DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Real-time Neural View Synthesis for Volumetric Datasets

Parika Goel

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Real-time Neural View Synthesis for Volumetric Datasets

Netzwerk-basierte Bildsynthese von Volumendaten in Echtzeit

Author:	Parika Goel
Supervisor:	Prof. Dr. Rüdiger Westermann
Advisor:	M.Sc. Sebastian Weiß
Submission Date:	November 15, 2020

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

Munich, November 15, 2020 Parika Goel

Acknowledgments

First and foremost, I would like to thank my thesis advisor, Sebastian Weiß for enlightening me on the relevant topics, and guiding me towards the right direction. I truly appreciate the constant support, encouragement and interesting insights that I was provided with throughout my thesis. I would also like to thank my supervisor, Prof. Dr. Rüdiger Westermann, for being such an inspiring teacher, enlightening me on the subject and motivating me to work on this topic.

Finally, I would like to express my profound gratitude to my family and friends who have always encouraged me and given me moral support throughout my Master's study and especially during the process of working on and writing this thesis. I have met some amazing people during my studies, made great friends and fond memories which I shall cherish forever.

Abstract

The task of view synthesis i.e. generating novel views of a scene or object using an existing set of views has gained a lot of attention in recent years due to its applications in virtual and augmented reality. Recent work by Mildenhall et al. [1] has shown impressive results on the task of view synthesis using Multi-layer Perceptron (MLP) to represent a 3D scene as a continuous function. Representing scenes as continuous functions instead of using discrete representations like voxel grids has a huge advantage in terms of memory usage. To render high-quality images, the voxel grids need to be sampled at high resolutions in which case the memory required to store the scene information increases exponentially. Using implicitly defined continuous functions parameterized by MLPs does not suffer from this disadvantage. Mildenhall et al. in their work titled Neural Radiance Field (NeRF) [1] generate a ray passing through a volume for each image pixel. Then they sample multiple points on each ray and use classical volume rendering techniques to calculate the final color of each ray. Though their approach shows impressive results, it is still expensive in the generation of 3D points for each ray. To get the color value for one pixel, the network has to learn the color and opacity information on multiple points in the volume. We present an approach to represent an object on a bounding sphere around it instead of taking a whole volume to represent a 3D object. We project the images of the object on the bounding sphere. The intuition behind this approach is that how the image information changes across the bounding sphere give an understanding of how the object appearance changes in the 3D world. We provide an analysis of how this approach works on the task of view synthesis for semi-transparent objects.

Contents

1. Introduction

Synthesizing novel views from an available set of input views has been a long-standing research problem in the computer vision and computer graphics community. The task is to take a set of images of a scene with known intrinsics and extrinsics and to extrapolate and interpolate these images to generate images from new viewpoints. This line of work has major applications in content generation. For example, in virtual reality applications, novel view synthesis can be used to create the virtual world from the limited set of images of the real world. It can also be used in catalog creation. The photographers can capture a few images of a scene and then post-process those images to create the desired catalog. It can also be used in the research community to augment existing datasets.

In this work, we address this problem by taking a set of 2D images with known camera poses and using deep learning techniques to train a neural network to learn an implicit representation of object geometry and appearance. This learned representation is then used to render images from any viewpoint. We represent a 3D object as a continuous function which takes a set of 3D points and view directions as input and outputs the view-dependent RGB color at that point. Our object representation has been motivated by the scene representation used by Mildenhall et al. in their recent work titled NeRF [1]. This continuous function is parameterized by the weights of the neural networks. We generate the 3D points and view directions by taking a bounding sphere around the object and project the input images onto the bounding sphere. For each pixel, we get a corresponding 3D point on the sphere and a direction vector specifying the direction from which the point is being viewed (from the camera position to the 3D point). The set of pairs of these 3D points and direction vectors collectively represent the 3D object. Given this input, we train the network to predict an RGB color at each point which gives us the rendered image. The network is trained over the loss between the rendered image and the ground truth image. At inference time, we generate the input w.r.t the query viewpoint, feed it into the network, and get the image of the object from the queried viewpoint.

As mentioned, we represent the 3D scene as a continuous function parameterized by the weights of the neural network. Tancik et al. [2] showed that using MLPs to represent 3D scenes has one limitation. MLPs suffer from a phenomenon called 'Spectral Bias' due to which they are restricted in their ability to learn fine details in the input data. They also showed how the approach of Fourier Features [3] can be used to overcome this limitation of MLPs. Motivated by the success of their approach, we have also used Fourier Encodings in our work.

The work of Mildenhall et al. [1] shows impressive results on the task of generating photo-realistic images of an object or a scene. But their approach is expensive in the generation of input points. Mildenhall et al. [1] sample multiple points on each ray (corresponding to

each pixel) and the network has to learn the opacity and color values on each of these points. To overcome this expensive operation of sampling multiple points on each ray, we came up with the approach of representing the object/scene with the help of the bounding sphere around it. We take the 3D point where the ray intersects with the bounding sphere instead of sampling multiple points on the ray. Also, our work differs from the work of Mildenhall et al. [1] in that we have worked on the images of a semi-transparent object (details of which are introduced in Section 4.3). Mildenhall et al. [1] have used the synthetic renderings of opaque objects (Lego, Ship, Microphone, etc.) and renderings of real-world scenes containing opaque objects (Fern, Orchid, etc.).

The analysis of how our approach works is first shown on isosurfaces and opaque objects and then on semi-transparent objects. We provide a quantitative and qualitative evaluation of how the change in Fourier Encodings affects the performance of the network. We also discuss the importance of the approach of taking multiple samples along the ray [1] for the network to be able to render good quality images.

Below is the overview of how the report is organized:

- **Chapter 2** discusses about the existing work on view synthesis which represents a 3D scene as a continuous function parameterized by the weights of the neural networks. We also discuss the existing research about the limitation of neural networks to learn fine detailed information in the input data and another line of research work which introduced an approach called Fourier Features to overcome this limitation.
- **Chapter 3** provides some background knowledge required to understand the technique of Fourier Features.
- **Chapter 4** discusses our scene representation in detail, how we generate the input data. It also introduces the datasets we have used in our work.
- **Chapter 5** discusses our approach. We introduce the architecture of our neural network, the loss functions we have used and explain the technique of Fourier Encoding.
- **Chapter 6** presents the qualitative and quantitative evaluation and analysis of our approach.
- **Chapter 7** presents the conclusions and the future work

2. Related Work

The computer vision and graphics community has made a lot of progress in the field of scene understanding and view synthesis. The classical methods of rendering images of a 3D scene require access to the 3D model of the scene which is not feasible for every real-world scene. Due to this, there has been increased interest in using deep learning techniques to learn a representation of geometry and/or appearance of an object or scene from a set of observed RGB images. One promising area of work has been to represent the scene as a function parameterized by neural networks (neural scene representation) instead of using volumetric representations (voxel or mesh-based). The disadvantage of using volumetric representations is their high memory requirements. Rendering photo-realistic views requires sampling voxel grids at higher resolutions. The memory required increases exponentially with the increase in grid resolution. Due to which, though these approaches have shown considerable performance for the task of view synthesis, they are limited by their capacity to scale to high-resolution images. The neural scene representations do not suffer from this limitation.

We will first discuss some of the recent works which use neural scene representation. Then we will go through a line of research work that discusses the limitation of MLP networks to learn fine details in the input data which restricts their ability to generate high-quality scene renderings. Then at last we will discuss an approach introduced in some recent works to overcome the mentioned limitation of MLPs.

2.1. Neural Scene Representation

A recent and promising line of research work represents the 3D scenes or objects as a continuous function that maps the input values to the desired output (feature vector [4]/volume density [1]/RGB [1]). This function is represented by a fully connected neural network, also called Multi-layer Perceptron (MLP), and is thus parameterized by the weights of the network. Sitzmann et al. [4] represent the 3D scene as a continuous function that maps the 3D world coordinate to a feature representation of the scene properties at that point. They introduce a neural rendering function that takes as input the scene representation and the camera parameters and outputs a 2D image. The scene representation function and neural rendering function are trained end-to-end with a set of 2D images and camera poses. They call their representation Scene Representation Networks (SRNs). Mildenhall et al. [1] represent the 3D scene as a continuous function that maps a 3D point (x,y,z) and the view direction (θ, ϕ) to a volume density and view-dependent RGB color at that 3D point. To render an image from a particular viewpoint, they march the camera rays (corresponding to each pixel) through the

scene and sample a set of 3D points on each ray. These sampled 3D points combined with the ray directions are fed as input to the network. Traditional volume rendering techniques are then used to accumulate the output data along camera rays and generate the rendered image of the scene from that viewpoint. Inspired by the voxel octree structure [5], Liu et al. [6] improved upon this by representing the 3D scene as a set of implicit fields organized in an octree fashion. Motivated by the work of [1], we used fully connected neural network to represent a 3D object and use the spatial 3D points and view directions as input. But our approach differs from the approach of Mildenhall et al. [1] in the way we generate the 3D points. Instead of using the technique of raymarching to sample the 3D points, we take a bounding sphere around the object and take the intersection of the camera rays with this bounding sphere as the input 3D points. This makes our approach less expensive in the generation of input points.

2.2. Spectral Bias of MLPs

Another line of research work focuses on studying the generalization capabilities of over-parameterized neural networks trained with stochastic gradient descent methods. Neural networks with many more training parameters than the size of training data can generalize well even though they are capable of overfitting. Xu et al. [7] studied the training process of neural networks for a class of 1-D functions with more low-frequency components than the high-frequency ones. They showed that for such functions, the network focuses more on learning the dominant low-frequency components first and then successively adds the lesser high-frequency components to better fit the training data. They called this phenomenon Frequency principle (F-principle). Arpit et al. [8] showed that the networks learn simple patterns in the training data first before learning the detailed information. Rahaman et al. [9] and Xu et al. [10] used Fourier analysis to study the training of neural networks and showed similar results that the neural networks are biased towards learning the low-frequency components (the smoother functions that are global over the input space) first. These results explain why a good generalization can be achieved by the early stopping of the training process. Counter-intuitively, due to this spectral bias, neural networks have difficulty learning the fine details in the input data i.e. the high-frequency components. Since the fine details present in natural images and scenes are important for the task of view synthesis, the inability of MLPs to represent these details restricts their ability to generate good quality renderings.

2.3. Fourier Features

Some recent works [1, 2] introduced an approach to overcome the spectral bias of neural networks and enable them to learn the high-frequency content. Mildenhall et al. [1] in their initial approach, Neural Radiance Fields (NeRF), experimentally found that mapping the low dimensional input data (3D points and view directions) into high dimensional space enables the network to learn high-frequency scene content. They perform this mapping using sinusoids with logarithmically-spaced axis-aligned frequencies, which they call 'positional

encoding'. In follow-up work, Tancik et al. [2] further researched this 'positional encoding' approach and showed that it is a special case of Fourier Features [3]. Fourier Features were introduced in the work of Rahimi et al. [3], in which they approximated an arbitrary stationary kernel using Random Fourier Features. Tancik et al. [2] showed that mapping the input points with Fourier Features, before feeding them into MLP, transforms the Neural Tangent Kernel (NTK) [11] into a stationary kernel. The range of frequencies learned by the network can be controlled by modifying the frequency vectors in Fourier Features. Motivated by the success of using Fourier Features to enable the MLPs to learn high-frequency scene content, we used the same approach to map our low dimensional input coordinates to a higher dimensional space.

3. Background

Fourier Features were introduced in the work of Rahimi et al. [3] to overcome the high memory and computational cost of the technique called **Kernel Trick** [12] used by **Kernel Methods** [12]. To understand Fourier Features, we will first introduce Kernel methods and the concept of Kernel Trick in Section 3.1. In Section 3.2, we will discuss how Fourier Features were introduced to overcome the limitation of Kernel Trick and the motivation behind using Fourier Features for training neural networks. This first brings us to the question of how are kernel methods and neural networks related.

Jacob et al. [11] in their work titled *Neural Tangent Kernel (NTK)* [11] proposed a theory connecting kernel methods and neural networks. They showed that in the infinite width limit (when the number of neurons in the hidden layer grows to infinity), the behavior of neural networks over the training process can be approximated by a kernel, which they called a *Neural Tangent Kernel (NTK)*. Let us consider a network function f represented by a deep neural network and parameterized by θ. They showed that the behavior of f_θ during gradient descent is similar to kernel gradient descent in function space with respect to NTK. This means we can study the dynamics of the neural networks in terms of the kernel regression with a Neural Tangent Kernel.

3.1. Kernel Methods

Given a training dataset (inputs and corresponding outputs), we want to learn a function f that maps the input data X to the desired output value y. If the function f is a linear function of input X, then we can use linear regression to learn this mapping. This learning task becomes complex when the function we are trying to learn is non-linear in X, e.g. quadratic or polynomial. Kernel methods are one of the techniques used to learn this non-linear mapping. To understand kernel methods, we will first review a technique known as **Basis Expansion**. It can be understood as expanding the input features or embedding the input data into a higher-dimensional space.

The idea behind transforming the raw low-dimensional data into a higher-dimensional space is that the function we are trying to learn might not be linear in the input space. But it might be linear in some higher-dimensional space. By transforming the data, we force the network to operate in that higher-dimensional space where the model has to learn a linear function of the data points. But sometimes the expanded feature vectors can be very large in dimensions such that it is impractical to work in such a higher-dimensional space. One way to get around this problem is by using a technique called **Kernel Trick** [12].

Let's say we have two datapoints x_1 and x_2 and ϕ represents the mapping to the higher-

dimensional space. Then our corresponding data points in higher-dimensional space would be $\phi(x_1)$ and $\phi(x_2)$. To compute the inner product between these two data points, we don't need to compute the data points themselves i.e to compute $<\phi(x_1).\phi(x_2)>$, we don't need $\phi(x_1)$ and $\phi(x_2)$. A kernel function or kernel is a function in the low-dimensional space that gives the inner product of the data points in high-dimensional space i.e. $k(x_1, x_2) =< \phi(x_1).\phi(x_2) >$. In other words, the Kernel Trick is a technique by which the operations can be performed in high-dimensional space without explicitly mapping the data from low dimension to high dimension and this is made possible by the existence of a function called kernel function.

Let's take a training dataset (X, y) where X is the set of input values and y is the set of corresponding output labels. Let f be a function that maps input point x_i to output value y_i. At any input point x_j, we want to predict the output y_j. The kernel function can be used to estimate this output value as [2]:

$$y_j = \sum_{i=1}^{n} (K^{-1}y)_i k(x_i, x_j) \tag{3.1}$$

where K is the kernel matrix and $k(x_i, x_j)$ gives a similarity measure between data points x_i and x_j. This method of using kernel function as a weighting function is also known as **Kernel Method** or **Kernel Regression**.

3.2. Fourier Features

As discussed above, the Kernel Trick is a simple way by which the inner product of the data points can be evaluated in the high-dimensional space $< \phi(x), \phi(y) >$, without evaluating the implicit lifting ϕ of the data points. This is realized using the kernel function k. The problem with this approach is that it requires the evaluation of the kernel function for each pair of data points which grows quadratically with the increase in the size of training data. For massive datasets, this can lead to a huge increase in storage and computational cost. Rahimi et al. [3] proposed that instead of this implicit lifting, we can perform an explicit lifting of the input data to a lower-dimensional space z such that the inner product between the data points in z approximates the evaluation by kernel matrix [3] i.e.

$$k(x, y) =< \phi(x), \phi(y) >\approx z(x)^T.z(y) \tag{3.2}$$

The idea was to perform an explicit mapping of the input data points so that we don't have to rely on the implicit lifting performed by the kernel matrix. Since the dimension of this feature map is much lower than the input observations, it incurs less computational and storage cost compared to the kernel matrix.

One of the mappings demonstrated by Rahimi et al. [3], called **Random Fourier Features** is a set of random Fourier bases functions $\cos(\omega^T x + b), \omega \in \mathbb{R}^D, b \in \mathbb{R}$. The data point x is first projected onto a randomly chosen direction ω. The resulting scalar value is passed through a sinusoidal function which wraps it around a unit circle. They demonstrated that the inner product of the transformed points approximates a shift-invariant kernel when the

directions ω are sampled from an appropriate distribution, which is the Fourier transform of the corresponding kernel. Figure 3.1 from the work of Rahimi et al. [3] shows visually the mapping performed by the Random Fourier Features

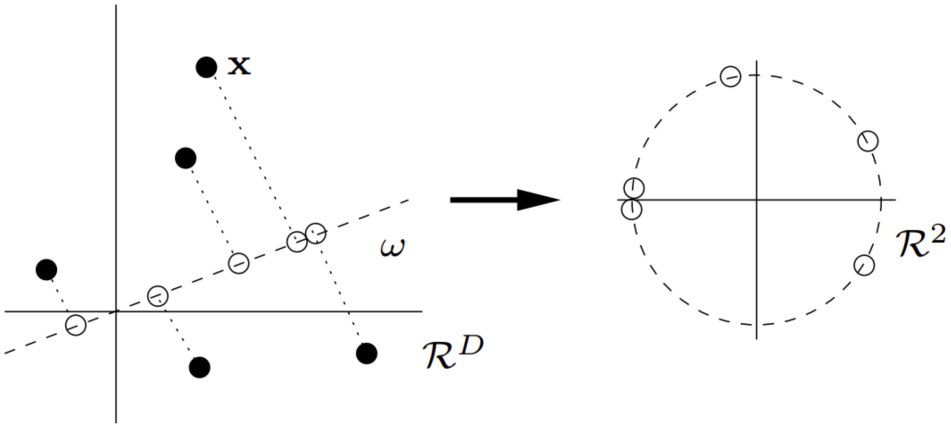


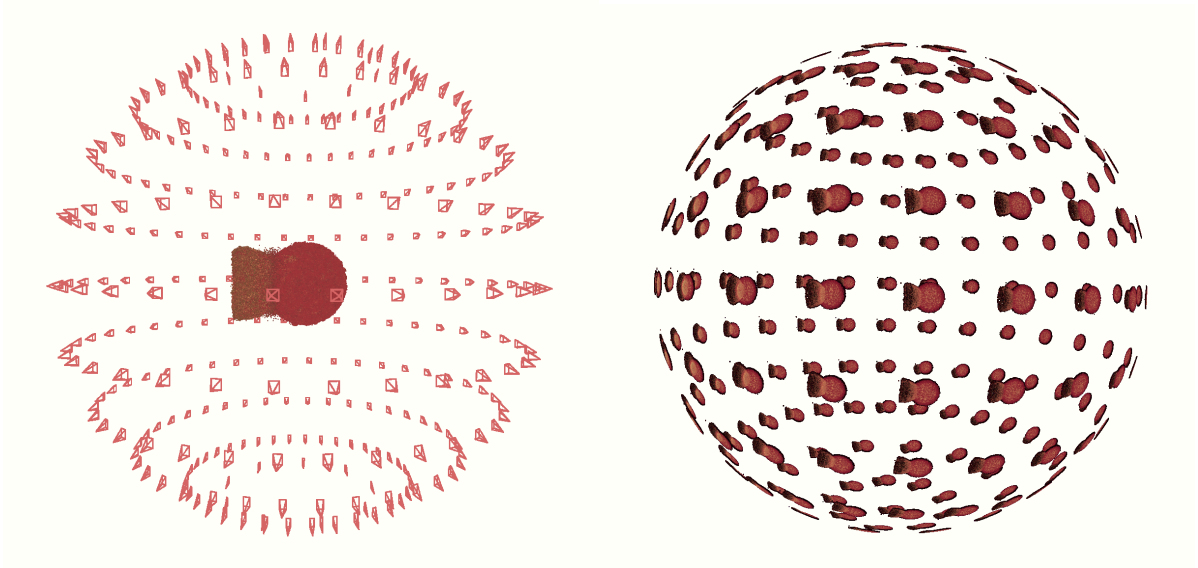
Figure 3.1.: Random Fourier Features [3]. The feature map $z(x) = \cos(\omega^T x + b)$ maps the data points from the input domain to the data points into a low/high dimensional domain. Each data point x is projected onto a random direction $\omega \in \mathbb{R}^D$. Passing the resulting scalar through a sinusoidal function wraps it onto a unit circle projecting the points onto the interval [0,1]. Rahimi et al. [3] demonstrated that the inner product of two data points (x_i, x_j) can be approximated by a shift invariant kernel when the random directions ω are sampled from a distribution which is the fourier transform of the corresponding kernel.

Motivation behind using Random Fourier Features A line of research work [13, 9, 14, 15] has shown that the fully connected deep neural networks (MLPs) are biased towards learning the low-frequency components in the data, a phenomenon referred to as **Spectral Bias**. Intuitively, this can be understood as that the networks prioritize learning simple patterns across the input data first. Counter-intuitively the networks have difficulty learning the high-frequency components i.e. the fine details in the data. This makes these networks unsuitable for view synthesis tasks since it is important for the network to learn the fine details in natural images and scenes to be able to synthesize visually appealing views of the scenes. Some recent works [2, 1] have shown that encoding the input data with Random Fourier Features [3] and using these encoded features as input to the network assists the network in learning the fine details in the input data.

4. Data Representation and Generation

In Section 4.1, we have discussed how we represent a 3D object i.e. we discuss our input representation. Section 4.2 explains how we generate this input representation. Section 4.3 introduces the datasets we have used.

4.1. Representation of 3D object



(a) Virtual Cameras positioned on a sphere

(b) Images of the object from multiple viewpoints projected on the bounding sphere

Figure 4.1.: Representation of a 3D object as a collection of its images from multiple viewpoints projected on its bounding sphere. (a) shows the position of the virtual cameras from which images of the object are rendered. (b) shows the respective images projected on the bounding sphere.

To represent a 3D object, we take a bounding sphere around the object. Then we project the input images of the object from multiple viewpoints onto the bounding sphere. The idea is to represent the object as a set of 3D points on the sphere and the associated view directions. Each 3D point when viewed from the corresponding view direction has a particular RGB color. This RGB color is obtained by the image projections on the sphere. To render an image of the object from a novel viewpoint, we give the network a set of 3D points on the sphere

along with the view direction corresponding to that viewpoint. The network has to predict the RGB colors on these 3D points which will give the desired image. The intuition behind this representation is that if the network can learn how the visual appearance and shape of the object changes over the sphere, it will be able to predict the RGB color of any point on the sphere when viewed from any possible view direction.

Figure 4.1 shows an example of this representation. Figure 4.1a shows the position of the virtual cameras from which the images of the object are rendered. Figure 4.1b shows these images projected onto the bounding sphere. To generate the visualization in Figure 4.1, we have taken the camera positions far from each other so that it is clearly visible how the object representation changes as we move around the sphere. The nearest camera positions are 15° away from each other. In this work, we describe the camera positions in terms of azimuth and elevation angle. By 15° away, we mean that either the azimuth angle is same and the difference between elevation angle is 15° or elevation angle is same and the difference between azimuth angle is 15°. In practice, if the viewpoints are not far, their corresponding projections on the sphere can overlap. An example of this can be seen in Figure 4.2. The corresponding viewpoints for the images shown are 6° away from each other. It could be seen that a point on the sphere can have a different color depending on the direction from which it is seen. Therefore in our input representation, it is important to take both view directions and 3D points to uniquely represent one pixel.

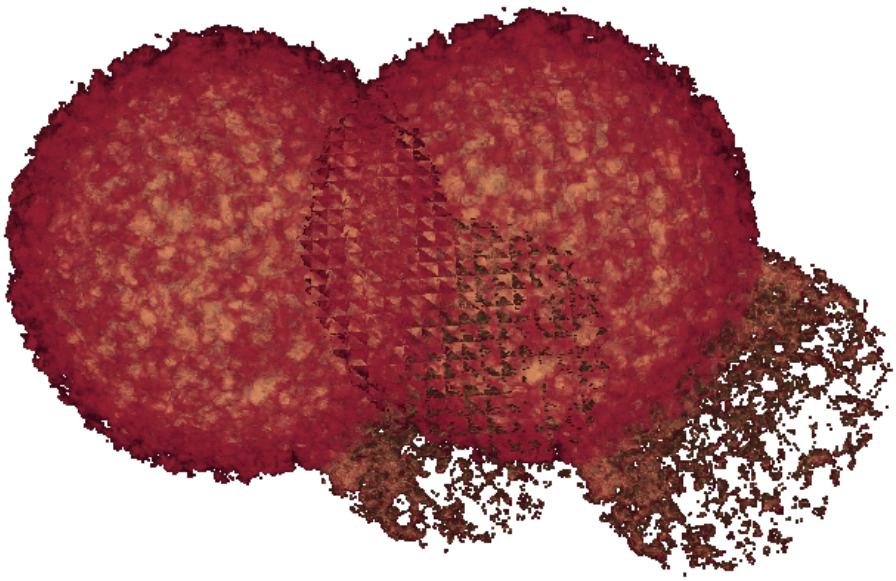


Figure 4.2.: Two input images projected on the bounding sphere. The elevation angle for corresponding camera positions is same and the difference between the azimuth angle is 6°. It can be observed that a point on the sphere can have different color value for different viewpoints. Therefore, to uniquely represent a pixel, we take both view direction and 3D point.

4.2. Generating Network Input

Our network takes as input 6D coordinate for each pixel of an input image. The 6D coordinate consists of a 3D point (x,y,z) and a 3D view direction (α, β, γ). So, the input for each ground truth image is of dimensions (H x W x 6). To generate the 6D coordinate for each pixel, we use the approach of raymarching.

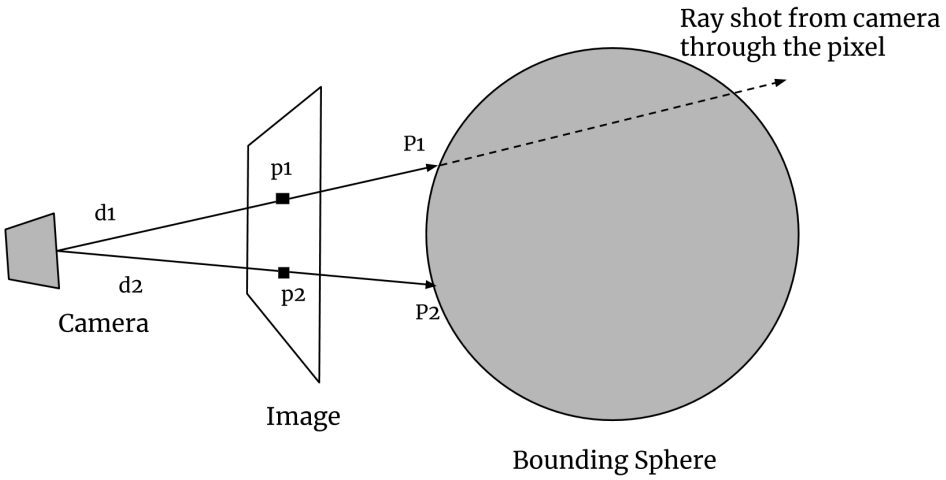


Figure 4.3.: Generation of input to the network i.e. pair of 3D points and ray directions. For pixels p_1 and p_2, rays are shot originating from the camera which intersects the bounding sphere at P_1 and P_2. The ray directions d_1 and d_2 combined with the corresponding intersection points P_1 and P_2 are fed as input to the network. Same process is applied for each pixel in the image.

We shoot a ray originating from the camera position towards the 3D object through each pixel in the input image. The camera position in the world coordinate is known. The world coordinate of each pixel can be calculated using camera intrinsics and extrinsics. We calculate the intersection of the ray with the bounding sphere and take it as the input 3D point. The direction vector of each ray serves as the input view direction. Figure 4.3 demonstrates visually the process of generating input data for two pixels p_1 and p_2. For an image of dimension H x W x 3, we have H x W x 6 dimensional input.

In other words, we feed a point on the sphere and the direction from which the point is viewed. The corresponding output is the RGB color on that point when viewed from the specified view direction. We want the network to learn a mapping such that it can predict the color on any point on the sphere when viewed from any possible view direction.

4.3. Datasets Used

Here we will discuss the datasets used in our work. We have used three datasets: Ejecta, Lego [1] and Shapenet [16]. Below we have explained the details of how we generate the

training data (images and camera parameters) or the sources from where we get the data.

Ejecta: Ejecta dataset is generated from a particle-based simulation of a supernova. We render the input images from the 3D volume via volumetric ray-casting. We used the implementation provided by Weiss [17] to render the images. The virtual camera from which the images are rendered can be visualized as moving on a sphere with a radius of 1.2. A visual representation of this can be seen in Figure 4.1a. We consider the 3D object to be centered at the world origin and take a bounding sphere of unit radius around it. Since we calculate the intersection point of the camera ray with the bounding sphere, our virtual cameras need to be positioned outside the bounding sphere. That's why we position the cameras on a sphere of radius 1.2. The camera parameters which we use for rendering the images gives us the intrinsic and extrinsic information which we need for generating the network input. The details of the camera positions used for each experiment are mentioned in the Experiments chapter (Chapter 6).

Lego: Lego dataset contains images of a toy Lego machine. We took the Lego dataset provided by Mildenhall et al. [1] which contains 100 training images and viewpoints, and 100 validation images and viewpoints. We were able to use the dataset provided by Mildenhall et al. [1] directly since it met our requirement for the camera positions to be outside the sphere of radius 1.0. The camera positions used to render Lego images are generated on a sphere of radius 4.0. The reason to use this dataset was to test our approach on the opaque objects first before testing it on a semi-transparent object like Ejecta. The reason being that semi-transparent objects have more information which the network will have to learn. So, if our approach does not work on opaque objects, there is a high probability it will not work on semi-transparent objects.

ShapeNet: ShapeNet [16] is a richly-annotated, large scale dataset of 3D shapes represented by 3D CAD models of objects. We took images of a couch and a chair. Here we will mention the sources from which we get the data. Later in the Experiments chapter (Chapter 6), we will discuss in detail why we specifically chose these two objects from the ShapeNet dataset. For couch data, we took the images (64 x 64 x 3) from the data made available by Choy et al. [18]. For chair data, we used the images (512 x 512 x 3) provided by Goel [19]. From these sources, we get 24 images for both couch and chair which we use for training. We get the information about the camera poses used for rendering the images from the mentioned sources but the poses are not made available. So, we generate camera positions based on the available information. For training data, the elevation of the virtual camera is kept at 30° and the azimuth is increased in steps of 15° starting from 15°. We generate some test viewpoints for which the elevation of the virtual camera is kept the same (30°) and the azimuth is increased in steps of 15° starting from 18°. We visualize the predictions made by the network on these test positions to get an understanding of how well the network is able to learn with given training data.

5. Method Overview

In Section 5.1 and Section 5.2, we have discussed the details of our network architecture and loss functions used respectively. We have experimented with two different input representations in optimizing the network. Section 5.3 explains these representations.

5.1. Network Architecture

Our neural network represents a continuous function that takes a 6D input (point + view direction) and outputs the RGB color at that point as viewed from the specified direction. Motivated by the success of MLPs [1] to represent a 3D scene, we used the deep fully connected neural network to represent this function. Instead of using the Fully Connected (FC) layers, we used convolutional layers with a kernel size of 1x1. This gives the same effect as using FC layers. The reason behind using the convolutional layers instead of FC layers is that we will not have to reshape the input tensors in the case of convolutional layers. Since the images and convolutional layers both are two dimensional, we can use our input and ground truth data without any restructuring of tensors.

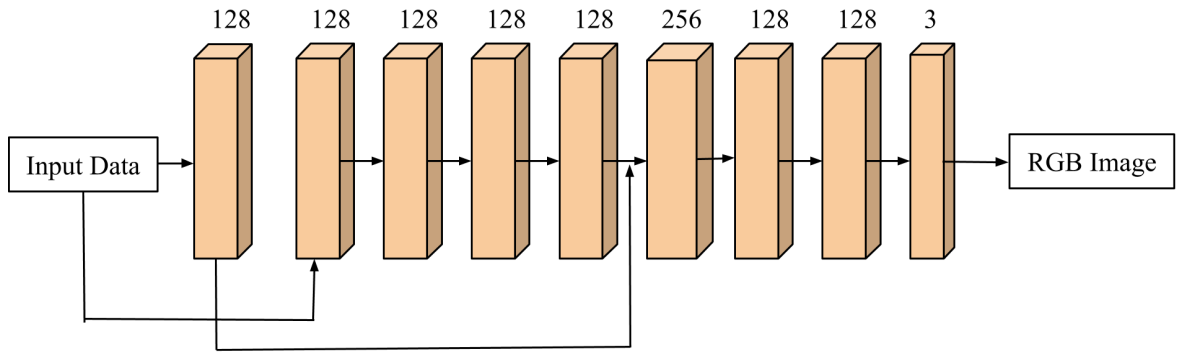
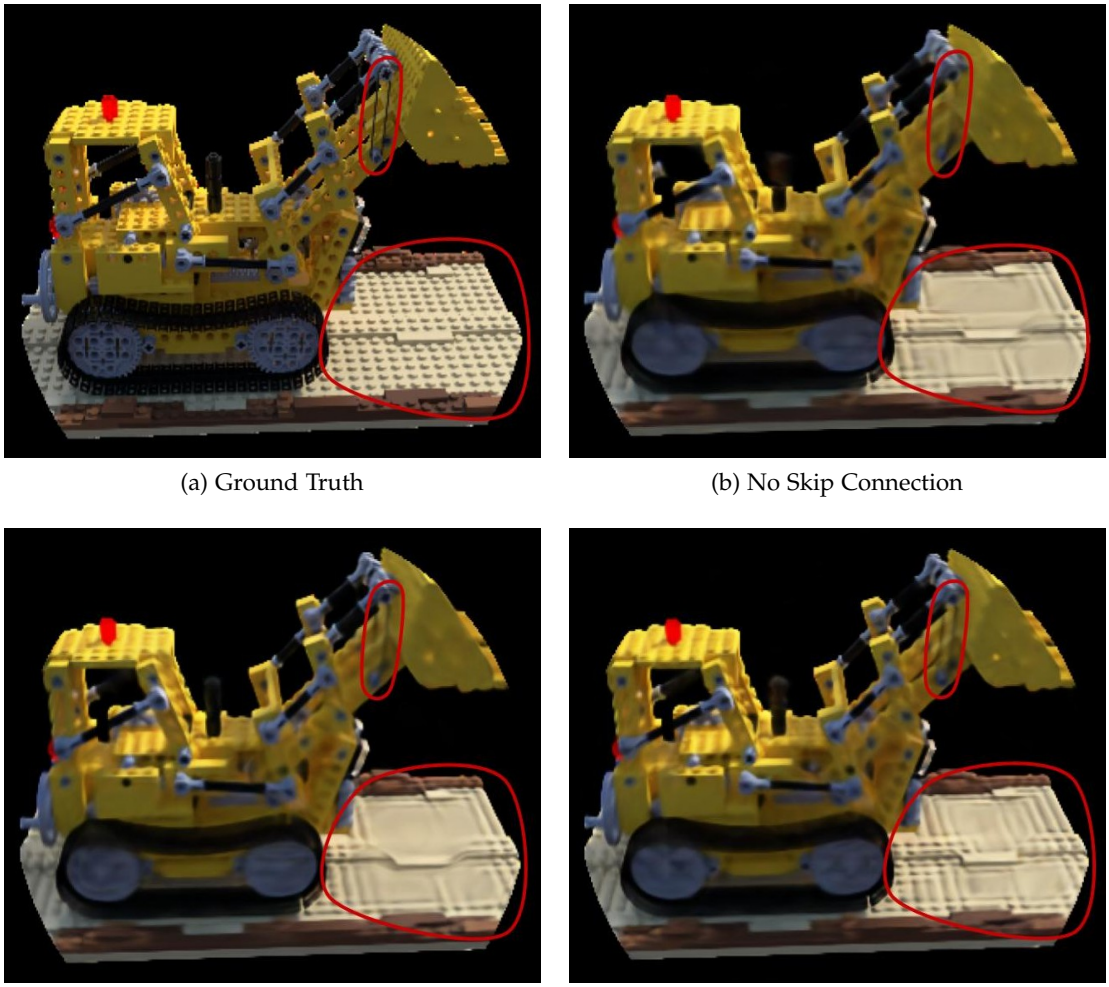


Figure 5.1.: ConvNet: Network Architecture. We have used 8 convolutional layers with the channel size of 128, 256, and 3 and kernel size of 1x1. We used a skip connection after the 4th layer. The input data is first passed through one convolutional layer with channel size of 128. The output of this layer is then concatenated with the output of the 4th layer as skip input. The output of the 7th convolutional layer is passed through the last convolutional layer with channel size of 3 which gives us the RGB predictions. Each layer (except last layer) is followed by ReLU.

Figure 5.1 shows the architecture of our network which we call **ConvNet**. We have used 8 convolutional layers with channel size of 128, 256 and 3 and kernel size of 1x1. Each layer



(a) Ground Truth

(b) No Skip Connection

(c) Skip Connection with Input Data

(d) Skip Connection with output of skip layer

Figure 5.2.: Effect of using skip connection: To see the effect of using skip connection, we trained for three variants of network architecture: (a) Without any skip connection, (b) Input data is skip connected to the output of 4th layer, (c) Input data is passed through a convolutional layer, which we call the skip layer. The output of this layer is skip connected to the output of the 4th layer. We observed that the third variant performs better among all three. We have highlighted some regions in the images above with the red curve where the differences are more visible. One image from the Lego dataset is used as ground truth for the network. The predictions show how well the network can reconstruct the same image. It can be observed that the predictions are less blurry (red circle in the bottom right) in the case of the third variant. Also, some parts are better reconstructed (red circle in the top right). Motivated by these observations, we used the third variant in our network architecture.

(except last layer) is followed by ReLU. We have used a skip connection in our architecture and we pass the input through a convolutional layer with channel size of 128 before skip connecting it to the output of the 4th layer. We reached this design choice after performing an experiment to visualize the effect of using skip connection. We trained the network for three variants of network architecture: (a) Without any skip connection, (b) Input data is skip connected to the output of the 4th layer, (c) Input data is passed through a convolutional layer, which we call the skip layer. The output of this layer is skip connected to the output of the 4th layer. The network is trained on one image from the Lego dataset i.e we take one image along with its camera parameters. We generate our input representation (3D points + view directions) using the camera parameters. This input is fed into the network and the network is trained to predict the RGB colors on the 3D points. We calculate the L_1 loss between the

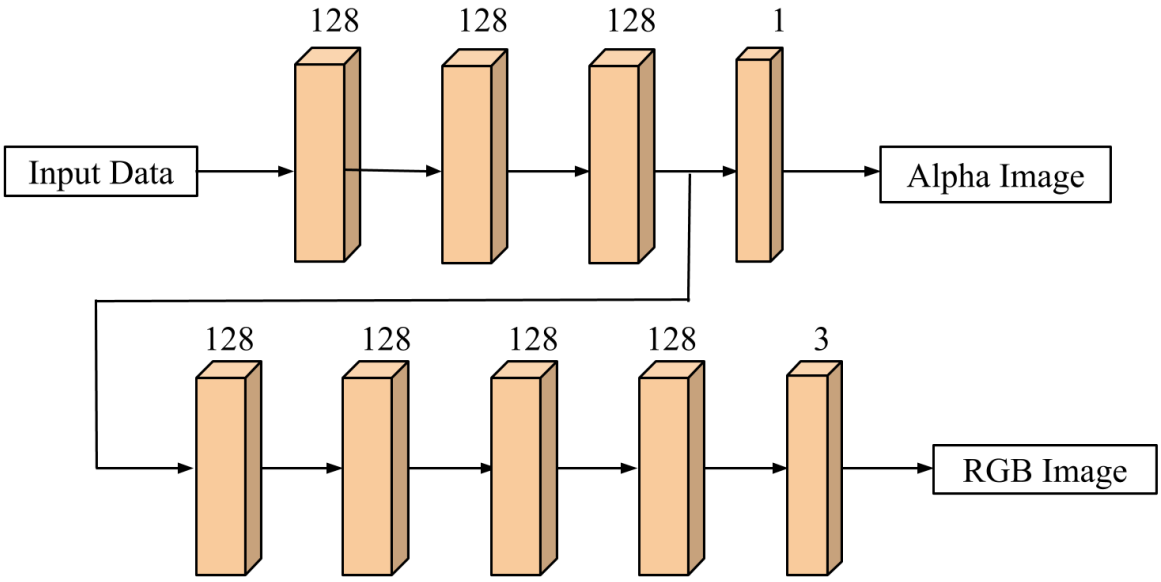


Figure 5.3.: Conv2Net: Network Architecture. We divide the network into two parts. The first three convolutional layers constitute the first part of the network. The output of the 3rd layer is passed through an additional convolutional layer with channel size 1 which gives us the alpha values. The next 5 convolutional layers constitute the second part. The output of the last convolutional layer with channel size 3 gives us the RGB values. Each layer (except last layers with channel size 1 and 3) is followed by ReLU. We calculate the loss on both the alpha and RGB output. The intuition behind such an architecture is that the first part of the network focuses on learning the mapping from input to opacity (alpha) values so that the second part of the network can focus on learning only the color values. This can also be understood as the input going into the second part of the network is being trained to be already closer to ground truth in terms of alpha values. Thus, the second part can give more weight to improving color values.

predicted image and the ground truth image. The network is trained at a learning rate of 1e-4. The experiment was performed to see which variant of network architecture can reconstruct the ground truth image better. Figure 5.2 shows the network predictions for all the three variants. We observed that the third variant gives better predictions. That is why, we use the third variant in our network architecture.

We also experimented with using another network architecture for our task. We call this second architecture **Conv2Net**. Here we will introduce this network architecture. Later in Section 6.4, we will discuss how well this architecture performs on predicting the RGB images from a given viewpoint and the comparison between the two architectures (ConvNet vs Conv2Net). Figure 5.3 discusses the details of the architecture. We divide the network into two parts where the first part outputs the alpha image and the second part outputs the RGB image. The first part of the network consists of 4 convolutional layers with channel size of 128 and 1. The second part consists of 5 convolutional layers with channel size of 128 and 3. All layers (except last layers with channel size of 1 and 3) are followed by ReLU. The intuition behind this architecture is that the first part of the network focuses on learning the mapping from input to opacity (alpha) values so that the second part of the network can focus on learning only the color values. This can also be understood as the input going into the second part of the network is being trained to be already closer to ground truth in terms of alpha values. Thus, the second part can give more weight to improving color values.

5.2. Loss Functions

In this section, we will introduce all the loss functions used in our work. For most of the experiments, we have used L_1 or L_2 loss since they are the most common and standard loss functions used for the image processing tasks.

***L_1* loss:** L_1 error is the mean of the absolute difference between the target vector t_i and predicted vector p_i. It is also known as Mean Absolute Error (MAE).

$$L_1 = \frac{1}{n} \sum_{0<=i<n} |t_i - p_i| \tag{5.1}$$

***L_2* loss:** L_2 loss is the mean of the squared difference between the target vector t_i and predicted vector p_i. It is also known as Mean Squared Error (MSE).

$$L_2 = \frac{1}{n} \sum_{0<=i<n} |t_i - p_i|^2 \tag{5.2}$$

SSIM loss: SSIM [20] stands for Structural Similarity Index Measure. SSIM is a perceptually motivated loss that takes into account luminance, contrast, and structural information while calculating the difference between two images. This loss considers the idea that when two pixels are spatially close to each other, they are strongly interdependent and this dependency information somehow represents the structure of the objects. The motivation behind using this metric over L_1 and L_2 loss is that L_1 and L_2 do not consider the relationship between

neighboring pixels since they calculate the loss on a per-pixel basis. Zhao et al. [21] discusses the effects of using SSIM loss for image processing tasks. We have used the SSIM implementation provided by Fang [22].

SSIM index is calculated on various windows of an image. The SSIM measure between two windows x and y of same size NxN is calculated as:

$$SSIM(x,y) = [l(x,y)^{\alpha}.c(x,y)^{\beta}.s(x,y)^{\gamma}] \tag{5.3}$$

$$l(x,y) = \frac{2\mu_x\mu_y + c_1}{\mu_x^2 + \mu_y^2 + c_1} \tag{5.4}$$

$$c(x,y) = \frac{2\sigma_x\sigma_y + c_2}{\sigma_x^2 + \sigma_y^2 + c_2} \tag{5.5}$$

$$s(x,y) = \frac{\sigma_{xy} + c_3}{\sigma_x\sigma_y + c_3} \tag{5.6}$$

SSIM metric between two windows depends on three comparison measurements: luminance (l), contrast (c) and structure (s). Means and standard deviations are calculated using a Gaussian filter. The mean of the SSIM indices over all the windows then gives the SSIM index for the image. The loss over SSIM metric can be calculated as below where i corresponds to each window pair (x, y):

$$L_{SSIM}(i) = \frac{1}{n} \sum_{0<=i<n} (1 - SSIM(i)) \tag{5.7}$$

5.3. Fourier Encoding

As discussed in Section 3.2, some recent works [1, 2] have shown considerable improvement in the network performance by encoding the input data with Random Fourier Features [3] before feeding them into the network. Inspired by their work, we have also used these Fourier Encodings to encode our input representation. We have used two input representations:

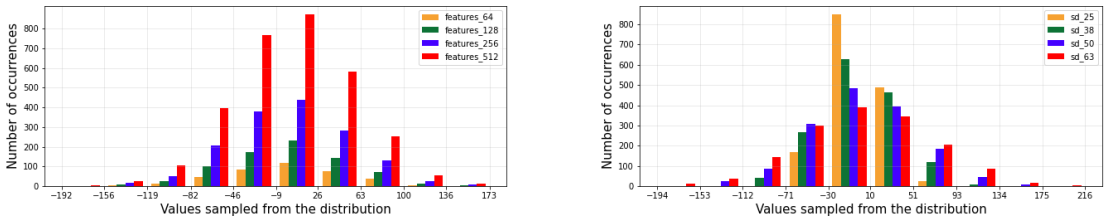
Raw Input Data: Each image pixel is represented by a 6D vector $v = (x, y, z, \alpha, \beta, \gamma)$ where (x, y, z) is a 3D point on the bounding sphere and (α, β, γ) is the 3D unit direction vector specifying the direction from which the point is viewed. In other words, for an image of dimensions (HxWx3), we have an input of dimensions (HxWx6).

Fourier Encoded Data: As mentioned above, each image pixel is represented by a 6D vector v. We pass this vector v through a feature mapping ϕ as shown below:

$$\phi(v) = [\cos(Bv), \sin(Bv)] \tag{5.8}$$

$B \in \mathbb{R}^{mxd}$ is a randomly sampled Gaussian matrix. Each entry in this matrix is randomly sampled from $\mathcal{N}(0, \sigma^2)$. The idea is to map the input data from d dimensional input space to m dimensional input space. Then we pass the resultant vector through *cosine* and *sine* which wraps our input data onto a high-dimensional hypersphere. Then we concatenate this

resultant vector with the original d-dimensional vector. For a d-dimensional input vector, we get a $(2m + d)$-dimensional Fourier Encoded input vector. In our case, d is 6 since we have 6D input. By changing the value of m (feature size), we can change the number of features or frequency values sampled from the underlying distribution. By changing the standard deviation (SD) of the distribution, we can control the range of frequency spectrum. If we take a higher standard deviation, more high frequencies will be fed as input to the network than in the case of a lower standard deviation. Figure 5.4 shows an example of the sampled Gaussian matrix B and the effect of change in sampled values with the change in standard deviation (SD) and feature size.



(a) (b)

Figure 5.4.: Plots showing the change in the values of randomly sampled Gaussian matrix B with change in one of the hyper-parameters: standard deviation (SD) or feature size (FS). In (a), we take the SD of 50.0 and plot the sampled feature values for four feature sizes: 64, 128, 256 and 512. Since the range from which the feature values can be sampled is same (same SD), we see an increase in the values sampled in all the ranges. In (b), we take the feature size of 256 and plot the sampled feature values for four standard deviations: 25.0, 38.0, 50.0 and 63.0. As we increase the standard deviation, the range of values increases. But since the feature size is same, we see a decrease in the number of low frequencies sampled for high SD relative to low SD. And for high SD, some values from a higher value range are present which are not present for low SD.

Tancik et al. [2] have shown that there is no defined way to know which Fourier Encoding is best suited for a particular task and data. To find the optimal feature mapping, the standard deviation (SD) needs to be tuned like a hyper-parameter. Tancik et al. [2] have also mentioned that at the optimal standard deviation, the choice of feature size m does not have a considerable effect. In our work, we have considered both the standard deviation (SD) and feature size as hyper-parameters to study the effect they have on the performance of the network.

6. Experiments

As explained in Section 4.1, we represent a 3D object by projecting the images on the bounding sphere. Each image pixel is represented by a pair of a 3D point on the sphere and the corresponding view direction. Inspired by the success of Random Fourier Features [3] in training neural networks to learn fine details [2, 1] in input data, we also encoded our input representation with the Fourier Features and trained the network on encoded data. The details on how we generate this encoded data has been explained in Section 5.3.

We start by showing how our approach works on isosurfaces and opaque objects. Then we move on to analyze the network performance on semi-transparent objects. In Section 6.1, we compare the performance of the network when trained on our raw representation (points on bounding sphere + view directions) and Fourier Encoded data. In Sections 6.2 and 6.3, we will discuss the performance of network when trained on Fourier Encodings with different standard deviations and with different feature sizes respectively. In Section 6.4, we will show the comparison of our two network architectures (ConvNet vs Conv2Net). In Section 6.5, we will show the comparison in network predictions when trained on different loss functions.

The difference between our approach and the NeRF approach [1] is that in NeRF, Mildenhall et al. [1] step through each ray passing through the volume and sample multiple points on each ray. We instead consider the first intersection point of the ray with the bounding sphere. To study how important is this approach of sampling multiple points on a ray, we tested with the same approach on our data of semi-transparent objects (Ejecta dataset). We will call it ray sampling. In the Section 6.6, we will discuss our observations on the importance of ray sampling.

6.1. Raw data vs Fourier Encoded Data

We performed this experiment on three different datasets: Ejecta, ShapeNet [16] and Lego [1]. We will start by discussing the performance of our approach on isosurfaces. An isosurface is a 3D surface representation of all the points with a constant value. For example, all the points in a volume with the same volume density can be represented by an isosurface. The reason why we chose to start with an isosurface is that there is lesser information for the network to learn compared to if we train the network on the renderings of a semi-transparent object. Isosurfaces deal with how a particular function value changes on the surface of the object, whereas in the case of renderings of a semi-transparent object the network has to learn how the information changes in the whole volume. Also in the case of isosurface, the network has to learn only the surface geometry of the object without focusing on how the texture information changes. With this motivation, we first tested our approach on the isosurface

(a) Without Ambient Occlusion (b) With Ambient Occlusion

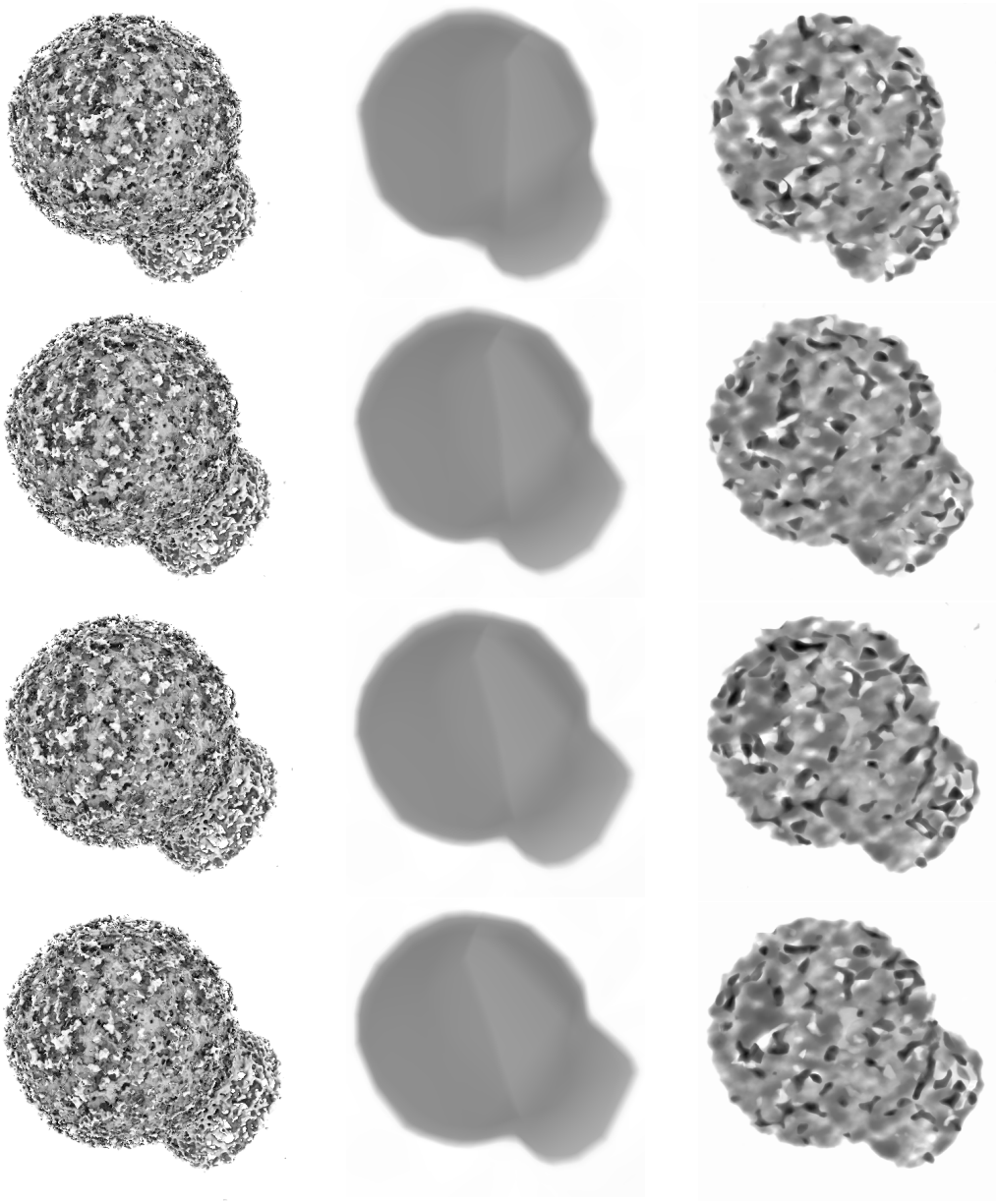
Figure 6.1.: Scene with and without Ambient Occlusion (`https://vr.arvilab.com/blog/ambient-occlusion`)

of the Ejecta dataset. We randomly chose to use the Ambient Occlusion values to represent Ejecta isosurface.

In computer graphics, global illumination is simulated by ambient light. Ambient Occlusion (AO) is a rendering technique that calculates how much of this ambient light is received by each surface point. AO is estimated by emanating rays from a surface point in all directions and checking for intersection with other surface points. Each ray that reaches the background or sky increases the brightness of the point. Rays that get occluded by other points do not add to the brightness. Figure 6.1 shows an example of a scene with and without Ambient Occlusion. On the surface of an object, AO enhances the cavities or locations surrounded by other surface points. AO at a surface point can be thought of as a function of how much it is surrounded by neighboring points.

For this experiment, we took the input data on a small portion of the sphere. We generated the viewpoints by keeping the elevation at 60° and increasing the azimuth from 30° to 60° in steps of 1°. We split the generated input data into 27 training viewpoints and 4 validation viewpoints. To make sure that the amount of training data does not restrict the network to generalize, we took more viewpoints for training. The reason why we experimented with the data on a small portion of the sphere instead of taking the whole data is that if the network is not able to generalize on this data for a particular standard deviation, it will not be able to generalize when the data is spread over the whole sphere. The reason being that there will be more information which the network will have to learn in case of the whole sphere. Note that in this case, we are changing the camera positions in one direction (azimuth angle) while in the case of data on the whole sphere, both azimuth and elevation angle will change.

We trained the network on our raw input representation (3D points on sphere + view direction) and Fourier Encoded data. For Fourier Encoding, we took the standard deviation of 32.0 and feature size of 256. We have used ConvNet architecture and trained the network with a learning rate of 1e-4 and the L_1 loss function. Figure 6.2 shows the network predictions on the validation data for both the input representations.



Ground Truth Raw Input Data Fourier Encoded Data

Figure 6.2.: Network predictions on validation data of Ejecta isosurface dataset for different input representations. With the raw input representation, the network is able to differentiate between the object surface and background (i.e. it can generalize on the shape) but it can not learn the change in the intensity values over the surface of the object. With Fourier Encoded data, the network is able to differentiate between the surface points with large difference in the intensity values.

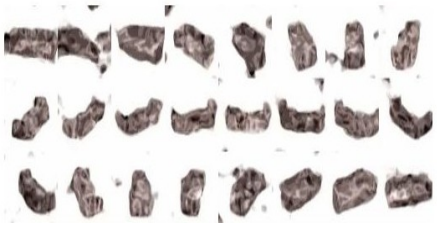
	Training Data	Test Data
Ground Truth		
Network Predictions on Raw Input Data		
Network Predictions on Fourier Encoded Data (SD-4.0)		
Network Predictions on Fourier Encoded Data (SD-10.0)		

Table 6.1.: Ground Truth and Model predictions on ShapeNet Couch data (training and test data). Our raw input representation is sufficient for the network to generalize on the shape of the object in case of simple shapes like Couch but can not learn fine details in the texture. With Fourier Encoding, the network is able to learn fine details like checkered pattern on the couch. With increase in standard deviation of Fourier Encoding, the quality of network predictions on training data increases. On test data, network learns more details with increase in standard deviation but starts losing the generalization capability on shape.

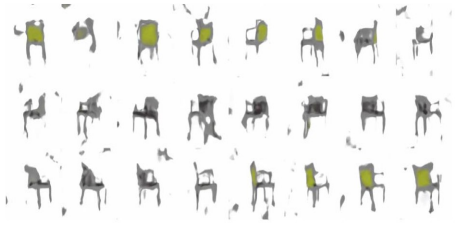
	Training Data	Test Data
Ground Truth		
Network Predictions on Raw Input Data		
Network Predictions on Fourier Encoded Data (SD-4.0)		
Network Predictions on Fourier Encoded Data (SD-10.0)		

Table 6.2.: Ground Truth and Model predictions on ShapeNet Chair data (training and test data). With our raw input representation, the network can somewhat predict the shape of the chair but the predictions are fuzzy at places like legs and arms of chair. With Fourier Encoding, the network better predicts the shape of the chair. With increase in standard deviation, the quality of network predictions on training data increases. On test data, the color predictions become less blurry with increase in standard deviation but the network starts losing the generalization capability on shape.

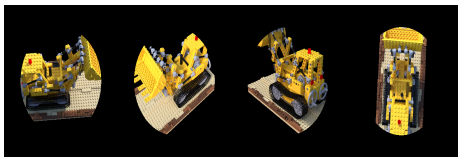
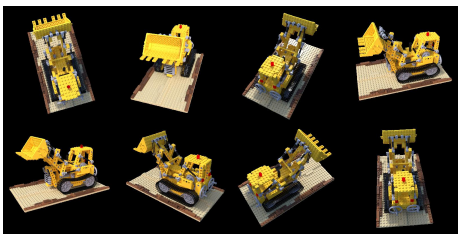
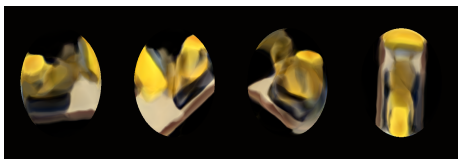
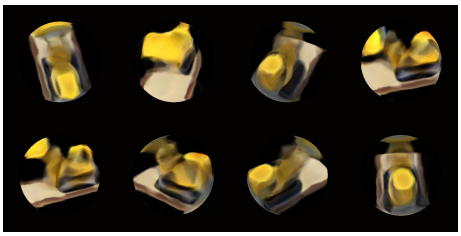
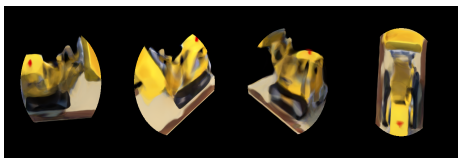
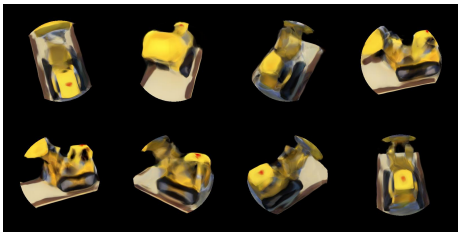
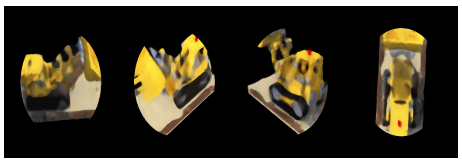
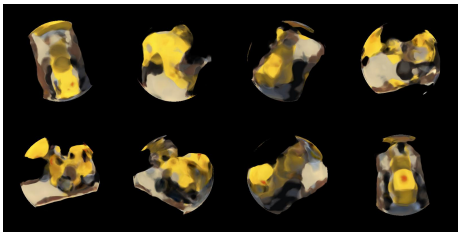
	Training Data	Validation Data
Ground Truth		
Network Predictions on Raw Input Data		
Network Predictions on Fourier Encoded Data (SD-2.0)		
Network Predictions on Fourier Encoded Data (SD-4.0)		

Table 6.3.: Ground Truth and Network predictions on Lego dataset (training and validation data).With our raw input representation, the network is able to somewhat predict the shape of the machine but the shape prediction is fuzzy. Also it is able to predict the major color values but no fine details. With Fourier Encoding, network is able to predict the details in the color information and the shape predictions also become better on training data. With increase in standard deviation, the prediction quality on training data increases. On validation data the generalization capability of the network starts decreasing with increase in standard deviation.

From the experiment on the Ejecta isosurface dataset (Figure 6.2), we observed that with our raw input representation (column 2 of Figure 6.2), the network can differentiate between the foreground and background pixels but it is not able to learn the change in intensity values over the surface of the object. In other words, the network can generalize on the shape of the object but is not able to learn high frequency information. With Fourier Encoded data (column 3 of Figure 6.2), the network can differentiate between the surface points with a large difference in intensity values i.e. it can learn some details on the surface of the object.

To explore more into the capability of the network to generalize on the shape of the object, we tested our approach on images of objects with different shapes. We trained the network on ShapeNet (Couch and Chair) and Lego dataset. Also since with Fourier Encodings, the network is able to learn the change in intensity values over the surface of the object, we took it a step further and trained the network on RGB images.

Tables 6.1 and 6.2 show the ground truth and network predictions on Couch and Chair data from ShapeNet dataset [16] respectively. Table 6.3 shows the same for Lego dataset. For each of them, we have trained on multiple standard deviations to have an insight into effect of SD on network performance. As explained in Section 5.3, standard deviation (SD) is one of the hyper-parameters which has to be tuned for a particular dataset and task. From the network predictions on all these objects (Tables 6.1, 6.2, 6.3 and Figure 6.2), we observed that:

- With our input representation, the network is able to generalize on the shape of the object in case of simple shapes like Ejecta and Couch.
- With Fourier Encodings, the network can learn fine details in the appearance of the object and is also able to better predict the shape of the object in case of complex shapes like Couch and Lego. Our intuition is that as the shape becomes complex, there are more details to be learned to generalize on the information that describes the shape. It has been observed in a recent work [2] that Fourier Encodings assist the neural networks in learning fine details in input data (high-frequency information). This could be the reason why the performance of the network in predicting the shapes increases with Fourier Encodings.
- With an increase in standard deviation, the quality of network predictions on training data increases. On validation or test data, Fourier Encoding gives better network predictions than raw input representation. But the network starts losing the generalization capability on unseen data as we increase the standard deviation. This gives us an introductory insight into the effect of standard deviation on the quality of network predictions. We will discuss in detail about this effect in Section 6.2.

Now we will discuss how our approach performs on the renderings of semi-transparent objects. We will work on the direct volume renderings of the Ejecta dataset. Table 6.4 shows the ground truth images and the images predicted by the network. We render the images as discussed in Section 4.3. The training data consists of 212 viewpoints and validation data consists of 315 viewpoints. To generate the input data, we move the virtual camera in steps of 15° for both azimuth and elevation. For training data, we start with an angle 0° for both

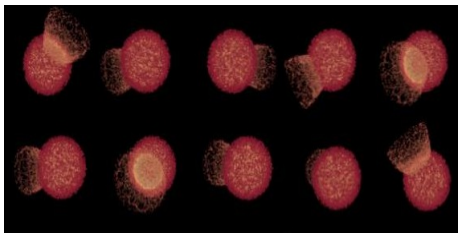
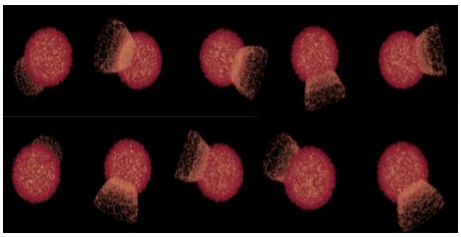
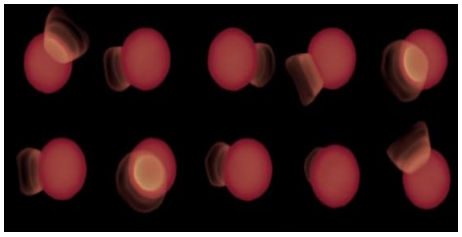
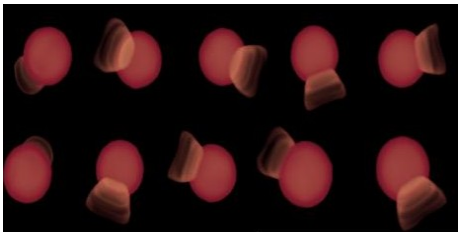
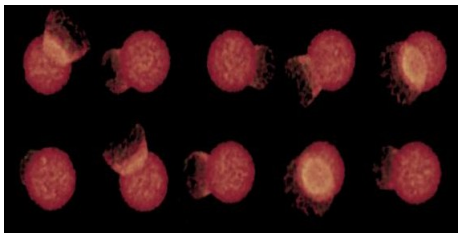
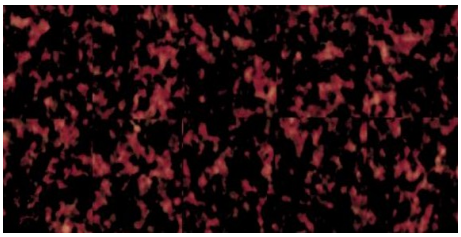
	Training Data	Validation Data
Ground Truth		
Network Predictions on Raw Input Data		
Network Predictions on Fourier Encoded Data (SD-32.0)		

Table 6.4.: Ground Truth and Network predictions on direct volume renderings of semi-transparent Ejecta dataset (training and validation data). With our raw input representation, the network is able to generalize on the shape of the object and predict major colors but the network is not able to learn the fine details. With fourier encoding, the network is able to learn fine details i.e. high frequency information on training data but gives noisy predictions on validation data.

azimuth and elevation. For validation data, we start with an angle of 3° for both azimuth and elevation. The difference between the nearest training and validation viewpoints is 3° for both azimuth and elevation. The images are rendered at a resolution of (512 x 512). Our choice of camera positions and the image resolution is completely random. With this input data, we trained the network on two input representations: Raw Input Data (points on bounding sphere + view directions), and Fourier Encoded Data with a standard deviation of 32.0. The feature size used for Fourier Encoding is 256. The network is trained on L_1 loss with a learning rate of 1e-4.

We observed that on isosurface (Figure 6.2) and opaque objects (Tables 6.1, 6.2 and 6.3), the network is able to generalize on the shape of the object but is not able to learn fine details in the input data with our input representation. We observed similar behaviour on

semi-transparent object (Table 6.4). Our input representation is sufficient for the network to generalize on the shape but no fine details are learned. With Fourier Encodings, the network can learn high-frequency information on training data. Using Fourier Encodings with our input representation becomes important for our task since for good-quality view synthesis, it is important for the network to learn fine details in the input data. We also observe that with the current Fourier Encoding, the network gives noisy predictions on validation data. There could be multiple reasons for this loss in generalization capability:

- With Fourier Encoding, the network becomes powerful enough such that the available training data becomes insufficient for the network to generalize over the input domain. Due to which, though the network gives good predictions on training data, it gives noisy predictions on validation data. To solve this problem, we can use more training data.
- The hyper-parameters used for this experiment i.e. standard deviation (SD) of 32.0 and feature size of 256 are not optimal for the available training data. This led us to explore more into the effect of these hyper-parameters (standard deviation and feature size) on the network performance with a sufficient amount of training data. We will discuss these in detail in Sections 6.2 and 6.3.

6.2. Effect of standard deviation

In Section 6.1, we observed that with an increase in the standard deviation of the Fourier Encoding, the network can learn the fine details in the training data but starts losing generalization capability on validation or test data. To study this effect in detail, we trained the network with 12 different standard deviations with a feature size of 256. We used the same camera viewpoints that we used for the experiment on Ejecta isosurface (on a small portion of the sphere (Figure 6.2)). The details of the camera viewpoints have already been described in Section 6.1. The reason why we experimented with the data on a small portion of the sphere is that if the network is not able to generalize on this data for a particular standard deviation, it will not be able to generalize when the data is spread over the whole sphere. We trained the network with L_1 loss and a learning rate of 1e-4.

Figures 6.3(a) and (b) show the plot of training loss and validation loss against the standard deviation respectively. We can observe in Figure 6.3(a) that with an increase in standard deviation, the training loss decreases i.e the capability of the network to learn more information over the input domain increases. However in Figure 6.3(b), we observe that the validation loss first decreases and then starts increasing with an increase in standard deviation. This means that if the standard deviation is too less, the network exhibits underfitting i.e. it is not able to learn enough information both on training and validation data. If the standard deviation is too high, the network overfits i.e. it learns more information on training data but then loses the generalization capability thus giving noisy predictions on validation data.

Figure 6.4 shows the training and validation loss curves over the optimization process for different standard deviations. It can be observed that when the standard deviation is less, the

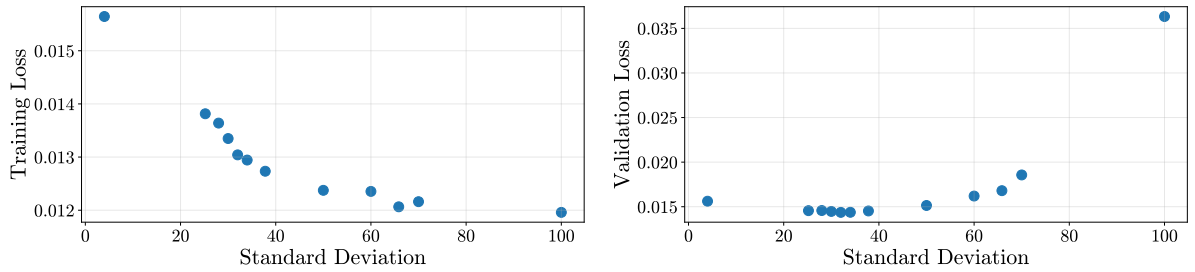


Figure 6.3.: Scatter plot of loss values depicting the effect of standard deviation of sampled fourier features on network performance. With increase in standard deviation, the training loss decreases but validation loss first decreases and then starts increasing. In other words, initially the network learns more information in the input data with the increase in standard deviation. But after a certain point, the network starts losing the generalization capability leading to increase in validation loss. The network exhibits underfitting on very low standard deviations and overfitting on high standard deviations.

network reaches the saturation point earlier and then does not improve much. For higher standard deviation, the network takes more time to improve on the training data but does not improve much on validation data. This can be visualized clearly in the loss curve for standard deviation 100.0.

In Figures 6.5 and 6.6, we show the predictions made by the network on training and

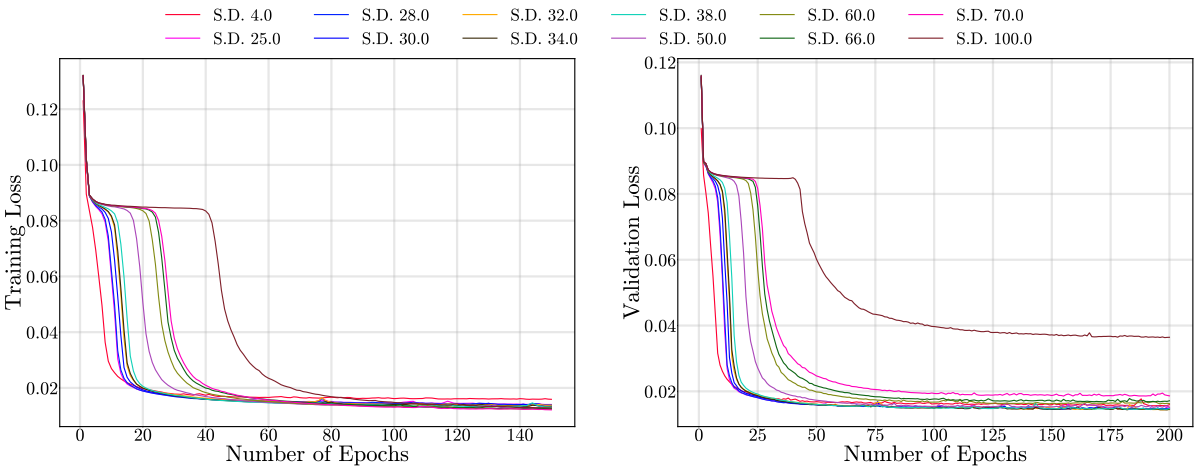
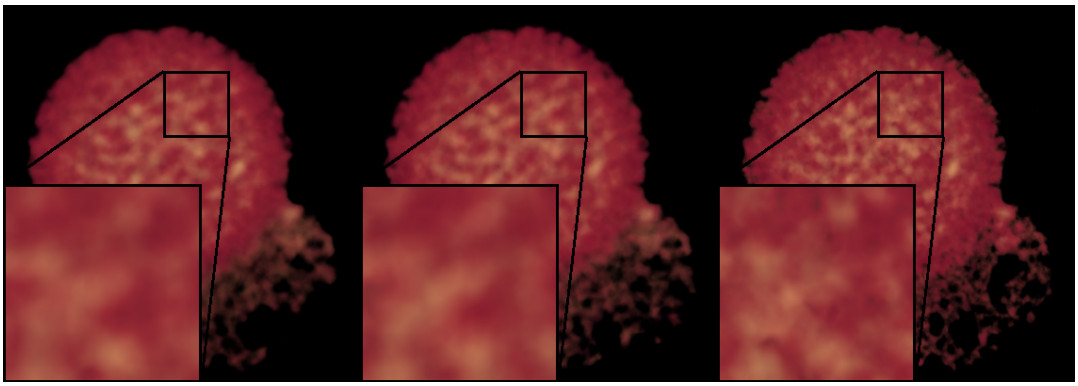


Figure 6.4.: Training and validation loss curves depicting the effect of standard deviation of sampled fourier features on network performance. The network reaches convergence faster for low standard deviations. For very high standard deviations, the network takes more time to converge on training data but does not improve much on validation data (as can be seen in the loss curve for SD 100.0)

(a) Ground Truth (b) SD-4.0 (c) SD-32.0



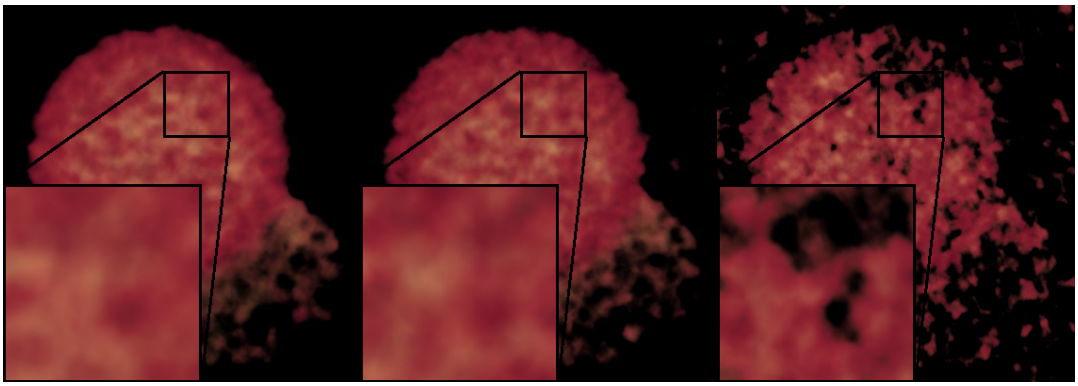
(d) SD-50.0 (e) SD-60.0 (f) SD-100.0

Figure 6.5.: Network Predictions on Ejecta Training Data: In this figure, we show the network predictions on the training data for different standard deviations. It can be observed that with increase in standard deviation of the fourier encoding, the model is able to better predict the fine details in the data.

validation data respectively for 6 different standard deviations. The image visualizations support our observations (Figure 6.3). With the increase in standard deviation, the capability of the network to learn high frequency information on the training data increases (Figure 6.5). On validation data the quality of network predictions first increases and then decreases (Figure 6.6). We can observe that for very high SD (SD 100.0), the prediction on the validation data is noisy (Figure 6.6(f)).

Our insights from this experiment are that for a particular scene or object, there exists an optimum standard deviation point at which the network shows better generalization capability than other standard deviations. After this point, we will observe a decrease in the quality of the predictions on the validation data. Until now, this optimum point could only be found by a brute force search over the possible standard deviations.

(a) Ground Truth (b) SD-4.0 (c) SD-32.0



(d) SD-50.0 (e) SD-60.0 (f) SD-100.0

Figure 6.6.: Network Predictions on Ejecta Validation Data: In this figure, we show the model predictions on the validation data for different standard deviations. The quality of the predictions first increases and then decreases with increase in standard deviation. For very high standard deviations, the model starts giving noisy predictions (as can be seen in the case of SD-100.0)

6.3. Effect of Feature Size

In this section, we will discuss the effect of feature size (i.e. the number of Fourier Features sampled from the underlying distribution) on the performance of the network. To study this, we sampled 64, 128, 256, and 512 Fourier Features for four different standard deviations (25.0, 38.0, 50.0, 63.0) and trained our network on these 16 combinations. Then we plot the training and validation loss values for all these 16 combinations against the standard deviation values (Figure 6.7). We observe that:

- For a particular standard deviation, the performance of the network increases both on training and validation data with an increase in feature size.

- For a particular feature size (except for very low feature size), the training loss decreases and validation loss increases with an increase in standard deviation.
- For very low feature size (feature size- 64), both training and validation loss increases with an increase in standard deviation.

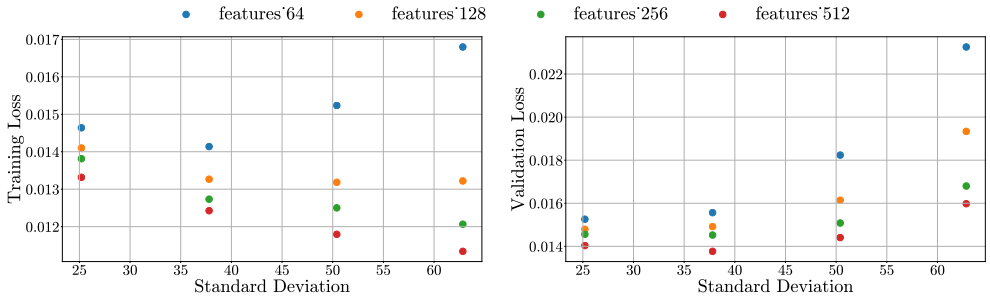
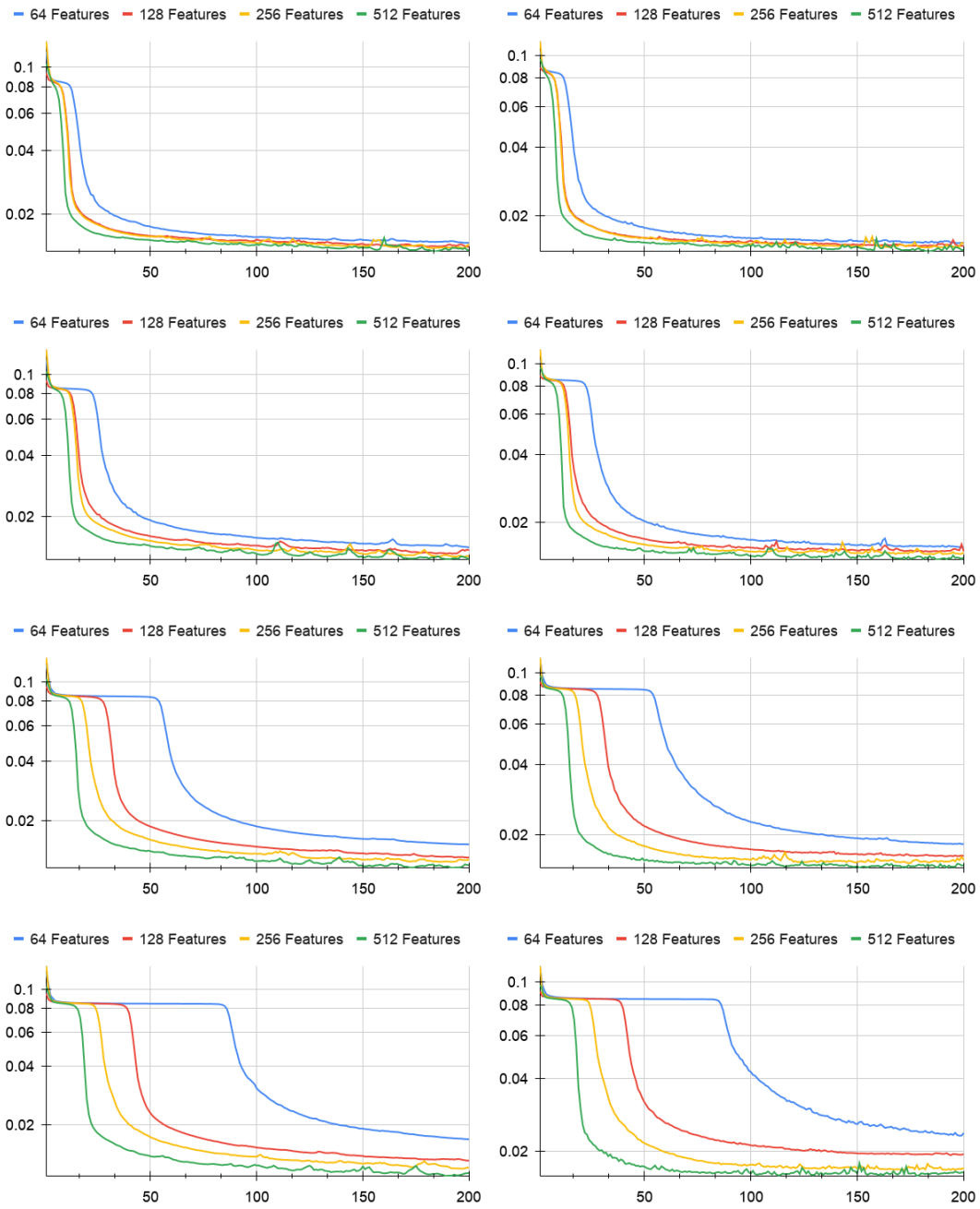


Figure 6.7.: Scatter plot of training and validation loss vs standard deviation for different feature sizes. We trained the network for different feature sizes: 64, 128, 256, and 512, and four different standard deviations: 25.0, 38.0, 50.0, 63.0. For a particular standard deviation, the performance of the network increases both on training and validation data with an increase in feature size. For a particular feature size, the network performance increases on training data but decreases on validation data with an increase in standard deviation. For very low feature size (feature size 64), the network performance decreases both on training and validation data.

We will discuss each of these observations one by one. Since we are sampling from a Gaussian distribution, both low and high features sampled from the distribution increase when we increase the feature size. This means the amount of information available to the network increases. With an increase in the amount of information, we observe an increase in the performance of the network.

We have already discussed the effect of standard deviation in Section 6.2 (second bullet point). But in the case of the lowest feature size, we observe the decrease in performance both on training and validation data with an increase in standard deviation (third bullet point). Increasing the standard deviation means that we are increasing the range of values from which the features can be sampled. Now since the feature size is already towards the lower end, it could be possible that the number of low features sampled goes below a threshold point which the network requires to learn the information over the input domain. This could be the reason why at the lowest feature size, we see an increase in both training and validation loss.

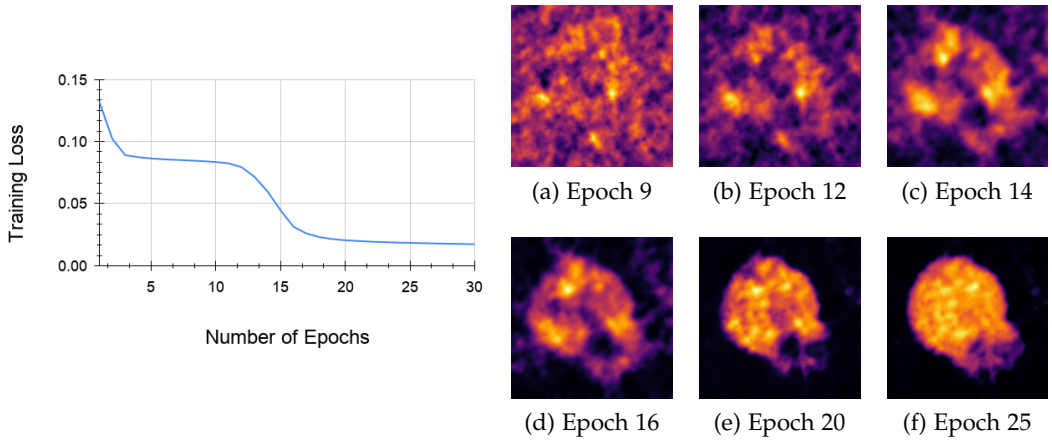
Figure 6.8 shows the training and validation loss curves over the optimization process for standard deviations 25.0, 38.0, 50.0 and 63.0 (top to bottom) for all the four feature sizes. We observed that when the feature size is too little (blue curve), the model takes more time to converge. The difference becomes more as we go towards higher standard deviations.



Training Loss vs Number of Epochs Validation Loss vs Number of Epochs

Figure 6.8.: Training and validation loss curves for network trained on standard deviation of 25.0, 38.0, 50.0 and 63.0 (top to bottom) for four different feature sizes: 64, 128, 256 and 512. When the feature size is too less (blue curve), the model takes more time to converge. The difference becomes more as we go towards higher standard deviations.

In all the loss curves, we observe that after the initial few epochs, there is always a sudden drop in the loss value (from approximately 0.08 to 0.02). Before and after this steep drop in the curve, loss value decreases gradually over the training process. To understand this, we will look at the change in loss values and change in the intensity values predicted by the network only for the first 30 epochs for the combination of standard deviation 38.0 and feature size 256. Figure 6.9 explains this sudden drop in loss values. The network first learns to differentiate between the foreground and background pixels. After initial few epochs, the network optimizes the intensity values of background pixels first as can be seen in the network predictions from epoch 12 to 20 (Figure 6.9(b) to (e)). This is where we see a sudden drop in loss values. After that, the network optimizes the intensity values of foreground pixels gradually.



(a) Epoch 9 (b) Epoch 12 (c) Epoch 14

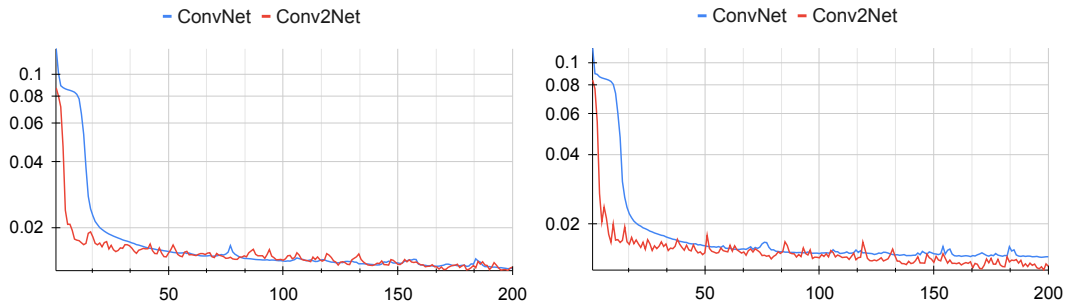
(d) Epoch 16 (e) Epoch 20 (f) Epoch 25

Figure 6.9.: On the left side is the train loss curve for first 30 epochs. On the right size are the intensity values of red color channel as they change over the epochs. The network first differentiates between the foreground and background pixels. From epoch 12 to 20, the network mainly optimizes the intensity values of the background. This is where we see the sudden drop in the loss values. After that, the network optimizes the intensity values of the foreground pixels which happens gradually.

6.4. Comparison of Network Architectures

In this chapter, we will compare two network architectures: ConvNet and Conv2Net (introduced in Section 5.1). The idea behind Conv2Net architecture is to divide the network into two parts where the output from the first part learns the alpha values. The second part of the network can then focus on learning only the color values. The ground truth image is of dimension HxWx4 where the first three are the color channels and the fourth is the alpha channel.

The input representation used is Fourier Encoded data with standard deviation (SD) of 32.0 and a feature size of 256. For input data corresponding to each image, Conv2Net predicts



(a) Training Loss vs Number of Epochs

(b) Validation Loss vs Number of Epochs

Figure 6.10.: Training and Validation Loss Curves (on RGB output) for ConvNet and Conv2Net architecture. Conv2Net architecture reduces the loss on RGB values earlier than ConvNet.

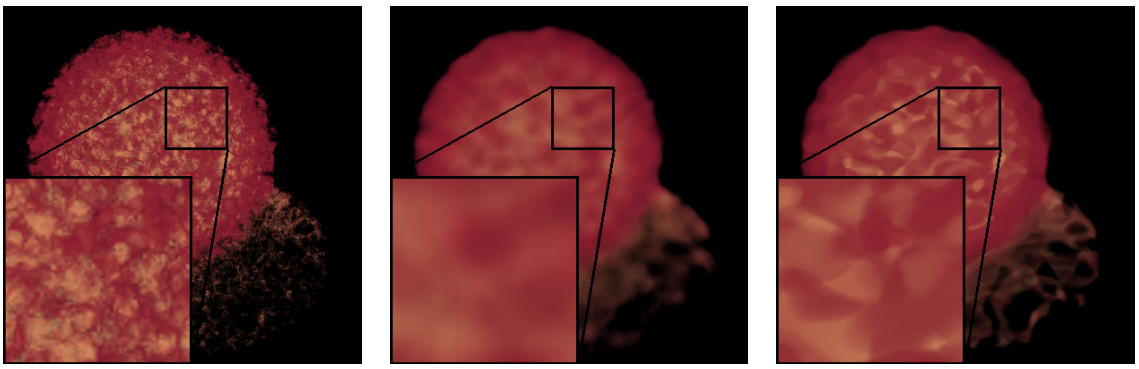
alpha and RGB output. First, we calculate L_1 loss between predicted and ground truth alpha values. The alpha values are then used as weights for the raw RGB output. Then we calculate L_1 loss between the weighted RGB (RGB * alpha) and ground truth RGB values.

Figure 6.10 shows the loss curves over the optimization process for both the architectures. Conv2Net architecture reduces the loss on RGB values earlier than ConvNet. Since the first part of the network in Conv2Net is trained to predict the alpha values closer to zero for background pixels, the weighted RGB values will already be closer to zero for those pixels. So the second part of the network in Conv2Net can focus on reducing the loss over the color of foreground pixels. This could be the reason why Conv2Net can reduce the loss on RGB values faster than ConvNet. Table 6.5 compares the value of evaluation metrics on validation data for both architectures. Conv2Net performs better when evaluated on the PSNR metric which compares two images on the pixel level. But on SSIM metric which considers the spatial relationship between neighboring pixels while comparing two images, ConvNet performs better.

	PSNR	SSIM
ConvNet	28.3309	33.2916
Conv2Net	29.1211	32.1221

Table 6.5.: Evaluation metrics on validation data for both the architectures. On per pixel comparison (PSNR), predictions by Conv2Net are better but when neighboring pixels are taken into account (SSIM), ConvNet performs better.

Figure 6.11 shows the qualitative comparison between two architectures. From the above comparisons, we observe that Conv2Net gives better predictions visually (Figure 6.11 (b) vs (c)) and on per pixel comparison (column 2 of Table 6.5). But when compared on SSIM metric (column 3 of Table 6.5), ConvNet gives better results. Even though Conv2Net gives better



(a) Ground Truth (b) ConvNet (c) Conv2Net

Figure 6.11.: Ground Truth and Network predictions on Ejecta Validation Data by ConvNet and Conv2Net. The visual quality of the predictions by Conv2Net are better than the predictions by ConvNet. The predictions by Conv2Net are less blurry and the areas on the surface with yellow color are also brighter.

predictions visually (Figure 6.11) than ConvNet, the predicted images are still very blurry and far from the quality of predictions made by current state of the art [1].

6.5. Comparison of Loss Functions

In Section 5.2, we introduced the SSIM loss function and discussed how this loss function considers the inter-dependence between pixels spatially close to each other. In this section, we will compare the network predictions when we trained the network on SSIM loss compared to L_1 loss. In Section 6.1, we showed how our approach works on Ejecta isosurface dataset when trained on L_1 loss. Ambient Occlusion (AO) on each surface point is a measure of how much surrounded that point is by neighboring points. This means that it is important for the network to learn the inter-dependence between the surrounding points to be able to better predict the AO values. With this in mind, we trained the network with the Ejecta isosurface dataset on SSIM loss as well. We trained the network on three different loss metrics: L_1 loss, SSIM loss and L_1+SSIM loss on Fourier Encoded data (SD-32.0, Feature Size-256). For the combination of L_1 and SSIM loss, we gave equal weight to both the loss metrics i.e. the loss value between a predicted image and ground truth image is the summation of L_1 loss and SSIM loss between them. Figure 6.12 shows the network predictions on validation data for all the three cases.

We observed that when trained with SSIM loss, the network predictions are better than when trained with L_1 loss as can be seen in columns 2 and 3 of Figure 6.12. But with SSIM loss, the network overshoots the predictions for the brighter and darker extremes in the image (this effect is better visible in rows 3 and 4 of Figure 6.12). To overcome this effect, we trained the network on equally weighted combination of L_1 and SSIM loss (column 4 of Figure 6.12).

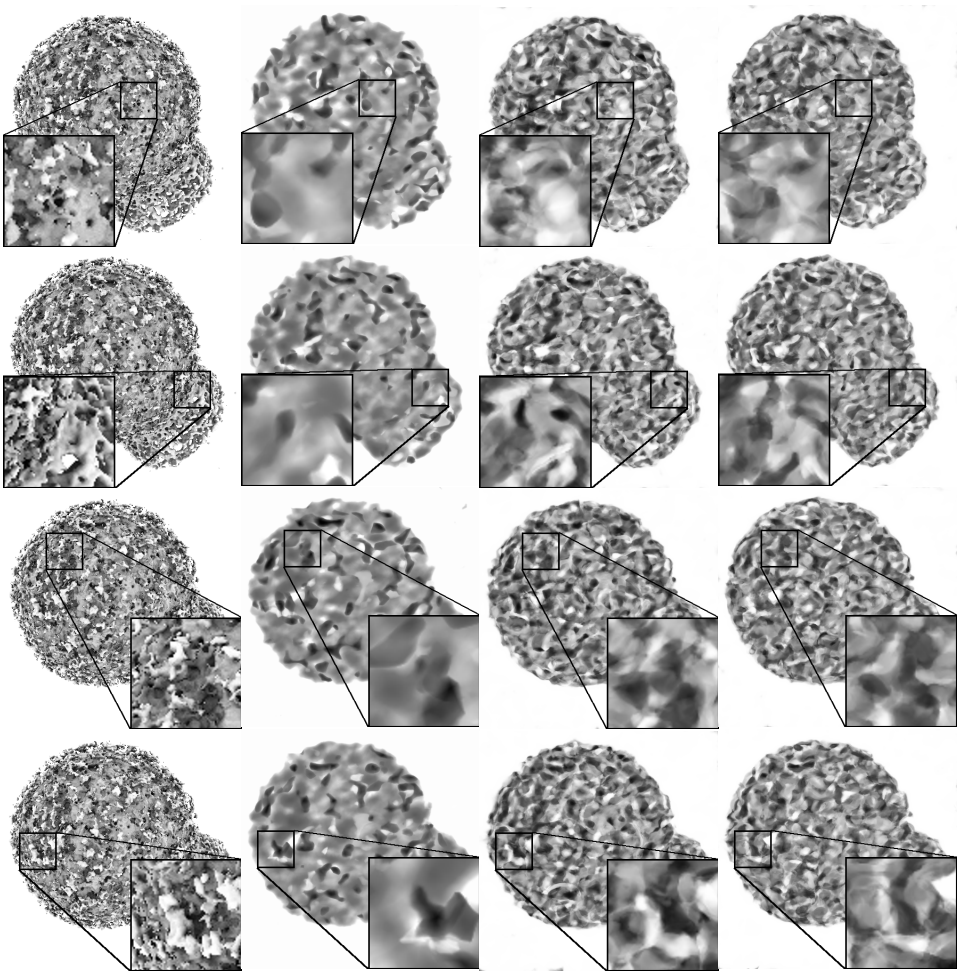
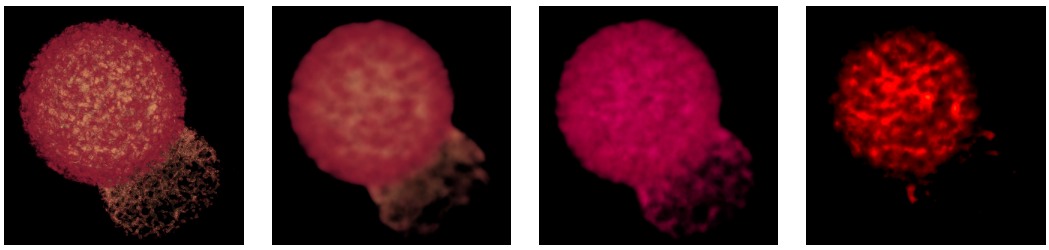


Figure 6.12.: Network Predictions on validation data of Ejecta isosurface dataset trained on different loss functions. It can be observed that with SSIM loss, the network is able to learn more information. But the brighter and darker extremes are predicted more brighter and darker by SSIM which creates a sort of highlighting effect. To overcome this, we used a combination of L_1 and SSIM loss in which case the network is able to learn more information but reduces the highlighting effect produced by SSIM.

We observed that the highlighting effect observed in the case of SSIM loss is reduced when trained with a combination of L_1 and SSIM.

Motivated by these results, we performed the same experiment on renderings of the semi-transparent Ejecta dataset. Figure 6.13 shows the network predictions on validation data of semi-transparent Ejecta dataset when trained on three different loss metrics: L_1 loss, SSIM loss and L_1+SSIM loss. For the combination of L_1 and SSIM loss, we gave equal weight to both loss values. The network gives better predictions when trained on L_1 loss among all three variants. Though SSIM loss gives better predictions than L_1 for the Ejecta isosurface (Figure 6.12), in case of direct volume renderings of the semi-transparent Ejecta, SSIM loss is unable to perform better than L_1 (Figure 6.13). In future work, it would be interesting to analyse this behaviour and understand the reason behind the difference in performance.



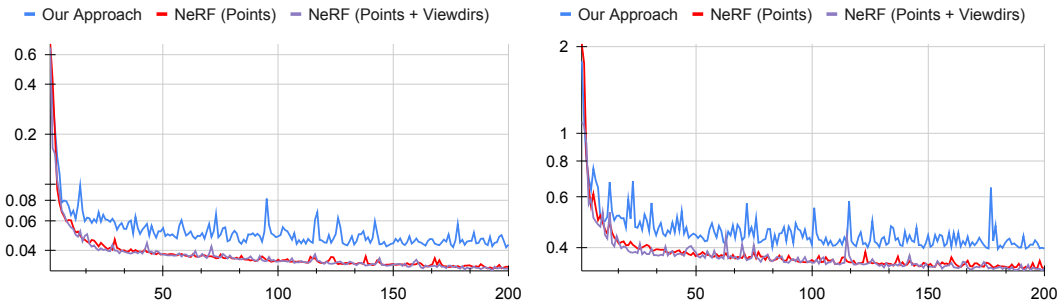
(a) Ground Truth (b) L_1 loss (c) SSIM loss (d) L_1 + SSIM loss

Figure 6.13.: Network Predictions on validation data of semi-transparent Ejecta data trained on different loss functions. We trained the network on three different loss metrics: L_1 loss, SSIM loss and L_1+SSIM loss. Among all three, the network gives better predictions when trained on L_1 loss.

6.6. Importance of ray sampling

The approach used by Mildenhall et al. [1] has shown impressive results for the task of view synthesis. An important part of their approach is to render the neural radiance field using principles from classical volume rendering [23]. Mildenhall et al. [1] sampled multiple 3D points on a ray (corresponding to one pixel) and predict the volume density and an RGB color on each point. The predicted volume densities and RGB colors are composited based on classical volume rendering to get a final RGB color for each ray.

To understand how important is the step of sampling multiple points on a ray, we perform a comparison experiment between our approach and NeRF approach: (1) Our approach: Input to the network are 3D points (where each ray corresponding to image pixel intersects with the bounding sphere) + ray direction, (2) NeRF (points): 128 points are sampled on each ray corresponding to each image pixel. These points are fed as input to the network and (3) NeRF (points + viewdir): ray direction along with sampled 128 points are fed as input to the network. We trained the network with a learning rate of 1e-2 and L_2 loss function.



(a) Training Loss vs Number of Epochs

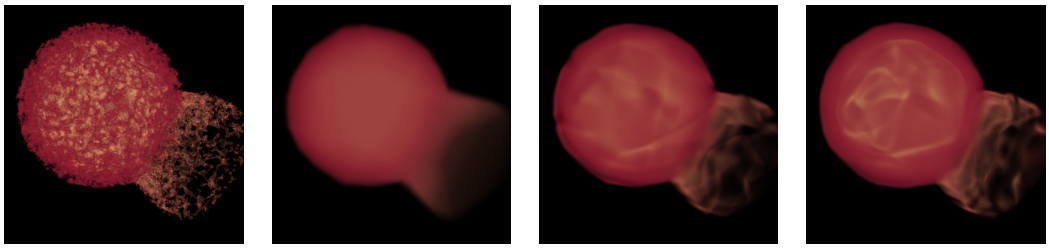
(b) Validation Loss vs Number of Epochs

Figure 6.14.: Comparison of Training and Validation Loss Curves over the optimization process between our approach and NeRF approach.

	PSNR
Our Approach	25.0099
NeRF(points)	26.2699
NeRF(points + viewdir)	26.5415

Table 6.6.: Comparison of PSNR metrics on validation data between our approach and NeRF approach. NeRF approach gives better quantitative results than our approach.

Figure 6.14 shows the training and validation loss curves over the optimization process for all three variants. We observed that NeRF approach can achieve better convergence than our approach. Table 6.6 shows the quantitative comparison of the three variants. NeRF approach gives better results when evaluated on the PSNR evaluation metric. The third variant where the sampled 3D points along with the view direction are fed to the network performs best among all three variants.



(a) Ground Truth

(b) Our Approach

(c) NeRF (points)

(d) NeRF (points + viewdir)

Figure 6.15.: Comparison of network predictions on Ejecta validation data between our approach and NeRF approach. NeRF (points + viewdir) gives best predictions among all three variants.

Figure 6.15 shows the visual comparison of the network predictions on validation data between our approach and NeRF approach. Similar to quantitative results (Table 6.6), NeRF approach gives better predictions visually as well, compared to our approach. The importance of sampling multiple points along the ray can be understood from the comparison of the first (Our Approach) and second variant (NeRF (points)). Even though the view directions are not provided to the network in the second variant, it still performs better than our approach where we take both the intersection point and view direction. From the visual comparison between the first and second variant (Figure 6.15(b) vs (c)), we can observe that in the case of (NeRF (points)), the network starts predicting the yellow color which is missing in case of our approach. The performance of the network becomes better when view directions are also used in addition to sampled 3D points (NeRF (points + viewdir)).

For this experiment, we have sampled 128 3D points on each ray. As a continuation of this experiment in future work, it would be interesting to train the network with different number of sampled points and visualize the effect of change in the density of sampled points on the quality of network predictions.

7. Conclusion and Future Work

We introduced an approach of representing a 3D object on a bounding sphere around the object. Our approach is inspired by the approach of Mildenhall et al. [1] in which the network is trained to learn the scene information inside the whole volume. To enable this, Mildenhall et al. [1] sample multiple points on each ray and train the network to learn the scene content on each point. This ray sampling step is expensive in terms of input generation and the amount of information the network has to learn. With the motivation of finding an inexpensive way to avoid learning the scene content in the whole volume, we came up with the idea of bounding sphere representation. The intuition behind our approach is how the projected images change over the sphere gives an understanding of how the structure and appearance of object changes in the 3D world.

We observed that with our input representation (Section 6.1), the network is able to generalize on the shape of the objects in case of simple shapes like Couch and Ejecta but the network is not able to learn the high-frequency information in the input data. For complex shapes like Chair and Lego machine, even the shape of the predictions given by the network is fuzzy.

Inspired by the success of Fourier Encodings (Section 5.3) in enabling the neural networks to learn fine details in input data [1, 2], we also experimented with using Fourier Encodings with our input representation. Though with Fourier Encoded data, the capability of the network to learn scene content increases, it is still not comparable to the current state of the art [1]. From our analysis on the effect of standard deviation and feature size of Fourier Encodings on the performance of the network (Section 6.2, 6.3), we have observed that for a particular task and data, there is an optimum standard deviation point where the network shows better generalization capability than other standard deviations (Figure 6.3). In other words, the validation loss first decreases and then increases with an increase in standard deviation. With an increase in feature size, we have observed an increase in the performance of the network. Until now, the optimal standard deviation and feature size for a particular task can only be found by a brute force hyper-parameter search.

In our analysis of Fourier Encodings, we have observed a correlation between low frequencies and the generalization of the network. With increase in higher frequencies (i.e with an increase in standard deviation), we observed an increase in the capability of the network to learn fine details on training data (Figure 6.5). On validation data (Figure 6.6) the capability of the network to predict fine details first increases. But after a certain point, the network starts giving noisy predictions.

An interesting future work would be to explore more into the relation of the frequencies sampled and the learning capability of the network. For example, how the network will perform if trained only on higher frequencies or if trained on higher frequencies from the pool

of multiple standard deviations. To study this, an analysis of the gradients of the network could be done. This study will provide more insights into how important the low and high frequencies are.

The Fourier Encodings transform the scene information in the spatial domain to the frequency domain. So, the network trains on frequency domain data. From recent works [1, 2], we have seen that this has shown impressive results. An interesting future work would be to see how the network performance will change if the loss calculation is also done in the frequency domain. Instead of calculating loss directly on RGB values, we can take the fourier transform of the predicted and ground truth images and then calculate loss between them.

List of Figures

List of Tables

Acronyms

AO Ambient Occlusion. 20, 35, 42

FC Fully Connected. 13

MAE Mean Absolute Error. 16

MLP Multi-layer Perceptron. iv, v, 1, 3–5, 13

MSE Mean Squared Error. 16

NeRF Neural Radiance Field. iv, 1, 4, 19, 37–39, 43, 44

NTK Neural Tangent Kernel. 5, 6

PSNR Peak Signal-to-Noise Ratio. 34, 38, 44

ReLU Rectified Linear Unit. 13, 15, 16

SD standard deviation. 18, 22–30, 33, 35, 42

SSIM Structural Similarity Index Measure. 16, 17, 34–37

Bibliography

[1] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. *NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis*. 2020. arXiv: 2003.08934 [cs.CV].

[2] M. Tancik, P. P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. T. Barron, and R. Ng. *Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains*. 2020. arXiv: 2006.10739 [cs.CV].

[3] A. Rahimi and B. Recht. "Random Features for Large-Scale Kernel Machines". In: *Proceedings of the 20th International Conference on Neural Information Processing Systems*. NIPS'07. Vancouver, British Columbia, Canada: Curran Associates Inc., 2007, pp. 1177–1184. ISBN: 9781605603520.

[4] V. Sitzmann, M. Zollhöfer, and G. Wetzstein. "Scene representation networks: Continuous 3d-structure-aware neural scene representations". In: *Advances in Neural Information Processing Systems*. 2019, pp. 1121–1132.

[5] S. Laine and T. Karras. "Efficient Sparse Voxel Octrees". In: *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. I3D '10. Washington, D.C.: Association for Computing Machinery, 2010, pp. 55–63. ISBN: 9781605589398. DOI: 10.1145/1730804.1730814. URL: https://doi.org/10.1145/1730804.1730814.

[6] L. Liu, J. Gu, K. Z. Lin, T.-S. Chua, and C. Theobalt. *Neural Sparse Voxel Fields*. 2020. arXiv: 2007.11571 [cs.CV].

[7] Z.-Q. J. Xu, Y. Zhang, and Y. Xiao. *Training behavior of deep neural network in frequency domain*. 2018. arXiv: 1807.01251 [cs.LG].

[8] D. Arpit, S. Jastrzębski, N. Ballas, D. Krueger, E. Bengio, M. S. Kanwal, T. Maharaj, A. Fischer, A. Courville, Y. Bengio, and S. Lacoste-Julien. *A Closer Look at Memorization in Deep Networks*. 2017. arXiv: 1706.05394 [stat.ML].

[9] N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. A. Hamprecht, Y. Bengio, and A. Courville. *On the Spectral Bias of Neural Networks*. 2018. arXiv: 1806.08734 [stat.ML].

[10] Z. J. Xu. *Understanding training and generalization in deep learning by Fourier analysis*. 2018. arXiv: 1808.04295 [cs.LG].

[11] A. Jacot, F. Gabriel, and C. Hongler. "Neural Tangent Kernel: Convergence and Generalization in Neural Networks". In: *CoRR* abs/1806.07572 (2018). arXiv: 1806.07572. URL: http://arxiv.org/abs/1806.07572.

[12] T. Hofmann, B. Schölkopf, and A. J. Smola. "Kernel methods in machine learning". In: *Ann. Statist.* 36.3 (June 2008), pp. 1171–1220. DOI: 10.1214/009053607000000677. URL: https://doi.org/10.1214/009053607000000677.

[13] R. Basri, M. Galun, A. Geifman, D. Jacobs, Y. Kasten, and S. Kritchman. *Frequency Bias in Neural Networks for Input of Non-Uniform Density*. 2020. arXiv: 2003.04560 [cs.LG].

[14] R. Basri, D. Jacobs, Y. Kasten, and S. Kritchman. *The Convergence Rate of Neural Networks for Learned Functions of Different Frequencies*. 2019. arXiv: 1906.00425 [cs.LG].

[15] R. Heckel and M. Soltanolkotabi. *Compressive sensing with un-trained neural networks: Gradient descent finds the smoothest approximation*. 2020. arXiv: 2005.03991 [cs.LG].

[16] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. *ShapeNet: An Information-Rich 3D Model Repository*. 2015. arXiv: 1512.03012 [cs.GR].

[17] S. Weiss. *Basic Volume Renderer*. https://gitlab.com/shaman42/basic-volume-renderer. 2020.

[18] C. B. Choy, D. Xu, J. Gwak, K. Chen, and S. Savarese. "3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2016.

[19] P. Goel. *generate3D*. https://github.com/ParikaGoel/generate3D. 2020.

[20] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. "Image quality assessment: from error visibility to structural similarity". In: *IEEE transactions on image processing* 13.4 (2004), pp. 600–612.

[21] H. Zhao, O. Gallo, I. Frosio, and J. Kautz. *Loss Functions for Neural Networks for Image Processing*. 2018. arXiv: 1511.08861 [cs.CV].

[22] G. Fang. *pytorch-msssim*. https://github.com/VainF/pytorch-msssim. 2020.

[23] J. T. Kajiya and B. P. Von Herzen. "Ray tracing volume densities". In: *ACM SIGGRAPH computer graphics* 18.3 (1984), pp. 165–174.

A. General Addenda

A.1. Technology/Framework/Tools

- **Volume Renderer (Rendering Images of Ejecta Dataset):**
  - Programming Language: C++, CUDA
  - Libraries Used: lodepng, cuMat
- **Input Generation and Network Training:**
  - Programming Language: Python 3.x
  - Framework: Pytorch (1.5.x)
  - Libraries Used: Python Imaging Library (PIL)
- **Evaluation:**
  - Programming Language: Python 3.x
  - Framework: Pytorch (1.5.x)
  - Libraries Used: Python Imaging Library (PIL), Matplotlib, Pandas

A.2. Source Code Repositories

- **Volume Renderer:**
  https://gitlab.lrz.de/ge68tuq/volume-renderer
- **Network Training and Evaluation:**
  https://gitlab.lrz.de/ge68tuq/dvr