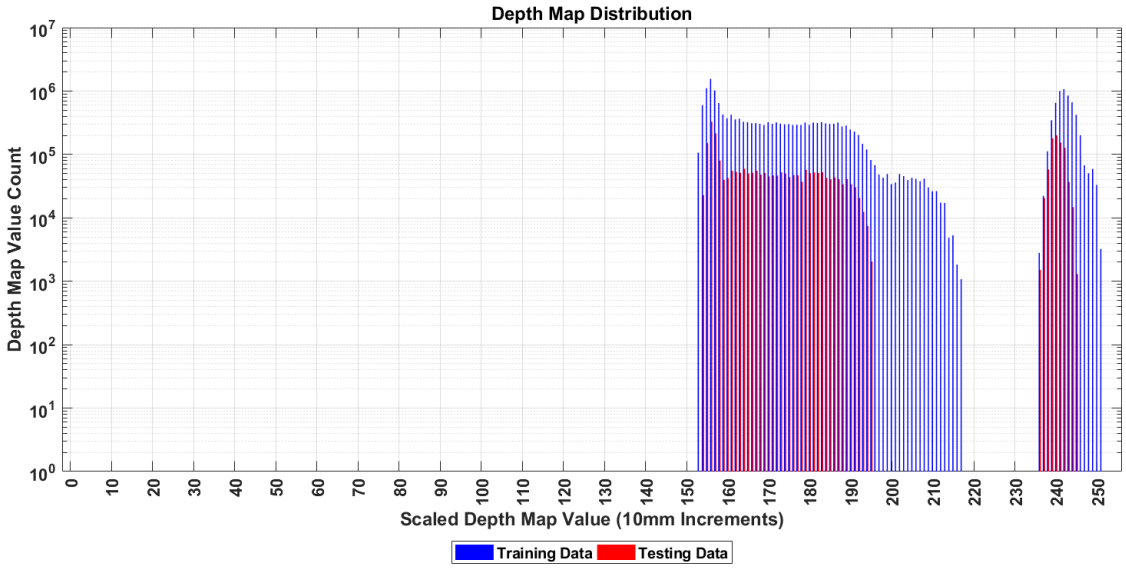


Fig. 7.13. Depth Map Value Distribution for the Real World Training and Testing Datasets

blur differences for the scenes located at 2.5m from the camera, the training and testing datasets located in column 1 and column 5 of Table 7.4 were excluded from the training and testing of the DfD-Net architecture on the real world dataset. Figure 7.14 illustrates the revised depth map distribution. The scenes within the real world dataset used from training and testing are only located between 1.5 and 2.2 meters.

7.2.3 DfD-Net Real World Results

In Chapter 5 the DfD-Net was trained on the synthetically blurred dataset. This dataset was generated using an entirely in-focus image and then blurring that image to create the out-of-focus image. Following the same logic the image recorded at voltage step 135, which was determined to be the sharpest image, was used as the in-focus image and the images recorded at the other voltage steps were used as the out-of-focus images. This led to 16 possible combinations of voltage steps to test. However, for the images recorded across the range of voltage steps, there was no clear

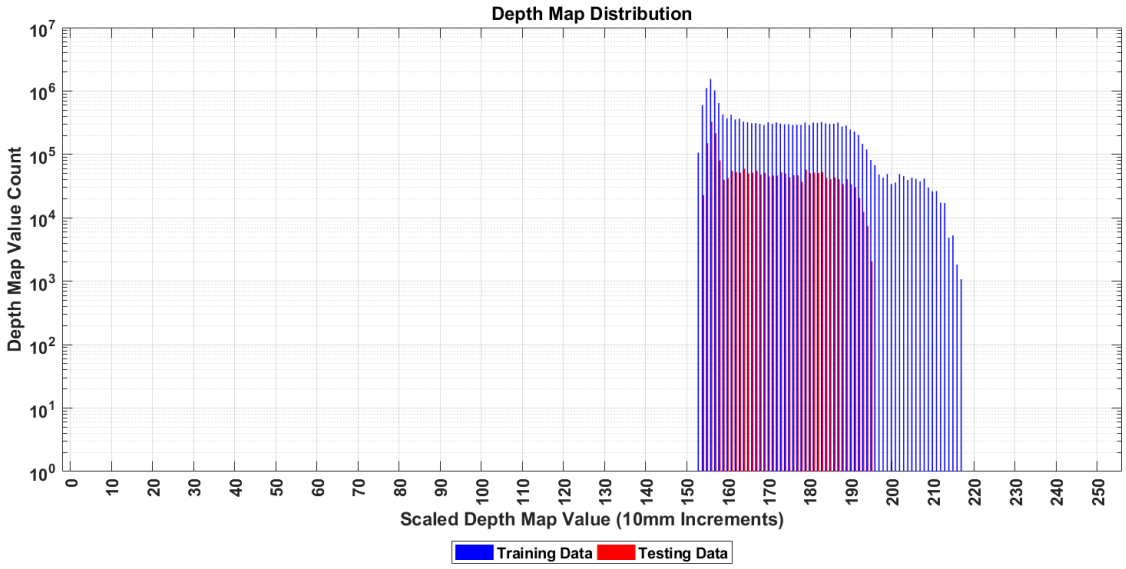


Fig. 7.14. Depth Map Value Distribution for the Pruned Real World Training and Testing Datasets

and decisive out-of-focus lens voltage step that produced significantly better results than any other voltage step, and overall the results were not very good.

After collecting and analyzing the quantized blur radius for each voltage step it was determined that the absolute differences between the voltage steps is just as important as the actual voltage steps themselves. Figure 7.15 shows an example comparison of the quantized blur radii between two voltage steps, in this case voltage step 141 and voltage step 129. The figure also shows the absolute difference between the two quantized blur radii.

By analyzing the absolute difference between the quantized blur radii produced by two different voltage steps at varying distances from the camera there are several potential candidates that should produce better results as compared to the traditional method of using an entirely in-focus image and selecting another voltage step as the out-of-focus image. There are two properties that the differences in blur radius should have, the first is required and the second is desired, but may not always be achievable. The first property is that the absolute difference between the quantized blur radii should be either monotonically increasing or monotonically decreasing in the region

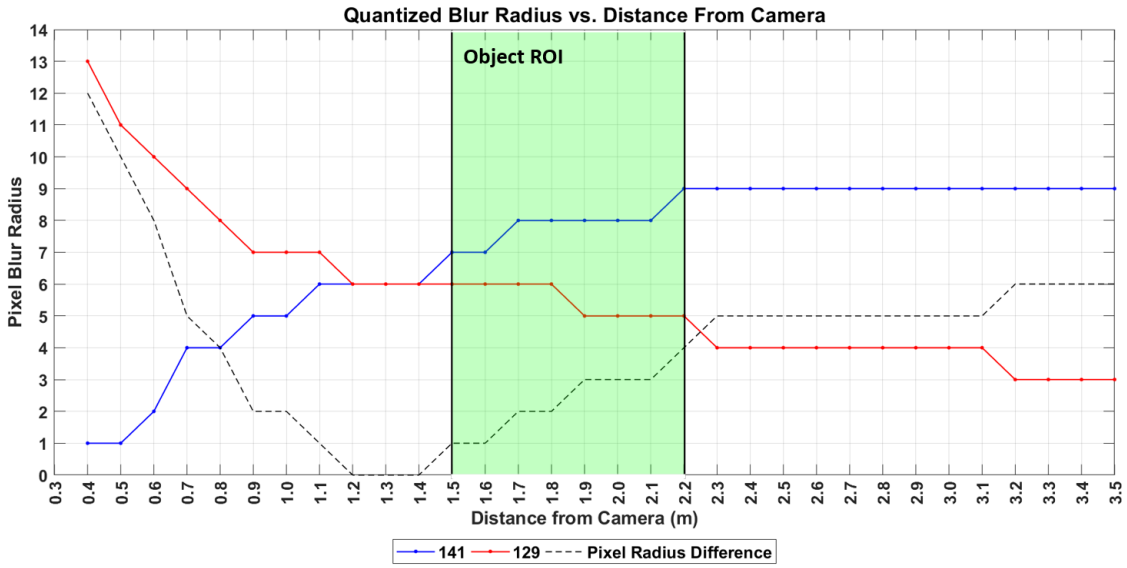


Fig. 7.15. Voltage Step Comparison: 141 and 129

where the depth map is intended to be inferred. For example, the region of interest for the surfaces in the real world dataset is between 1.5 and 2.2 meters.

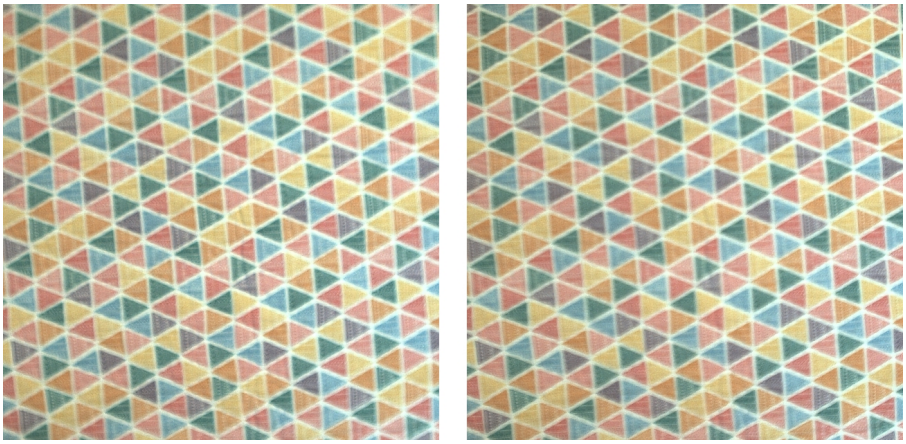
The second property is that the absolute difference between quantized blur radii should be unique within the region of interest, i.e. there should be no repeating values in the region. This, however, may not be achievable depending on the resolution/pixel size of the camera and the length of region of interest.

Table 7.5 shows the absolute difference in quantized blur radius between several of these candidate voltage step combinations. It can be seen that none of the potential candidates satisfy this requirement within the region of interest. But, they do come as close as possible based on the measured quantized blur radii. Table 7.5 also shows the average performance results of each of the three metrics for each of the potential candidates. The voltage step combination of 141 and 129 produced the best overall results. Figure 7.16 provides an example of the k35 test set with the image at voltage step 141 in Figure 7.16a and the image at voltage step 129 in Figure 7.16b.

Table 7.6 displays the overall performance results for for the voltage step combination of 141 and 129. Figure 7.17 shows the performance results for each scene and

Table 7.5.
Quantized Blur Radius Difference and Average Performance Results

Voltage Step	**Distance From Camera (m)**								**Performance Metric**		
Combination	**1.5**	**1.6**	**1.7**	**1.8**	**1.9**	**2.0**	**2.1**	**2.2**	**NMAE**	**NRMSE**	**SSIM**
142-130	4	4	5	5	5	6	7	7	0.0564	0.0700	0.8216
141-132	5	5	6	6	7	7	7	8	0.0584	0.0719	0.7963
141-131	4	4	5	5	5	6	6	7	0.0559	0.0693	0.8007
141-130	2	2	4	4	4	4	5	6	0.0526	0.0667	0.8115
141-129	**1**	**1**	**2**	**2**	**3**	**3**	**3**	**4**	**0.0515**	**0.0654**	**0.8099**
140-132	4	4	4	5	6	6	7	7	0.0595	0.0721	0.8421
140-131	3	3	3	4	4	5	6	6	0.0582	0.0712	0.8186
140-130	1	1	2	3	3	3	5	5	0.0553	0.0687	0.7894
137-128	4	4	3	3	2	1	1	1	0.0593	0.0712	0.8572



(a) (b)

Fig. 7.16. Example Input Image Pair for k35 (a) Image at Voltage Step 141 and (b) Image at Voltage Step 129

each exposure time. One of the more obvious trends is that the error produced by the third and sixth scenes is larger than those of the other scenes, especially the scenes that used the same texture. The k35 and k47 scenes are the ones that are located at approximately 1.5 meters from the camera (refer to Figure 3.10).

Table 7.6.
DfD-Net Real World Dataset Overall Test Performance Results

	NRMSE	**NMAE**	**SSIM**
Minimum	0.0506	0.0340	0.7360
Mean	0.0654	0.0515	0.8099
Maximum	0.1013	0.0936	0.8842
Std Deviation	0.0161	0.0173	0.0421

Table 7.7 outlines the top 5 and bottom 5 performance results for the pruned real world dataset. Figure 7.18 shows the top 5 and bottom 5 performance results of the DfD-Net trained and tested on the pruned real world dataset.

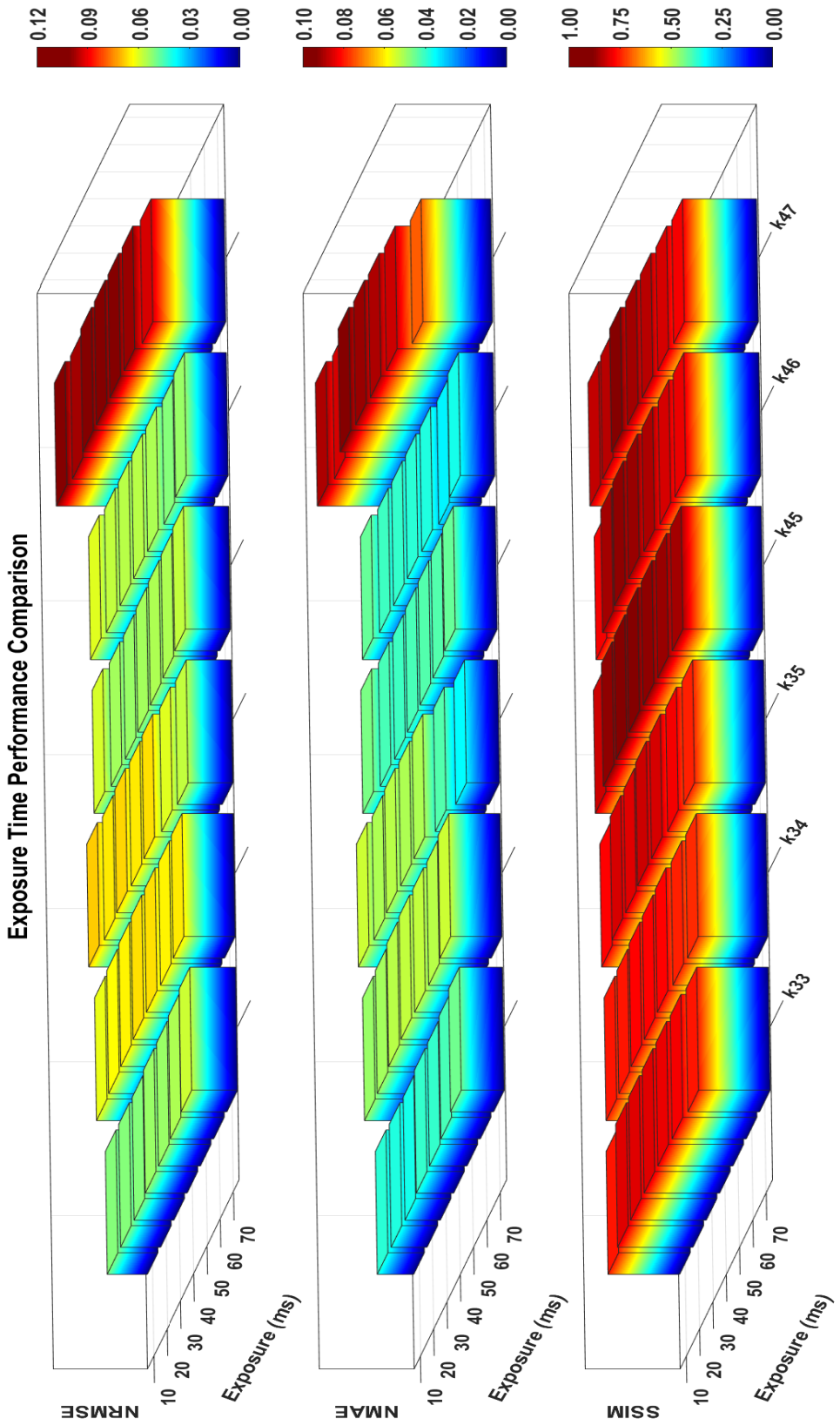


Fig. 7.17. DfD-Net Real World Exposure Time Performance Comparison

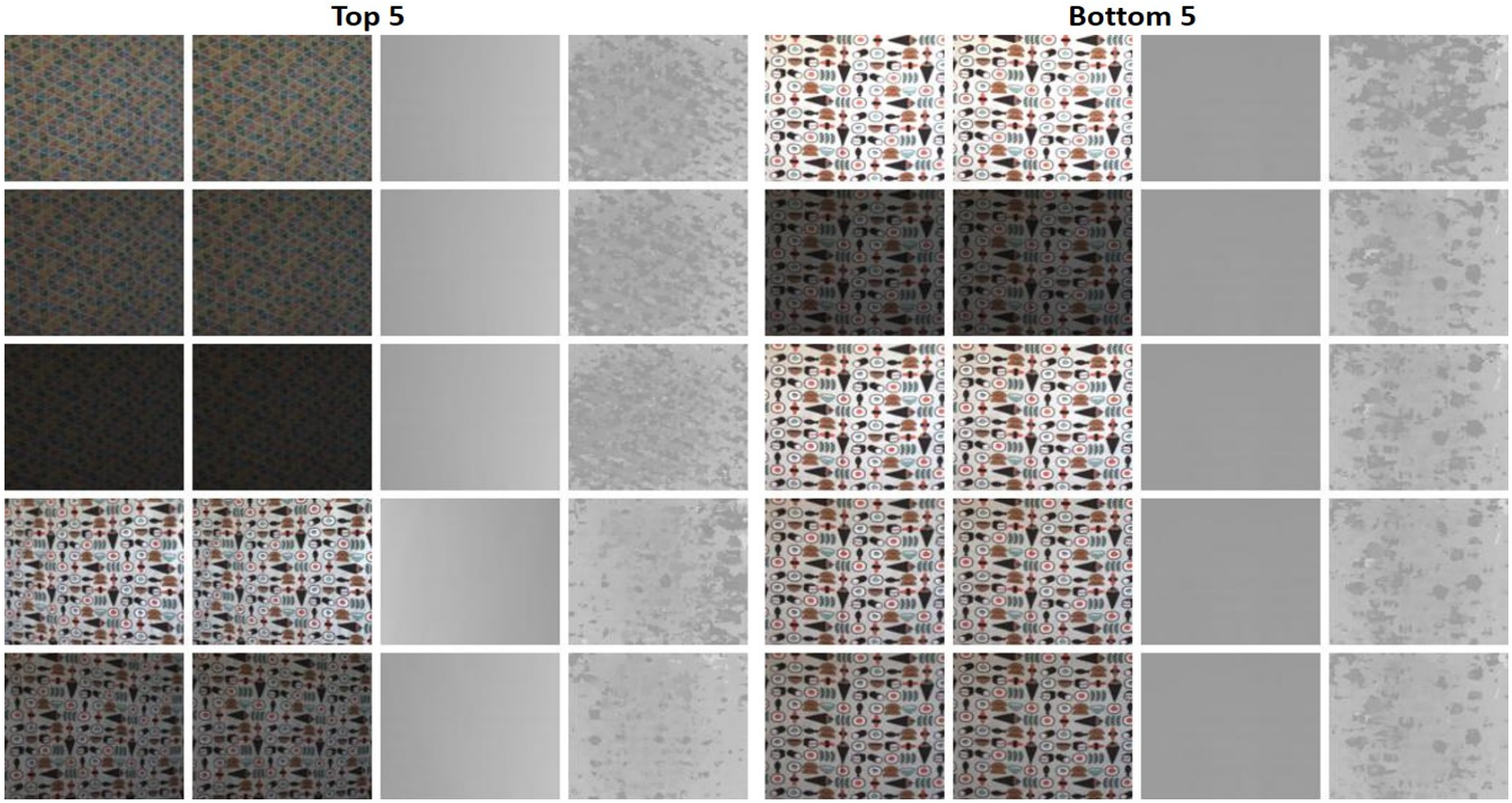


Fig. 7.18. Top 5 and Bottom 5 Performance Results for the Synthetically Blurred Real World Dataset. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) LIDAR Ground Truth Depth Map, (d) & (h) DfD-Net Computed Depth Map.

Table 7.7.
Top 5 and Bottom 5 DfD-Net Performance Results for the Real World Dataset

Name	**Exposure Time (ms)**	**NRMSE**	**NMAE**	**SSIM**
Top 5				
k33	30	0.05064	0.03733	0.80069
k33	20	0.05094	0.03794	0.79652
k33	10	0.05101	0.03853	0.75702
k46	60	0.05129	0.03396	0.81221
k45	30	0.05296	0.04092	0.87814
Bottom 5				
k47	60	0.0992	0.0869	0.8103
k47	50	0.1003	0.0898	0.8350
k47	40	0.1007	0.0912	0.8497
k47	10	0.1011	0.0895	0.8244
k47	30	0.1013	0.0936	0.8769

7.3 Summary

In this chapter the examination of the real world dataset was discussed. The real world dataset was initially blurred using the synthetic blurring process that was applied to the Middlebury College Stereo dataset. Analyzing the synthetically blurred results led to the discovery of issues with the camera resolution and the difficulties it has with resolving finer details at ranges greater than 2.5 meters. However, when analyzing the results of the real world dataset for the scenes that were between 1.5 and 2.2 meters from the camera the DfD-Net trained on the Middlebury dataset produced an average NRMSE, NMAE and SSIM of 0.0129, 0.0082 and 0.9702 respectively, with a standard deviation of 0.0105, 0.0069 and 0.0418. These results exceeded the test results of the DfD-Net tested on the Middlebury test set.

Further research into the microfluidic lens revealed issues with the lens's sensitivity to temperature and a hysteresis that occurs when changing between voltage steps. These issues need to be constantly monitored when creating a dataset, as they affect the image quality and can make it difficult to produce a consistent dataset.

Finally, this chapter discussed the performance of the DfD-Net trained and then tested solely on the real word dataset. The DfD-Net trained on the pruned real world dataset produced an average NRMSE, NMAE and SSIM of 0.0654, 0.0515 and 0.8099 with a standard deviation of 0.0161, 0.0173 and 0.0421 respectively. The next chapter will summarize the research presented in this dissertation.

8. SUMMARY

The motivation for this research was to improve upon the existing state of the art methods applied to the Depth from Defocus challenge. The current state of the art methods, while accurate are extremely computationally expensive and require a large amount of time to process. The recent advancements in deep learning architectures, especially in the realm of semantic segmentation lend themselves well to the DfD depth estimation task.

One of the major contributions of this research was the introduction of a deep learning architecture that can process a pair of in-focus and out-of-focus images and produce a depth map of the given scene. The new DfD-Net architecture is very resilient when it comes to varying lighting conditions. While the graph cuts method performance varied widely with exposure level, the DfD-Net performance is very consistent across the tested combination of illumination and exposure levels. Compared to the state of the art graph cuts algorithm, the DfD-Net architecture produced an average NRMSE of 0.0569, 0.0528 and 0.0558 for the images in the exposure level 0, 1 and 2 categories respectively for the synthetically blurred datasets. This is a improvement of approximately 43.66%, 31.25% and 24.70% for exposure level 0, 1 and 2 as compared to the overall average NRMSE results of the graph cuts method. Similarly, the DfD-Net architecture produced an average NMAE of 0.0188, 0.0142 and 0.0159 for each exposure level. This is a improvement of approximately 76.67%, 78.325% and 75.04% for exposure level 0, 1 and 2 as compared to the overall average NMAE results of the graph cuts method. The only metric where the graph cuts algorithm surpasses the DfD-Net is the SSIM metric. The DfD-Net produced an average SSIM of 0.9098, 0.9260 and 0.9182 for the exposure level 0, 1 and 2 categories, respectively. This is a decrease in performance of approximately 0.44%, 3.16% and 4.32% for exposure level 0, 1 and 2 as compared to the overall average SSIM results of the graph cuts method.

The run time of the DfD-Net far surpasses the run time of the graph cuts method. The average run time for the DfD-Net on a single CPU was approximately 10.0293 seconds per image, while the average run time of the graph cuts method was 209.65 seconds per image, a 95.22% reduction in run time. When running the DfD-Net on the Desktop 2 system, with the specifications outlined in Table B.1, the run time drops to an average of 0.3977 seconds per image. More importantly the algorithm runtime is not dependent on exposure of illumination level. This is not quite real time processing, however the run times are approaching this range.

To assess the robustness of the DfD-Net architecture a 9-fold cross validation was performed. The results of the 9-fold cross validation show that all but one fold produces similar results and those results were aligned with the test distribution presented in Section 5.4.1. The one fold that produced a poorly trained network was due to the fact that the training samples for the most heavily concentrated depth map values were under represented in the training process which means that the network did not see enough examples to accurately recreate the given depth values for the test dataset.

Another major contribution of this research (Section 6.1) is the application of the Particle Swarm Algorithm to improve the performance of the DfD-Net on the Middlebury College dataset. The PSO algorithm used 20 particles, where each particle was comprised of the number of filters, the height and the width for each convolutional filter in the DfD-Net. The particle also consisted of the type of activation layer and the binary choice of batch normalization layers. Finally, the particle also consisted of the training patch size.

The PSO algorithm determined a solution that produced an average NRMSE that was approximately 6.25% below the DfD-Net average NRMSE. Similarly the PSO DfD-Net produced an average NMAE that was 5.25% below the DfD-Net average NMAE. However, the PSO DfD-Net did not produce an average SSIM value that was better than the DfD-Net average SSIM value and was 0.26% lower than the DfD-Net SSIM value.

Additionally, we introduced (Section 6.2) a new method of clustering convolutional filter outputs to determine the minimum number of required convolutional filters within a deep learning network architecture. The results of this reduction method are network architectures that run faster than the baseline network architecture prior to reduction. This method also demonstrates that the network performance does not have to suffer as a result of the reduction process.

This method was applied to the DfD-Net network architecture and the Middlebury College dataset. The final outcome of the reduction method produced a DfD-Net that resulted in a decrease in the overall NRMSE value of approximately 3.4% when compared to the baseline mean NRMSE value. The NMAE and the SSIM were only slightly worse than the baseline means with a decrease of only 0.65% and 0.13% respectively. This network configuration produced an average speed increase of 44.85%, 40.04%, 14.91% and 26.70% for the Laptop, Jetson TX2, Desktop 1 and Desktop 2 respectively (hardware configurations detailed in Table B.1).

Finally, this research introduced the testing of the real world dataset (Chapter 7). Initially the dataset was synthetically blurred with the same synthetic blurring process that was applied to the Middlebury College Stereo dataset. This synthetically blurred dataset was tested using the DfD-Net that was trained on the Middlebury College dataset. Analyzing the results of the real world dataset for the scenes that were between 1.5 and 2.2 meters from the camera the DfD-Net trained on the Middlebury dataset produced an average NRMSE, NMAE and SSIM of 0.0129, 0.0082 and 0.9702 respectively, with a standard deviation of 0.0105, 0.0069 and 0.0418. These results exceeded the test results of the DfD-Net tested on the Middlebury test set.

Analyzing the synthetically blurred results led to the discovery of issues with the camera resolution and the difficulties it has with resolving finer details at ranges of 2.5 meters and greater. The real world dataset was pruned to only include surfaces in the range of 1.5 to 2.2 meters. The performance of the DfD-Net trained and then tested solely on the pruned real word dataset was assessed. The training produced an average NRMSE, NMAE and SSIM of 0.0654, 0.0515 and 0.8099 with a standard

deviation of 0.0161, 0.0173 and 0.0421 respectively. These results are visually very similar to the research conduct by Liu [21] with a similar microfluidic lens. However, his data was collected with objects that were very close to the camera (within 0.5 meters) and there was no means at the time to accurately measure the ground truth depth information, so a direct comparison cannot be made.

One final contribution of this research is the real world dataset itself. This dataset is available for future researchers and contains images taken with a microfluidic lens and ground truth data provided by a 2-D LIDAR. The final chapter in this dissertation will discuss some possible avenues for continued research in this area.

9. RECOMMENDATIONS FOR FUTURE RESEARCH

Based on the results of Chapter 7 the camera and lens used for the data collect have limitations that need to be overcome in order to further improve results. The limitations with the camera are the 1.3 MP resolution and sensor noise. Increasing the camera resolution will allow for capturing finer details at farther distances from the camera, which in turn will enable the DfD-Net, or newer deep learning algorithms to resolve depths at further distances. A reduction in camera noise will also improve the DfD-Net real world results.

The lens limitations with regards to temperature are manageable within the confines of laboratory experiments, but if a system like the one used for the real world data was employed outside the lab, controlling the temperature of the lens and camera may not be feasible. The second issue with the lens was the voltage step hysteresis observed during this research (Figure 7.12). This hysteresis is also manageable, but care must be taken to ensure that the lens is behaving as expected each time an image pair is captured. Because the lens driver system is an open loop control system there is no direct feedback, and therefore no guarantee that the lens is in the desired state. Each of these limitations affects the performance of the DfD-Net, and because control may not always be guaranteed it is recommended to begin looking at lenses that do not have these limitations.

An option for a lens is a voice coil motor lens which is what is currently in most cellphones. These lenses use a system similar to a speaker, and contain a stationary permanent magnet and a coil of wire around the lens. A voltage applied to the coil creates an opposing magnetic field that enables the lens to change focus distances and blurs. In addition, the resolution on most cellphone rear cameras far exceeds the current camera resolution that was used to collect the real world dataset.

Ground truth resolution is another major hurdle in developing a high quality dataset. While the LIDAR unit used in this research has very accurate distance measurements, the number of channels (64) prevents capturing small details at larger distances from the LIDAR. Newer promising technologies include solid state LIDAR units that claim to have similar resolution to today's cameras are starting to emerge on the consumer market.

On the algorithm side, a look at various other architectures should also be considered. The DfD-Net is designed to process a pair of images and produce a depth map of equal size. However, the ground truth LIDAR data is not to the same scale as the imagery data. Instead of re-sizing the LIDAR data to match the image data, development of a network architecture that would take in full size images, gradually reduce the size of the inputs in each dimension to produce a depth map equal in size to that of the original ground truth data. It would not be recommended to scale the imagery data down to the LIDAR data size, because the resolution of the image would be degraded to the point where blur information could potentially be lost.

The minimum number of required images for the DfD-Net and Depth from Defocus in general is two, however there is no maximum number. Some preliminary investigations were conducted using three input images, but the results were not conclusive. This is mainly due to the fact that the experiments were conducted prior to a complete understanding of the limitations of the data collection equipment. Future research should look at the use of 3 or more images to improve the overall performance of the DfD-Net and future architectures.

REFERENCES

REFERENCES

[1] D. F. Schaeffel, "Processing of Information in the Human Visual System," in *Handbook of Machine Vision*, A. Hornberg, Ed. Weinheim, Germany: WILEY-VCH Verlag GmbH & Co KGaA, 2006.

[2] A. Saxena, J. Schulte, and A. Y. Ng, "Depth Estimation Using Monocular and Stereo Cues," *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pp. 2197–2203, 2007.

[3] Wikipedia contributors, "Computer stereo vision - Wikipedia, The Free Encyclopedia," 2018, [Accessed 16-May-2018]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Computer_stereo_vision

[4] R. Jain, R. Kasturi, and B. G. Schunck, *Machine Vision*. San Fransisco: McGraw-Hill, Inc., 1995.

[5] Y. Schechner and N. Kiryati, "Depth from Defocus vs. Stereo: How Different Really Are They?" *International Journal of Computer Vision*, vol. 39, no. 2, pp. 141–162, 2000.

[6] T. Q. Sibley, *The Geometric Viewpoint: A Survey of Geometries*, ser. Addison-Wesley higher mathematics. Addison-Wesley, 1998.

[7] D. Scharstein and C. Pal, "Learning Conditional Random Fields for Stereo," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2007.

[8] H. Hirschmüller and D. Scharstein, "Evaluation of Cost Functions for Stereo Matching," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2007.

[9] H. Hirschmüller, "Accurate and Efficient Stereo Processing by Semi-Global Matching and Mutual Information," *Computer Vision and Pattern Recognition*, vol. 2, pp. 807–814, 2005.

[10] Y. Boykov, O. Veksler, and R. Zabih, "Fast Approximate Energy Minimization via Graph Cuts," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 11, pp. 1222–1239, 2001.

[11] J. Žbontar and Y. LeCun, "Stereo Matching by Training a Convolutional Neural Network to Compare Image Patches," *Journal of Machine Learning Research*, vol. 17, pp. 1–32, 2016.

[12] W. Luo, A. G. Schwing, and R. Urtasun, "Efficient Deep Learning for Stereo Matching," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 5695–5703.

[13] C. Hazirbas, L. Leal-Taixé, and D. Cremers, "Deep Depth From Focus," *CoRR*, vol. abs/1704.01085, 2017, [Accessed 15-May-2018]. [Online]. Available: http://arxiv.org/abs/1704.01085

[14] Wikipedia contributors, "Well-posed problem - Wikipedia, The Free Encyclopedia," 2017, [Accessed 16-May-2018]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Well-posed_problem

[15] M. Moeller, M. Benning, C. Schönlieb, and D. Cremers, "Variational Depth From Focus Reconstruction," *IEEE Transactions on Image Processing*, vol. 24, no. 12, pp. 5369–5378, 2015.

[16] V. Gaganov and A. Ignatenko, "Robust Shape from Focus via Markov Random Fields," in *GraphiCon*, October 2009, pp. 74–80.

[17] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," *CoRR*, vol. abs/1505.04597, 2015, [Accessed 10-July-2018]. [Online]. Available: http://arxiv.org/abs/1505.04597

[18] A. P. Pentland, "A New Sense for Depth of Field," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 9, no. 4, pp. 523–531, 1987.

[19] M. Watanbe and S. K. Nayar, "Rational Filters for Passive Depth from Defocus," *International Journal of Computer Vision*, vol. 27, no. 3, pp. 203–225, 1998.

[20] W. E. Crofts, "The Generation of Depth Maps via Depth-from-Defocus," Ph.D. dissertation, University of Warwick, 2007.

[21] C. Liu, "Three Dimensional Moving Pictures with a Single Imager and Microfluidic Lens," Ph.D. dissertation, Purdue University, 2016.

[22] S. Pasinetti, I. Bodini, M. Lancini, F. Docchio, and G. Sansoni, "A Depth From Defocus Measurement System Using a Liquid Lens Objective for Extended Depth Range," *IEEE Transactions on Instrumentation and Measurement*, vol. 66, no. 3, pp. 441–450, 2017.

[23] Z. Chen, X. Guo, S. Li, X. Cao, and J. Yu, "A Learning-based Framework for Hybrid Depth-from-Defocus and Stereo Matching," *CoRR*, vol. abs/1708.00583, 2017, [Accessed 11-July-2018]. [Online]. Available: http://arxiv.org/abs/1708.00583

[24] R. A. Herman, *A Treatise on Geometrical Optics*. London: Cambridge University Press, 1900.

[25] A. R. Greenleaf, *Photographic Optics*. New York: MacMillan Company, 1950.

[26] J. Geng, "Structured-light 3D surface imaging: a tutorial," *Adv. Opt. Photon.*, vol. 3, no. 2, pp. 128–160, June 2011, [Accessed 08-June-2018]. [Online]. Available: http://aop.osa.org/abstract.cfm?URI=aop-3-2-128

[27] D. Scharstein and R. Szeliski, "High-accuracy stereo depth maps using structured light," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, pp. 195–202, 2003.

[28] Corning, "Corning® Varioptic® Lenses, Market-Leading Adjustable Lens Solutions for Industrial Application," 2018, [Accessed 21-June-2018]. [Online]. Available: https://www.corning.com/media/worldwide/Innovation/documents/FINAL_CorningVariopticLenses_productbrochure_5.4.18_lowresWEB.pdf

[29] T. Chai and R. R. Draxler, "Root mean square error (RMSE) or mean absolute error (MAE)? Arguments against avoiding RMSE in the literature," *Geoscientific Model Development*, vol. 7, pp. 1247–1250, June 2014.

[30] C. J. Willmott and K. Matsuura, "Advantages of the Mean Absolute Error (MAE) Over the Root Mean Square Error (RMSE) in Assessing Average Model Performance," *Climate Research*, vol. 30, pp. 79–82, December 2005.

[31] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image Quality Assessment: From Error Visibility to Structural Similarity," *IEEE Transactions On Image Processing*, vol. 13, pp. 600–612, April 2004.

[32] R. J. Radke, *Computer Vision for Visual Effects*. Cambridge University Press, 2012.

[33] V. Kolmogorov and R. Zabih, "What Energy Functions can be Minimized via Graph Cuts?" *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 2, pp. 147–159, 2004.

[34] Y. Boykov and V. Kolmogorov, "An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 9, pp. 1124–1137, 2004.

[35] H. Ishikawa, "A Practical Introduction to Graph Cut," in *The 3rd Pacific-Rim Symposium on Image and Video Technology*, 2009, [Accessed 21-June-2018]. [Online]. Available: http://www.f.waseda.jp/hfs/PSIVT2009.pdf

[36] J. Lynn Arthur Steen, J. Arthur Seebach, *Counter Examples in Topology*. New York, New York: Holt, Reinhart and Winston, Inc., 1970.

[37] S. Park, S. Yu, B. Moon, S. Ko, and J. Paik, "Low-light Image Enhancement Using Variational Optimization-Based Retinex Model," *IEEE Transactions on Consumer Electronics*, vol. 63, no. 2, pp. 178–184, May 2017.

[38] C. Chen, Q. Chen, J. Xu, and V. Koltun, "Learning to See in the Dark," *CoRR*, vol. abs/1805.01934, 2018, [Accessed 12-July-2018]. [Online]. Available: http://arxiv.org/abs/1805.01934

[39] M. Everingham, L. V. Gool, C. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results," http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html, 2012, [Accessed 28-December-2017].

[40] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," *CoRR*, vol. abs/1411.4038, 2014. [Online]. Available: http://arxiv.org/abs/1411.4038

[41] R. Hahnloser and H. S. Seung, "Permitted and Forbidden Sets in Symmetric Threshold-Linear Networks," in *Conference on Neural Information Processing Systems*, December 2001.

[42] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 770–778.

[43] Wikipedia contributors, "Rectifier (neural networks) - Wikipedia, The Free Encyclopedia," 2018, [Accessed 15-July-2018]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Rectifier_(neural_networks)&oldid=847676984

[44] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," in *2015 IEEE International Conference on Computer Vision (ICCV)*, December 2015, pp. 1026–1034.

[45] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015. [Online]. Available: http://arxiv.org/abs/1502.03167

[46] D. E. King, "Dlib-ml: A Machine Learning Toolkit," *Journal of Machine Learning Research*, vol. 10, pp. 1755–1758, 2009.

[47] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *CoRR*, vol. abs/1412.6980, 2014, [Accessed 15-May-2018]. [Online]. Available: http://arxiv.org/abs/1412.6980

[48] D. E. King, "Automatic Learning Rate Scheduling That Really Works," http://blog.dlib.net/2018/02/automatic-learning-rate-scheduling-that.html, February 2018, [Accessed 16-July-2018].

[49] S. Falkner, A. Klein, and F. Hutter, "Practical Hyperparameter Optimization for Deep Learning," in *International Conference on Learning Representations (ICLR)*, 2018. [Online]. Available: \url{https://openreview.net/forum?id=HJMudFkDf}

[50] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian Optimization of Machine Learning Algorithms," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 2951–2959. [Online]. Available: http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf

[51] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and Robust Automated Machine Learning," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 2962–2970. [Online]. Available: http://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine-learning.pdf

[52] S. R. Young, D. C. Rose, T. P. Karnowski, S.-H. Lim, and R. M. Patton, "Optimizing Deep Learning Hyper-parameters Through an Evolutionary Algorithm," in *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*, ser. MLHPC '15. ACM, 2015, pp. 4:1–4:5. [Online]. Available: http://doi.acm.org/10.1145/2834892.2834896

[53] F. Ye, "Particle Swarm Optimization-Based Automatic Parameter Selection for Deep Neural Networks and Its Applications in Large-Scale and High-Dimensional Data," *PLoS ONE*, vol. 12, pp. 1186–1198, 2017. [Online]. Available: https://doi.org/10.1371/journal.pone.0188746

[54] P. R. Lorenzo, J. Nalepa, M. Kawulok, L. S. Ramos, and J. R. Pastor, "Particle Swarm Optimization for Hyper-Parameter Selection in Deep Neural Networks," in *2017 The Genetic and Evolutionary Computation Conference (GECCO)*, July 2017, pp. 481–488.

[55] J. Kennedy and R. Eberhart, "Particle Swarm Optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4, Nov 1995, pp. 1942–1948.

[56] M. Clerc and J. Kennedy, "The particle swarm - explosion, stability, and convergence in a multidimensional complex space," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 1, pp. 58–73, Feb 2002.

[57] "NVIDIA® Jetson Systems," https://www.nvidia.com/en-us/autonomous-machines/embedded-systems-dev-kits-modules/, [Accessed 11-March-2019].

[58] Y. Fu, "Xilinx ML Suite Overview," https://www.xilinx.com/publications/events/machine-learning-live/colorado/xDNN_ML_Suite.pdf, 2018, [Accessed 11-March-2019].

[59] J. L. Chu and A. Krzyżak, *Analysis of Feature Maps Selection in Supervised Learning Using Convolutional Neural Networks*, M. Sokolova and P. van Beek, Eds. Springer, Cham, 2014, vol. 8436.

[60] A. RoyChowdhury, P. Sharma, and E. G. Learned-Miller, "Reducing Duplicate Filters in Deep Neural Networks," in *Neural Information Processing Systems 2018*, 2018.

[61] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-Based Learning Applied to Document Recognition," *Proceedings of the IEEE*, vol. 11, no. 86, pp. 2278–2324, 1998.

[62] T. Yang, Y. Chen, and V. Sze, "Designing Energy-Efficient Convolutional Neural Networks using Energy-Aware Pruning," *CoRR*, vol. abs/1611.05128, 2016, [Accessed 9-April-2019]. [Online]. Available: http://arxiv.org/abs/1611.05128

[63] T. Kohonen, "The Self-Organizing Map," in *Proceedings of the IEEE*, vol. 78, no. 9, September 1990, pp. 1464–1480.

[64] C. Liu and L. A. Christopher, "Three Dimensional Moving Pictures with a Single Imager and Microfluidic Lens," *IEEE Transactions on Consumer Electronics*, vol. 60, no. 2, pp. 258–266, 2014.

[65] K. De and V. Masilamani, "Fast no-reference image sharpness measure for blurred images in discrete cosine transform domain," in *2016 IEEE Students Technology Symposium (TechSym)*, Sept 2016, pp. 256–261.

[66] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," in *4th International Conference on Learning Representations, (ICLR) 2016*, May 2016.

[67] X. Jin, C. Xu, J. Feng, Y. Wei, J. Xiong, and S. Yan, "Deep learning with s-shaped rectified linear activation units," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, ser. AAAI'16. AAAI Press, 2016, pp. 1737–1743.

APPENDIX

A. DATASET SCENE OVERVIEW

Figure A.1 shows an overview of the scenes within the Middlebury College Stereo Vision Dataset [7, 8]. The representative images are all from the illumination level 2, exposure level 1 images.

Figures A.2 and A.3 shows an overview of the scenes within the Real World dataset. The representative images shown were all taken from the 50 ms exposure time images and the voltage step of 135.

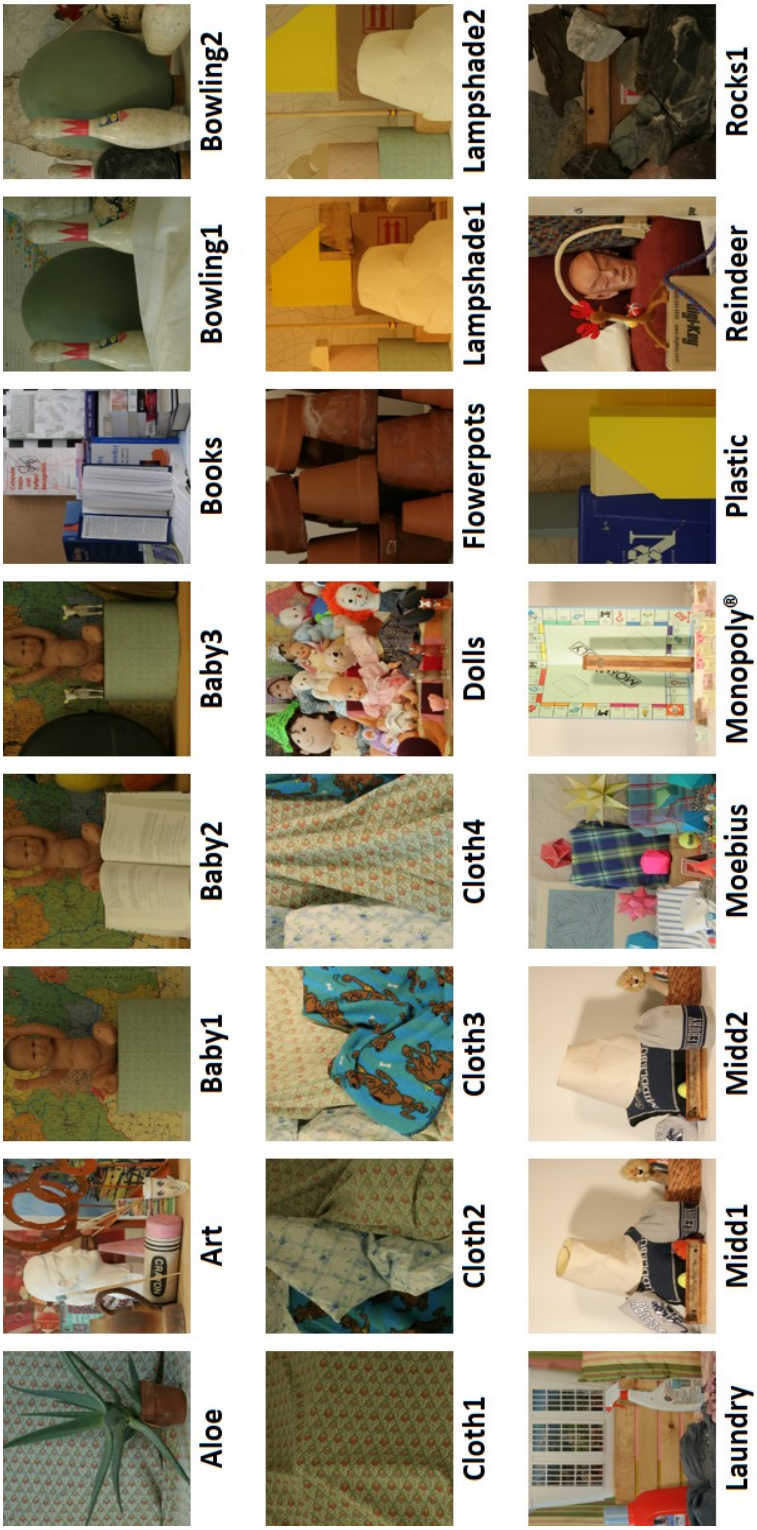


Fig. A.1. Middlebury College Dataset Overview

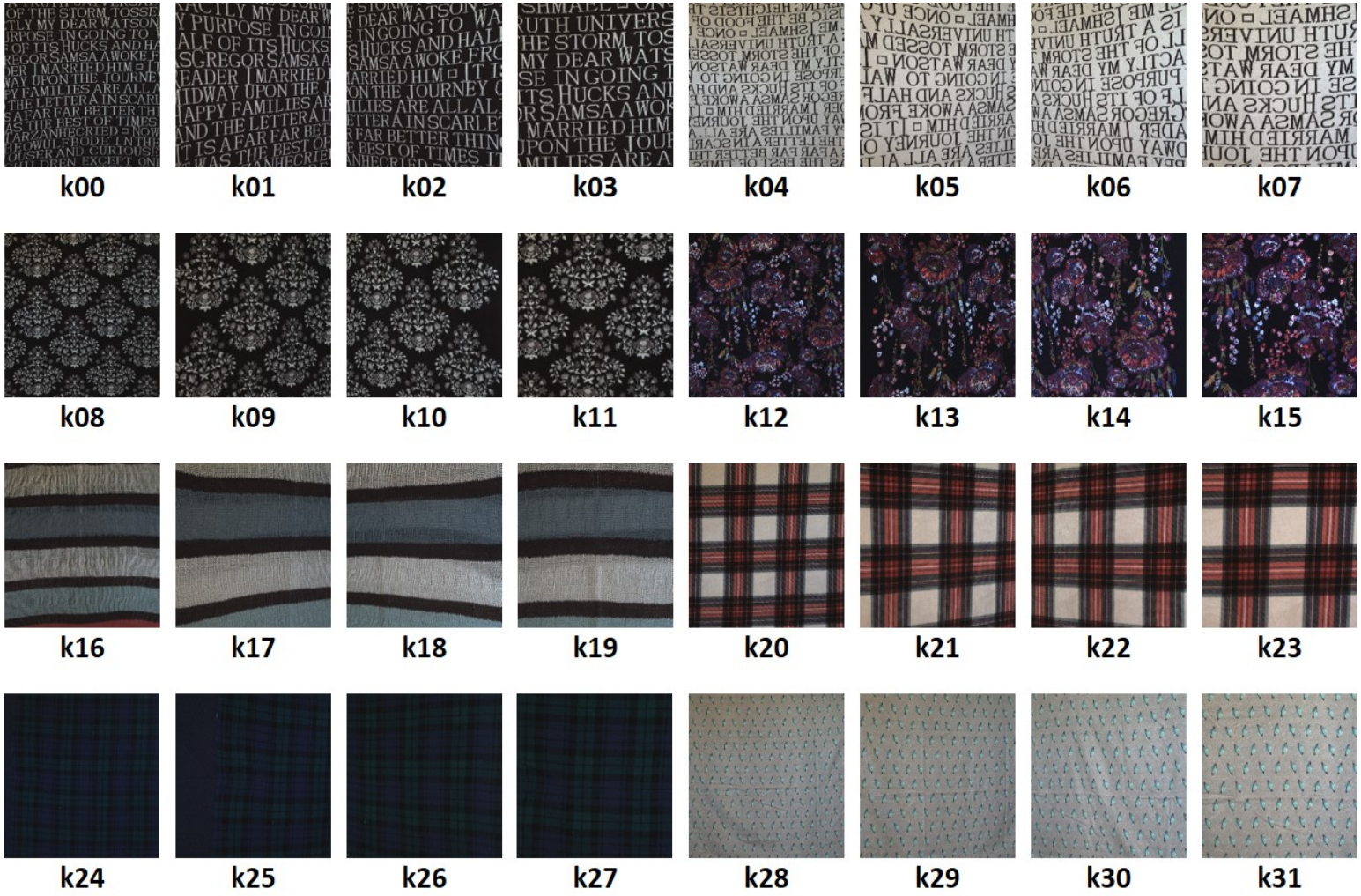


Fig. A.2. Real World Dataset Overview - Part 1

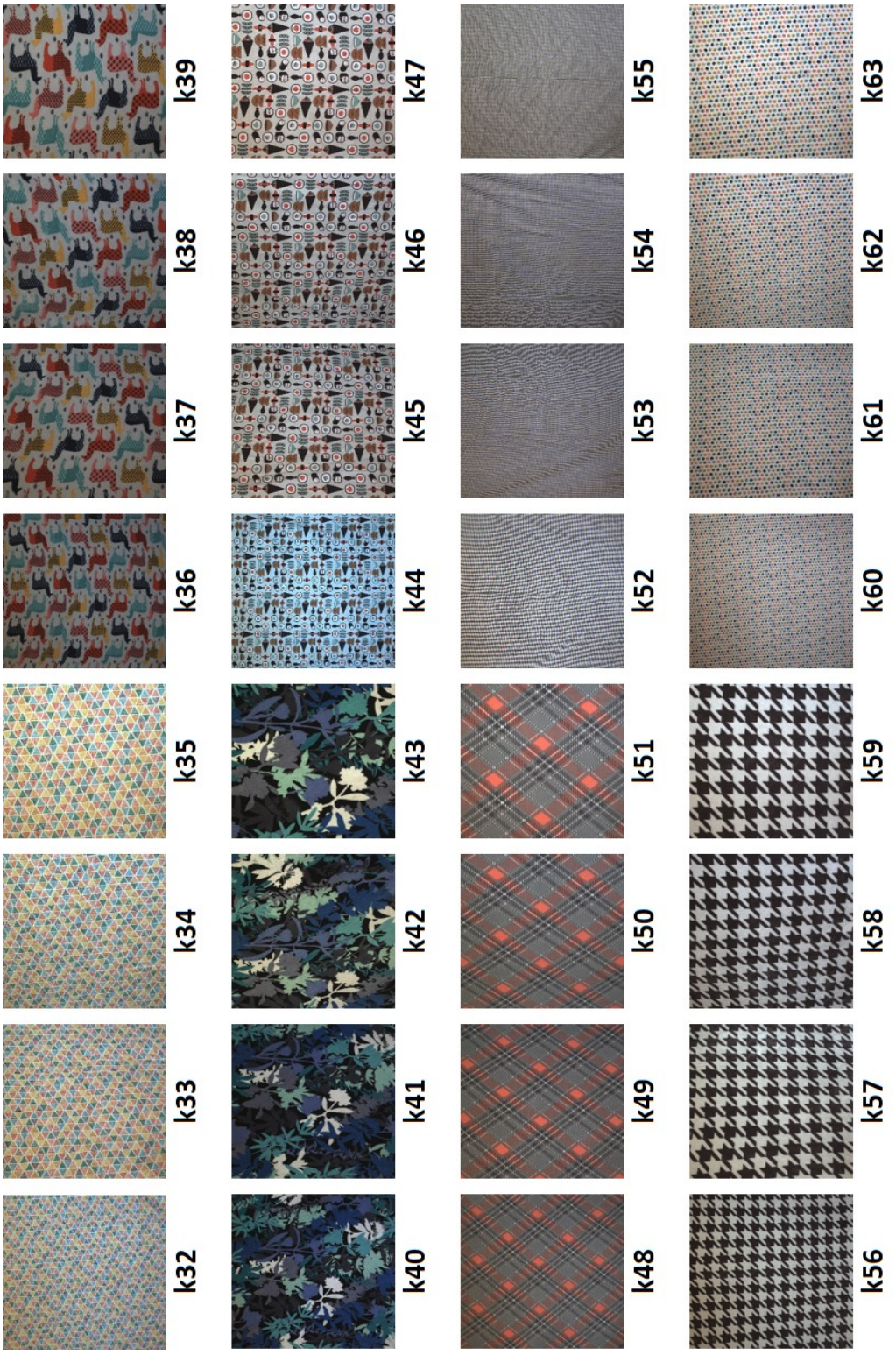


Fig. A.3. Real World Dataset Overview - Part 2

B. TIMING ANALYSIS TEST HARDWARE

Table B.1 outlines the operating system (OS) the CPU hardware and the GPU hardware used to conduct the timing tests. These platforms were chosen because they represent a broad spectrum of technologies that could be used for the implementation of deep learning architectures. The Jetson Tx2 was configured with NVIDIA®'s JetPack 3.3 with an OS derived from Ubuntu 16.04 but configured for an ARM processor.

Table B.1.
Test Hardware Platforms

Platform / OS	**CPU**	**GPU**
Laptop Windows® 8.1	Intel® Core™ i7-4700HQ @ 2.40GHz	NVIDIA® GTX770m Kepler @ 705 MHz
Jetson TX2 Ubuntu 16.04	ARM Cortex-A57 @2GHz, NVIDIA® Denver2 @2GHz	NVIDIA® Tegra Pascal @ 1300 MHz
Desktop 1 Windows® 10	Intel® Core™ i7-8700 @ 3.20GHz	NVIDIA® Titan Xp Pascal @ 1911 MHz
Desktop 2 Ubuntu 16.04	AMD Ryzen™ 7 1800X @ 3.60GHz	NVIDIA® GTX1080 Pascal @ 1810 MHz

C. DFD-NET MIDDLEBURY COLLEGE DATASET PSO RESULTS

C.1 PSO Algorithm Details

The PSO variant used in this research, developed by Clerc and Kennedy [56], uses a constriction factor to speed up the rate of convergence. The constriction factor is defined in Equation C.1 where $\phi = c_1 + c_2$ with $c_1 = 2.4$ and $c_2 = 2.1$. The constant c_1 is known as the cognitive constant as it influences the behavior of the particles for the current iteration. The constant c_2 is known as the social constant as it influences the behavior of the particles between the current iteration of the algorithm and the current global best result from the current iteration and all previous iterations.

$$\kappa = \frac{2}{\left|2 - \phi - \sqrt{\phi^2 - 4\phi}\right|} \tag{C.1}$$

The update function for the velocity component of the PSO particle is defined in Equation C.2 where $x_i^{(k)}$ is the i$^{\text{th}}$ particle at the k$^{\text{th}}$ iteration of the algorithm. This particle contains all of the optimization parameters. The $p_i^{(k)}$ term, also known as the personal best or p-best term, is the i$^{\text{th}}$ particle's position that has resulted in the smallest objective function value for all previous algorithm iterations. The $g^{(k)}$ term, or g-best, is a single particle that has minimized the objective function for all particles and iterations. The '$\circ$' symbol is the Hadamard product operator and indicates a point-wise multiplication of vectors or matrices versus the traditional matrix multiplication. The velocity of the particle is used to control the direction and rate of movement for a given particle. This movement is what allows the PSO algorithm to search the objective function's space for an optimal solution. Once the updated

velocity for each particle has been calculated, each element in the velocity component is run through a limiting function. This function constrains the minimum/maximum amount of movement that the particle can move in a single iteration. This prevents the particle from making excessively large jumps across the search space. This clamping effectively allows the particle to explore a more local area within the search space while moving towards the globally optimal solution.

$$v_i^{(k+1)} = \kappa \left(v_i^{(k)} + c_1 r_i^{(k)} \circ \left(p_i^{(k)} - x_i^{(k)} \right) + c_2 s_i^{(k)} \circ \left(g^{(k)} - x_i^{k} \right) \right) \tag{C.2}$$

The particle update function is defined in Equation C.3. Once the particle has been updated it is run through another limiting function to ensure that the individual particle components do not go out-of-bounds for the problem.

$$x_i^{(k+1)} = x_i^{(k)} + v_i^{(k+1)} \tag{C.3}$$

The steps required to perform the PSO algorithm are outline in Algorithm C.1. Where k is the iteration number and N is the number of particles used for each iteration. In this research N was set to 20. There are two sections for this algorithm: 1) initialization and 2) the main routine. The initialization section begins by setting $k = 0$ and then for each particle a random set of parameters ($x_i^{(0)}$) and velocities ($v_i^{(0)}$) were generated. Next each particle is placed into $p_i^{(0)}$. The DfD-Net was then trained with each of the $x_i^{(0)}$ particles. Once each of the N DfD-Net variants was trained the objective function (Equation 6.1) was evaluated and the particle that resulted in the smallest objective function was placed into $g^{(0)}$.

The main routine begins by generating two uniformly distributed random vectors $r_i^{(k+1)}$ and $s_i^{(k+1)}$ in the range of (0,1), each of which are the same size as the $x_i^{(k)}$ particle. Next the velocity and particle position are updated according to equations C.2 and C.3 respectively. The DfD-Net is then trained again with each of the twenty $x_i^{(k+1)}$ particles. Once each of the twenty DfD-Net variants finished training, the

Algorithm C.1. Particle Swarm Optimization Algorithm

Initialization:

k $\leftarrow 0$

for $i = 0$ **to** N **do**

generate random $x_i^{(0)}$ and $v_i^{(0)}$

set $p_i^{(0)} = x_i^{(0)}$

train DfD-Net using $x_i^{(0)}$ parameters

evaluate $f(x_i^{(0)})$ according to Equation 6.1

$g^{(0)} = \underset{x \in \left(x_1^0, \ldots, x_N^0\right)}{\arg\min} f(x_i^{(0)})$

Main Routine:

repeat

for $i = 0$ **to** N **do**

generate uniform random $r_i^{(k+1)}$ and $s_i^{(k+1)}$ in the range (0,1)

update $v_i^{(k+1)}$ according to Equation C.2

update $x_i^{(k+1)}$ according to Equation C.3

train DfD-Net using $x_i^{(k+1)}$ parameters

evaluate $f(x_i^{(k+1)})$ according to Equation 6.1

if $f(x_i^{(k+1)}) < f(p_i^{(k)})$ **then**

$p_i^{(k+1)} = x_i^{(k+1)}$

else

$p_i^{(k+1)} = p_i^{(k)}$

if $\exists\, i \in (1, \ldots, N)\, s.t.\, f(x_i^{(k+1)}) < f(g^{(k)})$ **then**

$g^{(k+1)} = x_i^{(k+1)}$

else

$g^{(k+1)} = g^{(k)}$

k $\leftarrow$ k + 1

until *stopping criteria is met*

objective function was again evaluated. Next, the current objective function value was compared to the previous iteration objective function value for each particle. If the current value is less than the previous value the $x_i^{(k+1)}$ particle is placed into $p_i^{(k+1)}$. Otherwise, the previous p-best for the particle ($p_i^{(k)}$) is placed into $p_i^{(k+1)}$. Once all of the particles have been evaluated, if there exists a particle in the current iteration that produced a smaller objective function value than the current global best, it is placed into the new global best particle ($g^{(k+1)}$). Otherwise the previous iteration's globally best particle is placed into $g^{(k+1)}$. Finally the iteration number is incremented by one. The main routine is repeated until the stopping criteria was met, which for this research the only stopping criteria used was to set the maximum number of iterations to 40.

Table C.1 outlines the parameters within the convolutional filter layer. The particle limits are the lower and upper limits for the particle values. These limits represent a filter size ranging between 1x1 and 9x9. For the number of filters the range is between 8 and 512. The velocity limit column indicates the minimum/maximum change that the velocity component can take in one iteration. These limits were applied to all convolutional layers. The upsampling and downsampling convolutional blocks had a reduced set of parameters to optimize. The filter sizes were kept at their original values of 2x2 to ensure that tensor dimensions were maintained and only the number of filters in the layer was optimized. The final convolutional layer was also left completely unchanged.

Table C.1.
Convolutional Layer Optimization Parameters

Parameter	**Velocity Limits**	**Particle Limits**	**Mapping Function**
Filter width	[-2, 2]	[0, 4]	$2\lfloor w + 0.5 \rfloor + 1$
Filter height	[-2, 2]	[0, 4]	$2\lfloor h + 0.5 \rfloor + 1$
Filter number	[-16, 16]	[8, 512]	$\lfloor n + 0.5 \rfloor$

Table C.2.
Activation Layer Optimization Parameters

PSO Mapping	**Activation**	**Activation Function Equation**
0	ReLU [41]	$f(x) = \begin{cases} x, & x > 0 \\ 0, & otherwise \end{cases}$
1	pReLU [44]	$f(x) = \begin{cases} x, & x > 0 \\ \alpha x, & otherwise \end{cases}$
2	Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$
3	Hyperbolic Tan	$f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
4	ELU [66]	$f(x) = \begin{cases} x, & , \quad if x \geq 0 \\ \alpha(e^{x}-1), & otherwise \end{cases}$
5	sReLU [67]	$f(x) = \begin{cases} t_i^r + \alpha_i^r(x_i - t_i^r), & x_i \geq t_i^r \\ x_i, & t_i^r > x_i > t_i^l \\ t_i^l + \alpha_i^l(x_i - t_i^l), & x_i \leq t_i^l \end{cases}$

Table C.2 outlines the activation functions and their governing equations that were used as potential optimization candidates. Because the PSO algorithm only operates on numerical values a mapping from an activation function to a numerical value was created (first column). The velocity limit for the activation function component of the particle was limited to [-1, 1]. The particle limit for the activation function was limited to [0, 5]. However, instead of hard clamping the particle value to the

minimum or maximum limits the actual particle values were allowed to wrap around in a modulo n fashion, where n is the number of activation functions available to the PSO algorithm to test. For example, if the PSO algorithm determined that an activation function mapping to the number six should be used the actual numerical mapping would be wrapped around to the first entry and the ReLU activation function would be selected. Similarly, if the PSO algorithm determined that the activation function mapping to the number -1 should be used then the sixth activation function would be selected. This was done because there is no numerical relationship between the activation functions and they could have be placed in any order.

For the batch normalization layers the optimization choice was to either use a batch normalization layer or not to use a batch normalization layer. This was mapped into a binary decision of either '0' (don't use the batch normalization layer) or '1' (use the batch normalization layer). The velocity component for the batch normalization elements of the particles were limited to [-1, 1] and the batch normalization portion of the particle was limited to [0, 1].

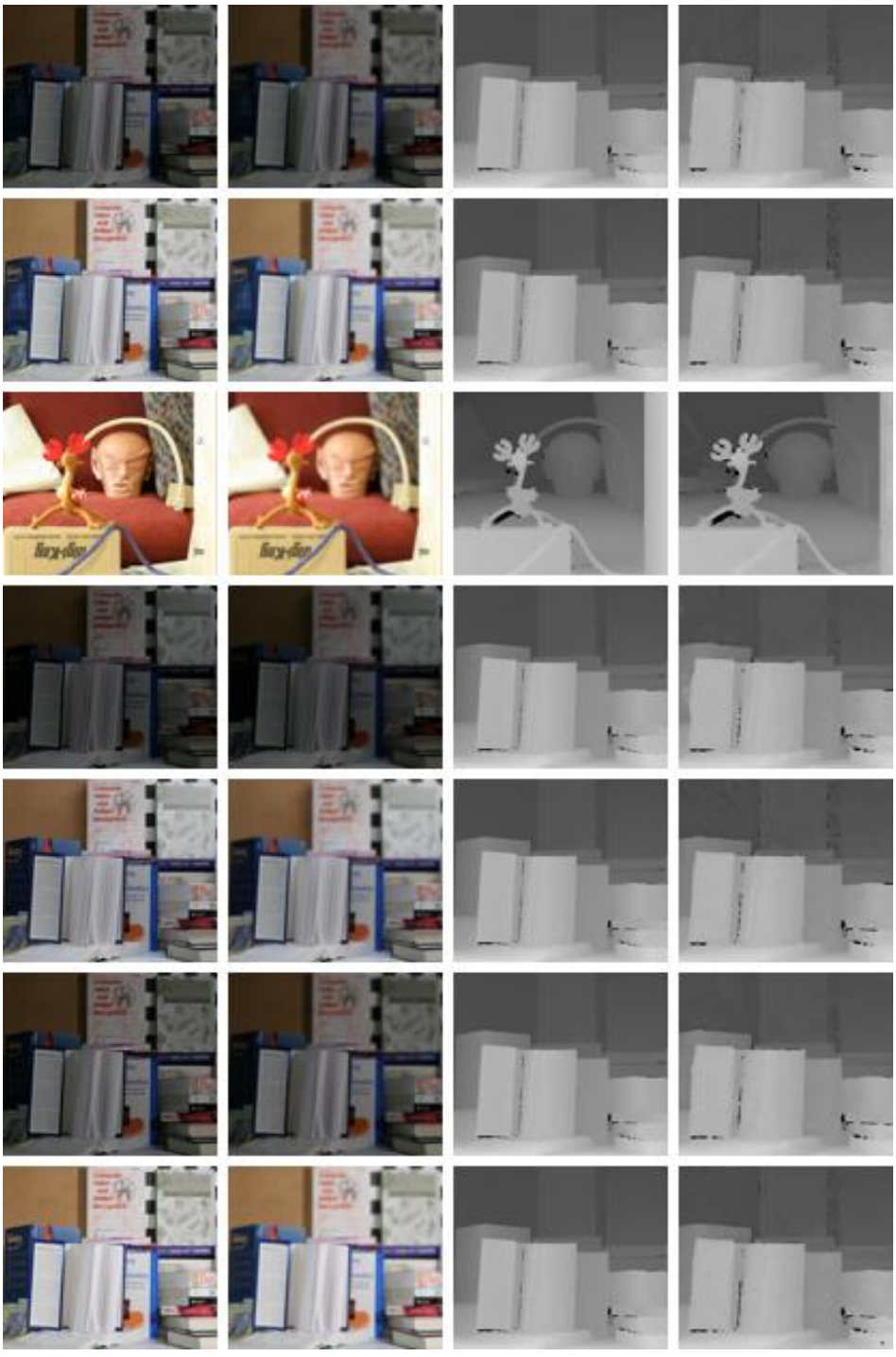
Table C.3.
Training Crop Size Optimization Parameters

Parameter	**Velocity Limits**	**Particle Limits**	**Mapping Function**
Crop width	[-1, 1]	[0, 13]	$4\lfloor n + 0.5 \rfloor + 12$
Crop height	[-1, 1]	[0, 13]	$4\lfloor n + 0.5 \rfloor + 12$

Table C.3 outlines the training patch size limiting values and mapping function. The particle limits represent a crop size that was allowed to vary between 12x12 pixels on the low end and 64x64 pixels on the upper end in 4x4 pixel increments. Only one parameter was used to determine the crop size which means that the crop sizes were always square.

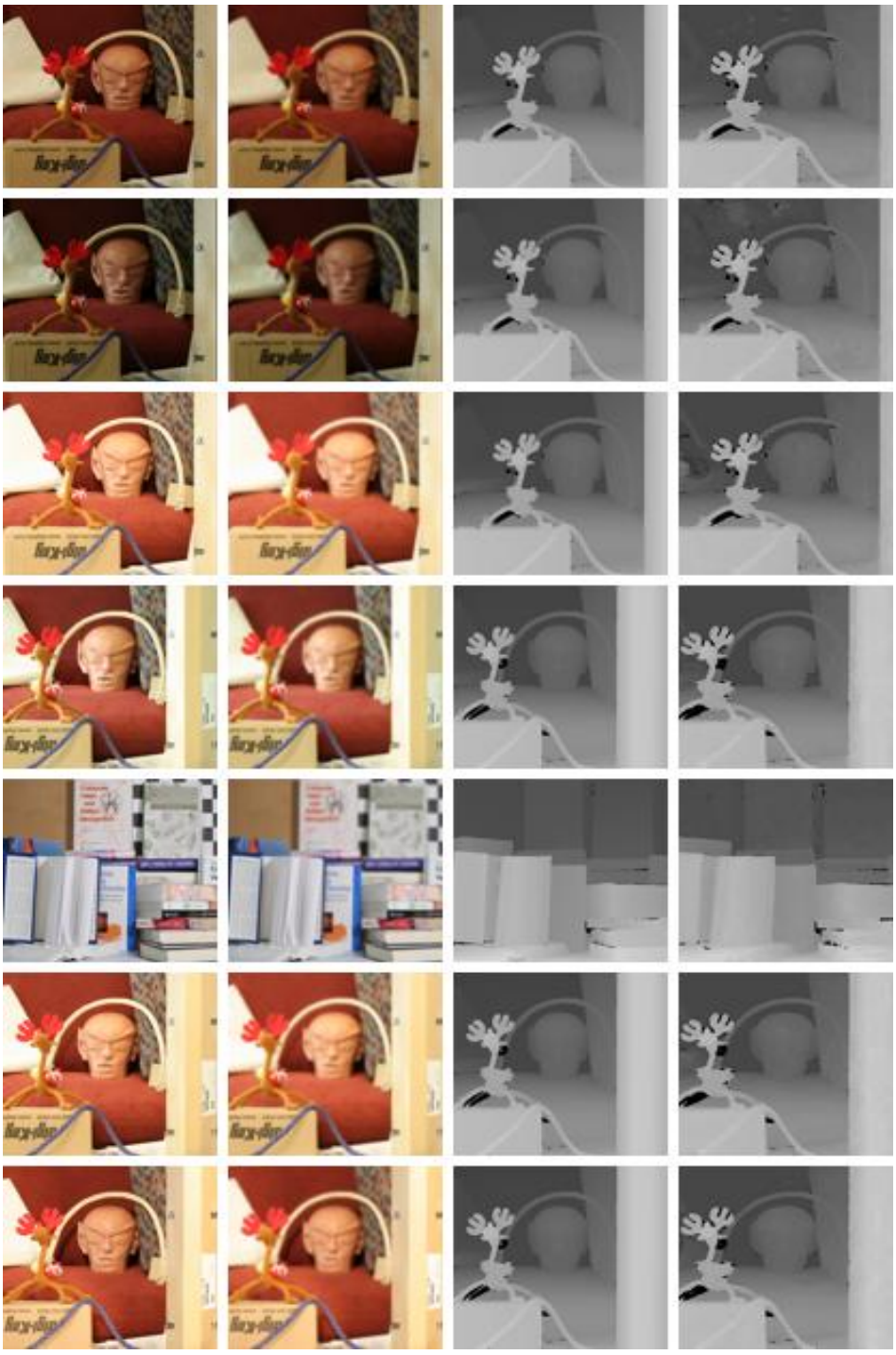
C.2 PSO Algorithm Results

Figures C.1 through C.8 show the results of the PSO algorithm applied to the DfD-Net trained and tested on the Middlebury College dataset [7,8]. The images are arranged in the order of lowest NRMSE score to highest NRMSE score.



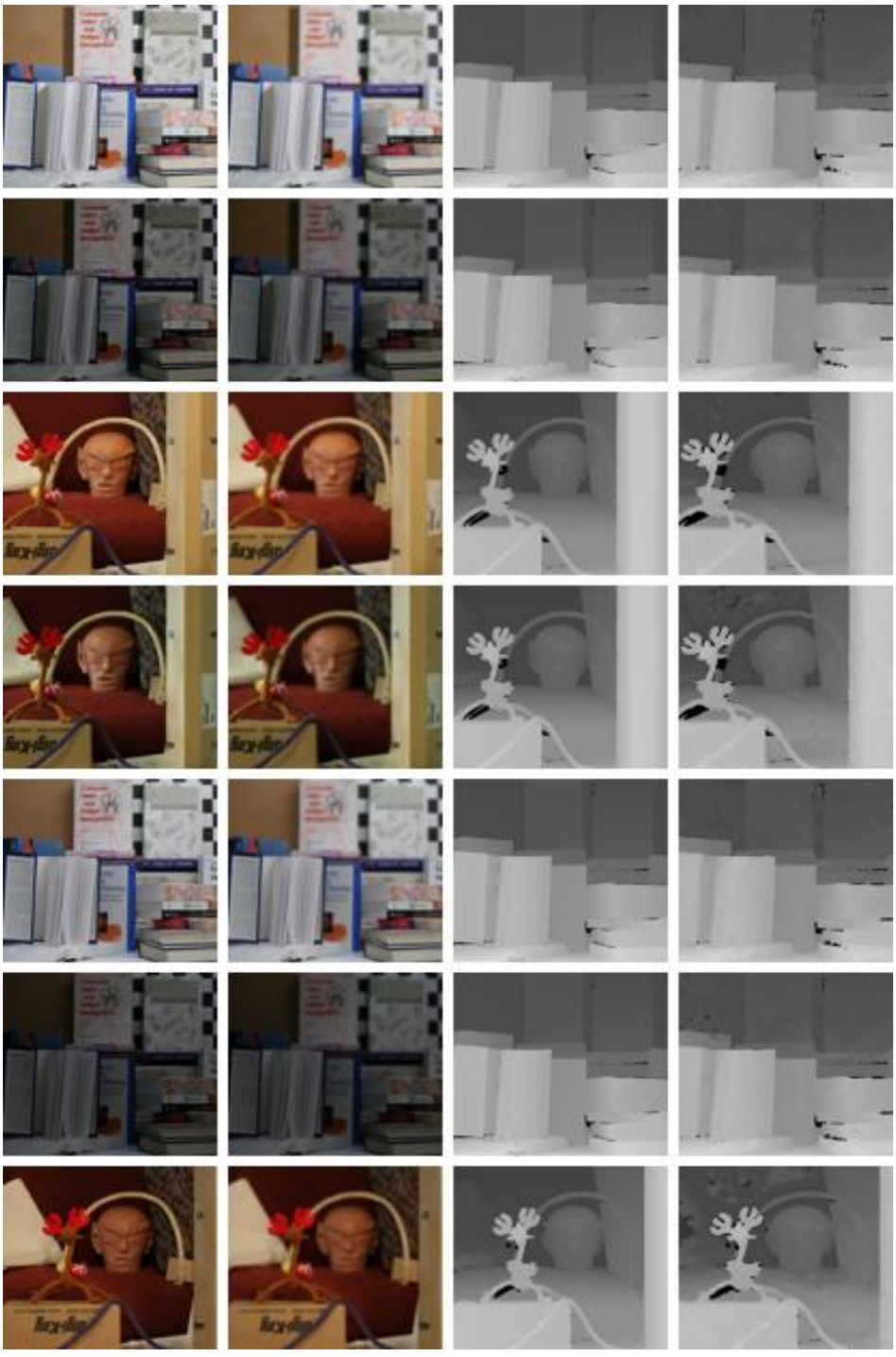
(a) (b) (c) (d)

Fig. C.1. PSO Performance Results for the Middlebury College Dataset - Part 1. (a) In-focus Image, (b) Out-of-focus Image, (c) Ground Truth Depth Map and (d) DfD-Net Computed Depth Map.



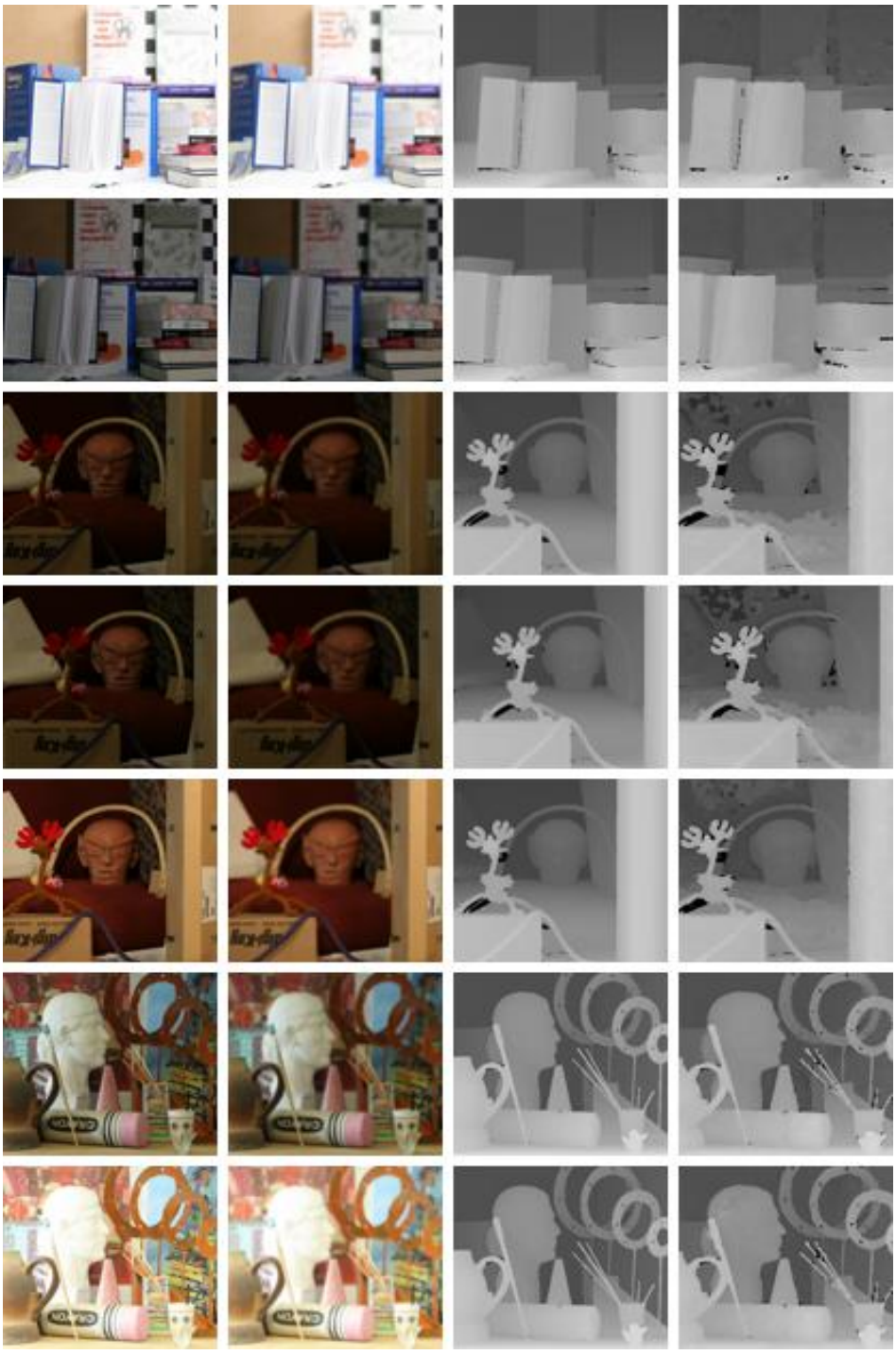
(a) (b) (c) (d)

Fig. C.2. PSO Performance Results for the Middlebury College Dataset - Part 2. (a) In-focus Image, (b) Out-of-focus Image, (c) Ground Truth Depth Map and (d) DfD-Net Computed Depth Map.



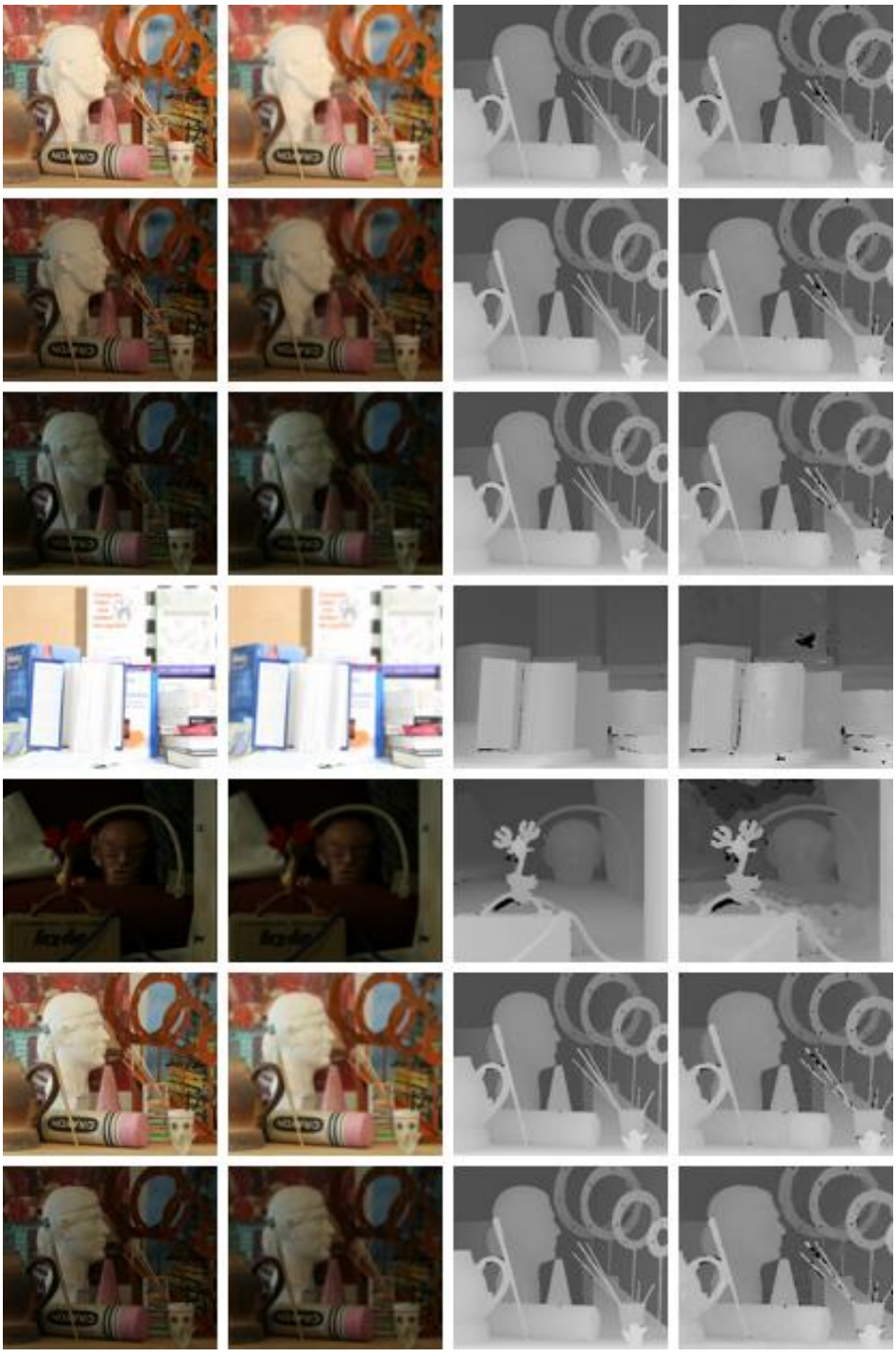
(a) (b) (c) (d)

Fig. C.3. PSO Performance Results for the Middlebury College Dataset - Part 3. (a) In-focus Image, (b) Out-of-focus Image, (c) Ground Truth Depth Map and (d) DfD-Net Computed Depth Map.



(a) (b) (c) (d)

Fig. C.4. PSO Performance Results for the Middlebury College Dataset - Part 4. (a) In-focus Image, (b) Out-of-focus Image, (c) Ground Truth Depth Map and (d) DfD-Net Computed Depth Map.

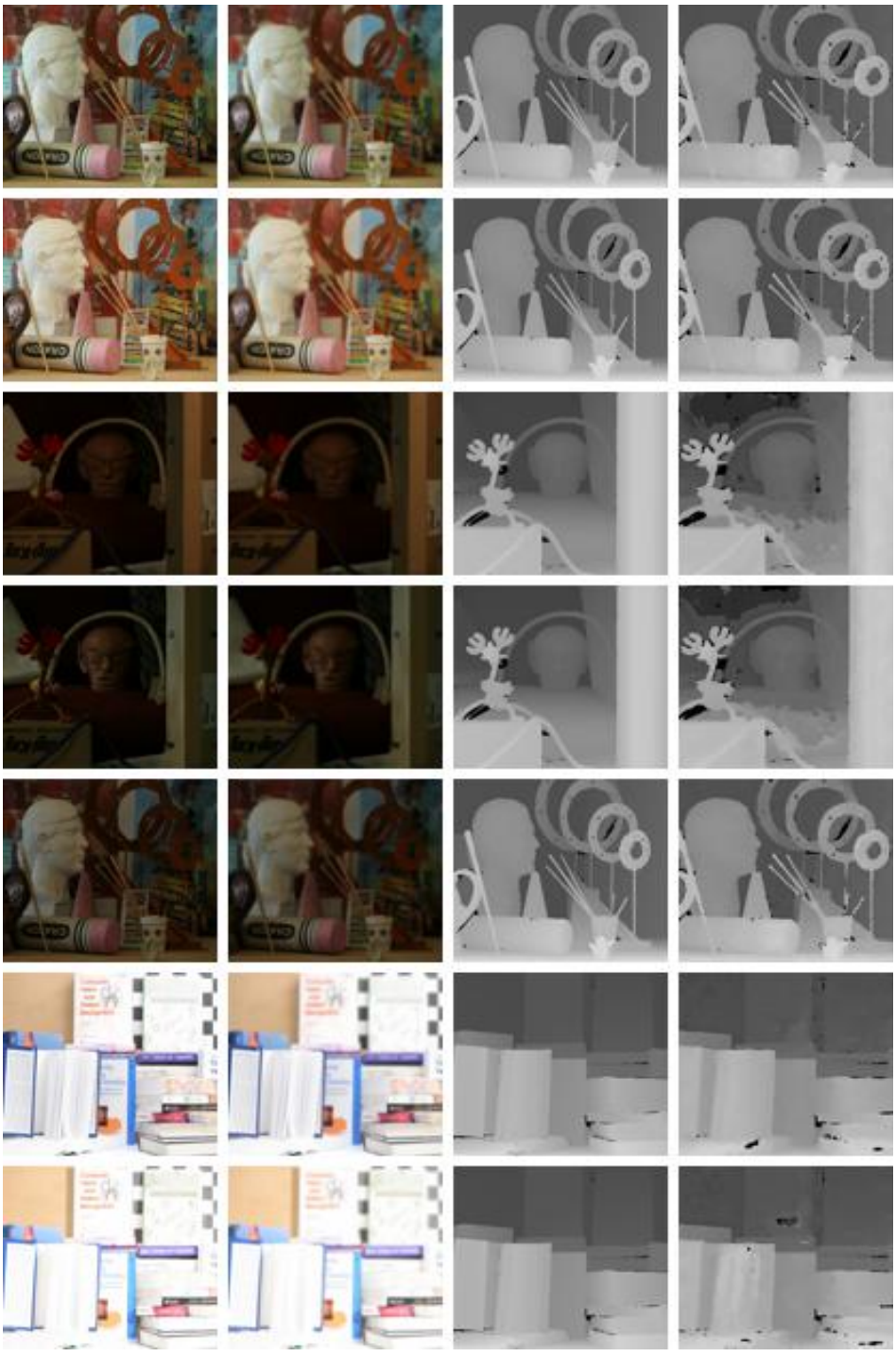


(a) (b) (c) (d)

Fig. C.5. PSO Performance Results for the Middlebury College Dataset - Part 5. (a) In-focus Image, (b) Out-of-focus Image, (c) Ground Truth Depth Map and (d) DfD-Net Computed Depth Map.

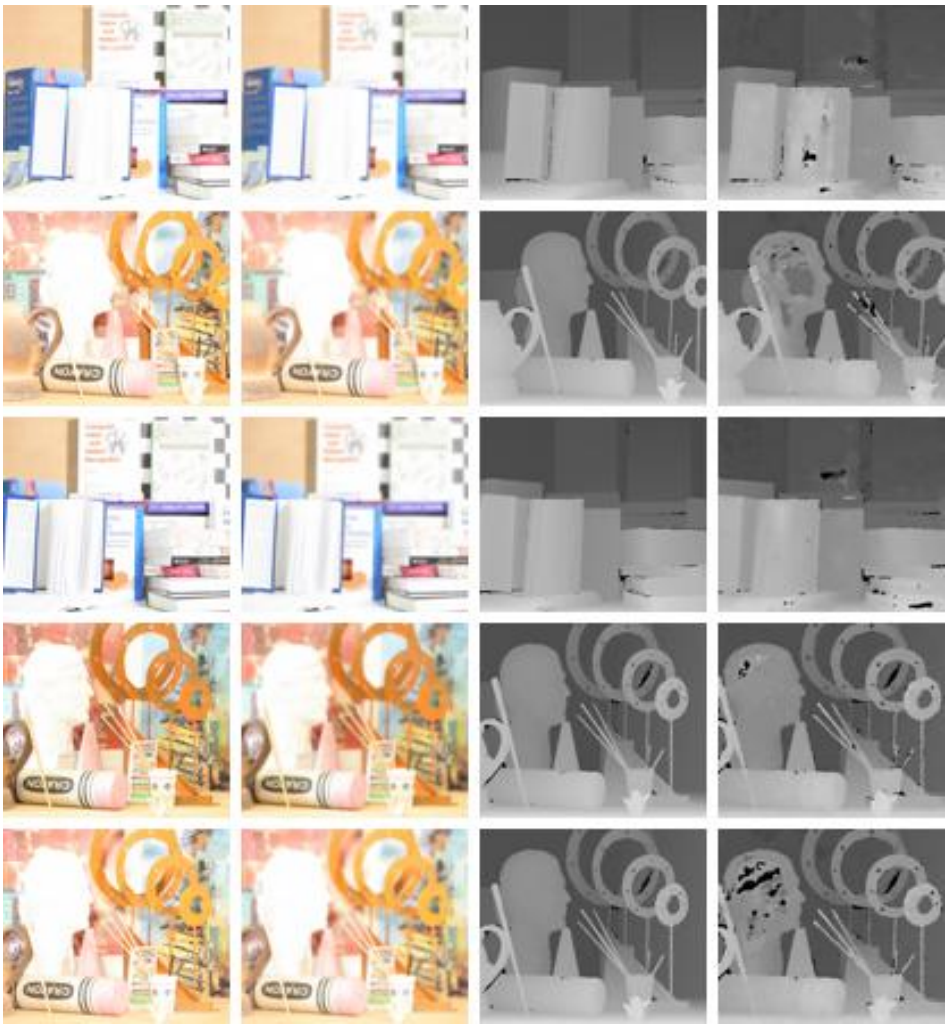
(a) (b) (c) (d)

Fig. C.6. PSO Performance Results for the Middlebury College Dataset - Part 6. (a) In-focus Image, (b) Out-of-focus Image, (c) Ground Truth Depth Map and (d) DfD-Net Computed Depth Map.



(a) (b) (c) (d)

Fig. C.7. PSO Performance Results for the Middlebury College Dataset - Part 7. (a) In-focus Image, (b) Out-of-focus Image, (c) Ground Truth Depth Map and (d) DfD-Net Computed Depth Map.



(a) (b) (c) (d)

Fig. C.8. PSO Performance Results for the Middlebury College Dataset - Part 8. (a) In-focus Image, (b) Out-of-focus Image, (c) Ground Truth Depth Map and (d) DfD-Net Computed Depth Map.

D. DFD-NET MIDDLEBURY COLLEGE DATASET FILTER REDUCTION RESULTS

D.1 Filter Reduction Algorithm Details

The SOM algorithm algorithm uses a distance function to compute the negative distance between the neuron weights and the input vector for each example (in this case the input vectors are the outputs of a given DNN layer). The function used to determine the distance between each of the neurons was the link distance function. Algorithm D.1 defines how the neuron distances are calculated, where 'S' is the initial number of neurons. The competitive neuron with weights that result in the smallest negative distance wins for that particular input. The winning neuron outputs a "1" while the other neurons output a "0". The winning neuron also exerts an influence on the losing neurons within a certain neighborhood to move their weights towards the winning neuron's weights. Ultimately the neurons approach an equilibrium, thereby defining which input vector activates a particular neuron.

Algorithm D.1. Link Distance Calculation Algorithm

$D_{ij} = 0$, if $i == j$

$D_{ij} = 1$, if $\sqrt{\sum (P_i - P_j)^2} \leq 1$

$D_{ij} = 2$, if $k\ exists, D_{ik} = D_{kj} = 1$

$D_{ij} = 3$, if $k_1, k_2\ exists, D_{ik_1} = D_{k_1k_2} = D_{k_2j} = 1$

$D_{ij} = \mathrm{N}$, if $k_1..k_N\ exists, D_{ik_1} = D_{k_1k_2} = D_{k_Nj} = 1$

$D_{ij} = \mathrm{S}$, otherwise

For each layer analyzed, the number of neurons used in the SOM clustering was set to the number of output filters in each layer, given N filters, there could be no more than N clusters. We don't necessarily care which outputs get clustered together, only how many total clusters are determined. In fact each of the sample inputs yields a different number of clusters for a given layer. For this reason the cluster minimum, mean and maximum were used as the basis for the filter reduction. Table D.1 shows the minimum, mean and maximum number of clusters as determined by the SOM algorithm. The layer numbers are color coded to match the layer type as depicted in Figure D.1. Layer 1 cannot be reduced since this layer is the final classification layer.

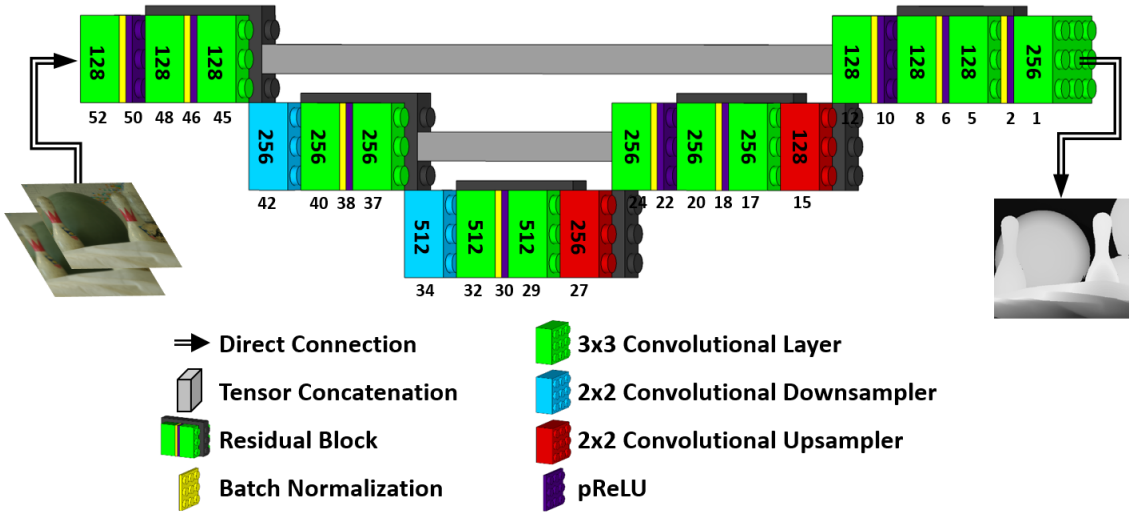


Fig. D.1. Graphical Representation of the DfD-Net Network Architecture

Table D.1.
SOM Clustering Results for the Baseline DfD-Net Architecture

Layer	**52**	**48**	**45**	**42**	**40**	**37**	**34**	**32**	**29**	**27**	**24**	**20**	**17**	**15**	**12**	**8**	**5**	**2**
min	33	39	96	162	152	115	133	270	126	121	119	125	121	91	20	18	65	50
mean	41	46	99	166	163	122	145	275	140	128	122	135	125	95	20	18	71	61
max	46	54	104	171	171	129	152	283	155	133	127	148	129	100	20	19	74	70

The residual layer adds an additional level of complexity when trying to determine the number of convolutional filters to use. This complexity can be illustrated by examining the first residual block (layers 48-45) at the input and the buffering convolutional filter and pReLU (layers 52 and 50). The minimum number of filters as determined by the SOM algorithm shows that layer 52 should have 33 filters, layer 48 only needs to have 39 filters and layer 45 should have 96 filters. To avoid the issue of tensor addition imbalance either 33 or 96 should be selected for layers 52 and 45. So, for every residual block there are two possible candidates. The last four layers also present an additional level of choice. Layers 12-5 form the same structure as previously described. The added pReLU (layer 2) before the final classification layer adds an additional reduction option. For example using the minimum SOM results the last residual block (layer 12, 8 and 5) could take on the configuration of 20-18-20 and 65-18-65. Or if we used the output of the pReLU (layer 2) the configuration would be 50-18-50. This results in three possible choices for the minimum, mean and maximum SOM results.

D.2 Filter Reduction Algorithm Results

Figures D.2 through D.9 show the results of the filter reduction method applied to the DfD-Net trained and tested on the Middlebury College dataset [7,8]. The images are arranged in the order of lowest NRMSE score to highest NRMSE score.

Table D.2.
DfD-Net Reduction Test Configurations

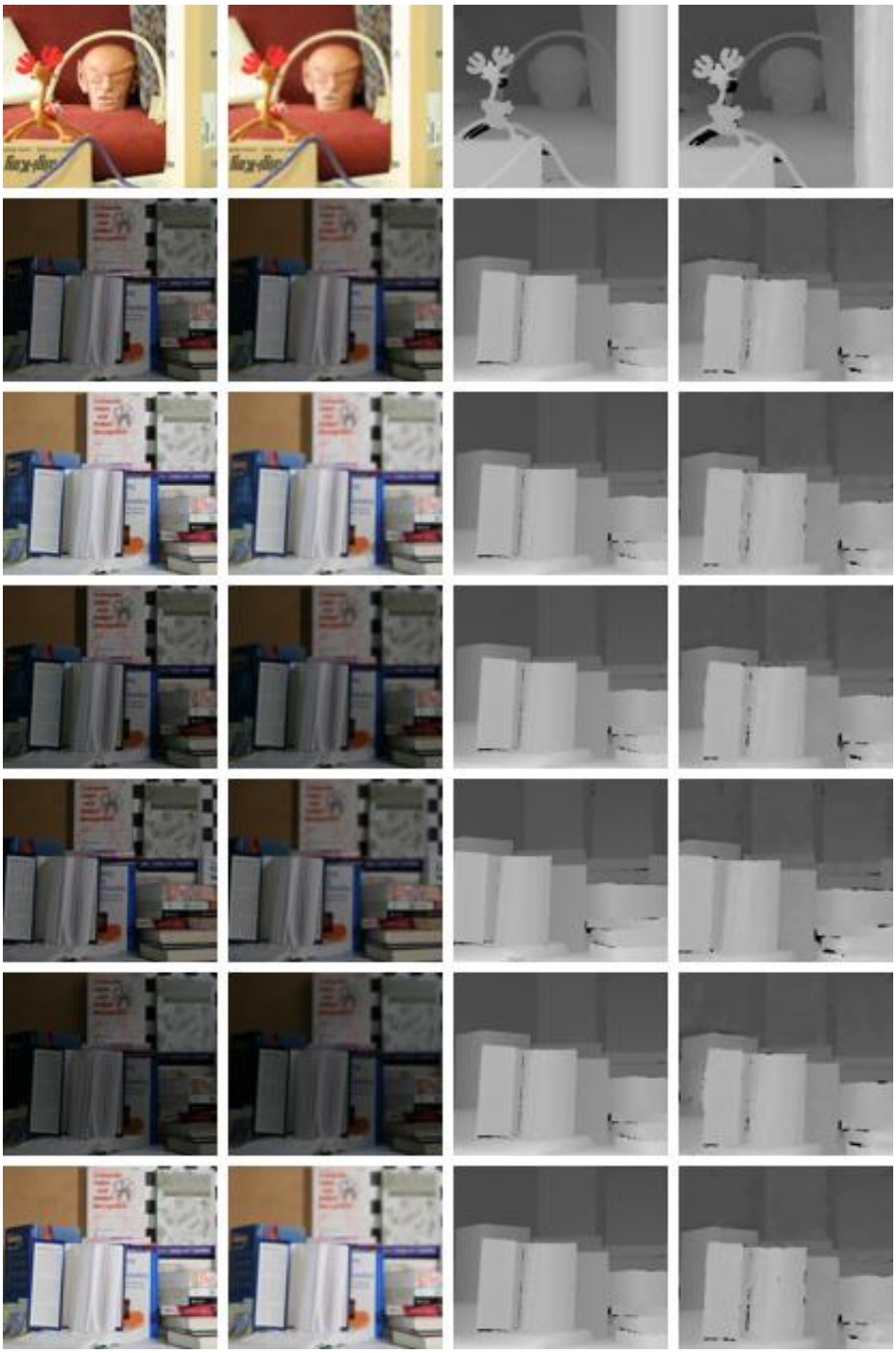
Network Configuration	**Layer Number**																	
	52	**48**	**45**	**42**	**40**	**37**	**34**	**32**	**29**	**27**	**24**	**20**	**17**	**15**	**12**	**8**	**5**	**1**
Original	128	128	128	256	256	256	512	512	512	256	256	256	256	128	128	128	128	256
B-01-01	33	39	33	115	152	115	126	270	126	121	119	125	119	91	20	18	20	256
B-01-02	96	39	96	162	152	162	133	270	133	121	121	125	121	91	65	18	65	256
B-01-03	96	39	96	162	152	162	133	270	133	121	121	125	121	91	50	18	50	256
B-02-01	41	46	41	122	163	122	140	275	140	128	122	135	122	95	20	18	20	256
B-02-02	99	46	99	166	163	166	145	275	145	128	125	135	125	95	71	18	71	256
B-02-03	99	46	99	166	163	166	145	275	145	128	125	135	125	95	61	18	61	256
B-03-01	46	54	46	129	171	129	152	283	152	133	127	148	127	100	20	19	20	256
B-03-02	104	54	104	171	171	171	155	283	155	133	129	148	129	100	74	19	74	256
B-03-03	104	54	104	171	171	171	155	283	155	133	129	148	129	100	70	19	70	256
F-01-01	96	44	96	104	150	104	145	206	145	129	112	109	112	93	71	18	71	256
F-01-02	96	44	96	104	150	104	185	206	185	129	112	109	112	93	71	18	71	256
F-02-01	96	44	96	104	150	104	160	253	160	145	112	109	112	93	71	18	71	256
F-02-02	**96**	**44**	**96**	**104**	**150**	**104**	**190**	**253**	**190**	**145**	**112**	**109**	**112**	**93**	**71**	**18**	**71**	**256**
F-03-01	96	44	96	104	150	104	155	226	155	139	112	109	112	93	71	18	71	256
F-03-02	96	44	96	104	150	104	188	226	188	139	112	109	112	93	71	18	71	256
F-02-02L	96	40	96	104	144	104	192	256	192	144	112	112	112	96	72	16	72	256
F-02-02H	96	48	96	104	152	104	192	256	192	152	112	112	112	96	72	24	72	256

Table D.3.
DfD-Net Configuration Average Runtime Results

Network Configuration	**Average Runtime/Image (s)**			
	Laptop	**Jetson TX2**	**Desktop 1**	**Desktop 2**
Original	1.75069	3.89837	0.52668	0.41274
B-01-01	**0.74395**	**1.85388**	0.41012	**0.27135**
B-01-02	0.99336	2.46213	0.44992	0.31245
B-01-03	0.96670	2.32889	0.43593	0.30252
B-02-01	0.79274	1.95407	0.41960	0.27731
B-02-02	1.04594	2.57182	0.45406	0.31956
B-02-03	1.01773	2.44731	0.44365	0.31157
B-03-01	0.84491	2.04646	0.42211	0.28223
B-03-02	1.13332	2.70476	0.45760	0.32747
B-03-03	1.13474	2.75342	0.45197	0.33035
F-01-01	0.94893	2.32520	0.40292	0.30775
F-01-02	0.95420	2.32717	**0.40229**	0.29967
F-02-01	0.95743	2.33158	0.41362	0.30315
F-02-02	0.96554	2.33764	0.44816	0.30252
F-03-01	0.95621	2.32580	0.40632	0.30231
F-03-02	0.95424	2.33922	0.41007	0.30233
F-02-02L	0.96019	2.34308	0.41034	0.30163
F-02-02H	0.97198	2.35826	0.42731	0.30432

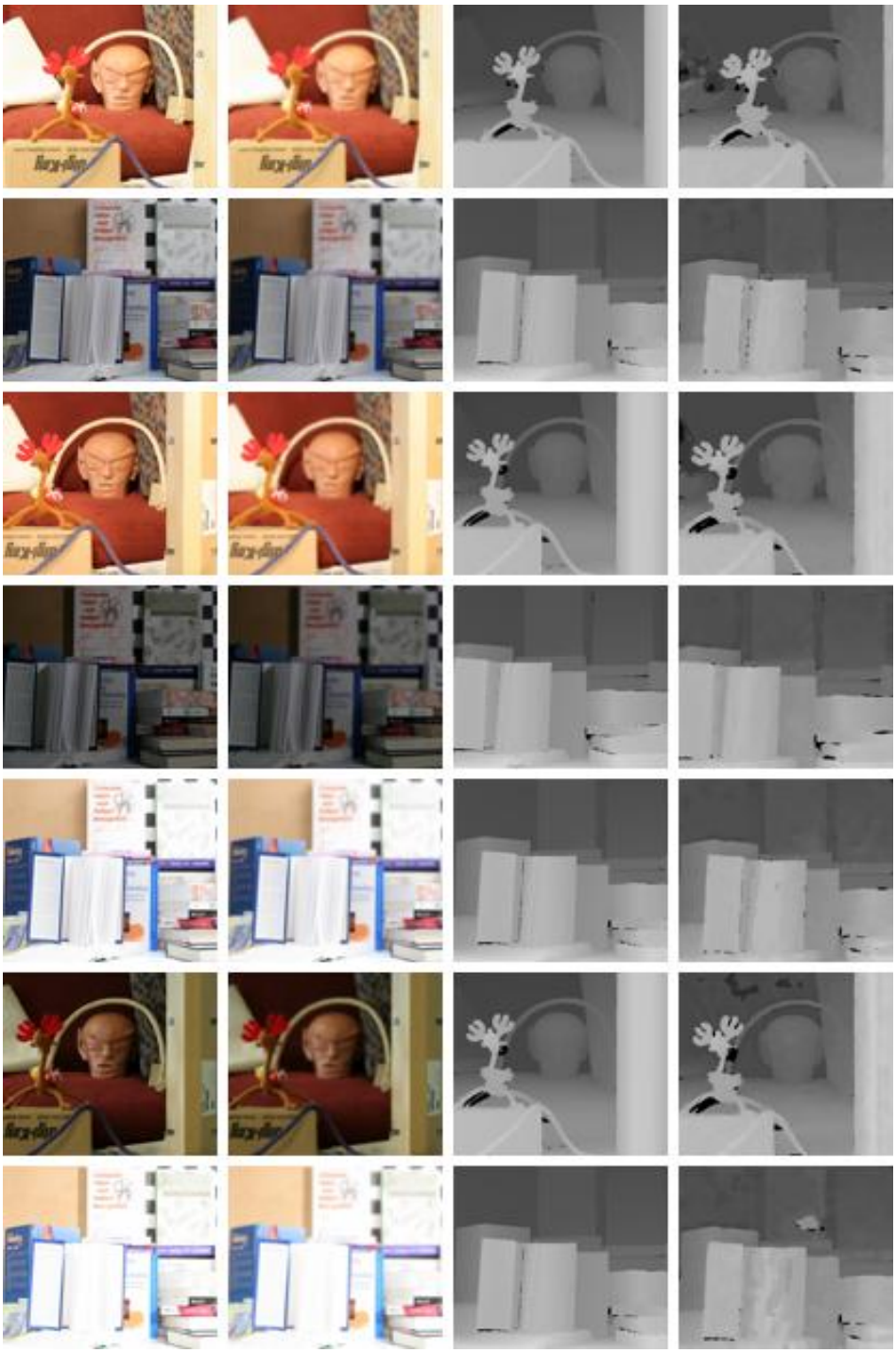
(a) (b) (c) (d)

Fig. D.2. Filter Reduction Performance Results for the Middlebury College Dataset - Part 1. (a) In-focus Image, (b) Out-of-focus Image, (c) Ground Truth Depth Map and (d) DfD-Net Computed Depth Map.



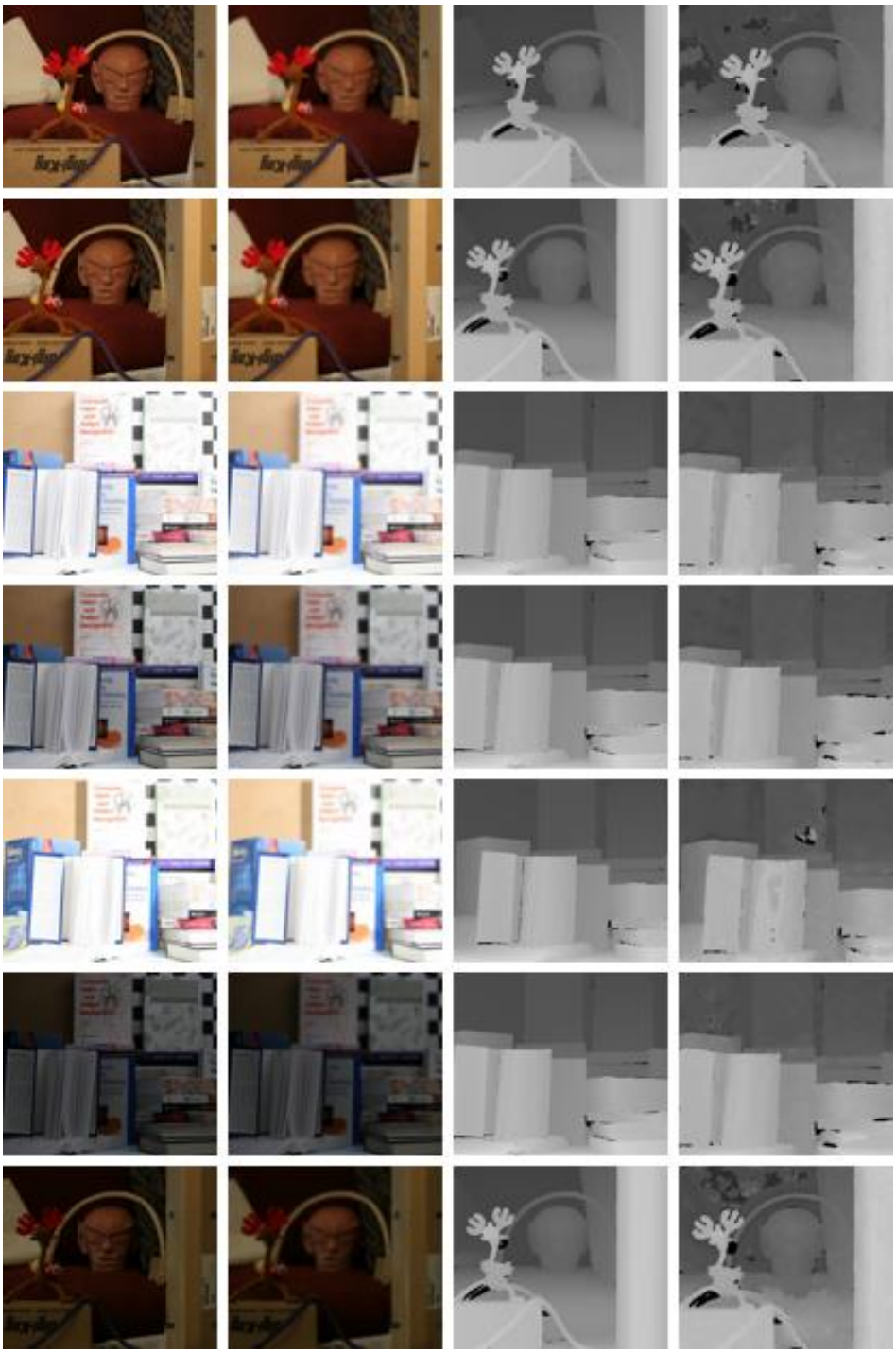
(a) (b) (c) (d)

Fig. D.3. Filter Reduction Performance Results for the Middlebury College Dataset - Part 2. (a) In-focus Image, (b) Out-of-focus Image, (c) Ground Truth Depth Map and (d) DfD-Net Computed Depth Map.



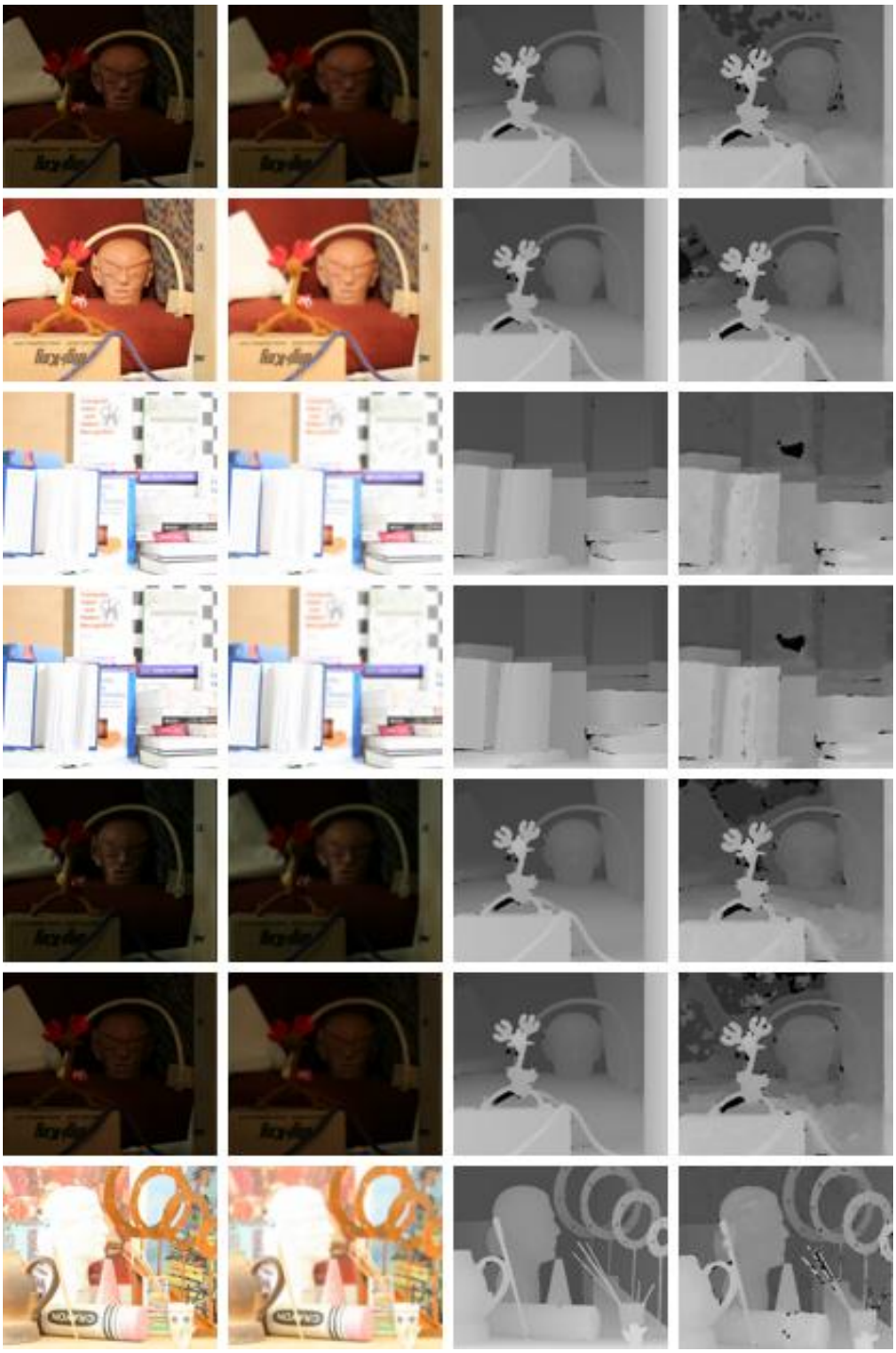
(a) (b) (c) (d)

Fig. D.4. Filter Reduction Performance Results for the Middlebury College Dataset - Part 3. (a) In-focus Image, (b) Out-of-focus Image, (c) Ground Truth Depth Map and (d) DfD-Net Computed Depth Map.



(a) (b) (c) (d)

Fig. D.5. Filter Reduction Performance Results for the Middlebury College Dataset - Part 4. (a) In-focus Image, (b) Out-of-focus Image, (c) Ground Truth Depth Map and (d) DfD-Net Computed Depth Map.



(a) (b) (c) (d)

Fig. D.6. Filter Reduction Performance Results for the Middlebury College Dataset - Part 5. (a) In-focus Image, (b) Out-of-focus Image, (c) Ground Truth Depth Map and (d) DfD-Net Computed Depth Map.

(a) (b) (c) (d)

Fig. D.7. Filter Reduction Performance Results for the Middlebury College Dataset - Part 6. (a) In-focus Image, (b) Out-of-focus Image, (c) Ground Truth Depth Map and (d) DfD-Net Computed Depth Map.

(a) (b) (c) (d)

Fig. D.8. Filter Reduction Performance Results for the Middlebury College Dataset - Part 7. (a) In-focus Image, (b) Out-of-focus Image, (c) Ground Truth Depth Map and (d) DfD-Net Computed Depth Map.

(a) (b) (c) (d)

Fig. D.9. Filter Reduction Performance Results for the Middlebury College Dataset - Part 8. (a) In-focus Image, (b) Out-of-focus Image, (c) Ground Truth Depth Map and (d) DfD-Net Computed Depth Map.

E. DFD-NET REAL WORLD DATASET SYNTHETICALLY BLURRED RESULTS

Figures E.2 through E.35 show the results of the DfD-Net trained on the Middlebury College dataset [7, 8] and tested on the subset of the synthetically blurred real world dataset that are derivatives of the k01, k02 and k03 scenes. The images are arranged in the order of lowest NRMSE score to highest NRMSE score. Figure E.1 shows the order of arrangement for the images for each page.

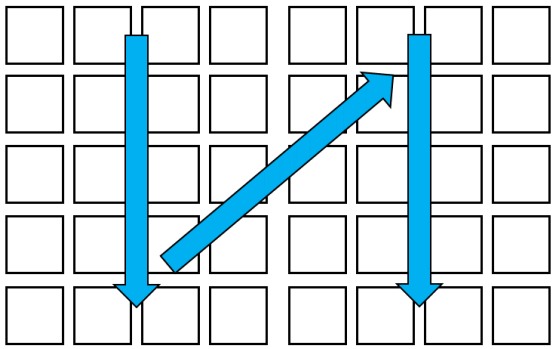


Fig. E.1. Example Image Order

(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.2. K01-K03 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 1. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.

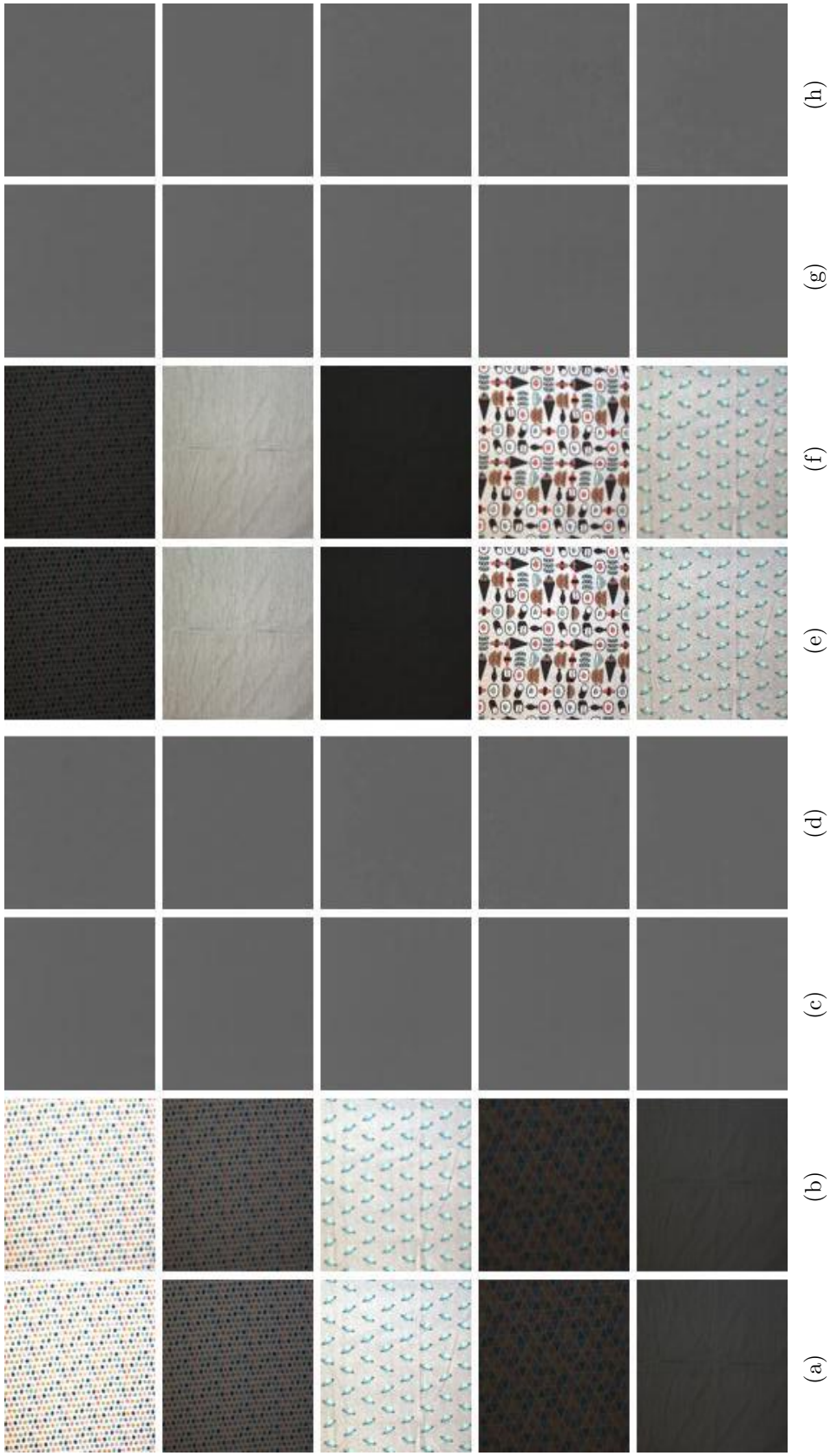
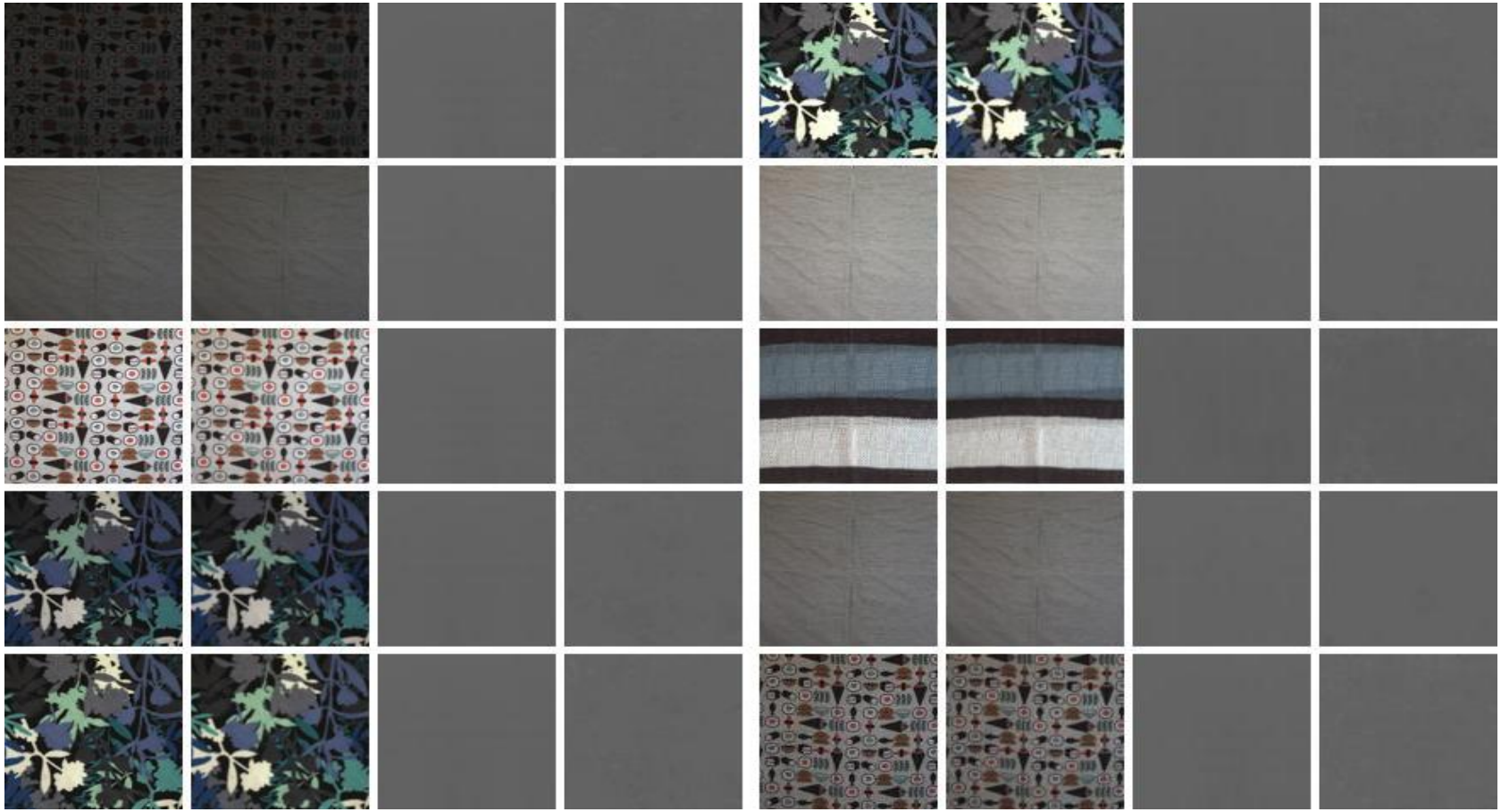
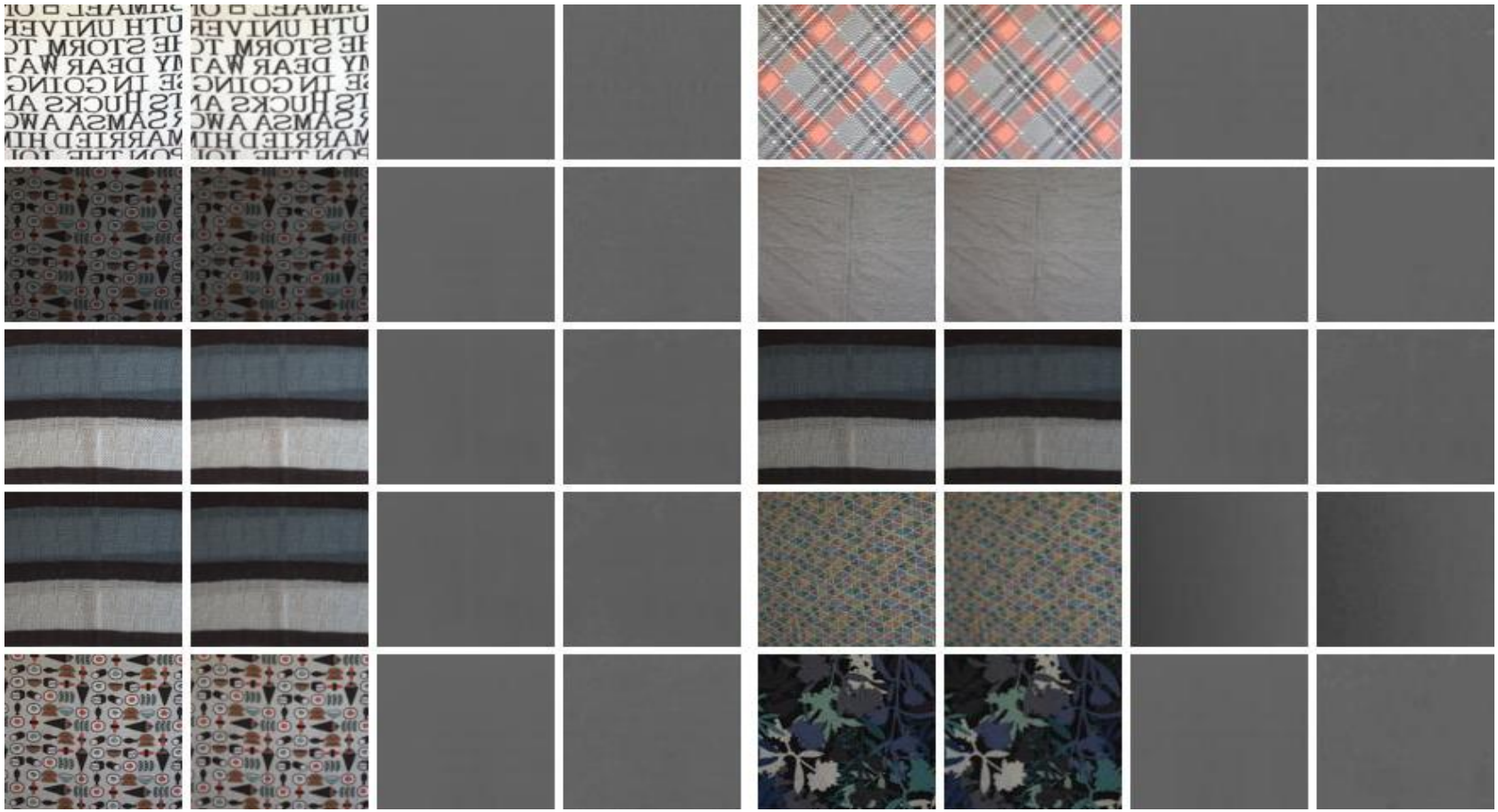


Fig. E.3. K01-K03 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 2. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.



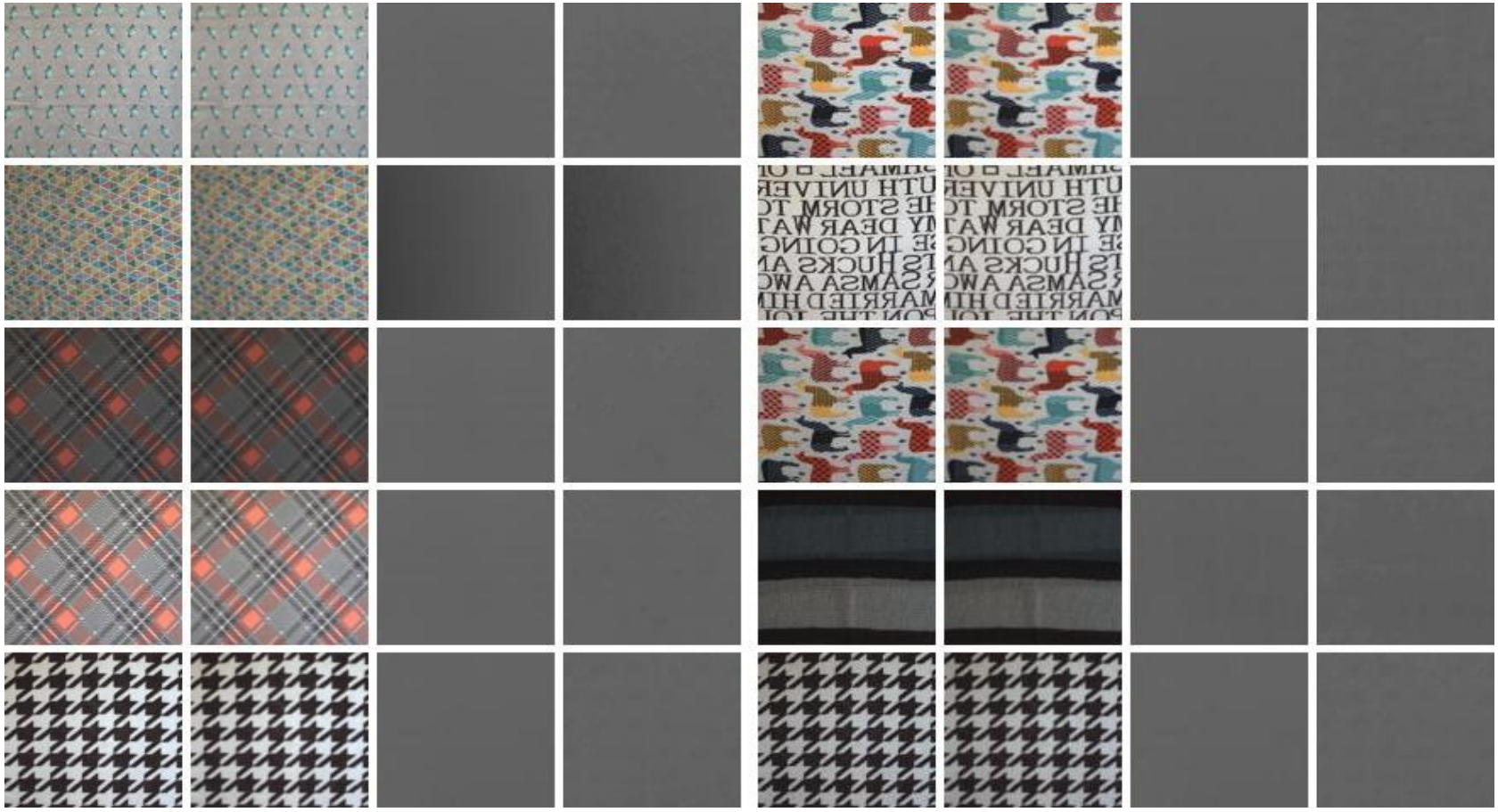
(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.4. K01-K03 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 3. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.



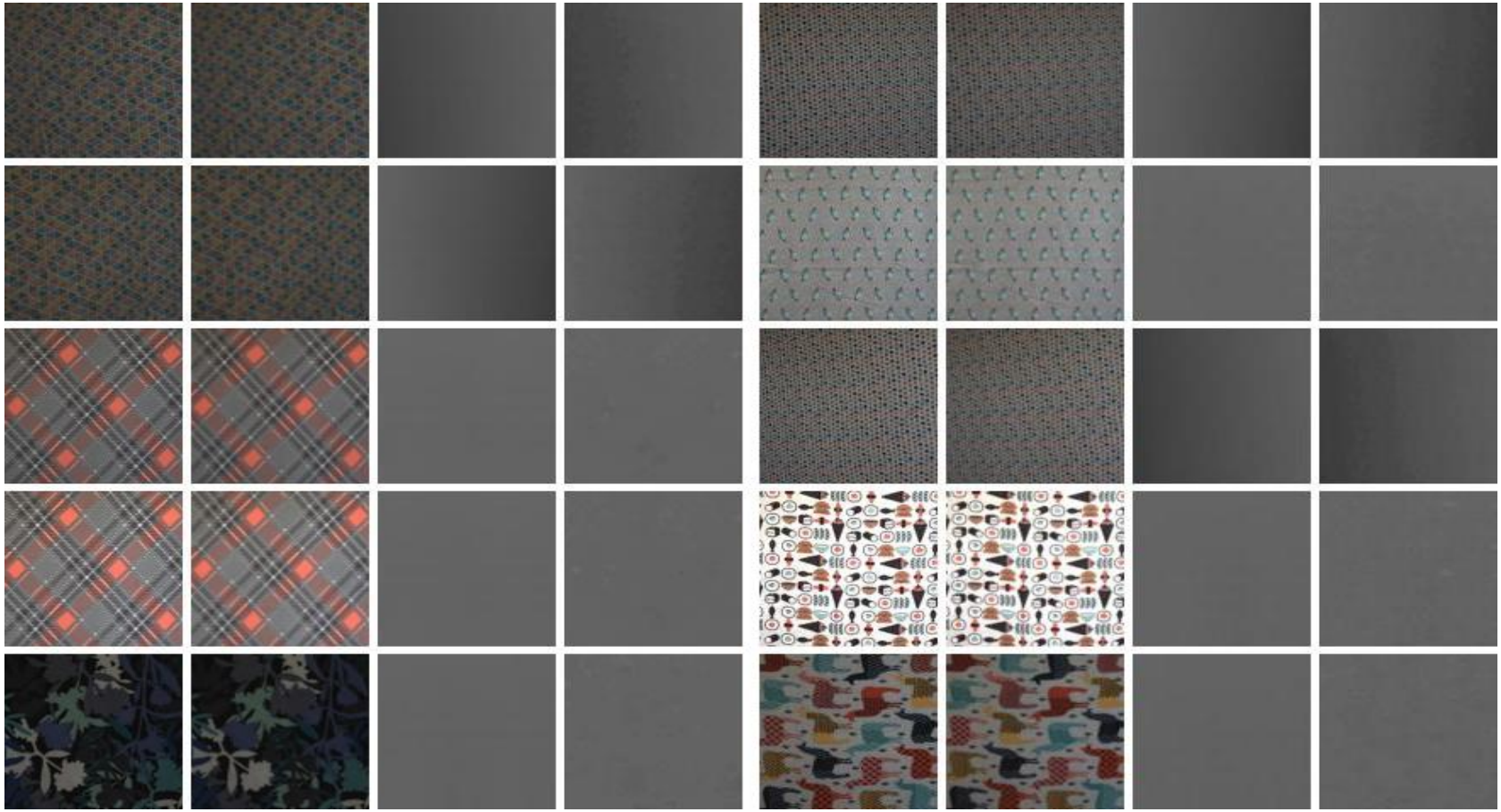
(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.5. K01-K03 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 4. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.



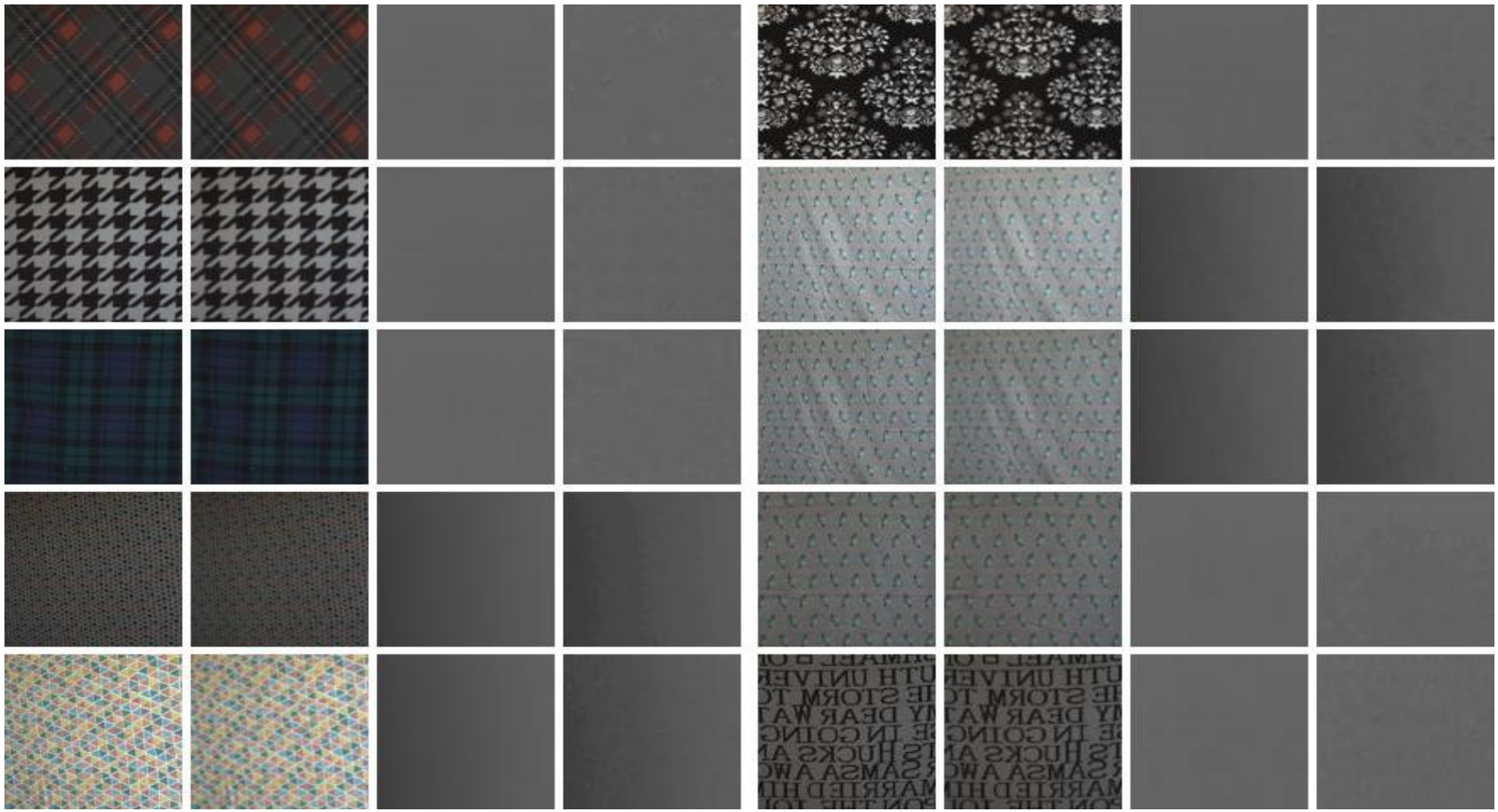
(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.6. K01-K03 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 5. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.



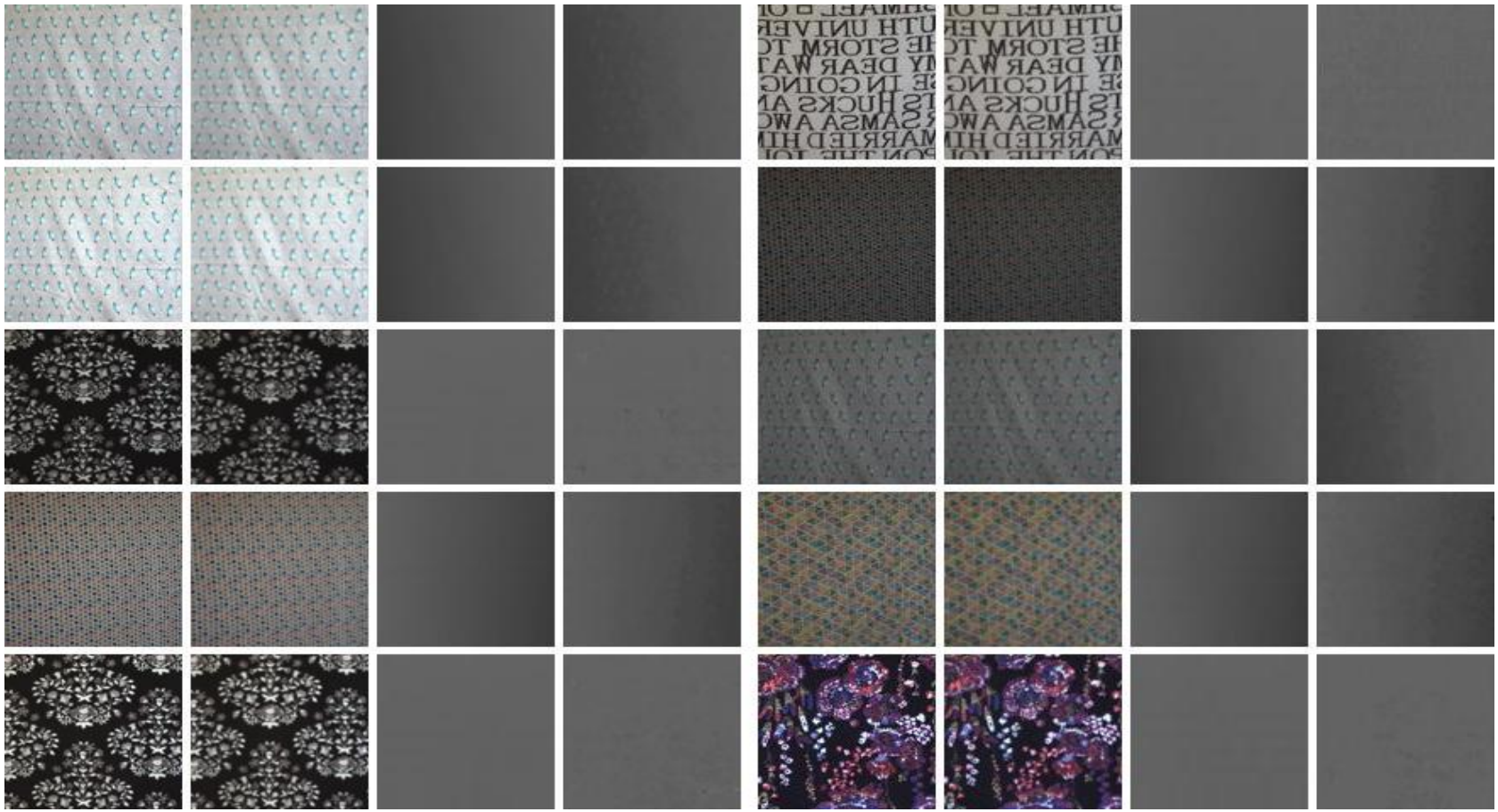
(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.7. K01-K03 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 6. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.



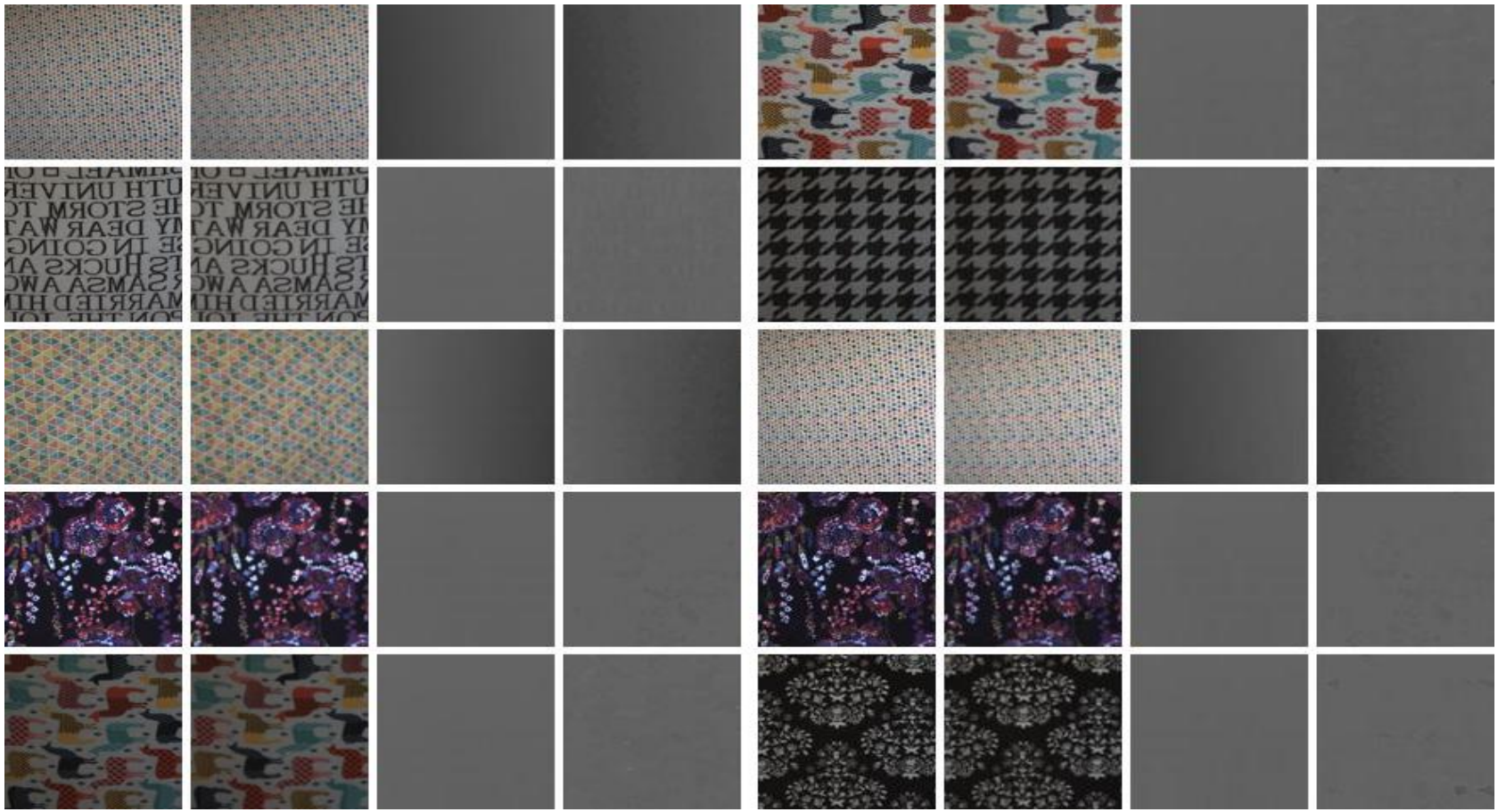
(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.8. K01-K03 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 7. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.



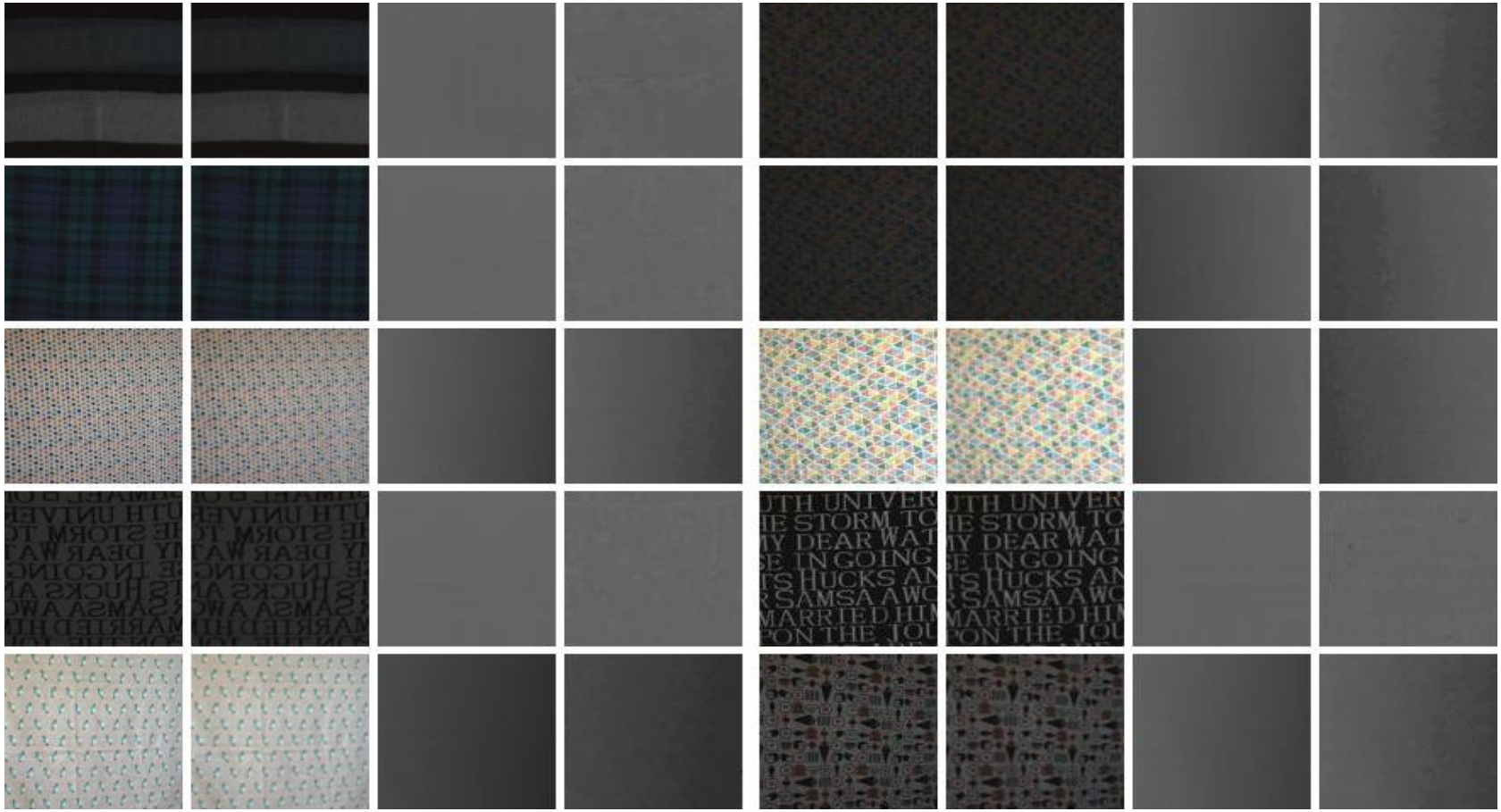
(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.9. K01-K03 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 8. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.



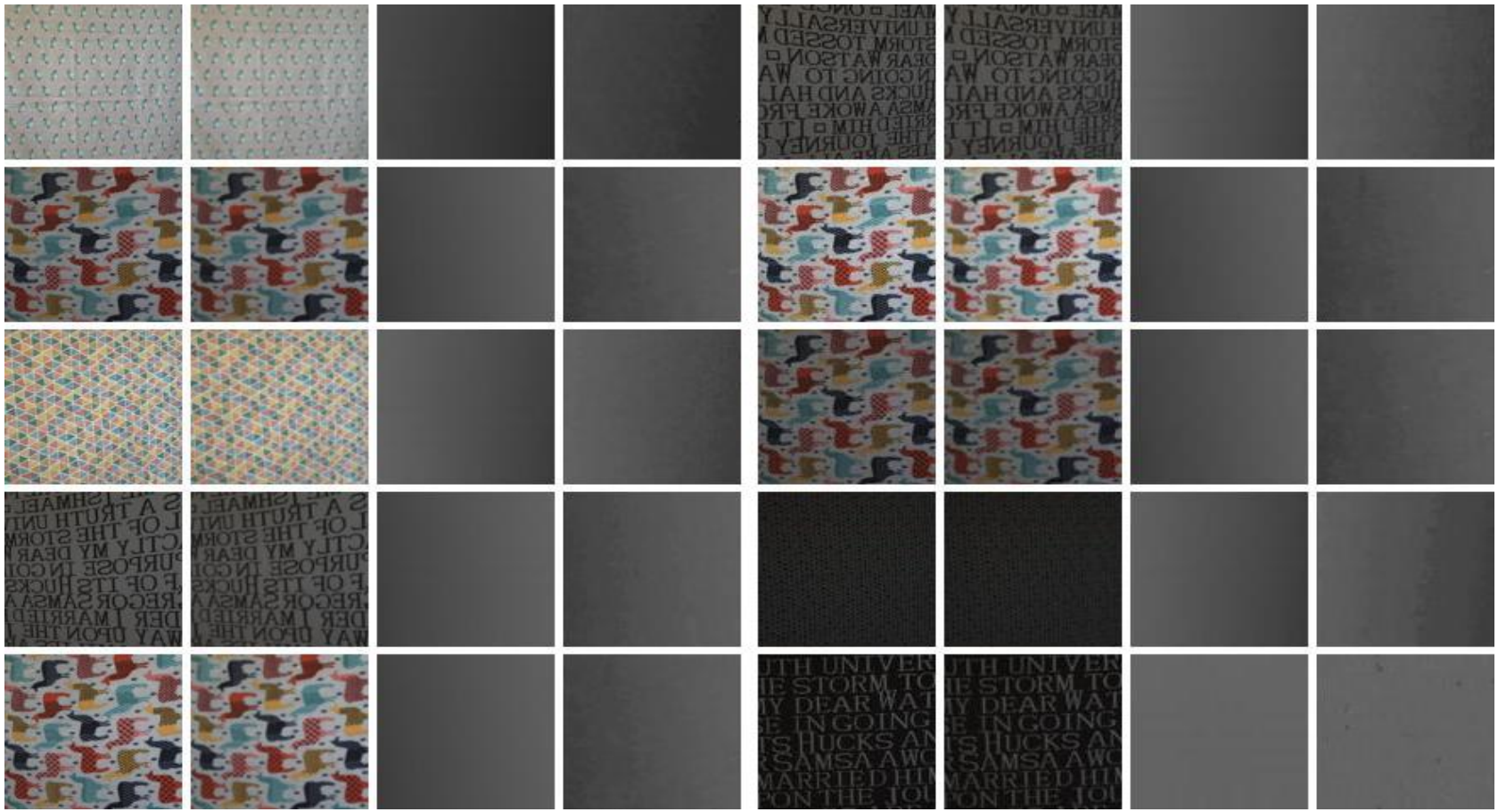
(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.10. K01-K03 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 9. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.



(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.11. K01-K03 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 10. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.

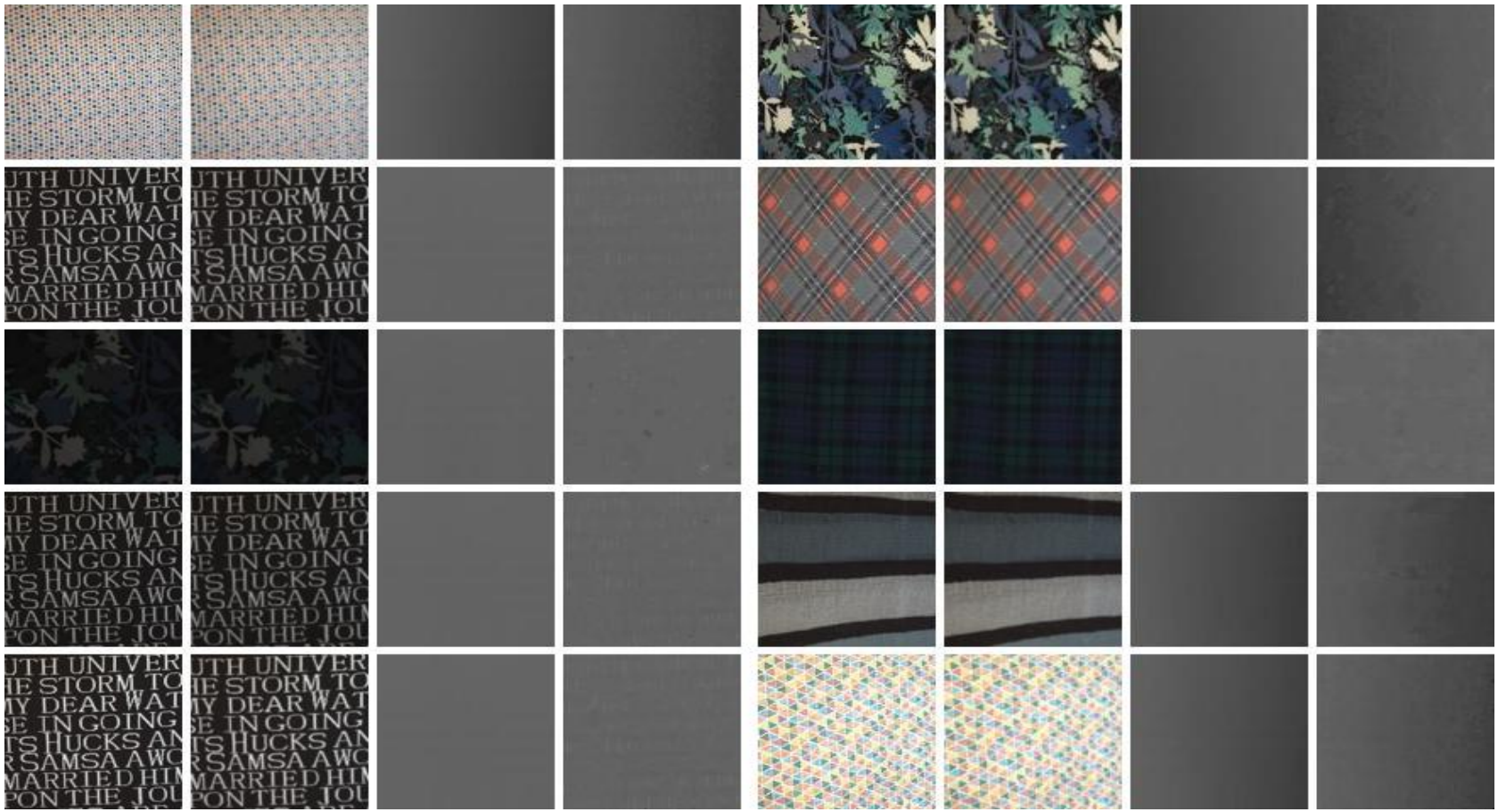


(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.12. K01-K03 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 11. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.

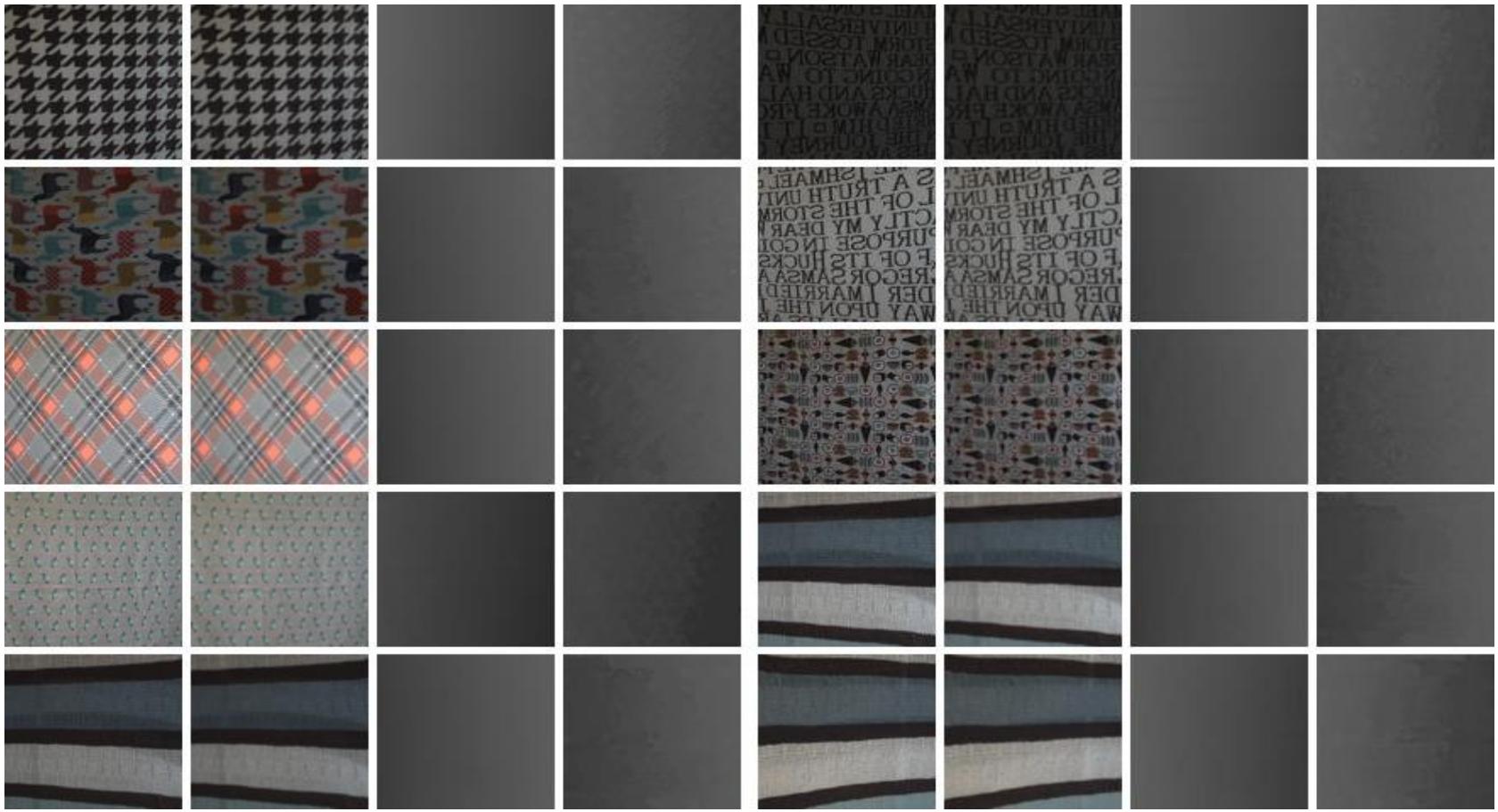
(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.13. K01-K03 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 12. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.



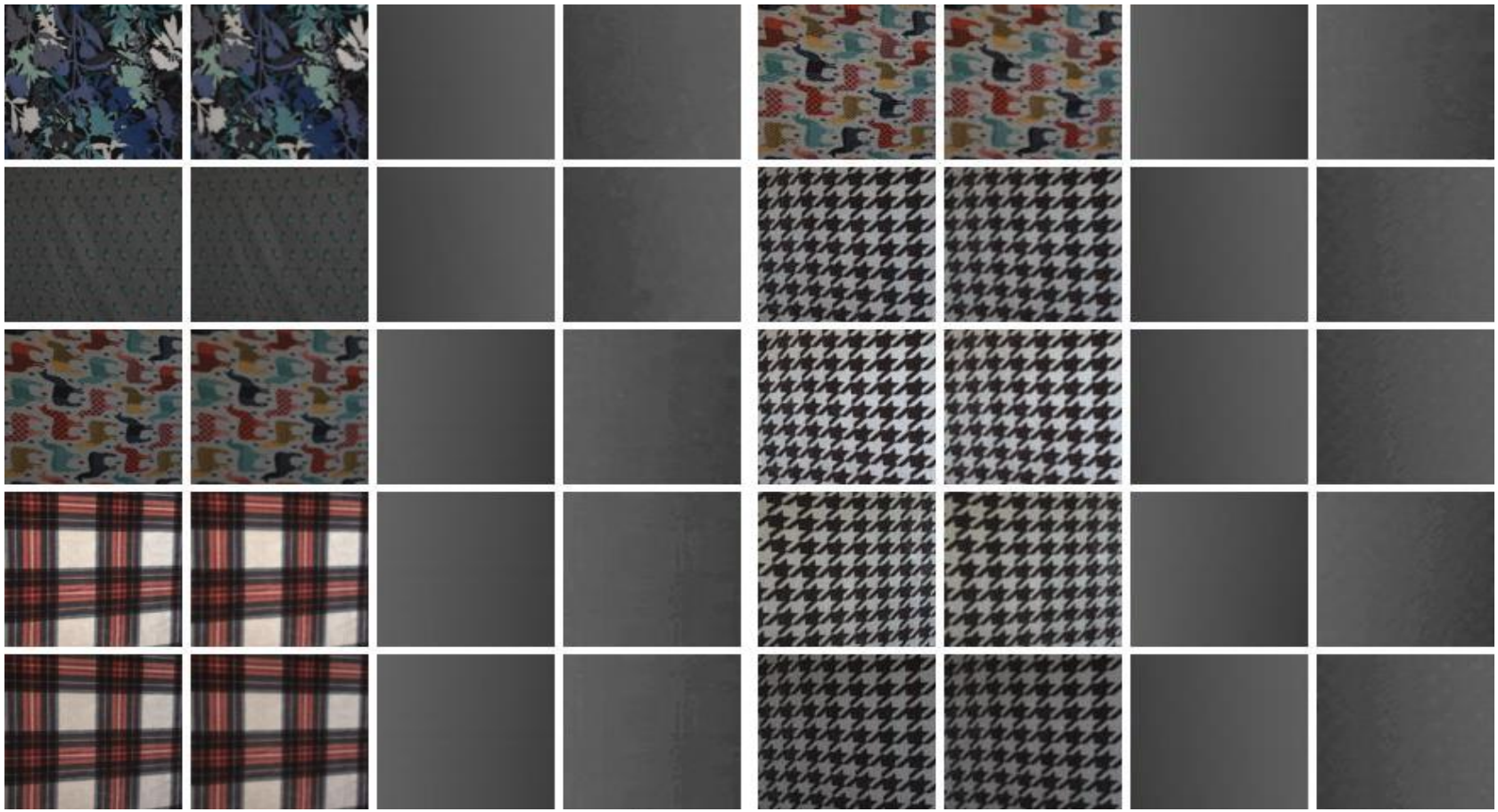
(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.14. K01-K03 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 13. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.



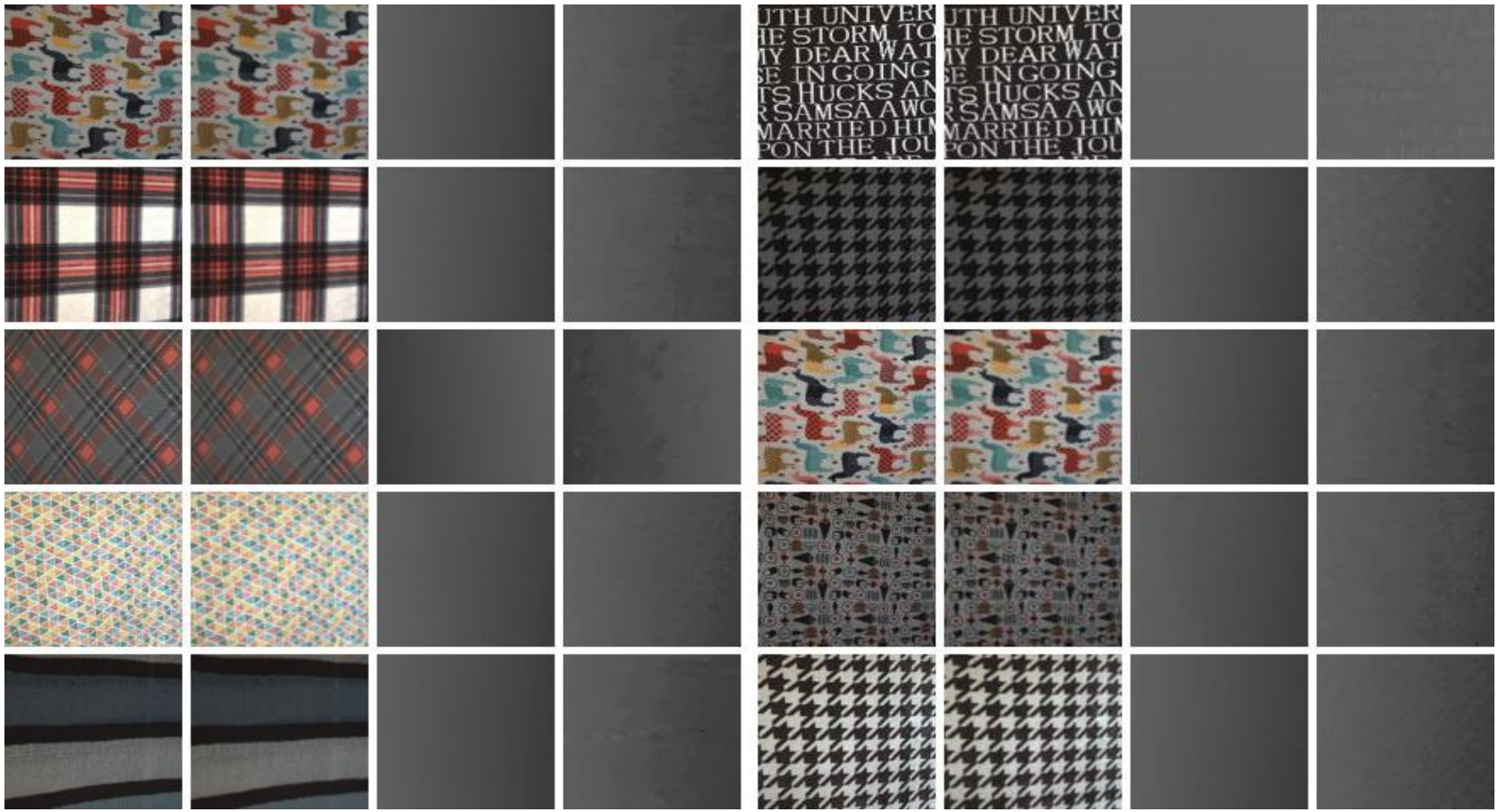
(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.15. K01-K03 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 14. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.



(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.16. K01-K03 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 15. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.

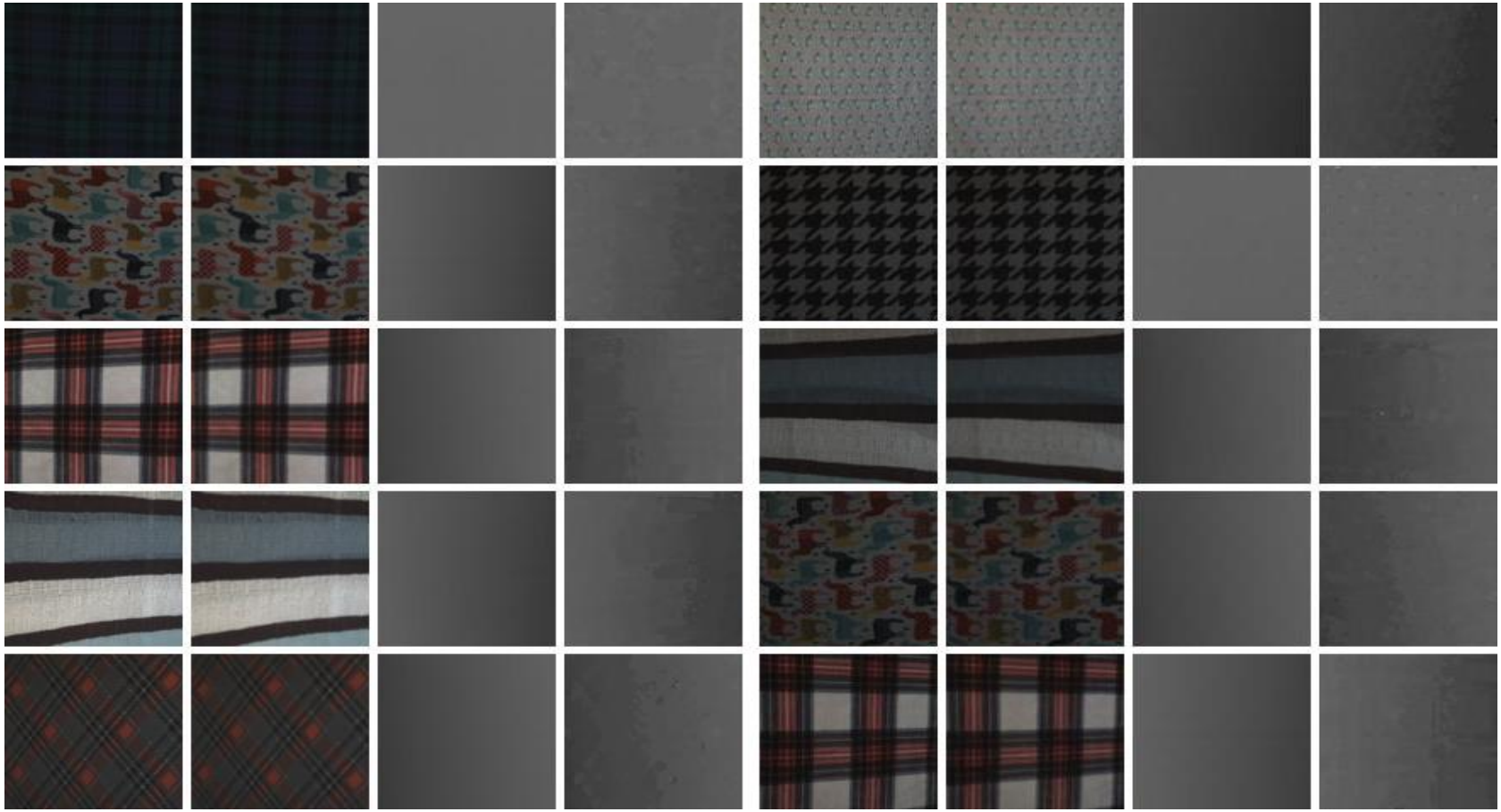


(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.17. K01-K03 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 16. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.

(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.18. K01-K03 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 17. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.



(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.19. K01-K03 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 18. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.

(a) (b) (c) (d) (e) (f) (g) (h)

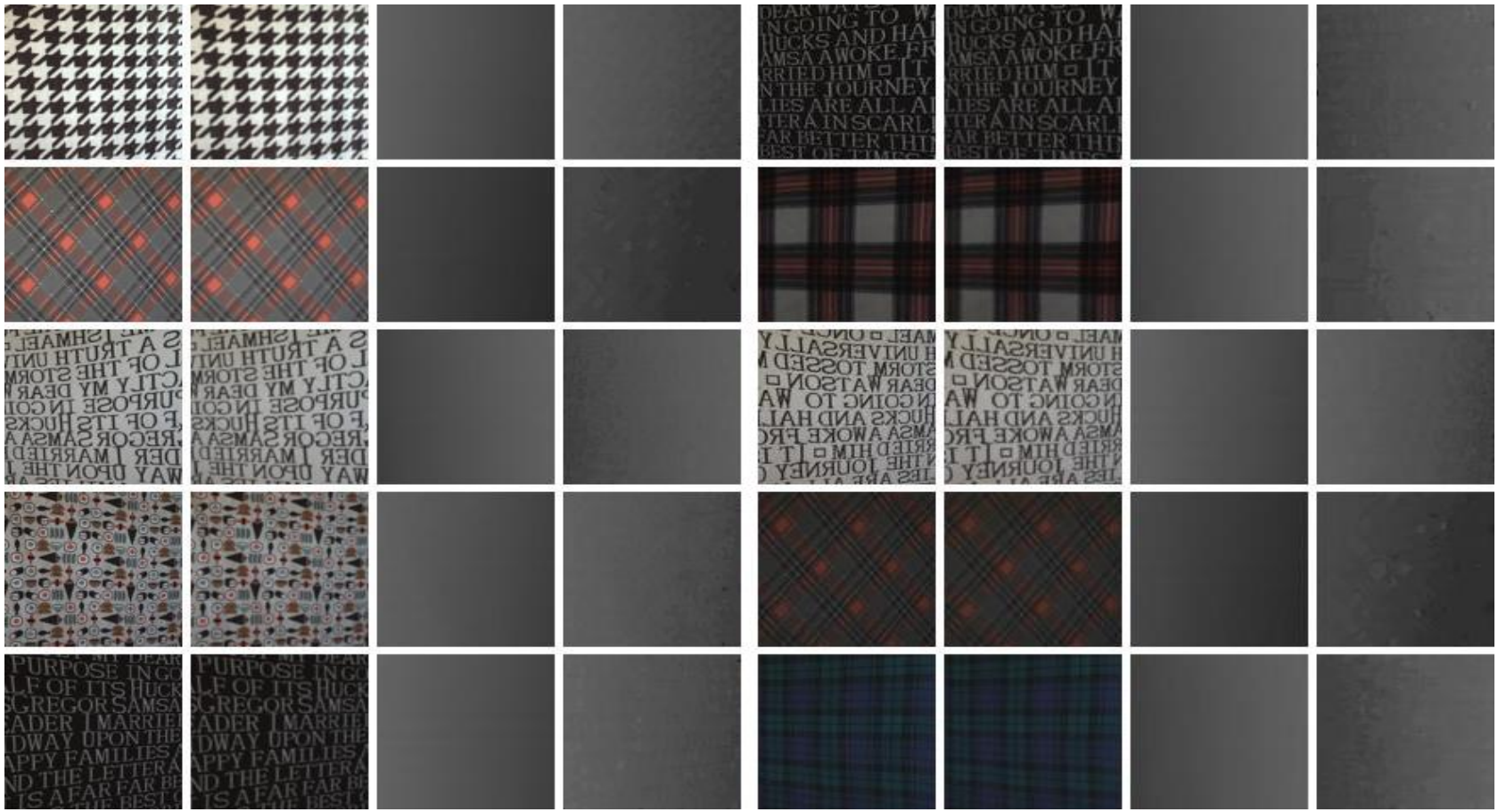
Fig. E.20. K01-K03 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 19. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.

(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.21. K01-K03 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 20. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.

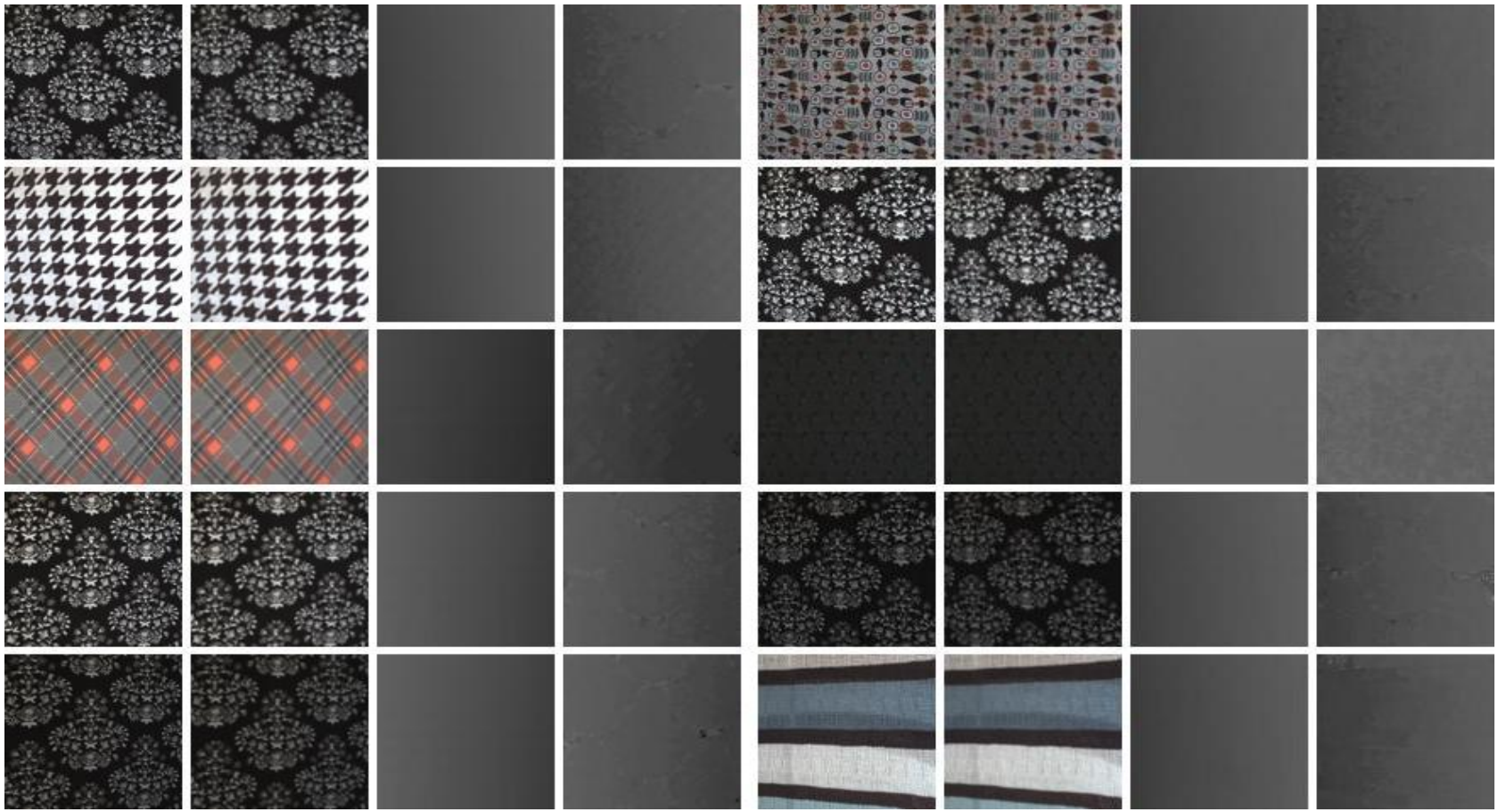
(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.22. K01-K03 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 21. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.



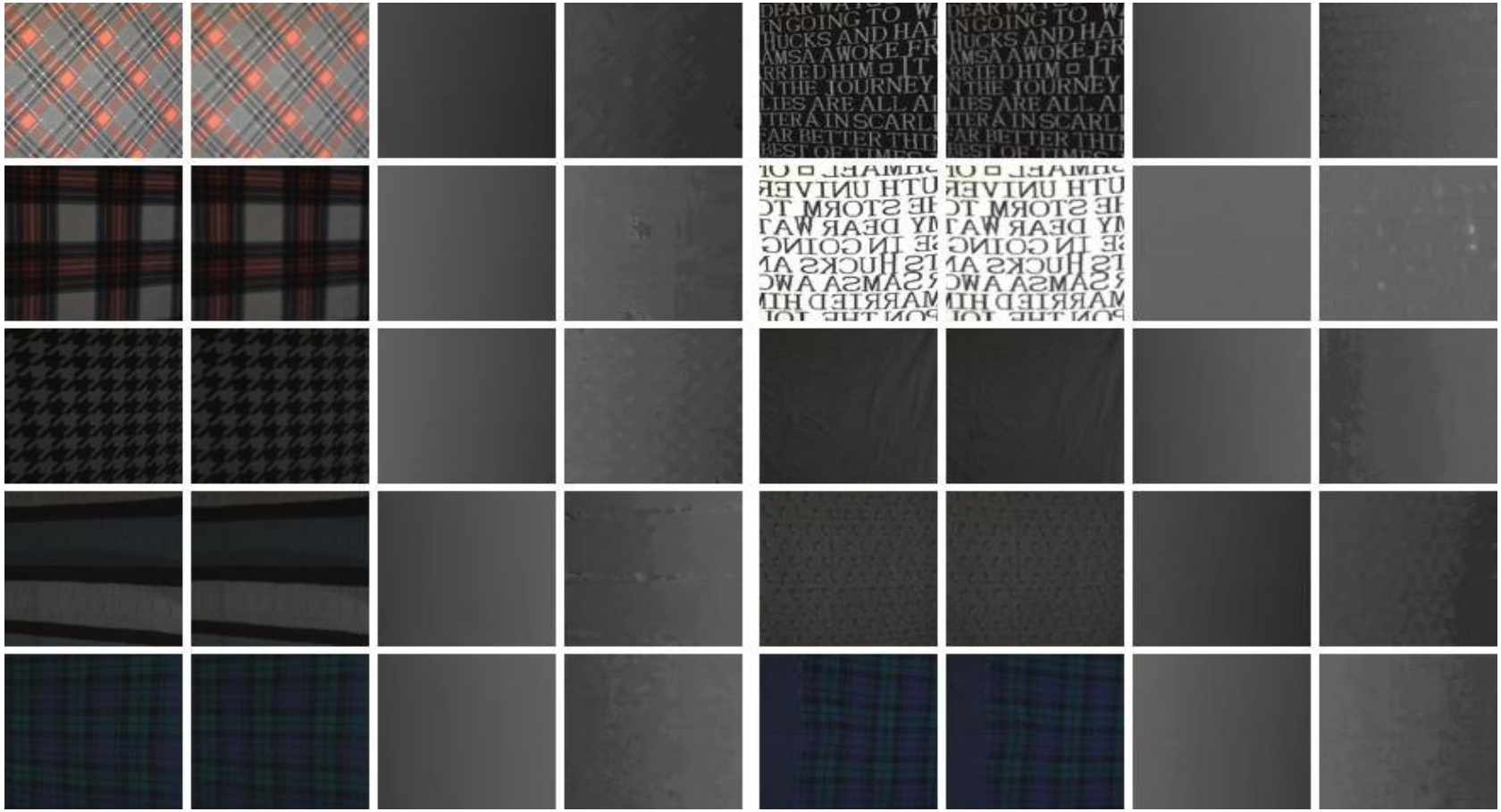
(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.23. K01-K03 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 22. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.



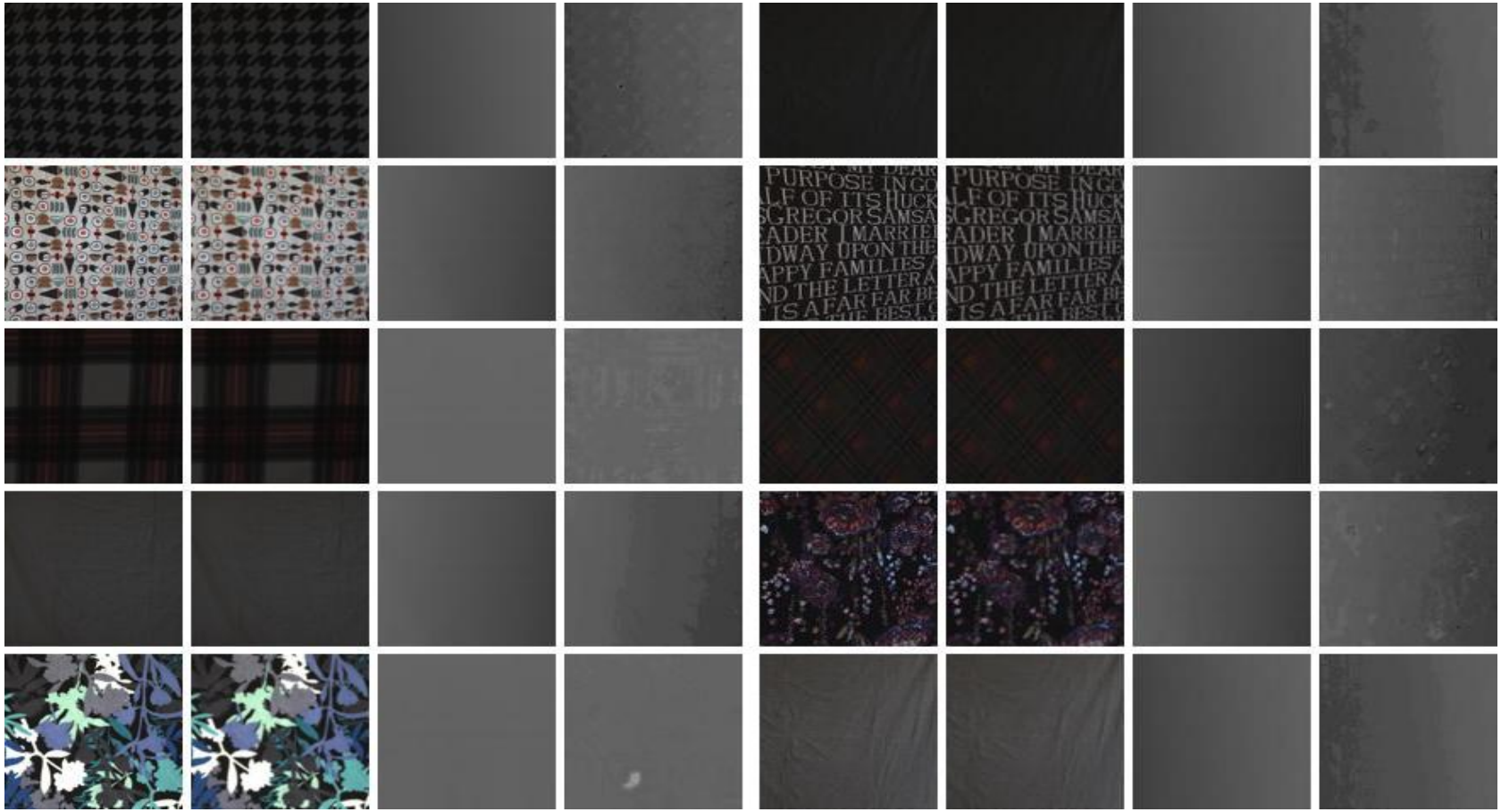
(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.24. K01-K03 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 23. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.



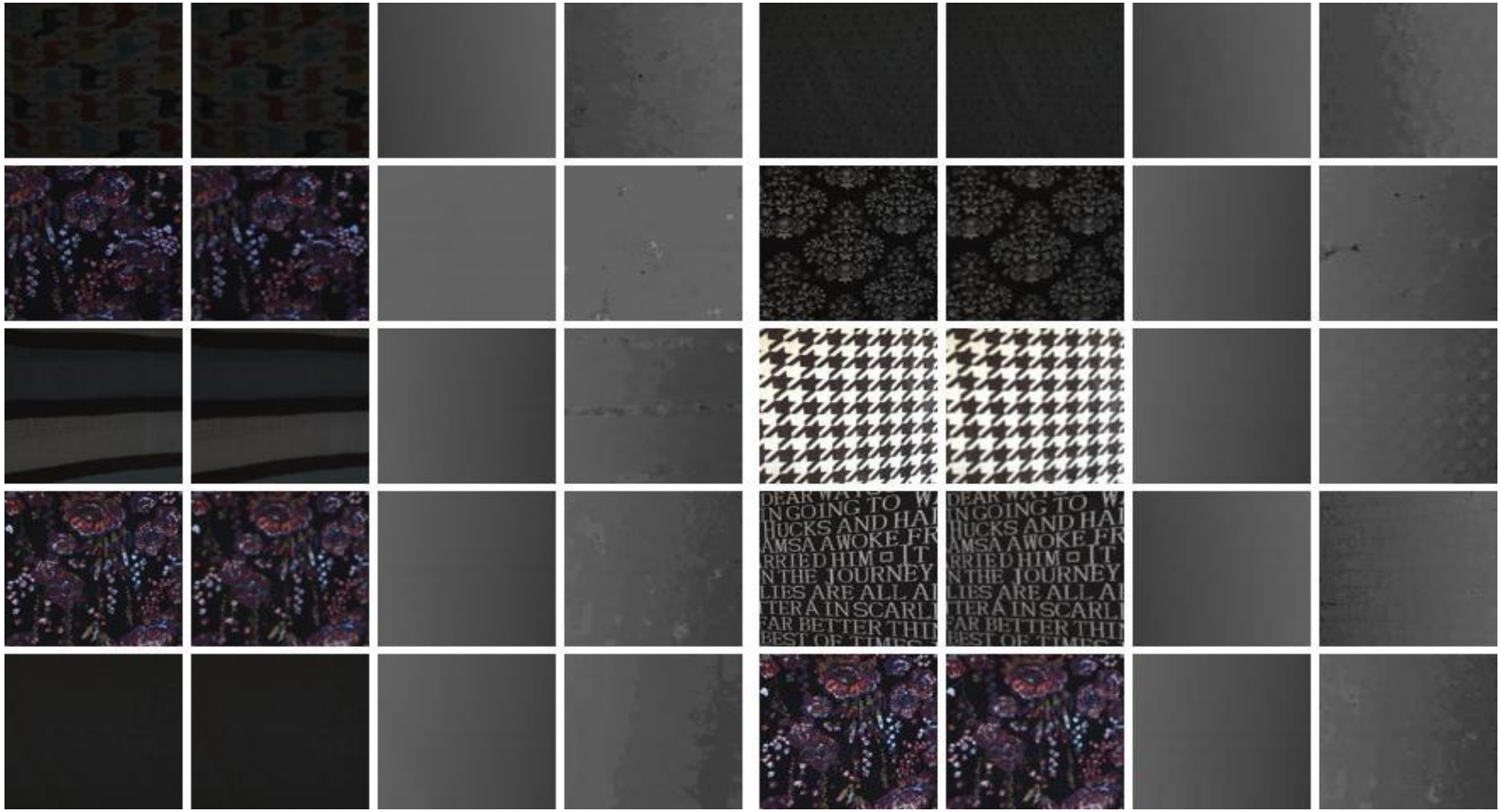
(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.25. K01-K03 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 24. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.



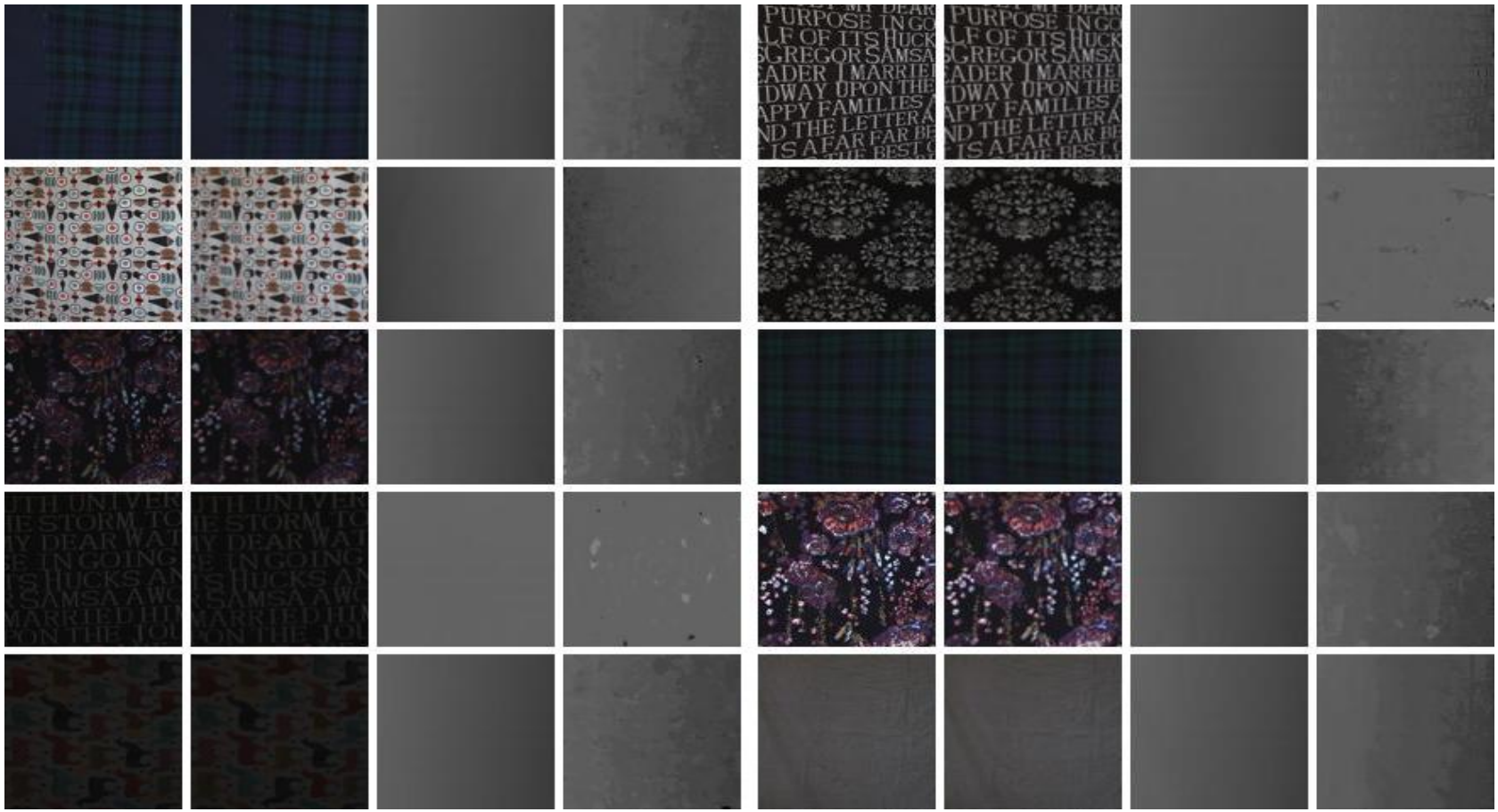
(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.26. K01-K03 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 25. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.



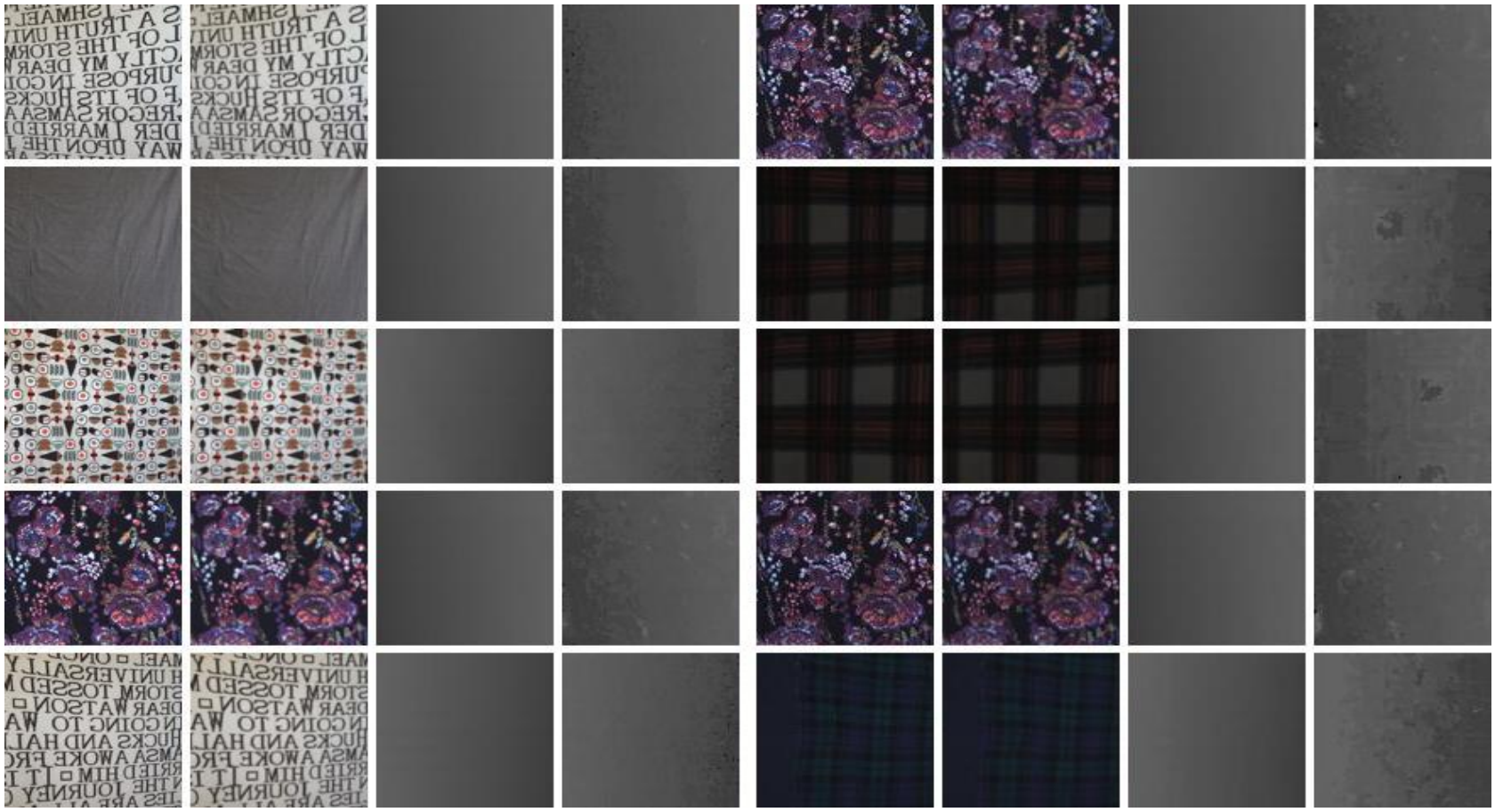
(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.27. K01-K03 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 26. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.



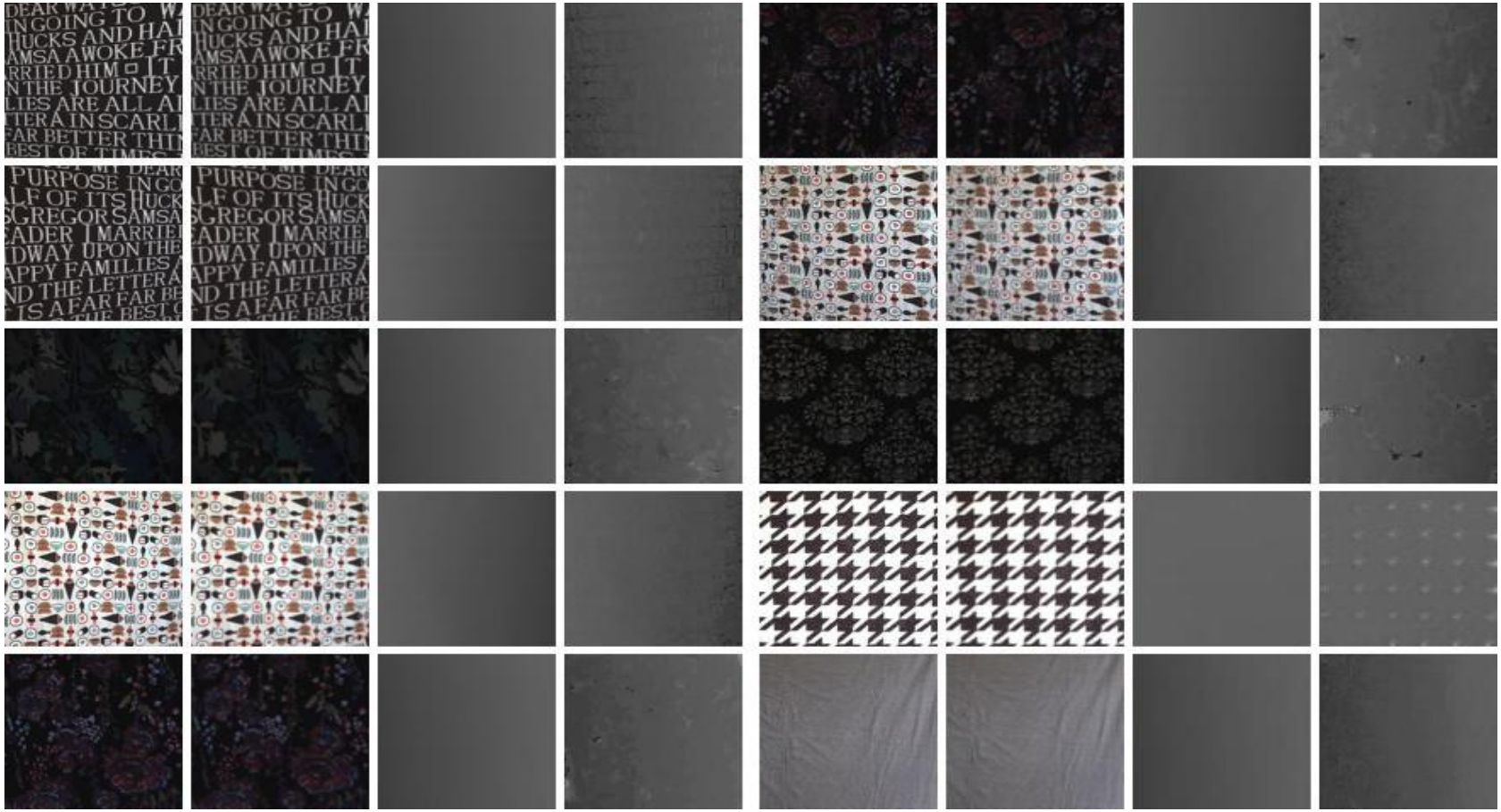
(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.28. K01-K03 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 27. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.



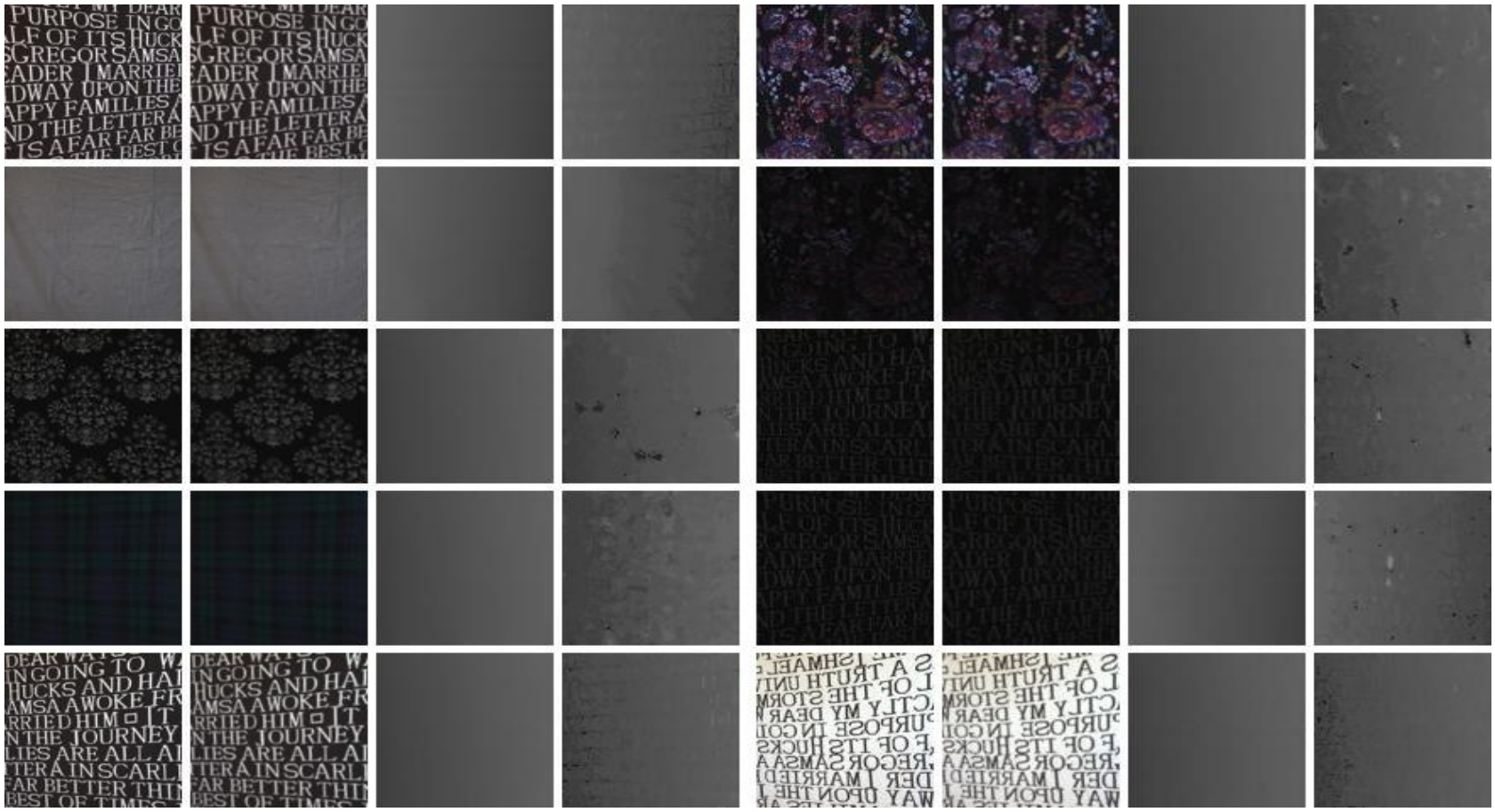
(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.29. K01-K03 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 28. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.



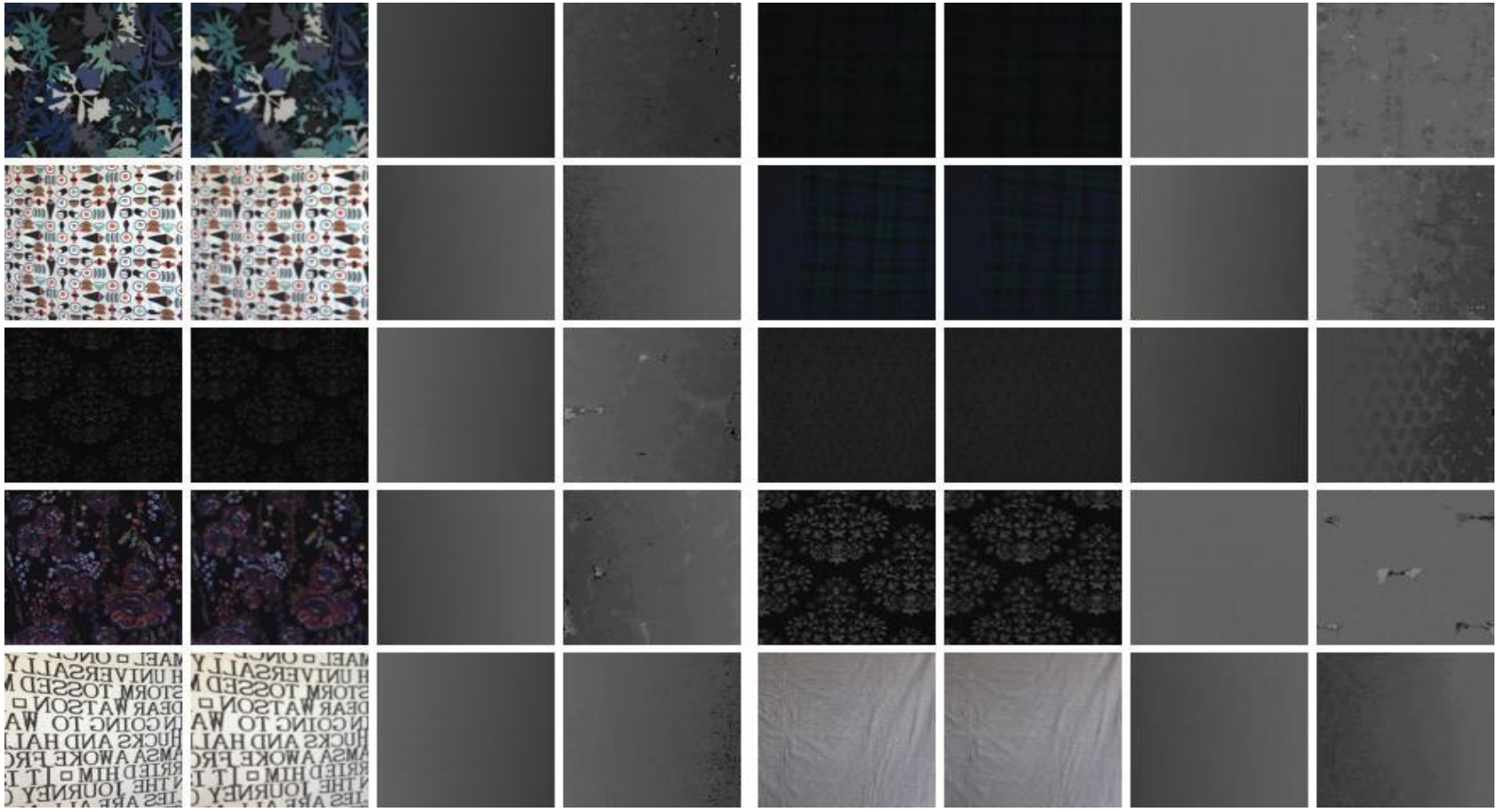
(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.30. K01-K03 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 29. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.



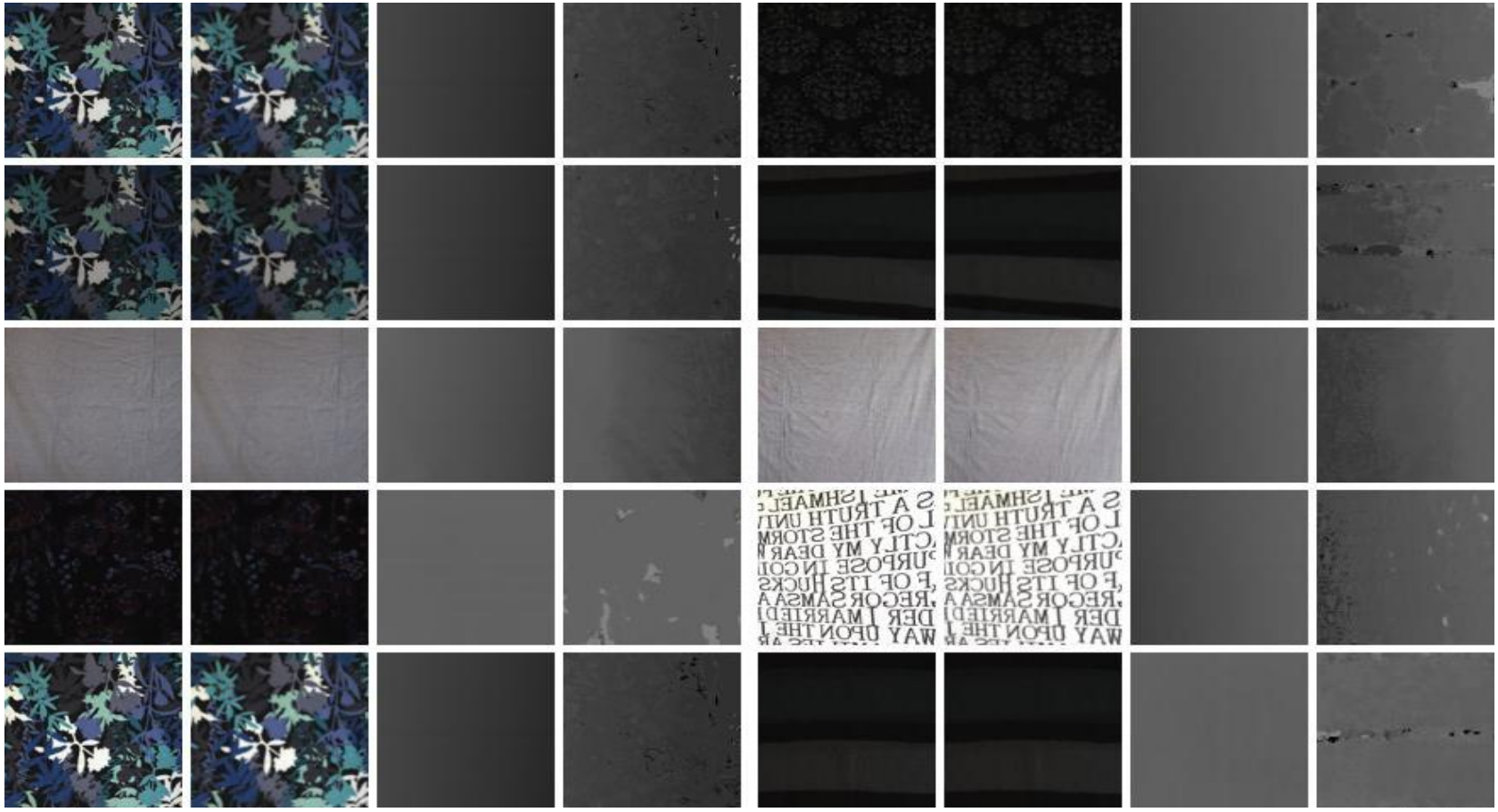
(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.31. K01-K03 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 30. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.



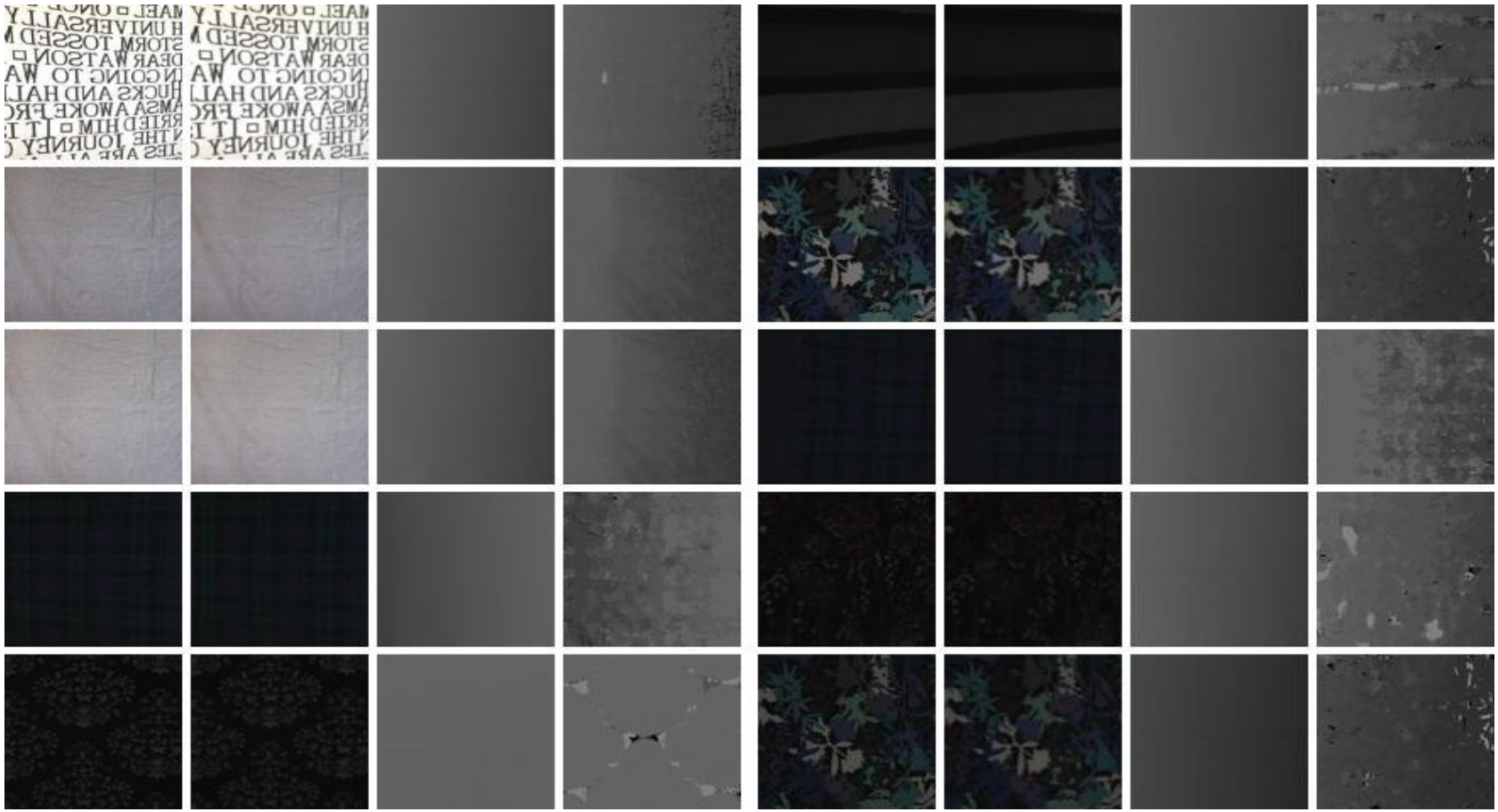
(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.32. K01-K03 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 31. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.



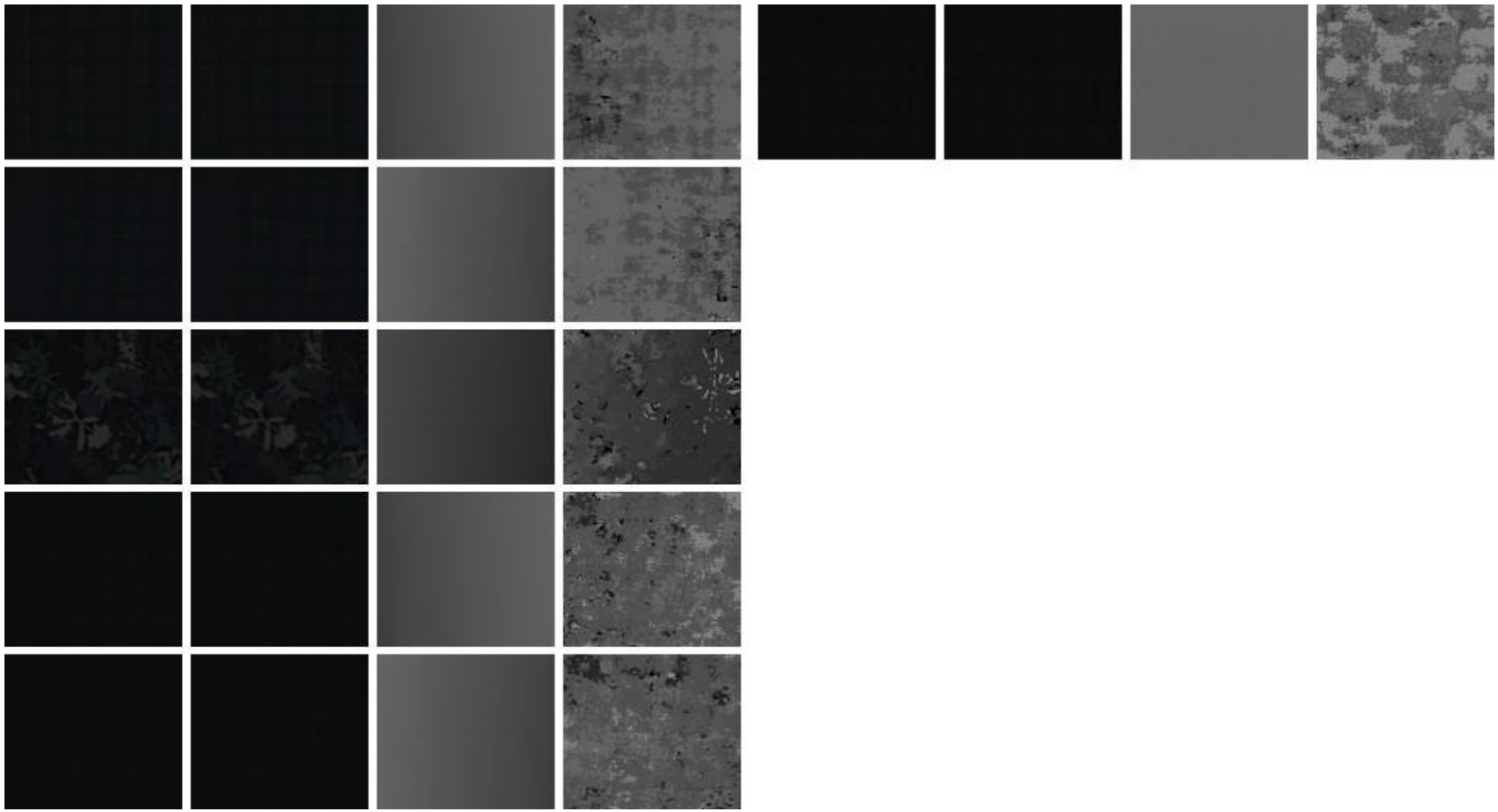
(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.33. K01-K03 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 32. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.



(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.34. K01-K03 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 33. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.



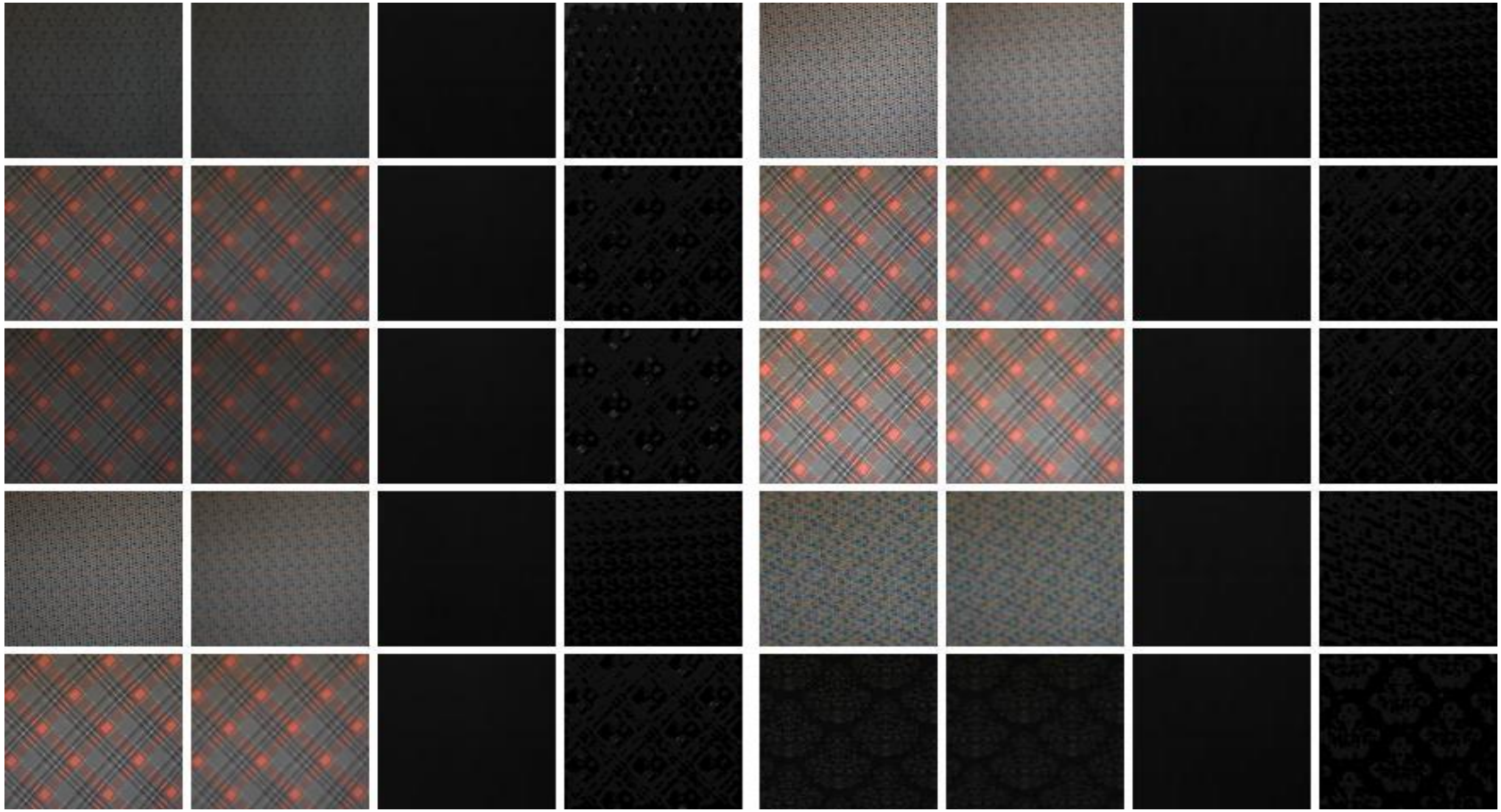
(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.35. K01-K03 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 34. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.

Figures E.36 through E.47 show the results of the DfD-Net trained on the Middlebury College dataset [7, 8] and tested on the subset of the synthetically blurred real world dataset that is a derivative of the k00 scene. The images are arranged in the order of lowest NRMSE score to highest NRMSE score. The order of arrangement of the images follows the same format outlined in Figure E.1.

(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.36. K00 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 1. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.



(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.37. K00 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 2. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.

(a) (b) (c) (d) (e) (f) (g) (h)

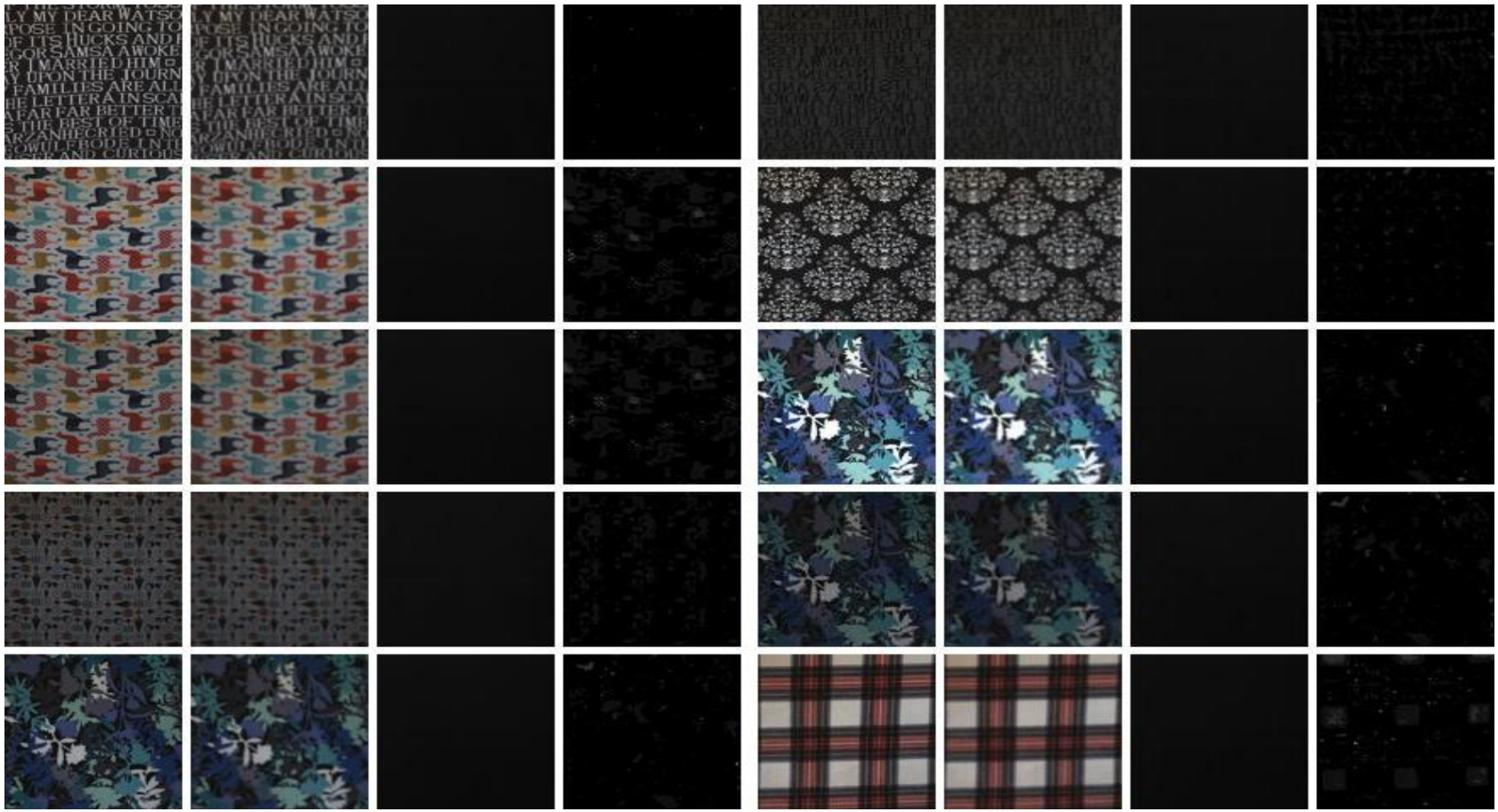
Fig. E.38. K00 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 3. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.

(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.39. K00 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 4. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.

(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.40. K00 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 5. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.

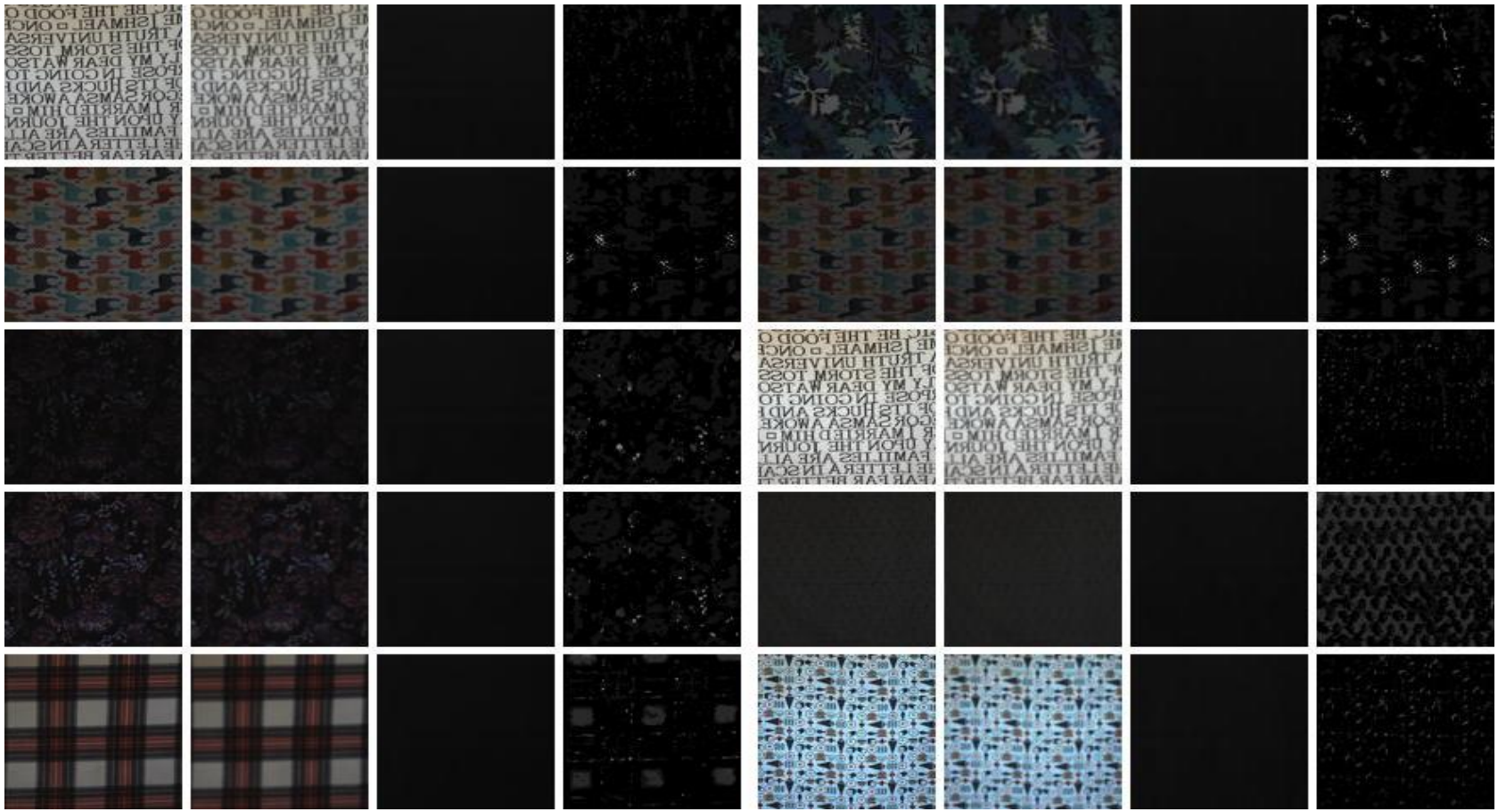


(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.41. K00 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 6. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.

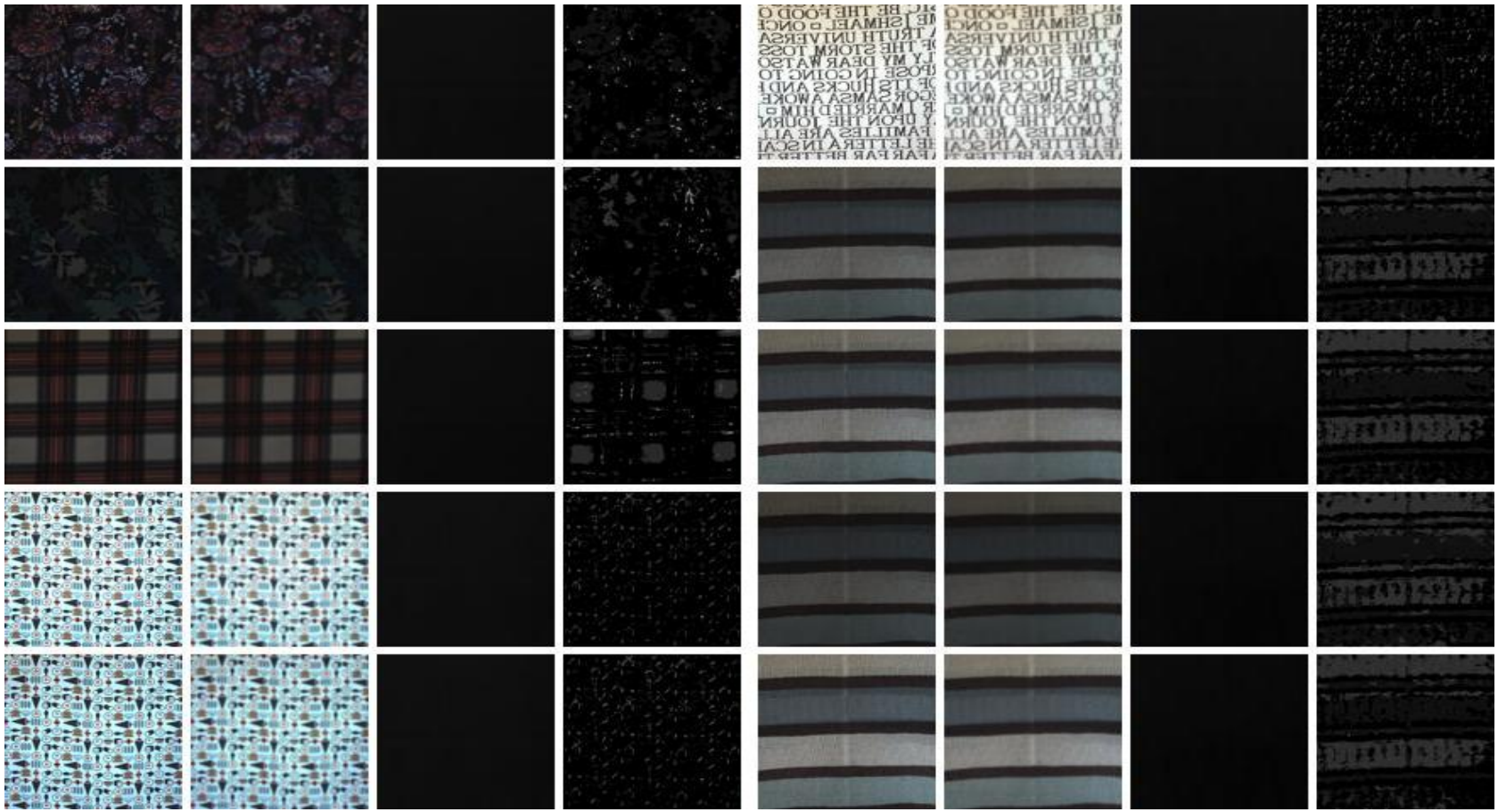
(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.42. K00 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 7. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.



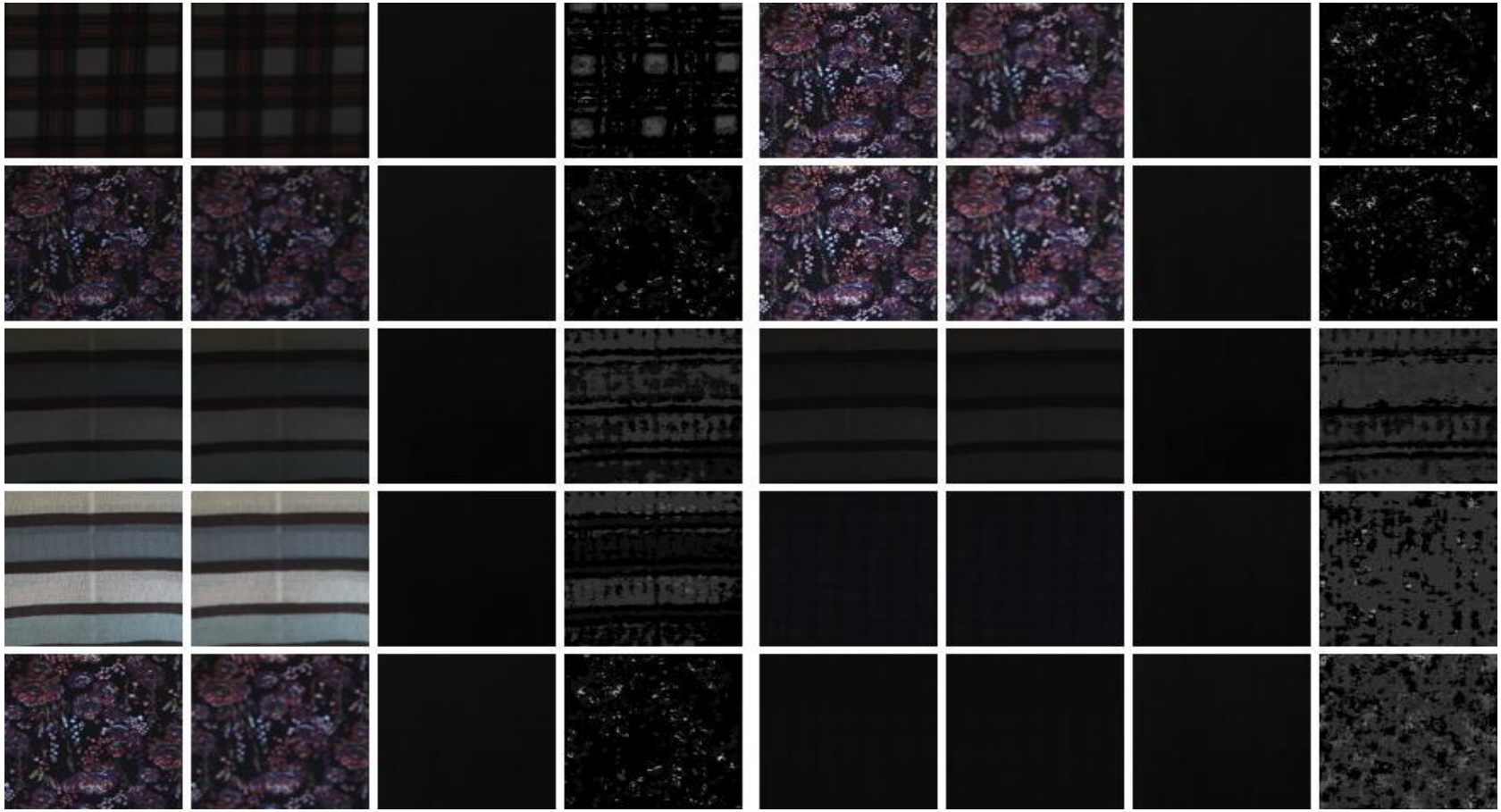
(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.43. K00 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 8. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.



(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.44. K00 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 9. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.

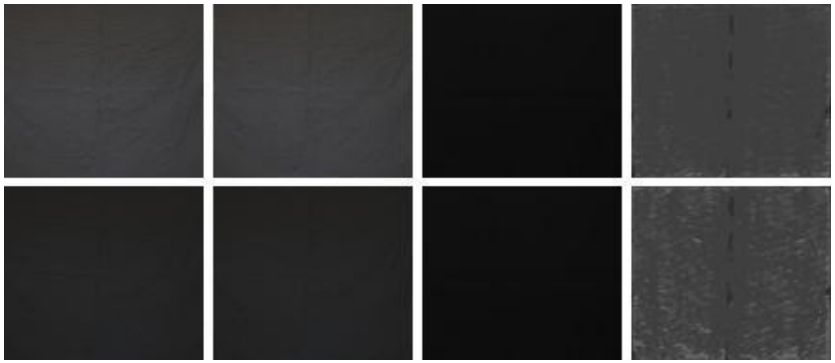


(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.45. K00 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 10. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.

(a) (b) (c) (d) (e) (f) (g) (h)

Fig. E.46. K00 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 11. (a) & (e) In-focus Image, (b) & (f) Out-of-focus Image, (c) & (g) Inverted Ground Truth Depth Map and (d) & (h) DfD-Net Computed Depth Map.



(a) (b) (c) (d)

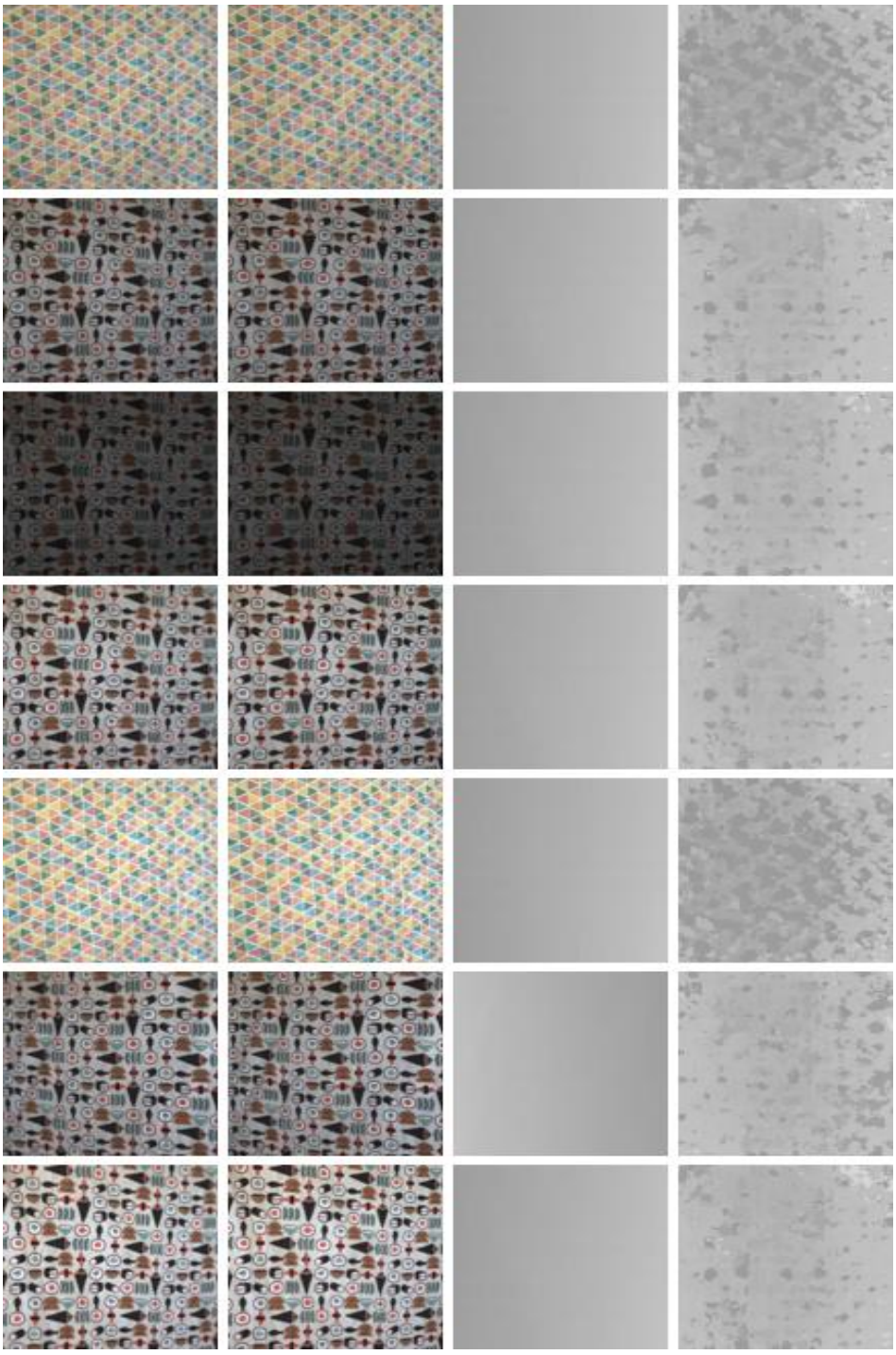
Fig. E.47. K00 Scene Type Performance Results for the DfD-Net on the Synthetically Blurred Real World Dataset - Part 12. (a) In-focus Image, (b) Out-of-focus Image, (c) Inverted Ground Truth Depth Map and (d) DfD-Net Computed Depth Map.

F. DFD-NET REAL WORLD DATASET RESULTS

Figures F.1 through F.6 show the results of the DfD-Net trained and tested on the real world dataset. The images are arranged in the order of lowest NRMSE score to highest NRMSE score.

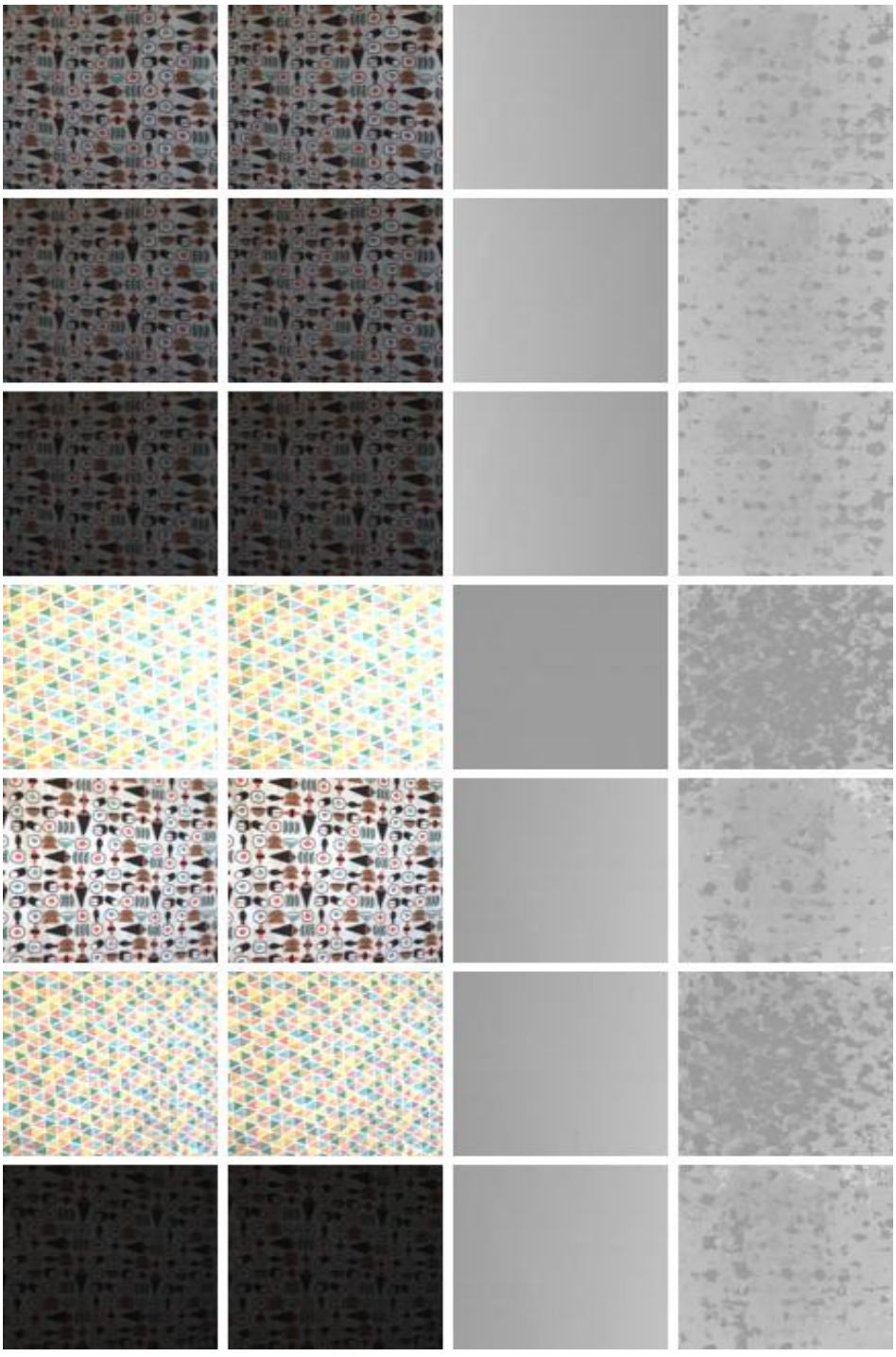
(a) (b) (c) (d)

Fig. F.1. Performance Results for the DfD-Net on the Real World Dataset - Part 1. (a) In-focus Image, (b) Out-of-focus Image, (c) Ground Truth Depth Map and (d) DfD-Net Computed Depth Map.



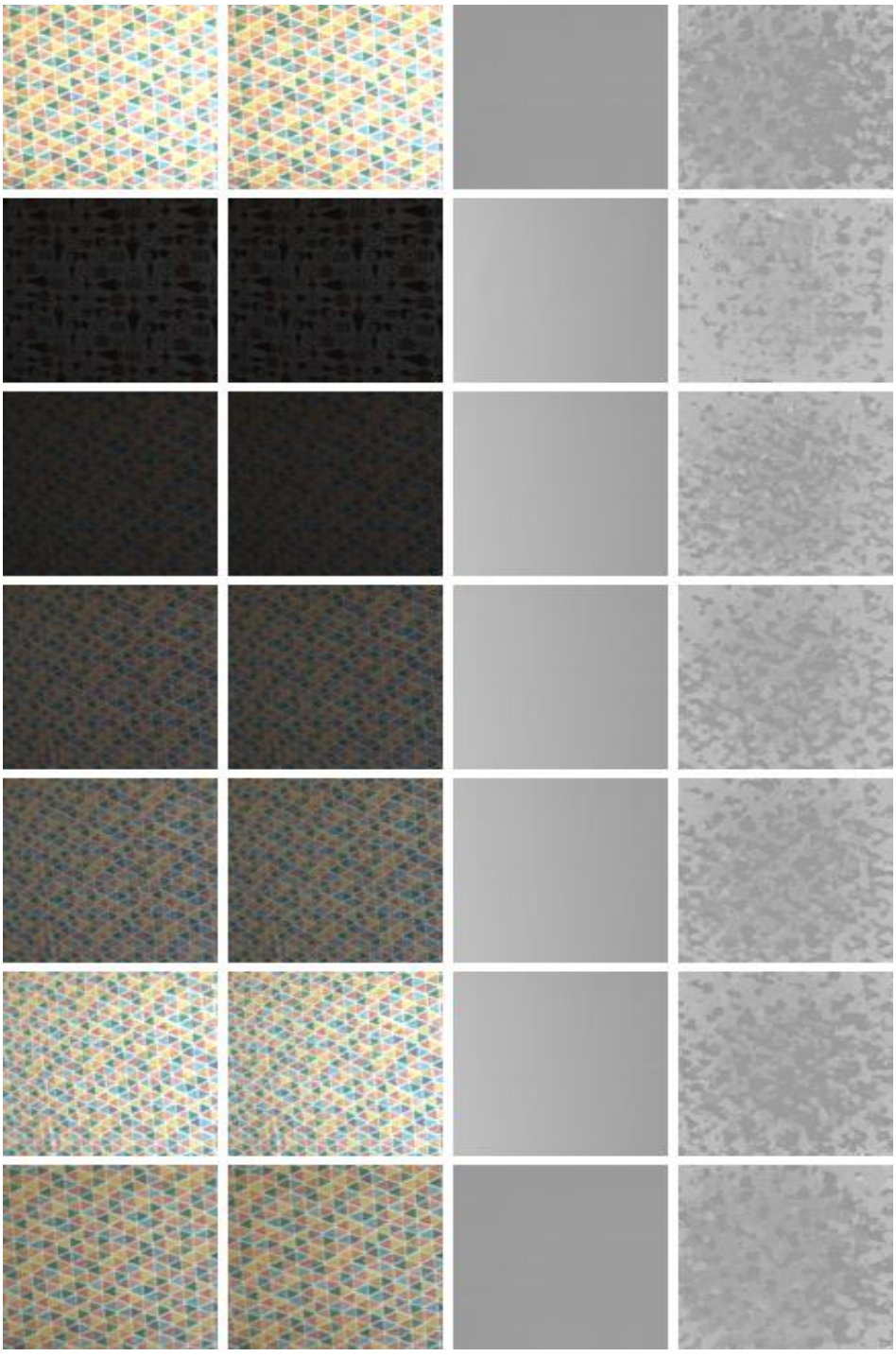
(a) (b) (c) (d)

Fig. F.2. Performance Results for the DfD-Net on the Real World Dataset - Part 2. (a) In-focus Image, (b) Out-of-focus Image, (c) Ground Truth Depth Map and (d) DfD-Net Computed Depth Map.



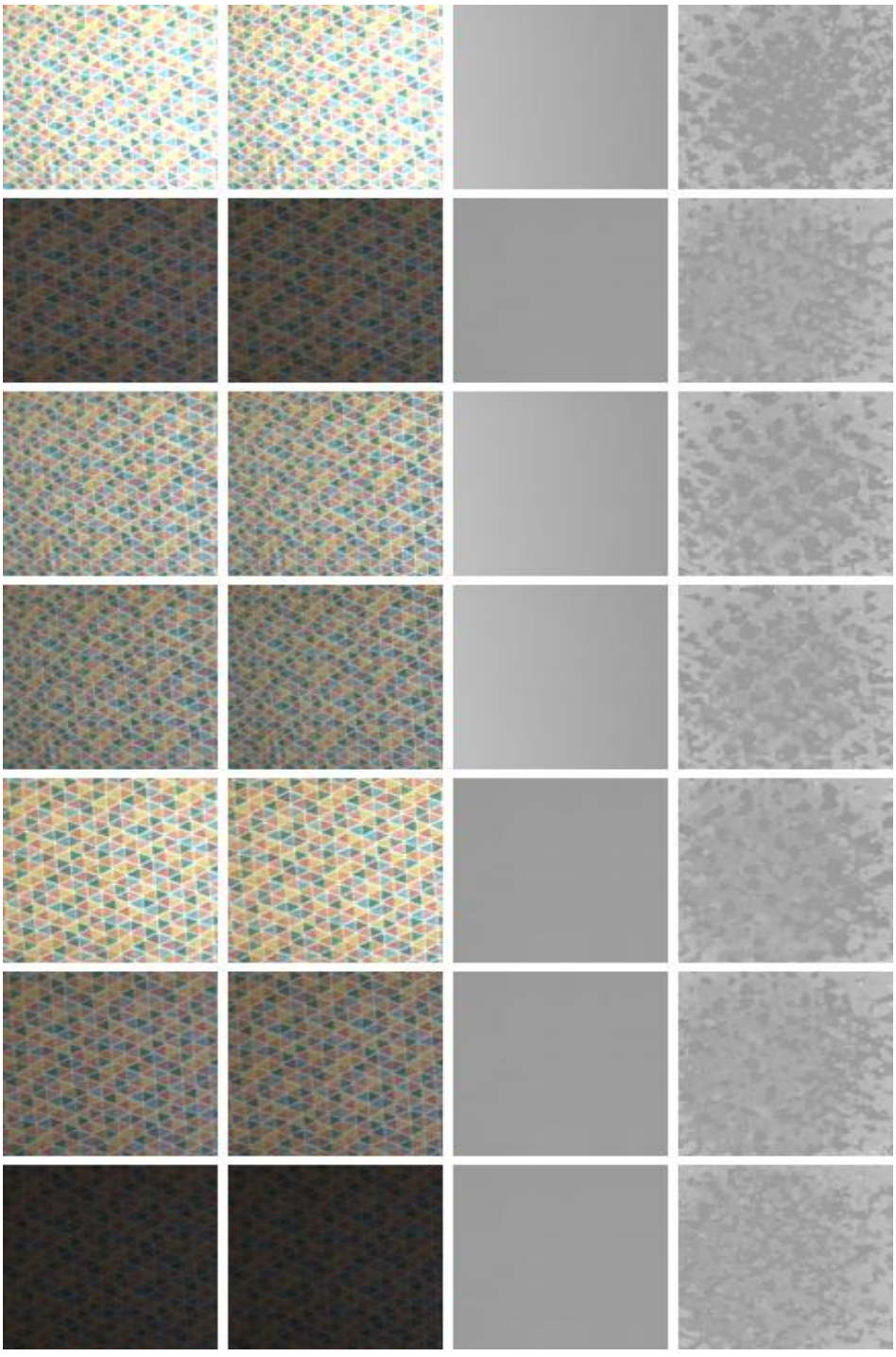
(a) (b) (c) (d)

Fig. F.3. Performance Results for the DfD-Net on the Real World Dataset - Part 3. (a) In-focus Image, (b) Out-of-focus Image, (c) Ground Truth Depth Map and (d) DfD-Net Computed Depth Map.



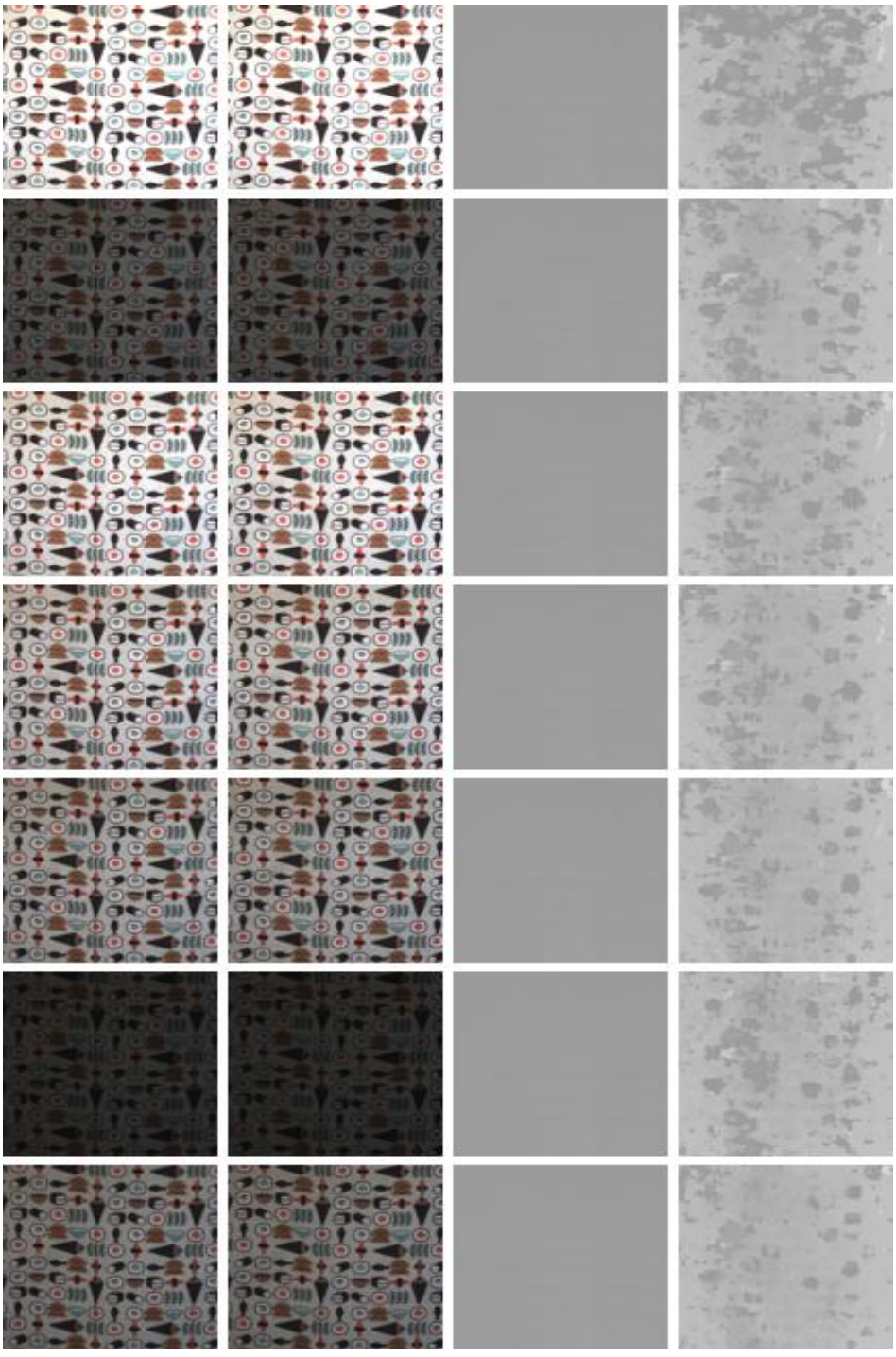
(a) (b) (c) (d)

Fig. F.4. Performance Results for the DfD-Net on the Real World Dataset - Part 4. (a) In-focus Image, (b) Out-of-focus Image, (c) Ground Truth Depth Map and (d) DfD-Net Computed Depth Map.



(a) (b) (c) (d)

Fig. F.5. Performance Results for the DfD-Net on the Real World Dataset - Part 5. (a) In-focus Image, (b) Out-of-focus Image, (c) Ground Truth Depth Map and (d) DfD-Net Computed Depth Map.



(a) (b) (c) (d)

Fig. F.6. Performance Results for the DfD-Net on the Real World Dataset - Part 6. (a) In-focus Image, (b) Out-of-focus Image, (c) Ground Truth Depth Map and (d) DfD-Net Computed Depth Map.