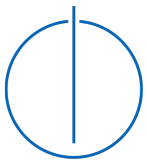
DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Deep implicit network for single-view textured 3D reconstruction

Kuan-Hsun Wu



DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Deep implicit network for single-view textured 3D reconstruction

Tiefes implizites Netzwerk für 3D-Farbrekonstruktion in einer Ansicht

Author:	Kuan-Hsun Wu
Supervisor:	Prof. Dr. Federico Tombari
Advisor:	Shun-Cheng Wu
Submission Date:	May 15, 2021

I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, May 15, 2021 Kuan-Hsun Wu

Acknowledgments

Here, I would like to thank everyone who has supported me throughout my studies and especially for this thesis. First and foremost, a heartfelt thank to my advisors, Shun-Cheng Wu, for offering me the topic and for all the helpful discussions and suggestions during this period.

A special thank goes to my friends Yu-Shan Ruan, Ariel Shann, and Chia-Teng Chang who support, encourage me since Bachelor. Especially during the Covid-19 quarantine time, when I struggled with my thesis, I always received your warm care even though we were thousands of miles away from each other.

Next, I want to express my gratitude to the friends, Jenny Shen, Cho-Chin Cheng, and Hung-Yueh Liao, who I met in Munich. No matter is the difficulty of academic or abroad life, you always help me to go through. Could not imagine life in Germany without being together with you.

Last but not least, I would especially like to thank my parents for their endless support during the course of my studies. This would never have been possible without you.

Abstract

3D reconstruction from single-view images has been a popular research task. However, existing works mainly focus on shape recovery. The color textured 3D reconstruction either only concentrates on a single category or needs several fine-tuned models to overfit each class. In this paper, we propose a Deep implicit network for single-view textured 3D reconstruction, an end-to-end network which can reconstruct both 3D surface and texture from a single 2D image for multiple classes. Our network implicitly represents the 3D surface as the continuous decision boundary of a deep neural network classifier. In contrast to the explicit representation like voxel, mesh, and points cloud, our representation encodes a description of the 3D output at an infinite resolution without more memory usage. Furthermore, Our network first predicts geometry, followed by color, so that the color predictions can be directly informed by the geometric structure.

To the best of our knowledge, this is the first method that uses a single model to recover multiple categories of color textured 3D shapes from single-view images. Even though our experiments demonstrate there is not huge progress in shape reconstruction and the color recovery, our approach achieves the plausible performance on both shape and texture reconstruction.

Contents

1 Introduction

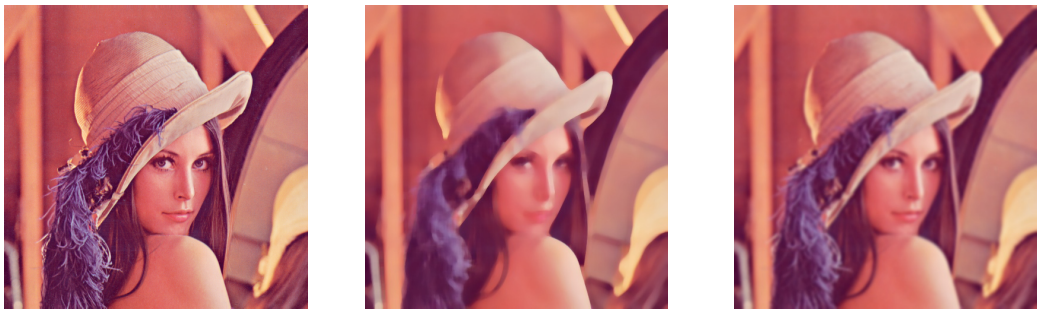
.

1.1 Introduction

Computer vision is a research area after human stored picture in digital format. The digitalized pictures can be processed as multiple layers matrices representing red, green, and blue in computers. Scientists have been implementing algorithms on these matrices to replace human eyes and brains for many tasks. These tasks are from low-level de-noising, blurring, de-blurring, edge detection, keypoints extraction (See the example in Figure-1.1) to high-level image-classification, image-segmentation, and image understanding (See the example in Figure-1.2).

Besides only 2D images, scientists started looking into the corresponding between 2d and 3d area. They would like to imitate the human perception of 2D-3D mapping which maps a 2D image to a 3D object. With the prior knowledge, when we look at a single colorful image of object, it is not hard to know the shape and the color of the object in the 3D space even for the occluded part. If creating an entire object in 3D is as simple as taking a picture, there would be no need for sophisticated 3D scanning devices, multi-view stereo algorithms, or tedious capture procedures, where a sensor needs to be moved around. Modularizing this process will speed up all 3D reconstruction related tasks and even apply to virtual reality and augmented reality in the future.

In recent years, 3D reconstruction from a single image using deep learning has achieved remarkable results (See the example in Figure-1.3). Unlike the traditional computer vision methods such as Oswald et al. [29] and Toppe et al. [42] need the given prior knowledge, e.g. based on user scribbles, and assumptions, and the category of object is limited or it fails when an input image is out of the dataset. The deep learning based methods such as DISN [47] and Occupancy network [25] do not need any additional input and usually could use a single model to recover the shape of multiple categories and generalize to the other dataset. However, most of them only focus on 3D geometry



(a) Original image (b) Median filter (c) Gaussian filter

Figure 1.1: Example of blurring tasks. Figure-(a) is original image. Figure-(b) is the output after applying median filter, and Figure-(b) is output of gaussian filter

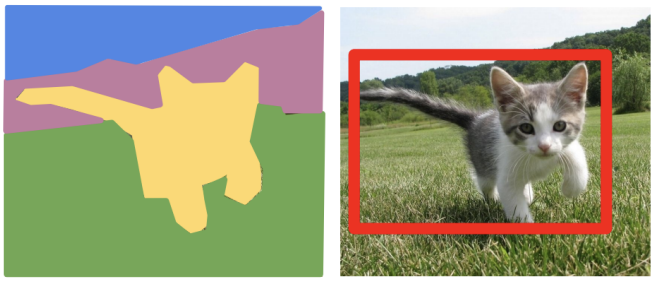


Figure 1.2: Examples of image-segmentation, image-classification. Left is an example of an image-segmentation task. Yellow, green, purple, and blue represent cat, grass, tree, and sky. Right is an example of an image classification task. The object is classified with bounding box. (pictures are from CS231n [40])

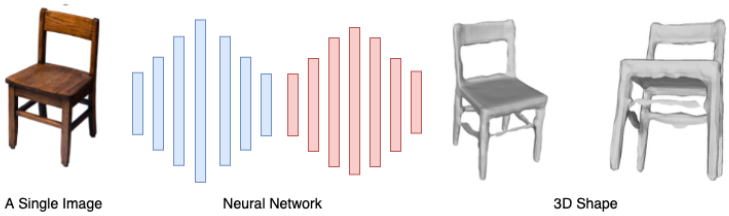


Figure 1.3: Examples of 3D reconstruction from a single image. Input a single image to the trained deep learning model and get the 3D shape as output. (Pictures are from DISN [47])

and ignore the color information such as [24, 45, 46, 5, 11, 44, 25, 47]. Some of them also try to estimate color information but they require separate models to recover different class of objects, such as PIFu [37] and Im2Avatar [41]. PIFu [37] is able to predict both the shape and texture of 3D objects in high resolution by given a single image. However, their method is not an end-to-end method, requires two stages, and can only handle one class at a time. Besides to PIFu [37], Im2Avatar [41] uses an architecture that decomposes the task into shape and color predictions. Im2Avatar [41] applies one stage inference and uses 2 models to recover the 3D shape and the corresponding texture at the same time. Nonetheless, Im2Avatar [41] needs to train different models for different classes. The limitation of PIFu [37] and Im2Avatar [41] intrigue the interest of how to design a single end-to-end model to reconstruct textured 3D shape for multiple classes.

To address the problem of using 2 stages of inferences, we found that SPSG [6] completes the textured 3D scene in a single end-to-end model. The model of SPSG [6] demonstrates that it is possible to recover shape and color at the same time in a single stage inference. We also research how to prevent using multi-models for both shape and texture recovery. The related experiments in Im2Avatar [41] show that jointly predicting color and shape is accessible, but it deteriorates performance slightly.

Therefore, we propose a deep implicit network for single-view textured 3D reconstruction. Our approach not only jointly recovers color and shape within a single inference, but also generalizes to multi-classes recovery. Due to the implicit function, the output of our model could be extracted to a colorful mesh at any resolution. Moreover, the designed structure of predicting color on top of the geometry prediction guarantee that the predicted geometry will not be interfered by the color predictions.

In experiments, we compare our method with state-of-the-art methods. Our method is able to reconstruct textured 3D shapes for multiple categories end-to-end with a single model and achieve comparable results. Furthermore, we analyze the performance under different viewpoints and the failure cases. The results show that our model has the same trend as human being. Both of them fail to recognize the object under the critical viewpoints (ex: parallel or orthogonal angles) and have difficulty identifying the contour of an object when it has the same color as the background.

1.2 Related work

In this section, we briefly overview the related work of 3D reconstruction. 3D shape reconstruction has attracted substantial research interests recently and there are many unsolved problems such as recovering high resolution colorful 3D shape by multi-view images and real-time 3D shape reconstruction in this area. To narrow down the topics which is considered in this chapter, we only discuss the reconstruction of single objects given a single RGB or RGB-D image. Many researchers have explored a variety of geometric shape representations, including voxel grids, mesh, point clouds, and implicit surface representations. In addition to the shape reconstruction, recovering surface color texture is also the popular trend [37, 41, 41]. However, colorful 3D reconstruction is still a challenging and unsolved problem since the 2D-to-3D recovery involves both satisfying geometric constraints and inferring visually appealing textures.

1.2.1 3D shape reconstruction

With the emergence of high-capacity deep neural networks, task of 3D reconstruction has witnessed great progress to enable powerful modeling of 2D-to-3D mapping. 3DShapeNet [24] regards a 3D voxel grid as a binary variable and use a Convolutional Deep Belief [19] to estimate the probability distribution of each grid. It recovers 3D voxel grid shapes from a view-based 2.5D depth map (e.g. images from Microsoft Kinect) for different object categories.

Instead of using 2.5D depth map as input, MarrNet [45] takes only a single RGB image as input and employs 2.5D estimation as an intermediate component which recovers object normal, depth, and silhouette images from an RGB image. It then regresses the 3D shape from the 2.5D sketches and introduces a reprojection consistency loss to ensure the estimated 3D shape aligns with the 2.5D sketches. The whole network is trained end-to-end by encoding-decoding architecture. Inspired by MarrNet [45], ShapeHD [46] reconstructs 3D shapes via modeling 2.5D sketches but incorporating a naturalness loss to reach higher quality. The naturalness loss function is an adversarially pre-trained convolutional net that serves as a regularizer, penalizing the model only if its output is unrealistic.

From a perspective of a learning agent, Perspective Transformer Nets [48] formulates the learning procedure as a transformation between 3D and 2D representations. The whole network is an encoder-decoder structure with a projection loss which applies projection transformation as regularization. More importantly, the projection loss enables unsupervised learning using 2D observation without explicit 3D supervision.

Differ from the encoder-decoder structure, TL-embedding network [12] design the T-network, L-network for training and testing. The novel architecture regards the 3D object as an embedding space and tries to learn a vector representation in the autoencoder layers. Instead of the 3D convolutional neural network, C. B. Choy proposes a 3D Recurrent Reconstruction Neural Network (3D-R2N2) [5] which takes advantage of long-short term memory (LSTM) to reconstruct a 3D occupancy grid for single image or multi-view images.

Besides the voxel grid, points cloud and mesh are also commonly used as 3D representation. Haoqiang Fan et al. proposes the point set network [11] based on a point cloud representation with two prediction branches, one enjoys high flexibility in capturing complicated structures and the other exploits geometric continuity. Pixel2Mesh [44] is based on a graph-based convolutional neural network and produces a watertight mesh object by progressively deforming an ellipsoid, leveraging perceptual features extracted from the input image.

The aforementioned methods use explicit 3D representations which often suffer from problems such as limited resolution and fixed mesh topology. Implicit surface representations such as Signed Distance Functions (SDF) and truncated signed distance function (TSDF) use continuous field implicitly represents the underlying 3D shape to overcome these limitations. In contrast to existing explicit 3D representations, implicit representations output at an infinite resolution without excessive memory. At inference time, given a set of signed distance values, the shape can be extracted by identifying the iso-surface using methods such as Marching Cube [22] algorithm.

Several deep learning approaches have utilized SDFs recently. DeepSDF [30] represents SDF of shapes that enables high-quality shape representation, interpolation, and completion from partial and noisy 3D input data via latent code-conditioned decoder networks. However, their network requires optimizing the embedding vector during test time and could not deal with the multi-classes object which limits the efficiency and capability of the approach.

Occupancy networks [25] considers continuous decision boundary as a binary classifier which predicts the probability of each cell's occupancy in a volumetric grid i.e., being inside or outside of a 3D model. By iteratively subdividing each active cell (i.e., cells surrounded by occupied and empty cells) into sub-cells and repeating the prediction

for each sub-cell, they alleviate the problem of the limited resolution of volumetric grids. Furthermore, in order to capture details such as holes and thin structures present in 3D shapes from single-view images, instead of predicting the sign (i.e., being inside or outside) of sampled points, Deep Implicit Surface Net (DISN) [47] predicts the continuous distance by extracting global features and local features from projecting the 3D points. This design enables the network to learn the relations between projected pixels and 3D space and achieves state-of-the-art performance in single-view reconstruction.

1.2.2 3D textured reconstruction

While such methods have shown impressive geometric reconstruction, the colored texture mapping from 2D to 3D has been less explored. Texture Fields [28] inputs 3D shape and a single 2D image to learn a continuous function which maps any 3D point to a color value. The idea is parameterizing this function through a deep neural network which consist of conditional, generative adversarial network (GAN) and Variational Autoencoder (VAE) models.

Nonetheless, Texture Fields [28] doesn't recover the 3D shape, only uses input image to inpaint the known 3D object. Pixel-aligned implicit function (PIFu) [37] recover the high-resolution 3D textured surfaces of clothed humans from a single input image by adopting the implicit function to regress RGB values at each queried point. The structure could be divided into surface reconstruction and texture inference. Given an input image, surface reconstruction predicts the continuous inside/outside probability field of a clothed human. Then, texture inference infers RGB values at given 3D positions of the surface geometry.

PIFu [37] reconstructs highly plausible geometry including largely unseen regions such as the back of a person, but it only focuses on the clothed human. Im2Avatar [41] shows the capability of applying various objects (five classes in Shapenet [4] and human class in Colorful Human dataset [16]). It applies 3D convolution layers to predict the 3D shape in the voxel grid with corresponded color information. Unlike PIFu [37] which reconstructs geometry firstly and uses 3D output shape as input to inpaint the surface's texture, it outputs shape and the corresponding color at the same time by two branches of encoding-decoding network.

However, in Im2Avatar [41], the shape and color learning processes are decomposed,

they are trained independently and each category is also trained separately. In contrast, our proposed method is an end-to-end network which reconstructs textured 3D shape for multiple classes. The network also applies implicit representation which is not limited by the high memory requirements of explicit representations and could be transformed to color mesh at any resolution.

2 Fundamentals

This chapter we introduce some core topics which are used in the reconstruction methods we consider. In order to understand how such a system works, how it can be improved, it is crucial to have some basic knowledge about its algorithmic aspects. Firstly, we introduce how to represent a 3D object in computer. Afterwards, we show the transformation between 3D object and 2D image. Finally, we talk about the basic knowledge of deep machine learning.

2.1 3D representation

How to digitalize the 3D object which computer could apply the transformation, light effect and multiple matrix computation such as doing image process is important. The representation will directly affect the level of difficulty in the reconstruction tasks and the appearance of 3D objects in the 3D space. There are several ways to represent 3d space in computer vision. Each one has different pros and cons and all of them are still commonly used in any task related to 3D.

- **Point clouds :** The most intuitively method is simply collect the points of object in 3D space (See in Figure-2.1 left). Each point is specified by x, y, z 3D location, optionally along with some other attributes such as r, g, b color information. This format is usually the raw data of depth sensor such as LiDAR and Microsoft Kinect. In the simultaneous localization and mapping (SLAM) related tasks, RGB-D data (which consist of an image labeled with per-pixel depth values) are usually converted into point clouds before further processing.

- **Voxel grids :** Voxel grids are made of small cubes and is derived from point clouds (See in Figure-2.1 middle). A voxel represents a value on a regular grid in three-dimensional space similar to pixels in 3D, with quantized, fixed-sized point clouds. Compared to the point clouds which is a set of points anywhere in space, and doesn't have a solid entity, voxel grids have a fixed resolution and solid shape that is friendly for the visualization. The benefit of voxel grids is easy to find the neighbors and the connected grids form the surface of object. The main drawback is huge memory consumption.

- **Polygon meshes :** Meshes consist of a set of polygonal faces with shared vertices that approximate a geometric surface (See in Figure-2.1 right) i.e. the shell of the object, not its volume. Each vertex stores x, y, and z coordinate information, and every polygon, which is typically quadrangles or triangles, is composed of multiple vertices. Instead of collecting sampled 3D points from an underlying continuous geometric surface, polygon meshes gather the polygonal faces aiming to represent those 3D objects in a way that can be easily rendered. Compared to points cloud and voxel grids, Mesh is an efficient way to present the geometry of the 3D object, and often widely reduce the number of needed points. Almost all 3D visual tasks such as games, films use it as the default representation.

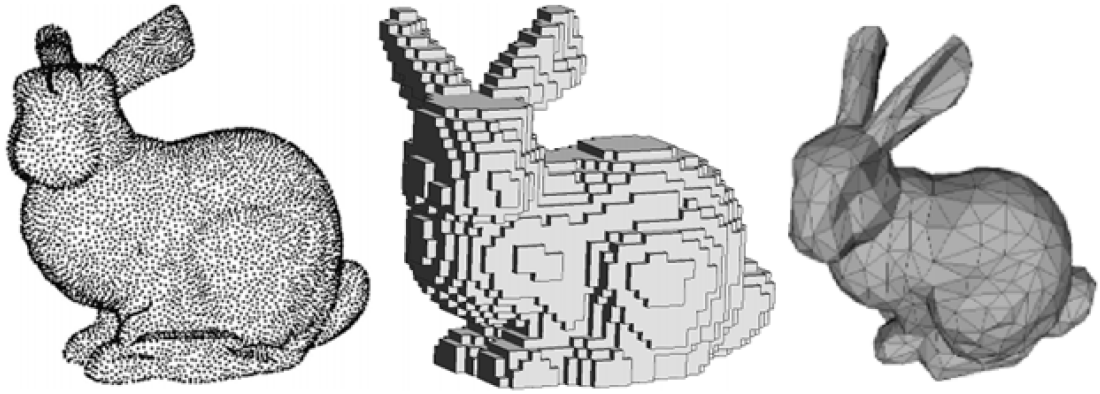


Figure 2.1: Example of explicit representations. Left: points cloud, middle: voxel grids, right: polygon meshes. (Stanford bunny model is from Greg Turk [43] and Marc Levoy [20] in 1994 at Stanford University)

The above-mentioned representations of 3D object are explicit which resolution is limited to the number of points, grid, vertices and meshes. The implicit representation applies the signed distance function (*SDF*) which represents the surface at the set of zeros in the continuous space to get rid of the limitation and does not require extra memory usage. Given an Object Ω, *SDF* is defined by the distance metric d (See the equation 2.1) which determines the distance of a given point $p = (x, y, z) \in \mathbb{R}^3$ from the boundary of the object Ω ($\partial\Omega$ denotes the boundary of Ω) with the sign determined by whether p is in object Ω. The function $s = SDF(p)$, $s \in \mathbb{R}$ has negative values at points inside object Ω, where the signed distance function is zero at the object surface, and it takes negative values inside of object Ω (See in Figure-2.2).

$$SDF(p) = \begin{cases} d(p, \partial\Omega) & p \notin \Omega \\ -d(p, \partial\Omega) & p \in \Omega \end{cases} \tag{2.1}$$

To visualize the implicit function, we could transfer it to the explicit representation. There are several methods to convert the implicit representation to Polygon meshes, such as Marching Cube [22]. Even though the final representation is still the explicit mesh format, the continuous function SDF could be extracted in any resolution. The final output is more accurate and smoother than the explicit mesh representation (See in Figure-2.2).

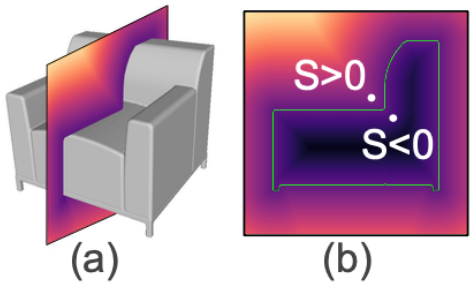


Figure 2.2: Illustration of SDF in 3D space. Figure (a) Rendered 3D surface with s = 0. Figure (b) Cross section of the SDF. A point is outside the surface if s > 0, inside if s < 0, and on the surface if s = 0. (pictures are from DISN [47])

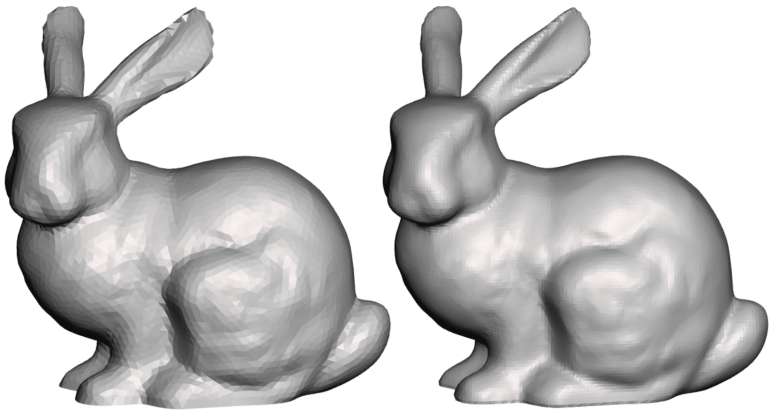


Figure 2.3: Examples of mesh in explicit representation and the mesh extracted from SDF. Left is mesh in explicit representation and right is the mesh extracted from SDF (Stanford bunny model is from Greg Turk [43] and Marc Levoy [20] in 1994 at Stanford University)

2.2 Transformation from 3D space to 2D image

Taking a picture of an object is a process which projects 3D object into a 2D image. The whole process is a linear transformation which can be implemented by a simple matrix multiplication. This matrix is called camera projection matrix which maps the 3D points in the world to 2D points in an image. The 3D-2D mapping transforms coordinate between world, camera, and image coordinate systems (See in Figure-2.4).

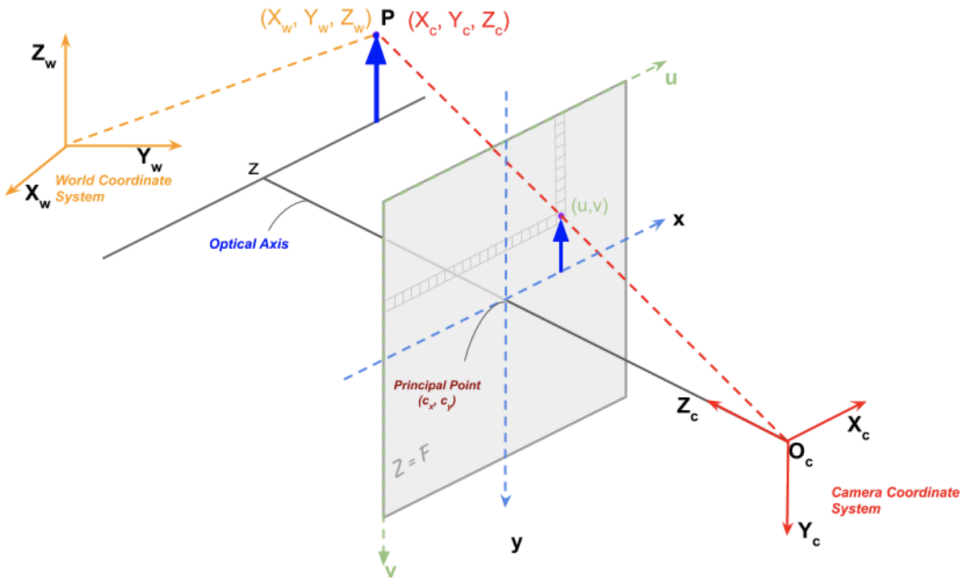


Figure 2.4: World coordinate systems (yellow) : Define the absolute position of 3D points.
Image coordinate systems (green) : The origin is located at up left corner and the unit of a pixel is determined by the image resolution.
Camera coordinate system (red) : The origin is located at camera center and Z axes is aligned to the principal axis.(picture is from Satya Mallick [23])

A camera projection matrix is consist of an intrinsic matrix and an extrinsic matrix which are respectively responsible for the transformation from a camera to an image and from world to camera coordinate systems. In this section, we introduce how to obtain camera projection matrix step by step. Start with the simplest example pinhole camera model. Afterward, how to get intrinsic matrix and extrinsic matrix and finally, how to combine them to calculate the camera projection matrix.

2.2.1 Pinhole Camera Model

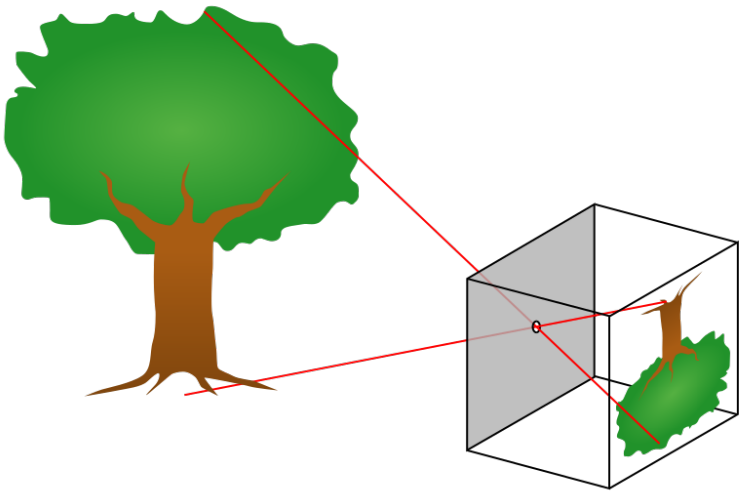


Figure 2.5: A diagram of a pinhole camera (picture is from Bob Mellish, Pbroks13 [3])

The pinhole camera model is the simplest model which describes the projection of an ideal pinhole camera, where the lens of the camera is a tiny aperture (See in Figure-2.5). Without the lens, the model does not take geometric distortions or blurring of unfocused objects into account. Light from a scene passes through the aperture and projects an inverted image on the opposite side of the box, which is known as the camera obscura effect. The projected image plane is perpendicular to light ray from the object and is located at distance of the camera focal length f from the aperture (See in Figure-2.6 left).

In order to simply the coordinate frame, we usually set the aperture as origin O in the center of the hole. The Z direction is along the axis perpendicular to the projected image plane. The X and Y directions are in the plane of the projected image plane. The projected image plane is the plane $Z = -f$. According to that the lengths of corresponding sides are proportional in similar triangles, the point $P = (X, Y, Z)$ in the world coordinates could be calculated to the image coordinates $p = (x, y)$ (See in equation-2.2).

$$x = -f\frac{X}{Z}, \quad y = -f\frac{Y}{Z} \tag{2.2}$$

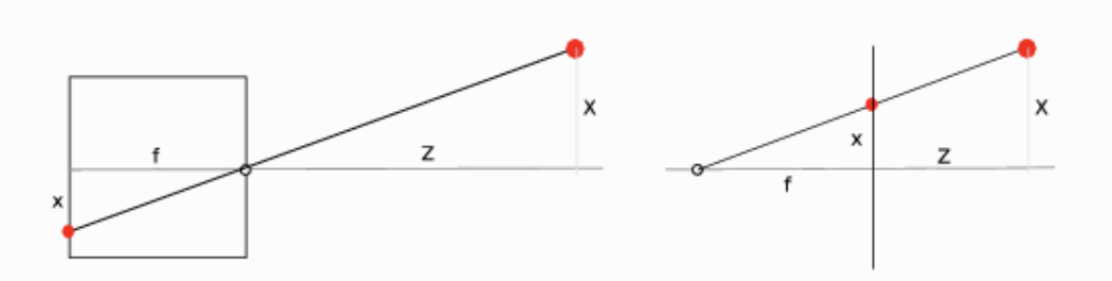


Figure 2.6: On the left a physical model of the pinhole camera, on the right the virtual model where the projection plane is put in front of the "pinhole".

The minus sign indicates that the projected image plane is on the back of the pinhole camera and the projected image is upside down. To get rid of this mirroring in our model, we usually use a virtual pinhole camera in which the the projected image plane is in front of the pinhole camera $Z = f$ (See in Figure-2.6 right). In this model we rewrite the equation to equation (See the matrix-2.3):

$$x = f\frac{X}{Z}, \quad y = f\frac{Y}{Z} \tag{2.3}$$

In the field of computer vision, this model is usually written in a matrix form to speed up the computation (See the matrix-2.4) :

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{2.4}$$

2.2.2 Intrinsic Matrix

In the practical application, we need to measure the coordinates in the pixel distance. The projected image plane in the pinhole camera model is still the 3D plane with the collection of points, not the real image taken by the digital camera. The computer image is the matrix with fixed pixel size and we usually define the origin is on the up left corner (See in Figure-2.7. Therefore we introduce

- (u_0, v_0) : indicate the image center in the image pixel coordinate system
- k_u : the number of pixels per mm along X axis.
- k_v : the number of pixels per mm along Y axis.

into the equation-2.2 and update the equation-2.2 to equation-2.5

$$x = u_0 + k_u f \frac{X}{Z}, \quad y = v_0 + k_v f \frac{Y}{Z} \tag{2.5}$$

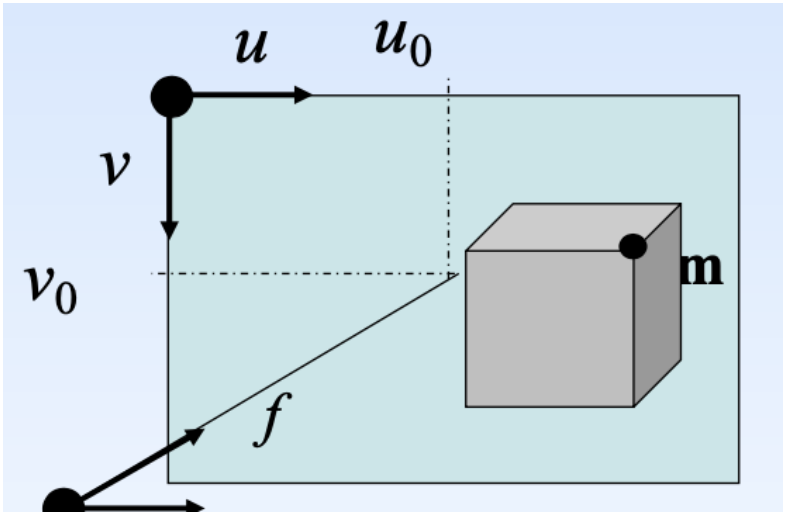


Figure 2.7: Illustration of image coordinate. The transformation into the image pixel coordinates.(u_0, v_0) is the center of the image, and (u, v) is the pixel size on X, Y directions. (picture is from Ilic Slobodan [17])

We could convert it to the matrix form (See the matrix-2.6) :

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} k_u f & 0 & u_0 \\ 0 & k_v f & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{2.6}$$

Finally, a skew factor α that accounts for a shear of the coordinate system is introduced into the matrix-2.6 . Thus we arrive at the following matrix (See the matrix-2.7):

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} k_u f & \alpha & u_0 \\ 0 & k_v f & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{2.7}$$

And call the matrix K intrinsic matrix (See the matrix-2.8) :

$$K = \begin{pmatrix} k_u f & \alpha & u_0 \\ 0 & k_v f & v_0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.8}$$

2.2.3 Extrinsic Matrix

We introduce the extrinsic matrix to deal with the camera relative position to the 3D object, because camera position won't be always in front of the 3D object and align at Z axis at the focal length distance. The extrinsic matrix is composed of a rotation matrix and a transition matrix. The rotation matrix indicates the rotation angle for camera looking in any arbitrary direction and the transition matrix describe the the camera relative position and distance to the origin in world coordinate system.

A basic rotation is a rotation about one of the axes of a coordinate system. The following three basic rotation matrices rotate vectors by an angle θ about the X, Y, or Z axis, in three dimensions (See the matrix-2.9).

$$
\begin{aligned}
R_x(\theta) &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \\
R_y(\theta) &= \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \\
R_z(\theta) &= \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}
\end{aligned}
\tag{2.9}
$$

By matrices multiplication of basic rotation (rotate angle γ, β, α), we could reach any angle in the 3 * 3 matrix R (See the matrix-2.10).

$$
R = R_z(\alpha)R_y(\beta)R_x(\gamma) = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix} \tag{2.10}
$$

Add the transition vector T, we get extrinsic matrix (See the matrix-2.11) which is parameterized by 6 values: 3 for the rotation, 3 for the translation.

$$
(\mathbf{R} \mid \mathbf{T}) = \begin{pmatrix} \mathbf{R}_{11} & \mathbf{R}_{13} & \mathbf{R}_{13} & \mathbf{T}_1 \\ \mathbf{R}_{21} & \mathbf{R}_{22} & \mathbf{R}_{23} & \mathbf{T}_2 \\ \mathbf{R}_{31} & \mathbf{R}_{32} & \mathbf{R}_{33} & \mathbf{T}_3 \end{pmatrix} \tag{2.11}
$$

In order to multiply the $3 * 4$ extrinsic matrix to a 3D point, it is often to convert the point from Cartesian to its homogeneous coordinates by adding an additional

dimension and multiply all values by a common non-zero scalar value k:

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \quad \rightarrow \quad \begin{pmatrix} kX \\ kY \\ kZ \\ k \end{pmatrix} \tag{2.12}$$

To go back to the 3-vector, one just has to divide the last one k which is defined up to a scale factor.

2.2.4 Camera Projection Matrix

The full transformation, Camera Projection Matrix from a 3D point in the world coordinate system to its projection in the image could be reached by the chain of aforementioned intrinsic matrix and extrinsic matrix (See the equation-2.13).

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} k_u f & \alpha & u_0 \\ 0 & k_v f & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{R}_{11} & \mathbf{R}_{13} & \mathbf{R}_{13} & \mathbf{T}_1 \\ \mathbf{R}_{21} & \mathbf{R}_{22} & \mathbf{R}_{23} & \mathbf{T}_2 \\ \mathbf{R}_{31} & \mathbf{R}_{32} & \mathbf{R}_{33} & \mathbf{T}_3 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{2.13}$$

Here u, v, w are the homogeneous representation of the camera coordinate x, y with a non-zero scalar value w:

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} u/w \\ v/w \\ 1 \end{pmatrix} \tag{2.14}$$

After the multiplication, is written in the equation-2.15

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{2.15}$$

Matrix P is called camera projection matrix.

2.3 Neural Network

This section introduces the fundamentals of neural network. With the advanced GPU and memory, the neural network have achieved astonished results in many tasks in

different areas including 3D reconstruction. Neural network is a series of algorithms that endeavors to find the relationships in a set of data through a process. Inspired by the biological neural networks such as human brain, neural network is compose of connected units or nodes called perceptron, which loosely model the neurons in a biological brain.

Each connection, similar to the neurons in a biological brain (See the Figure-2.8), propagates a signal to other neurons. A neuron receives a signal then processes it and outputs the signal to the connected neurons as input. The signal at a connection is a real number, passed to the neuron doing the multiple linear regression, and the output of each neuron is computed by non-linear function. The non-linear function describes the threshold of the propagating signal. A signal is sent to other neurons only if the aggregate signal crosses that threshold.

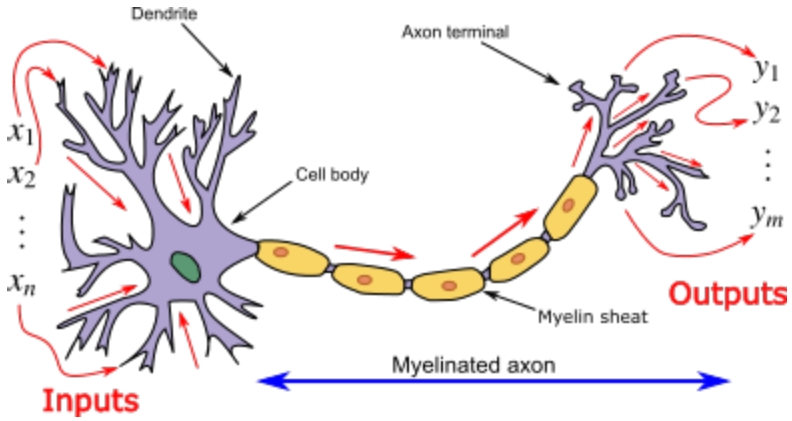


Figure 2.8: Neuron and myelinated axon, with signal flow from inputs at dendrites to outputs at axon terminals which is the prototype of neuron in neural network (picture is from Prof.Loc Vu-Quoc [33])

Typically, neurons are aggregated into layers. All neurons connect to all neurons in the next layer which is the basic neural network layer called fully-connected layer. Different layers may perform different transformations on their inputs. By adding more layers and more units within a layer, a deep network (See the Figure-2.9) can increase model's complexity.

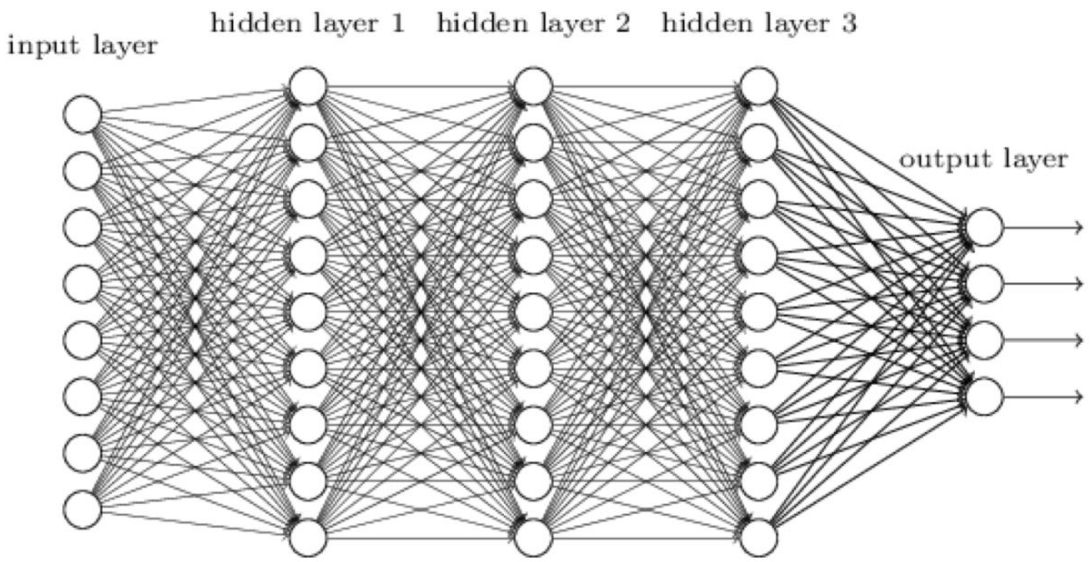


Figure 2.9: Illustration of neural network architecture. Neurons usually pile up as layers. Fully-connected layer is the simplest layer which shows each node has connections to the nodes in next layer. Signals go through all layers from input, hidden layers to output layer. (picture is from Prof.Leal-Taixé [18] and Prof.Nießner [27])

2.3.1 Perceptron

A neuron in neural network called perceptron is a mathematical function that combines a simple linear regression and follows a non-linear function (See the Figure-2.10). The non-linear function provides the non-linearity into the network. Otherwise, without the non-linear function, no matter how many perceptrons the network has, would behave just similar a single perceptron, because summing these perceptrons would give you just another linear function. Besides non-linearity, the non-linear function also plays the role of a threshold to determine the output passed to the next perceptron only if it is larger than the threshold. Perceptron takes all the input values $X_1, ..., X_n$ plus one constant bias and usually write in the vector form $X = [X_1, ..., X_n, bias]$. Similar to linear regression, input vector multiply with their weights $W = [W_0, ..., W_n]$. Then, all of these multiplied values are summed to the weighted sum. The weighted sum is then sent to the activation function as input, producing the output. The activation function plays the integral role of ensuring the output is mapped between required intervals. The whole process could be written in the equation-2.16.

$$\begin{aligned} &\textit{Relu as activation function} : f(x) = max(0, x) \\ &\textit{Perceptron output} : f(WX) \end{aligned} \tag{2.16}$$

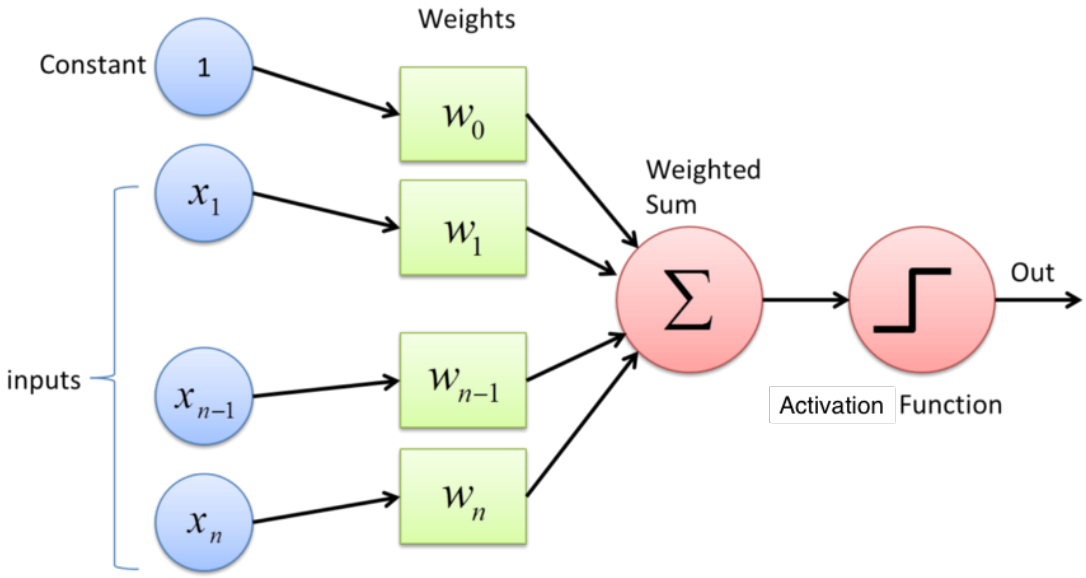


Figure 2.10: Illustration of perceptron architecture. A linear regression with a non-linear activation function form a perceptron output. (picture is from Sagar Sharma [38])

It is important to note that the charateristic of perceptron resemble linear regression. The weight is indicative of the strength of the input. Similarly, bias value of an input gives the ability to shift the function curve up or down.

2.3.2 Optimization

To optimize the whole neural network which has thousands of perceptrons through training process. First, we introduce the function called loss function or cost function which maps the weights and biases in the neural network to their associated costs. Our goal is to find the corresponding biases and weights in the network which could minimize the loss function.

Then we regard the whole neural network as the convex function and apply the mathematical technique called gradient descent to update the weights in the network iteratively. The intuition of finding minimum of convex function is computing derivatives and then using them to find places where is an extremum of function. However, the neural network has thousands of weights and variables which is too complex to get the analytics derivative. Gradient descent is an alternative way to find the minimum by updating the weights iteratively. The idea is to take a small step in the opposite direction of the gradient of the function at the current weight setting (See Figure- 2.11). The weights update themselves by subtracting the multiplication of step size with the steepest descent. Iteratively, stepping in the opposite direction of the gradient will lead to the minimum of that function. Iteratively, stepping in the opposite direction of the gradient will lead to the minimum of that function.

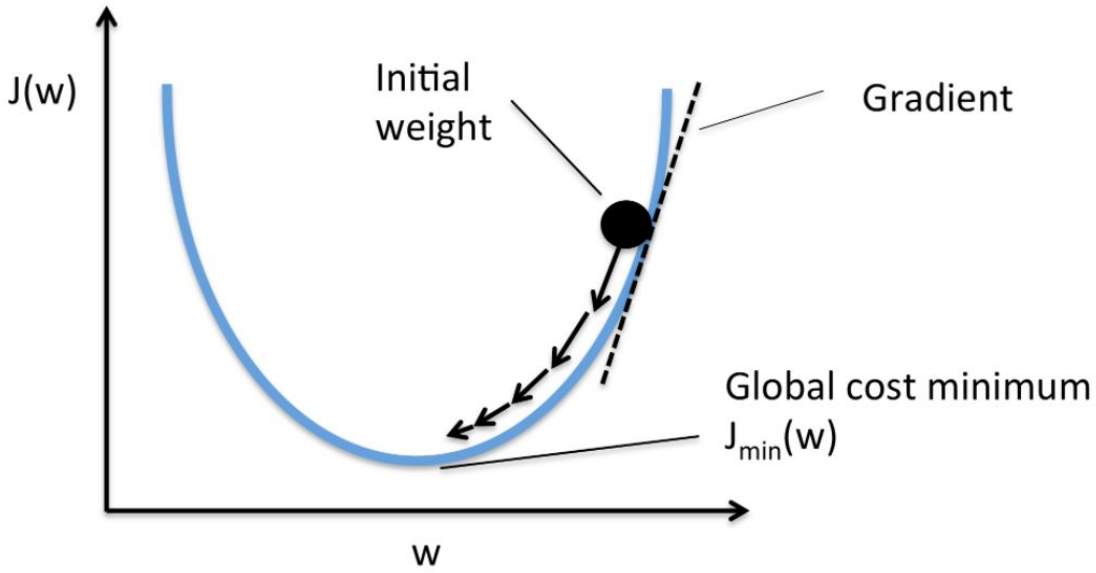


Figure 2.11: Illustration of gradient descent. $J(w)$ indicates the loss of corresponding weight w. In n iteration, compute the gradient of w and get $\nabla J(w)$. Following the steepest direction take a small step δ and update weights by $w_n = w_{n-1} + \delta * \nabla J(w)$. (picture is from Sebastian Raschka [34])

To optimize iteratively via gradient descent, the whole neural network is usually designed to be differentiable including activation function. We could compute the gradient of the loss function with respect to parameters by the chain rule and propogate the gradient back from the last layer to the first layer. This process is known as

backpropogation, short for backward propagation of errors, which makes it feasible to use gradient methods for training multi-layer networks, updating weights to minimize loss. Here is an example at Figure 2.12.

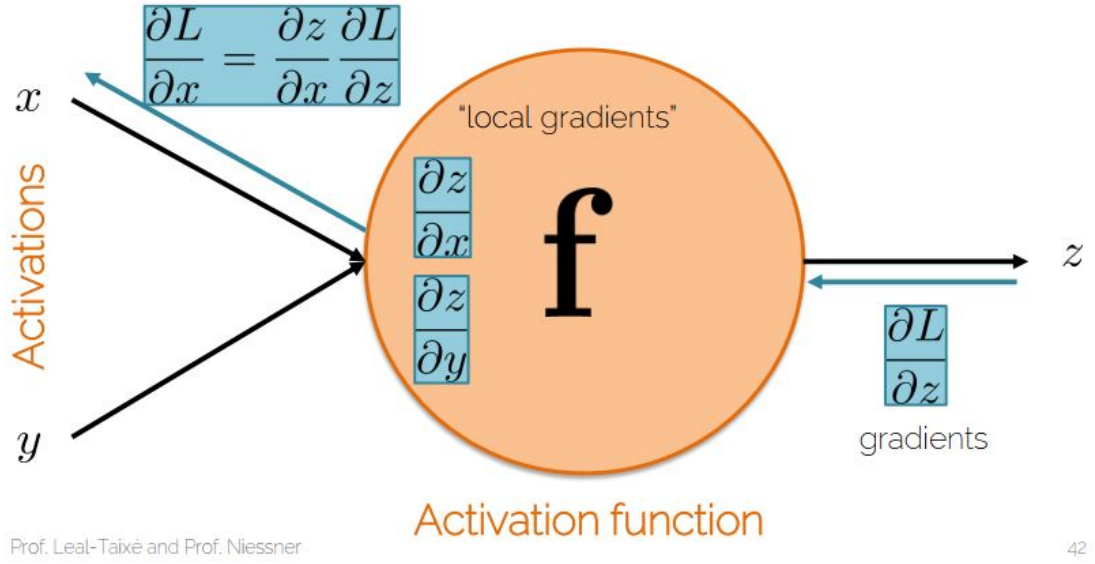


Figure 2.12: Illustration of backpropagation. The image shows a perceptron in the neural network during backpropogation. L is Loss function, z is output weight of perceptron, and x, y are the input weight of perceptron. The grdient of z is computed by $\frac{\partial L}{\partial z}$ and gradient of x could be calculated by chain rule $\frac{\partial L}{\partial x} = \frac{\partial z}{\partial x} * \frac{\partial L}{\partial z}$. Similarly, all gradients of weights could be computed in the same way. (picture is from Prof.Leal-Taixé [18] and Prof.Nießner [27])

2.3.3 Convolutional Layer

Convolutional layer is a type of neural network layer which most commonly applied to the visual image. Combine the mathematic convolution into the neural network architecture to extract image features with the translation, size, and rotation invariant characteristics. Convolutional layer replaces the fully-connected layer in the computer vision tasks because the fully-connected layer architecture was found to be inefficient. Images represent a large input for a neural network. Take an example of a colorful image with the height 255, width 255 pixels, the input needs 255 *255 (pixels)* 3 (r,g,b 3 channels) = 195075 size in a fully-connected layer. This requires a huge number

of variables, weights, and network parameters for training. A convolutional layer leverages the fact that an image is composed of smaller details such as edges, local features, and creates a mechanism to analyze each feature independently.

Before we introduce the detail of convolution layer in the neural network, we start with mathematics convolution which is widely used in the traditional image processing tasks. The process of convolution uses a filter or kernel to extract certain features from an input image. The general expression of a convolution is equation-2.17

$$g(x,y) = \omega * f(x,y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} \omega(dx,dy) f(x+dx, y+dy), \tag{2.17}$$

where $2a+1$ indicates the width of the filter and $2b+1$ is the height of the filter. $g(x,y)$ is the output image, f(x, y) is the original image, ω is the filter. Every element of the filter is considered by $-a \leq dx \leq aand - b \leq dy \leq b$.

Here is an example of a convolution operation (See the Figure-2.13).

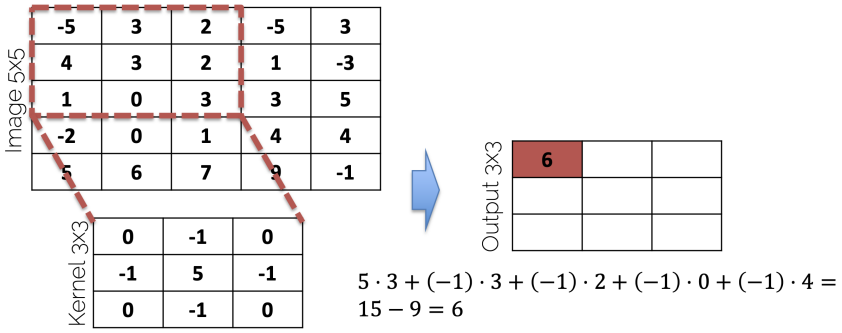


Figure 2.13: Illustration of convolution computation. Input image size is 5 * 5 and filter size is 3 * 3. Each parameter in the filter computes the dot product with the corresponding pixel value in the image and sum them as the output. filter window slides over the whole image and repeats the same process to get the output image (picture is from Prof.Leal-Taixé [18] and Prof.Nießner [27])

As picture shown in Figure-2.13, the convolved output image (3*3) reduces the dimension compared to the input image (5*5). For the pixels near the image boundary are not applied the convolution thoroughly. In order to keep the information of pixel around the edges, we fill up the extra pixels around the boundary of input image which

is called padding (See the Figure-2.14). There are 3 most commonly used modes of padding, zeros, reflect, replicate.

- Zero padding : pad all extra pixels with zero.
- Reflect padding: pad pixels are computed by reflecting the input image.
- Replicate padding: replicate the values of edge pixels as value of extra pixels.

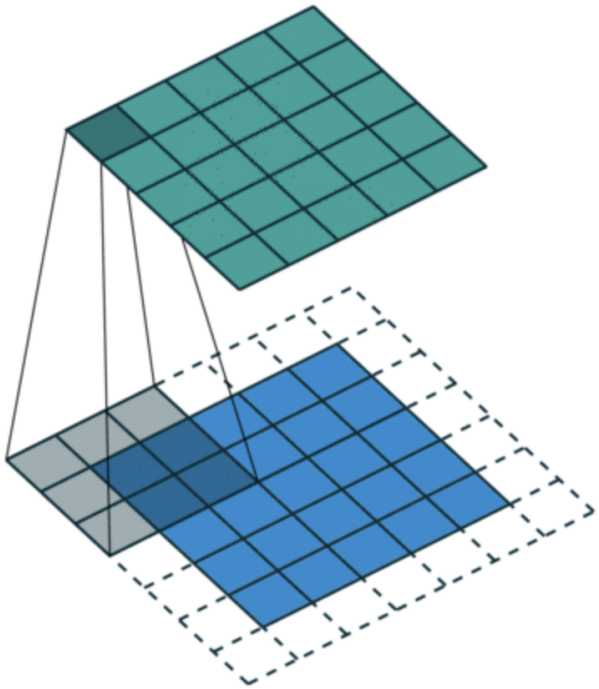


Figure 2.14: Illustration of padding image. Typically, we add the extra pixels around the boundary to reserve dimension and the information of edge. we usually pad the value zero to the extra pixels (picture is from Sumit Saha [36])

After padding , filter slides across the whole image from left to right, up to down. For every input image pixel, filter aligns the center to calculate the convolution. Then output an image with the desired effect such as blurring, sharpening, and edge detection (See the Figure-2.15).
Instead of manually determining the filter parameters, convolutional layer considers filter parameters as neural networks weights which are updated iteratively through

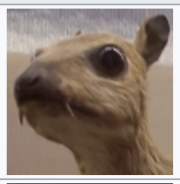
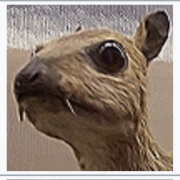
Operation	Kernel ω	Image result g(x,y)
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur 3 × 3 (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Figure 2.15: Examples of effects achievable by convolving filters and images. (pictures are from Michael Plotke [32])

gradient descent. Convolutional layer trains multiple filters at the same time and stacks

the 2-dimensional convolved outputs called activation map or feature map along the channel dimension forming the 3-dimensional volume (See the Figure-2.15). Then the output volume could be passed to the next convolutional layer as input.

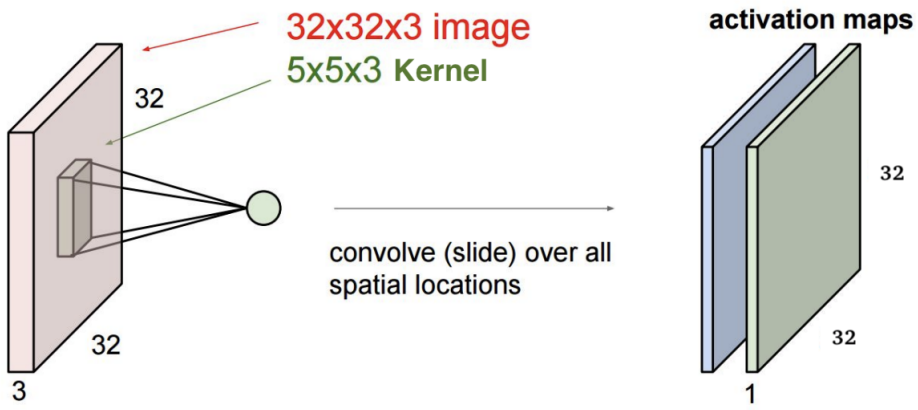


Figure 2.16: Illustration of process of convolutional layers. Each filter provides a 2-dimensional activation map. Stack all the activation maps convolved from filters forms 3-dimensional volume. (picture is from CS231n class of Stanford [40])

We could pile up the convolutional layers to increase the complexity of the neural network. In the convolution process, filters exploit spatial locality by doing dot product with all surrounding input pixels. This process ensures that the learned filters produce the strongest response to a spatially local input volume called the receptive field. After forward passing through multiple layers, each output value at the activation map has larger receptive field at original input image (See in the Figure-2.17).

This means the trained multiple convolutional layers is capable to extract the high-level features. Conventionally, the first few convolutional layers are responsible for capturing the low-level features such as edges, corners, gradient orientation, etc. With added layers, the architecture extract the high-level features as well (See the Figure-2.18). The high-level features allow the neural network has better image understanding to achieve astonishing results on computer vision tasks.

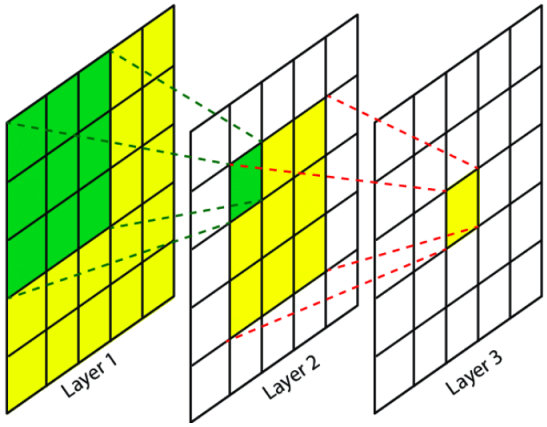


Figure 2.17: Illustration of receptive field in layers. In convolutional layers, each neuron is connected to only a small region of receptive field. Through layers, the deeper layer has larger receptive field at original input image. (picture is from [21])

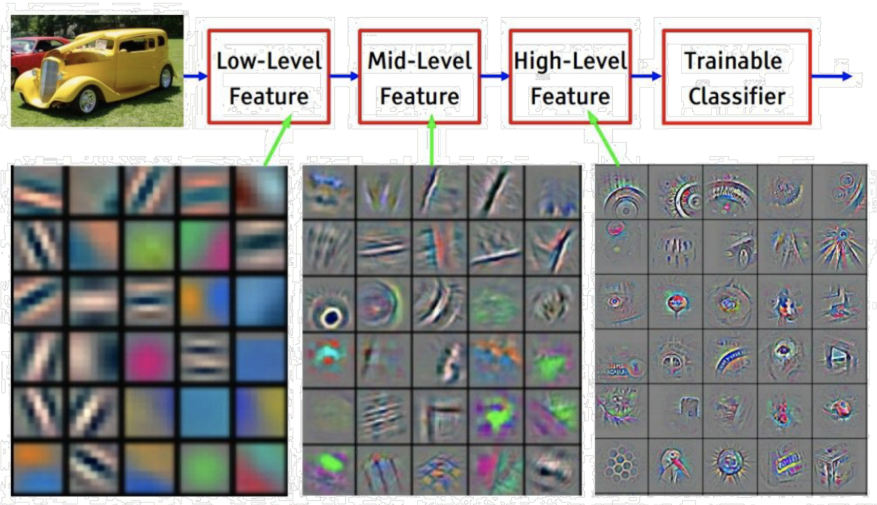


Figure 2.18: Illustration of extracted features in layers. Shallow layers respond to corners and other edge/color conjunctions. Mid layers have more complex invariances, capturing similar textures. Deeper layers extract significant variation. (pictures are from [49])

2.3.4 Pooling layer

Similar to the convolutional layer, pooling layer applies filter sliding across the input volume to reduce the spatial size , which is a form of non-linear down-sampling (See the example in Figure-2.19). Dimension reduction is not only decreasing the number of computations,but also is useful for extracting dominant features which are positional invariant.

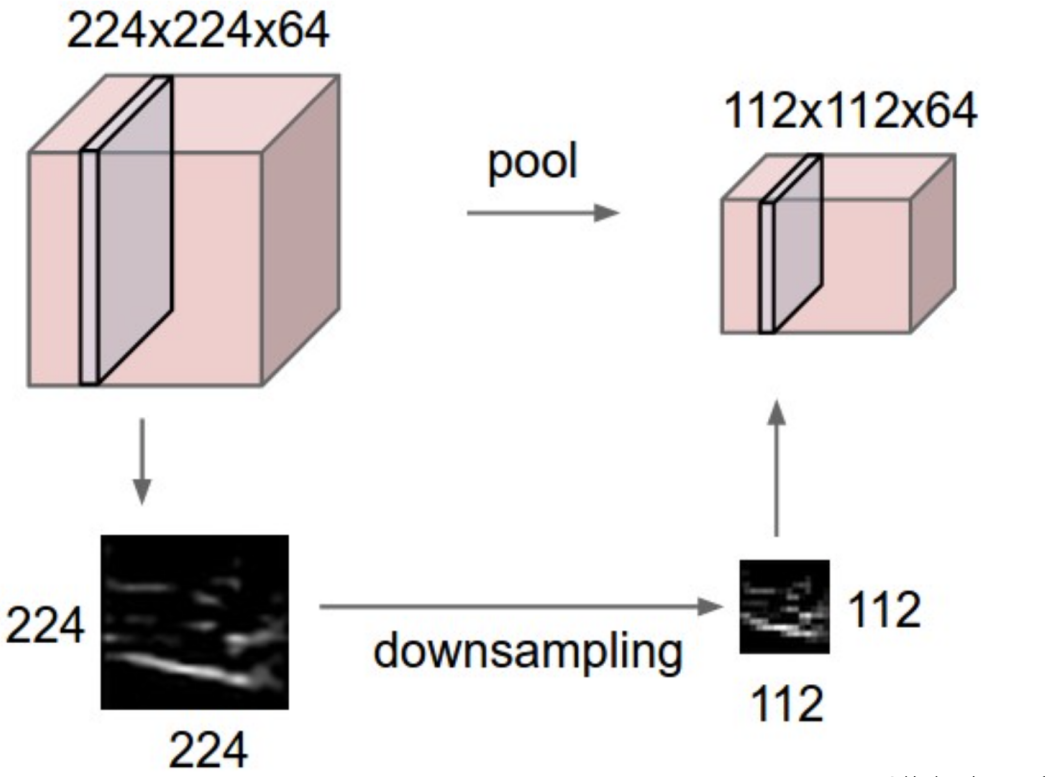


Figure 2.19: Example of pooling layers operation. Input volume size is (224*224*64). After forward passing through a pooling layer with a filter size 2*2 and a stride of 2, volume size is shrunk to (112*112*64). The output volume always remains the depth dimension which represents the number of filters. (picture is from Prof.Leal-Taixé [18] and Prof.Nießner [27])

The filter of the pooling layer does not contain learnable parameters. It is the fixed-function which operates independently on every depth slice of the input. The most common form is max pooling which returns the maximum value from the portion of

the image covered by the filter. See the example in the Figure-2.20 with filters of size 2*2 applied with a stride of 2 (window sliding step size) downsample at every depth slice in the input by 2 along both width and height, discarding 75% of the input size.

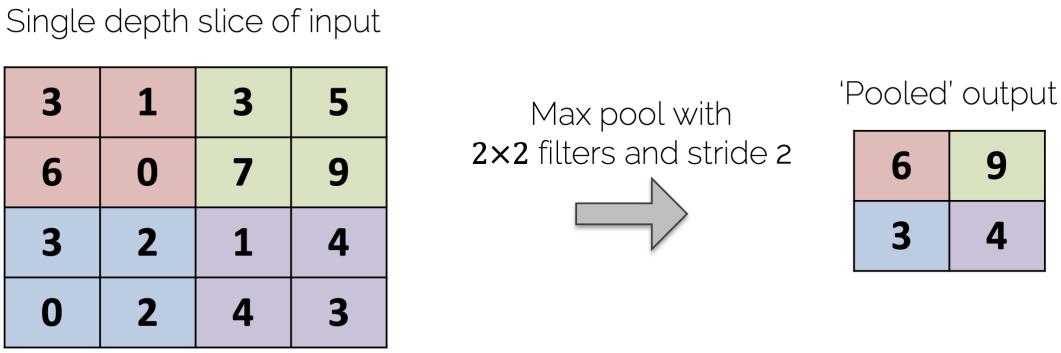


Figure 2.20: Illustration of Max pooling layers. In this case, every max operation is over 4 numbers. The depth dimension remains unchanged. (picture is from Prof.Leal-Taixé [18] and Prof.Nießner [27])

In addition to max pooling, pooling filter can use other functions, such as average pooling or $l2$-norm pooling. In the practice, Max Pooling is most commonly applied, because it also performs as a Noise Suppressant. It discards the noisy activation maps altogether and also performs de-noising along with dimensionality reduction.

2.3.5 Convolutional Neural Network

Convolutional Neural Network (CNN) consists of convolutional layer, pooling layer for image understanding related tasks. The architecture usually uses several convolutional layers followed by the pooling layer as a block for image feature extraction. After multiple feature extraction blocks, flatten the output high-level features and pass them to the fully-connected layers to find the relationship between image pattern and ground true in corresponding computer vision tasks.

Take the image classification task for example (See the Figure-2.21). A unit consists of a convolutional layer followed by a Relu activation function where the Relu function is a non-linear function that outputs the input directly if it is positive, otherwise, it outputs zero. We pile up several units and add the pooling layer in the end, together forming the block. Concatenate several blocks to extract the image features.

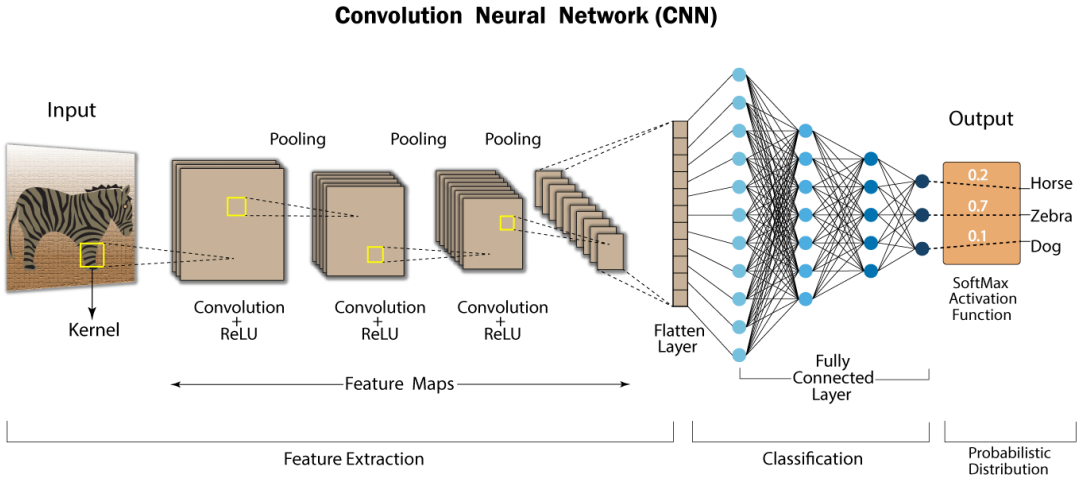


Figure 2.21: Architecture of Convolutional Neural Network in the image classification task. Typically, we group multiple convolutional layers + ReLU followed pooling layer for feature extraction. The deeper of the convolutional layers, the more filters are assigned to learn the more abstract features. After building the feature maps, flatten the volume and pass them to fully-connected layers for class prediction. The activation function of the last layer is SoftMax which outputs the probability of each class in a vector. (pictures are from Swapna K E [8])

After going through the above process, the model extracts the high-level features. We have to make the model learn the relationship between these features and the class of item. Using fully-connected layers is an intuitive way of learning non-linear combinations of the high-level features as represented by the output of the convolutional layer. The activation function of the last layer is SoftMax function (See the equation-2.18). The SoftMax function takes a vector z of K real numbers as input, and normalizes it into a probability distribution. After applying SoftMax, each component will be in the interval [0,1], and the components will add up to 1, so that they can be interpreted as probabilities.

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \text{ for } i = 1, \ldots, K \text{ and } \mathbf{z} = (z_1, \ldots, z_K) \tag{2.18}$$

The last layer outputs a [number of items * 1] vector with probability. The vector position with the highest value is the prediction of CNN.

2.3.6 VGG Net

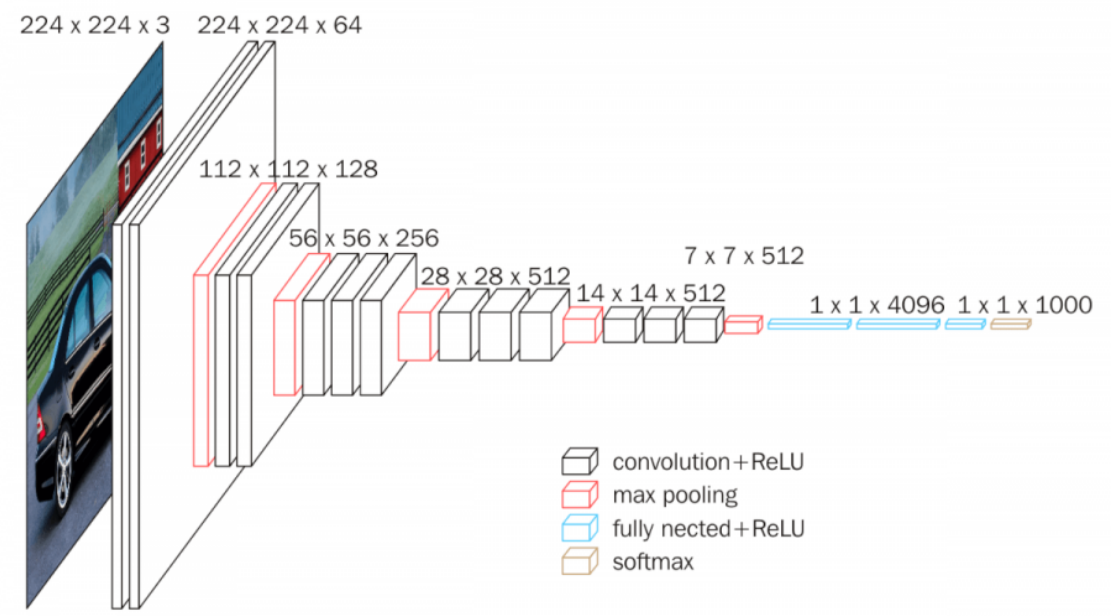


Figure 2.22: The overview of VGG16 [39] architecture. The input to VGG16 [39] is fixed to size 224 x 224 RGB image. The image is passed through a stack of convolutional layers, where the filters were used with size 3*3. The convolution stride is fixed to 1 pixel. Max pooling is performed over a 2*2 filter, with stride 2. Fully-connected layers follow a stack of convolutional layers. All hidden layers are equipped with the ReLU non-linearity. (pictures are from Muneeb ul Hassan [14]

VGG Net [39] is a convolution neural net architecture which is considered to be one of the excellent vision model architecture for image classification task. Most special thing about VGG Net [39] is that instead of using different hyper-parameters combination of filter such as size or strides they fixed the most of parameters. They applies convolution layers of 3x3 filter with a stride 1 and always used same padding and max pooling layer of 2x2 filter of stride 2 (See the Figure-2.22). It follows this arrangement of convolution and max pool layers consistently throughout the whole architecture. Therefore, we use the number of layer to indicate the model which follows this arrangement. For example, VGG16 [39] refers to it has 16 layers and applies to VGG model architecture. The detail of other configurations are shown in Figure-2.23.

The VGG Net [39] achieves 92.7% top-5 test accuracy in ImageNet [7], a dataset of over 14 million images belonging to 1000 classes. VGG Net [39] is also applicable to other image recognition datasets such as PASCAL VOC-2007 [9], VOC-2012 [10] and Caltech-256 [13]. The generalized ability of image feature extraction is popularly used in other computer vision related tasks. Thus, the primary machine learning libraries including TensorFlow [2] and PyTorch [31] provide pre-trained VGG Net models.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 2.23: VGG Net [39] configurations (shown in columns). The number layer increases from the left (A) to the right (E) (the added layers are shown in bold). The convolutional layer parameters are described as "conv(receptive field size)-(number of channels)". (picture is from [39])

3 Approach

Given a single image observation of an object and a camera matrix, our goal is to reconstruct the textured 3D geometry of the object. We inherit DISN [47] applying implicit function which represents a surface by a function over all space. To recover surface texture, we add color predictions aligned to the predicted 3D shape. Our approach regards the whole neural network as an implicit function which assigns the binary signed distance value with an RGB color values to any spatial point p $p(x, y, z) \in \mathbb{R}^3$ in 3D space. To extract a mesh object, we sample a points cloud and query all the SDF values of points in the points cloud. After getting the SDF value for every point p $p(x, y, z)$ in the points cloud, we could apply SDF-to-Mesh related algorithm such as Marching Cubes [22] to extract the 3D mesh. Besides SDF values of points, predicted color value outputs could fill the color texture of the 3D shape.

The whole process begins with features extraction from an image. Then, the features are passed to the rest of network. Finally, our model firstly predicts the binary SDF value and then follow by the color values, so that the color predictions can be directly informed by the geometry information, and the predicted geometry will not be interfered by the color predictions.

For the features extraction, we use a pre-trained model, such as VGG Net [39] and ResNet [15], as an image encoder to extract a global image features. In addition, we project each input point p onto the image and collect multi-scale local features, extracted from different levels of the feature encoding layers inside the image encoder, to preserve the local details.

For the geometry, our neural network predicts a binary signed distance value $[-1, 1]$ which indicates whether an input point p = (x, y, z) is inside an object surface. The iso-surface $S_0 = \{p|SDF(p) = 0\}$ implicitly represents the underlying 3D shape in the continuous space which results in a memory-efficient representation of an object. The continuous space in the surface is embedded and does not need to be explicitly stored. In contrast to the explicit representation such as points cloud, voxels, and meshes, our model produces a continuous field with arbitrary resolution.

For color texture, inspired by PIFu [37] and SPSG [6] which input the prediction of geometry information to optimize the color texture prediction, we also consider the prediction of geometry as the input to predict the RGB values. After knowing the relative corresponding geometry between the input image and each 3D point p, the network is able to learn the texture information from the image. For example, if the 3D point p is in the occluded area from the camera perspective, the network will not learn any information from the input image or the local feature. Likewise, the network will learn the color information if the 3D point p is shown in the image. Instead of two stages of training such as PIFu [37] trains geometry network first, then uses the predicted signed distance value and the extracted features to train the color texture network, we directly train an end-to-end network for both geometry and color texture prediction.

3.1 Architecture

The overview of our method is illustrated in Fig 3.1. Given an image of an object and a camera matrix, our method reconstructs the textured 3D shape by iteratively querying all the points in the sampled points cloud.

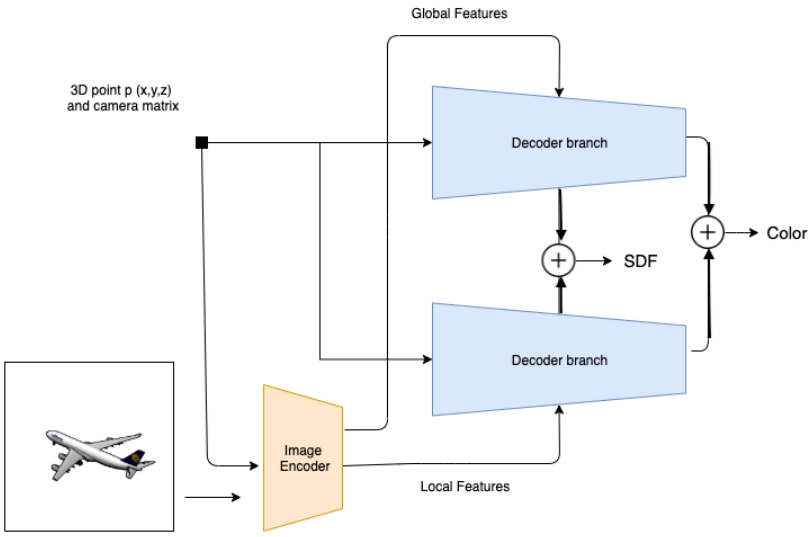


Figure 3.1: Illustration of our network. Our network is compose of an image encoder and two decoder branches. For each 3D point p $p(x, y, z)$ in the sampled points cloud, our network takes the given image and camera matrix as input to predict the SDF and RGB values.

For each point p (x, y, z) in the sampled points cloud, the image encoder extracts the global features. We use the camera matrix to project p onto the image plane. After getting the projected location, we identify it on the feature maps of the image encoder and extract it as the local features of point p. Then, both global and local features with the 3D location of point p are passed into two decoding branches respectively for SDF and color predictions. Each branch will output the SDF value and color values. We sum two SDF values and color values separately as a final prediction to ensure our model takes both global and local features into account.

The architecture of the decoder branch is shown in Figure-3.2. Both two decoder branches are fully-connected neural networks and each layer is implemented by a 1D convolutional neural network with the Relu activation function. Each branch consists of the geometry decoder and the color texture decoder. The prediction of SDF value is passed to a color texture decoder as input to make sure the RGB prediction will consider the shape geometry and camera viewpoint.

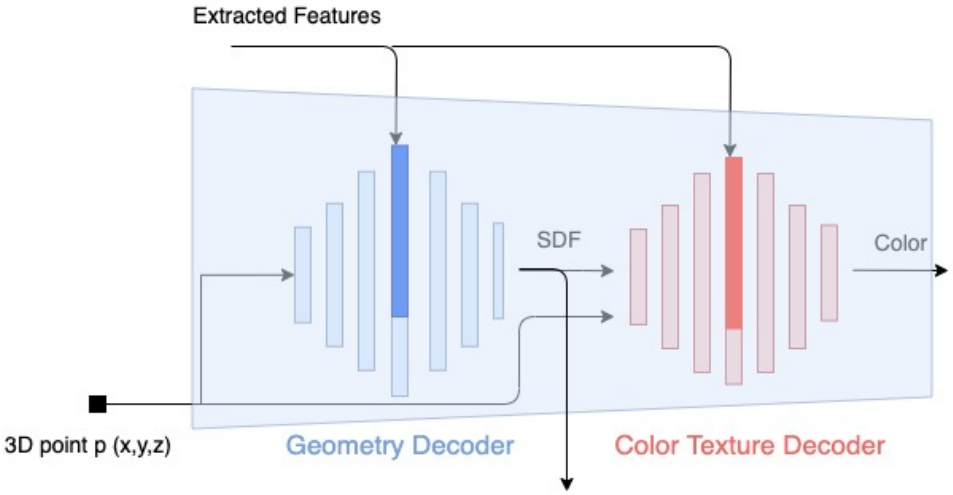


Figure 3.2: Illustration of decoder branch. Pass the 3D location of the point p $p(x, y, z)$ and the extracted features from the image encoder. The extracted features are either local or global image features. We highlight the merged layers to indicates that in additional to encoder increases the dimension of output, the extracted features are also merged to enlarge the dimension

Both geometry decoder and the color texture decoder are encoder-decoder structures and take 3D point coordinate position and extracted image features as input. We have bath size B, number of N sample points and x, y, z position of each sample point. Then the input dimension is $[B * N * 3]$ which will be transposed to $[B * 3 * N * 1]$. The 3 axis coordinates are regarded as the number of input channels in the 1D convolutional layer. After input the 3D position, the encoder increases filters layer by layer until concatenate extracted image features. Then decrease filters to make the prediction. The color texture decoder is similar to the geometry decoder except the geometric predictions will be passed to the color texture decoder for the color prediction.

3.2 Local Feature Extraction

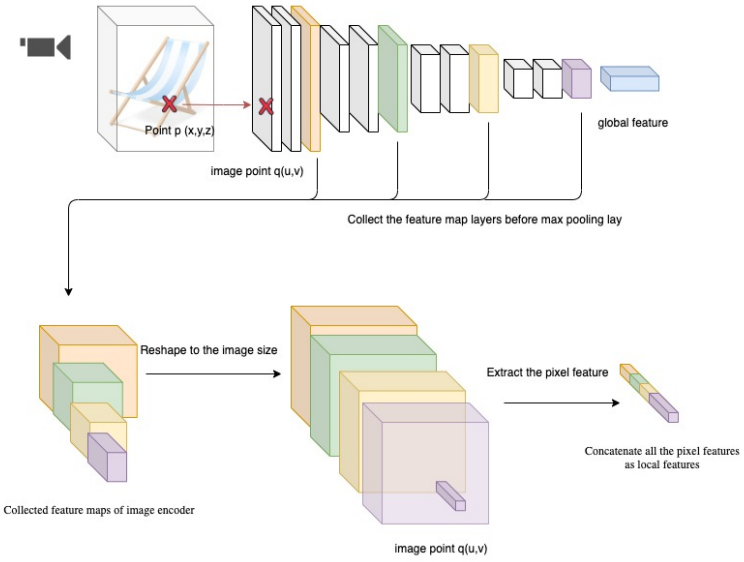


Figure 3.3: Illustration of local feature extraction. Pass the point p $p(x, y, z)$ and a camera matrix, we could get the project point $q(u, v)$ in image coordinate. For each feature map, we reshape the size to the same size as the input image. According to the project point $q(u, v)$, we could extract the pixel features from each resized feature map. We concatenate all the pixel features as the local features.

After getting the global features by a forward pass through the image encoder, We extract local features from feature maps of the image encoder to capture the detail of the object. Firstly, when extracting the global features, we record all the output

feature maps from convolutional layers of image encoder. Then, using the given camera matrix projects the point p (x,y,z) to the image coordinate. For each feature map, we interpolate the size to original input image by bilinear sampling and extract the pixel features by the projected image coordinate. Concatenate all the pixel features as the local features forward to the rest of network (See the whole process in Figure-3.3).

3.3 Loss Function

The loss function is composed of geometry shape and color as well. In the geometry part, Our model predicts whether the point is inside the object as the decision boundary of a deep neural network binary classifier. We apply cross-entropy as the loss function. In the color part, we minimize the error by $L1$ loss which is the sum of the all the absolute differences between the ground true color values and the predicted color values.

3.3.1 Cross-entropy loss

Cross-entropy loss measures the performance of a classification model whose output is a probability value between 0 and 1.The true probability p is the true label, and the given distribution q is the predicted probability value of our neural network model. The probability $\hat{y}$ indicates the point is inside the object. Conversely $1-\hat{y}$ is the probability of being outside the object. Having set up our notation,$p \in \{y, 1-y\}$, $q \in \{\hat{y}, 1-\hat{y}\}$, the equation of cross-entropy is defined as equation 3.1. We can use cross-entropy to get a measure of dissimilarity between p and q.

$$\begin{aligned} L_{shape}(p,q) &= -E_p[\log q] \\ &= -\sum_{i=1}^{N} p_i \log q_i \\ &= -\sum_{i=1}^{N} y \log \hat{y} - (1-y)\log(1-\hat{y}) \end{aligned} \tag{3.1}$$

Cross-entropy loss increases at a very high rate when the wrong prediction occurred. The Figure 3.4 shows relationship between loss and predicted probability in cross entropy given a true label equal to 1. As the predicted probability approaches 1, loss slowly decreases. When the predicted probability declines, however, the loss rises rapidly. Cross-entropy loss penalizes errors efficiently. Thus, Both DISN [47], Occupancy Network [25] apply it as loss function to train the whole network. We also

use Cross-entropy loss for our neural network to predict the binary SDF value of points.

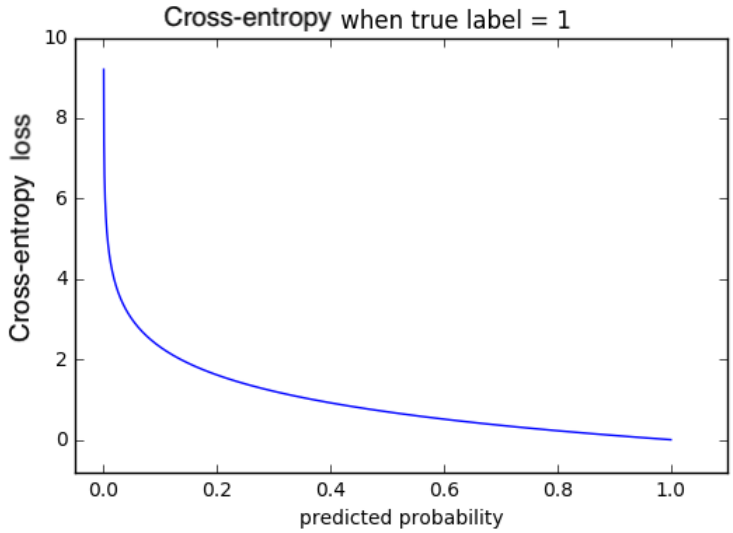


Figure 3.4: Illustration of relationship between loss and predicted probability in cross-entropy. With the log function, cross-entropy loss penalize the wrong prediction efficiently and effectively

3.3.2 Color loss

For the textured color part, inspired by SPSF [6] and PIFu [37] which utilize L1 loss as one of the losses in 3D model color reconstruction task. Therefore, we simply apply the L1 loss to compare the difference between prediction and ground true. L1 loss is written in equation 3.2 which sum the absolute difference between each ground true color value C_i and predicted color value $\hat{C}_i$.

$$L_{color} = \frac{1}{N}\sum_{i=1}^{N} \left\| C_i - \hat{C}_i \right\| \tag{3.2}$$

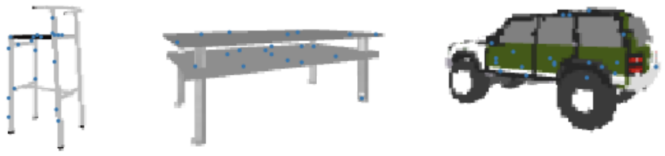
Our image input is between the interval [0,1] in order to emphasize the color training, we assign the weight 255 to color loss. The overall loss L_{loss} is equation-3.3

$$L_{loss} = L_{shape} + 255 * L_{color} \tag{3.3}$$

3.4 Implementation Detail

During the training, the sampled points of the object and the input images with the given camera matrix are passed through the mentioned deep neural network model (See the Fig3.1). However, in practice, the points of the object are too many to fit in the GPU memory at once. For each sample, we randomly sample points to partially train the network. The selected points are projected to the input image for collecting the positions of the local features (See the example in Figure-3.5).

(a) Original image



(b) Re-projected points

Figure 3.5: Illustration of projected points. All the selected points are projected to the image plane for local feature extraction. Figure(a) is the input image. Figure(b) shows the example of 20 projected points. Blue points indicate the position of projected 3D points

We use the pre-trained VGG16 [39] (The D type in Figure 2.23) as the image encoder to extract features. The feature map layers of numbers 1, 3, 6, 9, 12 are retrieved and enlarged through bilinear interpolation to size [138*138] the same height and width as the input image. According to the projected positions of points, we extract the pixel features from these five feature map layers of pre-trained VGG16 [39]. The extracted pixel features are concatenated as the local feature. For the global features, we fine-tune

the last few fully-connected layers in VGG16 [39] and take output as the global features. In DSIN [47] and Occupancy networks [25], with only global features, the network is able to predict the overall shape. We inherit this idea, In a similar fashion, the global features should capture global shape properties and general color texture. We multiply the global features by the number of sampled points so that every picked point corresponds to the same global features.

After getting the global and local features, we pass them separately to the decoder branch (See the Figure-3.1) with the 3D location of points. In each decoder branch, we use the method of Plfu [37] and SPSG [6] which take advantage of geometry information to optimize the color texture prediction. Thus, decoder branch uses predicted SDF value as input to make RGB color predictions.

3.5 Training

Our network is implemented by TensorFlow [2]. For each sample, we randomly pick 2048 points from the points cloud of the object. For training, we use the Adam optimizer with a learning rate of $1 * 10^{-4}$ and a batch size of 20 . The whole network took 4 days for training by GeForce GTX TITAN X 12G.

3.6 Surface Reconstruction

To generate a colorful mesh, we firstly define the resolution of a dense 3D grid. According to the resolution, divide the 3D volume into the grid. For each 3D position of the grid, we create a 3D point. Input the 3D points with the image, camera matrix into our trained model and record all the information including position of input points, predicted SDF, and color values. Here might be an issue with GPU memory, the number of points grows exponentially when we increase the resolution. Hence, we split the points and iteratively input our trained model with the same image, camera matrix. Our model is the decision boundary of a deep neural network binary classifier which only predicts whether the point is inside or outside the object. The number of points as input won't affect the prediction, so we could create any resolution 3D grid and iteratively predict the output.

Once we get the SDF value for each point in the dense grid, we use Marching Cubes[22] to obtain the 3D mesh. After getting the 3D mesh, we assign the RGB values to each mesh triangle by searching the nearest 3D point.

4 Dataset and metric

4.1 Dataset

We follow the Im2avater [41] dataset which voxelizes 3D shapes from 4 categories (car, chair, table, and guitar) of ShapeNet [4] with color texture to recreate the dataset. ShapeNet [4] is a richly-annotated, large-scale dataset of 3D shapes in the textured mesh format. The 3D shapes in these subsets not only have fine-grained detail geometry shapes but also possess abundant colorful texture on the surface. Im2avater [41] transfers the textured mesh into colorful voxel volumes (See the Figure-4.1) and then generates 12 2D views for each 3D shape from horizontal viewpoints with an azimuth interval of 30. Each volume is stored in [64 * 64 *64 *3] array in the h5 file, where 64*64*64 is the resolution of voxel and in each voxel use r, g, b for color representation. The whole dataset collects 3,389 car models, 2,199 chair models, 2,539 table models, and 631 guitar models, and each follows a split ratio of 0.7/0.1/0.2 for training/validation/testing.

(a) Original mesh chair (b) After voxelized

Figure 4.1: Illustration of Voxelization. Fig(a) shows the original chair object in the mesh format. Fig(b) illustrates the same chair model after voxelization in Im2avater [41] dataset. Compared to the Fig(a), Fig(b) reduce the resolution overall and lose the smoothed representation on the surface

However, Im2avater [41] dataset doesn't record the camera matrix when generating the render views. We re-generate the dataset by the python Open3d [51] library. Firstly, visualize the voxel model every 30 degrees (See the example in Figure-4.2) and store each translation, rotation matrix during the rendering process. Then set SDF value -1 for the points inside the model and randomly select the points outside the model assigning 1 for SDF value. In addition to 12 2D rendered views with camera matrix and SDF value of points, we give the r, g, b color value to the points as well. We provide the first dataset which consists of points with both SDF value and color information.

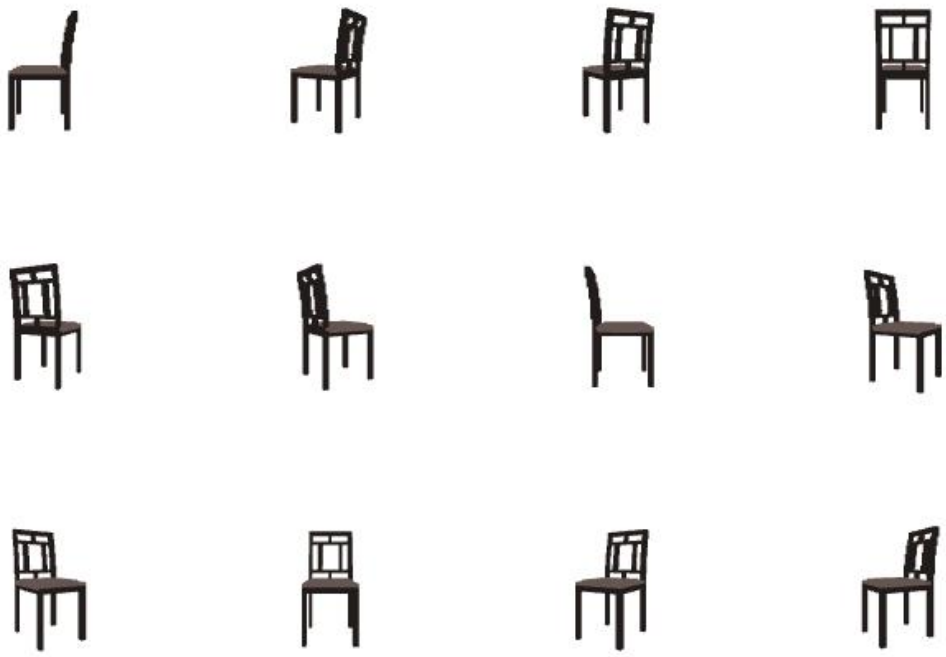


Figure 4.2: Example of rendered image in different viewpoints. Each object in dataset has 12 different rendered image as input for training of our model.

4.2 Metrics

To evaluate the colorful 3D reconstruction model. We measure the performance of shape geometry and color texture separately. Shape geometry is evaluated using Intersection-over-Union (IoU), and the color is evaluated using surface Peak Signal-to-Noise Ratio

(PSNR).

4.2.1 IoU

IoU is a standard evaluation metric for 3D shape reconstruction. The equation-4.1

$$IoU(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{4.1}$$

shows the IoU of 2 object A and B is computed as the ratio between overlap part and union part (See the Figure-4.3). The higher ratio represents the better performance of prediction.

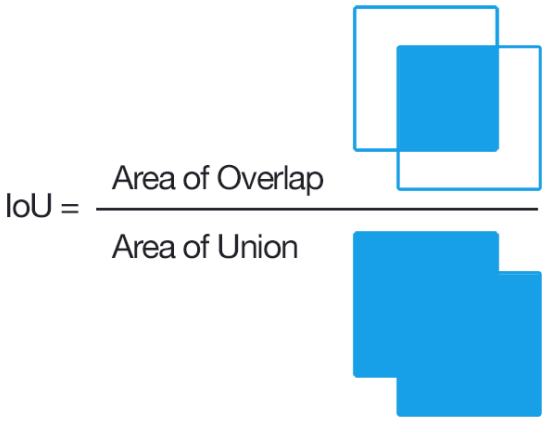


Figure 4.3: Illustration of IoU metric. Computing the Intersection over Union is as simple as dividing the area of overlap between the bounding boxes by the area of union (picture is from Adrian Rosebrock)

In order to apply IoU, we must voxelize the object with fixed resolution. The points from dataset or predicted object represent a single voxel cube in the 3D space. According to the testing resolution, we compute the IoU overlap part by counting the points which distance between ground true are less voxel size.

4.2.2 PSNR

Peak signal-to-noise ratio (PSNR) is most commonly used to measure the quality of image compression or reconstruction. PSNR is defined via the mean squared error

(MSE). Given a m*n image I and predicted output K, MSE is written in equation-4.2

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2 \tag{4.2}$$

The PSNR (in dB) is defined as equation-4.3:

$$\begin{aligned} PSNR &= 10 \cdot \log_{10} \left(\frac{MAX_I^2}{MSE} \right) \\ &= 20 \cdot \log_{10} \left(\frac{MAX_I}{\sqrt{MSE}} \right) \\ &= 20 \cdot \log_{10} (\mathrm{MAX}_I) - 10 \cdot \log_{10} (MSE) \end{aligned} \tag{4.3}$$

MAX_i is the maximum possible pixel value of the image.

(a) Original image (b) PSNR = 32 (c) PSNR = 14

Figure 4.4: Example of PSNR values for image at various quality levels. (a) is the input image. (b) shows the image of PSNR value 32 dB. (c) shows the image of PSNR value 14 dB. With high value of PSNR, we nearly couldn't tell the difference between (a) and (b) by human eyes. In the contrast, we could see the clearly noise in figure (c) with PSNR=14. (picture is from Sandipan Dey)

To use this metric we must present all values in bit form. If we have 8-bit pixels, then the values of the pixel channels must be from 0 to 255. We regard the predicted output as the original image with noise. PSNR shows a ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation (See the example at Figure-4.4). PSNR could be an approximation to human perception of reconstruction quality.

In our experiments, we apply this metric to evaluate the color of the reconstructed 3D object. To calculate the PSNR of predicted output, we collect all the points from IoU overlap part. Take the color values from ground truth as original image I, and regard the color values of predicted points as noisy image K. Using the equation-4.3 calculates the PSNR value. Typically, the higher the two metrics, the better quality of the reconstructed 3D models.

5 Experiments

In this section, we do the multiple experiments on single-view textured 3D reconstruction in comparisons with state-of-the-art methods. For the competing methods, we concentrate on a single stage of inference across diverse object categories such as Im2Avatar [41] instead of others approaches such as PIFu [37] which needs 2 stages of inference and only focuses on a single class. We not only analyze the recovery performance of shape and color, but also discuss the performance under the different viewpoints. Lastly, we look into the failure cases and figure out the reason of the wrong predictions.

For the comparison methods, besides Im2Avatar [41], we also compare with the adapted R2N2 which is the baseline model of Im2Avatar [41]. The authors of Im2Avatar [41] adapt the popular volumetric reconstruction model 3D-R2N2 [5] and name it adapted R2N2. The 3D-R2N2 [5] originally applies a recurrent module, which helps refine reconstructed 3D volumetric shapes iteratively when inputs are multi-view images. Since the task is single-view reconstruction, they drop this recurrent module. They change the input from multi-view images to a single image and add additional channels in the last layer of the original 3D-R2N2 [5] to output a volume which contains r,g,b values. The empty voxel is represented by [-0.5, -0.5, -0.5]. Therefore, the adapted R2N2 could jointly reconstruct the shape and color of the 3D object.

All the models are trained by 4 categories of Shapenet [4] namely car, chair, table and guitar. For evaluation, we evaluate the output of all methods by given image from 12 different angles of each testing instance. In order to show the generalized ability from a single category to multi-classes, our model is trained on all categories. Except our model, both adapted R2N2 and Im2Avatar are trained independently for each shape category. We use IoU to measure the performance of shape recovery and use PSNR as the texture recovery metric. The higher IoU and PSNR indicate the better textured 3D reconstruction.

We first provide the qualitative results in Table 5.1. The quantitative results will be introduce in section 5.1, 5.2.

Input view	Adapted R2N2	Im2Avatar	Ours

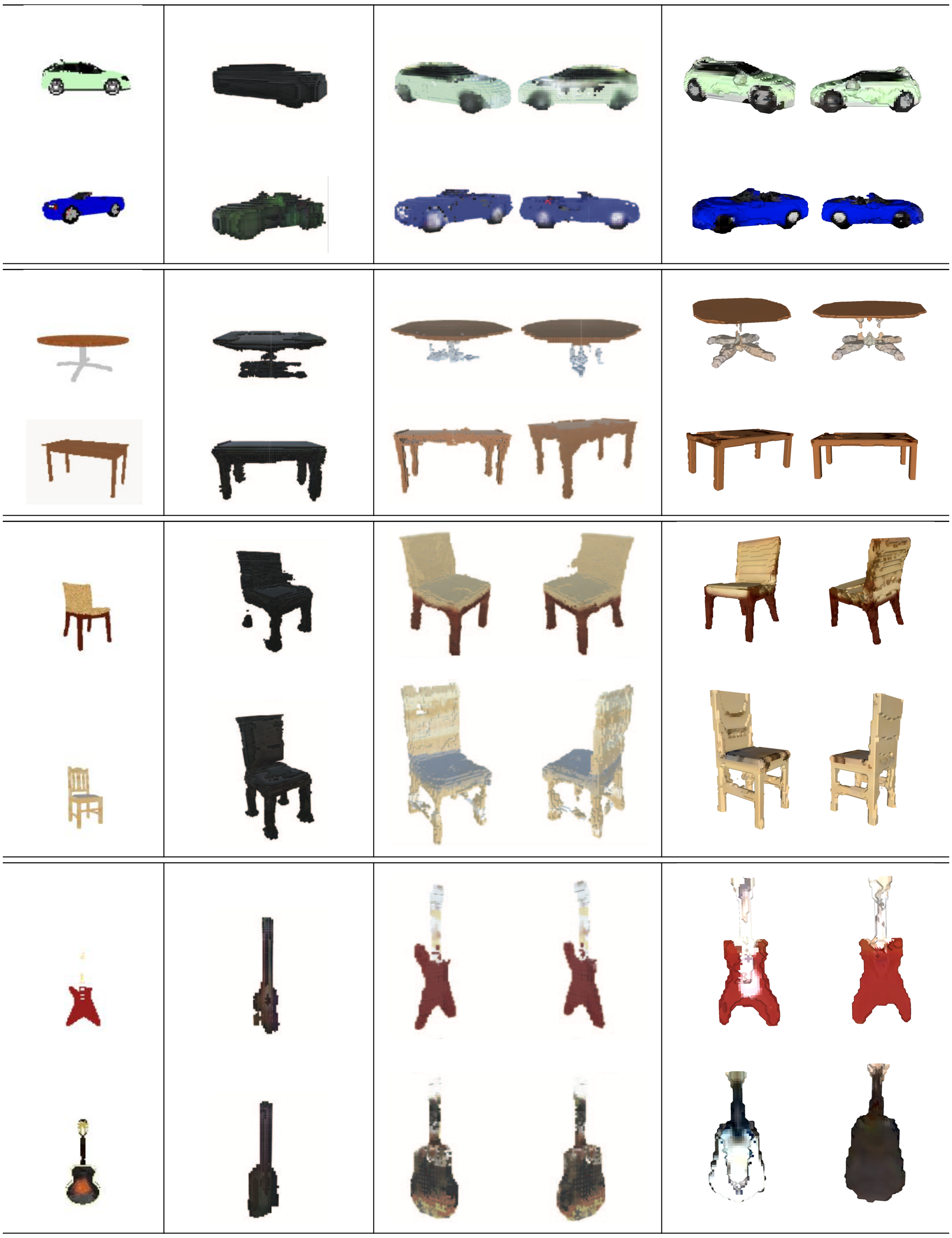


Table 5.1: Visualization of results on testing samples under 2 different views.

As can be easily noticed, the adapted R2N2 results present correct geometric structures, but their surface colors tend to be black. This implies that directly using the 3D shape reconstruction network with modified output layers for color prediction fails to capture the color texture of the object surface.

We also observe that our model provides a smoother shape than Im2Avatarr [41] and adapted R2N2, but a closer look reveals that the detail parts such as the back poles of a chair are not captured. The same results occur in color predictions. The fine details, e.g. headlight of cars, button on the guitars tend to be ignored. However, except the guitar prediction of the last row which has complex pattern, our model recovers surface texture sharper than other models.

5.1 Shape Analysis

	car	table	chair	guitar	avg
Adapted R2N2	0.238	0.216	0.179	0.308	0.235
Im2Avatar	0.395	0.2381	0.180	0.374	0.298
Ours	**0.419**	**0.314**	**0.252**	**0.376**	**0.340**
Ours (only shape)	0.415	0.325	0.262	0.376	0.344

Table 5.2: Mean IoU on testing samples

We measure all the reconstructed shapes with the metric of mean IoU by averaging the IoU of all generated 3D shapes in the same category. The results are showed in Table-5.2. Our model outperforms the other two models in all shape categories. Especially in the chair and table categories which have various shapes (See the Figure-5.1) and hard to generate the general shape for every object. We notice the difference IoU of these two categories from Im2Avatar [41] and adapted R2N2 is relatively small, only 0.001 in chair and 0.025 in table. However, compared to our model, we reach 0.314 in table and 0.252 in chair which improved around 0.072 in both categories relatively to Im2Avatar [41]. This result shows our model is able to perform well in the object with fine structure.

To ensure that the performance of shape recovery is not affected by jointly predicting shape and color, we re-train the model which only recovers the 3D shape (only shape). By simply removing the color texture decoder, our model is modified to only reconstruct

Figure 5.1: Illustration of examples in chair and table classes. Especially for the legs of chairs and tables, the cases are too diverse to reconstruct by the general shape

3D shapes. The results are also shown in Table-5.2. It is noticeable that our models outperform other methods with and without color prediction. Both of them reach to similar performance of 3D shape recovery. The results verify that the predicted geometry of our model will not be interfered by color predictions.

5.2 Textured Color Analysis

	car	table	chair	guitar	avg
Adapted R2N2	6.502	6.901	7.063	5.668	6.533
Im2Avatar	10.881	**19.728**	**18.252**	10.617	**14.870**
Ours	**10.90**	17.822	15.710	**10.820**	13.813
Ours (only chair)	- -	- -	18.514	- -	- -

Table 5.3: Mean PSNR on testing samples

The PSNR metric is used to evaluate the perceptual quality of estimated surface colors. We calculate PSNR value for all the reconstructed shapes, and their mean values are reported for each category in Table-5.3. In general, Im2Avatar [41] makes the better color prediction, especially in chair and table classes. However, the difference between our approach and Im2Avatar [41] are relatively small. In chair and table classes, Im2Avatar [41] defeats ours by an average of 2.22, and in classes of car and guitar, our

model outperforms Im2Avatar [41] with 0.106. The disparity between each other is too small. Thus, we regard that the color recovery capability of our model achieves the same level as Im2Avatar [41].

The main reason for better color prediction of Im2Avatar [41] is that they trained several models to overfit the dataset. They trained independently for each category, and for each category they need two models, which one is for shape reconstruction and the other is for color prediction. Im2Avatar [41] needs 8 different models in total (4 categories and each category needs 2 models) so each model could fully focus on only one task. To compare with one model for one category method, we re-train a model independently on a single category chair (only chair). The result is shown in Table-5.3 as well. Our approach could reach the same performance as Im2Avatar [41]. One model for one category method increases the prediction performance, but at the same time needs multiple models for different classes.

In terms of jointly predicting shape with color or using two models to predict shape and color separately, the authors of Im2Avatar [41] did the related experiment confirming our assumption. They trained a unified network, which jointly learns shape and surface colors in a single network, on car category. Compared it with the original separated framework which decomposes shape and color learning into two models. The mean IoU for the unified network model is 0.386, which is less than 0.395 the result of the separated framework. The Figure-5.2 illustrates the results of the separated framework and the unified network. As can be observed, there is more noise in the output of the unified network. In contrast, surface colors generated by the separated framework are smoother and more visually satisfying.

Even though Im2Avatar [41] reaches better results than other methods generally, we prefer to utilize a network, which generalizes to multiple objects and achieves acceptable textured color prediction, instead of dividing the task into several simple tasks and training a model to overfit each task. Our approach provides an end-to-end neural network which jointly predicts shape and surface color for multiple categories. The reconstructed 3D shapes have more fine-grained detail and the surface textured color prediction reaches a certain extent of Im2Avatar [41].

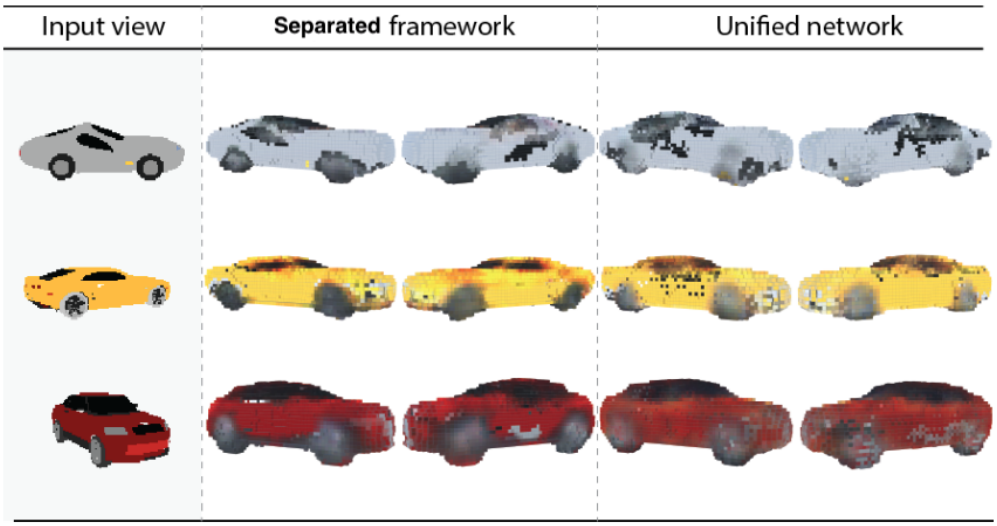


Figure 5.2: Visualize randomly selected colorful reconstruction results of the separated framework and the unified network. The 3D shapes are rendered from two different views (picture is from Im2Avatar [41])

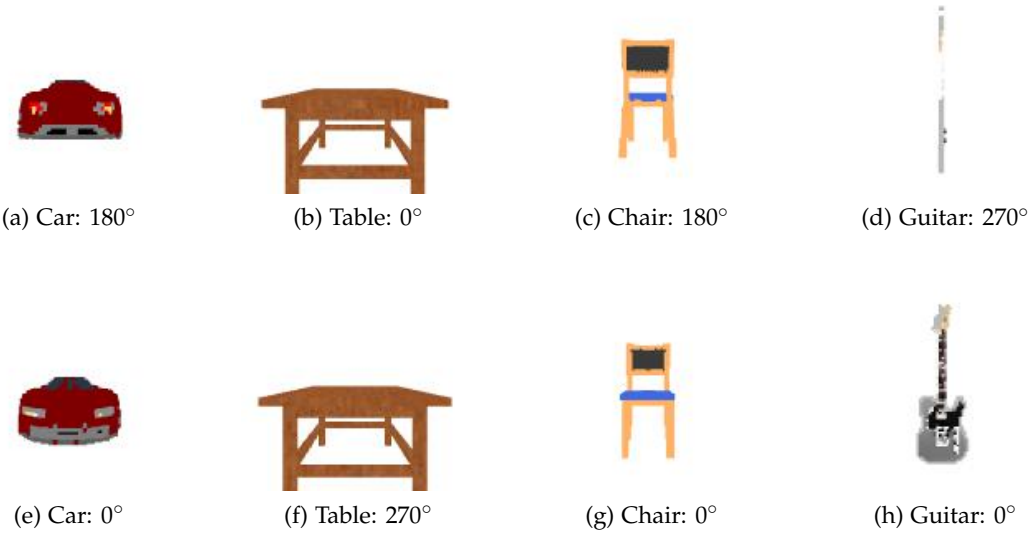
5.3 Angle Analysis

	car	table	chair	guitar
90°	0.421	0.298	0.260	0.372
120°	0.421	0.316	0.259	0.381
150°	0.420	0.318	0.253	0.379
180°	0.403	0.314	0.234	0.376
210°	0.419	0.318	0.252	0.378
240°	0.420	0.318	0.257	0.381
270°	0.420	0.300	0.261	0.370
300°	0.423	0.316	0.257	0.382
330°	0.423	0.318	0.251	0.372
0°	0.406	0.313	0.241	0.369
30°	0.423	0.316	0.249	0.374
60°	0.423	0.318	0.256	0.381

Table 5.4: IoU under 12 different viewpoints

To understand the impact of input viewpoint on shapes and textured color reconstruc-

tion, we record the IoU and PSNR values respectively under the 12 different viewpoints. Table-5.4 shows the result of IoU part and the worst two angles are marked. Overall, the worst cases occur in the orthogonal or parallel angles 0°, 90°, 180°, 270°. Figure-5.3 illustrates the viewpoints of worst cases corresponding to categories. Without knowing the classification label, sometimes as human beings, we are not able to recognize some objects in Figure-5.4. Accordingly, Our model conforms to human visional experience.



(a) Car: 180° (b) Table: 0° (c) Chair: 180° (d) Guitar: 270°

(e) Car: 0° (f) Table: 270° (g) Chair: 0° (h) Guitar: 0°

Figure 5.3: Illustration of the viewpoints of the worst cases in each category. Each label indicates the name of category and the angle of the viewpoint.

The orthogonal and parallel pictures of objects disclose the least information of what is the object including shape, structure. Especially for guitar, the 270° angle only show the side of guitar which restricts image encoder to extract any useful feature. Furthermore, the back side of guitar viewpoint is similar to the front side that will confuse the model to generate the correct prediction. Besides thorny viewpoints of guitar, both car and chair have the worst predictions under the 0°, 180° orthogonal angles. Only front or back sides of images of car and chair do not provide any further information of overall shape.

Table-5.5 shows the results of PSNR under different perspectives. The worst cases also occur in the orthogonal or parallel angles. Likewise, the viewpoints also influence

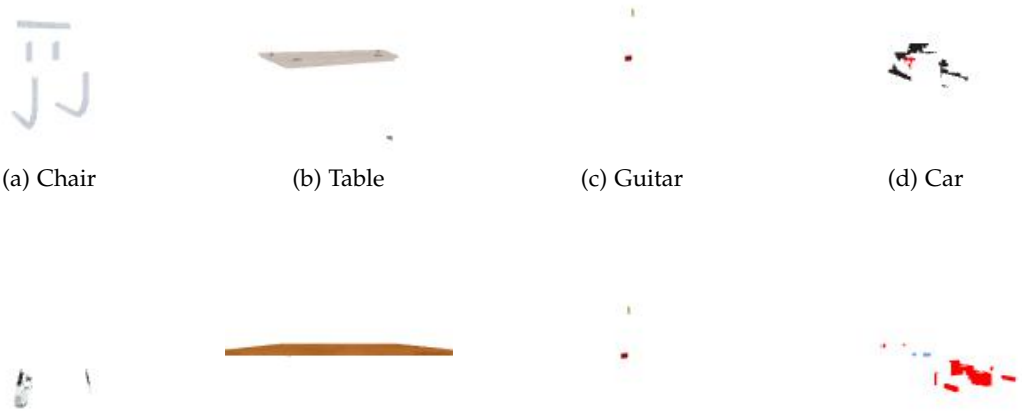
	car	table	chair	guitar
90°	11.02	17.73	15.62	10.86
120°	11.00	17.82	15.73	11.08
150°	10.89	17.80	15.72	10.76
180°	10.45	17.77	15.55	10.74
210°	10.89	18.06	15.64	10.90
240°	11.02	17.94	15.75	10.95
270°	11.05	17.65	15.61	10.87
300°	11.06	17.85	15.77	10.80
330°	10.96	17.76	15.79	10.76
0°	10.44	17.71	15.88	10.55
30°	10.94	17.83	15.78	10.75
60°	11.06	17.95	15.69	10.83

Table 5.5: PSNR under 12 different viewpoints

the color textured reconstruction. The interesting found is that most of time shape performance accords to color performance. The lower IoU prediction will produce a relatively bad color prediction. For example, the worst two angles of the car category in IoU and PSNR are both 0°, 180°. The reason is in the model architecture the geometry prediction is passed to the color textured decoder for color prediction (See in Figure-3.2). The performance of the shape will affect the color reconstruction.

5.4 Failure Analysis

To investigate the deficit of our model, we look into the failure test cases. Try to figure out the reason of failure reconstruction. We collect the predicted objects whose IoU is below 0.1. Among these objects, we find out that besides the intricate structure of objects, the white objects (See the example of white objects in Figure-5.4) always result in wrong predictions. Even as human being, those white objects are too hard to be recognized. Because white is also the background of images, the white objects will merge into the background and confuse our model to make the wrong prediction. As shown in Figure-5.5, the input view is a white chair, but get the 3D object of a table. The white chair back in Figure-5.5 (a) is blended into the background. The model only classifies the object by chair legs which are similar to the table legs. Thus it predicts the white table as output.



(a) Chair (b) Table (c) Guitar (d) Car

(e) Chair (f) Table (g) Guitar (h) Car

Figure 5.4: Example of white objects. In each category, there are white objects which are too difficult to identify, so we put the label to every example image

(a) Input view (b) Result

Figure 5.5: Visualization of failed shape prediction. Figure (a) is a white chair as the input image. Figure (b) is the output of our model. Obviously, our model regards Figure (a) as a white table. This kind of misclassification happens to all white objects.

In addition to the failure of shape recovery, the failure of color prediction in the last row of Table-5.1 intrigues our interest in our model's capability of color recovery. We investigate other objects which have the complex color pattern such as the guitar in the Table-5.1 last row. Table-5.6 shows 2 examples of car with vivid color patterns. Our

model fails to recover the pattern on the car side and tends to predict white or grey color to replace the area of complex color pattern. If we look closely at results of surface color in both Table-5.1 and Table-5.6, the textured of universal part of an object such as chair legs, table legs, guitar necks, car windshields could be captured. However, the complex pattern of the object is missed by our model.

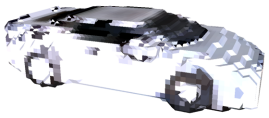
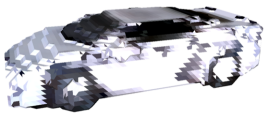
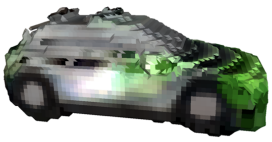
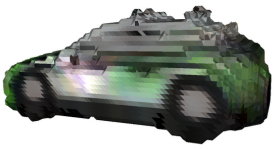
Input view	Results
	 
	 

Table 5.6: Visualization of failed color prediction. In first row, our model only captures the general color of car and in second example even though it can predict the green in front and back areas, it miss all the other patterns.

6 Discussion

In this section, we discuss the approaches we have tried to enhance our model, but for several reasons they failed. We tried to use better pre-trained model as image encoder, change the architecture of the model and add one more loss called render loss to optimize the performance.

6.1 Image Encoder

Besides geometry location of points, a single image is the only input of neural network. Therefore, feature extraction of images would be pivotal to determine the performance of colorful 3D reconstruction. In our approach, we apply the VGG16 [39] pre-trained model as an image decoder for features extraction. However, there are a lot of pre-trained models whose capability are better than VGG16 [39]. PIFu [37] did the experiments show the comparison of VGG16 [39], Resnet-34 [15], and HG [26] as image encoder for human shape reconstruction. The experiments were done on datasets of RenderPeople [1] and BUFF [50]. They measure the average point-to-surface Euclidean distance (P2S) in cm from the vertices on the reconstructed surface to the ground truth and the Chamfer distance which computes the distance between the reconstructed and the ground truth surfaces. In addition, They introduce the normal re-projection error. For both reconstructed and ground truth surfaces, calculate the L2 error between their normal maps which are rendered to 2D image space from the input viewpoint.

	RenderPeople			Buff		
Methods	Normal	P2S	Chamfer	Normal	P2S	Chamfer
VGG16	0.125	3.02	2.25	0.144	4.65	3.08
ResNet34	0.097	**1.49**	**1.43**	0.099	1.68	1.50
HG	**0.084**	1.52	1.50	**0.092**	**1.15**	**1.14**

Figure 6.1: Ablation study on different image encoders (picture is from PIFu [37])

Figure-6.1 shows the results of the experiments and Figure-6.2 visualizes the P2S error under different image encoders. In general, Resnet-34 [15] outperforms VGG16 [39] in

shape reconstruction by multiple metrics. Therefore, we based on analysis of PIFu [37] changing our image encoder from VGG16 [39] to Resnet-34 [15] pre-trained model which is provided in Pytorch library as well. We extract the features before every pooling layer in ResNet-34 [15] and follow the process of local feature extraction-3.2 to extract the local features. However, the 3D reconstruction results do not make any progress. We guess the reason would be the low image resolution [138*138] input which limits Resnet-34 [15] to extract more complex features. Thus, increasing the resolution of input images is the most potential work which we could try in the future.

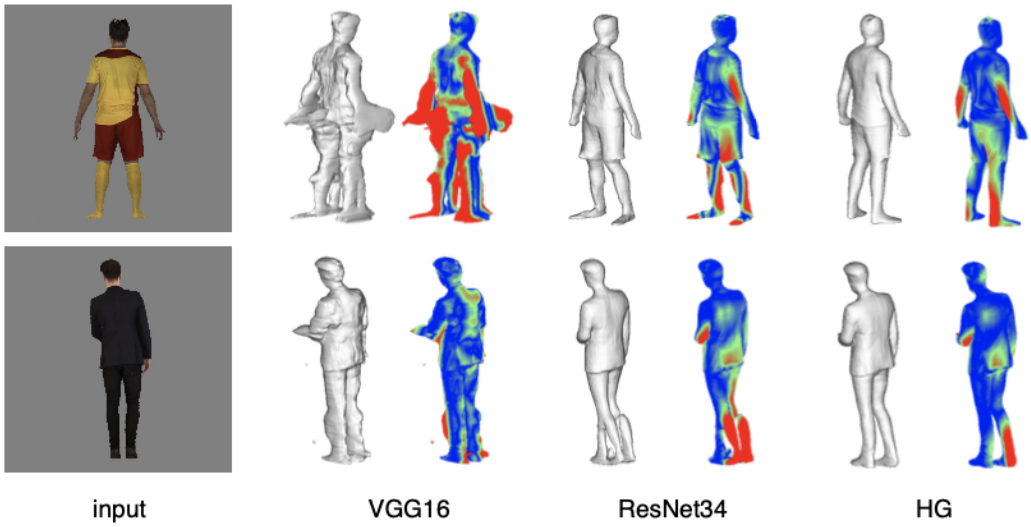


Figure 6.2: Reconstructed geometry and point to surface error visualization using different architectures for the image encoder. The colors indicate the error range in order from small to large is blue, green and red. VGG16 [39] shows the worst result in three of different architectures. (picture is from PIFu [37])

6.2 Architecture

We also tried the different architecture of model to enhance the overall performance. Our model utilizes the same extracted features for both geometry and texture color decoder. However, the geometry and color recovery may focus on the different part of the image. Using the same feature as input for both shape and color prediction might keeps prediction from reaching the optimized performance. Inspired by PIFUu [37] and Im2Avatar [41] which apply different networks for shape reconstruction and color

textured prediction, we think the extracted features for the geometry decoder and color textured decoder should be different. Thus, we use two Restnet-34 [15] as image decoders to extract the features corresponding to geometry decoder and color texture decoder respectively.

To simplify the network, we change the two decoder branches to single decoder branch architecture, because in the experiments of DISN [47] single-decoder architecture is only slightly inferior to two-decoder architecture. The decoder branch architecture remains the same structure shown in Figure-3.2. Figure-6.3 shows the overview of the new architecture, the input data is passed to two image encoders for geometry and color features extraction. Then geometry and color texture decoder takes the corresponding features and input data as input to predict SDF values and RGB colors separately.

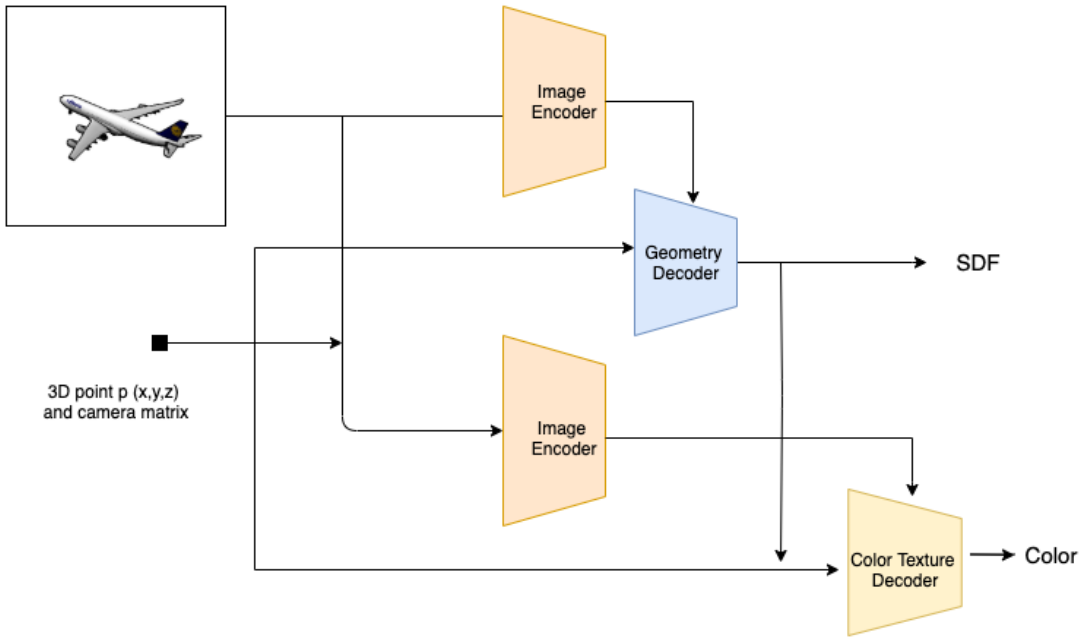


Figure 6.3: Illustration of new architecture. We tried to use different image encoder for SDF and color predictions. Besides the image encoder, we alter to single branch decoder.

For each image encoder, we discard the pre-trained weight from the library and train

it from scratch. We follow the local feature extraction-3.2 process to get local features and concatenate local features with global features. Then, the concatenated features forward to the corresponding decoder branch.

The new architecture did not improve the recovery performance of shape and texture. Since it did not improve the performance, we did not conduct any further experiments to explore the causes reasons. We only guess the reasons might cause this result. One is the same as mentioned in Image encoder section 6.1, the low image resolution input restricts the model to reaching better performance. Another is even though shape and color reconstruction will need the different image features as input, it will fail if train both of them within an end-to-end network.

6.3 Render Loss

To improve the color reconstruction, we try to add one loss called render loss which renders the predicted colorful 3D object and ground truth as 2D images and calculates the $L1$ loss to optimize our model. The idea is inspired by SPSG [6] which aims to complete incomplete RGB-D scan observations by 2D view-guided synthesis, leveraging rendered views of the predicted 3D model. To formulate 2D-based losses, they render the predicted TSDF (Truncated Signed Distance Field) 3D volumetric cube by raycasting to generate color, depth, and normal images, for a given viewpoint. The render loss helps SPSG generate a compelling color prediction. Thus we incorporate render loss into our model via Pytorch3d [35] toolkit which is a differentiable renderer implemented by PyTorch [31] tensors.

However, adding render loss did not improve the performance of color prediction. The main factor might be our output type is a points cloud with SDF values instead of a TSDF 3D cube. The added render loss did not provide further constraint to optimize weights of the our model.

7 Conclusion

In this thesis, we present a deep implicit network for single-view textured 3D reconstruction. It is an end-to-end network which can reconstruct both 3D surface and texture by a given single 2D image for multiple classes.

Our approach shows the results with smoother shape and plausible color recovery. It outperforms Im2Avatar [41] and adapted R2N2 in mean IoU metric, and reaches a certain extent of Im2Avatar [41] in terms of PSNR metric. We also compare the performance from different perspectives. Both shape and texture recovery are affected by the orthogonal or parallel viewpoints just like human being. Besides experiments, we investigate the failed cases. The white objects will merge into the background which interferes with the shape recovery. For color reconstruction, the complex patterns of objects are not captured by our model.

Lastly, we discuss the approaches which we failed to improve our model. We have tried the better pre-trained model of classification task as image encoder, different architecture, and render loss which provide further information to the people who are interested in the related field.

8 Future Work

In this final section of the thesis we want to state some possible future works. A straightforward modification is to improve the problem which causes the bad performance of white object mentioned in failure analysis-5.4. One of the solutions to deal with white objects is to provide the r, g, b, a images as input. The additional channel 'a' defines the opacity as a number between 0.0 (fully transparent) and 1.0 (fully opaque). Fully opaque indicates the object shape in the image, though the object's color is the same as the background. However, all the pre-trained image encoder are trained with r, g, b images. We either remove these white object or retrain image encoder with r, g, b, a images in the future.

The other modification is to change the input dataset. As mentioned in Discussion-6, the low image resolution might affect the performance of the neural network. All the latest related work such DISN [47], SPSG [6], PIFu [37], Occupancy network [25] use image with resolution 224*224 or even higher resolution as input to make sure the image encoder could extract complicated features. Thus, we believe that higher resolution of input images can possibly make progress in overall performance.

Besides image input, the sampled points could be changed to higher precision. In practice, we are interested in points near the iso-surface. Therefore, we could sample SDF points non-uniformly near the surface by the method which is proposed in the DeepSDF [30]. Instead of the dataset of Im2Avatar [41] with the 64*64*64 grid, the sampled SDF points could represent the surface more precisely.

As a neural network, the quality of input data will contribute a large proportion of overall performance. To improve the performance of our model, our future work aims to get the higher quality of input data by aforementioned modifications.

List of Figures

List of Tables

Bibliography

[1] https://renderpeople.com/ 3d- people. "Renderpeople." In: 2018.

[2] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. "Tensorflow: A system for large-scale machine learning." In: *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*. 2016, pp. 265–283.

[3] Pbroks13 Bob Mellish. URL: `https://commons.wikimedia.org/wiki/File:Pinhole-camera.svg`.

[4] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. "Shapenet: An information-rich 3d model repository." In: *arXiv preprint arXiv:1512.03012* (2015).

[5] Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. "3d-r2n2: A unified approach for single and multi-view 3d object reconstruction." In: *European conference on computer vision*. Springer. 2016, pp. 628–644.

[6] Angela Dai, Yawar Siddiqui, Justus Thies, Julien Valentin, and Matthias Nießner. "Spsg: Self-supervised photometric scene generation from rgb-d scans." In: *arXiv preprint arXiv:2006.14660* (2020).

[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "Imagenet: A large-scale hierarchical image database." In: (2009), pp. 248–255.

[8] Swapna K E. URL: `https://developersbreach.com/convolution-neural-network-deep-learning/`.

[9] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. "The PASCAL visual object classes challenge 2007 (VOC2007) results." In: (2007).

[10] Mark Everingham and John Winn. "The pascal visual object classes challenge 2012 (voc2012) development kit." In: *Pattern Analysis, Statistical Modelling and Computational Learning, Tech. Rep* 8 (2011).

[11] Haoqiang Fan, Hao Su, and Leonidas J Guibas. "A point set generation network for 3d object reconstruction from a single image." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 605–613.

[12] Rohit Girdhar, David F Fouhey, Mikel Rodriguez, and Abhinav Gupta. "Learning a predictable and generative vector representation for objects." In: *European Conference on Computer Vision*. Springer. 2016, pp. 484–499.

[13] Gregory Griffin, Alex Holub, and Pietro Perona. "Caltech-256 object category dataset." In: (2007).

[14] Muneeb ul Hassan. URL: https://neurohive.io/en/popular+-networks/vgg16/.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition." In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[16] http://www.makehuman.org/. "MakeHuman." In:

[17] Slobodan Ilic. URL: http://campar.in.tum.de/Main/SlobodanIlic.

[18] Laura Leal-Taixé. URL: https://www.professoren.tum.de/leal-taixe-laura.

[19] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations." In: *Proceedings of the 26th annual international conference on machine learning*. 2009, pp. 609–616.

[20] Marc Levoy. URL: http://graphics.stanford.edu/~levoy/.

[21] Haoning Lin, Zhenwei Shi, and Zhengxia Zou. "Maritime Semantic Labeling of Optical Remote Sensing Images with Multi-Scale Fully Convolutional Network." In: *Remote Sensing* 9 (May 2017), p. 480. DOI: 10.3390/rs9050480.

[22] William E Lorensen and Harvey E Cline. "Marching cubes: A high resolution 3D surface construction algorithm." In: *ACM siggraph computer graphics* 21.4 (1987), pp. 163–169.

[23] Satya Mallick. URL: https://learnopencv.com/geometry-of-image-formation/.

[24] Jonathan Masci, Davide Boscaini, Michael Bronstein, and Pierre Vandergheynst. *Shapenet: Convolutional neural networks on non-euclidean manifolds*. Tech. rep. 2015.

[25] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. "Occupancy networks: Learning 3d reconstruction in function space." In: (2019), pp. 4460–4470.

[26] Alejandro Newell, Kaiyu Yang, and Jia Deng. "Stacked hourglass networks for human pose estimation." In: *European conference on computer vision*. Springer. 2016, pp. 483–499.

[27] Matthias Niessner. URL: https://www.professoren.tum.de/niessner-matthias.

[28] Michael Oechsle, Lars Mescheder, Michael Niemeyer, Thilo Strauss, and Andreas Geiger. "Texture fields: Learning texture representations in function space." In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 4531–4540.

[29] M. R. Oswald, E. Toeppe, and D. Cremers. "Fast and Globally Optimal Single View Reconstruction of Curved Objects." In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Providence, Rhode Island, July 2012, pp. 534–541.

[30] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. "Deepsdf: Learning continuous signed distance functions for shape representation." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 165–174.

[31] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. "Pytorch: An imperative style, high-performance deep learning library." In: *arXiv preprint arXiv:1912.01703* (2019).

[32] Michael Plotke. URL: https://en.wikipedia.org/wiki/Kernel_(image_processing).

[33] Loc Vu-Quoc. URL: https://commons.wikimedia.org/wiki/File:Neuron3.png.

[34] Sebastian Raschka. URL: https://sebastianraschka.com/Articles/2015_singlelayer_neurons.html.

[35] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. "Accelerating 3D Deep Learning with PyTorch3D." In: *arXiv:2007.08501* (2020).

[36] Sumit Saha. URL: https://medium.com/@_sumitsaha_.

[37] Shunsuke Saito, Zeng Huang, Ryota Natsume, Shigeo Morishima, Angjoo Kanazawa, and Hao Li. "Pifu: Pixel-aligned implicit function for high-resolution clothed human digitization." In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 2304–2314.

[38] Sagar Sharma. URL: https://medium.com/@sagarsharma4244.

[39] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." In: *arXiv preprint arXiv:1409.1556* (2014).

[40] cs231n stanford. URL: http://cs231n.stanford.edu/.

[41] Yongbin Sun, Ziwei Liu, Yue Wang, and Sanjay E Sarma. "Im2avatar: Colorful 3d reconstruction from a single image." In: *arXiv preprint arXiv:1804.06375* (2018).

[42] Eno Töppe, Martin R Oswald, Daniel Cremers, and Carsten Rother. "Image-based 3d modeling via cheeger sets." In: *Asian Conference on Computer Vision*. Springer. 2010, pp. 53–64.

[43] Greg Turk. URL: https://www.cc.gatech.edu/~turk/.

[44] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. "Pixel2mesh: Generating 3d mesh models from single rgb images." In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 52–67.

[45] Jiajun Wu, Yifan Wang, Tianfan Xue, Xingyuan Sun, William T Freeman, and Joshua B Tenenbaum. "Marrnet: 3d shape reconstruction via 2.5 d sketches." In: *arXiv preprint arXiv:1711.03129* (2017).

[46] Jiajun Wu, Chengkai Zhang, Xiuming Zhang, Zhoutong Zhang, William T Freeman, and Joshua B Tenenbaum. "Learning shape priors for single-view 3d completion and reconstruction." In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 646–662.

[47] Qiangeng Xu, Weiyue Wang, Duygu Ceylan, Radomir Mech, and Ulrich Neumann. "Disn: Deep implicit surface network for high-quality single-view 3d reconstruction." In: *arXiv preprint arXiv:1905.10711* (2019).

[48] Xinchen Yan, Jimei Yang, Ersin Yumer, Yijie Guo, and Honglak Lee. "Perspective transformer nets: Learning single-view 3d object reconstruction without 3d supervision." In: *arXiv preprint arXiv:1612.00814* (2016).

[49] Matthew D. Zeiler and Rob Fergus. "Visualizing and Understanding Convolutional Networks." In: *CoRR* abs/1311.2901 (2013). arXiv: 1311.2901. URL: http://arxiv.org/abs/1311.2901.

[50] Chao Zhang, Sergi Pujades, Michael J Black, and Gerard Pons-Moll. "Detailed, accurate, human shape estimation from clothed 3D scan sequences." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 4191–4200.

[51] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. "Open3D: A Modern Library for 3D Data Processing." In: *arXiv:1801.09847* (2018).