Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Material picker: Material recognition in images using deep learning

Bachelor Thesis

2020
Filip Jurčák

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Material picker: Material recognition in images using deep learning

Bachelor Thesis

Study Programme:	Computer Science
Field of Study:	Computer Science
Department:	FMFI.KI - Department of Computer Science
Supervisor:	prof. RNDr. Roman Ďurikovič, PhD.
Consultant:	Mgr. Petr Vévoda

Bratislava, 2020
Filip Jurčák

29514543

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	Filip Jurčák
Študijný program:	informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor:	informatika
Typ záverečnej práce:	bakalárska
Jazyk záverečnej práce:	anglický
Sekundárny jazyk:	slovenský

Názov: Material picker: Material recognition in images using deep learning
Material picker: Rozpoznávanie materiálov v obrazoch pomocou hlbokého učenia

Anotácia: Jedným z dôležitých krokov pri modelovaní realistických 3D scén je nastavenie vzhľadu povrchu rôznych objektov v scéne. Cieľom tohto projektu je zjednodušiť túto často zdĺhavú úlohu tým, že poskytneme 3D umelcovi inteligentný nástroj na výber materiálu. Tento nástroj umožní „vybrať" materiál z ľubovoľne vybraného obrázka jednoduchým kliknutím na daný objekt. Na dosiahnutie tohto cieľa sa použije hlboká neurónová sieť. K dispozícii bude rozsiahly súbor trénovacích dát, kde bude k dispozícii komplexná korešpondencia medzi obrazovými pixelmi a podkladovým objektovým materiálom. Neurónová sieť bude natrénovaná na obnovenie tejto korešpondencie s pixelmi z nových, predtým nevidených snímok.

Vedúci:	prof. RNDr. Roman Ďurikovič, PhD.
Konzultant:	Mgr. Petr Vévoda
Katedra:	FMFI.KI - Katedra informatiky
Vedúci katedry:	prof. RNDr. Martin Škoviera, PhD.
Dátum zadania:	29.10.2019

Dátum schválenia: 30.10.2019 doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

....
študent vedúci práce

29514543

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Filip Jurčák
Study programme:	Computer Science (Single degree study, bachelor I. deg., full time form)
Field of Study:	Computer Science
Type of Thesis:	Bachelor´s thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: Material picker: Material recognition in images using deep learning

Annotation: One of the important steps in modeling realistic 3D scenes is setting material appearance of the various scene objects. The goal of this project is to simplify this -- often tedious -- task by providing the 3D artist with an intelligent material picker tool. The tool will allow to 'pick' a material from any given input image by simply pointing to an object. A deep neural network will be trained to achieve this nontrivial goal. An extensive set of training data will be provided, where the complex correspondence between the image pixels and the underlying object material will be available. The network will be trained to recover this pixel-material correspondence from new, previously unseen images.

Supervisor:	prof. RNDr. Roman Ďurikovič, PhD.
Consultant:	Mgr. Petr Vévoda
Department:	FMFI.KI - Department of Computer Science
Head of department:	prof. RNDr. Martin Škoviera, PhD.
Assigned:	29.10.2019
Approved:	30.10.2019

doc. RNDr. Daniel Olejár, PhD.
Guarantor of Study Programme

Student

Supervisor

Acknowledgments: I want to say sincere thank you to all people involved in writing this thesis, specifically to: Jaroslav Křivánek (doc. Ing., Ph.D.) who came up with the idea of this thesis but sadly passed away in early stages of writing the thesis; prof. RNDr. Roman Ďurikovič, PhD., for taking over the supervision and providing important insight into computer graphics; Petr Vévoda for his advice and day-and-night support with all kinds of questions; Matej Sládek, my friend in need; my family for giving me space and support throughout the whole process of writing; and lastly, but probably most importantly to my girlfriend Eďa for helping me to stay afloat and sane, as I would certainly not be able to finish this thesis without her support.

Abstrakt

Proces nastavenia vlastností materiálu k nadobudnutiu realistického vzhľadu je zvyčajne únavný a často vyžaduje zručnosť na dolaďovanie parametrov, keďže rôzne kombinácie týchto parametrov môžu vyprodukovať rôzne materiály. Aby sme tento proces zjednodušili predstavujeme návrh riešenia pozostávajúceho z hlbokých neurónových sietí na segmentovanie materiálu a odhad vnútorných vlastností scény, ako difúzne a lesklé albedo, povrchové normály, lesk, pohľadový vektor a osvetlenie z jedného obrázka. Naša metóda teda poskytuje riešenie dvoch z najzákladnejších problémov počítačového videnia a počítačovej grafiky - inverzného renderovania a segmentácie materiálu. Použité siete sme trénovali na datasete vygenerovanom pomocou fyzikálne korektných techník pre zabezpečenie dobrej generalizácie na reálnych obrázkoch.

Kľúčové slová: Hlboké učenie, inverzné renderovanie, materiálová segmentácia, strojové učenie, počítačová grafika, rozpoznávanie materiálov

Abstract

The process of setting material properties for realistic appearance is usually tiresome and often demands skill for fine-tuning the parameters, as different combinations of these parameters can produce different materials. To simplify this process, we introduce a pipeline consisting of deep neural networks to segment material and predict intrinsic scene characteristics, like diffuse and specular albedo, surface normals, glossiness, view vector, and illumination from a single image. Our pipeline thus provides solution to two of the most fundamental problems in computer vision and computer graphics - inverse rendering and material segmentation. We trained the networks on the dataset generated using physically-based techniques to ensure good generalization on real images.

Keywords: Deep learning, inverse rendering, material segmentation, machine learning, computer graphics, material recognition

Contents

List of Figures

Introduction

Setting material appearance is one of the most crucial steps in modeling 3D scenes and probably the most important one for creating a realistic model look. This task is often long, tedious, and requires non-trivial skill, as a lot of parameters need and can be set up for a model to look realistic after rendering. The number of parameters varies between used material models, but realistic models often need tens of correctly set up parameters.

As a result of this thesis, we want to ease the whole process by providing artists with a material picker tool. This tool is a deep neural network that would estimate a lot of intrinsic properties of an image, which would help us recover material from a user-specified object in an image.

We do so by inventing a pipeline for solving two fundamental problems in computer graphics and computer vision - inverse rendering and material segmentation, both from a single image. In addition to segmenting the input image, our method performs per-pixel estimation of the number of intrinsic scene characteristics, such as diffuse and specular albedo, surface normals, glossiness, and view vector. We use all of these inferred properties to simulate the rendering process, yielding close approximation of an input image.

To train all models in our pipeline, we present a new way to create a modern dataset by using advanced features of a physically-based V-Ray renderer to bridge the gap between synthetic data and real images. This is crucial, as we most often want to generalize well on real images, which is hard to achieve with synthetic images only.

Chapter 1

Problem statement and background

In this chapter, we shall define what kind of problem we want to solve, so we better understand why it is interesting to work on such a task, and also specify the terminology and methods that we will use throughout this thesis.

1.1 Our goal

Defining properties of materials in the scene is essential for matching appearance of real world materials, but it is time consuming. This procedure can be dramatically simplified, as artists and graphic designers create new looks from already existing artworks. If they were able to transfer material characteristics from an image of these designs, it would be a time saver. Our goal is therefore to offer them a tool that would determine the most used material attributes from a single image.
To accomplish this goal, we need to perform per-pixel material segmentation, and then for this segmented material approximate selected scene or material characteristics. This approximation can be achieved by doing inverse rendering of a scene.
As we will show in chapter 2, there has been a lot of research lately regarding inverse rendering. Using deep learning techniques to tackle problems from different areas of interest proved to be very successful, so it was naturally applied to the computer vision field, and in our case, to inverse rendering as well. We believe that combination of these approaches will enable us to solve our task successfully.
As this work requires knowledge of terminology, methods and concepts from both computer graphics and machine learning, we elucidate both of these areas in the next sections.

1.2 Terminology and methods in rendering

1.2.1 Rendering

Rendering is a major subfield of the computer graphics area. Rendering is a sequence of steps to produce a 2D image from 3D representation of a scene stored in a computer. During this sequence – which is also called the rendering pipeline – the algorithm for handling rendering of a scene needs to take model representations, apply transformations, illuminate the scene from all lights presented, map textures to objects, throw away parts of the scene which will not be rendered, and finally draw an image from the camera view. There are several types of renderings based on different rendering algorithms, mostly divided into two categories: non-photorealistic rendering and photorealistic rendering, sometimes also called physically based rendering (PBR). The latter implements the concepts of transport and scattering of light in the real world, which is far more computationally expensive than the former approach but produces more plausible results. To look real when rendered, PBR needs (among other parameters) to have correctly set up material model, usually referred to as BRDF.

1.2.2 Bidirectional Reflectance Distribution Function

Bidirectional Reflectance Distribution Function (or BRDF for short) is a probabilistic function $f_r(\omega_i, \omega_o)$ ($f_r(\omega_i \to \omega_o)$) describing how light is reflected based on surface attributes. More specifically, given incoming light direction ω_i and outgoing direction ω_o, it gives the probability that a photon arriving from direction ω_i will be reflected to direction ω_o.

There are several categories of BRDF models, of which the most impactful ones are the physically based BRDFs. To consider a BRDF model to be physically based, it must meet the following properties:

- positivity: $f_r(\omega_i, \omega_o) \geq 0$
- obeying Helmholtz reciprocity: $f_r(\omega_i, \omega_o) = f_r(\omega_o, \omega_i)$
- conserving energy: $\int_\Omega f_r(\omega_i, \omega_o) \cos\theta_i \, d\omega_i \leq 1 \;\; \forall \omega_o$

where $\cos\theta_i$ represent decrease of radiance with increasing θ_i (angle between ω_i and surface normal).

To achieve realistic material look, it is important to use such BRDF that satisfies these properties. Example of such BRDF can be physically based Phong BRDF, which is equal to

$$f_r^{Phong} = \frac{\rho_d}{\pi} + \frac{\rho_s(n+2)\cos^n\theta_r}{2\pi} \tag{1.1}$$

where ρ_d stands for diffuse albedo, ρ_s for specular albedo, n for glossiness and θ_r for angle between view vector and reflected light vector.

1.2.3 Reflection equation

Knowing the BRDF of a surface allows us to compute how much light coming from direction ω_i is reflected from the surface to direction ω_o. For that one has to multiply radiance L_i from direction ω_i, BRDF $f_r(w_i \to w_o)$ and $\cos\theta_i$. Summarized in mathematical notation:

$$L_r(\omega_i \to \omega_o) = L_i(\omega_i) \cdot f_r(\omega_i \to \omega_o) \cdot \cos\theta_i \tag{1.2}$$

In order to compute the total radiance reflected to direction ω_o we need to sum up the contributions from all light sources, direct or indirect. This can be done by integrating these contributions over upper neuron hemisphere $H(x)$, which gives us the following equation

$$L_r(\omega_o) = \int_{H(x)} L_i(\omega_i) \cdot f_r(\omega_i \to \omega_o) \cdot \cos\theta_i \, d\omega_i \tag{1.3}$$

also called reflection equation. This integral generally does not have an analytical solution and has to be computed numerically.

1.2.4 Monte Carlo integration

A typically used numerical method for solving integrals in rendering is Monte Carlo integration. This technique uses random numbers to sample points at which the integrand is evaluated. Let's denote an integral that we want to approximate, as

$$I = \int g(x)dx \tag{1.4}$$

Monte Carlo estimation of I is defined as

$$\langle I \rangle = \frac{1}{N} \sum_{k=1}^{N} \frac{g(X_k)}{p(X_k)}, \tag{1.5}$$

where N is the number of samples taken, X_k, $k = 1, ..., N$ are the samples and $p(x)$ is a probability density function from which the samples were drawn.
By substituting equation 1.3 into equation 1.5, the result is

$$\langle L_r(\omega_o) \rangle = \frac{2\pi}{N} \sum_{k=1}^{N} L_i(\omega_{i,k}) f_r(\omega_{i,k} \to \omega_o) \cos\theta_{i,k} \tag{1.6}$$

where 2π stands for the probability density function ($p(X_k) = \frac{1}{2\pi}$) of uniform sampling directions on the hemisphere and $\omega_{i,k}$, $k = 1, ..., N$ are the sampled directions.
An image can be rendered by evaluating equation 1.6 for each of its pixels. Given such an image, our goal is to estimate what material (i.e. $f_r(\omega_{i,k} \to \omega_o)$) was used when the image was rendered.

1.2.5 Inverse rendering

One of the possible methods for estimating what materials are present in an image is inverse rendering. Inverse rendering is one of the principal and long-standing problems in computer vision and computer graphics. Its main goal is to, provided an image or several images of a scene, estimate intrinsic properties of a scene, like depth, albedo, normals, reflectance, lighting and so on. This problem is hard for several reasons, mainly, as stated in [15]: „This is an ill-posed task: these scene factors interact in complex ways to form images and multiple combinations of these factors may produce the same image.“ As we can see, there is an infinite number of solutions for parameters for a single image, which makes the problem hard or almost impossible to solve. However, some solutions are statistically more admissible than others. Citing [2], which says: „Our goal is therefore to recover the most likely explanation that explains an input image.“ To make this work, we need to come up with such statistics that would correctly approximate the real world. This is not straightforward, but recent advances in both optimization and learning based approaches show that it's possible to estimate a handful of properties correctly [2] and even better results when physically based datasets were used for training neural networks [23][15]. With these properties in hand, we want to estimate what is the material on the user-specified object in the image.

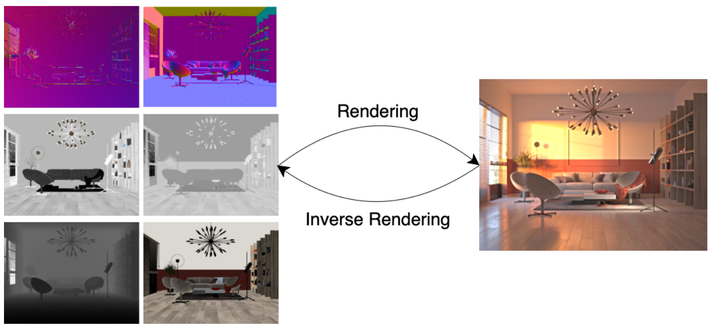


Figure 1.1: Rendering vs inverse rendering

1.3 Terminology and methods in machine learning

1.3.1 Machine learning

For quite some time, mathematicians wondered if it's possible to create thinking machines. The solution to this idea was to allow computers to learn complex concepts from simple ones or experience. We provide this experience in the form of a dataset, which consists of information (usually called features) about the task that we want to train the model for. Generally, we let the algorithm decide which features are important and how will individual features contribute to its final prediction. We call this ability of computers as machine learning. Machine learning is an outstandingly fast advancing area of research, and it helps to push research forward in other areas as well. Computer graphics is not an exception as machine learning techniques are used in calculating direct illumination [27] for example.

1.3.2 Neural networks

Human brains consist of nerve cells, which are called neurons. These neurons form large networks where they can propagate information from one neuron to the other. The purpose of neural networks (NNs) as a machine learning method is to mimic these networks to be able to learn and make decisions. The main difference between neural networks and traditional programming is that while in traditional programming we explicitly instruct a computer what to do in each step of the program, we don't instruct neural networks how to behave or how to solve the task. We simply allow it to examine the provided data and let it propose a solution. This solution can be viewed as mapping $\mathcal{F}'$, where $\mathcal{F}$ is the underlying mapping that we want to approximate. $\mathcal{F}'$ should be optimal in some sense - we need such $\mathcal{F}'$ that minimizes

$$\frac{\sum_{x \in X} error(\mathcal{F}(x) - \mathcal{F}'(x))}{|X|}$$

where X is a set of inputs to the neural network.
Neural network consists of several layers, which are called input, hidden and output layer, with input and output layers required in every neural network, but any non-negative number of hidden layers is allowed. Every layer consists of several neurons. In the most common type of neural network, all neurons from previous layer are connected with all neurons in the next layer. These connections are called **weights** and network learns them throughout the training. We can see weights between one layer to other in figure 1.2 and an example of a simple neural network in figure 1.3. Neural network performs two operations – **forward propagation and back-propagation**. The former is used to get the prediction, the latter to adjust weights in the system to account for the computed error. During forward propagation, neural network computes values for all neurons in the next layer based on the previous layer. These values are then fed through some non-linearity function f to keep all the values in certain range (for example between $0 \leq x \leq 1$ or $-1 \leq x \leq 1$). Computed values are called **activations** of neurons. This process repeats until the network compute values in output layer. Equation 1.7 summarizes the process of computing activation of one neuron, where n and m are the number of neurons in first layer and second layer respectively. Common thing to help neural network learn better is to add a **bias term** to the layer and set it

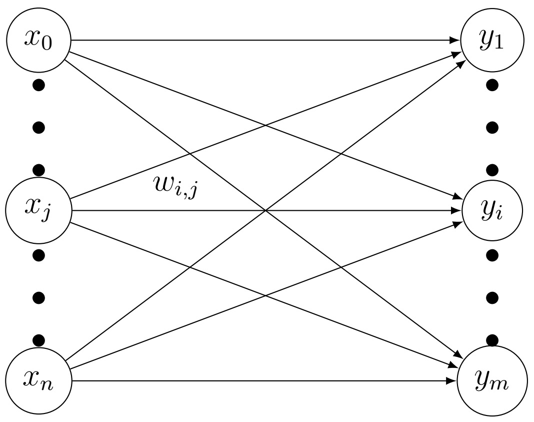


Figure 1.2: Connections between two layers of neural network, circles are neurons and lines represent weights, weight $w_{i,j}$ represents connection between neuron j in first layer and neuron i in second layer

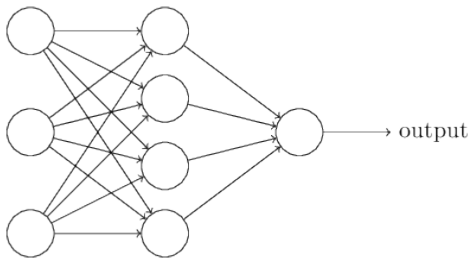


Figure 1.3: Example of a simple neural network with input, hidden and output layer. Circles represent neurons, lines between neurons show connections from neuron in one layer to neuron in the next layer. Taken from [18].

as $x_0 = 1$.

$$y_i = f(\sum_{j=0}^{n} w_{i,j} * x_j) \quad \forall i \in \{1, \dots, m\} \tag{1.7}$$

When we have computed the prediction, we need to adjust weights in a network to account for the difference between predicted value and the actual value. The error is then propagated back through the network in order to compute gradient, which is in turn used by some optimization method (for example gradient descent) to find local minimum of an error function, which is a metric for evaluating network's performance. The process of forward and back-propagation is repeated many times for every entry in the dataset until convergence.

1.3.3 Deep learning and deep neural networks

Deep learning is a special kind of machine learning which is capable to learn more complex functions than simpler methods of machine learning. Every neural network that has more than one hidden layer can be considered a deep neural network. These multiple layers help the network to develop several levels of abstraction, which can give deep networks an upper hand in recognizing complex patterns over other methods or models [7]. This is why so many solutions to pattern recognition problems employ this technique, but because of the relatively high computation power required for it's training, it wasn't used until very recently. Most models for inverse rendering use deep convolutional neural networks, which we will define in the next section.

1.3.4 Convolutional neural networks

Convolutional neural networks (CNNs, or sometimes just convolutional networks), are neural networks that are specially designed to process grid-like structured data, like images or videos.

Neural networks use matrix multiplication and activation function to compute the activation of neurons in the next layer. CNNs on the other hand, use a different approach - at least in one of their layers, they use a special kind of linear operation called convolution, which is defined as

$$s(t) = \int x(a) * w(t-a)\mathrm{d}a$$

with x is often referred to as the input and function w as the kernel. The output of the convolution is referred to as the feature map(s). Convolutional layers convolve the input with the help of the kernel function – which is just a function that transforms original input space into space, where it can be easier to train the model due to change from non-linear to a linear problem – and pass its result to the next layer. This is similar to the response of a neuron in our brain to a specific impulse. Because of this property, CNN is a great model for extracting edge information from images. Convolution layer in CNNs consists of convolution stage, detector stage and pooling stage. During convolution stage, several convolutions are run in parallel to produce many layers of linear activations. During detector stage, all of these layers are run through some non-linear function to produce activations in certain range. And finally, during pooling stage, we use pooling function to produce statistical analysis of a specific neighbourhood in every layer. Typical CNN architecture for digit recognition can be seen in figure 1.4.

Another important property of CNN is its effectiveness when compared to traditional neural networks - performing convolution in layers of CNN is faster and requires orders of magnitude less storage then using NN for the same kind of problem [7]. As a result, CNNs perform tremendously on image recognition tasks and are now one of the state-of-the-art solutions for this challenging problem.

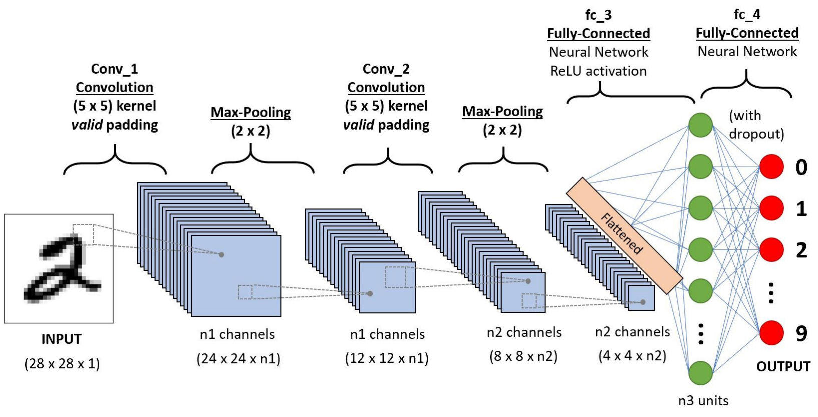


Figure 1.4: Typical CNN architecture for digit recognition, taken from [22]. The original image is run through several convolutional layers before finally being flattened with digit predictions as output

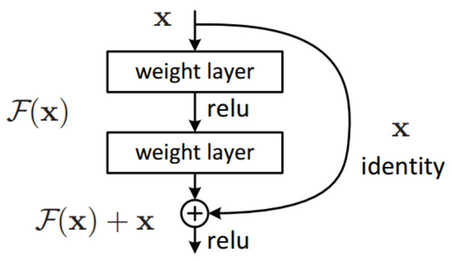

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 1.5: Example of residual block

1.3.5 Residual neural network

As we stated in section 1.3.3, deep networks have the ability to learn several layers of abstraction, which means that depth is important. This is especially valuable when working with images or videos, as these layers can help decompose input image into low to high level features of the image. However, just adding more and more layers brings problems like non-convergence of the whole network or accuracy degradation. The former was mostly resolved by normalized initialization, the latter by introduction of residual learning, with residual neural network as its architecture [12]. Residual neural network is network consisting of residual blocks, as shown in figure 1.5. Idea behind this block architecture is that rather than finding mapping $\mathcal{H}(x)$ that would be optimal for this block without any prior, we reformulate the mapping that the block should learn to $\mathcal{F}(x) = \mathcal{H}(x) - x$, so the output mapping then becomes $\mathcal{F}(x) + x = \mathcal{H}(x)$. This helps with training particularly where output of the block should be very similar to its input, like in cases where identity mapping is the optimal mapping. We use residual blocks in our networks a lot because they enable us to train deeper models, as they are easier to optimize than conventional CNN networks [12].

Chapter 2

Prior work

In this section, we would like to point out research that was done related to the two problems we are trying to solve: inverse rendering and material segmentation.

2.1 Inverse rendering from a single image

As inverse rendering of a scene is difficult, previous research in the field focused either on subproblems of this problem (like inverse rendering of an object instead of a whole scene), or the research focused on estimation of a small number of properties of a scene [15] [23] or a small number of materials [3].
To estimate intrinsic characteristics of an image authors in [15] [23] used neural networks. To obtain the data for training, they augmented SUNCG dataset [25] by mapping photorealistic materials to geometries in this dataset or completely re-render images by using physically based renderer, as the original dataset was rendered only with OpenGL using Phong BRDF model and does not look realistic.
In [23], authors proposed pipeline for estimating diffuse albedo, environment map and normals from a single image by using Inverse Rendering Network (IRN) and combination of two modules - direct renderer for computing direct illumination and Residual Appearance Renderer (RAR) for computing shading and reflections - to re-synthesize the input image from estimated components and to learn from real images where ground-truth data is not available. To train IRN to correctly predict environment map, they had to generate ground-truth data, as the environment map that the scene was rendered with was used as exterior lighting, which does not reflect illumination inside the scene. To address this issue, they also trained neural net (EnvMap net) to predict best average environment map for the whole scene (including illumination inside the scene), which they then set as their ground-truth for this parameter of IRN. The environment map predicted by IRN was then used in the direct renderer to approximate incoming illumination using numerical quadrature.

Different approach was presented in [15]. In this paper, authors were able to predict diffuse albedo, normals, specular roughness, depth, and spatially-varying lighting, which is a technique for estimating per-pixel illumination. Obtaining such data unwisely is computationally expensive and memory consuming, thus they resolved to use spherical Gaussian lobes, that preserve all lighting frequencies but require far less parameters to store. This very detailed pre-computed irradiance enabled them to include differentiable renderer in their pipeline to simulate image creation process without any rendering related code written by the authors. Due to this precise estimation of parameters, state-of-the-art object insertion and material editing were made possible.

2.2 Material Classification and Segmentation

Material segmentation is especially challenging, as real-world materials have a rich texture, and the final look of the material is a combination of many scene properties like lighting, depth, normals, and so on.
There exists a large number of classifiers for classifying images into classes (like dogs, cats, etc.): e.g. AlexNet [14], VGG [24] and GoogLeNet [26]. These classifiers take an input image and their output is per-class probability of the object in the image belonging to that class. Most used approach to image segmentation we found was the use of transfer learning on models pre-trained as classifiers [16] [3]. Transfer learning is method for re-using parts of already trained model (and possibly change the output layers), and retrain only those layers that were not taken from the pre-trained model.
In [16], authors removed final classification layer and used several upsampling layers to output 21 feature maps of the same size as input image. These 21 maps represented per pixel probability of the pixel belonging to 21 classes they had in the dataset. To get the final image they had to apply post-processing by taking per-pixel maximum over these feature maps, with index of the map that contained maximum assigned as the final value. It is worth noting that the new model was trained on the exact same dataset as the pre-trained model.
On the other hand, Bell et al. [3] introduced completely new and larger dataset with 23 material categories on which they fine-tuned a pre-trained model. Authors thus proved that transfer learning also works on different dataset than it was originally trained on, at least for image segmentation.
Unsurprisingly, the deeper the trained model was, the better it performed, with either GoogLeNet (22 layers) or VGG (16 layers) as winners in both publications.
After introduction of residual networks, the state-of-the-art network for segmentation became DeepLab [4], taking advantage of its unprecedented depth - model that was retrained had more than 100 layers.

Chapter 3

Our approach

3.1 Inverse rendering

As we explained in chapter 2, the two publications related to inverse rendering used improvements of SUNCG dataset. This dataset however, is subject to ongoing lawsuit [6], so authors of both papers could not made their datasets or trained models publicly available. At the time of writing this thesis, the lawsuit was still not resolved or settled, so we were not able to try and compare the trained models.
To solve this problem we decided to replicate paper by Sengupta et al. [23], as it was easier to reproduce and thus would serve as better baseline moving forward. Our approach, however, was little different. As most of the materials in real world are not only made of diffuse component, but rather as combination of diffuse and specular part, we aimed to (on top of 3 properties estimated by Sengupta et al.) train neural network to also predict specular albedo, glossiness and view vector, i.e. to estimate all Phong parameters from one image. To do this, we extended IRN architecture by stacking more residual blocks for each added parameter.
As only diffuse albedo ρ_d, environment map and normals were estimated from IRN in the original paper, the only choice for BRDF was ideal diffuse BRDF defined as

$$f_r = \frac{\rho_d}{\pi} \tag{3.1}$$

When inspecting code for direct renderer written by Sengupta et al., we found out that the equation for computing direct illumination was as follows:

$$f_{direct} = \frac{4\pi * \frac{\pi}{2}}{648} \sum_{i=1}^{648} \frac{\rho_d}{\pi} * L(\omega_i) * (\omega_i \cdot N) \tag{3.2}$$

where 648 corresponds to 18x36 light directions (one for each pixel of the environment map predicted by IRN), ρ_d stands for diffuse albedo (and thus the term $\frac{\rho_d}{\pi}$ for diffuse BRDF), ω_i for direction of incoming light vector, $L(\omega_i)$ for incoming illumination from

direction ω_i (i.e. value of the corresponding pixel of the environment map) and N for normal of the surface. Term $4\pi * \frac{\pi}{2}$ / 648 corresponds to size of one pixel of environment map when mapped to sphere.
Equation 3.2 is incorrect, as pixels mapped closer to the poles of the sphere will occupy less sphere surface than those mapped closer to equator of the sphere. To account for this distortion, the contribution of incoming light from direction ω_i should be multiplied by the constant size of a pixel $4\pi * \frac{\pi}{2}/648$ times cosine of deviation of the direction from the equator, i.e. $\cos\theta_l$ for $\omega_i = (\theta_l, \phi_l)$ in spherical coordinates. To summarize, fixed direct render function derived from equation 3.2 is then

$$f_{direct} = \frac{4\pi * \frac{\pi}{2}}{648} \sum_{i=1}^{648} \frac{\rho_d}{\pi} * L(\omega_i) * \cos\theta_l * (\omega_i \cdot N) \tag{3.3}$$

As our modified IRN predicts all Phong parameters, we decided to use physically based Phong as our BRDF model. To recall, BRDF for physically based Phong is

$$f_r^{Phong} = \frac{\rho_d}{\pi} + \frac{\rho_s(n+2)\cos^n\theta_r}{2\pi} \tag{3.4}$$

where ρ_d and ρ_s are diffuse and specular components respectively, n stands for glossiness and $\cos^n\theta_r = (V \cdot R)^n$, V representing view vector and R corresponds to reflected vector, which is a light vector flipped according to normal and its calculation is specified in equation 3.5.

$$R = 2(L \cdot N)N - L \tag{3.5}$$

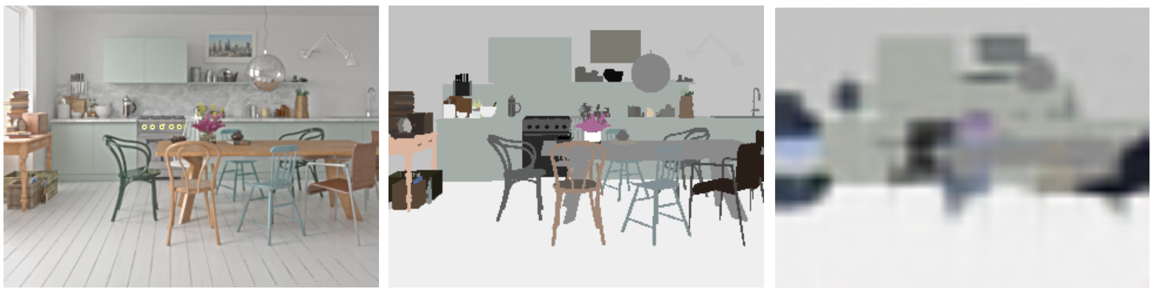
In conclusion, our direct render function with physically based Phong BRDF is defined as

$$f_{Phong} = \frac{4\pi * \frac{\pi}{2}}{648} \sum_{i=1}^{648} (\frac{\rho_d}{\pi} + \frac{\rho_s(n+2)\cos^n\theta_r}{2\pi}) \ * L_i(\omega_i) * \cos\theta_l * (\omega_i \cdot N) \tag{3.6}$$

When implemented by matrix multiplication, our direct renderer function runs almost as fast as the original implementation of the direct renderer, even though our function deals with twice as many parameters. Comparison of results from original direct render function and our own implementation can be found in Appendix.

3.2 Material Segmentation

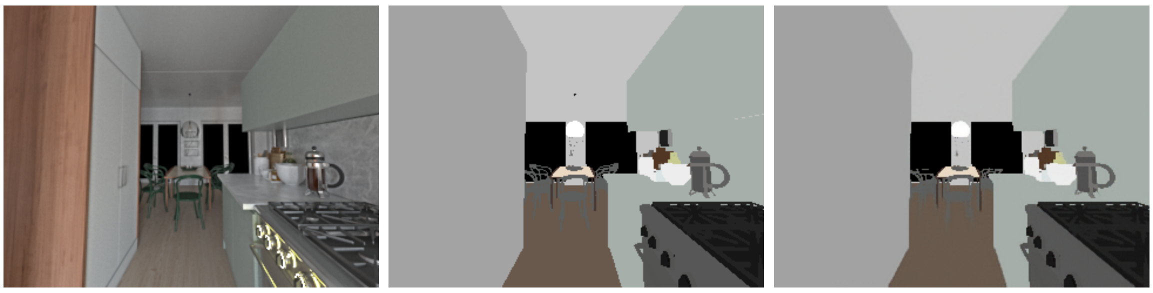
We tried to follow the transfer learning approach for material segmentation. As our primary goal is not to know precisely what is the class of the segmented material, our solution focused on material segmentation in the image without classification, which slightly simplifies the problem. When initialized, most architectures described in the previous chapter take number of classes as an argument, with this number describing how many feature maps should the model have in the output layer. To get a segmented image from neural network without post-processing (as we do not have the exact number of classes for materials in real world), we trained DeepLab model with 3 feature maps as output to mimic RGB image. To our surprise, the model was not able to learn underlying segmentation, even when trained from scratch, as shown in figure 3.1.



(a) Original image (b) GT segmentation (c) Predicted segmentation

Figure 3.1: Incorrect segmentation by DeepLab model

To solve this problem, we chose different architecture, in particular the one that we are using in IRN for estimating normals or albedo. This architecture had no problem to learn underlying segmentation, as presented in figure 3.2. We named this network Material Segmentation Net, or MSN for short. Exact architecture of the model is described in section 5.4.



(a) Original image (b) GT segmentation (c) Predicted segmentation

Figure 3.2: Proof of work - MSN

Chapter 4

Dataset

The main goal of machine learning is to gain the ability to generalize well on new, previously unseen data, in our case real-world images. This generalization is often only possible if the testing data comes from the same distribution as the training data. This distribution is sometimes difficult to obtain, especially in computer vision and computer graphics as we usually work with real-world imagery, for which is problematic to obtain ground-truth data. While we can collect depth and normals of a scene via depth sensors [17], it is complicated to generate data for albedo, lighting or other parameters.
This is where physically based rendering comes into play. When we render a scene following physically based techniques (thus, physically simulate light transportation in the scene), we can generate real-world like images for which we can obtain many properties of the rendered image, hence bridging the gap between synthetic datasets and real images.
Because of the lawsuit regarding SUNCG dataset we deciced to render our own dataset using PBR techniques to match the required image quality. Its scene complexity and richness of materials is unmatched by previously used datasets, which makes it superior for training. Some examples of images in our dataset are shown in figure 4.1.
We started generating the dataset from ≈140 scenes downloaded from Evermotion website [5] by placing ≈10 virtual cameras inside every scene using 3ds Max [1]. All images were rendered via V-Ray renderer [8] from the viewports of those cameras to produce unique geometry for every camera view, with example in figure 4.2.
To further enlarge the dataset, we make use of V-Ray's Light select feature [9] to turn on $\frac{1}{10}$ of lights presented in the scene, which generated up to 10 images of the same geometry under different lighting. As we found out during rendering, V-Ray optimizes computation for the main image, which has most of the lights turned on. When only a subset of lights is turned on for light select element, rendered image can be quite noisy, or due to poor selection of the lights, it can even be full black image. In scenes where this approach did not work and rendered light select images were noisy, we had to do

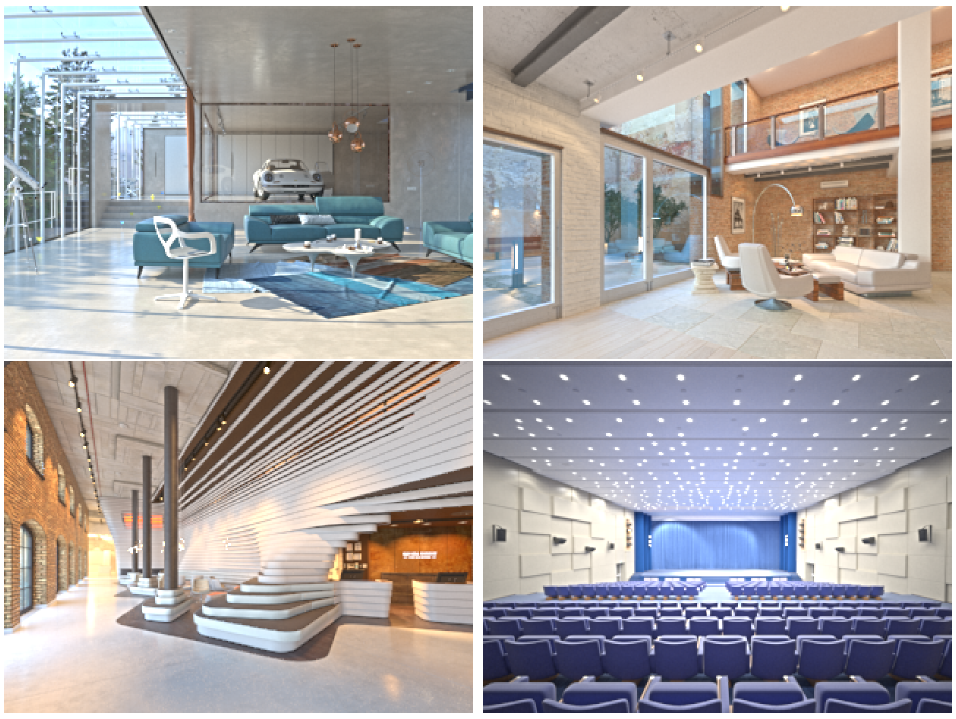


Figure 4.1: Examples of images in our dataset

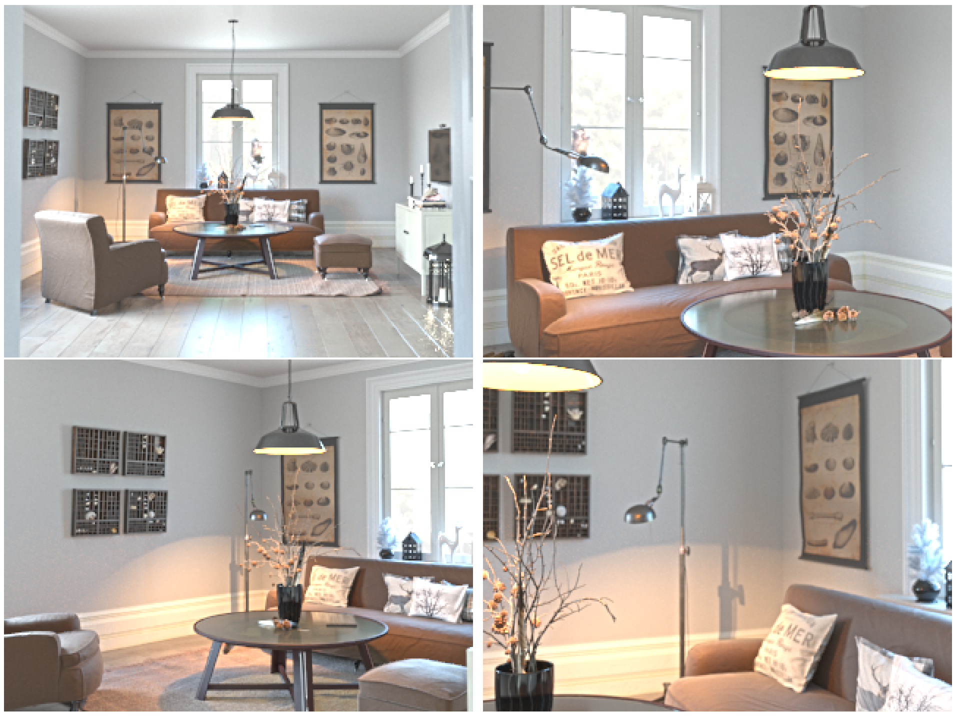


Figure 4.2: Different camera views for the same scene

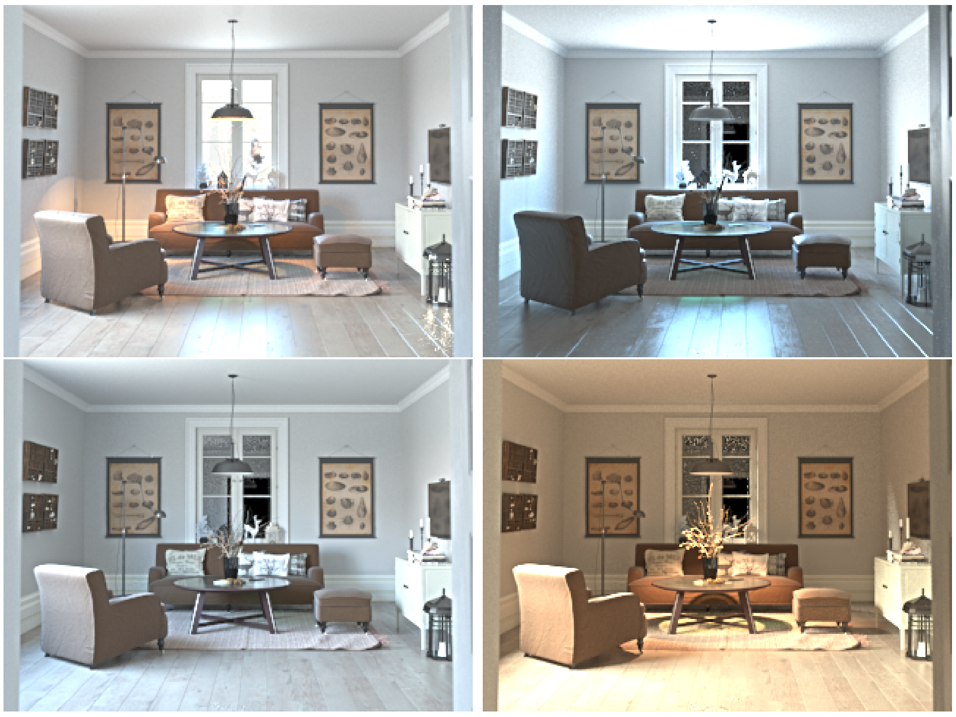
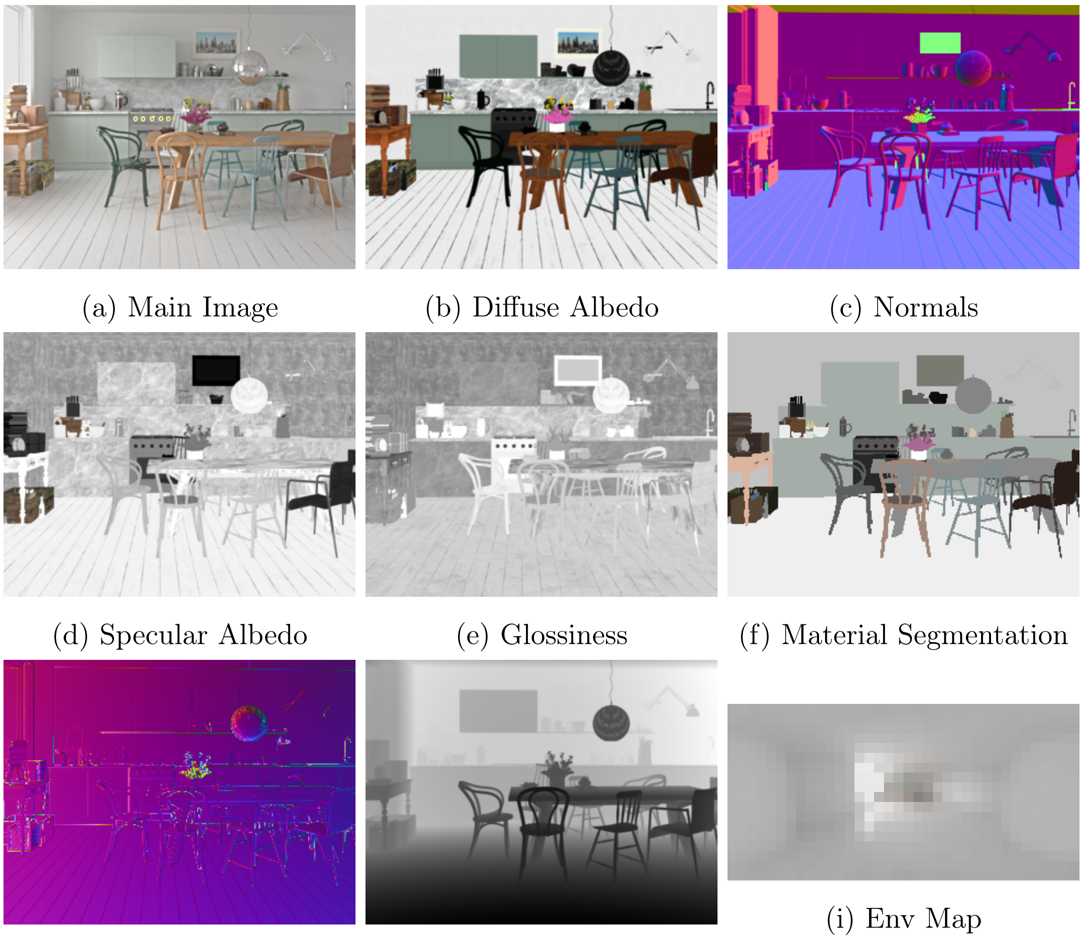


Figure 4.3: Different lighting for the same camera view

it the other way around - we turned on $\frac{9}{10}$ of all lights in the scene. Combination of these two approaches generated 5-7 different lighting conditions for most of the scenes. Examples of mentioned light alternations are shown in figure 4.3. As we decided to choose similar approach as showed in [23], we also had to include environment maps into our dataset. To ensure that we had enough maps for training, we opted for combination of publicly available HDRI maps on HDRI Haven website [11] - with 105 maps - and our own dataset of environment maps by generating 360° panoramas of size 18×36 pixels for every scene, yielding 1005 maps. In total, we have about $5\times$ more environment maps than was used in [23].
All of the rendering was done on powerful 30 core CPU machines. Altogether, the dataset took about 1000 hours to render. Some scenes took very long to render, so we also tried rendering on GPU, but for some unknown reason we were not able to match the same image quality. If we want to render larger dataset in the future, we will have to find a way to fix the setup for GPU rendering, as it presents great promises in reducing rendering time.
In the end, our dataset consists of 1041 unique scene geometries, 1110 environment maps and 5709 images in total under different lighting conditions with ground-truth data like diffuse albedo, specular albedo, normals, depth, glossiness and view vector obtained as render elements feature of V-Ray [10]. V-Ray provides also a render element containing index of a directly visible material for every pixel which we used to create ground-truth data for material segmentation. We modified output of the original render



(a) Main Image (b) Diffuse Albedo (c) Normals

(d) Specular Albedo (e) Glossiness (f) Material Segmentation

(g) View vector (h) Depth (i) Env Map

Figure 4.4: Example of ground-truth data for a scene

element by replacing each index by an RGB color computed by taking the most frequent diffuse albedo ρ_d^* and specular albedo ρ_s^* in all pixels with the same index and combining them using formula:

$$\frac{\rho_d^* + \rho_s^*}{2}$$

In our testing, we found this to work well in assigning different colors to different materials and not overlap too much. One problem, however, arises. For now, we do not have to know the values of diffuse and specular albedo that made the final pixel value in the segmented image. If we wanted to get those values (for example, to adjust values predicted by neural net), we would have to choose an invertible coding. In figure 4.4, you can see example of ground-truth data for one scene.
As V-Ray supports more than 60 render elements, our dataset is easily extendable with new characteristics of a scene for additional work in the future.

Chapter 5

Architecture Design

In this chapter, we present architecture for all models that we decided to train, specifically **EnvMap** for estimating environment maps, **IRN** for estimating intrinsic properties, **RAR** for indirect illumination and **MSN** for material segmentation. Our whole pipeline for inverse rendering and material segmentation is shown in figure 5.1.

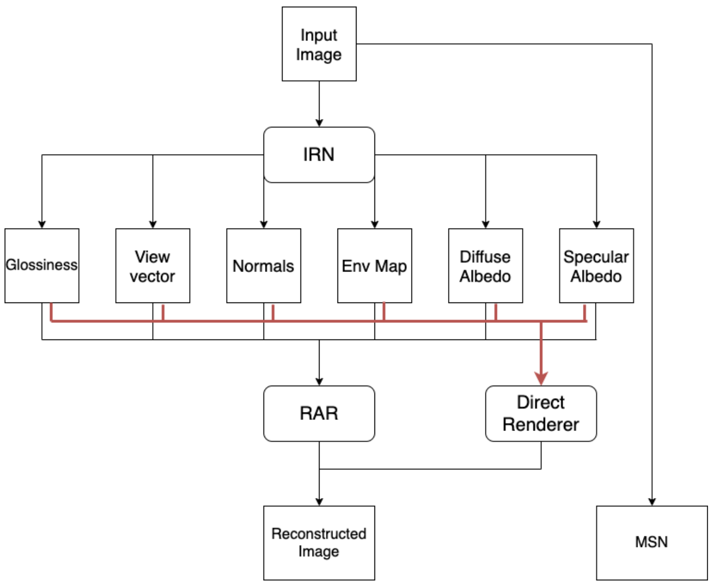


Figure 5.1: Our pipeline

5.1 EnvMap

Our EnvMap model's architecture is defined as follows:

ReflectionPad(3) → Conv7×7(3, 64) → Conv3×3(64, 128) → Conv3×3(128, 256) → 4 × ResNetBlock(256) → Conv1×1(256, 256) → Conv3×3(256, 256) → Conv3×3(256, 128) → Conv3×3Tanh(128, 3) → Upsample(18, 36) where ConvN×N(x, y) indicate 2D convolutional layer with kernel of size $N \times N$ and stride 2, x input channels and y output channels, succeeded by batch normalization and ReLU activation; ConvN×NTanh(x, y) stands for ConvN×N(x, y), but with Tanh as activation function; ReflectionPad(N) represents reflection padding with N padded items in each direction; Upsample(x, y) denotes layer that upsamples input into output with size $x \times y$ using bilinear interpolation and 4 × ResNetBlock(N) is a series of 4 consecutive block, with each ResNetBlock being

ReflectionPad(1) → Conv3×3(N, N) → BN(N) → ReLU →
ReflectionPad(1) → Conv3×3(N, N) → BN(N)

where BN(N) denotes batch normalization over input of size N × N.

5.2 IRN

IRN consists of encoder **Enc**, defined as

ReflectionPad(3) → Conv7x7(3, 64) → Conv3x3(64, 128) → Conv3x3(128, 256)

which output is then fed through 9 × ResNetBlock for each estimated parameter except for light estimation, which is handled separately. Each parameter (except for light) is then upsampled to original input size by decoder **Dec**, defined as

TransConv3×3(256, 128) → TransdConv3×3(128, 64) → ReflectionPad(3) →
Conv7×7(64, 3) → Tanh

with glossinness as an exception, which the second-to-last layer is Conv7×7(64, 1). TransConvN×N(x, y) represent transposed convolution with kernel size N × N, x input and y output feature maps respectively.
Input to light estimation module is concatenation of **Enc** output and output after 9 × ResNetBlocks for every estimated parameter along channel dimension, which yields

Conv1×1(1536, 256), → Conv3×3(256, 256), → Conv3×3(256, 128), →
Conv3×3Tanh(128, 3), → Upsample((18, 36))

5.3 RAR

RAR architecture is implemented in encoder-decoder fashion, which is taken from U-Net [21], alongside encoder for the input image. Encoder for the input image is defined as

ReflectionPad(3) $\rightarrow$ Conv7×7(3, 64) $\rightarrow$ Conv3×3(64, 128) $\rightarrow$
Conv3×3(128, 256) $\rightarrow$ Conv1×1(256, 128) Conv3×3S1(128, 64) $\rightarrow$
Conv3×3(64, 32) $\rightarrow$ Conv3×3(32, 16) $\rightarrow$ Linear(4800, 300)

with transformation between output of second-last layer of spatial resolution (16 × 15 × 20) into 4800 features as input into the last linear layer; ConvN×NS1(x, y) denotes the same as ConvN×N(x, y), but with stride 1.
U-net encoder **Enc** takes all estimated parameters, concatenated along channel dimension

Conv3×3S1(13, 64) $\rightarrow$ Conv3×3(64, 64) $\rightarrow$ Conv3×3(64, 128) $\rightarrow$
Conv3×3(128, 256) $\rightarrow$ Conv3×3(256, 512)

and decoder **Dec** specified as

UpConv3×3(513, 512) $\rightarrow$ UpConv3×3(768, 256) $\rightarrow$ UpConv3×3(384, 128) $\rightarrow$
UpConv3×3(192, 64) $\rightarrow$ Conv1×1Tanh(64, 3)

where UpConvN×N(x, y) is block defined as

Upsample(2) $\rightarrow$ Convolution3×3P1(x, y) $\rightarrow$ BN(y) $\rightarrow$ ReLU.

where Upsample(2) is layer that upsamples input by factor of 2 and Conv3×3P1(x, y) denotes 2D convolutional layer with kernel of size 3×3, x input a y output channels, stride 2 and padding 1. Skip connections are implemented between Conv3×3(*, N) and UpConv3×3(*, N).

5.4 MSN

Architecture for MSN is set as **Enc** $\rightarrow$ 9 × ResNetBlock $\rightarrow$ **Dec** with ResnetBlock defined in section 5.1 and **Enc** and **Dec** defined in section 5.2.

Chapter 6

Implementation and training

Because of its excellent machine learning support, we chose to write all of our code in Python and train all the models using the PyTorch framework [20] because of its straightforward setup for distributing training on multiple GPUs.
We've performed training on two powerful servers, both of them equipped with two NVIDIA GeForce RTX 2080 Ti graphic cards. Thanks to this graphic card's big RAM, we were able to fit reasonably large batch sizes, which significantly reduced training time and stabilized training across all models.
As PyTorch does not support any visualization of the training process out of the box, we integrated Tensorboard into our training procedure, logging training and validation error after each epoch of training, of which an example is shown in figure 6.1.

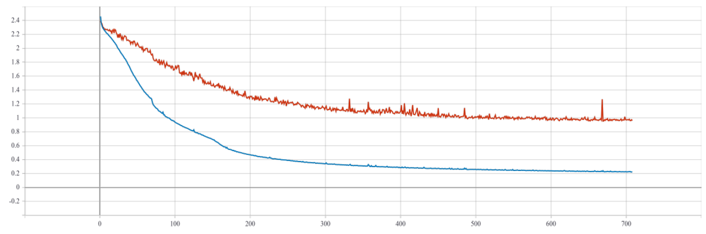


Figure 6.1: Our Tensorboard logging setup, with training (blue) and validation (red) error

6.1 Training procedure

Following training procedure suggested in [23], we started training EnvMap net, first on images synthesized by direct renderer presented by Sengupta et al. and then fine-tune it on raytraced images with loss functions specified in the original paper. With trained EnvMap net, we generated ground-truth environment map for each image in our dataset. As we have explained in chapter 3, using their direct renderer did not work well on our data, as our images were rendered not only with diffuse but also

specular materials, which caused problems. We realized this problem after we trained and fine-tuned the model, so we had to retrain it from scratch.
With all the ground-truth data ready, we trained IRN with the following loss:

$$\begin{aligned} L_{IRN} = & \|\hat{N} - N^*\|_1 + \|\hat{D} - D^*\|_1 + \|\hat{G} - G^*\|_1 + \|\hat{V} - V^*\|_1 + \|\hat{S} - S^*\|_1 \\ & + \frac{\|f_{Phong}(D^*, N^*, \hat{L}, G^*, V^*, S^*) - f_{Phong}(D^*, N^*, L^*, G^*, V^*, S^*)\|_1}{2} \end{aligned} \tag{6.1}$$

where * stands for ground-truth data, $\hat{}$ stands for predicted data, N, D, G, V, S correspond to surface normals, diffuse albedo, glossiness, view vector and specular albedo respectively and f_{Phong} denotes our Phong direct render function, as specified in chapter 3.
To train RAR

$$I_r = RAR(I) \tag{6.2}$$

we adjusted the original RAR loss to account for our new direct renderer as

$$L_{RAR} = \|I - (I_r + f_{Phong}(\hat{D}, \hat{N}, \hat{L}, \hat{G}, \hat{V}, \hat{S}))\| \tag{6.3}$$

where I denotes input image and $\hat{D}, \hat{N}, \hat{L}, \hat{G}, \hat{V}, \hat{S}$ image properties estimated by IRN.
We chose Adam [13] as our optimizer for minimizing cost function, as this optimization method outperformed all other methods by constantly giving lower training and validation error. All models were optimized with learning rate $\alpha = 0.001$.
The time required to fully train the model varied between architectures, from 2 days for the smaller ones to one week for IRN.

Chapter 7

Results

In this chapter, we present results of our trained models, on both training and testing datasets to see how well they generalize.

7.1 Inverse rendering

7.1.1 IRN

We observed that IRN had problems learning normals and view vector much more than other parameters and was only able to learn somewhat meaningful priors only in later stages of training. As complex geometries are used for scenes in our dataset, it did not come as a surprise that IRN had hard time learning normal or view vectors. Other estimated parameters look very similar to ground-truth data, even on test dataset, as shown in figures 7.2 and 7.3.

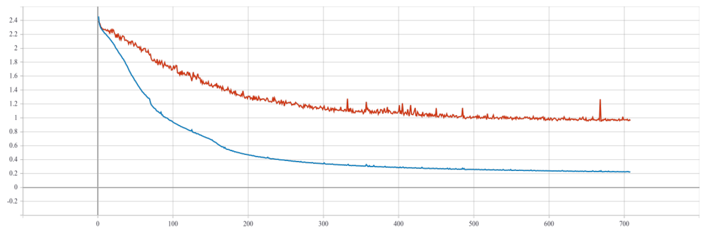
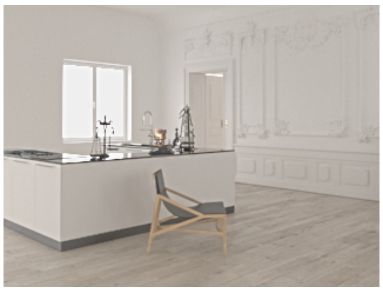
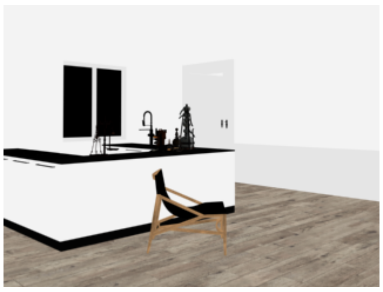


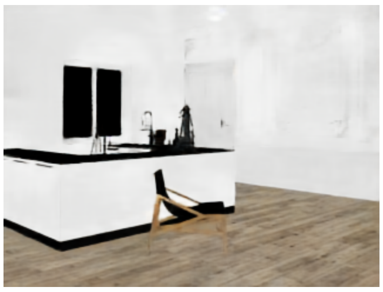
Figure 7.1: Training (blue) and validation (red) error during training IRN

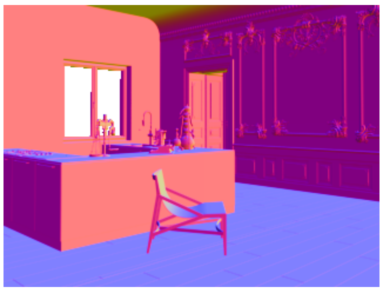
7.1.2 RAR

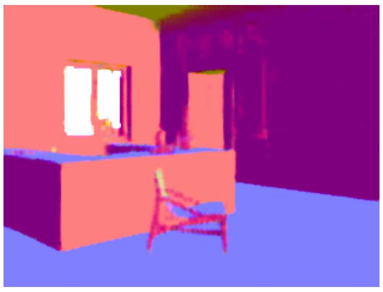
Training of RAR net was somewhat successful, as the network was able to learn illumination and shadows that are not present in image computed by direct renderer, even on test dataset, as shown in figures 7.5 and 7.6. But the reconstruction is not perfect, as we have used estimated data from IRN to compute direct image, which

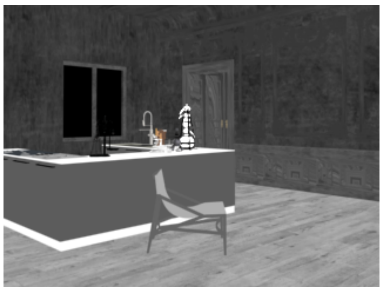

(a) Original image

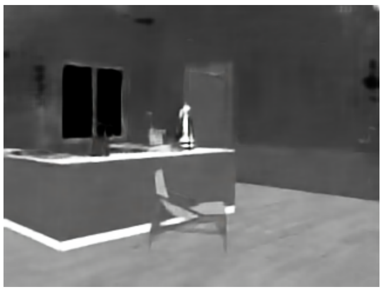

(b) GT diffuse albedo

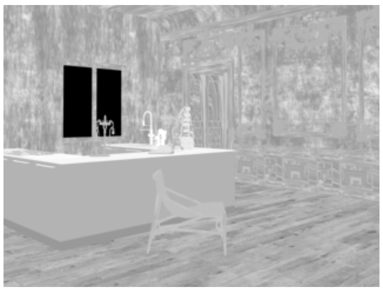

(c) Predicted diffuse albedo

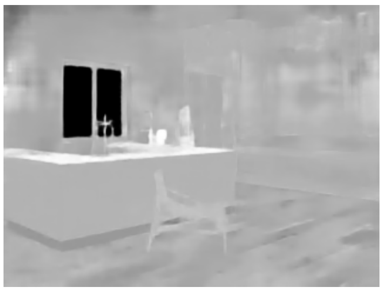

(d) GT normals

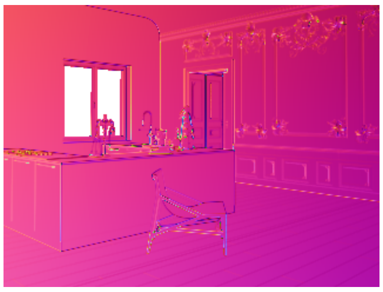

(e) Predicted normals


(f) GT specular albedo


(g) Predicted specular albedo


(h) GT glossiness


(i) Predicted glossiness


(j) GT view vector

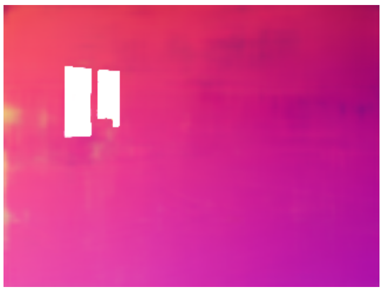
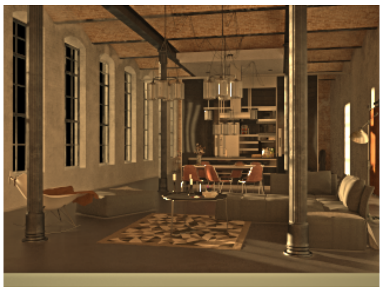
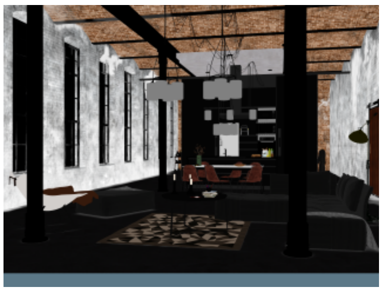
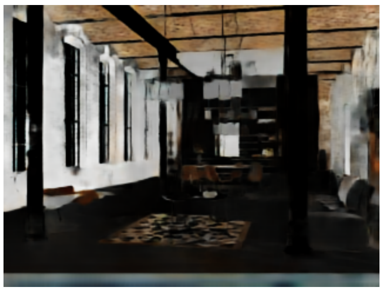
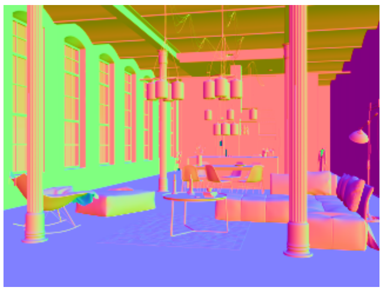

(k) Predicted view vector

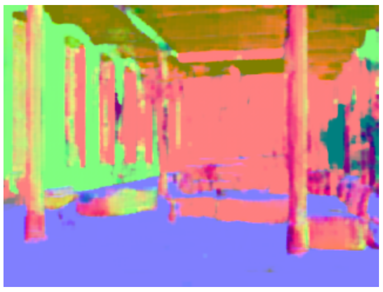
Figure 7.2: IRN results on train data

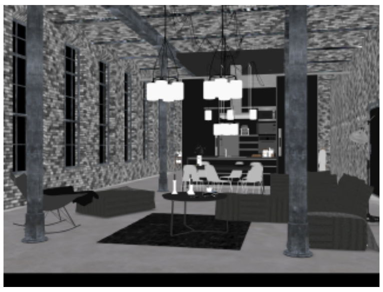

(a) Original image

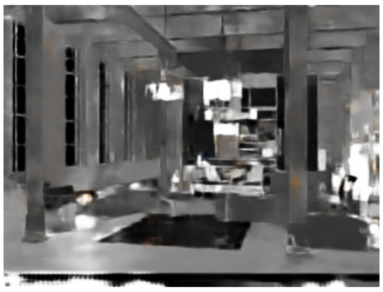

(b) GT diffuse albedo

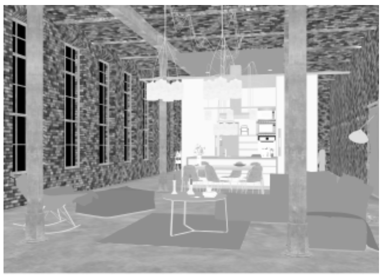

(c) Predicted diffuse albedo

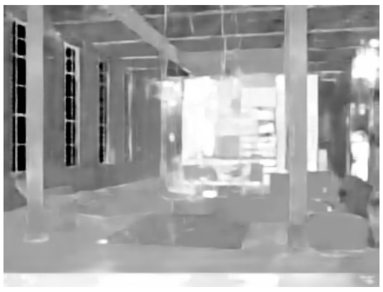

(d) GT normals

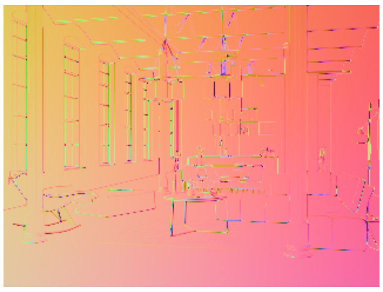

(e) Predicted normals


(f) GT specular albedo


(g) Predicted specular albedo


(h) GT glossiness


(i) Predicted glossiness


(j) GT view vector

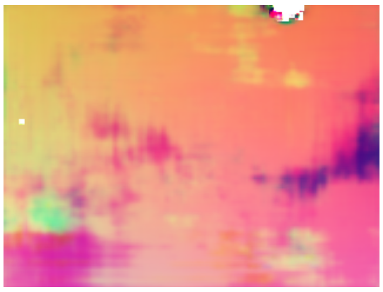

(k) Predicted view vector

Figure 7.3: IRN results on test data

already accounts for some error and this error gets amplified even more. We could fix this by train RAR from scratch with direct image being computed from ground-truth data.

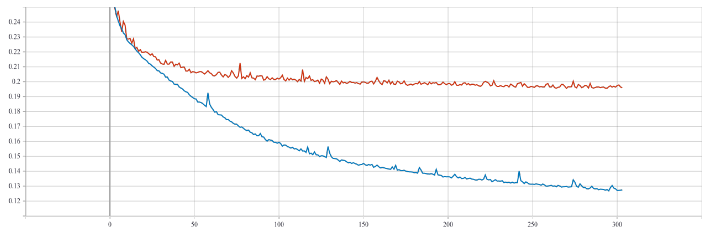
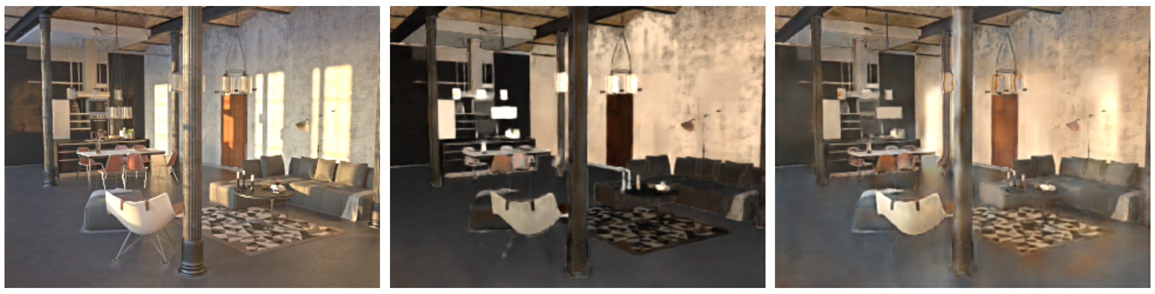
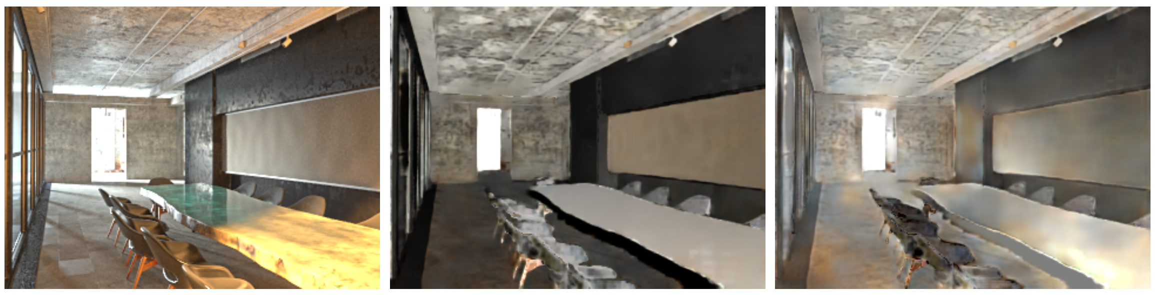


Figure 7.4: Training (blue) and validation (red) error during training RAR



(a) Original image (b) Direct image (c) Reconstructed image

Figure 7.5: RAR results on train data



(a) Original image (b) Direct image (c) Reconstructed image

Figure 7.6: RAR results on test data

7.2 Material Segmentation

MSN also performed well on both train and test sets, as shown below. Results could be improved by using deeper architecture, as architectures for material segmentation in the past used much more layers, which is also subject for our future work.

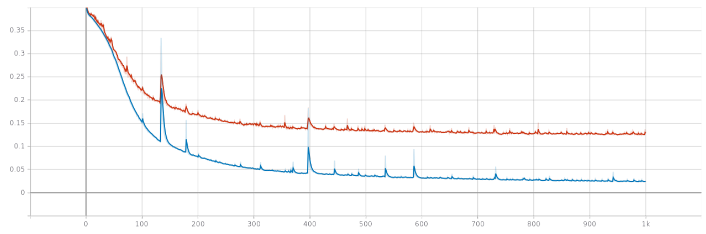
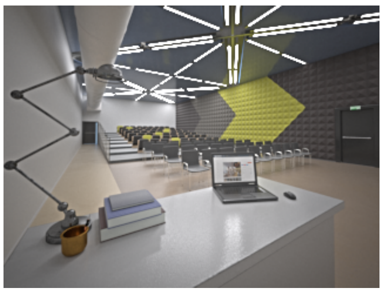
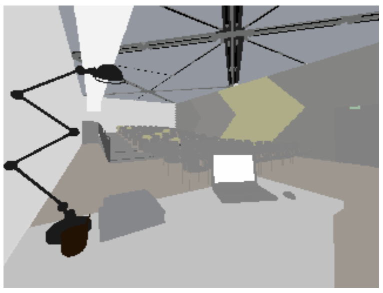


Figure 7.7: Training (blue) and validation (red) error during training MSN


(a) Original image


(b) GT segmentation

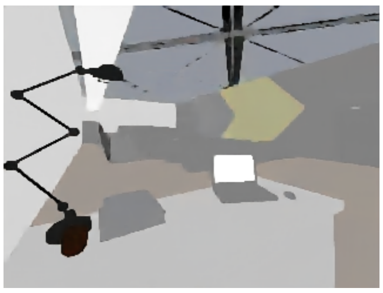
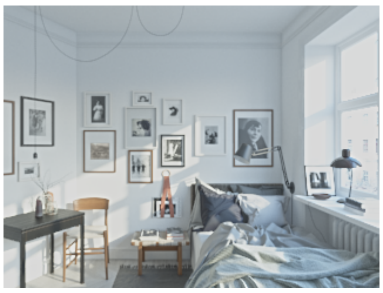
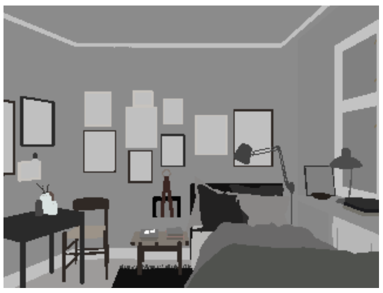

(c) Predicted segmentation

Figure 7.8: MSN results on train data


(a) Original image


(b) GT segmentation

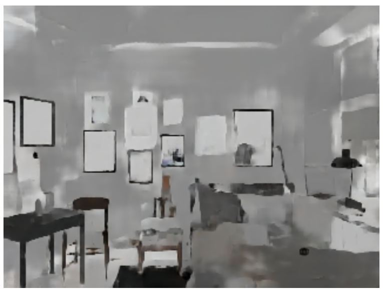

(c) Predicted segmentation

Figure 7.9: MSN results on test data

Conclusion

In this thesis, we presented a method for the per-pixel estimation of material properties in the image by training deep neural networks. We demonstrated that deep neural networks are powerful learning representations that can learn useful priors, even when it comes to such unconstrained problems like inverse rendering. Our pipeline collectively estimates diffuse and specular albedo, surface normals, glossiness, view vector, and illumination, alongside per-pixel material segmentation, from a single image. These properties are then used to re-synthesize the input image from its components. We provide an implementation of our direct render function that makes this synthesis possible.

We acknowledge, however, that our solution is not perfect and there is a significant amount of work to be done in the future, as our method infers only a handful of scene characteristics, which is not enough for most real-life use cases. First, we believe that our results could improve dramatically if we had more data in our dataset, thus leading to better generalization across all models. Because of that, enlarging our dataset will be our foremost goal.

To make our tool accessible to artists, we want to create a plugin for 3ds Max as a wrapper for our trained models. By integrating our models directly into the program, the estimated properties of a user-specified object in an image would be transferred onto the 3D model created in 3ds Max, also specified by the user.

One of the potential improvements to our work is to predict spatially varying lighting, approach very similar to that proposed in [15]. Currently, we have only one environment map per image, which is a very rough approximation of lighting in the scene. To improve this estimate, we could generate an environment map (or some less parameter dependant representation, like spherical Gaussians) at every pixel in the scene, which is doable, as we know (from V-Ray) what is the first intersection point in the scene from the camera view.

The next step to further simplify the process of setting material appearance is to enable texture transfer, as having good texture for the material is equally important for final material looks as setting correct values for material parameters. Previous work on texture transfer was only limited to some classes of objects [28], which is not satisfactory for our use case. We admit that this problem is extremely challenging, but

we want to try nevertheless.
Altogether, we are convinced that it is worth working on problems like inverse rendering and material segmentation, as it can find a large variety of applications in the real world, even beyond the computer science community.

Bibliography

[1] Autodesk. 3ds Max. `https://www.autodesk.com/products/3ds-max/overview`, 2020. Retrieved: 26-05-2020.

[2] Jonathan T. Barron and Jitendra Malik. Shape, illumination, and reflectance from shading. *TPAMI*, 2015.

[3] Sean Bell, Paul Upchurch, Noah Snavely, and Kavita Bala. Material recognition in the wild with the materials in context database. *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[4] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PP, 06 2016.

[5] Evermotion. Evermotion. `https://evermotion.org/`, 2020. Retrieved: 25-05-2020.

[6] Futurism. A startup is suing Facebook, Princeton for stealing its AI data. `https://futurism.com/tech-suing-facebook-princeton-data`, 2019. Retrieved: 26-05-2020.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[8] Chaos Group. V-Ray. `https://www.chaosgroup.com/vray/3ds-max`, 2020. Retrieved: 21-04-2020.

[9] Chaos Group. V-Ray light select. `https://docs.chaosgroup.com/display/VRAY4MAX/VRayLightSelect`, 2020. Retrieved: 25-05-2020.

[10] Chaos Group. V-Ray render elements. `https://docs.chaosgroup.com/display/VRAY4MAX/Render+Elements`, 2020. Retrieved: 21-04-2020.

[11] HDRI Haven. Hdri Haven. `https://hdrihaven.com/hdris/?c=indoor`, 2019. Retrieved: 26-05-2020.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[13] Diederik Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations*, 12 2014.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[15] Zhengqin Li, Mohammad Shafiei, Ravi Ramamoorthi, Kalyan Sunkavalli, and Manmohan Chandraker. Inverse rendering for complex indoor scenes: Shape, spatially-varying lighting and SVBRDF from a single image. *CoRR*, abs/1905.02722, 2019.

[16] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440, 2015.

[17] Pushmeet Kohli Nathan Silberman, Derek Hoiem and Rob Fergus. Indoor Segmentation and Support Inference from RGBD Images. In *ECCV*, 2012.

[18] Michael A Nielsen. *Neural networks and deep learning*, volume 2018. San Francisco, CA, USA:: Determination press, 2015.

[19] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2016.

[20] PyTorch. PyTorch. `https://pytorch.org/`, 2020. Retrieved: 28-05-2020.

[21] O. Ronneberger, P.Fischer, and T. Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 9351 of *LNCS*, pages 234–241. Springer, 2015. (available on arXiv:1505.04597 [cs.CV]).

[22] Sumit Saha. A comprehensive guide to convolutional neural networks — the ELI5 way. `https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53`, 2018. Accessed: 02-03-2020.

[23] Soumyadip Sengupta, Jinwei Gu, Kihwan Kim, Guilin Liu, David W. Jacobs, and Jan Kautz. Neural inverse rendering of an indoor scene from a single image. *CoRR*, abs/1901.02453, 2019.

[24] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556*, 09 2014.

[25] Shuran Song, Fisher Yu, Andy Zeng, Angel X Chang, Manolis Savva, and Thomas Funkhouser. Semantic scene completion from a single depth image. *arXiv preprint arXiv:1611.08974*, 2016.

[26] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[27] Petr Vévoda, Ivo Kondapaneni, and Jaroslav Křivánek. Bayesian online regression for adaptive direct illumination sampling. *ACM Trans. Graph.*, 37(4):125:1–125:12, July 2018.

[28] Tuanfeng Y. Wang, Hao Su, Qixing Huang, Jingwei Huang, Leonidas Guibas, and Niloy J. Mitra. Unsupervised Texture Transfer from Images to Model Collections. *ACM Trans. Graph.*, 35(6), 2016.

Appendix: Comparison of results of direct render implementations

In this appendix, we present comparison of results between fixed direct renderer implementation as defined in equation 3.3 and our own implementation using physically correct Phong BRDF. As we can see in figure 7.10, due to inclusion of specular albedo and glossiness into the implementation, we can render much better images that are more similar to the original image. Images were rendered with an environment map inferred by our trained EnvMap model and ground-truth data for each scene.

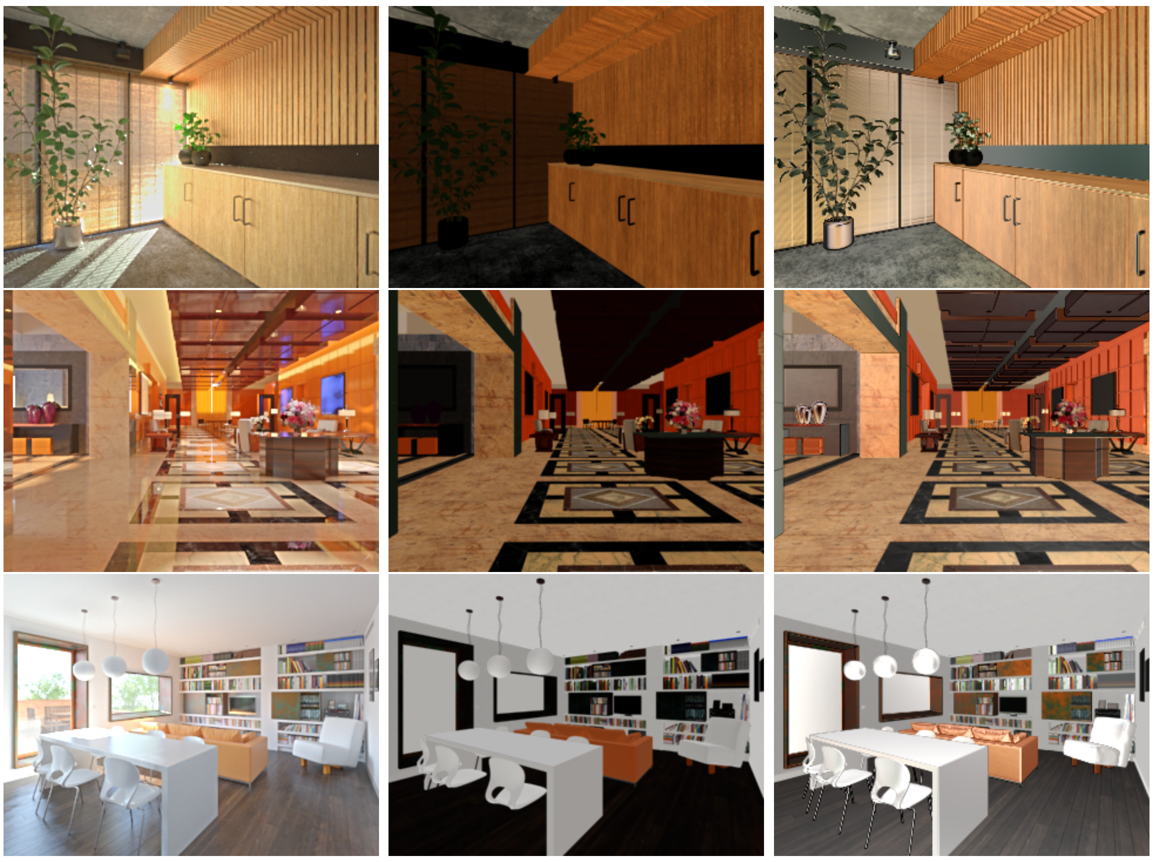


Figure 7.10: Comparison of direct render results, with original image (left), image rendered by original direct render implementation (middle) and image rendered by our own implementation of direct render (right)