RAW DEPTH IMAGE ENHANCEMENT USING A NEURAL NETWORK

by

Xuan Xie

A dissertation submitted in partial fulfillment
of the requirements for the degree

of

DOCTOR OF PHILOSOPHY

in

Electrical Engineering

Approved:

Scott E. Budge, Ph.D. Major Professor	Jacob Gunther, Ph.D. Committee Member
Todd K. Moon, Ph.D. Committee Member	Rose Q. Hu, Ph.D. Committee Member
Marvin W. Halling, Ph.D. Committee Member	Richard Cutler, Ph.D. Interim Vice Provost for Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2021

ABSTRACT

Raw Depth Image Enhancement Using A Neural Network

by

Xuan Xie, Doctor of Philosophy

Utah State University, 2021

Major Professor: Scott E. Budge, Ph.D.
Department: Electrical and Computer Engineering

With the advent of inexpensive optical range-measurement devices (Kinect, Lidar), depth images have become a widely used data format. However, cheap devices come at the cost of low accuracy and high errors. Compared to the already highly developed color image processing, the quality of depth images is however limited by the performance of the gathering equipment and the underdeveloped processing techniques. Traditional depth image post-processing techniques have been difficult to be genuinely applied to industrial environments. In recent years, processing depth images with deep learning techniques has become a promising research direction, and in this dissertation, two neural networks with different structures are proposed as two different techniques for augmenting raw depth data. Both methods take advantage of auxiliary color image information.

Ordinary optical cameras typically capture images at two to ten times or higher resolution than lidar. The super-resolution study of depth images has been very challenging. This paper proposes a convolutional network-based framework that uses techniques such as residual network and dense network to improve accuracy while using an universal coordinate system as a channel to connect depth images and auxiliary color images.

The original depth images usually have a massive number of errors, the most severe type of which is dropout error, which is the complete loss of depth information. The loss

of information is a massive problem for both post-processing and applications. In this dissertation, a framework based on generative adversarial network is built to repair the lost information. The framework's core technology is a two-channel attention sub-network for color and depth based on the attention mechanism.

This dissertation presents depth images processed by these two frameworks, and the quality of the processed images is significantly improved compared to the original images. The great potential of deep learning techniques in the field of deep image processing is shown.

(115 pages)

PUBLIC ABSTRACT

Raw Depth Image Enhancement Using A Neural Network

Xuan Xie

The term image is often used to denote a data format that records information about a scene's color. This dissertation object focuses on a similar format for recording distance information about a scene, "depth images". Depth images have been used extensively in consumer-level applications, such as Apple's Face ID, based on depth images for face recognition.

However, depth images suffer from low precision and high errors, and some post-processing techniques need to be utilized to improve their quality. Deep learning, or neural networks, are frameworks that use a series of hierarchically arranged nonlinear networks to process input data. Although each layer of the network is limited in its capabilities, the learning capacity accumulated by the multilayer network becomes very powerful. This dissertation assembles two different deep learning frameworks to solve two different types of raw image preprocessing problems. The first network is the super-resolution network, a nonlinear interpolation of low-resolution deep images through the deep network to obtain high-resolution images. The second network is the inpainting network, which is used to mitigate the problem of losing specific pixel data in the original depth image for various reasons.

This dissertation presents deep images processed by these two frameworks, and the quality of the processed images is significantly improved compared to the original images. The great potential of deep learning techniques in the field of deep image processing is shown.

"Isn't it a pleasure to learn and constantly practice what is learnt?"
Confucius

ACKNOWLEDGMENTS

Of all the people I would like to thank, I would first like to express my sincere gratitude to my supervisor, Dr. Scott Budge. Without his patient help and guidance, I definitely would not have made it through this journey. I would also like to say a special thank you to the most important one in my life, Dr. Wennan Zhu. Getting a Ph.D. degree was very difficult, but fortunately we all persevered. I also acknowledge my parents and elder brother for all the financial and emotional support they have given me over the years. I would also like to thank my committee members and department staffs. They gave me a lot of enthusiastic help.

I am thankful to the students in the Center for Advanced Imaging Ladar I've worked with, Neeraj Badamikar, Taylor Bybee, Cody Killpack, Bikalpa Khatiwada, and Trevor Welch. Each of you have given me valuable advice on my work. I'm lucky to be working with you guys. I would also like to thank the friends I meet in Logan. they are Dr. Yiran Xu, Dr. Haijian Sun, Zhengfei Rui, Qun Wang, Dr. Bo You, Dr. Han Zhang, and Qi Zhou. Life is richer because of all of you.

Finally, I'd like to thank my landlord's two cats, Pumpkin and Patrick, for staying with me through the hard time I wrote my dissertation alone at home because of the pandemic.

Xuan Xie

CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

ACRONYMS

CAIL	Center for Advanced Imaging Ladar
CMOS	Complementary Metal–Oxide–Semiconductor
CNN	Convolutional Neural Network
COP	Center Of Projection
EM	Earth-Move
EO	Electro-Optical
FOV	Field of View of a Camera
GAN	Generative Adversarial Network
GP	Gradient Penalty
HR	High Resolution
LIDAR	light detection and ranging
LR	Low Resolution
LSTM	Long Short-Term Memory
NLM	Non-Local Means
NN	Neural Network
ReLU	Rectified Linear Unit
RGB	Red-Green-Blue (a colorspace)
SGD	Stochastic Gradient Descent
TOF	Time of Flight
USU	Utah State University
WGAN	Wasserstein GAN

CHAPTER 1

INTRODUCTION

1.1 Big data and deep learning

If we were to make a ranked list of buzzwords across fields over the past decade, big data and deep learning must be at the top. Motivated by various factors, both these fields have achieved phenomenal success in the last decade and still have unlimited potential for the foreseeable future.

In the past, a scientific breakthrough usually took more than a decade or even decades to be applied to the real world. Taking Low-density parity-check codes (LDPC) as an example [1], it was firstly introduced in 1996. However, it was not until 2009 that this advanced technology was written into the WLAN standard 802.11n [2], and it will be another 2 to 3 years before consumers actually buy electronic devices that support it. Nevertheless, in these emerging fields, competition is so fierce that the most advanced discoveries can be used in the production environment within months and quickly replaced by more advanced technologies. This discussed reports to do some fundamental research on a particular class of data, depth images, using deep learning techniques.

1.1.1 Data and depth data

Data is the carrier of information and is the primary material required for the vast majority of scientific research activities. Research and data analysis can be divided into two major categories: professional data analysis and general data analysis. The former category analyzes highly specialized data, such as genetic profiles, stock trading records, weather data, medical records, social networks, etc. This data research category requires highly specialized knowledge combined with general data analysis techniques such as dimensionality reduction, regression, classification, visualization, etc. Generic data processing only deals

with the basic data storage formats, doing some general data analysis and research, and providing tools for higher-level research. There are roughly these basic data storage formats: array (including matrix), table, image, audio, video, and text. More sophisticated data formats are generally a combination of some of these formats. The research object of this dissertation is a subcategory of image data, depth images. Like photos that store color information, 3D images that store 3D information about objects have a long history. Kinect, introduced by Microsoft in 2010, was the first consumer-grade 3D information-gathering tool with huge sales. Despite its initial use as an experience-enhancing accessory for the gaming console Xbox, it soon found a place in academia. Since then, more inexpensive depth information gathering devices have appeared on the market, such as the latest iPad Pro with a small built-in light detection and ranging (LIDAR) camera. depth image applications at the consumer level are also maturing. The most iconic application is Apple's Face ID, which extracts features from depth images obtained by scanning a face. It then unlocks the phone by matching the stored features. However, compared to the highly sophisticated color image processing, depth image processing techniques, especially pre-processing, are still relatively few. Today, anyone can take clear and vivid high-resolution photos with their smartphone, even under strong sunlight, night, cloudy sky, and other less-than-ideal shooting conditions. This is because many advanced pre-processing algorithms, like noise reduction, inpainting, High-dynamic-range imaging (HDR), etc., have been applied to the raw optical image before it is converted into a photo. However, in the field of depth imaging, such techniques are still in the laboratory stage, and researchers are usually dealing directly with the raw data. 3-D information is stored in various formats, the most common being point cloud, voxel, and depth image. This dissertation focuses on the processing of depth images.

In three-dimensional computer graphics and computer vision, a depth map is a matrix stored in image form. In contrast to a color image, each element in a depth image stores not color information about the point at which the line of sight intersects the object, but the distance from that point to the point of view. This distance is not the Euclidean distance

between two points, but the Z-axis distance between two points in a three-dimensional coordinate system with the point of view as the origin and the principal ray as the Z-axis.

1.2 Related Work

1.2.1 Texel camera

The concept of the texel camera is first introduced by The Center for Advanced Imaging lidar (CAIL) at Utah State University (USU) in 2005. The first generation of texel camera is a simple alignment of two cameras [3]. In 2007, the second generation was developed by Boldt et al. [4]. From that time, the design of the texel camera is fixed. The second generation contains a Micron 1280×1024 CMOS imaging sensor, a Canesta 64×64 CMOS flash lidar sensor. The next version of texel camera was assembled in 2014. The lidar sensor was updated to PMD CamCube 2.0 204×204 sensor [5]. The latest texel camera currently in development and use was assembled in 2019 [6]. At this time a new OS-1-16 panorama lidar camera is adopted. Unfortunately, the resolution still does not meet the current needs for high accuracy tasks.

1.2.2 Deep learning

As a branch of machine learning, deep learning has become one of the hottest and fastest-growing technologies in this field. One of the main drawbacks of traditional machine learning is its inability to process raw data directly. Researchers and engineers have to acquire domain expertise and spend much time designing a feature extractor and then feeding the features to the later learning system. On the other hand, deep learning treats feature extraction and feature combination as nonlinear functions at different levels. One or more layers of subnetworks can approximate the nonlinear functions at each level. These subnetworks are cascaded together and trained with large amounts of data to get the desired result without human intervention. This is the core concept of deep learning.

Deep learning encompasses many research directions, with the most important ones being macro network structure, micro layer structure, and optimization techniques. The most

common network architectures include Convolutional Neural Network (CNN) [7], Recurrent Neural Network (RNN) [8], Long Short-Term Memory (LSTM) [9], and the latest Transformer [10], each containing many well-established frameworks. The layer structure includes various generic single-layer networks, the most common of which are the following, convolution layer, activation layer, normalization layer, and dropout layer. Any deep network is composed of these basic structures. Optimizers are also an important component of deep learning. The most frequently used optimizers are: Stochastic Gradient Descent (SGD), Adagrad [11], RMSprop, Adam [12], and Nadam [13]. Layer structure and optimizers will be covered in more detail in the next chapter.

The history of CNNs can be traced backs to the 1990s. After the initial success on the document recognition problem [14], its performance was limited by the lack of data and low computing speed. In the middle of the 2010s, deep learning went through a renaissance in processing images, video, speech, and audio [15]. Because the convolutional operation can take advantage of spatially-local correlation, CNNs are naturally suited for data with a regular structure like imagery. Three dimensional convolution based on the voxel system is a common approach to extend CNNs to a higher dimension [16]. However, 3D convolution consumes exponentially more computation resources and memory, which is not affordable in many cases. Moreover, the voxel system's resolution is relatively coarse for the interpolation problem, so we still choose 2D CNN as our primary approach.

1.2.3 Depth image upsampling

The traditional methods for image-guided depth upsampling fall into three categories [17], local methods, global methods, and other methods. Local methods are usually filter-based approaches. The bilateral filter is the most prominent technique, and it derives a lot of variants and extensions [18]. One representative variant is the joined bilateral filter, which requires a guidance image [19]. Riemens et al. extend the bilateral filter to a multistep implementation [20]. The non-local means (NLM) filter is famous for its extensive image processing usage, including the upsampling problem. NLM can be seen as a generalization of the bilateral filter. One successful application of NLM is presented by Huhle et al. [21].

Global methods are classic machine learning problems. The difference between measured and estimated depth data is obtained by designing several feature extraction layers and a cost function [22–27]. The feature extraction layers are optimized via the gradient descent algorithm. Many global applications are based on Markov Random Field (MRF) [23]. Other approaches use cost functions that contain similar terms in the MRF [25–27].

Other methods include segmentation of color and depth images [28], Bayesian approach [29], and the random sample consensus (RANSAC) algorithm [30]. Since CNNs demonstrate their power in the image process area, it becomes a promising research direction to solve the upsampling problem. Li et al. propose an original design of a CNN and exhibit some exciting results [31].

1.2.4 Depth image inpainting

Researchers use the terminology "image inpainting" to describe the challenge of repairing these depth images containing lost or corrupted parts. Image inpainting includes two sub-tasks. The first is to fill the missing pixels, and the second is to restore the deteriorated area. The location of missing pixels is determined so we can create a mask layer to indicate it. However, corrupt pixels may still contain raw information of the real world that would facilitate the restoring process. Nevertheless, extra work is needed to separate the corrupted and uncorrupted area. In this dissertation, only the first topic is discussed.

Like other research areas in computer vision and image processing, previous image inpainting methods fall into two categories, non-deep-learning-based methods and deep-learning-based methods. Since there is a genetic similarity between a depth and color image, and the latter is a more popular research field, existing mature color image inpainting methods inspired a large portion of depth image inpainting methods. For non-deep-learning based methods, a significant ideology is to find a possible approximate patch in the same image to fill the missing hole after some necessary adjustments, so the criterion to evaluate the "similarity" is the crux of the matter. The most straightforward thought is defining "similarity" as depth value "similarity", which is the core concept of the famous bilateral

filter method and its successors [19, 32]. The original bilateral filter is given by

$$I^{\text{filtered}}(x) = \frac{1}{\mathbf{W}_p} \sum_{\mathbf{x}_i \in \Omega} I(\mathbf{x}_i) f_r(\|I(\mathbf{x}_i) - I(\mathbf{x})\|) g_s(\|\mathbf{x}_i - \mathbf{x}\|), \tag{1.1}$$

where

$$\mathbf{W}_p = \sum_{\mathbf{x}_i \in \Omega} f_r(\|I(\mathbf{x}_i) - I(\mathbf{x})\|) g_s(\|\mathbf{x}_i - \mathbf{x}\|). \tag{1.2}$$

Equation 1.1 shows the two critical factors to determine the filtered value of a pixel $I^{\text{filtered}}(x)$, the difference of the two pixels' value $\|I(\mathbf{x}_i) - I(\mathbf{x})\|$ and the distance of the two pixels' location $\|\mathbf{x}_i - \mathbf{x}\|$. The other two parameters f_r and g_s are range (or depth) filter kernel and spatial filter kernel respectively to smooth the difference value. They are usually Gaussian kernels. $\mathbf{W}_p$ is a normalizing factor.

It is much more difficult to extract features from a depth image than from a color image because depth images are much smoother and have smaller variance than color images. Like other research topics in depth map enhancement, color information, i.e., the texture of objects, plays an important role in recovering depth image. Joint image filtering techniques treat texture "similarity" as another clue to infer the missing value in the depth image [33]. Besides that, many well developed color image inpainting methods can be adapted to reconstruct the 3D scenario, such as AGG-AR [34], smoothness priors [35], fast marching method [36], background surface extrapolation [37], color-depth edge alignment [38], low rank matrix completion [39], tensor voting [40], etc. Most of the methods as mentioned earlier, along with some other depth inpainting methods that utilize more than depth and color information, e.g., normals, illumination, shading, and semantic map [41], pay more attention to higher-dimensional features, such as edge, object surface, and rank matrix, etc. The common drawback to all methods stated before is the lack of versatility. Data collected by different devices are often missing one or more traits, so they do not apply to some state of the art inpainting frameworks. On the other hand, the auxiliary information is indispensable for specific inpainting technology. The smaller the number of categories of adopted data means better generalizability. It is a challenge but necessary to seek a balance

between the generalizability and the inpainting method's performance.

The rapidly evolving machine learning technology, especially deep learning technology, fueled many respects of the computer vision and image processing field. Several effective and mature frameworks have been applied to image inpainting task, e.g., basic CNN [42], context encoder [43], and Patch Synthesis [44].

In view of information theory, image inpainting, including both color and depth images, is trying to "fabricate" non-existing information from existing information. It can never fix ruined images to perfect real-like pictures as they should be. Furthermore, as an unsupervised machine learning topic, how to evaluate the quality of a fixed image is still a challenge. For the color image area, a few criteria were proposed to measure the performance of a generator, e.g., Inception Score (IS) [45] and Fréchet Inception Distance (FID) [46]. However, those criteria lack universality because they rely on a mature framework Inception V3 [47], which is trained on ImageNet. Only generated images that have considerable similarity to images in ImageNet can be judged by IS and FID. This issue is worse in the depth or more general 3D data learning field. Unlike the color image field which is primarily driven by a few community contests that need specific criteria to judge the performance in publications, the diversity of research subjects in the 3D area makes makes it less urgent to establish a universal evaluation criterion. The solitary judgment in the 3D area is still human recognition.

CHAPTER 2

RESEARCH BACKGROUND AND PREVIOUS WORK

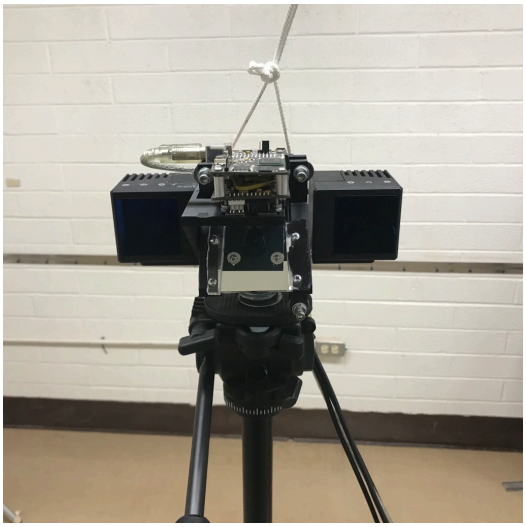
2.1 Texel Camera

This dissertation describes an attempt to overcome the defects and to exploit the potential of the texel camera. The texel camera hardware has evolved, e.g., lidar camera, EO camera, optical lens, and chips including a GPU, However, the data structure of the texel image remains concise and stable. The stable structure brings many conveniences to collect synthetic texel images from various fused 2D and 3D datasets, which is critical to the success of the result of the experiments. Since the number of texel images that can be captured by a texel camera in a reasonable time is far from adequate to train the neural network, synthetic texel images become a natural choice.

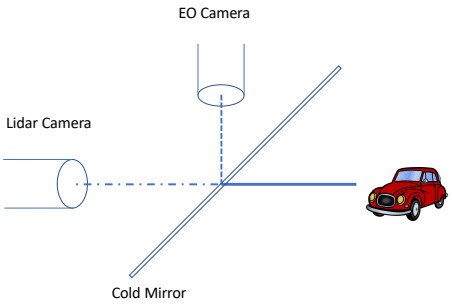
Texel camera is a combination of a lidar camera and an electro-optical (EO) camera. Both cameras are calibrated to be co-boresighted by a cold mirror. The cold mirror transmits the visible wavelength (for EO) and reflects near-infrared wavelength (for lidar). The cold mirror is placed correctly to ensure both cameras share the same center of projection, which means there is no parallax between the two cameras. Both cameras are assumed to follow the pinhole model. Usually, the resolution of the EO camera is much higher than that of the lidar camera. Figure 2.1a shows the lidar camera assembled and used in CAIL between 2014 and 2018, and Figure 2.1b is a diagram of the role of a cold mirror.

A special data format captured by the texel camera is called a texel image. A texel image is a 2.5-D image that means it is a 3-D image captured from a single perspective. Figure 2.2c is a diagram of a texel image, which is a fusion of the depth image of Figure 2.2b and the color image of Figure 2.2a. The color image can be seen as the texture to render the 3-D image.

If we compare the texel image with the prevailing RGB-D image, there are two different

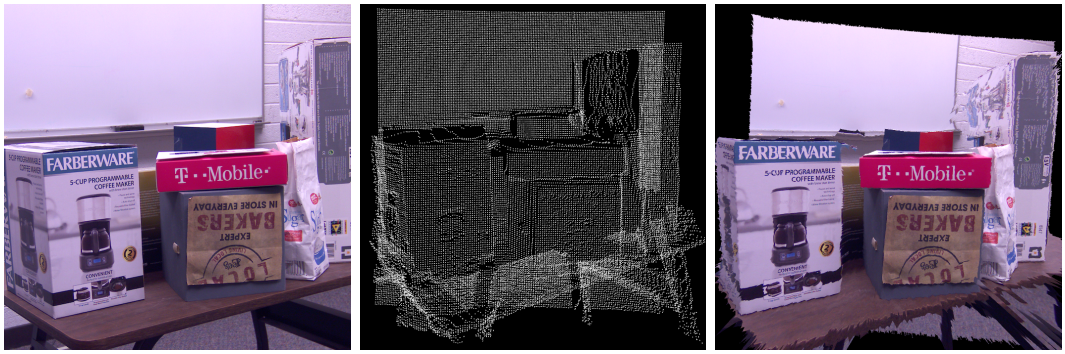


(a) Texel camera



(b) texel camera Structure

Fig. 2.1: texel camera



(a) Color Image (b) Depth Image (c) texel Image

Fig. 2.2: Texel image

significant properties between them. The first property is that the texel image has two different resolutions, and the RGB-D image has a single resolution. The second property is that for the RBG-D image, each pixel has the same depth of the information channels, but for the texel image, the pixels in the EO image usually do not directly correspond with the pixels in the lidar image. An interpolation operation is needed to render the 3-D data. In a texel image file, an EO image and a lidar image are stored separately. A polynomial mapping is used as an auxiliary way to represent the relationship between the two images. We choose the coordinate system of the EO image as the base system. A

normalized coordinate (x_n, y_n) of a point in lidar image is mapped to a uniform coordinate (u, v) in the EO image. The u and v value are both in the $(0, 1)$ range. The (u, v) values of the four corners of the image are $(0, 0)$ (bottom left), $(0, 1)$ (bottom right), $(1, 0)$ (top left), and $(1, 1)$ (top right). If there is an $M \times N$ image, the (u, v) values of a pixel at m^{th} row and n^{th} column are determined

$$u = 1 - \frac{m}{M} \qquad v = \frac{n}{N} \tag{2.1}$$

where M and N are the row and column numbers of the of the image.

As shown in Figure 2.3, The pinhole model is a coordinate system transformation Φ from depth image space $\boldsymbol{D}$ to EO image space $\boldsymbol{S}$, projecting a point P in S into a pixel P' in D. The P's coordinate system is the Euclidean coordinate system, in meters. The pixel P' is identified by its row number m and column number n. First, P's coordinates (X, Y, Z) is first projected to the normalized plane.

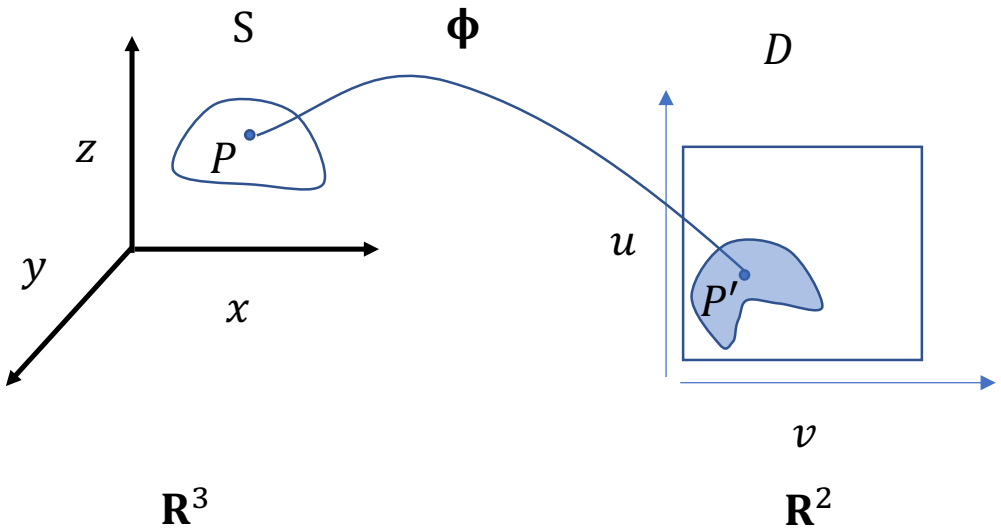


Fig. 2.3: uv mapping Φ from $\mathbb{R}^3$ to $\mathbb{R}^2$

The normalized coordinates x_n and y_n are described as

$$x_n = \frac{X}{Z} \qquad\qquad y_n = \frac{Y}{Z}. \tag{2.2}$$

If the pinhole model is ideal, then m and n are given by

$$\begin{pmatrix} m \\ n \\ 1 \end{pmatrix} = K \begin{pmatrix} x_n \\ y_n \\ 1 \end{pmatrix} \qquad\qquad K = \begin{pmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}, \tag{2.3}$$

where K is the camera matrix. For K, the parameters f_x and f_y are focal length, and s is the skew factor that accounts for a shear of the coordinate system. Because the projection of the optical axis on the normalized plane is centered on the plane and the final image is in the first quadrant, a translation vector $(c_x, c_y)^T$ is needed to move the origin to the upper left corner.

But the reality is that cameras and lenses are usually imperfect, so Equation 2.4 and 2.5 are adopted to calibrate the camera. The higher-order terms of Equation 2.4 and 2.5 are adjustments to various types of distortion, such as image tilting, the minor offset of the COPs, and the difference in curvature of the two systems.

If m and n is determined, u and v is also determined by

$$\begin{aligned} m = {} & g_1 + g_2\tilde{x}_n + g_3\tilde{x}_n^2 + g_4\tilde{x}_n^3 + g_5\tilde{y}_n + g_6\tilde{x}_n\tilde{y}_n \\ & + g_7\tilde{y}_n^2 + g_8\tilde{x}_n\tilde{y}_n^2 + g_9\tilde{x}_n^5 + g_{10}\tilde{x}_n\tilde{y}_n^4 + g_{11}\tilde{x}_n^3\tilde{y}_n^2 \end{aligned} \tag{2.4}$$

$$\begin{aligned} n = {} & h_1 + h_2\tilde{y}_n + h_3\tilde{y}_n^2 + h_4\tilde{y}_n^3 + h_5\tilde{x}_n + h_6\tilde{y}_n\tilde{x}_n \\ & + h_7\tilde{x}_n^2 + h_8\tilde{y}_n\tilde{x}_n^2 + h_9\tilde{y}_n^5 + h_{10}\tilde{y}_n\tilde{x}_n^4 + h_{11}\tilde{y}_n^3\tilde{x}_n^2 \end{aligned} . \tag{2.5}$$

It should be noted that the (u, v) values of each pixel in the lidar image is fixed when the calibration process is done. Then all the (u, v) values for each lidar measurement are stored in a lookup table.

2.2 Convolutional Neural Network

2.2.1 AlexNet

Although Convolutional Neural Network (CNN) was firstly introduced in the 1990s [7], the origin of the modern CNN we study and use today would have been AlexNet presented by Krizhevsky et al. in 2012 [48]. Designed as an object recognition framework, AlexNet is a traditional CNN with a deeper structure. It improves the performance of the image classification task significantly. Figure 2.4 shows the structure of AlexNet. In this diagram, each blue block except the left-most one represents a convoluted tensor and each red pyramid represents a convolution operation. The input image is a tensor with a large width and height (224×224) cropped from a 256×256 image, and its depth (3) represents the Red-Green-Blue (RGB) channels. The author claimed that the network suffered from substantial overfitting without the cropping operation. For each layer, the output tensor will have smaller height and width than the input tensor due to the convolutional stride greater than 1 and the max pooling operation. In the meantime, the number of kernels per layer, which can be defined by the designer, is growing, and the number of kernels determines the depth of the output. This is reflected in the diagram as the shape of the sensor becomes taller and slenderer. Each element in a tensor is called a "feature". From left to right, low-level features are extracted and combined into high-level features, which are the two major functions of deep neural network (DNN) mentioned in the previous chapter. At the end of the diagram, three fully-connected layers, which are labeled as "Dense", combine all the high-level features and provide an n-length vector (in Figure 2.4, $n = 1000$) as the output. The ith scalar in the vector indicates the possibility that the object in the input image falls into the ith category. AlexNet was also the first mainstream framework to use dropout technology. The next few sections will have an introduction to max pooing, dropout, and other techniques.

Inspired by the AlexNet, many more advanced CNNs are presented, such as ZFNet [49], VGGNet [50], GoogleNet [51], ResNet [52], and etc. In recent years, more and more new

structures and technology are emerging. In the following sections, A brief introduction of each category of work related to CNN is presented.

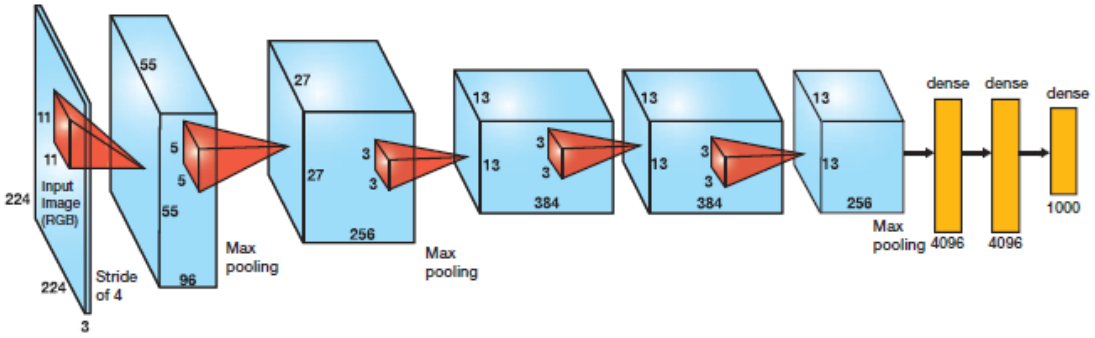


Fig. 2.4: AlexNet

2.2.2 Basic Concept

In the field of machine learning, a single convolutional operation is defined as a variable input matrix $\boldsymbol{x}$ (flatten into a vector) and an invariant matrix $\boldsymbol{w}$ (also flatten into a vector), called a "kernel", doing the vector inner product. In most cases, the inner product will be added to a scalar b, called the bias, to get a single scalar result z which can be defined as

$$z = \boldsymbol{w}^T \boldsymbol{x} + b. \tag{2.6}$$

When the input matrix is larger than the kernel, this operation is performed multiple times to get a output matrix. Figure 2.5 shows a diagram of a fully convolutional operation. Suppose there is a 10×10 input matrix $\boldsymbol{X}$, a 3×3 kernel $\boldsymbol{W}$, and an output $\boldsymbol{Z}$. First, a 3×3 patch in the top-left corner of $\boldsymbol{X}$ is cropped. This patch is convolved with $\boldsymbol{W}$, and the result is $z_{0,0}$. Next, slide the cropping window 1 step to the right and do another convolution, then $z_{0,1}$ is obtained. When this cropping window slides through the entire input matrix from left to right and from top to bottom, we get a complete 8×8 output $\boldsymbol{Z}$ which is called a "feature map".

Usually, the input has a third dimension, depth, and the corresponding kernel should

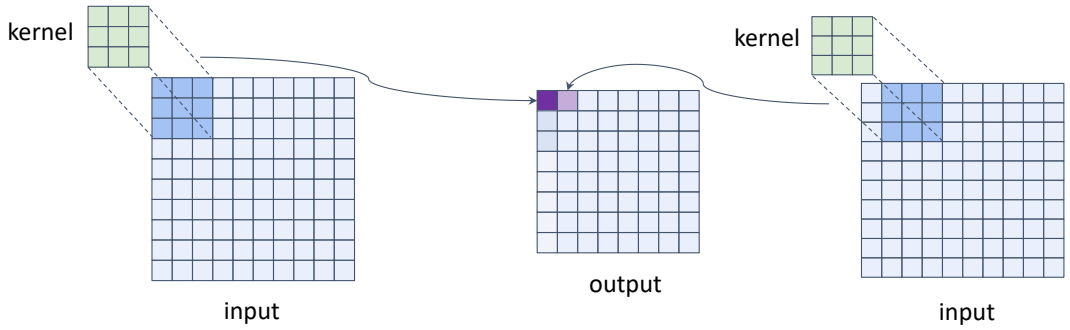


Fig. 2.5: A regular convolution diagram

have the same depth. Here we are going to introduce a new concept, "the tensor". In the field of information science, we usually call 0-dimensional data, i.e., single numbers, a scalar, 1-dimensional data a vector, and 2-dimensional data a matrix. But in the field of machine learning, especially deep learning, large amounts of data exist in higher dimensions, and this is where we call them tensors. Mathematically tensor has some additional properties, but that's beyond the scope of this dissertation. In some contexts, tensor can also refer to all dimensions of data in general. Both the input tensor and the kernel tensor are expanded into two 1-dimensional vectors when the convolution is computed, so Equation 2.6 still holds. In Figure 2.5, a single kernel results in one 8×8 feature map. If we create n different kernels and stack n different feature maps together, one $8 \times 8 \times n$ tensor is generated, and n is the total depth number or the total channel number of the tensor. So we can use a hyper-parameter n to control the depth of the output tensor. This is how to increase the depth of tensors in Figure 2.4.

The sliding steps, called "stride", can be greater than 1. The larger the stride, the smaller the width and height of the output. When stride is 1, the height and width of the output is less than the height and width of the input, because the size of the kernel is larger than 1. If we want the input and output to be equal in size, additional rows and columns need to be added around the input. This process is called "padding". The most common type of padding is called "valid padding" or "zero padding", which means that the elements in the added rows and columns are all 0. Assuming that the input is a square matrix with

side length I, the side length O of the output can be calculated by

$$O = \frac{I - K + 2 \times P}{S} + 1 \tag{2.7}$$

where K is the kernel size, P is the padding size, and S is the stride size. If the input is not a square matrix, just replace I with the height H and width W respectively. In a deep neural network (DNN), if a particular layer of the neural network uses the convolutional operation, that layer is called a convolutional layer, and if most layers of a neural network are convolutional layers, it is called a convolutional neural network (CNN). Corresponding to the convolutional layer is the fully-connected (FC) layer or "dense" layer, and a neural network built with fully-connected layers is call multilayer perceptron (MLP). Fully-connected layers can also be represented using Equation 2.6, except that in this case, $\boldsymbol{x}$ is not a small patch cropped from the input but the whole input itself, and $\boldsymbol{w}$ has the same size of $\boldsymbol{x}$. So for fully-connected layer, one $\boldsymbol{w}$ corresponds to one feature. Compared to the fully-connected layer, convolutional layer has a much lower computational complexity. Assuming that the size of an input to a layer in a neural network is 100×100, we want the output to have $100,000$ features. In the case of a fully-connected layer, we need $100 \times 100 \times 100,000 = 1 \times 10^9$ trainable parameters. If we use a convolutional layer with a kernel size of 5, we can set $S = 1$ and $P = 2$ to ensure that the input and output have the same size. Then we only need 100 kernels and the total number of parameters are $100 \times 5 \times 5 = 2,500$. The six order of magnitude difference between the two types of layer is huge. This is because for a fully-connected layer, each feature needs to use 100×100 independent parameters to generate it. But for a convolutional layer, every 100×100 features share 1 kernel, which contains 5×5 parameters. This is the parameter sharing mechanism of convolutional layer. Because of the enormous difference in the number of parameters, the learning ability of a single convolutional layer is certainly much weaker than that of a single fully-connected layer. However, as the network depth increases, the advantages of CNNs become apparent. The total number of layers of a CNN can be very large, and the parameters can grow at a manageable rate. In contrast, the depth of the MLP is severely limited. Moreover, the

parallel computing architecture of modern hardware is particularly well suited for training CNNs with parameter sharing mechanisms.

The large number of fully-connected layer's parameters also leads to a classic problem of machine learning, overfitting. In machine learning, overfitting is a phenomenon where a model is too precise for a particular data set, and it may fail with additional data. Overfitting is most intuitively observed by abnormally better performance on the training set than on the test set. The counterpart to overfitting is underfitting, which comes in the form of poor performance on both training and test sets. The fully-connected layer, because of its huge number of parameters, is highly susceptible to overfitting. This is because it has enough parameters to extract the noise and errors from the data set as features as well. On the other hand, the convolutional layer, with a smaller number of parameters and the parameter sharing mechanism, will be more inclined to extract features that are more general and representative. This is another advantage that a convolutional layer has over a fully-connected layer. It is important to note that CNN cannot completely eliminate overfitting. Overfitting has been a problem that the machine learning field has had to face for a long time. Some other approaches to overfitting will be presented in later sections of this chapter.

For all these reasons, convolutional layers are much more widely used than fully-connected layers. Early AlexNet still had three fully-connected layers at the end of the network, as shown in Figure 2.4. However, more advanced CNNs like GoogleNet have abandoned the use of fully-connected layers [50].

2.3 Improvements on convolution operation

The original convolutional operation is the foundation of modern CNNs and have always been widely used. In order to increase the flexibility of convolutional operations, some improved convolutional operations have been proposed. Two common types of improved convolution that are also used in this paper are described below. They are dilated convolution and deconvolution.

2.3.1 Dilated Convolution

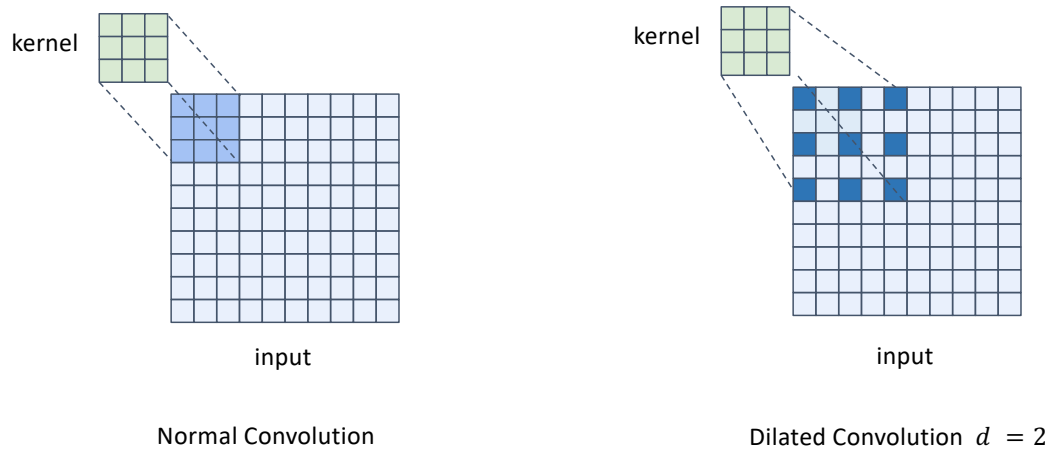


Fig. 2.6: Regular convolution *vs.* Dilated convolution

For a convolutional layer, there is an important index called the "receptive field", that refers to the area of the region of the original input which is involved to generate a feature in a specific layer. For regular convolution, the size of the receptive field of the first layer's output is the size of the kernel, and the higher the layer, the larger the receptive field of that layer's output. For example, if all layers' kernel is 3×3, the first layer's receptive field is 3×3, the second layer's receptive field is 5×5, and the nth layer's receptive field is $(2n + 1) \times (2n + 1)$. Ideally, the larger the receptive field is, the better the quality of the generated features. For regular convolution, the only way to increase the receptive field is to increase the size of the convolutional kernel. However, increasing the kernel area means that the amount of computation increases geometrically. This is usually unacceptable in practice with limited computing power. The dilated convolution is proposed to gain a larger receptive field but with the same kernel size [53]. Figure 2.6 shows a comparison of normal and dilated convolution. Dilated convolution contains a hyper-parameter d called "dilated rate". Instead of cropping contiguous rows and columns, dilated convolution crops the patch by picking one row or column every d rows or columns. When d equals 1, the

dilated convolution degenerates into a regular convolution. The right diagram of Figure 2.6 is a convolution operation with $d = 2$. It can be seen that the receptive field of an individual feature increases from 3×3 to 5×5 when the size of the convolutional kernel is kept at 3×3. If the input is data such as an image video, the change in value between two adjacent pixels will generally be small. Pixels that are ignored in this patch will also be involved in the generation of the next feature as the cropping window slides. So it is still worth to increase the receptive field even if some information are discarded in dilated convolution. But the dilated rate should not be set too large; in practice it is usually set to 2 or 3.

2.3.2 Deconvolution

From Equation 2.7, when $P \leq (K-1)/2$, the size of the output O is always less than $(S > 1)$ or equal $(S = 1)$ to the size of the input I. In many CNN applications, however, a convolution operation is needed that allows the output to be larger than the input. As an example, there is a very classical CNN framework called autoencoder [54]. Its basic structure is divided into two subnetworks, both consisting of convolutional layers. The first subnetwork, "encode subnetwork", takes the input, usually an image, and passes it through a series of convolutional layers to generate a n-length vector. The second subnetwork, "decode subnetwork", takes this vector and passes it through a series of convolutional layers to generate a new image that is as close as possible to the original image. The second subnetwork would require a convolutional layer with an output size larger than the input size. Of course, traditional upsampling methods such as nearest neighbor or linear interpolation can also be used here. However, they have at least two obvious flaws. The first is that they are not trainable, and the second is that decoding process is usually not a simple upsampling, so the results are not very good.

Deconvolution, or transposed convolution, was proposed for such needs [49]. It is not mean the same as the real "deconvolution" which is the inverse operation of the regular convolution in the field of signal processing. It can be seen as the backward operation of the regular convolution. This "backward" is in shape, not in value. For each forward regularity convolution operation, there is a corresponding backward deconvolution operation. Table

Table 2.1: Convolution *vs.* Deconvolution

Convolution		Deconvolution		
Stride	Constraint	Padding	Input	Output
$S = 1$	N/A	$P' = K - P - 1$	$I' = I$	$O' = I' + (K - 1) - 2P$
$S = 1$	$P = \frac{K-1}{2}$	$P' = P$	$I' = I$	$O' = I'$
$S = 1$	$P = K - 1$	$P' = 0$	$I' = I$	$O' = I' - (K - 1)$
$S > 1$	$(I + 2P - K) \mod S = 0$	$P' = K - P - 1$	$I' = (I - 1)(S - 1) + 1$	$O' = S(I' - 1) + K - 2P$
$S > 1$	N/A	$P' = K - P - 1$	$I' = (I - 1)(S - 1) + 1$ $A' = (I + 2P - K) \mod S$	$O' = S(I' - 1) + A' + K - 2P$

* K always equals to K'.
** S and S', although different in meaning, always have the same value.

2.1 shows the correspondence between them. Compared to regular convolution, there is only one more step to deconvolution. It is to first expand the input to a suitable shape. Then we can get an output that is larger than the input with regular convolution. The deconvolution is described by 6 hyper-parameters. They are input size I', output size O', stride S', kernel size K', padding size P', and additional row or column A'. In these 6 hyper-parameters, I', O', K' and P' maintain the same meaning as regular convolution. However, S' is no longer used to indicate the sliding step of the cropping window, which is always 1 in deconvolution. There are three cases for the expansion of the input. All the additional elements are 0. The first case is to add rows and columns around the input. The number of these rows and columns is defined by P' like regular convolution. S' is used to control the second case which is to add $S' - 1$ rows and columns between input unit. That is the case in the next to the second line from the bottom of Table 2.1. But this case is subject to the satisfaction of constraint $(I + 2P - K) \mod S = 0$. If this constraint is not satisfied, it is the third case, in which we need the last hyper-parameter A'. $'A'$ is used to determine the number of columns and rows to be added to the right and bottom edges of the input. Figure 2.7 shows all the 3 cases of deconvolution with different hyper-parameters.

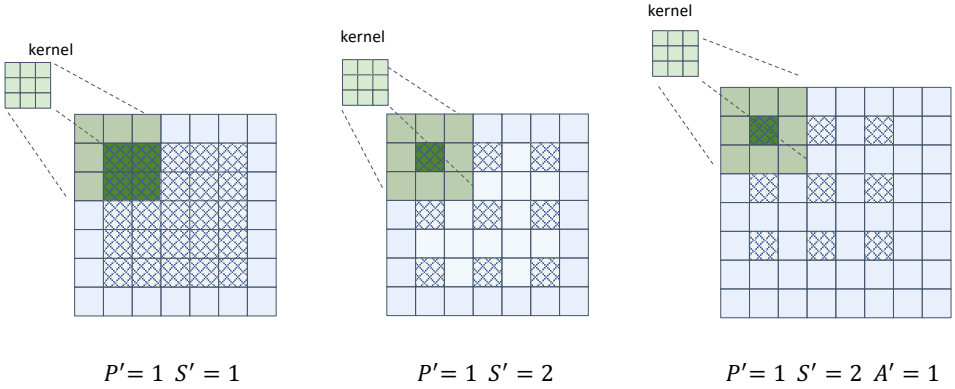


Fig. 2.7: Three types of deconvolution diagram, grids with crossing-line pattern are valid inputs, others are imaginary 0 elements

2.4 Activation Function

Equation 2.6 is a linear equation, so the convolutional layer and the fully-connected layer are linear. A linear combination of linear equations is still a linear equation. The cascade is exactly a linear combination. So, in the absence of an activation layer, a CNN, even if the layers are deeper, still only represents a giant linear equation. It can not fit any of the simplest, non-linear equations.

The activation layer is introduced to enable CNNs to have the ability to fit a nonlinear function. The term "activation function" comes from biology and refers to the rate of action potential firing in the cell [55]. In machine learning, an activation function is a class of nonlinear functions of simple shape and at least first order derivable. For deep learning, there are two additional requirements for activation functions. The first is monotonicity that ensures a single convolutional layer is convex [56]. The second is approximate identity near the origin, which is $f(0) = 0$, $f'(0) = 1$, and f' is continuous at the origin. If this condition is satisfied, the weights of kernel can be initialized with random small values, otherwise, the initialization needs a special care [57]. A single activation layer will not fit the nonlinear function very well, but when each convolutional layer is followed by another activation layer, then a multilayer CNN can theoretically fit any nonlinear function [58]. So the activation function significantly enlarges the representative ability of CNN. Choosing a

proper activation function is a critical factor in training success. In the following paragraphs, various activation functions are introduced. The activation function is applied on an input tensor pointwisely. We use a to represent the individual element of the input tensor and z to represent the corresponding individual element of the output tensor.

2.4.1 Hyperbolic tangent function

The hyperbolic tangent function (tanh) defined as

$$a = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}, \tag{2.8}$$

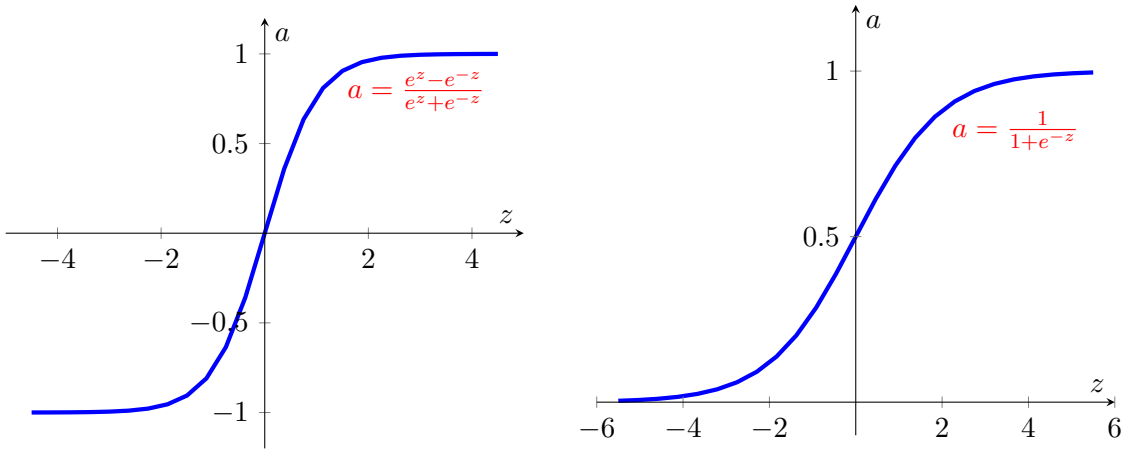
which is the most commonly used activation function for early neural networks. It has the following advantages. The tanh function is smoothly derivable over the entire domain. The tanh function and its derivative $\tanh'$ are bounded ($\tanh \in [-1, 1]$, $\tanh' \in [0, 1]$) over the entire domain. But a fatal property made it gradually become less popular. As shown in Figure 2.8a, when z moves away from the origin, the curve of the function becomes very flat, almost parallel to the x-axis. This means that its derivative is very close to zero. So the entire network cannot be updated, and convergence is very slow or even impossible. From another point of view, the tanh function requires that the outputs of the convolutional layer fall into a narrow interval close to the origin. This is very difficult to achieve in multi-layer networks. In some generative networks the final layer of some generative networks satisfies this condition because its output should be very close to the real data. In this case, the tanh function can be used as the activation layer.

2.4.2 Logistic function

Logistic function (sigmoid) is widely used in various areas of machine learning. It is given by

$$a = \text{sigmoid}(z) = \frac{1}{1 + e^{-z}}. \tag{2.9}$$

As shown in Figure 2.8b, it is not approximate identity near the origin. So as an



(a) Hyperbolic tangent function (tanh) (b) Logistic function (sigmoid)

Fig. 2.8: Hyperbolic tangent function and Logistic function

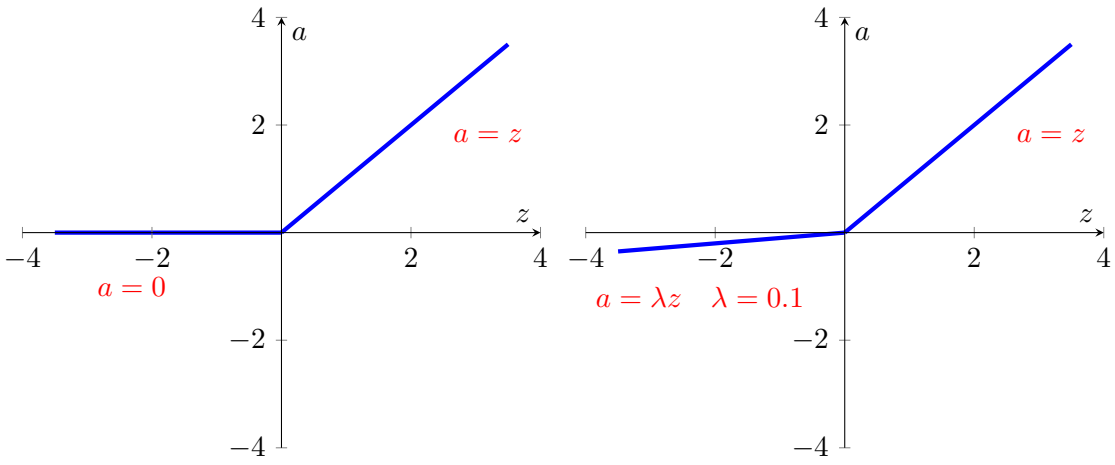
activation function, it is not suitable to be applied in hidden layers. This would cause the values of all parameters in the network to shift in one direction. It is typically used as the last layer of the entire network to limit the output to the interval 0 to 1.

2.4.3 Rectified linear unit

Rectified linear unit (ReLU) is the most notable non-saturated activation function and the basis for a series of modern activation functions [59]. It is defined as

$$a = \max(z, 0). \tag{2.10}$$

As shown in Figure 2.9a, the ReLU function acts like a circuit component. When the input value is less than 0, it is in a shorted state and the output is 0. When the input value is greater than 0, it is in a path state and does not change the input value. The ReLU function and its gradient are simple to compute compared to the old-school activation function, e.g., tanh. Many research papers show that it is better than tanh and sigmoid empirically. So it has become the most popular activation function in recent years.



(a) Rectified linear unit (ReLU)

(b) Leaky ReLU

Fig. 2.9: Rectified linear unit (ReLU) and Leaky ReLU

2.4.4 Leaky ReLU

One defect of ReLU is that if the input value is less than 0, the gradient is constant 0, which means it has no contribution to the training process. A 0 gradient also causes a slow learning process. The Leaky ReLU function was introduced to alleviate this problem [60]. The Leaky ReLU function is given by

$$a = \max(z, 0) + \lambda \min(z, 0), \tag{2.11}$$

where λ is a predefined hyper-parameter that should be very small (usually $\lambda = 0.01$ or $\lambda = 0.1$). The Leaky ReLU significantly compresses the negative value instead of setting it to 0. So the negative values still contribute to training weights. Figure 2.9b shows the curve of the Leaky ReLU function.

2.4.5 Parametric ReLU

Parametric ReLU (PReLU) is very similar to Leaky ReLU. It is defined as

$$a = \max(z, 0) + \lambda_k \min(z, 0). \tag{2.12}$$

The only difference is the parameter λ_k is not predefined but a learned parameter. For feature map k, we can assign a λ_k to it. The Parametric ReLU only introduces a small number of parameters but has a better performance to overcome overfitting.

2.4.6 Randomized ReLU

Another variant of Leaky ReLU is Randomized ReLU (RReLU) [61]. It is defined as

$$a^{(n)} = \max\left(z^{(n)}, 0\right) + \lambda^{(n)} \min\left(z^{(n)}, 0\right), \tag{2.13}$$

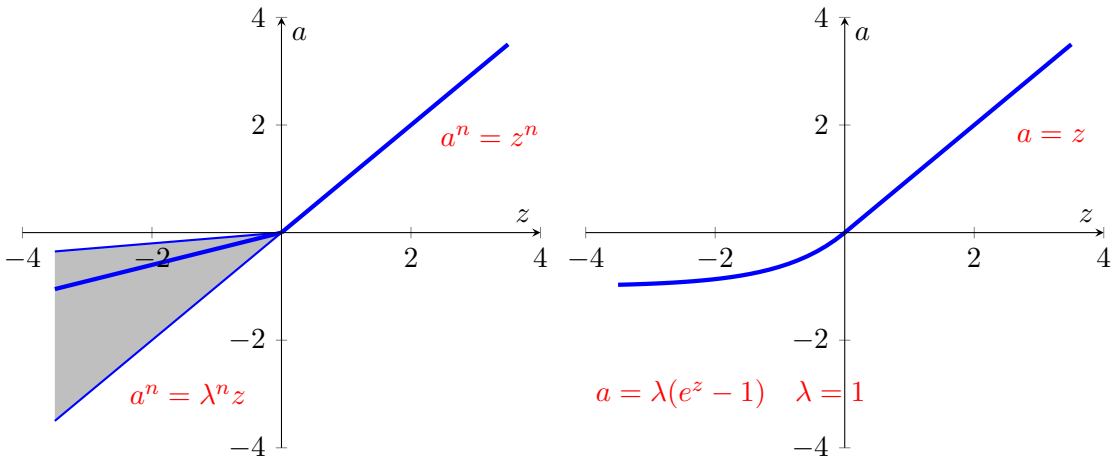
and its shape is shown as Figure 2.10a. The λ_k is randomly sampled from a uniform distribution in training and fixed in testing. Like PReLU, for each single input $a^{(n)}_{i,j,k}$, there is an independent $\lambda^{(n)}$.

2.4.7 Exponential Linear unit

There is a minor problem with these ReLU-based functions above, they have a mean value greater than 0 over the entire domain. This decrease the convergence speed in the training of large neural networks. Another problem is that the negative part or deactivation part of Leaky ReLU, RReLU and PReLU is not noise-robust. Exponential Linear Unit (ELU) introduces a saturation function as a negative part to alleviate the two problem [62]. It is defined as

$$a = \max(z, 0) + \min(\lambda(e^z - 1), 0). \tag{2.14}$$

As shown in Figure 2.10b. The saturation function on the negative part push the mean close to 0 that can lead to a faster learning speed. It also saturates the negative noise to



(a) Randomized ReLU (RReLU) (b) Exponential Linear unit (ELU)

Fig. 2.10: Randomized ReLU (RReLU) and Exponential Linear unit (ELU)

decrease the variation of the noise. The parameter λ can be any positive number, and if $\lambda = 1$, ELU is approximately the identity near the origin.

2.4.8 Softmax Function

The softmax function, also known as normalized exponential function, behaves differently than the above activation functions. The activated value a_i for an element z_i is not independent, but depends on all the values in the vector $\boldsymbol{z}$. it is given by

$$a_i = \sigma(\boldsymbol{z})_i = \text{softmax}(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}} \tag{2.15}$$

Its primary property is to normalize an arbitrary vector to a probability distribution. So it is often used as the last layer of an object recognition network to indicate the probability that an object belongs to a certain category.

These are some of the common activation functions that are introduced. New activation functions have still been proposed in recent years, such as Gaussian error linear unit (GELU) and scaled exponential linear unit (SELU) [63, 64]. The proponents claim that they have better performance. However, these new activation functions are too complex. SELU requires the whole network to be adapted to it. GELU are proposed for domain specific

learning and they perform well in natural language processing. So it will still take time to see if these new activation functions can be used on a large scale.

2.5 Optimization

For supervised learning, the most common learning process is to compare the output of the model to ground truth and to represent the comparison as an error function. For example, for target recognition, the error function is usually the cross-entropy function, and for image denoising, the error function is usually the root mean square error (RMSE) function. These error functions are also called loss functions. For complex machine learning tasks, a single loss function may not work well, and multiple loss functions with different weights need to be combined together. There are also times when we need to place some regularizations on the parameters of the model, such as the range of values and sparsity of the parameters. These regularizations will also join with the loss functions to form a more general function, the objective function. Our goal is to adjust the parameters of the model to continuously reduce the value of the objective function through mathematical methods. This whole process above is called optimization in machine learning. Common mathematical methods are gradient descent and Newton's method. The Newton's method requires that the object function and the model's function to be second-order derivable, which is often difficult to guarantee in deep learning due to the complexity of the model. Therefore, common deep learning optimization methods are based on gradient descent. In this section, some optimization methods will be introduced.

2.5.1 Stochastic Gradient Descent

Back propagation uses the gradient descent method to update the weight matrices, bias vectors and other trainable parameters. Standard gradient descent updates the parameter θ through loss function $\mathcal{L}(\theta)$ as $\theta_{t+1} = \theta_t - \eta \nabla_\theta E\left[\mathcal{L}\left(\theta_t\right)\right]$ where $E\left[\mathcal{L}\left(\theta_t\right)\right]$ is the expectation of $\mathcal{L}(\theta)$ over the whole training set. The learning rate η is a hyper-parameter set by the designer. Stochastic Gradient Descent (SGD) estimates the gradients on the basis of a single randomly picked example $(x^{(t)}, y^{(t)})$ from the training set [65]. The original SGD is

defined as

$$\begin{aligned} g_t &= \nabla_\theta \mathcal{L}\left(\theta_t; x^{(t)}, y^{(t)}\right) \\ \theta_{t+1} &= \theta_t - \eta_t g_t \end{aligned} . \tag{2.16}$$

In practice, SGD is computed with respect to a mini-batch as opposed to a single example. This can lead to a stable convergence and a more reliable result. It is difficult to find the best learning rate. A fixed learning rate is usually not optimal because there are different requirements for learning rate at different stages of learning. Momentum is a widely used method to dynamically adjust the learning rate [66]. The momentum is given by

$$\begin{aligned} v_{t+1} &= \gamma v_t - \eta_t g(t) \\ \theta_{t+1} &= \theta_t + v_{t+1} \end{aligned} . \tag{2.17}$$

There are many momentum-based optimization algorithms, and the one used in this dissertation is the Adam algorithm [12]. The steps are as follows:

$$\begin{aligned} m_t &= \mu m_{t-1} + (1-\beta_1) g_t, \\ n_t &= \nu n_{t-1} + (1-\beta_2) g_t^2, \\ \hat{m}_t &= \tfrac{m_t}{1-\beta_1^t}, \\ \hat{n}_t &= \tfrac{n_t}{1-\beta_2^t}, \\ \Delta\theta_t &= -\tfrac{\hat{m}_t}{\sqrt{\hat{n}_t+\epsilon}}\eta, \end{aligned} \tag{2.18}$$

where θ_t is adjusted by a series of moment estimates, and m_t, n_t are first- and second-order moment estimates of the gradient, respectively, which can be thought of as estimates of the expectations $E|g_t|$, $E|g_t^2|$. Both moments are corrected to $\hat{m}_t$, $\hat{n}_t$ so that they can be approximated as unbiased estimates of the expectations. The learning rate need a dynamic constraint $-\frac{\hat{m}_t}{\sqrt{\hat{n}_t+\epsilon}}$ to have a limited range.

2.6 Regularization

Modern CNN have millions of parameters, so the overfitting phenomenon happens easily. Regularization is necessary to overcome this problem. The most common regularization methods are l_p norm, Dropout, and DropConnect.

2.6.1 l_p norm

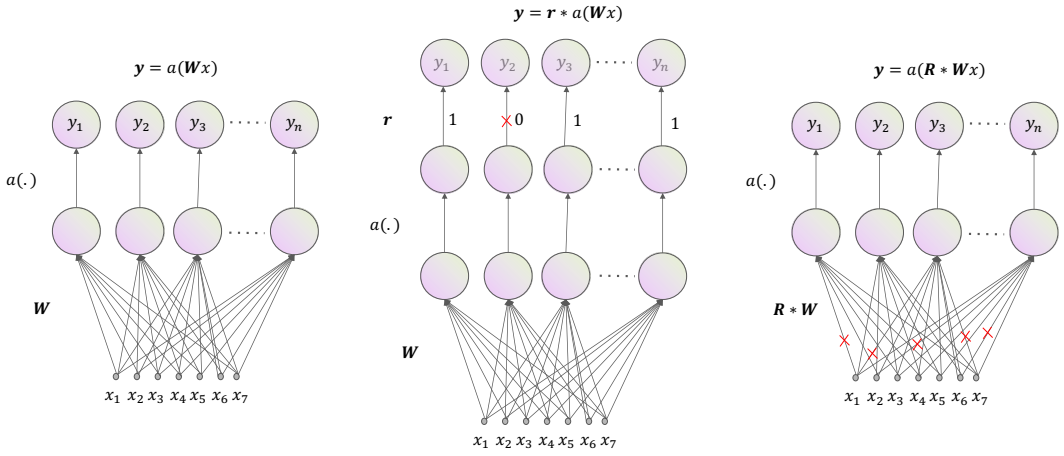
l_p norm adds a regularization term to the loss function. If the loss function is $\mathcal{L}(\theta, \boldsymbol{x}, \boldsymbol{y})$, the regularized loss function can be written as

$$E(\theta, \mathbf{x}, \mathbf{y}) = \mathcal{L}(\theta, \mathbf{x}, \mathbf{y}) + \lambda R(\theta) \tag{2.19}$$

where λ is the regularization strength. $R(\theta)$ is usually employed as $R(\theta) = \sum_j \|\theta_j\|_p^p$. When $p \geq 1$, the regularization term is a convex function. The special case when $p = 2$ is often called weight decay. The goal of $R(\theta)$ is to penalize the complexity of the model.

2.6.2 Dropout

Dropout is a simple regularization method [67]. However, empirically it performs very well on nearly all deep learning tasks and becomes a standard operation in training. In the training process, dropout randomly abandons a portion of the neurons in the middle layers and set them to 0. It prevents the network from becoming too dependent on one (or any small combination) of neurons. Although some information may have been lost, it greatly enhances the generalizability of the network. During training, the uncertainty of missing neurons stimulates the network to generate as many more generalizable features as possible without extracting some of the noise from the data as features.



(a) No-Drop (b) DropOut (c) DropConnect

Fig. 2.11: The comparison Among No-Drop, DropOut, and DropConnect

2.6.3 DropConnect

DropConnect goes further than Dropout. Instead of setting the neurons to 0, DropConnect set a portion of trained weights in each layer to 0 [68]. Figure 2.11 shows the difference between No-Drop, DropOut, DropConnect.

2.6.4 Batch Normalization

In the machine learning field, data normalization is usually the default process step at the beginning. However, the datasets used by deep learning are often very huge. Normalizing the whole dataset is unrealistic. When data flow into the processor, the distribution of the input will be changed. Batch Normalization is proposed to alleviate this problem [69], which normalizes the dataset on the batch size instead of the whole dataset. Suppose the input data is k-dimension. The data is first normalized using

$$\hat{x}_k = (x_k - \mu_{\mathcal{B}}) / \sqrt{\delta_{\mathcal{B}}^2 + \epsilon} \tag{2.20}$$

where $\mu_{\mathcal{B}}$ and $\delta_{\mathcal{B}}$ are the mean and variance of the mini-batch respectively. The parameter ϵ is a very small value to increase numeric stability. To enhance the representative ability,

the $\hat{x}_k$ further transforms into

$$y_k = \gamma \hat{x}_k + \beta \tag{2.21}$$

where γ and β are trained parameters.

2.7 Generative adversarial network

In 2014, Goodfellow et al. presented the vanilla Generative Adversarial Network (GAN) and led the deep learning community into a new era [70]. It has become one of the most active and prosperous research field in recent years. In most cases, a GAN consists of two sub-networks, a generator (G), and a discriminator (D). Either can be an existing or emerging deep learning model, and usually, the output of G is some artificial data like a camera picture, art paintings, video, or human speech, etc. These data, along with their counterparts gathered from the real world, are sent to D, and the discriminator provides a single scalar to judge that the input data is from real data or generator. During the training period, the generator maximizes its ability to create as realistic data as possible, and the discriminator identifies the reality of input example as accurately as possible. The G and D are trained simultaneously. The performance of G will influence D, and vice versa, and this process can be seen as a contest between the G and D. In every training iteration, the generator improves the quality of its output to confuse the discriminator, and the discriminator also needs to improve its ability to distinguish the generated samples from real samples. This competition is the origin of the critical concept "adversarial". The training process ends at a balanced status that both G and D can not be optimized anymore through a gradient-descent based method, and the output of D oscillates around 0.5 (if it is normalized), which means the discriminator can not distinguish generated data from real data. It has been proven that such a balance status can be achieved [70].

Both supervised learning and unsupervised learning benefit enormously from GAN since GAN can be seamlessly adopted in any existing framework. For supervised learning, the discriminator can be seen as a brand new evaluation criterion that does not rely on ex-

isting human knowledge. For unsupervised learning, which GAN made the most remarkable achievement, the discriminator becomes the supervisor and transfers unsupervised learning to supervised learning. The Figure 2.12 shows some artificial human faces generated from the latest NVIDIA styleGAN [71]. It is difficult to tell the difference between these vivid computer-generated pictures and real human faces.

Fig. 2.12: Human faces generated by styleGAN [71]

Since 2014, GAN continues to evolve to become more powerful and complicated. In the following section, a brief introduction will go through several aspects of GAN, including basic algorithms, training, and theory behind GAN, and a few typical applications.

2.7.1 Basic algorithms

Training a GAN is a min-max game to learn the distribution $p_{data}(x)$ of the real data. The generator $G(z, \theta_g)$, which is defined by its parameters θ_g and driven by a noise variable z, approximates the distribution $p_g(x)$ of its output to $p_{\text{data}}(x)$ through optimizing the object function. Different GANs usually use different object functions. The original object function adopted by the vanilla GAN [70] can be expressed as

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[\log D(x)] + E_{z\sim p_z(z)}[\log(1 - D(G(z)))]. \tag{2.22}$$

It has been proved that the global minimum is reached when $D^*_G(x)$ is given by

$$D^*_G(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)}. \tag{2.23}$$

If we put $D^*_G(x)$ back into Equation 2.22 and reformulate it as

$$\begin{aligned} \max_D V(D,G) = E_{x\sim p_{\text{data}}} &\left[\log \frac{p_{\text{data}}(x)}{\frac{1}{2}(p_{\text{data}}(x) + p_g(x))}\right] \\ + E_{x\sim p_g} &\left[\log \frac{p_g(x)}{\frac{1}{2}(p_{\text{data}}(x) + p_g(x))}\right] - 2\log 2, \end{aligned} \tag{2.24}$$

this is the convergence point of GAN. We can compare it with the definition of Kullback-Leibler (KL) divergence and Jensen-Shannon (JS) divergence between two random distributions $p(x)$ and $q(x)$ given by

$$KL(p\|\, q) = \int p(x) \log \frac{p(x)}{q(x)} dx \tag{2.25}$$

$$JS(p\|\, q) = \frac{1}{2} KL(p\|\, \frac{p+q}{2}) + \frac{1}{2} KL(q\|\, \frac{p+q}{2}). \tag{2.26}$$

Equation 2.24 is equal to

$$\begin{aligned}\max_D V(D,G) &= KL(p_{\text{data}} \| \frac{p_{\text{data}}+p_g}{2}) + KL(p_g \| \frac{p_{\text{data}}+p_g}{2}) \\ &= 2JS(p_{\text{data}} \| p_g) - 2\log 2.\end{aligned} \tag{2.27}$$

Early GANs suffered from the uncertainty of convergence and convergence speed. Goodfellow et al. provided a "trick" method that rather than minimizing $\log(1-D(G(z)))$, maximizing $\log(D(G(z)))$ results in the same fixed point but has a larger gradient in the early training period. However, this new object function brings new problems such as an unstable numerical gradient. If we use the definition of KL divergence and JS divergence and substitute $D(G(z))$ with $D^*_G(x)$ in $-\log(D(G(z)))$ (maximizing to minmizing), $-\log(D^*_G(x))$ is transformed to:

$$\begin{aligned}E_{x\sim p_g}\left[-\log\left(D^*_G(x)\right)\right] &= KL\left(p_g \| p_{\text{data}}\right) - 2JS\left(p_{\text{data}} \| p_g\right) \\ &\quad + E_{x\sim p_{\text{data}}}\left[\log D^*_G(x)\right] + 2\log 2\end{aligned} \tag{2.28}$$

Notice that the last two terms are irrelevant to the generator, and the first two terms reveal the inner contradiction of this object function. The positive KL-divergence requires the generator to pull its distribution close to the real data distribution, but the negative JS-divergence enlarges the two distributions' gap. This contradiction is the root of the unstable gradient problem. Furthermore, the KL-divergence is an asymmetric quantity, so the two extreme cases will result in totally different loss measurement.

- $KL(p_g \| p_{\text{data}}) = 0$ when $p_g \to 0$ and $p_{\text{data}} \to 1$,
- $KL(p_g \| p_{\text{data}}) = +\infty$ when $p_g \to 1$ and $p_{\text{data}} \to 0$.

The first case, $p_g \to 0$ and $p_{\text{data}} \to 1$, the generator produces images that are not similar to real images in the dataset even if they look good. On the contrary, when $p_g \to 1$ and $p_{\text{data}} \to 0$, the generator creates unplausible images that are very "unnatural". Since the two cases' penalty is profoundly unequal, the generator inclines to produce reduplicative but reliable examples instead of diversified but unreliable examples. This phenomenon

frequently happens and is called "model collapse" [72, 73], which is that the generator produces very few modes of data and ignores other possible modes. In some extreme cases, only 1 example is created [74,75]. From Equation 2.28, it is obvious that model collapse can not be mitigated by tuning the structure of G and D. How to avoid model collapse is the top issue of the vanilla GAN's modern successors. Table 2.2 lists loss functions of several significant descendants. Among them, Wasserstein GAN is worth discussing in detail [76].

Table 2.2: Loss functions of various GANs

GAN types	discriminator Loss	Generator Loss
Vanilla GAN	$-E[\log(D^{\dagger}(x))] - E[\log(1 - D^{\dagger}(G(z)))]$*	$-E[\log(D^{\dagger}(G(z)))]$
LS-GAN	$E[(D(x) - 1)^2] - E[D(G(z))^2]$	$E[D(G(z))^2 - 1]$
Wasserstein GAN	$E[D(x)] - E[D(G(z))]$	$[D(G(z))]$
COS-GAN	$E[\cos([D(x) - 1)] - E[D(G(z) + 1)]$	$-E[D(G(z) - 1)]$
EXP-GAN	$E[\exp(D(x))] + E[\exp(-D(G(z)))]$	$E[\exp(D(G(z)))]$
HINGE-GAN	$E[\min(0, D(x) - 1)] + E[\min(0, -D(G(z) - 1)]$	$-E[D(G(z)]$
EB-GAN	$E[D(x)] + E[\max(0, m - D(G(z)))]$	$E[D(G(z))]$
MD-GAN	$-E[\log D(G(z))]$ $+ E[\lambda_1 d(x, \mathbb{E}(G(x))) + \lambda_2 \log D(\mathbb{E}(G(x)))]$**	$E[\lambda_1 d(x, \mathbb{E}(G(x)))$ $+ \lambda_2 \log D(\mathbb{E}(G(x)))]$

* $D^{\dagger}(\cdot)$ equals to sigmoid$(D(\cdot))$

** $\mathbb{E}(\cdot)$ means the output of the encoder in EB-GAN

2.7.2 Wasserstein GAN

Before the Wasserstein GAN is presented, results from Arjovsky and Bottou are presented to give theoretical steps to solve practical problems of GAN [74]. Besides the model collapse problem, they pointed out another theoretical pitfall in the training period that repeatedly happens in various GAN practices. Suppose two independent random variables

$P_r(x)$ and $P_g(x)$. Based on whether their value is 0, their JS-divergence have four possible cases.

- $P_r(x) = 0$ and $P_g(x) = 0$, this is a trivial case and the JS-divergence is meaningless,
- $P_r(x) \neq 0$ and $P_g(x) \neq 0$, the JS-divergence is a differentiable function,
- $P_r(x) = 0$ and $P_g(x) \neq 0$, the JS-divergence is a constant $\log 2$,
- $P_r(x) \neq 0$ and $P_g(x) = 0$, the JS-divergence is also $\log 2$.

For cases 3 and 4, the constant JS-divergence results in a 0 gradient, which means the generator can not receive any helpful information from the discriminator to improve its performance.

To overcome the JS-divergence defect, Arjovsky and Bottou proposed the Wasserstein distance, also known as Earth-Move distance [76]. The Wasserstein distance is defined as

$$W(P_r, P_g) = \inf_{\gamma \sim \Pi(P_r, P_g)} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|] \tag{2.29}$$

where $\Pi(P_r, P_g)$ is the set which includes all joint distributions of P_r and P_g. In another words, every marginal distribution of any distribution in Π is either P_r or P_g. A real example x and a generated example y can be obtained through sampling γ. The infimum of the expectation of the distance between x and y is the Wasserstein distance. It can be explained in a straightforward way that $\mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|]$ is energy consumed to move "a pile of earth" P_g to the position of P_r under a path planning γ . Then $W(P_r, P_g)$ is the minimum energy consumed when the path planning is optimal. That is why it is also called the "Earth-Move distance".

The Wasserstein distance's distinct advantage is that even if two distributions do not have an overlap area, the Wasserstein distance is still an excellent metric to reflect the gap between them. Take a very simple case as an instance, in Figure 2.13, p_1 and p_2 are two 1-dimensional uniform distributions in a 2-dimensional space with the same support length.

A hyperparameter θ is set to control the distance between them. The KL-divergence, JS-divergence, and Wasserstein distance are given as

$$KL\left(\mathbb{P}_0, \mathbb{P}_\theta\right) = \begin{cases} +\infty & \text{if } \theta \neq 0 \\ 0 & \text{if } \theta = 0 \end{cases} \tag{2.30a}$$

$$JS\left(\mathbb{P}_0, \mathbb{P}_\theta\right) = \begin{cases} \log 2 & \text{if } \theta \neq 0 \\ 0 & \text{if } \theta = 0 \end{cases} \tag{2.30b}$$

$$EM\left(\mathbb{P}_0, \mathbb{P}_\theta\right) = |\theta|. \tag{2.30c}$$

The two divergences are polarized and have a 0 gradient everywhere except when θ equals 0. So any optimization method based on the gradient-descent algorithm does not work. However, the Wasserstein distance can still provide a smooth distance with a constant gradient. Similarly, if two distributions have no overlap area or the overlap part is negligible in high dimensional space, neither the KL-divergence nor the JS-divergence can present a meaningful metric of distance and a non-zero gradient, but the Wasserstein distance can. That is the most attractive property of the Wasserstein distance.

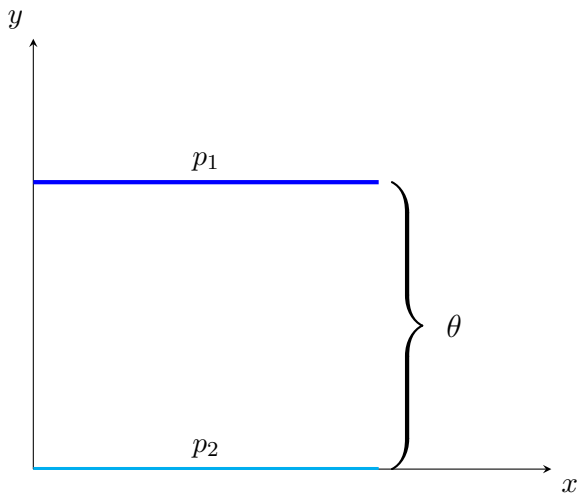


Fig. 2.13: A simple comparison Among KL-divergence, JS-divergence, and EM-distance

There are a few more steps of deduction from the Wasserstein distance to Wasserstein GAN (WGAN). First, it is nearly impossible to obtain the solution of $\inf_{\gamma \sim \Pi(P_r, P_g)}$. A transformation of Equation 2.29 is

$$W\left(P_r, P_g\right) = \frac{1}{K} \sup_{\|f\|_L \leq K} \mathbb{E}_{x \sim P_r}[f(x)] - \mathbb{E}_{x \sim P_g}[f(x)]. \tag{2.31}$$

The restriction $\|f\|_L \leq K$ is known as Lipschitz regularity which requires any two points x_1 and x_2 in the field of function f have the relationship

$$\left|f\left(x_1\right) - f\left(x_2\right)\right| \leq K\left|x_1 - x_2\right|.$$

If we use a group of parameters w to define f, an approximate solution of Equation 2.31 is

$$K \cdot W\left(P_r, P_g\right) \approx \max_{w:\, f_w|_L \leq K} \mathbb{E}_{x \sim P_r}\left[f_w(x)\right] - \mathbb{E}_{x \sim P_g}\left[f_w(x)\right]. \tag{2.32}$$

The function f_w can be fitted by any neural network, so the original form of the Wasserstein GAN is represented as

$$L = \mathbb{E}_{x \sim P_r}\left[f_w(x)\right] - \mathbb{E}_{x \sim P_g}\left[f_w(x)\right]. \tag{2.33}$$

The discriminator's goal is to enlarge the distance L of the real distribution and the generated distribution. On the contrary, the goal of the generator is to reduce L. Since the first term of Equation 2.33 is independent of P_g, the two networks' loss functions are

$$L_G = -\mathbb{E}_{x \sim P_g}\left[f_w(x)\right] \qquad \text{Generator} \tag{2.34a}$$

$$L_D = \mathbb{E}_{x \sim P_g}\left[f_w(x)\right] - \mathbb{E}_{x \sim P_r}\left[f_w(x)\right] \qquad \text{Discriminator.} \tag{2.34b}$$

It is a challenge to implement Lipschitz regularity on a neural network that usually contains more than tens of thousands of parameters. In the original paper, Arjovsky and Bottou [76] adopt an extremely simple method to limit the amplitude of variation of f_w, which is to clip all parameters in f_w in a very narrow range such that $w_i \in (-c, c), c = 0.01$ after back backpropagation for each iteration. in this situation, we have great confidence that the gradient with input x, $\frac{\partial f_w}{\partial x}$, would not be too large and the local variation of f_w is less than an unknown constant K of irrelevant specific value.

The clip method is an empirical idea and has no theoretical grounding. Researchers have reported several defects of the clip method, e.g., slowly convergence and failed training. The authors admitted these shortcomings. They proposed some observations to analyze them and an improved method called "gradient penalty" [77]. They have found two significant drawbacks. First of all, the goal of the discriminator is to widen the gap between real examples and generated examples as much as possible. However, the weight clipping method independently limits the range of all w_i. So the optimal strategy may push all the parameters to the peak value c or $-c$. Experimental results validate this hypothesis that all weights concentrated on both edges of the clipping range. Under these circumstances, the whole network can be seen as a binary network; that is a huge waste of a neural network's powerful fitting ability. The generator also suffered from it because the gradient that the "binary" discriminator sent back is not as useful as a well-trained discriminator.

The other problem is the old common problem of the neural network, vanishing gradient, or exploding gradient. It is also caused by the range value c. The whole network is very sensitive to the value of c. A small digit change would result in that each layer's gradient norm either grows or decays exponentially. It is very hard to find a single value of c that can make the training process stable. On the other hand, assigning a different c to tens or hundreds of different layers is far more unrealistic. A worse thing is that even if a suitable c is found, the network needs to be tuned again when its structure is partially changed.

After reviewing the weight clipping method, Gulrajani et al. proposed a brand new

innovative method to apply Lipschitz regularity called Gradient penalty (GP) [77]. Lipschitz regularity requires the norm of the gradient of f_w is less than K. However, we also do not want the gradient is too small, which will result in extremely slow convergence. So the ideal state is that the norm would stay around K. It is then straightforward that we can add another loss to Equation 2.34b to penalize a gradient that is far away from K. Now L_{GP} become

$$L_{GP} = \left[\|\nabla_x D(x)\|_p - K\right]^2 . \tag{2.35}$$

If we add L_{GP} to Equation 2.34b with a weight λ and simply set K to 1, we get a new version of L_D which is written as

$$L_D = \mathbb{E}_{x\sim P_g}\left[D(x)\right] - \mathbb{E}_{x\sim P_r}\left[D(x)\right] + \lambda \mathbb{E}_{x\sim \Pi}\left[\|\nabla_x D(x)\|_p - 1\right]^2 . \tag{2.36}$$

In Equation 2.36, all the three terms are expectation form. When implemented into a practical neural network, they must be transformed to sample form. Sampling from real image space and generated image space is trivial. Sampling from the joint distribution space seems impossible. The authors discovered a smart way to simplify this operation. We have no interest in most of Π but the real image space, generated image space, and the space between them. Instead of sampling images from the entire space, we only focus on sampling the 3 subspaces mentioned before by linear interpolating real images and generated images. Suppose $x_r \sim P_r$ and $x_g \sim P_g$, a sample $\hat{x}$ from the intermediate space $\P_{\hat{x}}$ is obtained by

$$\hat{x} = \epsilon x_r + (1-\epsilon)\, x_g \qquad \epsilon \in [0,1] , \tag{2.37}$$

and the final version of L_D is given by

$$L_D = \mathbb{E}_{x\sim P_g}\left[D(x)\right] - \mathbb{E}_{x\sim P_r}\left[D(x)\right] + \lambda \mathbb{E}_{x\sim P_{\hat{x}}}\left[\|\nabla_x D(x)\|_p - 1\right]^2 . \tag{2.38}$$

WGAN have two major differences compared with orginal GAN.

- Remove the last Sigmoid layer of the discriminator;

- Use the raw output of the discriminator as a loss value rather than the logarithm to the output.

Gulrajani et al. believe that the Wasserstein GAN with the gradient penalty is the Wasserstein GAN's best practice. This conclusion is validated by many successors [75,78–81]

CHAPTER 3

IMAGE GUIDED DEPTH IMAGE UPSAMPLING

3.1 Introduction

Convolutional Neural Networks (CNNs) immensely impact Deep Learning as a fundamental tool for handling arranged data, like image, video, audio, etc. CNNs have gained enormous success in the image processing and computer vision area, including object recognition, semantic segmentation, etc. Many public datasets are presented for researchers to evaluate their frameworks. Numerous advanced methods become rudimentary knowledge in a short period. However, how to use CNNs to solve other scientific and, moreover, engineering tasks are still a case-by-case problem.

In recent years, inexpensive 3D cameras are becoming common in public. A consumer-level time of flight (TOF) camera has a resolution of around 200, e.g. 200×200 for the PMD CamCube 3.0 or 144×176 for the SwissRanger SR-4000. High-resolution (HR) 3D cameras are much more expensive and usually not real-time. For example, a Matterport Pro2 3D camera, which costs more than 3000 dollars, needs 20 seconds per scan and 11 seconds to process the raw data. A compromise method uses an low-resolution (LR) camera to collect raw data and upsampling it to a higher resolution. A popular way to upsample an LR depth image is to leverage ancillary data. A color image captured from the same scene is a common choice for ancillary data.

An assembled device called a texel camera that can facilitate the process of capturing both a color image and a depth image is introduced. The texel camera collects a specific type of image called texel Image [3]. The goal of this dissertation is to propose a novel CNN framework called uvCNN to upsample an LR 3D depth image with the color image's guidance captured at the same viewpoint. The texel image satisfies this requirement. The uvCNN does not merely exploit a CNN as a solution to an existing problem, but also

illustrate the problem that CNNs find it challenging to learn the information hidden in a coordinate system. We explore this problem and present our solution. The idea behind the uvCNN is to provide more detailed information to the CNN. Ancillary data not only improves the quality of training, but sometimes is even the key to its success.

In the following sections, we will first introduce two common types of improved convolutional blocks, residual block and dense block. Then the structure of uvCNN is presented. After that we will discuss how to generate synthetic texel image from public 3D dataset. The last section is the experiment detail and the result.

3.2 Residual Unit and Dense Unit

We did not use the original convolutional module as the basic module for uvCNN because many improved modules have been proposed in the last few years. Numerous research results have also shown that they do outperform the original convolutional modules in terms of performance. We have chosen two types of these modules, residual block and dense block, as the base modules of our network.

3.2.1 Residual Network

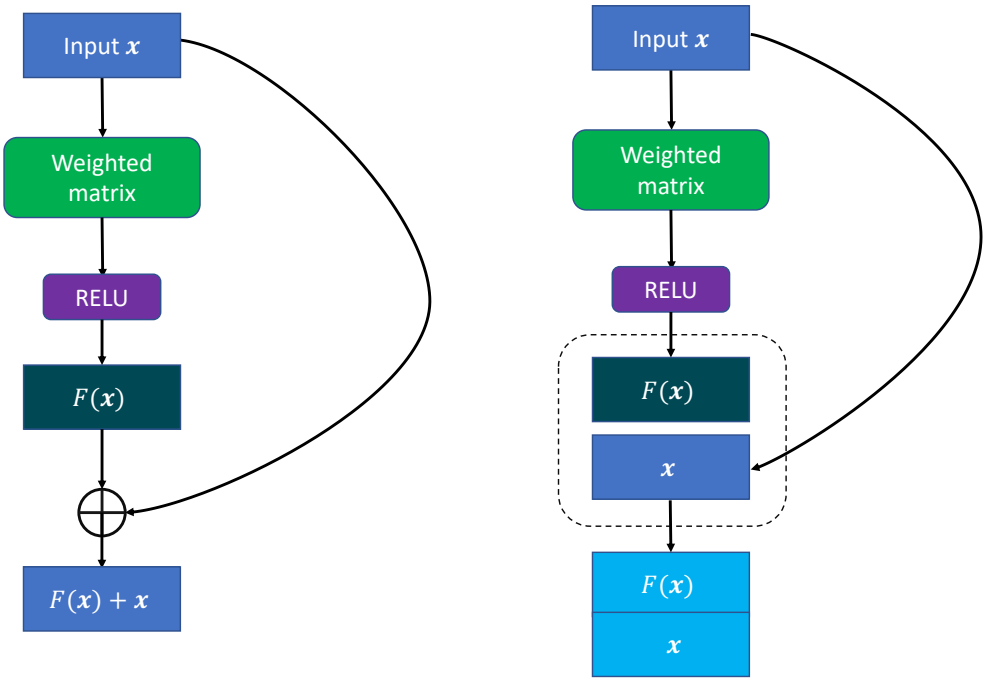
The traditional CNN suffers from vanishing or exploding gradients, overfitting, and error reincreasing. Many approaches are proposed to mitigate these problems. He et al. present a remarkable solution named ResNet [52]. In a classic CNN, a typical convolutional unit includes a set of input features $\boldsymbol{x}$, a convolutional kernel $W\boldsymbol{x}+\boldsymbol{b}$, a nonlinear activation layer(RelU, softmax), some regularization layers(batch normalization, pooling, dropout), and a set of output features $\boldsymbol{y}$. As the number of layers increases, the gradients of higher layers may become very small and cannot impact the backpropagation process. In ResNet, a shortcut is added to the convolutional unit to connect the input and the output. If the convolutional operation and all the regularize operations are treated as a whole nonlinear function F, then the classic mapping can be defined as $\boldsymbol{y} = F(\boldsymbol{x})$. The ResNet unit is defined as $\boldsymbol{y} = F(\boldsymbol{x})+\boldsymbol{x}$. The gradient of the unit is $\boldsymbol{y}' = F'(\boldsymbol{x})+\mathbf{1}$. This gradient oscillates around 1 when the norm of $F'(\boldsymbol{x})$ is very small, thus effectively avoiding the problem of vanishing

gradient. Because of the addition operation, the inputs and the outputs should have the same shape (width, height, and depth), or an extra layer is needed to align the shape. This requirement limits the options of some hyper-parameters. This method has shown to be efficient in color image processing. So it is adopted in the subnetwork of EO image processing. Figure 3.1a is the structure of a single residual block.

3.2.2 Densely Connected Convolutional Network

Another approach made to improve CNN is called the DenseNet [82]. The DenseNet preserves the input features differently. The inputs and outputs are stacked together after the nonlinear activation. If the input features have m channels and the output features of a classic unit have n channels, then the output features of a DenseNet unit has $m + n$ channels. Compared with ResNet, DensNet does not require $m = n$. In the original DenseNet paper [82], the first layer is a standard convolutional layer to shrink the input size. In our uvCNN, the initial LR depth image is preserved through the whole subnetwork.

It is not just empirical to adopt different structures in the two subnetworks. There are various ways to represent a pixel in EO images, such as RGB or HSL. The numeric value have no real physical meaning. Researchers focus on the correlation between adjacent pixels. On the other hand, the (x, y, z) value represents each point's spatial position in a lidar image. The low-resolution data itself is a part of the high-resolution data. That is the reason why we keep the raw input data as deep as possible. Figure 3.1b is the structure of a single dense block.



(a) Residual block (b) Dense block

Fig. 3.1: Residual block and dense block

3.2.3 Upsampling Layer

Unlike the object recognition and segmentation problem, where the output size is much smaller than the input size, the upsampling problem's output size is larger than the input size. That means there must be one or more layers to expand the size of the network. There are two simple methods to achieve this goal. The first method is to use some simple upsampling algorithms such as nearest-neighbor, linear interpolation, or bilinear interpolation. After the interpolation layers, the convolutional layers are still used to adjust the output. The other option is "deconvolutional network". The word "deconvolution" used here is not the same term that the mathematically reverse operation of convolution. It can be described as convolution with fractional strides or transposed convolution. We investigate several different methods mentioned above. The pre-upsampling method using bilinear interpolation is chosen for the best performance.

3.3 uvCNN

3D depth data has several formats to be represented in digital systems. Li et al. use the original depth image (1 channel) as the input [31]. The depth data is incomplete. It should be joint with extra information such as a camera matrix K to get each pixel's correct spatial position. Sun et al. use voxel system for 3D reconstruction [83]. The advantage of the voxel system is that it can use 3D convolution. However, for a quantitative problem, the voxel system is too coarse to measure metrics. Our choice is to use Euclidean coordinate (X, Y, Z) of each point. Suppose a lidar point in-depth image is in r^{th} row and c^{th} column and the camera matrix is K, the norm x_n and y_n of this point is first calculated by

$$\begin{pmatrix} x_n \\ y_n \\ 1 \end{pmatrix} = K^{-1} \begin{pmatrix} r \\ c \\ 1 \end{pmatrix}. \tag{3.1}$$

Then the X and Y are given by

$$\begin{pmatrix} X \\ Y \end{pmatrix} = Z \begin{pmatrix} x_n \\ y_n \end{pmatrix}. \tag{3.2}$$

Because we choose the Euclidean coordinate system, the loss function is straightforward using the Root Mean Square Error (RMSE) given by

$$L = \sqrt{\frac{\sum_{i=1}^{n} \left(\hat{X}_i - X_i^{gt}\right)^2 + \left(\hat{Y}_i - Y_i^{gt}\right)^2 + \left(\hat{Z}_i - Z_i^{gt}\right)^2}{n}}, \tag{3.3}$$

where $(X_i^{gt}, Y_i^{gt}, Z_i^{gt})$ is each pixel's ground truth of its 3-dimensional coordinates, and $(\hat{X}_i, \hat{Y}_i, \hat{Z}_i)$ is the output value of the network.

In the computer vision field, based on the types of training targets, tasks using CNNs may be roughly divided into three categories. The first categories are low-dimension target tasks. This category includes various favorite topics like object recognition, object segmentation, or detection. The output of such a task is usually a small vector. The target does

not require very high precision. Take object recognition as an example; if there is an object in an input image, the network's output is the possibility that this object falls into a specific category. If the object is a car, we only want the corresponding element to be the highest number and do not care about this element's numeric value. This category is well studied, and many methods are proposed to improve the performance. Pooling, dropout, and batch normalization are some widely used algorithms and become fundamental components of CNNs.

The second category is artificial high-dimension output tasks. Many image generative models have been developed, e.g., human face generator, artistic painting generator, and emoji generator. Most of these are unsupervised learning. The primary criteria to evaluate models is social acceptance. Generative Adversarial Networks is the most exciting technology presented in recent years [70].

The last category contains models that generate high-dimension output targets based on real data. Image or depth map upsampling, image noise reduction, and gray image colorization are some common research topics. Liu et al. design a synthetic high-dimension target problem [84]. The experiment shows a general inability of CNNs to learn from information related to the Cartesian coordinate. The simplest task is to generate black 9×9 squares in a white 64×64 canvas. The input is the coordinate of the center of the black squares. The state-of-art CNNs surprisingly only reach 86% test accuracy after 1 hour training time. This experiment suggests that the CNNs may not exploit the quantitative locational information of the pixels implicitly. This information should be explicitly delivered to the CNNs. In this paper, the researcher presents an improved CNN called coordCNN, that attaches two new channels to the input data. The first channel contains the row number of each pixel, and the second channel contains the column number. That is why it is named "coordCNN". An additional experiment shows the coordCNN dramatically mitigates the problem above. It also improves the performance of a lot of existing CNNs as a complementary approach.

In our problem, we go further to choose the normalized coordinates (u, v) as the auxiliary information. Compared to the row and column numbers, the uniform coordinates have at least three advantages. First, the normalized coordinates have real geometrical meaning, and they are more accurate because of their continuity. Secondly, different brands of lidar cameras have different lenses and CMOS sensors. Even the same type of lens may have different distortion. That results that for any particular lidar camera, the map of the uniform coordinates are unique. The most important reason is that the uniform coordinates combine the EO image and the lidar image with numeric measurements. In general, a pixel in the EO image does not have a corresponding point in the lidar image. Both cameras have an independent optical system and field of view (FOV). The same row and column numbers represent different areas in the EO and lidar image. On the other hand, the normalized coordinates are mapping the projection of the FOV of the EO camera to the FOV of the lidar camera. When a point in a depth image is projected back to the EO image plane, its corresponding place is usually located at a non-integer pixel location. Of course we could assign the point to its nearest neighbor pixel, but that would reduce the accuracy. As mentioned before, this projection system varies for different depth cameras and different optical cameras. We want the format of texel image to be device-independent, so we used the normalized coordinate system. The projection mechanism guarantees that if represented by (u, v) coordinates, the relative distance of a pixel in EO image and a point in lidar image keeps invariant.

Figure 3.2 describes the entire structure of the uvCNN, which can be divided into 3 subnetworks. The side length of color input is r times the side length of depth input where r is the upsampling ratio. The EO subnetwork on the top extracts color features. Each blue block represents a residual block introduced in Section 3.2.1. The number beneath each block is the output's depth of the correspond layer. Under the EO subnetwork is the Depth subnetwork. Both subnetworks are trained independently though several convolutional layers. Notice that the width and height of units in each layer keep unchanged, and the depth is continuously increasing to enhance the analytic capability without losing spacial

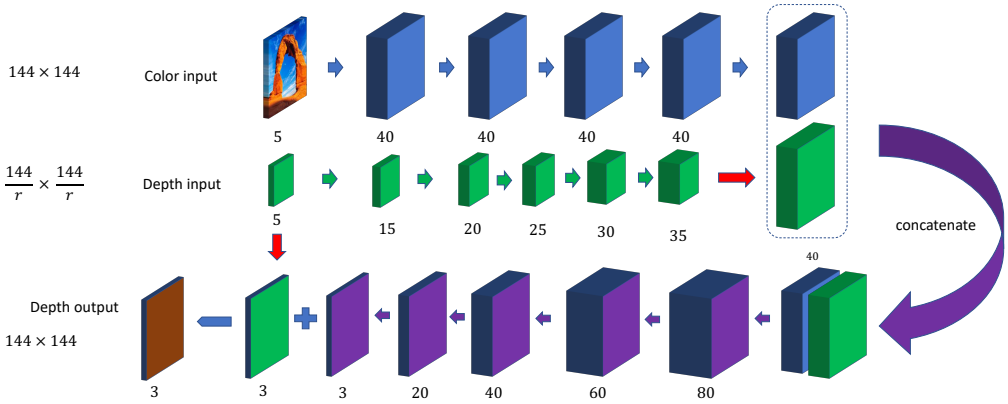
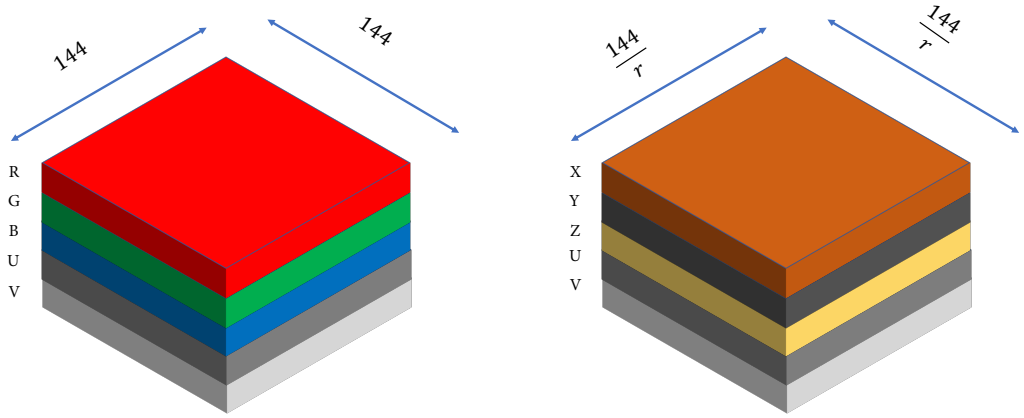


Fig. 3.2: uvCNN all the blue, green, and purple arrows represent a convolution block, respectively. The red arrow represents a "deconvolution" block or transpose convolution block, the stride number is equal to the upsampling rate

structure. Then the output of depth subnetwork is pre-upsampled as described in Section 3.2.3 to match the dimension of the output of the EO subnetwork. After that, the two outputs are concatenated together and sent to the third joint subnetwork. Finally, the output of the joint network is added with the pre-upsampled depth image to generate the final result. That makes the whole 3 networks only extract high-frequency components of the depth image to adjust the interpolated value, especially when depth value changes dramatically.

3.4 Synthetic Texel Image

Because the amount of texel images gathered from our texel camera is far from the amount that a well trained CNN needed, using only real data to training the uvCNN is impossible. Currently, the uvCNN is trained by synthetic data generated from a public 3D dataset. Most public 3D datasets do not meet our requirements. Some of them are only 3D data without color information. Some are simulated data that looks very unnatural. We also want the resolution of the dataset as high as possible. However, some datasets are collected using Microsoft Kinect, whose resolution is only 640×480. Finally 2D-3D-S dataset (S3DIS) is selected [85]. S3DIS uses a Matterport Pro2 3D camera to scan six large-scale indoor areas that originate from 3 different buildings on the Standford campus. It



(a) Image Patch (b) Depth patch

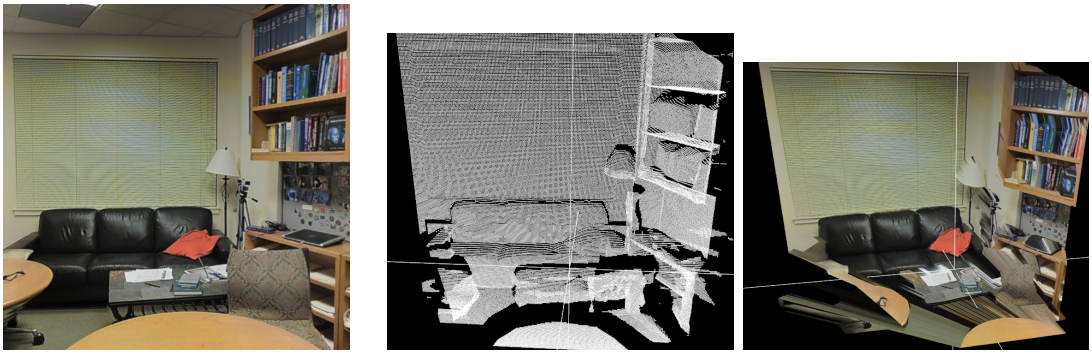
Fig. 3.3: Depth and image patches

provides 70,496 RGB images, along with their corresponding depths, and camera metadata (It contains other formats, but we only need these data). The resolution of the RGB image and the depth image is 1080×1080.

At first, a depth image is transformed to XYZ data according to Equation 3.1 and 3.2. Next, for each pixel in color images and depth images, we calculate its u value and v value according to Equation 2.1 and restore them in two matrices. These two matrices will be stacked with the RGB layers or the XYZ layers to form the color input and the depth input. Then the depth input is downsampled by a downsample rate r. A 144×144 patch of an RGB image and a $144/r \times 144/r$ patch of the corresponding downsampled depth image is cropped as the two inputs of the uvCNN. The 144×144 patch of the original depth image is the target. The color patch and depth patch send to uvCNN is shown as Figure 3.3.

We chose Building 1, 2, 4, 5 as the training set, and Building 3 as the testing set.

Fig. 3.4 shows an example of a synthetic texel Image generated from S3DIS dataset. Compared it to Fig. 2.2, the noise level of the S3DIS dataset is lower than the real texel camera. It reflects the difference between the two mechanisms of the depth camera. The scanner has high accuracy but needs a long time to collect data. TOF camera can achieve real-time but has higher noise and relatively lower resolution.



(a) Synthetic Color Image (b) Synthetic Depth Image (c) Synthetic Texel Image

Fig. 3.4: Synthetic texel image

3.5 Experiment and Result

The training set contains 56902 images, and the testing set contains 3705 images. We build our uvCNN using pytorch 1.0. The whole network runs on an Nvidia Geforce 1080 Ti graphics card. We choose a learning rate $\alpha = 0.00005$ and batch size 30. The uvCNN is trained on different upsampling rate $r \in \{2, 4, 8\}$. Most uvCNN layers share the same structure and hyperparameters with the different r, e.g., channel number, kernel size, patch size, padding size, etc. The only difference is the "deconvolution" layer, whose stride the size will correspond to the upsampling rate.

3.5.1 Preprocess the dataset

The depth data contains both noise and systematic error. Noise can not be avoided in any circumstances and is treated as a part of raw data. However, the training process is impeded and usually falls into a local minimum by the influence of systematic error.

Glass is the primary cause of the systematic error by the nature of the ToF camera that requires the surface of an object have diffuse reflection. Specular reflection and refraction of glass lead to a significant error in the measured distance. A simple method is adopted to relieve the impact of glass. There is a corresponding semantic map for each depth image to indicate the semantic category of an object and its location. If a semantic category "window" is detected in the input data, this piece of data is abandoned. The other exclusion criteria

are whether the input contains invalid data, which is the "missing" data where the camera may not gather the depth information due to its physical limit. The topic of how to rebuild the depth map from incomplete data will be discussed in Chapter 4.

Among all tuned hyperparameters, the kernel size s is a substantial one not often discussed. Many mature CNN frameworks are inclined to use a small kernel size like 3 and 1 to reduce computation load. We tested uvCNN on several different kernel sizes and got an unexpected result. Besides training time cost, kernel size 5 achieved the best performance compared with kernel size 3 and kernel size 7. The average loss is 4 dB lower than the other 2 sizes. Reasonable speculation keeps the width and height of units in each layer unchanged, and zero-padding is applied with the convolutional operation. The larger kernel size indicates more zero elements, which "dilute" existing valid elements. Notice that the "dilution" happens on every convolutional layer. It may be less harmful to qualitative tasks such as object recognition or object detection, but it will enormously impact quantitative tasks' accuracy. Size 5 may be a proper balance point between a larger receptive field and less zero padding elements.

With the growth of the convolutional layers' number, the best performance for different upsampling ratios does not happen on the same layer number. The upsampling ratio 8 achieves the minimal MSE 3 layers less than the ratio 2 and 4.

Meanwhile, to reduce the influence of zero padding elements, a certain amount of columns and rows (2 in uvCNN) on each edge are removed before calculating loss with the ground truth to force the network to focus on the center part of the input and hidden units.

3.5.2 Result and Analysis

We use the MSE between the uvCNN input and the corresponding ground truth as the quantitative result of the network. The two most common up-sampling techniques, Nearest neighbor (NN) [86] and Bilinear Interpolation (BI) [87] are also used for comparison. The training result is shown in Table 3.1. The implementation of the NN and BI method is the built-in function of pytorch. Each cell is the average pixel MSE in millimeters. Our method achieve the best performance on every upsampling ratio. For testing set, Figure

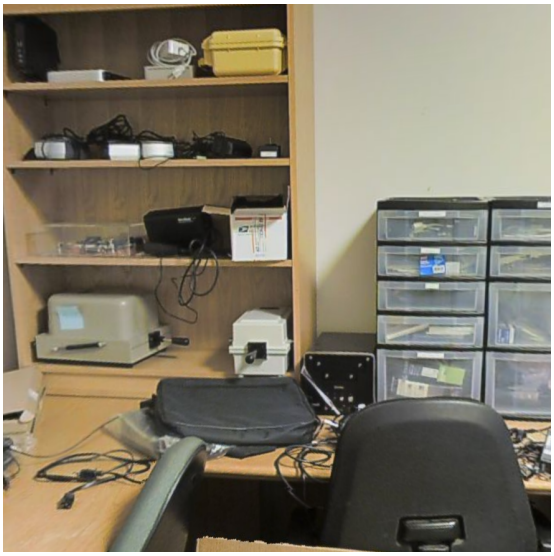
3.5 is an example of 4× upsampling ratio. We present an interior scene with a desk, office chair, bookshelves, and various miscellaneous objects, including a ground truth of depth information, a downsampled low-resolution input, a high-resolution output generated by uvCNN, and a corresponding color image. The proposed method performed well. It can be seen that the quality of the output image is significantly improved, and there is no jaggedness or blurred edges associated with common upsampling techniques. The edges of objects are sharp and the quantization noise is suppressed. More examples are shown in Figure 3.6. The quality of all results is similar to that in Figure 3.5.

Table 3.1: Upsampling Result (Error in mm.)

Upsampling Ratio	**NN**	**BI**	**Training Error(uvCNN)**
2×	1.4	1.0	0.4
4×	4.1	7.6	1.2
8×	9.8	8.2	6.0

3.6 Conclusion

The novel framework uvCNN described in this chapter is not only an improved solution to a long-standing problem using modern deep learning technology, but also an improvement in general thinking on how to design a neural network for a specific real-world task. Upsampling a depth image-guided by a color image is an example of multiple inputs and quantitative output. In this case, the EO image and depth image are related or restricted by the physical (optical) and geometrical laws. The idea behind uvCNN is to provide this relation or restriction numerically. The network can understand that. The uvCNN is a successful attempt to solve this problem. We believe uvCNN is the correct research direction of modern CNN.



(a) Color image

(b) LR input

(c) HR output

(d) Ground truth

Fig. 3.5: The jointed depth image and its ground truth of a desk and a chair

(a) Color (b) LR input (c) HR output (d) Ground truth

Fig. 3.6: More example upsampled by uvCNN

CHAPTER 4

DEPTH IMAGE INPAINTING

4.1 Three-dimensional depth image inpainting

Even today, the actual depth detection and ranging techniques still suffer from measured noise and data that is entirely missing or corrupted. Figure 4.1 is a scene picked out from the Middlebury stereo dataset [88]. Pixels without depth value spread across the whole picture. All the white parts in Figure 4.1b are areas with no depth information, they are manually set to an impossible value (512 meters is the theoretical maximum value) The next few figures illustrate several raw depth images gathered from different types of equipment.

(a) Color Image (b) Depth Image th = $+\infty$ (c) Depth Image th = 20.0

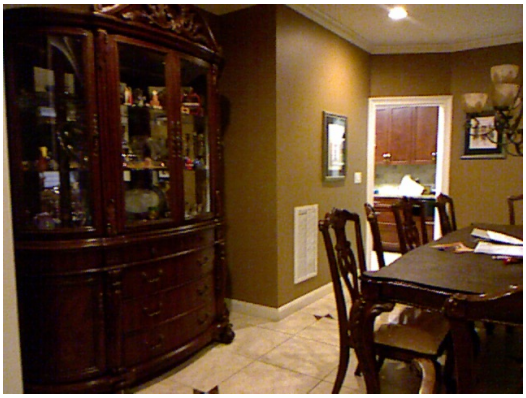
Fig. 4.1: Stanford Depth Image Dataset Collected by Matterport Camera, the depth threshold of the middle depth image is set to infinity, the depth threshold of the right depth image is set to 20.

In Figures 4.1, 4.2, and 4.3, bad points in depth images fall into two categories, scattered single bad points and clusters of bad points. There are a bunch of discrete "bad" pixels or a small "bad" area which may appear on anywhere and seems to have no significant regularity. The only apparent cause is related to the instability of the equipment that gathers depth

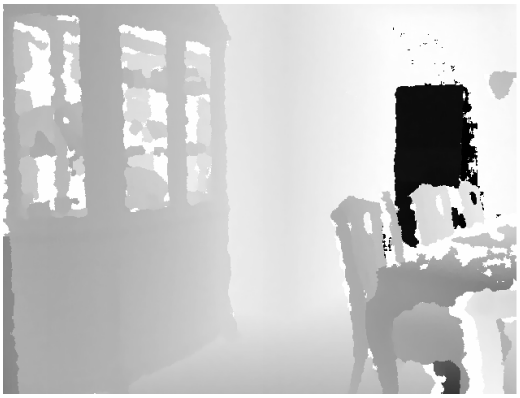
(a) Color Image

(b) Depth Image

Fig. 4.2: Middlebury Stereo Datasets Collected by Canon EOS 450D DSLR cameras



(a) Color Image



(b) Depth Image

Fig. 4.3: NYU Depth Dataset V2

data. On the other hand, clusters of bad points usually appear at abnormally illuminated areas or the edge of objects. The phenomenon indicates that range measurement is disturbed in these areas. Three significant factors may cause this problem. The first situation is that the slope of the surface changes too fast, such as the edges of objects, or a plane that is approximately parallel to the camera's viewing direction. The second situation is that strong specular reflection will cause extensive missing data. The third situation is that the object is too far or too near to the camera, and the distance exceeds the design ranging limit of the camera.

In certain circumstances, the Lidar camera can distinguish whether the received data

is valid, e.g., the received light signal is too intense or too weak, or measured distance is out of scope. The distorted distance value is mixed up with a regular value in the rest of the circumstances, and there is no simple solution to distinguish between reliable and unreliable measurements.

4.2 Attention Layer

In the 1990s, the first breakout application of the neural network was digit recognition [89]. From then on, most milestone level achievements in computer vision happened in the object recognition field [47, 48, 52, 90–92]. It is believed that a CNN-based neural network can fit any complicated function and then solve any existing problem in machine learning using more layers and faster computational units. Sadly it is far from the truth.

The convolutional operation is good at extracting features from spatial ordered data. Higher-level features can be obtained by composing lower level features. As the levels of hierarchies increase, top-level features should determine the original input data's fundamental property. However, the premise is that the input should be valid information. Features generated from partially missing data are unreliable, and this unreliability will be inherited and magnified by higher levels. That is the exact challenge that inpainting research needs to overcome. Iizuka et al. presented a 2-discriminator inpainting framework, which is the best practice of pure CNN based structure [93]. The examples presented in this paper show significant inconsistency. Figure 4.4 is several well-inpainted pictures presented in this paper. All of them are indoor or outdoor scenes of a house building composed of elements with clear geometry structures and patterns like windows, pillars, doors, walls, etc. CNN did a pretty good job of repairing these scenes. However, Figure 4.5 shows some animal images presented in the same paper seem poorly inpainted. The inpainted parts are unnatural and uncorrelated with the surrounding part of the masked area. Furthermore, even these examples are not the worst ones the authors provided. There are far more factors that would influence the inpainting effort than extracting and combining features. The critical standard is whether the inpainted patch looks in harmony with its neighbor, and it seems this is beyond regular CNN's capacity.

Fig. 4.4: Well-inpainted color images created by Global-local Network. The first row is input images with mask, the second row is inpainted images. [93]

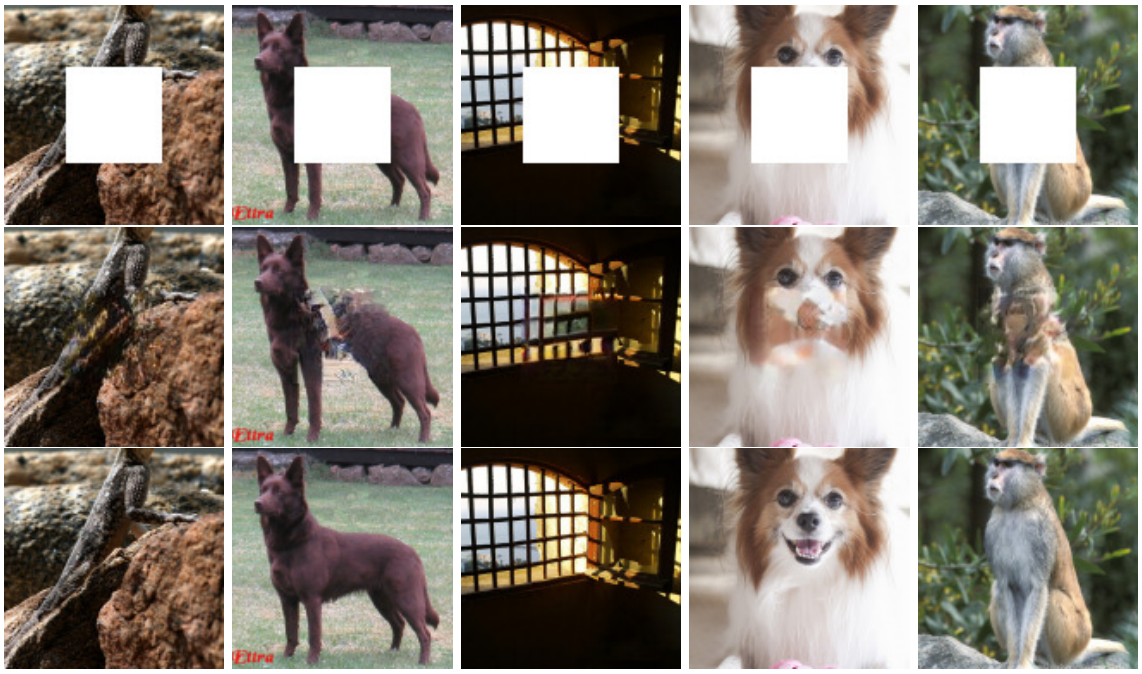


Fig. 4.5: Poor-inpainted color images created by Global-local Network. The first row is input images with mask, the second row is inpainted images, the third row is ground truth images. [93]

In 2018, Yu et al. proposed Contextual Attention Network [94] (CAN). Presently, it is still the state of art image inpainting architecture. Even some new publications claim they achieve a better result, but most of them are still based on CAN, and the advancement is often a tiny difference.

Let us review art designers' empirical method for how to repair a painting or a photograph using Photoshop. When they try to restore a corrupted spot on a picture, they will usually not manually redraw the missing spot. Instead, they will first investigate what the missing spot is. Sometimes it is just a piece of surface that needs the texture to render it. Otherwise, it may be a meaningful motif, part, or object. The designers will then look up some similar texture, motif, part, or object in the same picture or their images database. If found, the filling patch will be copied and pasted to the location. The last step is a lot of tuning and adjustment to make the whole picture natural. Figure 4.6 is a rough instance of how human inpainting works. As it shows, the left half eyeball in the top left image is missing. There is another picture at the top right that contains another eyeball, and the left half picture has a certain degree of "similarity" with the left half part of the top left image. So we may conclude that the right half part of both images should also have some similarities. If there is no better choice, the right half of the top right image can be used to inpaint the top left picture with some necessary adjustment. For example, the diameter of both pupils should be identified as the bottom"inpainted" image.

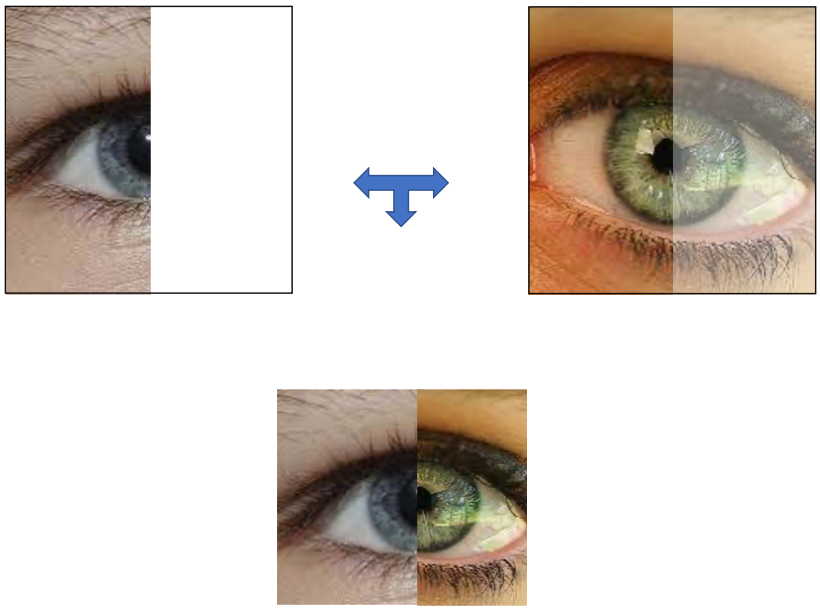


Fig. 4.6: Human eyeball inpainting example

Among all these procedures, the most interesting step is how to determine the similarity of two arbitrary patches. The attention mechanism is an attempt to simulate this process. There are 4 steps to execute the whole attention operation. In the beginning, the input data which is represented as a matrix is duplicated. One of them is called "background", and the other is called "foreground". The original input is also kept for further reconstruction. Then the background is decomposed to a group of 3×3 patches by a sliding window. These patches are reshaped as convolutional kernels. There are many ways to measure the similarity which is called "attention intensity" of two patches. The author chose cosine similarity. We use $f_{x,y}$ to denote the foreground patch centered at (x, y) and $b_{x,y}$ to denote the background patch. Both patches are expand to 1-dimensional vectors. the cosine similarity of the two patches can be written as

$$s_{x,y,x',y'} = \left\langle \frac{f_{x,y}}{\|f_{x,y}\|}, \frac{b_{x',y'}}{\|b_{x',y'}\|} \right\rangle. \tag{4.1}$$

Sagong et al. reported that truncated Euclidean distance resulted in a superior performance [95] which is given by

$$\widetilde{d}_{(x,y),(x',y')} = \tanh(-(\frac{\left\|f_{x,y} - b_{x',y'}\right\| - m\left(\left\|f_{x,y} - b_{x',y'}\right\|\right)}{\sigma\left(\left\|f_{x,y} - b_{x',y'}\right\|\right)})). \tag{4.2}$$

The raw attention intensity is normalized by passing through a scaled softmax function $\sigma(\lambda s_{x,y,x',y'})$. In practice, (4.1) and scaled softmax is equivalent to convolution with background patches as kernels and followed by a channel-wise softmax operation.

The next step is an optional one called "fuse". Consider that neighbor pixels often have close value to enhance the coherency of final attention maps. A left-right propagation is executed to obtain $\hat{s}^{lr}$ as

$$\hat{s}^{lr}_{x,y,x',y'} = \sum_{i\in\{-k,\dots,k\}} s^{*}_{x+i,y,x'+i,y'}, \tag{4.3}$$

then a top-bottom propagation is executed to obtain $\hat{s}^{fused}$ as

$$\hat{s}^{fused}_{x,y,x',y'} = \sum_{i\in\{-k,\dots,k\}} s^{lr}_{x,y+i,x',y'+i}. \tag{4.4}$$

This fuse operation is simply implemented as convolution with an identity matrix as kernels. Though optional, the authors recommend it in practice because it can benefit both the training process by enriching the gradients and testing process by significantly improving the inpainting results.

After the fuse step, all $\hat{s}^{fused}$ blocks are stacked to form a $L \times H \times W$ tensor which is called an "attention map", where H and W is the height and width of the original input, and L is the number of background patches. The value of each pixel in the attention map represents the pixel's attention score at the same location in the foreground. Finally, the attention map is deconvolved with the original input as kernels to reconstruct the foreground. The diagram of the attention layer is shown in Figure 4.7.

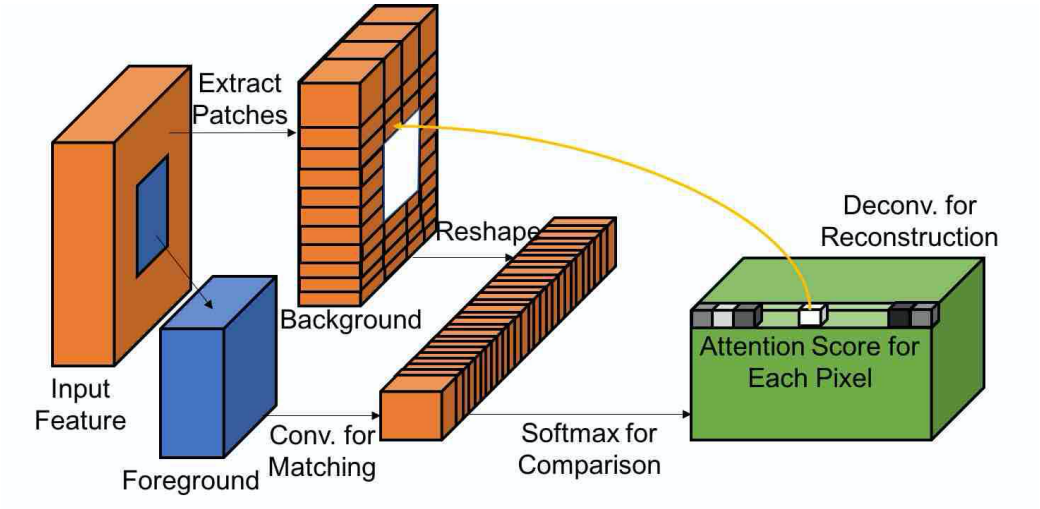


Fig. 4.7: Attention Layer structure [94]

All the mathematical operations mentioned above are convolution, deconvolution, and softmax. It means the attention layer is fully differentiable to be seamlessly integrated into any existing DNN frameworks.

4.3 Surface Attention

As mentioned in Section 4.1, extracting features from depth images are much harder than extracting features from color images. Even if some depth features are obtained, they are often less distinctive and unstable than color features. That is primarily because of less variation of the raw data, and the noise added by the low accuracy device makes it even worse. Inferior features lead to slow convergence or even non-convergence since they dampen the evaluation ability of loss functions and hurt the loss minimization. A pre-process method to represent depth information is necessary to facilitate training indirectly. Chen et al. created relative depth images pairs [96]. Charkrabarti et al. and Li et al. proposed depth derivatives [97,98]. The most prevailing method is surface normal [99–103]. How to calculate the surface normal under a deep learning environment is still an issue worth considering. Nakagawa et al. proposed a complete theoretical algorithm to calculate the precise surface normal for depth images [104], but it needs the depth data that is collected by a pinhole model camera. Calibrated camera parameters f and (c_x, c_y) must be known to infer partial derivatives at every pixel. These premises are not met in all scenarios. A more straightforward approximate surface normal is chosen [80]. To calculate the surface normal around a random pixel $P_{i,j}$, an approximate x-axis gradient vector $(1.0, 0.0, P_{\Delta i})$ is created. Here $P_{\Delta i}$ is the x-axis difference of $P_{i,j}$. Using the same method, we get the approximate y-axis gradient vector $(0.0, 1.0, P_{\Delta j})$. Then the surface normal is the normalized cross product of the two gradient vectors. The whole procedure is shown as

$$
\begin{aligned}
P_{\Delta i} &= \frac{P_{i+1,j} - P_{i-1,j}}{2} \\
P_{\Delta j} &= \frac{P_{i,j+1} - P_{i,j-1}}{2} \\
v_{\vec{\Delta i}} &= (1.0, 0.0, P_{\Delta i}) \\
v_{\vec{\Delta j}} &= (0.0, 1.0, P_{\Delta j}) \\
\vec{v} &= v_{\vec{\Delta i}} \times v_{\vec{\Delta j}} \\
\vec{n} &= \frac{\vec{v}}{\|\vec{v}\|}.
\end{aligned}
\tag{4.5}
$$

Notice that a surface normal vector has 3 channels, so the whole surface normal matrix can be visualized as an RGB image called a surface normal map. The surface normal will be used to pre-process the depth data and compose the vector loss function, which will be discussed in a later section.

4.4 Image Attention Layer

Compared to depth-only images and color-only images, the most significant advantage of RGB-D images and Texel images is that the latter have extra correlated information. The ideal situation is that the color lens and lidar lens are co-boresighted, which means the principal ray of both lenses are coincident, and the principle point of both lenses are equal. If the two lenses are co-boresighted, there is no system parallax, which is the difference in each lens's points of view. Our Texel camera uses a cold mirror as a beam splitter to achieve co-boresighted. Pictures captured by cameras that are not co-boresighted, like Kinect and matterport camera, need post-processing alignment and calibration [85, 105, 106]. Precise alignment of color pixels and depth pixels makes a qualitative and primarily quantitative analysis of their correlation a reality. The extent to which the color information can be used determines the RGB-D data study's performance for both the non-deep-learning field and the deep-learning field. However, there is no universal guideline on analyzing and utilizing color information to solve an arbitrary depth image issue.

We propose uvCNN to upsample a low-resolution depth image into a higher resolution depth image in the previous chapter. In uvCNN, the color image's RGB channels are treated as extra 3 channels along with XYZ channels of the depth image. Moreover, the universal coordinate (u, v) are two "bridge" channels that connect the depth and color data. However, when we apply the same idea to the inpainting network, surprisingly, the network's performance deteriorates slightly compared with the training without a color image. This result is very interesting since it implies that color features extracted by traditional CNN is not directly helpful in reconstructing depth information. It is very difficult to explain how the same type of information acts differently in seemingly similar tasks. We can only make a few guesses that even though both problems belong to the data enhancement category, the

upsampling problem is a uniform data enhancement problem, and the inpainting problem focuses on some specific area. For the former, we want the neural network can generate a particular pattern to refine the raw input data, and that is what the color features are good at. For the latter, we hope our network has an ability to "improvise" something from missing data. In this case, the color features seem a little useless. However, it is a huge waste to abandon the use of color information. We need to find an alternative strategy to incorporate color information into our network.

Remember that the attention mechanism was first introduced to solve the color image inpainting problem. Is a parallel color image attention layer helpful to improve the quality of the inpaited depth image? We believe that answer is yes. The attention map is an indicator to determine where and whether to borrow exact features from the background based on the similarity between the background and the foreground. Cosine similarity is a scalar measure that is unit-independent. So we do not need to take unit transformation from RGB representation to depth measurement into consideration. It is a common sense that two patches with similar depth distribution have a high possibility that they are similar in color, and vice versa. Moreover, the color attention map is superior over depth attention map, which has a higher credibility of its attention score. This viewpoint can be illustrated with the following instance.

Fig. 4.8: Color patch vs. Depth patch

Take Figure 4.8 as an example. The depth measurement of white pixels on the right

depth image is missing. The two green boxes are at the same location, and the right green box contains many bad pixels. When convoluted with background patches, these blank pixels downgrade the reliability of the convolution result. Things would get worse when the background patches also contain void pixels. With current mainstream camera technology, color data collected through the lens is generally considered to be complete and accurate relative to depth data. So this issue does not exist on color attention maps. Here is an extra step that applies the mask on the attention map to force patches to repair the masked area from another place, or the highest scores always happen at the same location. However, in the end, color information and depth information are two completely different kinds of information, so they cannot replace each other. The depth attention layer is still essential.

4.5 Joint Bi-Attention Depth Inpainting Network

We introduce our Joint Bi-Attention Depth Inpainting Network (JBADIN), as shown in Figure 4.9. Arrows with different colors represent different mathematical operations, and blocks with different colors represent the different shapes of tensors. The network is based on [80] with several critical modifications for our task.

The whole network is divided into three sub-networks: the coarse generator, the fine generator, and the discriminator. The vast majority of the modules in Figure 4.9, except for the red ones, indicate the output tensor of the previous convolutional layer, and tensors of the same color indicate that they have the same shape. The red modules are the attentional layers introduced in Section 4.3 and 4.4. Arrows with different colors represent different mathematical operations.

The module in the top dashed box is the coarse generator, whose input is the defective depth image concatenated with the corresponding mask layer, and whose output is the coarsely repaired depth image. We'll discuss the mask layer in more detail in Section 4.7. The coarse generator's purpose is to initially fix small pieces of bad spots, especially discrete ones. The coarse generator would ensure that before being sent into the attention layer, each missing pixel would be assigned a raw value, or the attention layer would become unstable.

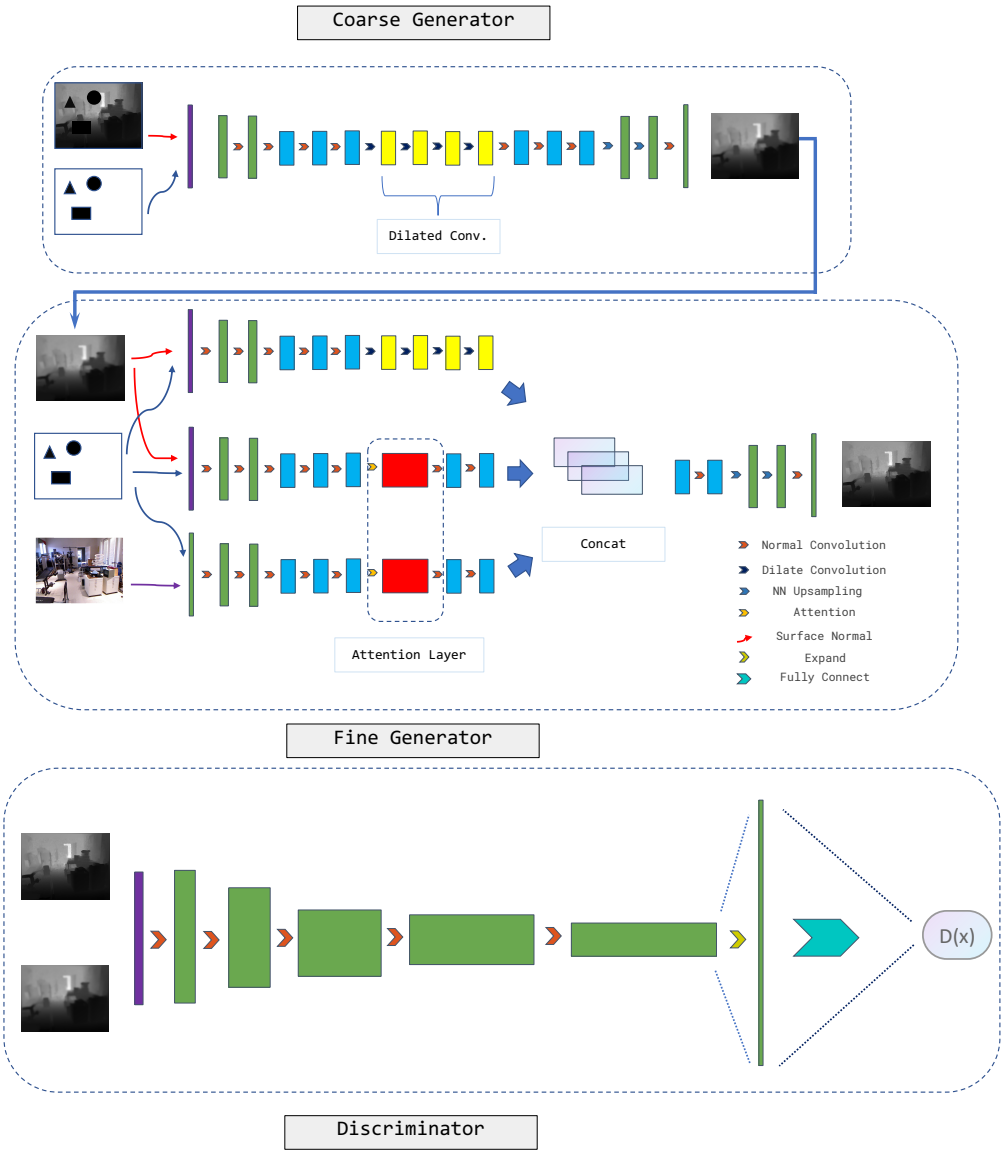


Fig. 4.9: Joint Bi-Attention Depth Inpainting Network

The fine generator, which is in the middle dashed box, is the core part of the entire network. It has two parallel inputs. One is the coarsely inpainted image concatenated with the same mask layer, and the other is the corresponding EO image. It consists of a federation of three parallel channels, and an ordinary CNN that maintains the same structure as the coarse generator. The other two channels are depth attention layer and

color attention layer. At the end of the three channels, three tensors of the same shape are concatenated together and passed to the final convolution blocks. All depth images are processed through surface normal layer and concatenated with original depth data as the depth input. As described in Section 4.2, the attention layer is used to find similar patches and then use these patches to repair the original image. Nevertheless, the selected patches usually are not fit perfectly into the original image. Adjustments to make the whole picture look natural are essential. Furthermore, that is what a traditional CNN is good at. That is why we keep a CNN channel.

All convolutional blocks are composed of a convolutional layer followed with an ELU activation layer. There is no batch normalization layer and dropout layer in every convolutional block because they conflict with the attention layer. There are downsampling and upsampling blocks in convolutional paths and attention layers designed for memory efficiency. It will cause GPU memory overhead if all the hidden layers tensors keep the same height and width. If more powerful GPUs with more significant memory are deployed, these blocks could be removed.

The discriminator in the bottom dashed box is just a classic with 5 convolutional blocks (stride is 2) followed by a fully connected layer. The ground truth and generated depth image are independently send into the discriminator without any indication. Since it is a WGAN discriminator, there is no sigmoid layer to normalize the discriminator's output. The output is the WGAN loss which will optimize both the discriminator and the generator through back-propagation. More details of the implementation of the discriminator will be discussed in the following sections. The whole procedure is presented in algorithm 1.

4.6 Dataset and Data Pre-Processing

We evaluate our Joint Bi-Attention Depth Inpainting Network on Stanford 2D-3D-Semantics Dataset [85] and NYU Depth Dataset V2 [106]. The whole NYU Dataset (1449 images) is used as training set 1. Building 1, 2, 4, 5a, 5b in Stanford Dataset are used as training set 2, and Building 3 is used as the testing set. The Stanford Dataset contains more than $50,000$ images. The researchers use a Matterport Camera to scan the whole building

Algorithm 1 Train Joint Bi-Attention Depth Inpainting Network

while $n \neq N_{iteration}$ **do**
  Slice input images into batch images $\boldsymbol{x}$ from training data
  $m \sim \mathrm{U}(0, 1)$
  if $m < 0.3$ **then**
    Generate Mask Matrix $\boldsymbol{m}$ contains scattered 0 value pixels
  else
    Generate Mask Matrix $\boldsymbol{m}$ contains basic geometry shape of 0 value pixels
  end if
  Construct masked image $\boldsymbol{z} \leftarrow \boldsymbol{x} \odot \boldsymbol{m}$. ($\odot$ denotes pointwise multiplication.)
  Construct surface norm $\boldsymbol{s} \leftarrow \mathrm{sn}(\boldsymbol{z})$
  concatenate $\boldsymbol{z}$, $\boldsymbol{m}$ and $\boldsymbol{s}$ as input $\boldsymbol{y} \leftarrow [\boldsymbol{z}, \boldsymbol{m}, \boldsymbol{s}]$
  Obtain inpainted result $\tilde{\boldsymbol{y}} \leftarrow \boldsymbol{z} + G(\boldsymbol{y}) \odot (1 - \boldsymbol{m})$
  Obtain interpolated data $\hat{\boldsymbol{y}} \leftarrow t\tilde{\boldsymbol{y}} + (1 - t)\boldsymbol{x}$, $t \sim \mathrm{U}(0, 1)$
  Update discriminator with $\hat{\boldsymbol{y}}$, $\tilde{\boldsymbol{y}}$ and $\boldsymbol{x}$
  Repeat *Steps 2 to 14*, Obtain a new set of $\boldsymbol{x}$, $\tilde{\boldsymbol{y}}$, and the output $\dot{\boldsymbol{y}}$ of the coarse generator
  Update Generator with $\tilde{\boldsymbol{y}}$, $\boldsymbol{x}$, and $\dot{\boldsymbol{y}}$
end while

at many scan locations. The 3D textured meshes of the scanned area, the raw RGB-D images, and camera metadata are collected at each location. All the data are registered to form a united 3D building model. Then the model is processed in 3D software blender [107]. Blender can create virtual depth and EO image pixels at the exactly the same location, so that it creates a captured image pair. The location and orientation of each camera are randomly chosen from a predetermined set. The result is that it is inevitable that the vast majority of images in the dataset are very monotonous scenes. Although the researchers took some steps to remove part of those low-quality images, the remaining images were intentionally left intact for the dataset's diversity. Those images are composed of very a few essential building elements like walls, ceilings, and grounds. The information entropy of these images, both color, and depth, are extremely low. Training over low-entropy data results in slow convergence and hurts the quality of training results. Furthermore, these building elements have very simple geometric shapes and are usually flat. Even if they have missed spots, inpainting them does not require a sophisticated model. We decided to remove all these low-entropy images to accelerate the training processing and improve the

training quality. In addition to depth images and color images, the Stanford dataset also provides corresponding semantic maps. The total number of semantic objects in each image varies from 3 to 315, so we set a threshold of 215 to retain only 30% of the data. After discarding the low-entropy images, we get a dataset with 7402 images.

Another factor that will influence the convergence is normalization. Using raw depth values as input also causes a convergence problem. Unlike RGB images with a fixed range of values, e.g., $0 \sim 255$ for 8-bit images and $0 \sim 65536$ for 16-bit images, the range of depth measurements is determined by the measuring device. In our experiment, the NYU dataset collected by Kinect ranges from 0.2 to 9.8 meters, and Stanford dataset collected by Matterport Camera ranges from 0.3 to 48.7 meters. We defined $s = 0.5$ as the range mean of NYU dataset and $s = 24.5$ as the the range mean of Stanford dataset. The inputs also need to be normalized to keep them at the same scale to enlarge the model's generalizability. However, batch normalization, the most common normalization method, would deteriorate the attention score, so we adopt a simple normalization method, that every input has the range mean s of the dataset it belongs removed and then divided by s. That will make sure all inputs will be limited to $(-1, 1)$. The color images are normalized to $(-1, 1)$ also.

4.7 Mask Generation

In this section we will first discuss why we need a mask layer. The target of inpainting technique is to fix bad spots where the location is known. This location information can come from hardware, such as some place on the CMOS sensor is not receiving the photons that should be coming back. The location of a bad spot area may also come from the software, which may also mark a piece of data as unavailable when it is clearly abnormal, such as over-saturated. The mask layer is used to explicitly record and present the location information of these regions.

With the significant improvement of modern camera technology, restoring images is no longer the primary purpose of image inpainting. The main applications of research in this field are image enhancement techniques like watermark removal and object removal. For these applications, the area to be inpainted usually has a fixed location and a fixed

size. Based on this trait, the researchers have simplified the problem that they set the mask to a settled location (usually at the center) with a regular shape (usually square or rectangular) [93, 94]. This simplification speeds up the research process and also allows for more focused research. Nevertheless, for our 3D inpainting topic, it is still the most critical issue to repair the raw, flawed depth images. The lack of features in depth images does not allow us to remove a large portion of the original images. The principle to determine how to generate the mask layer is to seek a balance between generation speed and simulation effort. As mentioned in section 4.1, there are two different kinds of bad points; scattered bad points, and the cluster of bad points. For the first type, If the total number of bad spots is too high or too low, the training quality will be reduced. In order to keep the total number of bad spots generated to fall in a reasonable range, we set each pixel in the mask layer as an independent random variable of Bernoulli distribution with $p = 0.005$. We randomly generate some basic geometric shapes for the second type, e.g., square, triangular, and circle with different sizes, randomly distributed in the layer. Each shape is generated independently so that they may overlap. An extra random morphological operation, like closing, opening, erosion, and dilation, is applied on the mask layer and repeated 1 to 3 times after the generation. Each input will be assigned by one type of mask. A hyper-parameter $\gamma = 0.5$ is set to control the ratio of type 1 and type 2. The two types of mask layers we generated, as shown in Figure 4.10.

4.8 Objective Function and Optimization

The Objective function is another critical factor in determining whether training is successful. Inappropriate objective functions cause training to converge very slowly or even fail to converge. Our depth inpainting network is based on Wasserstein GAN with a gradient penalty ($\lambda = 10$). So the objective function of the discriminator is the same as Equation 2.38, and we repeat it here:

$$L_D = \mathbb{E}_{y \sim P_y} \left[D(\tilde{y}) \right] - \mathbb{E}_{x \sim P_r} \left[D(x) \right] + \lambda \mathbb{E}_{y \sim P_y} \left[\| \nabla_y D(\hat{y}) \|_p - 1 \right]^2 . \tag{4.6}$$

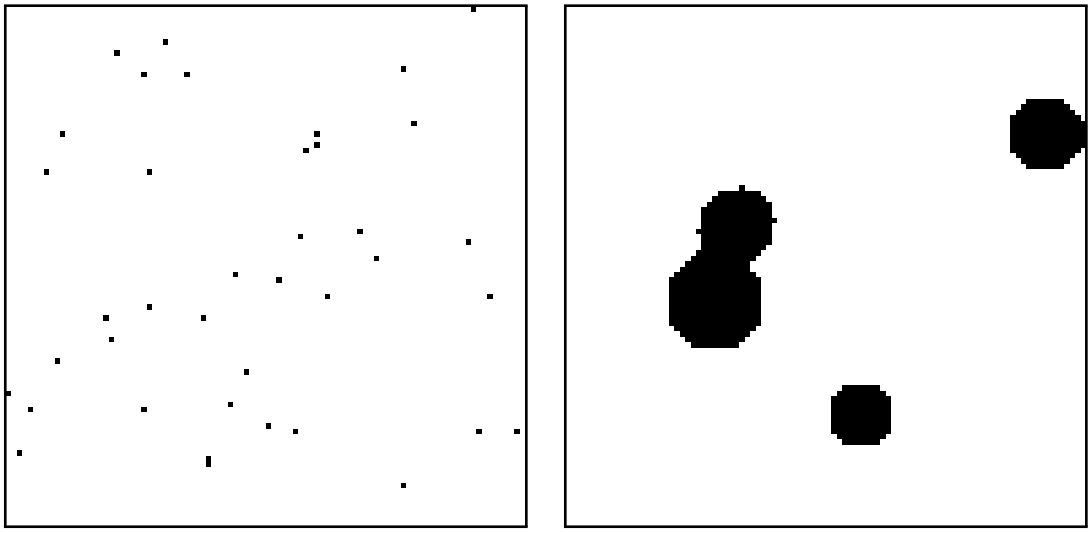


Fig. 4.10: Two types of mask layer, the value of black pixels and white pixels are 0 and 1.

However, the objective function of the generator is much more complicated. For unsupervised learning like artificial image generation, different WGAN loss types listed in Table 2.2 are usually the only available choice. Many state-of-the-art GANs, such as styleGAN-2 and LOGAN, still use a single GAN loss function with coefficient regularization, despite their very complex structure and enormous size [108, 109]. For supervised learning, an additional loss function is usually essential. This is because they provide a more fine-grained optimization strategy for the overall model and can be adjusted depending on the training task. Our objective function of the generator is composed of 3 loss functions. The first loss function is the WGAN loss of the generator is given by

$$L_G^{GAN} = -\mathbb{E}_{y\sim P_y}\left[D(\tilde{y})\right]. \tag{4.7}$$

GAN loss is a global loss function. Referring to other supervised learning, we add a pixel-level loss function to enhance the training results' accuracy when ground truth is available. The pixel-level loss function usually has two choices, L_1 loss and L_2 loss (MSE). There is no fundamental difference between the two loss functions, and which one to use needs to be determined by the specific task. In our experiment, L_1 loss performed slightly

better than L_2 loss, so L_1 loss was chosen. As with GoogleNet [92], the output of the coarse generator, which is the intermediate stage of our model, is a preliminary result that has the same resolution as the final output. Since our network is 29 layers deep, the shallow layers' update would be relatively slow. To speed up the update of the shallow layers, like GoogleNet, we also add the L_1 loss of the coarse generator output $\dot{\boldsymbol{y}}$ to the L_G^1 loss function given by

$$L_G^1 = \frac{1}{N}\sum_{i\in N}\|\tilde{\boldsymbol{y}}(i) - \boldsymbol{x}(i)\|^1 + \frac{1}{N}\sum_{i\in N}\|\dot{\boldsymbol{y}}(i) - \boldsymbol{x}(i)\|^1. \tag{4.8}$$

It is like arranging a shortcut for the coarse generator to be updated twice when backpropagation occurs, which effectively overcomes slow updates.

We simulated two bad spot patterns, with the second type of bad spot, the aggregated bad spot, being the more difficult and challenging issue. However, neither of the two previous loss functions does an excellent job of targeting this scenario. We need to have a loss function that is optimized for small regions. In Section 4.7, we mentioned previous works that generate regular masks like square or rectangular. In these cases, they build another discriminator called "local discriminator" that only evaluates the masked regions. Since our masked regions are irregular and randomly located, this "local discriminator" does not apply to our problem. Compared to a single pixel's depth value, a surface normal vector is a higher-level feature representing the surface depth distribution. It places more emphasis on the local structure's consistency rather than just the depth value errors at the pixel level. Consistency should be a more useful measurement than pixel accuracy when regenerating parts that are already missing. So we add the third term of the objective function: the L_1 error between the surface normal map of the generated depth image and the surface normal map of the ground truth which is defined as

$$L_G^{SN} = \frac{1}{N}\sum_{i\in N}\|SN(\tilde{\boldsymbol{y}})(i) - SN(\boldsymbol{x})(i)\|^1 \tag{4.9}$$

where SN is the surface normal of a pixel.

We now have three loss functions that can optimize the generated depth image at

different levels. GAN loss is used to evaluate the fidelity of the input depth image on a global scale by providing a WGAN score. The L1 loss is designed to reduce the depth error on individual pixels. The surface norm loss plays an intermediate role between the first two, mainly to optimize the local structure reconstruction process.

The three loss functions are multiplied by their respective weight hyper-parameters, α, β, and γ and then summed up to get the generator's objective function L_G which is given by

$$L_G = \alpha L_G^{GAN} + \beta L_G^1 + \gamma L_G^{SN}. \tag{4.10}$$

The value of the weight hyper-parameters is mostly determined empirically. However, a basic rule of thumb is that the value of the 3 hyper-parameters should set the three weighted loss functions within roughly the same range of values. This rule is to ensure that the different loss functions affect the optimization process to a similar degree. If one loss function is too heavily weighted, it causes the other two loss functions to be ineffective. Notice that the WGAN discriminator does not have the last softmax layer so that the output of the generator is not limited to 0 to 1 and can be an arbitrary value. In our case, this value may be in the order of magnitude of 10. L1 loss and surface normal loss is in the order of magnitude of 10^{-4} when the training reaches a stable period. Currently, α, β, and γ are set to 0.0005, 1.2 and 0.8, respectively.

4.9 Experiment and result

We evaluate our inpainting model on an Intel i7-4790K platform with 32 gigabytes of RAM and an Nvidia 1080 Ti graphics card. The generator has 5.0 million trainable parameters, and the discriminator has 1.0 million trainable parameters. The whole model is trained with Pytorch v1.4, CUDNN v10.1, and CUDA v10.1. Due to platform computing power limitations, it is not possible to use the entire image as input. The resolution of the original image is 1080×1080, which is first downsampled to 540×540. Then 32 small 96×96 patches are cropped at random location in the image and stacked as a batch of input. In the

training period, all patches that contains bad points are excluded. each patch is masked by the generated mask layer independently. The discriminator gives a judgment on the authenticity of the generated image and the original image, and then the generator and the discriminator optimize according to their respective objective functions, so this process is a supervised learning. In the testing phase, we try to repair the original image that is actually flawed, generating a mask in the flawed area and using the mask along with the original image as input. The repaired part of the generated result is substituted for the flawed part of the original image to get the final result. Since there is missing information in the original image, the patched original image has no supervised object, it is an unsupervised learning. For the Stanford dataset, we trained 20 epochs; for the NYU dataset, we trained 100 epochs, so as to ensure that the total sample sizes were roughly equal.

Figures 4.11 through 4.16 show an entire training case. They are, in order, the input generated according to Step 9 in Algorithm 1, the randomly generated mask layer according to Step 5 and 7, the generator output, the fusion result generated according to Step 12, the ground truth, and the corresponding color image. Each image is composed of 32 patches that are stacked togehter to form an input batch. If we compare Figures 4.11, 4.14, and 4.15, we can see that the inpainting network performs its task very well. The restored image is very close to the real data, and the contours objects which are corrupted by masks are almost accurately reproduced.

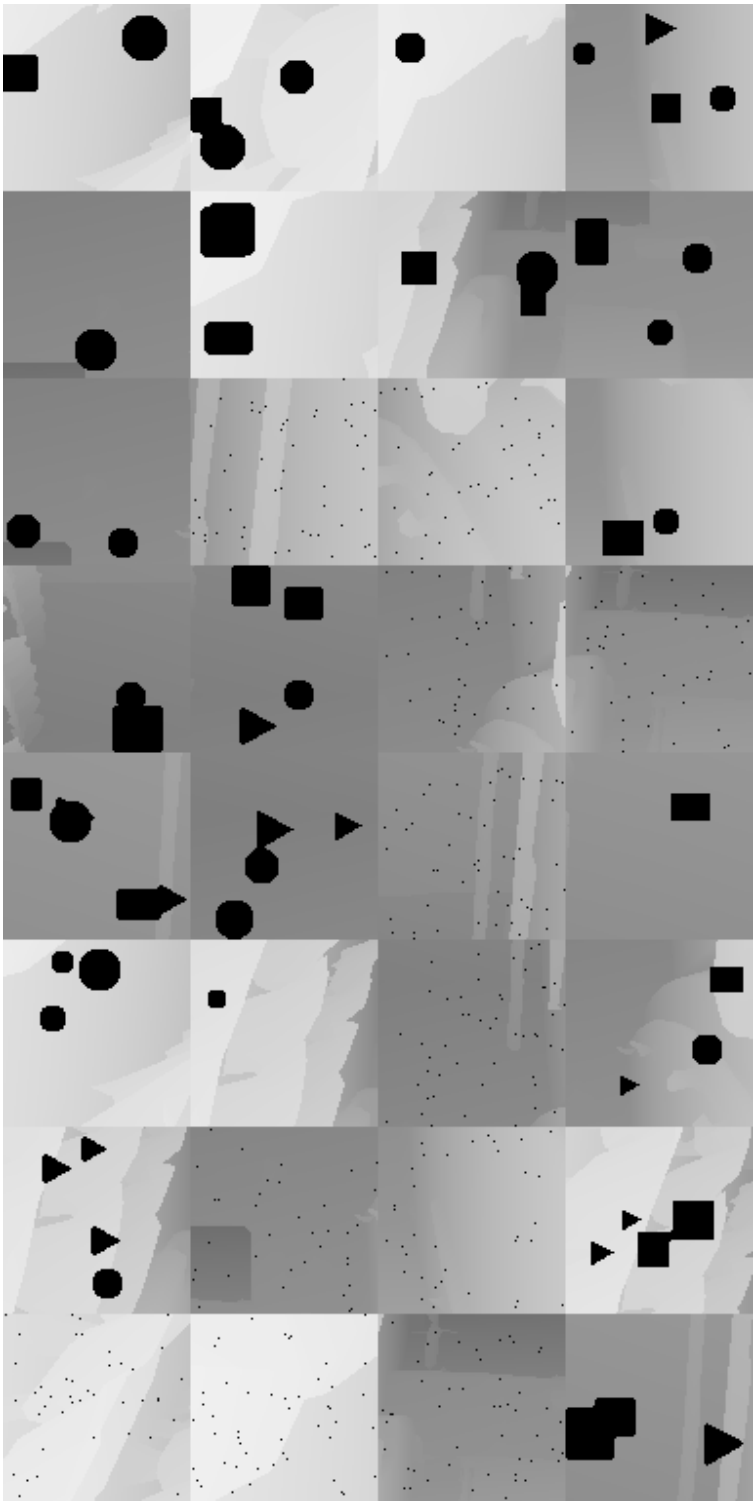


Fig. 4.11: Masked input batch sent to generator

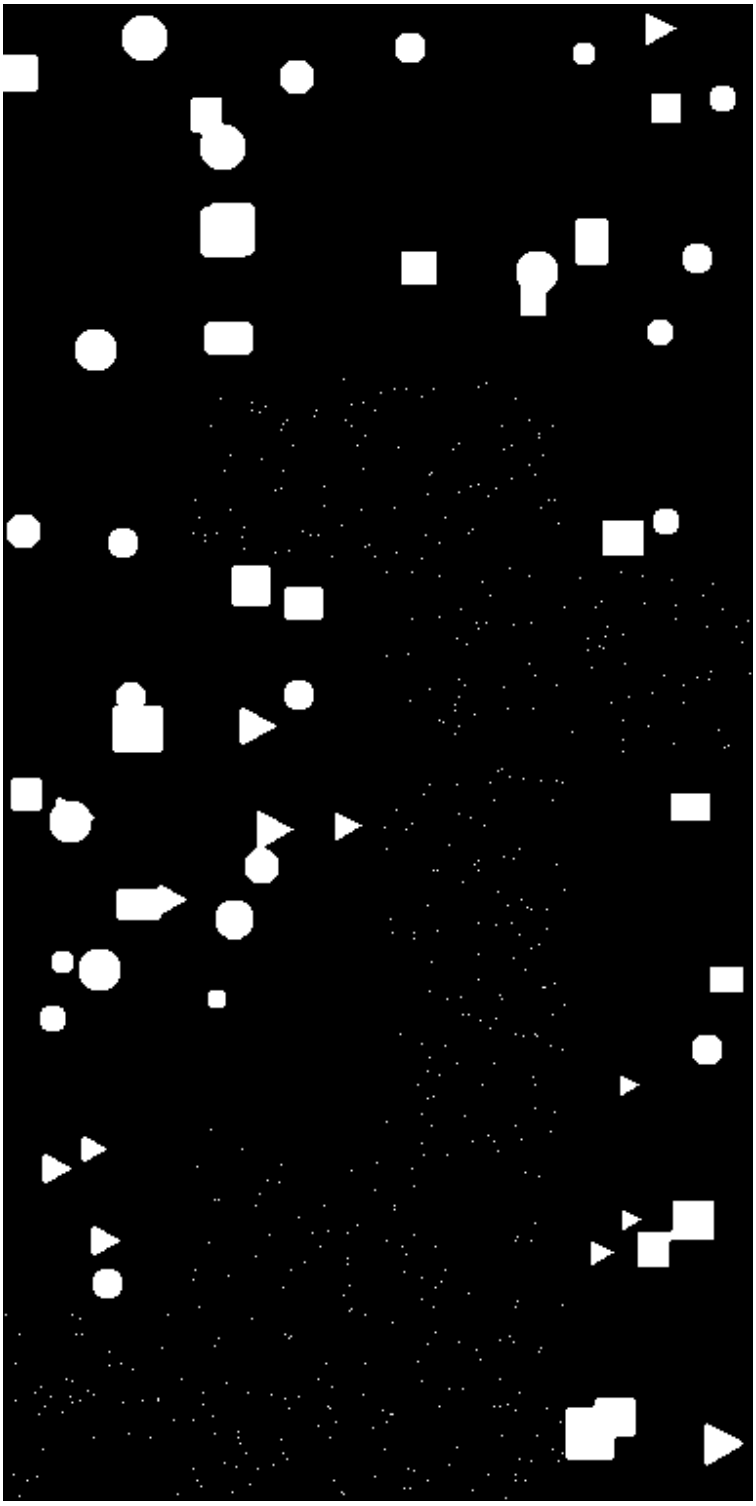


Fig. 4.12: Randomly generated mask layers

Fig. 4.13: The generator's output

Fig. 4.14: The combined result

Fig. 4.15: The ground truth

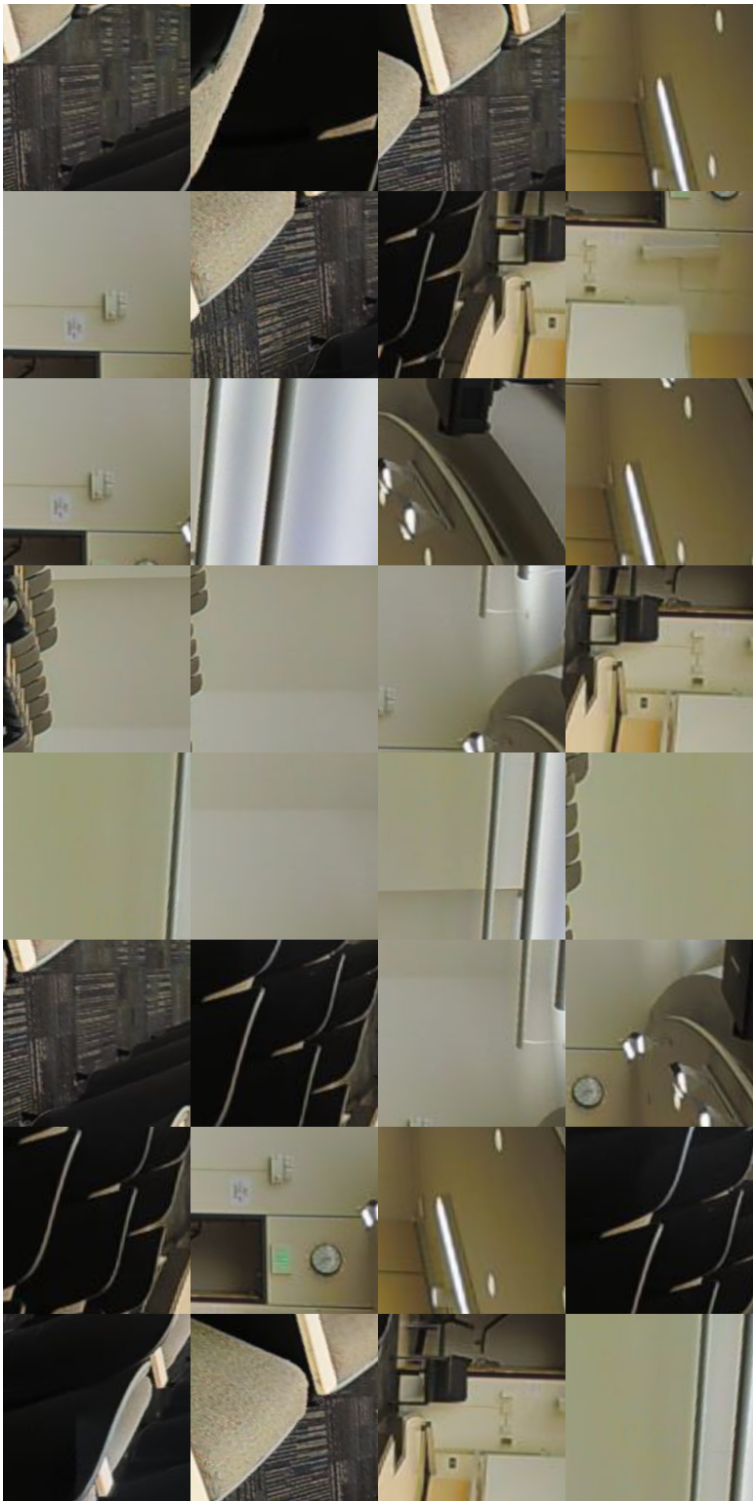


Fig. 4.16: The correspond color images

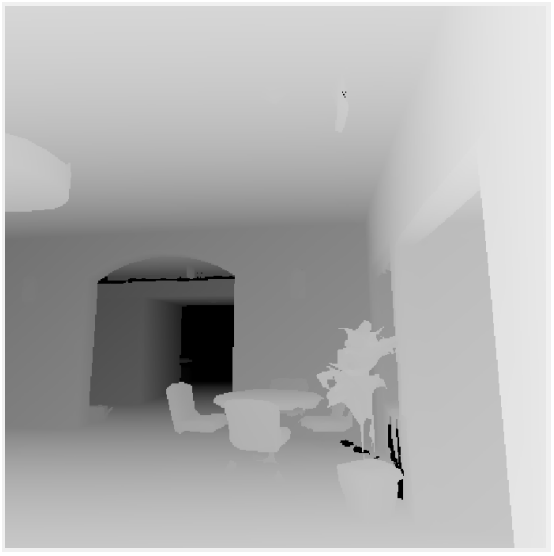
When the training is complete, we use a completely new test dataset to test the effectiveness of the trained model. The depth images in the test dataset have real, not artificially generated, bad spots, so there is no ground truth for these images. All bad points are set to an upper limit of the depth value (512 m), so we can easily generate a mask layer from the original image. The follow images are some test results. Figures 4.18, 4.20, and 4.22 are three examples of tests, each including the input image (top left), the mask layer (top right), the output of the network (bottom left), and the fused result (bottom right). Figures 4.17, 4.19 and 4.21 are their corresponding color images.

Fig. 4.17: The correspond color image of the test scenario 1

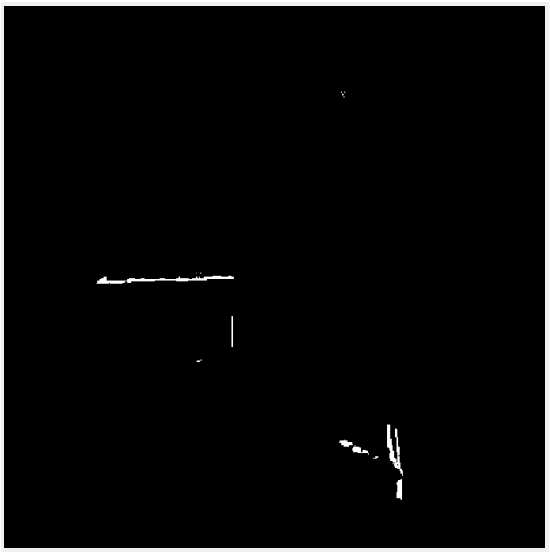
4.10 Conclusion

Judging from the above training and test samples, our joint attention network does its job immensely. In the training sample, even though the mask crosses the boundary of the object, the inpainted image still retains the outline of the object well. And this is exactly what we hope the inpainting task will do. For a less accurate 3D depth image, we don't expect it to generate as much realistic detail as a color image, because the original image is inherently low in detail. The contours and boundaries of objects are the features that are most needed in subsequent applications of 3D images. For the case generated with the test set, we can see that even though the mask is very irregular in shape, the network still does a excellent job of repairing it.

Due to the constraint of our computation power, we had to limit the size of each piece during training. If we can increase the size of the input, we believe that we will get better results in training. A more comprehensive analysis will be presented in the next chapter.

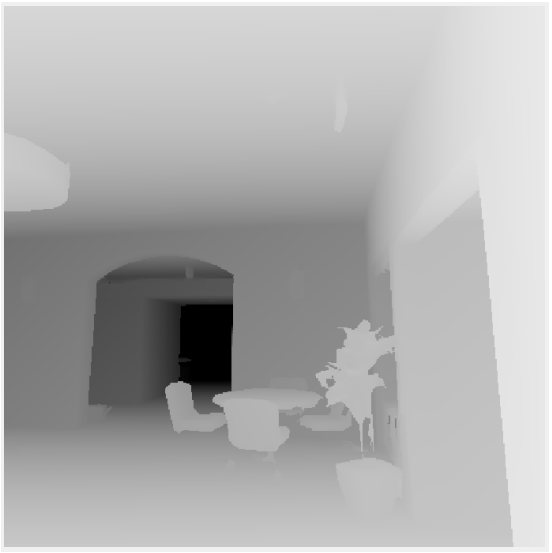


(a) Input depth image



(b) Mask layer

(c) Raw output of the generator



(d) Combined output

Fig. 4.18: The depth input, mask layer, raw output, and combined output of the test scenario 1

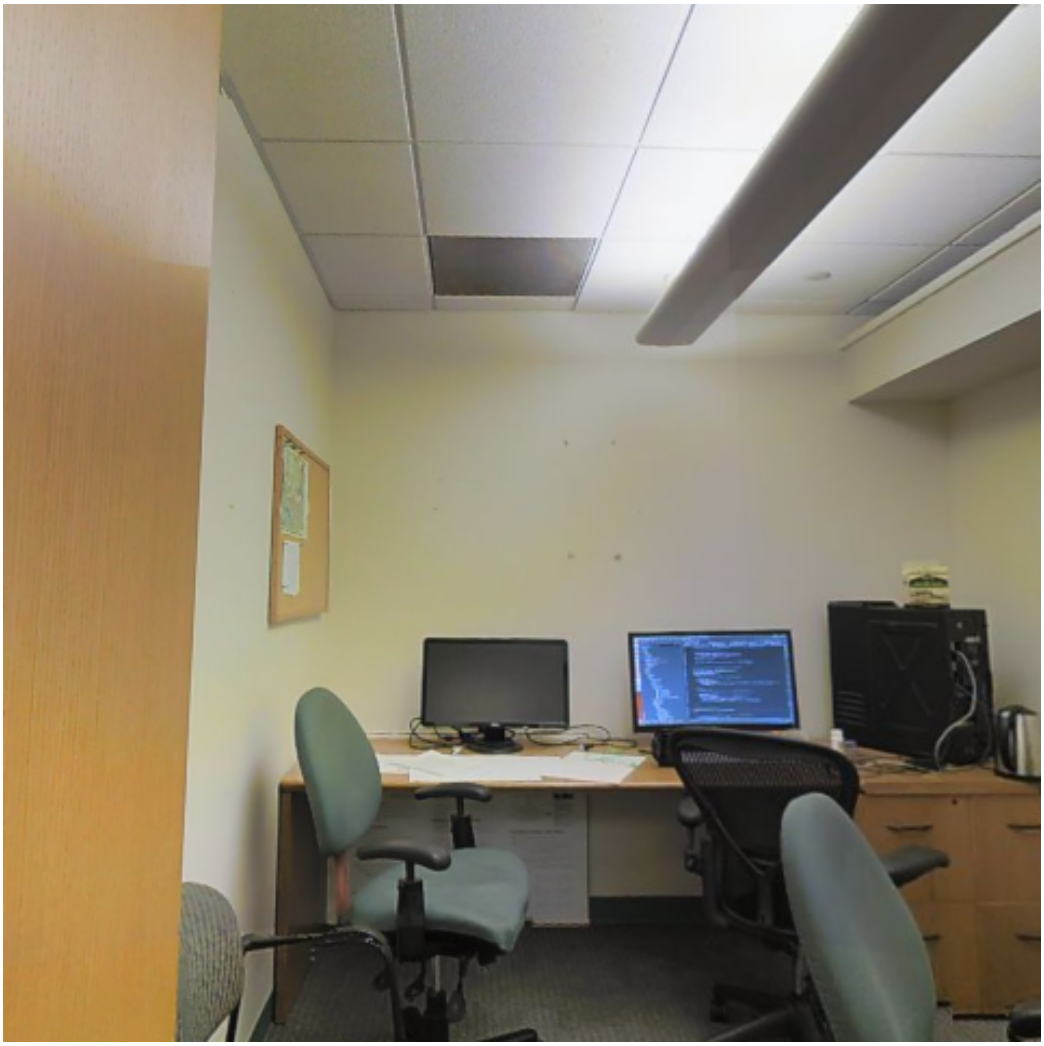
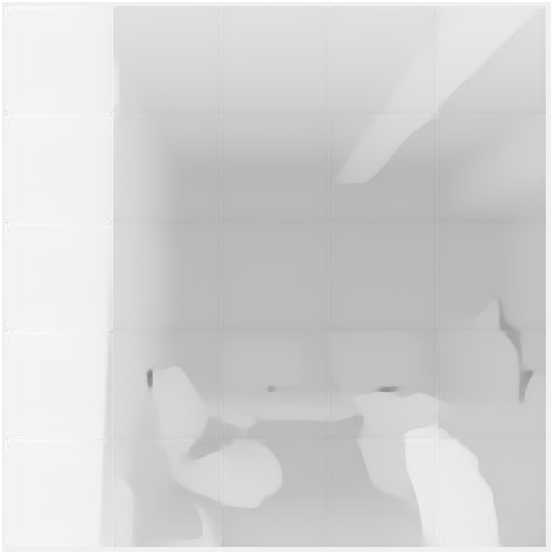


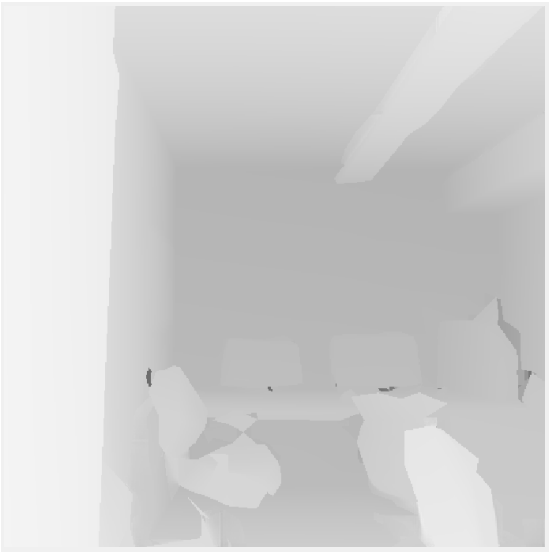
Fig. 4.19: The correspond color image of the test scenario 2

(a) Input depth image

(b) Mask layer



(c) Raw output of the generator

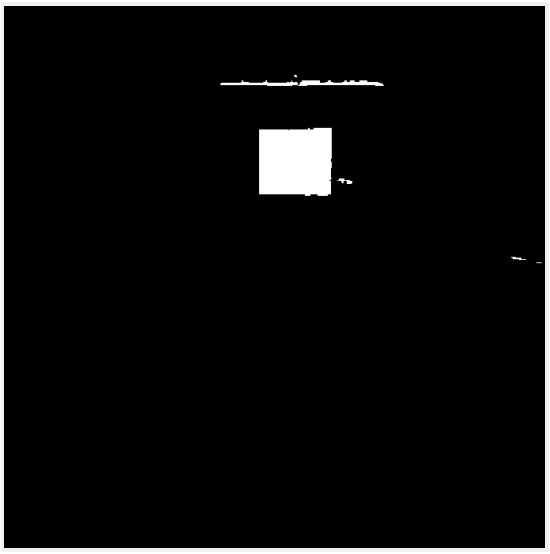


(d) Combined output

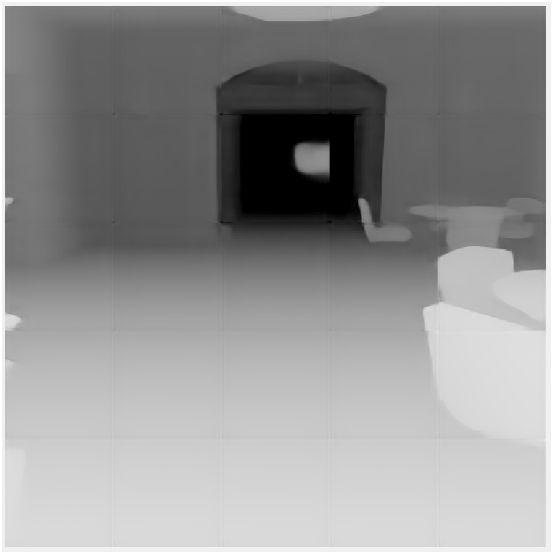
Fig. 4.20: The depth input, mask layer, raw output, and combined output of the test scenario 2

Fig. 4.21: The correspond color image of the test scenario 3

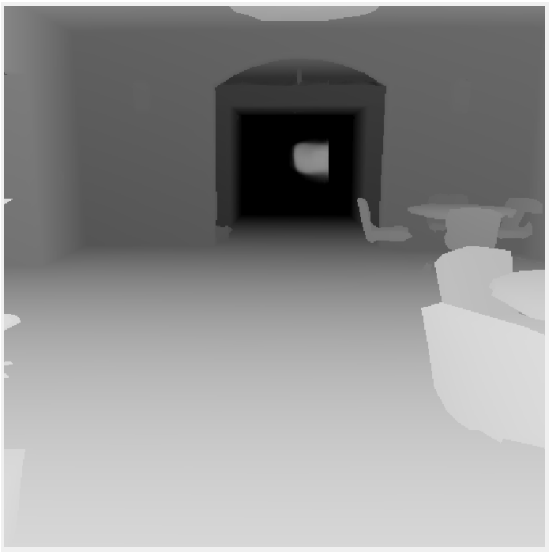
(a) Input depth image



(b) Mask layer



(c) Raw output of the generator



(d) Combined output

Fig. 4.22: The depth input, mask layer, raw output, and combined output of the test scenario 3

CHAPTER 5

CONCLUSION AND FUTURE WORK

The popularity of low-cost 3D data gathering devices has opened the door to virtual reality, augmented reality, robotics, automated driving, and other hot areas of the next era. However, high-quality data is a prerequisite for all of these applications. And resolution and error are two very important indicators of quality. Take virtual reality as an example, the biggest problem with the current mainstream virtual reality devices is that the scenes created by the devices are still not sophisticated enough. Crude "reality" causes instinctive rejection by the human eye and brain, making it impossible for participants to have an immersive experience.

The vast majority of virtual reality devices and applications are at least still safe. For life-safety applications like automated driving, there is an even greater need for high-quality data. Data is the basis for all high-level algorithms. As we mentioned in Section 4.1, currently affordable lidar cameras are unlikely to offer bad-point-free inputs. Automated driving is currently the hotspot of research and development of the world's leading research institutions and commercial companies. A huge amount of money and manpower are invested in this field. Low-quality data is one of main reasons why it is still far from being able to truly put into practice on the road.

The purpose of our study is to address some of the inherent flaws of realistic Lidar imaging and to facilitate subsequent research. Unlike other theoretical inpainting studies, where the shape of the part that needs to be fixed rarely exists in reality, we tried to fix the missing areas on real lidar images.

5.1 Conclusion

The upsampled depth images shown in Chapter 3 and the inpainted images shown in Chapter 4 are very pleasing. The upsampled and patched images look natural. The patched area blends so well with its surroundings that the outline of the previously missing area is not visible to the human eye. The auxiliary color image also does enhance the quality of the output image. Since the dataset was collected from scenes inside the building, many of the images include corridors. Because the length of the corridors is usually well beyond the measurement limit of the lens, there is a bad area with no data at the end of the corridor. In this case, neither convolutional branch nor depth attention branch can learn any knowledge from other place to fill this area, because the world beyond the measuring distance is like a "black hole" for lidar. But in some test results, e.g., the third test example of the Chapter 4, we can see that our framework is still trying to fix this type of "black hole". Although the depth information generated cannot be very accurate, our model draws the outline of the deeper corridor. This contour could only have been generated by the color-attention branch because, as explained earlier, the color information can be considered true and complete in our model, so the network can look for similar parts elsewhere to fix the black hole at the end of the corridor.

One of the main drawbacks of our approach is that it must rely on a mask layer, which requires us to know the exact location of the area to be fixed, and the network can do nothing about the various noises generated by the camera, which are "unknowable" to the model. These noise is also a major factor for the poor quality of depth images. This leads to the output we get that still requires a posterior noise reduction module, whether it's based on deep-learning way or non-deep-learning way. Another real issue is that we still lack high quality texel-image datasets, we use existing public 3D or RGBD datasets to simulate texel images, most of these datasets are not collected with lidar cameras, and some scenes and objects are too monotonous and repetitive.

5.2 Future Work

Our research is exploratory and there is much future work worth investigating. Both of our networks have a number of directions in which they can be optimized, with the primary optimization goal being the size of the network. When designing the networks, we took a more aggressive design approach in order to speed up the progress of the research. The size of the network is the determining factor for the amount of computation, i.e. the power consumption, and the computation time of the system. If the model is to be applied to Internet of Thing (IoT) devices, or in an environment where real-time system performance is required, the network size must be optimized to find an appropriate balance between performance and load.

Besides the network size, the entire network contains a large number of hyper-parameters that could be tuned to boost the network to a better state, such as faster convergence and better repair quality. Thinking further, the various neural networks developed recently are already so large and complex that manual tuning is very difficult and inefficient, and automated tuning tools should be investigated.

Loss functions have been one of the core research directions of machine learning. Excellent loss functions enhance the quality of learning, speed up convergence, and reduce overfitting phenomena. Compared with hyper-parameter tweaking, the repetitive effort to improve loss functions is much less, but more exploratory research is needed. Unfortunately, loss function research on 3D data, especially depth data, is still rare. The loss function used in this dissertation is a useful attempt at a loss function for depth data. More new 3D loss functions should be investigated in the future.

In the previous chapter we pointed out that our inpainting network can only handle missing parts of the data that have been explicitly tagged. For all the noise mixed with real data, other methods are needed to deal with it. These include specialized 3D noise reduction algorithms, additional independent noise reduction neural networks, or optimally extending an existing inpainting network to include noise reduction capabilities.

The quality of the dataset directly determines the upper limit of the training results,

and there are three main indicators of the quality of the dataset, precision, quantity and diversity. The current 3D dataset has been greatly improved over the past in terms of precision and quantity, but diversity is still severely lacking. The field lacks a high quality dataset like ImageNet that can be used as a benchmark for research. And for our study, for each depth image, there also needs to be a corresponding co-boresighted color image. Breakthroughs in research on deep data are to be expected if higher quality datasets are available in the future.

Indicators for evaluating the quality of 3D data are also worth investigated. When the research becomes more sophisticated, it is unreal to evaluate the quality of 3D images only based on the judgment of the human eye alone. For studies with ground truth such as super-resolution, some simple evaluation criteria such as Euclidean distance could be used. For generative learning such as inpainting, it is imminent to have evaluation indexes like FID [46] or IS [45] that do not rely on subjective feelings.

Most current research-based neural networks are single-task driven. To accomplish complex comprehensive tasks, such as deep image preprocessing systems, it is intuitive to simply cascade different single-task systems together, with the output of one system directly or simply processed and then used as input to another. However, this wastes a lot of computing power and is not easy to integrate on various hardware devices. Deep networks have many substructures such as feature extraction that can be shared. Many tasks also have the same part of the objective function. Naturally, more emphasis should be placed on how to develop an efficient complex multitasking network.

In summary, deep neural networks have demonstrated their ability to process structured data, including deep data, but there are still many limitations on whether they can achieve the results we want. These limitations may be theoretical, or the network design may need to be optimized, or the network may be constrained by equipment and require more powerful hardware. Overall, these frameworks are promising groundwork for future research, and making processed depth images clear, accurate, and reliable.

REFERENCES

[1] D. MacKay and R. Neal, "Near Shannon limit performance of low density parity check codes," *Electronics Letters*, vol. 33, no. 6, pp. 457–458, Mar. 1997.

[2] "IEEE Standard for Information technology– Local and metropolitan area networks– Specific requirements– Part 11: Wireless LAN Medium Access Control (MAC)and Physical Layer (PHY) Specifications Amendment 5: Enhancements for Higher Throughput," *IEEE Std 802.11n-2009 (Amendment to IEEE Std 802.11-2007 as amended by IEEE Std 802.11k-2008, IEEE Std 802.11r-2008, IEEE Std 802.11y-2008, and IEEE Std 802.11w-2009)*, pp. 1–565, Oct. 2009.

[3] R. T. Pack, P. Israelsen, and K. Sealy, "A co-boresighted synchronized ladar/EO imager for creating 3D images of dynamic scenes," in *Laser Radar Technology and Applications X*, vol. 5791. International Society for Optics and Photonics, May 2005, pp. 42–51.

[4] B. M. Boldt, S. E. Budge, R. T. Pack, and P. D. Israelsen, "A Handheld Texel Camera for Acquiring Near-Instantaneous 3D Images," in *2007 Conference Record of the Forty-First Asilomar Conference on Signals, Systems and Computers*, Nov. 2007, pp. 953–957.

[5] T. C. Bybee and S. E. Budge, "Textured digital elevation model formation from low-cost UAV LADAR/digital image data," in *Laser Radar Technology and Applications XX; and Atmospheric Propagation XII*, vol. 9465. International Society for Optics and Photonics, May 2015, p. 94650H.

[6] T. Welch, C. Coopmans, and S. E. Budge, "Development of a small unmanned aerial system-mounted texel camera," in *Laser Radar Technology and Applications XXIV*, vol. 11005. International Society for Optics and Photonics, May 2019, p. 110050F.

[7] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *Advances in Neural Information Processing Systems*, 1990, pp. 396–404.

[8] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986.

[9] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[10] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is All you Need," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 5998–6008.

[11] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of machine learning research*, vol. 12, no. 7, 2011.

[12] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv:1412.6980 [cs]*, Jan. 2017.

[13] T. Dozat, "Incorporating Nesterov Momentum into Adam," Feb. 2016.

[14] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[15] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.

[16] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3D ShapeNets: A Deep Representation for Volumetric Shapes," *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pp. 1912–1920, Jan. 2015.

[17] I. Eichhardt, D. Chetverikov, and Z. Jankó, "Image-guided ToF depth upsampling: A survey," *Machine Vision and Applications*, vol. 28, no. 3, pp. 267–282, May 2017.

[18] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, Jan. 1998, pp. 839–846.

[19] J. Kopf, M. F. Cohen, D. Lischinski, and M. Uyttendaele, "Joint Bilateral Upsampling," in *ACM SIGGRAPH 2007 Papers*, ser. SIGGRAPH '07. New York, NY, USA: ACM, 2007.

[20] A. K. Riemens, O. P. Gangwal, B. Barenbrug, and R.-P. M. Berretty, "Multistep joint bilateral depth upsampling," in *Visual Communications and Image Processing 2009*, vol. 7257. International Society for Optics and Photonics, Jan. 2009, p. 72570M.

[21] B. Huhle, T. Schairer, P. Jenke, and W. Straíer, "Fusion of Range and Color Images for Denoising and Resolution Enhancement with a Non-local Filter," *Comput. Vis. Image Underst.*, vol. 114, no. 12, pp. 1336–1345, Dec. 2010.

[22] J. Diebel and S. Thrun, "An application of markov random fields to range sensing," in *Advances in Neural Information Processing Systems*, 2006, pp. 291–298.

[23] J. Lu, D. Min, R. S. Pahwa, and M. N. Do, "A revisit to MRF-based depth map super-resolution and enhancement," in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2011, pp. 985–988.

[24] O. Choi, H. Lim, B. Kang, Y. S. Kim, K. Lee, J. D. K. Kim, and C.-Y. Kim, "Discrete and continuous optimizations for depth image super-resolution," in *Three-Dimensional Image Processing (3DIP) and Applications II*, vol. 8290. International Society for Optics and Photonics, Jan. 2012, p. 82900C.

[25] D. Ferstl, C. Reinbacher, R. Ranftl, M. Ruether, and H. Bischof, "Image Guided Depth Upsampling Using Anisotropic Total Generalized Variation," in *Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 993–1000.

[26] J. Park, H. Kim, Y.-W. Tai, M. S. Brown, and I. Kweon, "High quality depth map upsampling for 3D-TOF cameras," in *2011 International Conference on Computer Vision*, Nov. 2011, pp. 1623–1630.

[27] J. Park, H. Kim, Y. Tai, M. S. Brown, and I. S. Kweon, "High-Quality Depth Map Upsampling and Completion for RGB-D Cameras," *IEEE Transactions on Image Processing*, vol. 23, no. 12, pp. 5559–5572, Dec. 2014.

[28] M. Tallón, S. D. Babacan, J. Mateos, M. N. Do, R. Molina, and A. K. Katsaggelos, "Upsampling and denoising of depth maps via joint-segmentation," in *2012 Proceedings of the 20th European Signal Processing Conference (EUSIPCO)*, Aug. 2012, pp. 245–249.

[29] J. Li, G. Zeng, R. Gan, H. Zha, and L. Wang, "A bayesian approach to uncertainty-based depth map super resolution," in *Proceedings of the 11th Asian Conference on Computer Vision - Volume Part IV*. Springer-Verlag, Nov. 2012, pp. 205–216.

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

[31] Y. Li, J.-B. Huang, N. Ahuja, and M.-H. Yang, "Deep Joint Image Filtering," in *Computer Vision – ECCV 2016*, ser. Lecture Notes in Computer Science, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Springer International Publishing, 2016, pp. 154–169.

[32] F. Banterle, M. Corsini, P. Cignoni, and R. Scopigno, "A Low-Memory, Straightforward and Fast Bilateral Filter Through Subsampling in Spatial Domain," *Computer Graphics Forum*, vol. 31, no. 1, pp. 19–32, Feb. 2012.

[33] D. Miao, J. Fu, Y. Lu, S. Li, and C. W. Chen, "Texture-assisted Kinect depth inpainting," in *2012 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2012, pp. 604–607.

[34] J. Yang, X. Ye, K. Li, C. Hou, and Y. Wang, "Color-Guided Depth Recovery From RGB-D Data Using an Adaptive Autoregressive Model," *IEEE Transactions on Image Processing*, vol. 23, no. 8, pp. 3443–3458, Aug. 2014.

[35] D. Herrera C., J. Kannala, L. Ladický, and J. Heikkilä, "Depth Map Inpainting under a Second-Order Smoothness Prior," in *Image Analysis*, ser. Lecture Notes in Computer Science, J.-K. Kämäräinen and M. Koskela, Eds. Springer Berlin Heidelberg, 2013, pp. 555–566.

[36] J. Liu, X. Gong, and J. Liu, "Guided inpainting and filtering for Kinect depth maps," in *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, Nov. 2012, pp. 2055–2058.

[37] A. K. Thabet, J. Lahoud, D. Asmar, and B. Ghanem, "3D Aware Correction and Completion of Depth Maps in Piecewise Planar Scenes," in *Computer Vision – ACCV 2014*, ser. Lecture Notes in Computer Science, D. Cremers, I. Reid, H. Saito, and M.-H. Yang, Eds. Springer International Publishing, 2015, pp. 226–241.

[38] Y. Zuo, Q. Wu, J. Zhang, and P. An, "Explicit Edge Inconsistency Evaluation Model for Color-Guided Depth Map Enhancement," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 28, no. 2, pp. 439–453, Feb. 2018.

[39] H. Xue, S. Zhang, and D. Cai, "Depth Image Inpainting: Improving Low Rank Matrix Completion With Low Gradient Regularization," *IEEE Transactions on Image Processing*, vol. 26, no. 9, pp. 4311–4320, Sep. 2017.

[40] M. Kulkarni and A. N. Rajagopalan, "Depth inpainting by tensor voting," *JOSA A*, vol. 30, no. 6, pp. 1155–1165, Jun. 2013.

[41] J. T. Barron and J. Malik, "Intrinsic Scene Properties from a Single RGB-D Image," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 17–24.

[42] Z. Yan, X. Li, M. Li, W. Zuo, and S. Shan, "Shift-Net: Image Inpainting via Deep Feature Rearrangement," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 1–17.

[43] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros, "Context Encoders: Feature Learning by Inpainting," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2536–2544.

[44] C. Yang, X. Lu, Z. Lin, E. Shechtman, O. Wang, and H. Li, "High-Resolution Image Inpainting Using Multi-Scale Neural Patch Synthesis," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 6721–6729.

[45] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," in *Advances in Neural Information Processing Systems*, 2016, pp. 2234–2242.

[46] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 6626–6637.

[47] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2818–2826.

[48] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," 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. 1097–1105.

[49] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*. Springer, 2014, pp. 818–833.

[50] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[51] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[52] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[53] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *arXiv preprint arXiv:1511.07122*, 2015.

[54] J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber, "Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction," in *Artificial Neural Networks and Machine Learning – ICANN 2011*, ser. Lecture Notes in Computer Science, T. Honkela, W. Duch, M. Girolami, and S. Kaski, Eds. Berlin, Heidelberg: Springer, 2011, pp. 52–59.

[55] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *The Journal of physiology*, vol. 117, no. 4, p. 500, 1952.

[56] H. Wu, "Global stability analysis of a general class of discontinuous neural networks with linear growth activation functions," *Information Sciences*, vol. 179, no. 19, pp. 3432–3441, Sep. 2009.

[57] D. Sussillo and L. F. Abbott, "Random Walk Initialization for Training Very Deep Feedforward Networks," *arXiv:1412.6558 [cs, stat]*, Feb. 2015.

[58] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, Dec. 1989.

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

[60] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[61] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.

[62] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)," *arXiv:1511.07289 [cs]*, Feb. 2016.

[63] D. Hendrycks and K. Gimpel, "Gaussian Error Linear Units (GELUs)," *arXiv:1606.08415 [cs]*, Jul. 2020.

[64] A. Shah, E. Kadam, H. Shah, S. Shinde, and S. Shingade, "Deep Residual Networks with Exponential Linear Unit," in *Proceedings of the Third International Symposium on Computer Vision and the Internet*, ser. VisionNet'16. New York, NY, USA: Association for Computing Machinery, Sep. 2016, pp. 59–65.

[65] R. G. Wijnhoven and P. de With, "Fast training of object detection using stochastic gradient descent," in *Pattern Recognition (ICPR), 2010 20th International Conference on*. IEEE, 2010, pp. 424–427.

[66] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural networks*, vol. 12, no. 1, pp. 145–151, 1999.

[67] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[68] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus, "Regularization of neural networks using dropconnect," in *International Conference on Machine Learning*, 2013, pp. 1058–1066.

[69] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[70] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Nets," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 2672–2680.

[71] T. Karras, S. Laine, and T. Aila, "A Style-Based Generator Architecture for Generative Adversarial Networks," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2019, pp. 4396–4405.

[72] A. Srivastava, L. Valkov, C. Russell, M. U. Gutmann, and C. Sutton, "VEEGAN: Reducing Mode Collapse in GANs using Implicit Variational Learning," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 3308–3318.

[73] D. Bau, J.-Y. Zhu, J. Wulff, W. Peebles, H. Strobelt, B. Zhou, and A. Torralba, "Seeing What a GAN Cannot Generate," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 4502–4511.

[74] M. Arjovsky and L. Bottou, "Towards Principled Methods for Training Generative Adversarial Networks," *arXiv:1701.04862 [cs, stat]*, Jan. 2017.

[75] S. Arora, R. Ge, Y. Liang, T. Ma, and Y. Zhang, "Generalization and equilibrium in generative adversarial nets (GANs)," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17. Sydney, NSW, Australia: JMLR.org, Aug. 2017, pp. 224–232.

[76] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN," *arXiv:1701.07875 [cs, stat]*, Dec. 2017.

[77] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved Training of Wasserstein GANs," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 5767–5777.

[78] Y. Qin, N. Mitra, and P. Wonka, "How does Lipschitz Regularization Influence GAN Training?" *arXiv:1811.09567 [cs]*, Nov. 2018.

[79] L. Mescheder, A. Geiger, and S. Nowozin, "Which Training Methods for GANs do actually Converge?" Jan. 2018.

[80] L. P. N. Matias, M. Sons, J. R. Souza, D. F. Wolf, and C. Stiller, "VeIGAN: Vectorial Inpainting Generative Adversarial Network for Depth Maps Object Removal," in *2019 IEEE Intelligent Vehicles Symposium (IV)*, Jun. 2019, pp. 310–316.

[81] J. Gui, Z. Sun, Y. Wen, D. Tao, and J. Ye, "A Review on Generative Adversarial Networks: Algorithms, Theory, and Applications," *arXiv:2001.06937 [cs, stat]*, Jan. 2020.

[82] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 4700–4708.

[83] Y. Sun, Z. Liu, Y. Wang, and S. E. Sarma, "Im2Avatar: Colorful 3D Reconstruction from a Single Image," *arXiv:1804.06375 [cs]*, Apr. 2018.

[84] R. Liu, J. Lehman, P. Molino, F. P. Such, E. Frank, A. Sergeev, and J. Yosinski, "An Intriguing Failing of Convolutional Neural Networks and the CoordConv Solution," *arXiv:1807.03247 [cs, stat]*, Jul. 2018.

[85] I. Armeni, S. Sax, A. R. Zamir, and S. Savarese, "Joint 2D-3D-Semantic Data for Indoor Scene Understanding," *arXiv:1702.01105 [cs]*, Feb. 2017.

[86] N. S. Altman, "An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression," *The American Statistician*, vol. 46, no. 3, pp. 175–185, Aug. 1992.

[87] K. T. Gribbon and D. G. Bailey, "A novel approach to real-time bilinear interpolation," in *Proceedings. DELTA 2004. Second IEEE International Workshop on Electronic Design, Test and Applications*, Jan. 2004, pp. 126–131.

[88] D. Scharstein, H. Hirschmüller, Y. Kitajima, G. Krathwohl, N. Nešić, X. Wang, and P. Westling, "High-Resolution Stereo Datasets with Subpixel-Accurate Ground Truth," in *Pattern Recognition*, ser. Lecture Notes in Computer Science, X. Jiang, J. Hornegger, and R. Koch, Eds. Springer International Publishing, 2014, pp. 31–42.

[89] Y. LeCun, L. D. Jackel, L. Bottou, A. Brunot, C. Cortes, J. Denker, H. Drucker, I. Guyon, U. A. Muller, and E. Sackinger, "Comparison of learning algorithms for handwritten digit recognition," in *International Conference on Artificial Neural Networks*, vol. 60. Perth, Australia, 1995, pp. 53–60.

[90] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A Fast Learning Algorithm for Deep Belief Nets," *dx.doi.org*, May 2006.

[91] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv:1409.1556 [cs]*, Apr. 2015.

[92] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper With Convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1–9.

[93] S. Iizuka, E. Simo-Serra, and H. Ishikawa, "Globally and locally consistent image completion," *ACM Transactions on Graphics (ToG)*, vol. 36, no. 4, pp. 1–14, 2017.

[94] J. Yu, Z. Lin, J. Yang, X. Shen, X. Lu, and T. S. Huang, "Generative Image Inpainting With Contextual Attention," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5505–5514.

[95] M.-c. Sagong, Y.-g. Shin, S.-w. Kim, S. Park, and S.-j. Ko, "PEPSI : Fast Image Inpainting With Parallel Decoding Network," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 11 360–11 368.

[96] W. Chen, Z. Fu, D. Yang, and J. Deng, "Single-Image Depth Perception in the Wild," in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 730–738.

[97] A. Chakrabarti, J. Shao, and G. Shakhnarovich, "Depth from a Single Image by Harmonizing Overcomplete Local Network Predictions," in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 2658–2666.

[98] J. Li, R. Klein, and A. Yao, "A Two-Streamed Network for Estimating Fine-Scaled Depth Maps From Single RGB Images," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 3372–3380.

[99] A. Bansal, B. Russell, and A. Gupta, "Marr Revisited: 2D-3D Alignment via Surface Normal Prediction," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 5965–5974.

[100] D. Eigen and R. Fergus, "Predicting Depth, Surface Normals and Semantic Labels With a Common Multi-Scale Convolutional Architecture," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2650–2658.

[101] B. Li, C. Shen, Y. Dai, A. van den Hengel, and M. He, "Depth and Surface Normal Estimation From Monocular Images Using Regression on Deep Features and Hierarchical CRFs," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1119–1127.

[102] X. Wang, D. Fouhey, and A. Gupta, "Designing Deep Networks for Surface Normal Estimation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 539–547.

[103] Y. Zhang, S. Song, E. Yumer, M. Savva, J.-Y. Lee, H. Jin, and T. Funkhouser, "Physically-Based Rendering for Indoor Scene Understanding Using Convolutional Neural Networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5287–5295.

[104] Y. Nakagawa, H. Uchiyama, H. Nagahara, and R.-I. Taniguchi, "Estimating Surface Normals with Depth Image Gradients for Fast and Accurate Registration," in *2015 International Conference on 3D Vision*, Oct. 2015, pp. 640–647.

[105] A. Chang, A. Dai, T. Funkhouser, M. Halber, M. Niebner, M. Savva, S. Song, A. Zeng, and Y. Zhang, "Matterport3D: Learning from RGB-D Data in Indoor Environments," in *2017 International Conference on 3D Vision (3DV)*, Oct. 2017, pp. 667–676.

[106] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus, "Indoor Segmentation and Support Inference from RGBD Images," in *Computer Vision – ECCV 2012*, ser. Lecture Notes in Computer Science, A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid, Eds. Berlin, Heidelberg: Springer, 2012, pp. 746–760.

[107] B. O. Community, *Blender - a 3D modelling and rendering package*, Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. [Online]. Available: http://www.blender.org

[108] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, "Analyzing and Improving the Image Quality of StyleGAN," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 8110–8119.

[109] Y. Wu, J. Donahue, D. Balduzzi, K. Simonyan, and T. Lillicrap, "LOGAN: Latent Optimisation for Generative Adversarial Networks," *arXiv:1912.00953 [cs, stat]*, Jul. 2020.

CURRICULUM VITAE

Xuan Xie

Published Journal Articles

- Automatic registration of fused lidar/digital imagery (texel images) for three-dimensional image creation, Budge, Scott E., Neeraj S. Badamikar, and Xuan Xie. *Optical Engineering 54.3 (2014): 031105.*

Published Conference Papers

- Improved registration for 3D image creation using multiple texel images and incorporating low-cost GPS/INS measurements, Budge, Scott E., and Xuan Xie, *Laser Radar Technology and Applications XIX; and Atmospheric Propagation XI.*