Dominik Hirner

Photometric Stereo in Multi-Line Scan Framework under Complex Illumination via Simulation and Learning

MASTER'S THESIS

to achieve the university degree of
Diplom-Ingenieur

Master's degree programme
Computer Science

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Dipl.-Ing. Dr.techn. Thomas Pock
Institute for Computer Graphics and Vision

Mgr. Svorad Štolc PhD
Austrian Institute of Technology

Graz, Austria, Mar. 2017

To my parents Gabriele and Günther Hirner

If you thought that science was certain - well, that is just an error on your part.

Richard P. Feynman (1918 - 1988)

Abstract

In this work we present a neural network implementation of photometric stereo formulated as a regression task. Photometric stereo estimates the surface normals by measuring the irradiance of any given visible point under different lighting angles. Instead of the traditional setup, where the object has a fixed position and the illumination angles changes around the object, we use two constant light sources. In order to produce different illumination geometries, the object is moved under a multi-line scan camera. In this thesis we explore an approach where we present an artificial neural network with a number of intensity vectors (i.e. points with constant albedo under different illumination angles) from randomly chosen pixels of six materials with different reflectance properties. The network is trained to estimate the gradient of the surface along the transport direction of the given point. This eliminates the need of the explicit knowledge of the light source positions or direction while still retaining a competitive accuracy even when presented with materials which have non-Lambertian surface properties. Due to the random pooling of the pixels our implementation is also independent from spatial information.

Keywords. photometric stereo, neural network, brdf, machine learning, non-Lambertian surface

Kurzfassung

In dieser Arbeit präsentieren wir eine Implementation eines Neuronalen Netzwerkes für Photometrisches Stereo das als Regressions Aufgabe formuliert wurde. Photometrisches Stereo schätzt Oberflächennormalen durch das Messen von Irradianz von jedem gegebenen sichtbaren Punkt anhand von verschiedenen Beleuchtungswinkeln. Anstatt der traditionellen Konfiguration, bei welcher das Objekt an einem fixierten Punkt liegt und nicht bewegt wird, während sich die Beleuchtungswinkel um das Objekt herum verändern, verwenden wir zwei statische Lichtquellen. Das Objekt wird unter der multi-line scan Kamera bewegt um die verschiedenen Beleuchtungswinkel zu produzieren. In diesem Werk erkunden wir einen Vorgang, bei welchem wir ein Neuronales Netz mit einer Nummer von Intensitätsvektoren von zufällig gewählten Pixeln (hier Punkte mit konstanten Albedo unter verschiedenen Beleuchtungswinkeln) aus sechs Materialkategorien präsentieren. Wir trainieren das Netzwerk damit es Oberflächennormal Gradienten in Transportrichtung des gegebenen Punktes (Pixels) schätzt. Das erlaubt uns unabhängig von der Beleuchtungsposition- und Winkels zu sein und trotzdem noch immer wettbewerbsfähige Genauigkeit zu haben, sogar bei nicht diffusen Oberflächen. Durch das zufällige Ziehen aller Datenpunkte eliminieren wir auch die Abhängigkeit räumlicher Information.

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used.

The text document uploaded to TUGRAZonline is identical to the present master's thesis dissertation.

Date	Signature

Acknowledgments

First and foremost I want to thank my two supervisors Univ.-Prof. Dipl.-Ing. Dr.techn. Thomas Pock and Mgr. Svorad Štolc PhD for their very detailed support and all their contributions for this work. I also want to thank them for the opportunity to take part in the ÖAGM workshop and to help me with my research paper in that regard. Furthermore I want to thank the team of the Institute of Computer Graphics and Vision and fellow master students in this institute for all their tips and discussions. Last but not least I want to thank the Austrian Institute of Technology for the possibility to work in their research department (formerly Digital Safety & Security, now Vision, Automation & Control) and for the great working environment. From this department I want to especially thank Dr. Reinhold Huber-Mörk and Doris Antensteiner for their help and discussions. I also want to thank Priska Rothammer for her help in creating the schematic drawings in this work. Thanks to Linux, TensorFlow and Blender.

Contents

List of Figures

List of Tables

1 Introduction

Contents

1.1 Reflection

The reflection of light on many real-world surfaces behaves in a very predictable manner. The most fundamental law that describes how light-rays are reflected on a surface is called the *Law of Reflection* (sometimes also referred to as normal reflection). This law states that the reflected ray (i.e. the outgoing ray) has the same angle as the incident ray (i.e. the light ray approaching the surface of the object) with respect to the surface normal (line perpendicular to the surface). This behavior was already known to the Hellenistic Roman Empire, where it was first described by Heron of Alexandria [18]. However this assumption only holds true if the reflective surface is a perfectly smooth, mirror-like surface, which most everyday object aren't. Most everyday objects rather have a rough surface, which means on a microscopic level the surface structure is not smooth but has

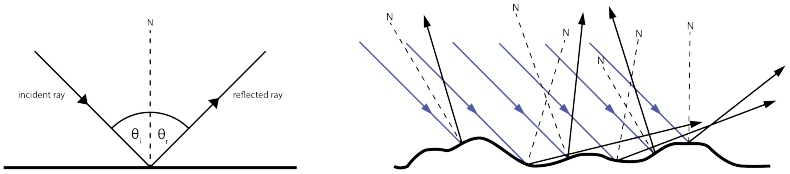

Figure 1.1: (left)Law of Reflection on mirror-like surface, also called specular reflection. (right) Reflection on a diffuse surface.

many irregularities. These small irregularities or bumps influence the luminance of the surface greatly. While the Law of Reflection still holds on a microscopic level, the surface normals on such a small scale varies greatly for most objects and thus the light will appear to be scattered in many different directions (shown on the right side of Fig. 1.1). This behavior is called *diffuse reflection*. This reflectance behaviour was first described by J.H. Lambert in 1760 in his work *Photometria* [26]. The reflection behaviour of both surface types (perfectly smooth and diffuse) are illustrated in Fig. 1.1.

The title of Lamberts work was back then a neologism and is composed from the greek word for light, *photos* and the greek word for measurement *metria*. This is the origin of the modern word *photometry* which is literally translated as *the measurement of light*. In this work Lambert introduced many concepts and terms that are used to this day.

One example is the term of an *ideal diffuse* surface, which he stated is a surface that scatters the light equally in all directions (i.e. it will have equal luminance from all viewing directions). Such surfaces follow the *cosine emission law*, nowadays more famously known as *Lambert's cosine law*. This law states, that the observed luminosity (or the lights intensity) of a perfectly diffuse surface (with constant albedo) is directly proportional to the cosine of the incident light to the surface normal.

$$I = \frac{l \; cos \; \Theta}{r^2} \tag{1.1}$$

Here I is the observed intensity, l is the luminous intensity, Θ is the angle between the surface normal and the incident ray and r is the distance between the light source and the object. The division by r^2 is called the *inverse-square law* and means that the observed intensity, or energy, decreases with the square of the distance between the light source and the illuminated object. However the inverse-square law can be neglected and factored out, if the light source is bright enough with respect to the distance and object size. For instance if one thinks about the sun as a light source, the observed luminance of an object stays almost the same if you put an object on the ground or a couple of meters up in the air.

Also one has to keep in mind that Eq. 1.1 is for a point-wise representation of light (i.e. single photons as rays). However it should be noted that light-rays usually occur in bundles, so that looking at a small area and rewriting the formulation as a differential equation of a small area can be beneficial. As this would not change the principle of

Figure 1.2: Specular highlights of an polished apple. The left highlight stems from an open window while the brighter right highlight is from an artificial light source.

Lambert's cosine law, the differential formulation is omitted.

Materials that exhibit this diffuse behaviour (equal luminance from all observer angles that change with the cosine of the incident light) are called *Lambertian*. Of course in reality there are no completely Lambertian materials, however it is often a nice approximation or simplification for diffuse reflecting surfaces in praxis. Most materials have both a diffuse and a specular component. For example even Lambertian materials can exhibit *specular highlights*. This highlights appear on smooth surfaces where the surface normal is exactly halfway between the incident angle and the observation angle (i.e. where it follows exactly the law of reflection and therefore has the highest luminance). As one would change either the illumination or observer angle, the perceived specular highlight would also wander to again appear in this described half-angle point.

Such specular highlights can be seen in Fig. 1.2 showing a polished apple placed on a table and illuminated by two different light sources (sun light from an open window and an artificial light source.) In theory these specular highlights should be perfectly sharp reflections of the light sources, however this is not always the case. The specular highlights can be more or less blurry. This blurriness of the specular reflection can be explained by the fact that one cannot think of the surface of an object as a continuum, rather it is composed of many smaller fractions, so called *microfacets*. Each of these microfacets is a perfect reflector (i.e. follows exactly the law of reflection), however their normals may deviate from the surface normal of the approximating perfectly smooth surface of the object. The more the surface normals of the microfacets deviate from the approximating surface normal (e.g. the average over the orientations of the microfacets), the *rougher* the specular reflection becomes and the blurrier the specular highlight is. This is illustrated

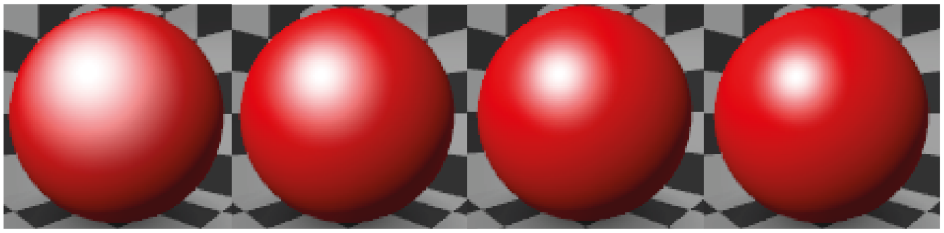

Figure 1.3: Spheres with different specular roughness parameters.
From l.t.r: roughness: 10, roughness: 20, roughness: 30 and roughness: 40.
Here the Cook-Torrence algorithm was used in order to calculate the specular highlight.

in Fig. 1.3 which shows the specular highlights of spheres created with blender. How much the microfacets deviate from the approximating surface normal, and therefore how "rough "the resulting specular highlight is can be changed via a parameter in the Cook-Torrence algorithm that was used in order to calculate the reflectance. Objects that have a larger diffusive reflecting component and a smaller (to none) specular component are often called *matte* and objects that have a large specular lobe are often called *glossy*. *Glossy* materials may also have a diffuse component. This can (to some extend) be contributed to *subsurface scattering*. Not all light rays are reflected at the very surface of an object but some light rays rather penetrate the outer surface of the object and are then reflected internally in an irregular manner until they again exit at a seemingly random point of the surface. L. Wolff has explored exactly this behaviour for dielectric materials in more detail in his paper [50]. In computer graphics subsurface scattering is used to more realistically describe slightly translucent materials, for example skin or wax.

How the light is reflected on the surface of an object, including the diffuse as well as the specular component is described by the *Bidirectional Reflectance Distribution Function* (BRDF) (as described by Nicodemus in 1992 [32]). In Fig. 1.4 one can see the Lambertian sphere (or Lambertian reflectance) in violet, which shows exactly all possible diffuse components regarding the incident angle ω_i. The intersection of the light ray with the Lambertian sphere gives the radius of the actual diffuse component with respect to ω_i (in the figure given by the blue sphere). The glossy component (in the figure given as the blue sphere) is dependent on the incident angle ω_i and the observer angle ω_r with regard to the surface normal and where the light is reflected with the highest intensity, with the maximum being the specular component (in the figure shown as a red sphere), which is exactly as aformentioned where the surface normal is halfway between ω_i and ω_r.

1.2 Albedo

The word albedo originates from the latin word for *white* and was introduced as a term by Lambert in [26]. It is a scalar value in range $a = [0, 1]$ which defines how much of the

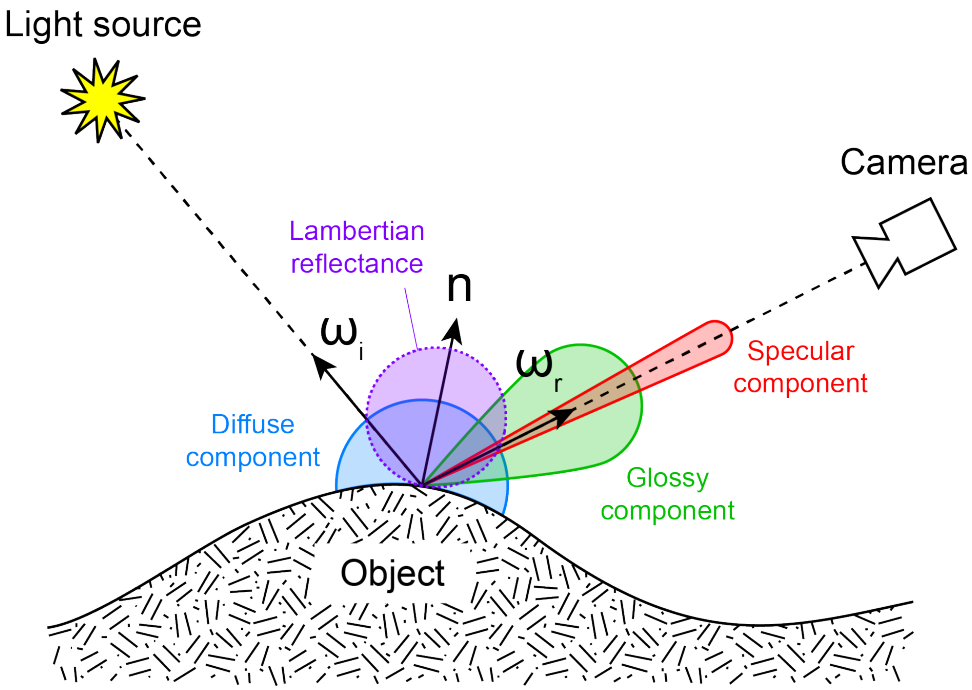

Figure 1.4: Schematic illustration of the Bidirectional reflectance distribution function (BRDF)

Figure 1.5: from l.t.r: the albedo image, the shading image and the resulting color image. Images courtesy of [29]

incoming light of a diffuse object (i.e. a non-emitting object) is reflected back. If an object has the albedo of 0 no light is emitted back, which means it appears completely black. With the albedo of 1 all of the light is emitted back so it will appear completely white. For example charcoal has the albedo of 0.05, which is why it appears almost completely black while snow appears very white with the albedo of around $0.8 - 0.9$. Another way to define this term is based on how much of the incident light is absorbed by the object. This fact is important for many applications, such as astronomy or environmental science.

In computer science, or more precise in computer graphics, albedo is used in order to calculate the color image, so that:

$$I = aS. \tag{1.2}$$

Here, a stands for the albedo, which in this example is an RGB image, and S stands for the shading. By point-wise multiplying these two parts one gets the final color image. The images in Fig. 1.5 are taken from the paper from T. Narihira et al. [29] and show each of the components, namely albedo, shading and albedo mutliplied with shading, individually.

In their paper T. Narihira et al. took the MPI Sintel database [6] in order to learn the albedo and shading pass from the color image using CNN.

1.3 Photometric Stereo

The previous sections have shown that the reflectance of light behaves in a very predictable manner if the material is known and that we can predict pretty well how light will reflect using knowledge from photometry and physics. It was established that the important factors for predicting how small or big the observed intensity is, are the angle of the incident light ω_i, the orientation of the surface normal $\mathbf{n}$ of an object and the angle of observation ω_r. With the assumption of a perfectly matt surface with constant albedo, the reflection of an object follows exactly Lambert's cosine law [26] as shown in Eq. 1.1. This law has been exploited for a well studied and difficult problem in computer vision, the *3D reconstruction* of an object, more precisely the surface normals estimation, using only 2D information (images). This approach is called *photometric stereo* and was first introduced by R.J. Woodham in 1980 [51]. In his work he stated that, while holding the observation angle constant one would need varying incident illumination angles of point-like light sources, at which the object is observed. These incident angles and light intensities need to be known a priori. Since the object is stationary, corresponding pixels are on the same position for each acquisition. After all acquisitions are done, one has a vector of observed intensities for every image point, as well as the corresponding incident illumination vectors. With this information Woodham formulated the equation:

$$\mathbf{e} = \mathbf{L}\mathbf{n}a \tag{1.3}$$

where $\mathbf{e} = (e_1 \dots e_n)^T$ is a vector of observed intensities, $\mathbf{L}$ is a matrix describing the light vectors and $\mathbf{n}$ denotes the surface normal $\mathbf{n} = (N_x, N_y, N_z)^T$. Symbol a denotes the albedo which is a scalar value in range $a \in [0, 1]$.

For one pixel for example this equation would look as follows:

$$\begin{bmatrix} l_x^1 & l_y^1 & l_z^1 \\ l_x^2 & \dots & \\ \vdots & \ddots & \\ l_x^n & \dots & \end{bmatrix}_{N\times 3} \begin{pmatrix} N_x \\ N_y \\ N_z \end{pmatrix}_{3\times 1} a = \begin{pmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{pmatrix}_{N\times 1}, \tag{1.4}$$

where N is the number of the light directions. Another way of writing this is the system of linear equations:

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{1.5}$$

If N = 3 and A is not singular, this equation has an exact solution given by:

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b} \tag{1.6}$$

or in this case

$$\mathbf{n}a = \mathbf{L^{-1}e} \tag{1.7}$$

However if one has more than three light directions ($N > 3$) then this would lead to an overdetermined system with no exact solution. However the final solution is usually more exact if one uses more light directions than just three, especially when the measurements are noisy.

As one can not find the exact solution, the goal is to find a best possible solution. This can be formulated as a least squares problem of the following form:
Find an $\mathbf{x}$ (the unknowns), such that $||\mathbf{Ax} - \mathbf{b}||^2$ is *minimized* (close to zero), or in this case $||\mathbf{Ln}a - \mathbf{e}||^2$. This is called the *linear least square method*.

$$\min_n \frac{1}{2} ||\mathbf{Ln}a - \mathbf{e}||^2 \tag{1.8}$$

Lets substitute: $\mathbf{n}a$ for $\tilde{\mathbf{n}}$, then to solve for a:

$$a = ||\tilde{\mathbf{n}}|| \tag{1.9}$$

$$\mathbf{n} = \frac{\tilde{\mathbf{n}}}{a} \tag{1.10}$$

This can be re-written as:

$$\begin{aligned} \mathbf{L}^T(\mathbf{L}\tilde{\mathbf{n}} - \mathbf{e}) &= 0 \\ \mathbf{L}^T\mathbf{L}\tilde{\mathbf{n}} - \mathbf{L}^T\mathbf{e} &= 0 \\ \mathbf{L}^T\mathbf{L}\tilde{\mathbf{n}} &= \mathbf{L}^T\mathbf{e} \end{aligned} \tag{1.11}$$

Now to get rid of $\mathbf{L}^T\mathbf{L}$ on the left side in order to get the solution for $\mathbf{n}$, multiply with $(\mathbf{L}^T\mathbf{L})^{-1}$ from the left side (provided the inverse exists) which then leads to:

$$\tilde{\mathbf{n}} = (\mathbf{L}^T\mathbf{L})^{-1}\mathbf{L}^T\mathbf{e} \tag{1.12}$$

and $\mathbf{n} = \frac{\tilde{\mathbf{n}}}{||\tilde{\mathbf{n}}||}$.

Here $(\mathbf{L}^T\mathbf{L})^{-1}\mathbf{L}^T$ is the well-known Moore-Penrose Pseudo-Inverse that gives the solution in the least square sense. From now on $(\mathbf{L}^T\mathbf{L})^{-1}\mathbf{L}^T$ will be abbreviated with $\mathbf{L}^+$.

1.4 Light Field

Light field describes the amount of light flowing from *every possible direction* through *every possible point* in the 3D space. The terminology light field was first described by A. Gershun in 1936 [13]. In a general setting light field can be described as a *5D*

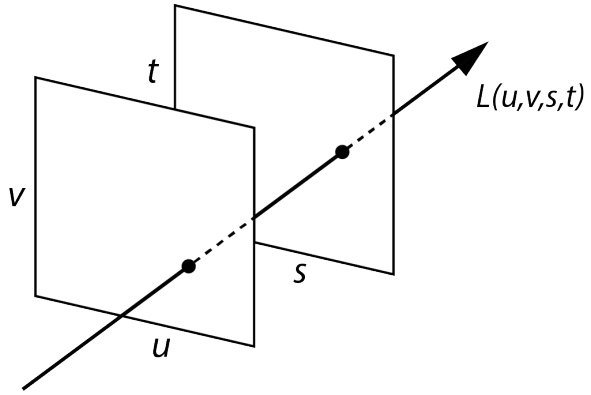

Figure 1.6: Schematic illustration of the two plane rasterization.

plenoptic function $L(x, y, z, \theta, \Phi)$. In this function x, y, z denotes the parametrization of the spatial dimension, which is the 3D space coordinates of a point and θ, Φ denotes the parametrization of the angular dimension, describing the angle at which an incoming light ray hits a point. This formulation allows to theoretically describe a scene (or an image of the scene) from every possible observer location with every possible angle.

Since the radiance of a light ray is constant along the same direction, if the ray is not blocked in free space, this general formulation can be simplified to a 4D function by getting rid of the z parameter of the spatial parametrization. This formulation of light field can be visualized via a *two plane parametrization*. In this formulation each point is described by a ray intersecting two planes in general position (see Fig. 1.6). In literature the spatial coordinates of these two planes used in this parametrization are often given as u, v and s, t, so every point is uniquely represented by a quadruple $L(u, v, s, t)$.

The u, v plane parameterizes the observer angle while the position of the intersected point on the s, t plane parametrizes the observed object point with this perspective. An easier way to interpret this is as a 2D collection of 2D images, each image taken from a different observer angle. An example of such a collection of images can be seen in Fig. 1.7.

Light-field processing via light-field cameras can be seen as an add-on to the general stereo idea. To capture a light-field with a camera, a number of different approaches exist, for instance commercially available plenoptic cameras such as Lytro [31] or multi-camera arrays [49].

1.5 Machine Learning

To find patterns or regularities in data and describe them is a well studied but non-trivial problem. Seemingly easy tasks for humans, such as e.g. digits recognition or speech

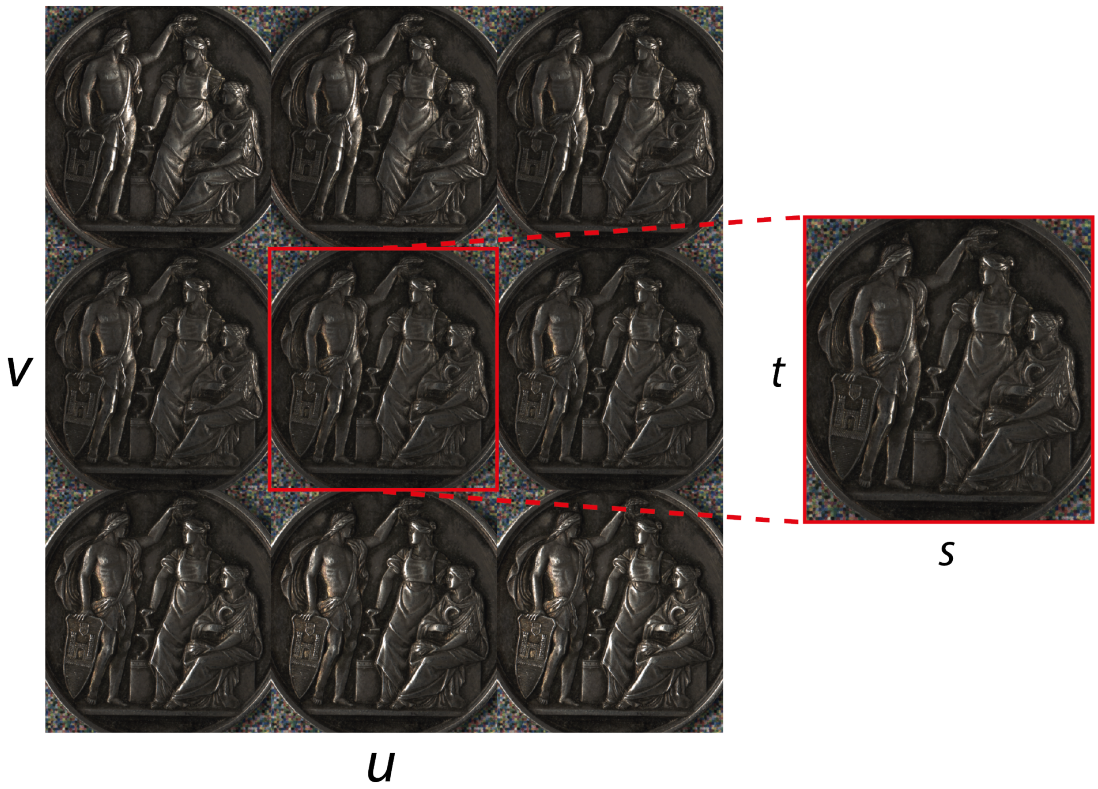

Figure 1.7: Example of a 4D light field illustration as a 2D collection of 2D images.

recognition pose a problem for machines. The need for the construction of (preferably autonomous) algorithms or procedures to solve such tasks gave birth to machine learning. Over the years many such algorithm arose with different pros and cons , some tailored to solve very specific problems. The superset of these algorithms can be divided into two (not necessarily distinct) subsets, namely *supervised* and *unsupervised*. In *supervised* machine learning, an algorithm is presented with a set of *labeled* data. The label of a data point is the desired value the procedure should output when presented with the values of the corresponding information of the data point. This *information* is called the *features* of the object. These *features* can be seen as the attributes an object has and all features together build the feature space. For example if your task is to decide whether an given object belongs to a specific species of fish, such attributes or features could be, along with others, the length of the fish or the color of the abdomen. It is important to note, that if one wants to deploy such a learning algorithm and is in the data generation/acquisition step, one should keep in mind the feature scaling as attributes tend to have different units (for example kg, cm, candela etc.) and can be in completely different ranges. This poses a problem to some machine learning algorithms and should be taken into consideration. For this, one could either rescale the features to be in range $[0, 1]$ or $[-1, 1]$ or scale the whole

feature vector to unit length ($\hat{\mathbf{x}} = \frac{\mathbf{x}}{||\mathbf{x}||}$). Another way to describe features is in binary form, i.e. is it present (1) or not (0) in a specific data point. Individual data samples are typically denoted as $\mathbf{x} = (x_1, \ldots, x_d)^T$ where d is the number of features and $x_i \in \mathbb{R}$ denotes the value of one specific feature. The label of the data sample is typically written as y, where $y \in \mathbb{R}$ (in literature sometimes also referred to as t which stands for target vector). Often the datapoint and the label are denoted as a tuple $(\mathbf{x}, y)$.

In *supervised* learning a subset of the data set of such tuples $S = \{(\mathbf{x}^1, y^1), \ldots, (\mathbf{x}^N, y^N)\}$ are presented to the algorithm in order to adapt the system to the task and improve it's error. Such a subset is called a *trainings set* and should be chosen just large enough for the learning algorithm to solve the inference task. It is important that only a subset of the data is chosen for adaption and improvement of a machine learning algorithm as one wants to have some data, previously unseen by the algorithm in order to test it's generalization capabilities. Learning by heart is a problem in machine learning and should be avoided. This *learning by heart* in this context is known as *overfitting*. In order to test for overfitting another distinct (previously unseen) subset is chosen from the data set and is presented to the algorithm after every iteration. As one wants to test the actual accuracy of the predictor that has been learned, this subset is called the *testing set*. The intersection between the training and testing set has to be empty, because one only wants to have data points in the testing set that weren't used for it's optimization. Usually the error of the predicted value, or the output, of the machine learning algorithm to the target value should converge toward zero. At some point however the algorithm may overfit, which means the error on the testing set increases while the error on the trainings set still converges. At this point one may choose to stop the training process.

In *unsupervised* learning no target values, or labels, are necessary. However this also means that there is no possibility of error or accuracy measurement of these algorithms as information about the expected values are missing. This type of machine learning focuses more on finding structure in the data via density or correlation in the point cloud of the feature space.

Some examples for *supervised* learning are neural networks, support vector machines or decision trees.

Some examples for *unsupervised* learning are principal component analysis (PCA), k-means or Gaussian mixture model.

Another kind of stochastic optimization technique that should be mentioned at this point is called *Evolutionary Algorithms* (EA). As the name suggests, these kind of algorithms are inspired by the biological evolution. Generally speaking, the EA always deals with how to improve the *fitness* of a population over generations. In order to do so it uses concepts like: *(sexual or asexual) reproduction* and *recombination* of " genes " , *random mutation* and *selection*. Usually, the *fittest* individuals in a generation pass on their genes in some fashion, while the individuals with the worst fitness die out.

This kind of optimization technique is not heavily used in the field of computer vision,

however it yielded some impressive results in biology and bio-robotics [7][10] as well as medicine [14].

1.6 Learning Problem

Section 1.5 gave a broad overview about what machine learning does and gave some examples about the different kinds of machine learning algorithms. However it was not explained when or why this approach is needed.

In general such a method is needed if the underlying function or set of operations in order to map an input to its desired output is unknown. If the correlation of the example from Section 1.5 between the length of a fish and the color of the abdomen to the species of the fish is well known there would be no need for it to be solved using machine learning. One could just use the known operations and input the features of one sample in order to get the desired output. If such a set of operations is known and generally accepted the problem is said to be *closed form*.

Machine learning deals with problems that do not have such a known set of operations to get from a given input to the desired output. The learning problem is to find such a mapping function. This can be written as, for any given $\mathbf{x}$:

$$f(\mathbf{x}) = y. \tag{1.13}$$

However most of the time it will not be possible to find such a perfect general mapping function and all one can hope for is to find a function that is the best possible approximation of this unknown underlying function f.

Machine learning can be seen as the search for a function that approximates the underlying *true* function the best (in the sense of generality and precision) using the given knowledge, namely the data (or more precise the feature distribution of the data) and the target value, or label.

A common source of error is that data tends to be noisy (for instance through some error or inaccuracy during the data acquisition) which can lead to wrong or biased solutions. Also the amount of data one has for a given problem and which features were acquired can pose problems for the function search. If one does not have enough data to find the underlying correlation between the features and the output or they are simply uncorrelated, trying to find a best approximated mapping function will be futile or will at best give bad results.

The kind of machine learning algorithm that is chosen for a specific problem restricts the search space of all possible mapping functions, e.g. by only allowing linear or non-linear models and defines how this smaller finite space is navigated.

The *linear support vector machine* (linear SVM) for instance tries to find the best line (or hyperplane) that tries to seperate two classes in feature space. Here the limitation in the search space would be that the mapping function can only be linear. The linear

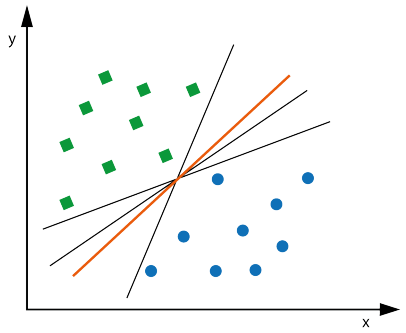

Figure 1.8: Possible solutions for f found by the SVM algorithm (black lines) with the chosen best solution (red line) which is the best possible found approximation of f

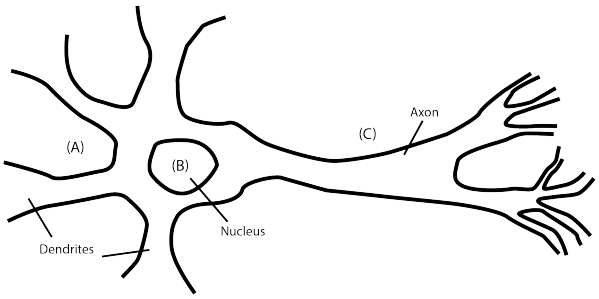

Figure 1.9: Schematic presentation of a biological neuron

SVM tries to optimize the amount of samples that are classified right but also the margin between the seperating line (or hyperplane) in respect to the nearest samples on each side (the so called *max-margin condition*, see Fig. 1.8). This for instance would dictate how the possible mapping space is navigated and searched until one final and (hopefully) best approximation for f is found.

1.7 Neural Networks

The idea of neural networks, more precisely artificial neural networks, as a machine learning algorithm has a long and successful history. Over the years this approach has gone through many iterations and had many "*renaissances*" as it went in and out of fashion in the scientific community. The concept of these algorithms is inspired by how biological neurons form connections in the brain and transmit data, which is how they got their name. Fig. 1.9 shows an illustration of a biological neuron and its parts.

However it is important to note, that the goal of artificial neural networks (ANN) is not to mimic the behavior of the biological brain, it only loosely borrows some aspects of the biological brain in order to do inference on the presented data. A lot of research is

done in the field of brain computing (i.e. simulating the spiking of the biological neurons in a machine) and the acquired knowledge often inspires new movements in the ANN field. However this will not be discussed here in greater detail.

The term artificial neural network is a broad description which includes a wide variety of different approaches. For example there exists a variety of recurrent neural network (RNN) techniques, which allows loops in the graph or even fully connected graphs, such as the Boltzmann machine [19] and many more. In this thesis we will only deal with so called *artificial hierarchical feed-forward* neural networks. For historical reasons these networks are sometimes also called *multi-layer perceptrons* in literature and hereinafter, this two definitions will be used interchangeably. However it is important to note that this has, as mentioned before, more historical reasons and is not really an accurate description for this architecture. Perceptrons could be seen as the grandfathers of the modern neural networks and where already introduced in the late 1950s [38]. However the big difference to the architecture nowadays is that it was a linear classifier, or more precise, it used a linear activation function.

A *artificial hierarchical feed-forward neural networks* is a directed graph without loops, which means that once information has passed through a certain point, or node, it cannot reach that node again. Furthermore a feed-forward neural network is ordered hierarchically into layers where a node in the graph only has incoming edges from the previous layer and outgoing edges to only the next layer. If it is *fully connected*, every node of a layer l_i is connected to all nodes in the next layer l_{i+1} (with no connection between nodes within one layer). The layers of a feed-forward neural network are typically ordered into three categories, namely the *input layer*, where the information starts to propagate through the network, one or more *hidden layers* and the *output layer*, where the prediction of the network can be read out (Note: For a n-layered structure one usually just counts the number of hidden layers, as input and output layer are a given by the application).

Each node in this schematic presentation of a neural network as seen in Fig. 1.10 is a *perceptron* which together with the weights build an artificial neural network. However each node in the hidden layer(s) and the output layer is such a perceptron and is abstracted as a circle for the sake of readability. The circles in the input layer are not perceptrons but rather just a visualization on where and how the data begins to propagate through the network. However this kind of schematic has been popularized. The internal structure of each perceptron is shown in Fig. 1.11.

Each perceptron consists of some incoming weights that are multiplied with the values of an input vector that is send through the network. This is comparable with the Dendrites (A) of a biological neuron (see Fig. 1.9). These values are then summed up and a bias term is added. Formally this can be expressed as follows:

$$u \;=\; b + \sum_{i=1}^{d} x_i w_i \tag{1.14}$$

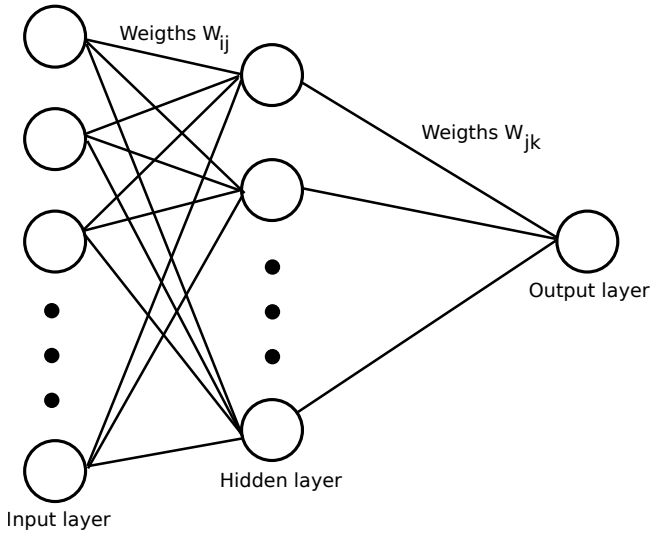

Figure 1.10: Schematic presentation of an artificial neural network with one hidden layer

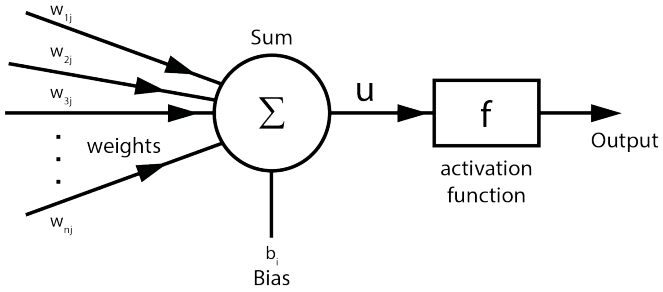

Figure 1.11: Schematic presentation of an artificial neuron

This summed up term u is then send into the activation function (sometimes also referred to as transfer function). The role of this activation function is to bring *non-linearity* into the decision boundary of a network. This is needed because the weight coefficients and bias are a combination of linear models, which again accounts to a linear model. Historically so called *sigmoid neurons* were mostly used, which employed the sigmoidal activation function (Eq. (1.17)). The sigmoid activation function has the nice property that for any given input it will output something that is between 0 and 1. Furthermore, if a input value is large positive, then the output of the function is ≈ 1 which has a nice correlation of the firing of a biological neuron. This value $f(u)$ is then send to a neuron (or neurons) in the next layer, which is comparable to the Axons (C) of the biological neuron (see Fig. 1.9). Here some of the most commonly used activation functions nowadays are listed below (see

also Fig. 1.12)

$$Linear : f(u) = u \tag{1.15}$$

$$RELU : f(u) = \max(0, u) \tag{1.16}$$

$$Sigmoid : f(u) = \frac{1}{1 + e^{-u}} \tag{1.17}$$

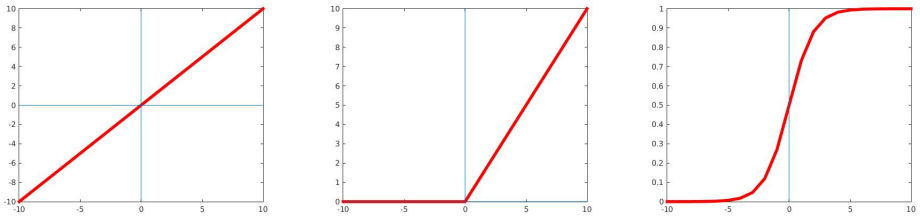

Figure 1.12: Characteristic plots of different activation functions. From left to right: linear, relu and sigmoid activation function

The weights and bias' are the adaptable part (i.e. the part that is learned) of an artificial neural network. After a random initialization of these values (or using some better heuristic), these are the parameters that are changed iteratively in the learning step, usually via error back propagation algorithm (see Section 1.9).

1.8 Gradient Descent

Sec. 1.5 gave a braod overview about machine learning algorithms and Sec. 1.6 gave a small introduction when this algorithms are needed. Many of these machine learning algorithm iteratively adapt their parameters in order to converge to an optimal solution. However, in order to perform this "adaptation" a definition of quality measurement or improvement is needed. In *supervised* machine learning this measure of quality is calculated via a *cost function* (sometimes also referred to as *loss function* in literature). There are many of these *cost functions* with different advantages and disadvantages, like *mean-square error*, *mean-absolute error*, *information gain* or *0-1 loss*. These loss functions calculate an error-measurement E of the prediction with respect to its target label. All these loss functions have in common, that they take the prediction and compare it with the label in some sort in order to quantify how much the prediction still deviates from the expected value. Since the task of a machine learning algorithm is to improve over time, now some *optimization method* is needed. Intuitively one can say that a learning algorithms performance improves, if the overall error of the prediction becomes smaller (or inversely formulated the accuracy increases). In any way, in order to improve the procedure, one needs to do some sort of *optimization algorithm* that either finds a minimum or maximum of a defined cost

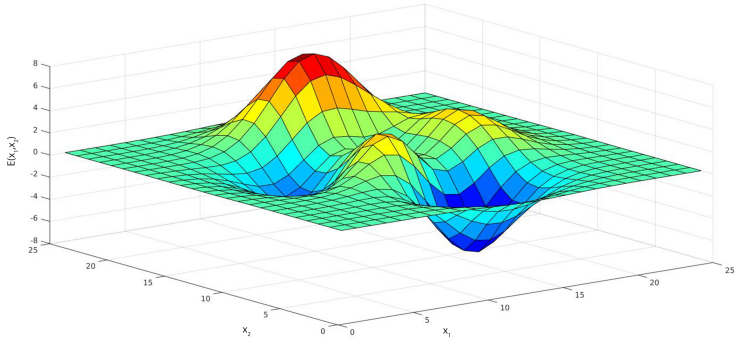

Figure 1.13: Example visualization of the surface of a cost function with two variables

function, depending on how the problem is formulated. An two dimensional example of how the surface of such a cost function may look like can be seen in Fig. 1.13.

One of the most used optimization algorithms is called *Gradient Descent* (GD). There are a number of variations and improvements of this idea (like Gradient Descent with momentum), however they all follow the general principle of the basic Gradient Descent which will be described in this chapter.

Lets think about the error as some function that is dependent on a set of variables $E(\mathbf{x})$, where $\mathbf{x}$ can be a set of arbitrary many real valued numbers, written as a vector (often referred to as *hyperparameter*). For the sake of simplicity, lets assume that $\mathbf{x}$ consists of two independent variables, x_1 and x_2 so that $\mathbf{x} = (x_1, x_2)$. So for every possible value combination of x_1 and x_2, $E(\cdot)$ will give back some real value. In this simple case the result of this is easy to visualize, as it results in a 3D surface. However one should keep in mind that usually one deals with much higher dimensional, more complex problems but for the sake of simplicity we will continue with this example.

What one wants is to find is the global minimum of $E(x_1, x_2)$. As the name would suggest, in order to iteratively converge towards a minimum, gradient descent uses local gradients. Gradient vectors have the nice property to be oriented into the direction of the largest increase of the function value, i.e. in direction of the largest incline of $E(\cdot)$. So if one wants to find the minimum of the function, one could do a "step" into the negative direction of the gradient vector direction, thus the name *Gradient Descent*. The gradient vector is defined by the partial derivates of $E(\cdot)$ with respect to x_1 and x_2 as follows:

$$\nabla E = \left(\frac{\partial E}{\partial x_1}, \frac{\partial E}{\partial x_2} \right) \tag{1.18}$$

Now the update rule, i.e. one step of the algorithm shouldn't come as a suprise. After a random initialization of the random variables (here x_1 and x_2):

$$\begin{aligned} x_{1_{new}} &:= x_{1_{old}} - \eta \, \nabla E(x_{1_{old}}, x_{2_{old}}) \\ x_{2_{new}} &:= x_{2_{old}} - \eta \, \nabla E(x_{1_{old}}, x_{2_{old}}) \end{aligned} \tag{1.19}$$

It is important that the updates of the variables have to be done simultaneously. In Eq. 1.19, η is the step size, i.e. how much one should go into the direction of the negative gradients (sometimes also referred to as learning rate). This can be an important parameter, as it can cause the algorithm to diverge if it is chosen too large or make it very slow if chosen too small. The step size also doesn't have to be fixed but can be changed during the run time of the algorithm. This is called *Adaptive Gradient Descent*.
The extension to a higher dimensional space is straight forward. Rather then having the optimization problem of the previous simple case:

$$\min_{x_1, x_2} \; E(x_1, x_2) \tag{1.20}$$

one can generalize it to:

$$\min_{x_1, x_2, \ldots x_n} \; E(x_1, x_2, \ldots x_N). \tag{1.21}$$

The updates of these variables are done accordingly.
There are many different stopping criteria for this algorithm that can be chosen. The easiest is to just set a hard limit of exactly how much iterations are done. One could also threshold if the value of the cost function is beneath a certain range close to 0. Another method for the stopping criteria could be if the improvement after each iteration has been very small for a set amount of iterations (e.g. the improvement of the function has been < 0.01 for the last 10 iterations) or a combination of these stopping criteria.

However, this optimization approach has some disadvantages. For instance the initial values of the variables are very important as the algorithm can end up in a local minimum by chance. So one can never be sure if the algorithm ended up in the global minimum or just a local one. Furthermore as the gradient of the cost function ∇E is calculated using all the training samples in the standard case, each iteration can be very computationally expensive and thus slow. An improvement in this regard is the *Stochastic Gradient Descent*. Rather then using all the data, one training data point is chosen randomly in each iteration, or differently said each partial derivative ∇E is computed only using one random sample of the trainings set. This approach is a lot faster than standard *Gradient Descent*, however it also has another nice property. *Stochastic Gradient Descent* introduces randomness into the algorithm, which can be advantageous regarding getting stuck in local minimum. Since the stochastic gradient descent is not deterministic and can "wander" in different direction, even increasing the error in one iteration, it is more likely to find the global minimum.

The implementation of gradient descent in artificial neural networks is straight forward, as one can simply use the partial derivative of the error (or cost) function E in respect to

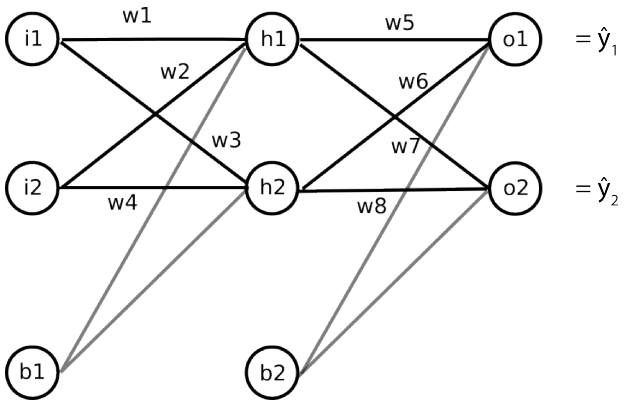

Figure 1.14: Visualization of the small dummy network used in this example

the weights (or bias).

1.9 Back Propagation

In Sec. 1.7 it was explained how a neural network looks like and how information propagates through it and how a prediction is made. It was also mentioned that the weights and the bias of the connections are the adaptable parameters. In Sec. 1.8 it was shown how a cost function is used as quality measurement for the prediction and how local gradients can be used in order to minimize the error. In this chapter the way of how this information is used in order to change the weights and bias, according to how much they have contributed to the overall error is shown.

This procedure is called *Back Propagation* (often shortened to just *backprop*). In literature this is often referred to as the *backwards pass* phase of the neural network. In the first phase, the *forward pass*, information is send through the network, a prediction is made and at the end an error measure is calculated using this prediction. In the *backward pass*, the partial derivative of the overall error in respect to the weight that should be changed is taken and propagated recursively through the network starting at the end of it. At the end of this, the weights and bias have been updated accordingly. In the most general sense, one forward and backward pass can be seen as one iteration of the neural network learning algorithm.

In order to illustrate how this *backward pass* works let's look at a short example. The notation and structure of this example was taken from an blog post from M. Mazur[28].

Let's assume a small neural network with two hidden neurons, two output neurons and sigmoid activation function. The data points have two real-valued features, thus $\mathbf{x} = (x_1, x_2)$ and $x_1, x_2 \in \mathbb{R}$. Therefore two input neurons are used. This example network can be seen in Fig. 1.14.

Let's define how the input and output looks like for the hidden and output neurons. We will use this information later when we derive the update rule for the weights. Following

Eq. 1.14 for *h1* and *h2* this would yield:

$$\begin{aligned} net_{h1} &= w_1 i_1 + w_2 i_2 + b_1 \\ net_{h2} &= w_3 i_1 + w_4 i_2 + b_1 \end{aligned} \tag{1.22}$$

This is called the *net input* for the hidden neurons. The output of this neuron is then the activation function used on the net input as seen in Fig. 1.11, $f(net_{hi})$ (here $f(\cdot)$ is the sigmoid function as described in 1.17). So the outputs of the two hidden neurons would be:

$$\begin{aligned} out_{h1} &= \frac{1}{1 + e^{-net_{h_1}}} \\ out_{h2} &= \frac{1}{1 + e^{-net_{h_2}}} \end{aligned} \tag{1.23}$$

This is what would then be passed to the next layer, here the output layer. Again following Eq. 1.14 for these would yield:

$$\begin{aligned} net_{o1} &= w_5 out_{h1} + w_6 out_{h2} + b_2 \\ net_{o2} &= w_7 out_{h1} + w_8 out_{h2} + b_2 \end{aligned} \tag{1.24}$$

and:

$$\begin{aligned} \hat{y}_1 &= \frac{1}{1 + e^{-net_{o_1}}} \\ \hat{y}_2 &= \frac{1}{1 + e^{-net_{o_2}}} \end{aligned} \tag{1.25}$$

The initialization of the bias and weights is usually done in some random fashion, as it is not know what good values for these are in the beginning, for example randomly normally or uniformly distributed. Let's assume that such a random initialization would be:

$$\begin{array}{ll} w_1 = 0.9106 & w_2 = 0.1818 \\ w_3 = 0.2638 & w_4 = 0.1455 \\ w_5 = 0.1361 & w_6 = 0.8693 \\ w_7 = 0.5795 & w_8 = 0.5499 \\ b_1 = 0.1450 & b_2 = 0.8530 \end{array}$$

Let's further assume that the features of one training sample are: $x_1 = 0.01$ and $x_2 = 0.10$ and the label (or targets) of this sample are: $t_1 = 0.01$ and $t_2 = 0.99$ (for output $\hat{y}_1$ and $\hat{y}_2$ respectively). Using the previous formulated equations one would then get: $net_{h_1} = 0.20871$, $net_{h_2} = 0.17274$ and $out_{h1} = 0.5519$, $out_{h2} = 0.5431$. The output layer would be: $net_{o_1} = 1.3948$, $net_{o_2} = 1.4483$ and $\hat{y}_1 = 0.8014$, $\hat{y}_2 = 0.8097$.

For convenience sake let's continue with a square error function as follows:

$$E = \sum_{i=1}^{N} \frac{1}{2}(\hat{y}_i - y_i)^2 \tag{1.26}$$

Note: The factor $\frac{1}{2}$ is such that the exponent of the derivative (times 2) is canceled which will be used later on. Since it is only a constant factor this can be done. Using this square error function we will get a total error of: $E = 0.32941$.

This was the forward pass of the neural network. For the backwards pass now we will start with the weight of w_5. What one wants to know is how much the value of this weight took part in the total error. In order to get this we have to calculate the partial derivative of the error E in respect to the weight w_5, like:

$$\frac{\partial E}{\partial w_5} \tag{1.27}$$

Using the chain rule this derivative can be written as:

$$\frac{\partial E}{\partial w_5} = \frac{\partial net_{o1}}{\partial w_5} \frac{\partial out_{o1}}{\partial net_{o1}} \frac{\partial E}{\partial out_{o1}} \tag{1.28}$$

First let's look at the partial derivative of the standard logistic function, which has the known form of:

$$f(x) = \frac{1}{1 + e^{-x}} = 1 \cdot (1 + e^{-x})^{-1} \tag{1.29}$$

and the first derivative:

$$\begin{aligned} f'(x) &= 0 \cdot (1 + e^{-x})^{-1} + 1 \cdot (-1) \cdot (1 + e^{-x})^{-2} e^{-x} \cdot (-1) \\ &= 1 \cdot (1 + e^{-x})^{-2} e^{-x} \\ &= \frac{e^{-x}}{(1 + e^{-x})^2} \\ &= \underbrace{\frac{1}{1 + e^{-x}}}_{\text{f(x)}} \underbrace{(1 - \frac{1}{1 + e^{-x}})}_{\text{1-f(x)}} \end{aligned} \tag{1.30}$$

In this case this would lead to:

$$\frac{\partial \hat{y}_1}{\partial net_{o1}} = \hat{y}_1(1 - \hat{y}_1) \tag{1.31}$$

Next let's take a look at the partial derivative of the error in respect to the output neuron $o1$ which is $\hat{y}_1$. As in this example the summation of the error only consists of two

elements, it is easy to write the whole notation:

$$E = \frac{1}{2} \cdot (\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 \tag{1.32}$$

As we want the derivative in respect to $\hat{y}_1$, the second part of the equation $(\hat{y}_2 - y_2)^2$ becomes 0 in the derivative and we end up with:

$$\frac{\partial E}{\partial net_{o1}} = 2 \cdot \frac{1}{2} \cdot (\hat{y}_1 - y_1) \cdot -1 = -(\hat{y}_1 - y_1) \tag{1.33}$$

Now the last part is how the input of the neuron $o1$ changed in respect to the weight w_5.

$$\begin{aligned} net_{o1} &= w_5 out_{h1} + w_6 out_{h2} + b_2 \\ \frac{\partial net_{o1}}{\partial w_5} &= 1 \cdot out_{h1} \cdot 1 + 0 + 0 \\ &= out_{h1} \end{aligned} \tag{1.34}$$

By combining these three partial derivatives and some rewriting and shortening of the equation one ends up with the general formula of:

$$\frac{\partial E}{\partial w_5} = -(\hat{y}_1 - y_1)\hat{y}_1(1 - \hat{y}_1)out_{h1} \tag{1.35}$$

This gives a total of:

$$\frac{\partial E}{\partial w_5} = 0.06954 \tag{1.36}$$

Now let's assume that for the update we use gradient descent weight update such as described in Eq. 1.19 with a learning rate of $\eta = 0.2$, so that:

$$\begin{aligned} w_{5new} &= w_{5old} - \eta \frac{\partial E}{\partial w_5} \\ w_{5new} &= 0.1361 - 0.2 \cdot 0.0695 = 0.1222 \end{aligned} \tag{1.37}$$

This is then the updated weight for w_5 after the first backwards pass. The same can be done for the weights of the hidden layer, however one has to slightly extend the formulation as the weights of the hidden layer contributes to the error of more neurons (exactly the two output neurons). In case of w_5 it just contributed to o_1. If one wants to update for example w_1, again one would come up with the chain rule for the partial derivative of:

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial out_{h_1}} \frac{\partial out_{h_1}}{\partial net_{h_1}} \frac{\partial net_{h_1}}{\partial w_1} \tag{1.38}$$

Now $\frac{\partial E}{\partial out_{h_1}}$ is the partial derivative of the total error in respect to the first hidden neuron. As this layer propagates it's information to both $o1$ and $o2$, this can be further

written as:

$$\frac{\partial E}{\partial out_{h_1}} = \frac{\partial E_{o1}}{\partial out_{h_1}} + \frac{\partial E_{o2}}{\partial out_{h_1}} \tag{1.39}$$

Then one can further write:

$$\frac{\partial E_{o1}}{\partial out_{h_1}} = \frac{\partial E_{o1}}{\partial net_{h_1}} \frac{\partial net_{o1}}{\partial out_{h_1}} \tag{1.40}$$

Furthermore:

$$\frac{\partial E_{o1}}{\partial net_{o1}} = \frac{\partial E_{o1}}{\partial out_{o1}} \frac{\partial out_{o1}}{\partial net_{o1}} = 0.7914 \cdot 0.159 = 0.1260 \tag{1.41}$$

We know that $\frac{\partial net_{o1}}{\partial out_{h_1}} = w_5$, so $\frac{\partial E_{o1}}{\partial out_{h_1}} = 0.1260 \cdot 0.1361 = 0.0172$
Let's continue with:

$$\begin{aligned}
\frac{\partial E_{o2}}{\partial out_{h_1}} &= \frac{\partial E_{o2}}{\partial net_{o1}} \frac{\partial net_{o1}}{\partial out_{h_1}} \\
\frac{\partial E_{o2}}{\partial net_{h_1}} &= \frac{\partial E_{o2}}{\partial \hat{y}_2} \frac{\partial \hat{y}_1}{\partial net_{o1}} \\
\frac{\partial E_{o2}}{\partial \hat{y}_2} &= -(0.99 - 0.8097) = -0.1803 \\
\frac{\partial E_{o2}}{\partial net_{h_1}} &= -0.1803 \cdot 0.1592 = -0.0287 \\
\frac{\partial E_{o2}}{\partial out_{h_1}} &= -0.02870376 \cdot 0.1316 = -0.0038
\end{aligned} \tag{1.42}$$

So finally:

$$\frac{\partial E}{\partial out_{h_1}} = 0.0172 + -0.0038 = 0.0134 \tag{1.43}$$

Then:

$$\frac{\partial out_{h_1}}{\partial net_{h_1}} = out_{h_1}(1 - out_{h_1}) = 0.2473 \tag{1.44}$$

and

$$\frac{\partial net_{h_1}}{\partial w_1} = i1 = 0.05 \tag{1.45}$$

So one would end up with: $0.0134 \cdot 0.2473 \cdot 0.05 = 0.0002$
Then using the update rule:

$$w_{1_{new}} = 0.9106 - 0.2 \cdot 0.0002 = 0.9106 \tag{1.46}$$

As one can see here the weight has not changed much, which is good since the learning rate is small and the initial prediction was already pretty close.

Note: The update for the biases is identical to the weight updates. In fact in some illustrations of neural networks the biases are drawn as just another weight, often w_0.

Furthermore the update of all weights can be viewed as simultaneously, meaning that for example for the update of w_1 one still uses the non-updated value of w_5.

1.10 Online and Batch-Based Learning

In Section 1.8 it was shown how the learning of a machine learning algorithm can be written as an optimization problem. Then in Section 1.9 it was shown how specifically in neural networks one can back propagate the differential error in order to find out which neuron makes the biggest mistake and change its weight accordingly. One thing that wasn't mentioned until now is how to present the data to a neural network. In what fashion the data is presented to the network can influence the result immensely. There are two commonly used ways how to do this: *online*, sometimes also referred to as incremental and *batch-based*, or sometimes referred to as offline, learning. In *online* learning the weights of the neural network network are updated with the new delta weight produced by one sample.

$$\Delta w \;=\; \eta \nabla E(w, \mathbf{x}^j) \tag{1.47}$$

where η is the learning parameter, $f(\cdot)$ is the learning rule, w is the weight (the parameter to be adjusted) and $\mathbf{x}^j$ is the feature vector of the j'th (randomly chosen) data point. In Section 1.8 this was introduced as *stochastic gradient descent*. This is done sequentially for every sample in the training set, always updating the weights before the Δw of a new sample is calculated. Once the Δw of each data point in the training set has been calculated and the weights of the network has been updated, one iteration (also called *epoch*) is done. Online learning is a stochastic process, as the error after each step is only calculated using one sample of the training set that is usually randomly drawn from the set. This is why the global error can also increase in one step, for example if one has by chance drawn a very strong outlier that does not fit in the current model, or a very noisy sample. In contrast to that, *batch-based*, or offline learning is a deterministic process. Here rather than updating after every training sample, the $\Delta w's$ of all data points are accumulated and afterwards the weights of the network are updated, for example with the average Δw of all data points.

$$\Delta w \;=\; \eta \frac{1}{N} \sum_{j=1}^{N} \nabla E(w, \mathbf{x}^j) \tag{1.48}$$

Since the $\Delta w's$ of all training samples are accumulated and averaged, the ordering in which the data points are presented to the network does not matter and should produce the same result, hence being deterministic. An often used relaxation of the batch-based learning approach is the so called *mini-batch* learning approach. Here instead of accumulating the

delta weights of all samples, a strictly smaller number of samples is taken for the weight update rule.

$$\Delta w \;=\; \eta \frac{1}{M} \sum_{j=1}^{M} \nabla E(w, \mathbf{x}^j) \tag{1.49}$$

where $M \;<\; N$ is the batch-size.

1.11 Classification vs. Regression

In Section 1.5 the problem of finding patterns in (often high-dimensional) set of data points was described and how this leads to the construction of automatic procedures, called machine learning algorithms. In Section 1.7 a *supervised* machine learning algorithm, namely a multilayer perceptron was explained in more detail.

This section will explain what different types of problems can be solved using this procedures and how to categorize them.

Let us for example assume, that the problem task consists of differentiating between two different kind of birds (A and B) given a set of their attributes. These attributes, or features (as explained in section 1.5) could be the length and coloration of their beak, their weight or other similar information. If an algorithm is now presented with a vector with these features x_i, it should decide with only this information if the shown object is part of species A ($C(x_i) \;=\; A$) or species B ($C(x_i) \;=\; B$)(or even better the probability of belonging to A or B). If one thinks about the features as axis in the euclidean space, one wants to find a (not necessarily linear) *decision boundary* which splits the space into two distinct subsets, where each point on one side of the decision boundary is part of one group and each point on the other side is part of another group. This grouping of objects is called *classification*. If one wants to only separate two classes, as in the previous dummy example one speaks about *binary classification*. Fig. 1.15 shows such an example of two linear separable classes with two features and a learned decision boundary. The extension to multi-class classification is straight forward, as there would just be more than one decision boundary. The difficulty of the classification task stems from the fact that each attribute individually is usually a so-called *weak-classifier*. This means that one attribute alone doesn't give much information about what class an object belongs to. So in order to get an accurate prediction one has to combine many of these weak classifiers in some form which then build a *strong classifier* for a better prediction. There are also many statistical problems with the class inference, like strong outliers, non-separability of the classes and so on. As the output of the classification task should be the most probable class of an object, the possible output space is *discrete*.

The exact opposite is true for the so-called **regression** task. The possible output space of the prediction is *continuous*. An often used example for regression in literature is the prediction of house prices. The features on which the price of the house depends could be, e.g. the number of bedrooms, the size of the house, the location. Instead of a decision

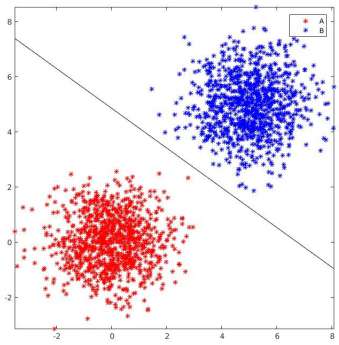

Figure 1.15: Example of decision boundary of classification of two synthetic classes A and B. The two classes where created via white gaussian noise (with different offsets/centers) and the decision boundary was learned via a simple *linear support vector machine* (SVM)

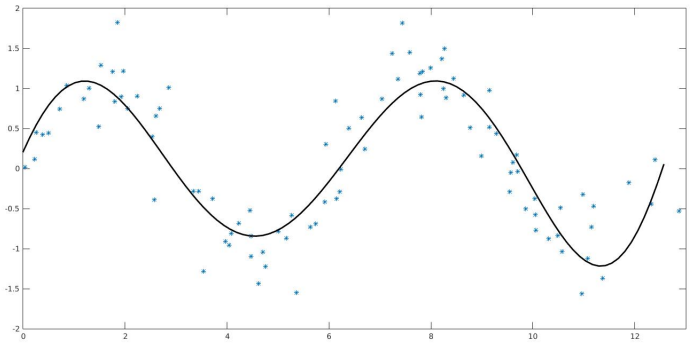

Figure 1.16: Example of a non-linear regression. The data points are 100 regularly sampled points along a sinus curve with an amplitude of 0.4π. Afterwards white Gaussian noise with different magnitudes was added in x and y direction. A polynomial of 8-th degree was then fitted to the data and is shown here in black.

boundary that splits the data into distinct parts we want to find a describing function of the independent variables (i.e. the features). In *linear regression* this correlation of the independent variable (the prediction, for example the house prices) and the *independent explanatory features* is modeled with a straight line. However for many application this is a weak model. For example a smaller house with a good location may still cost more than a bigger house in a not so good neighborhood. So to learn a non-linear relationship between the prediction and the features may be advantageous in many applications. This is called *non-linear regression*. An example of such a non-linear regression and the learned characterizing polynomial can be seen in Fig. 1.16

2

Existing Mulit-Line Scan Light Field Camera Setup and Goals

This thesis is part of an already existing activity focusing at a multi-line scan camera setup developed at the Austrian Institute of Technology (AIT). This camera was first proposed by Štolc et al. in [40]. The goal of this thesis is to create a proof of concept procedure in respect to the surface normal estimation especially regarding highly specular, glossy surfaces, that outperforms part of the current setup and will be incorporated into their setup in the future (see Fig. 2.4 for a simplified version of the existing pipeline and the part that should be improved). This will be done using a new machine learning approach, in particular using neural networks.

This setup consists of a multi-line camera, two constant light sources (light strips) and an object that is placed under the camera and moves during the acquisition process in a predefined direction on a transport stage. A foto of the existing demonstrator of this setup with its labeled parts can be seen in Fig. 2.1.

The camera allows for multiple active single lines on the sensor with multiple inactive sensor-lines between them. These lines on the sensor are randomly accessible, so that one can choose which lines are active and inactive in the acquisition process. Note: only the lines are random access, not the columns. With the disparity, or distance, between active lines one can produces different observation angles of the object. If, for example, only the first and the last line would be chosen to be active, one would have a standard stereo vision setup. Fig. 2.2 shows a schematic presentation of the camera and colored different views with different angles.

Between two acquisition steps s_i and s_{i+1} the object has to move the distance equivalent to exactly one pixel. After the acquisition process, the single lines acquired by one such step of each active line on the sensor are concatenated and thus all possible lighting angles and a number of different views are created. This produces a 3D light field structure (two spatial and one directional dimension), instead of the usual 4D structure. This 3D light field can be represented as an image stack that can be seen in Fig. 2.3.

This image stack is the resulting data after the multi-line scan acquistion process. The middle part shows the epipolar lines, or short EPI-lines (here a slice through the image

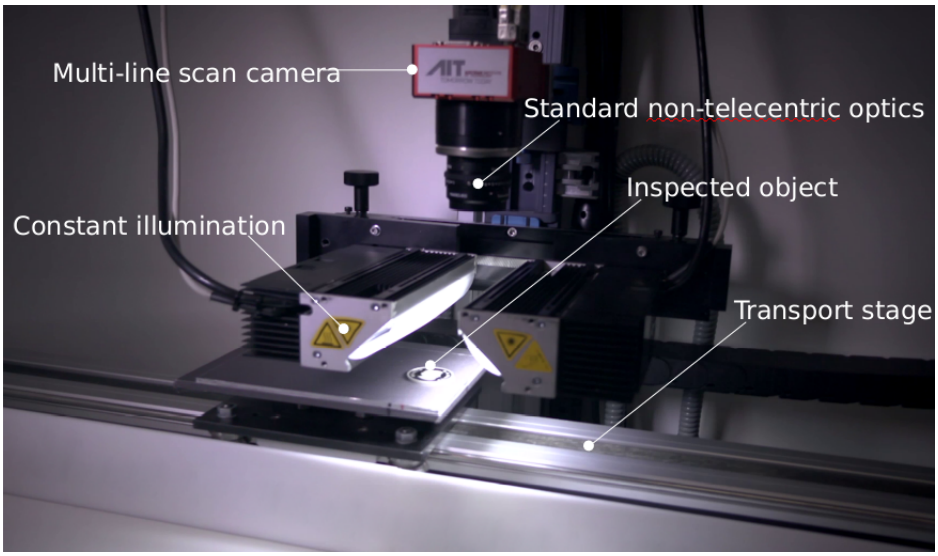

Figure 2.1: Foto of the multi-line scan camera prototype

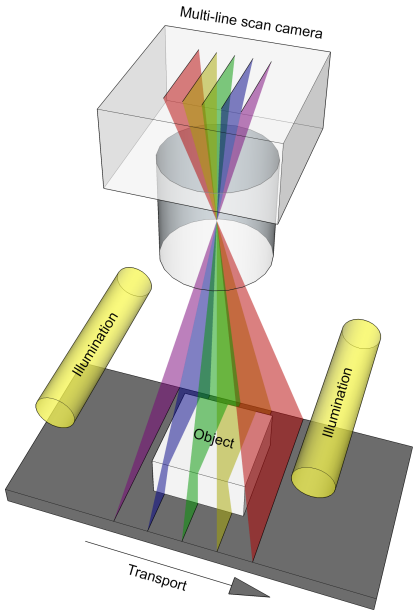

Figure 2.2: Schematic presentation of the multi-line scan camera with painted different active lines.

stack). The dashed line represents the read out of one such EPI-line with the respective RGB intensity vector **e**.

This allows for a fast in-line acquisition suitable for industrial inspection. However, as the production of different lighting angles is dependent on the movement of the object, the strongest lighting response is in the transport direction. Therefore geometry orthogonal to the transport direction was viewed as noise and omitted from the procedure (i.e. N_y).

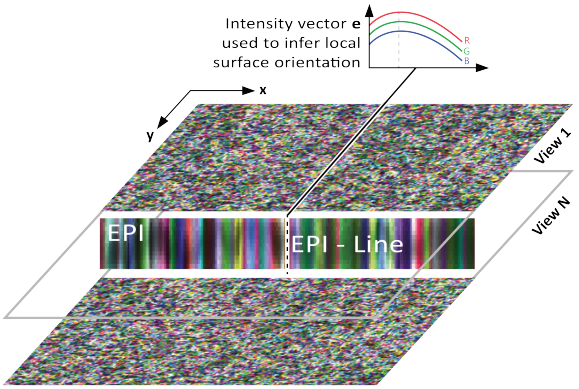

Figure 2.3: Visualization of the image stack

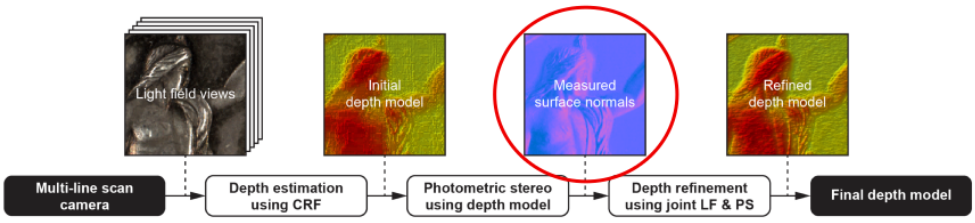

Figure 2.4: Simplified version of the process pipeline in the multi-line scan camera. The result of the circled step should be elaborated and improved by this project.

These intensity vectors **e** are used as input for a neural network, training a mapping between the intensity profile (dotted line in Fig. 2.3) of one point (i.e. pixel) to the surface normal. Since there did not exist any accurate measurements of surface normals of objects acquired with this system that could be used for training, it was needed to create a dataset.

Fig. 2.4 shows a simplified version of the setup as it is implemented now. It shows that the alignment of the light field data (here shifting of the view dimension so that corresponding pixels are on top of each other) and an estimation of the depth information is done before the surface normal estimation and is therefore handled as a given input for this project.

3 Related Work

The most relevant papers for this thesis, namely the paper that introduced *photometric stereo* in 1980 by R.J. Woodham [51] and the paper that introduced the camera setup used for the image acquisition in 2014 [41], have already been discussed in detail in the previous section (see Sec. 1.3 for photometric stereo and Sec. 2 for the camera setup). Since they first introduced the multi-line scan camera setup in 2014, a number of different papers was published regarding the camera.

For example in 2016 D.Antensteiner et al. published a paper describing that the fusion between the light field and the photometric stereo approach is advantageous and can improve the overall prediction of the depth [1]. In this work they described that the light-field depth estimation uses the epipolar geometry information (or more detailed the slope of the EPI-lines when sliced through the 3D image stack, see Fig. 2.3 for reference) compared to a reference frame for a depth estimation. She further describes that while this approach gives a better absolute depth accuracy (and a higher range of depth), this method on its own gives poor depth accuracy.
This means that detailed smaller structures of a model will be lost and the depth over all will be quite noisy in high frequency surface detail. The inverse is true of the photometric stereo approach. While the relative depth gives a good estimation (small structures are still visible, not as noisy), the absolute depth is not as good (small depth range). By combining both results in a linear fashion using weights and some filtering an overall better result can be achieved.

In 2017 another paper was published further refining and describing this hybrid technique by D. Antensteiner et al. [2]. There they describe the depth estimation algorithm in four steps (see Fig. 2.4 for a simplified version of this pipeline).

The first step is described as a multi-view correspondence analysis. This is done in the EPI domain using *census transform* image features. During the correspondence analysis also a set (discrete) number of disparities is tried for each location. This results in a cost volume at each location ($\mathbb{R}^{X \times Y \times Z}$, where X and Y is the spatial image coordinates and Z is the cost, or similarity of the location in the view dimension). This cost-volume structure

is the output of the first step. The matching method has been described in more detail by K. Valentín et al. in 2015 [47].

The second step then uses the information obtain by the first step in order to calculate an initial depth estimation. For this the conditional random field optimization technique as described by A. Shekhovtsov et al. in [39] was used. This method finds an approximation of the optimal discrete depth label at each location using some prior constraints and minimizing an energy function. The obtained optimal solution is then the initial depth model and the light-field estimate of the hybrid model.

The third step then uses the photometric stereo information. Once the depth information is approximately known, which again corresponds to the slope in the EPI domain, one can use this information in order to extract the intensity vector $\mathbf{e}$. This can than be used to estimate the surface normals as explained in Sec. 1.3. Here they also used some multiplicative scalar vectors on the gradients in order to account for inaccuracies in the conversion from real-world to discrete values.

In the fourth and last step they use both the light-field an photometric stereo information in order to find a continues refined disparity labeling. In order to do so they use a energy minimization technique which combines both results with some regularization parameters. The result of this step then represents the final refined depth model.

3D reconstruction using 2D images is a well studied problem in the field of computer vision. Over the years many different methods to solve this problem arose and many papers have been published regarding this topic. Other than *stereo matching* or *structure from motion* techniques (SfM), many of those papers use *light field* or *photometric stereo* in order to estimate depth and/or surface normals of objects.

In [22] S. Im et al. use light field for calculating depth estimation. In order to do so, they first extract corner feature points from a reference image using Kanade–Lucas–Tomasi (KLT) feature extractor and tracker [46]. Afterwards they use a RANSAC-based approach in order to calculate the essential matrix between the reference frame and the other frames. Features of moving objects are removed as they create artefacts. In the next step they use a small-motion bundle adjustment in order to get a sparse depth estimation. Using this sparse depth map they calculate a dense depth map by using a smoothness assumption (i.e. pixels around the sparse estimate should have similar depth if the color is similar) and creating a cost function using this information (sparse pixel plus neighborhood plus color) and some regularization parameter.

In [16] D. Goldman et al. used an alternating optimization technique between solving for the shape and the varying BRDF parameters. The main idea here is that while there are materials with non-homogeneous BRDF, most objects, according to their model, can be represented by at most two different BRDF's. They call these *fundamental materials*. They further write that these BRDF combination can spatially change and are a constraint of one pixel. These constraint on the pixel (being a linear combination of two different materials) helps alleviate the underdetermined problem of solving for the shape and BRDF.

They then start by estimating parameters of the BRDF (using a non-linear least squares method) by holding the surface normals fixed. These initial estimates of the surface normals are done using Lambertian photometric stereo. They use manual thresholds for specular pixels as to uphold the lambertian requirement. Afterwards they hold the previously estimated parameters of the BRDF fixed and compute the surface normals (and material weights) by brute force and linear projection using the previously mentioned constraints.

M. Tao et al. combined depth from focus with light field in their paper [44]. They exploited the fact that in commercially available plenoptic cameras (such as Lytro or Raytrix [15]) one can not only refocus an image but also change the viewpoint (given via the sub-apertures on the lense). With the different viewpoints they have a light field and can therefore also use the epipolar image analysis. They use this information in order to refocus by shearing the epipolar images. By trying a couple of such shearing factors and calculating some variance along the spatial horizontal axis (i.e. one line of the image, e.g. vertical direction in Fig. 2.3 they get a *defocus cue response*. This is done using the Laplacian operator. By computing the angular variance (i.e. in y direction, slope of EPI-line, see Fig. 4.9) they get a *correspondence cue response*. This is done straight-forward by using a standard deviation operator. In [23] they describe how to combine both cues in order to get a better depth map. They further show in their paper that combining these two method (depth from focus and depth from light field) can be advantageous as the depth from repeating patterns, for instance gravel or sand, as well as very noisy areas can have high ambiguities if one uses only light field data for depth reconstruction. By combining the result of the light-field cue with the defocus cue they get better results also in the previously mentioned cases.

Deep learning was used in [27] by F. Lu et al. in order to estimate object BRDF's. They use the additional angular information of a 4D light field structure (see Sec. 1.4) for their network architecture. They propose two different networks. One is called *StackNet*, were they use the angular information of a single light field image in an independent dimension, while *Ang-convNet* (their second proposed network structure) directly learns the angular information from the training data. For the data creation they used five different types of materials, namely plastic, diffuse, and three different type of metals (different specular lobes) with different 3D shapes and illumination angles. This led to about 47k (synthetic) data samples. These 4D light field images are then the input of the CNN. In StackNet they first convert the light field data into an 3D image stack (2D spatial dimension and views in third dimension). This is then subjected to a convolutional layer with 64 kernels and the softmax cross-entropy as the used loss function. For Ang-convNet (their second proposed network structure), they use the 4D light field structure directly as input to the convolution neural network. They use 64 filter kernels of size 4x4 (receptive field) which only filters the angular area in the light field image. This corresponds to a stride of $[4, 4]$. The results of both networks are then combined and send through a standard CNN structure (RELU, max-pooling). The result of this is then a

highly dimensional 1D vector. This 1D vector is than subjected to a softmax layer were the result of the predicted BRDF can be read out. They then show the results of the classification of the material types (confusion matrix) and compare it with results of a random forest implementation. They conclude that their network implementation works well and outperforms the random forest in most cases.

In [30] range scanning with stripe pattern were combined with photometric stereo with five light sources in order to recover the 3D surface of an object. Using epipolar plane image (EPI) structures from motion analysis for depth reconstruction was introduced in [5]. The paper from Tao deals with incorporating a shading term to depth from defocus with correspondence cues in order to refine the shape estimation [45]. In [17] Hayakawa used a singular-value decomposition (SVD) of a formulated matrix in order to get a surface normal estimation without the need of a-priori knowledge of the light source direction under the Lambertian assumption. Some machine learning approaches have been explored, such as [36] where a multi-layered neural network was used in order to learn the mapping between image intensities and the surface normal orientation, using a Gaussian sphere with average reflectance as the training data. In [8] Cheng used a symmetrical 6-layer neural network to train a mapping between the vectorized image and a reflectance value for each pixel. Another machine learning approach has been investigated in [9], where a neural network was used to solve the shape from shading problem, previously introduced by B. Horn et al. in [20].

4 Experiments

4.1 Generating Ground Truth

Cycles Renderer of Blender 2.78 [4] was used to generate the ground truth data. This artificial ground truth data has some advantages over real-world acquisition, such as the ease of modification of the setup, feasibility of generating many images quickly as well as being less prone to errors. However, in order to make the resulting images more realistic, some artefacts, such as jitter or salt&pepper noise, can be taken into consideration. The goal while creating the ground truth data was to cover as much ground as possible with the synthetic data regarding the task. A simplified version of the node setup in Blender that was used in order to create the different material BSDFs can be seen in Fig. 4.1.

The network should learn a mapping between the RGB intensity vectors of the different views and surface properties to the surface normal gradient. As it is infeasible to cover all possible mappings between color, light reflectance and surface normals, a random approach was chosen. A uniformly distributed, 8-bit random color pattern was created (each RGB color channel uniformly distributed between 0-255) and used as a texture. The blender-intern noise texture and displacement map node was used in order to create a random surface normal structure on a flat surface. With this approach the possible mapping space is sparsely covered.

Furthermore six different material datasets were created with different gloss values using a mixture of the *Diffuse BSDF* and *Glossy BSDF* node shaders. Fig. 4.2 shows these six materials as points in a Cartesian coordinate system were the axes represent the gloss (x-axis) and the roughness values (y-axis) that were chosen for each material type and the acronyms used for them. In this model two parameters can be changed, namely the gloss factor (controlled by the mix node) and the roughness of the two BSDF nodes. For the sake of simplicity, the roughness is the same for both, the Diffuse and the Glossy BSDF nodes. A schematic illustration of this setup can be seen in Fig. 4.1. This model is

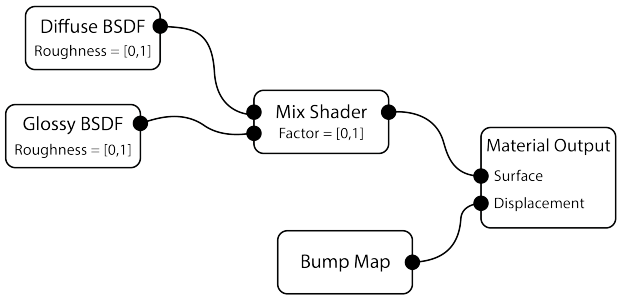

Figure 4.1: Schematic illustration of the *Blender Node Setup* for creating different materials.

based on a presentation from Gastaldo [12], where he states:

$$R + T + A = 1 \tag{4.1}$$

where R denotes reflectivity, T denotes transparency and A denotes absorption. Furthermore he states that reflectivity can be divided into diffuse reflectivity (R_d) and specular reflectivity (R_s). With this he derives:

$$R_d + R_s + T \leq 1 \tag{4.2}$$

In the setup R_d correlates to the Diffuse BSDF node and R_s to the Glossy BSDF node. Transparency was not taken into consideration (i.e. is always 0) as glass like materials were excluded from the data. The roughness parameter of the Diffuse BSDF node corresponds with the roughness of the *Oren-Nayar reflectance model* [33]. The model used for the glossy factor of the material was GGX [48]. The roughness parameter of the GGX model simulates microscopic bumps in the surface, so that the reflections of the material looks blurrier the higher the roughness parameter is.

A glossy dataset with a roughness value of 0 was excluded, which would imitate a mirror like behavior. However a glossy material with a roughness of 0.2 still shows highly specular behavior sufficient for this experiment.

The multi-line scan camera setup described in Sec. 1 was recreated in Blender, where a plane with a random color texture and a bump map (see Fig. 4.1) was moved underneath the camera. This Blender model can be seen in Fig. 4.3.

During each animation step the plane was moved by exactly one pixel. The resulting images were concatenated and reshaped in order to create a 3D image stack representation

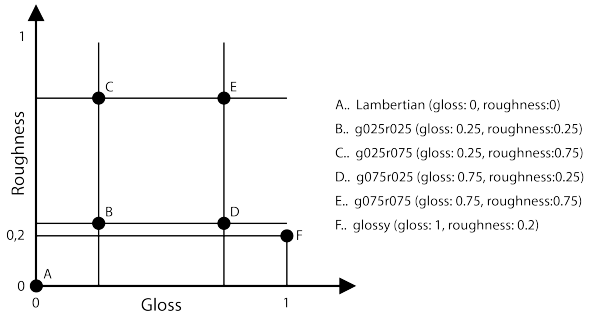

Figure 4.2: Visualization of the six different datasets created by changing the specular roughness and the percentage of which the glossy or diffuse node is taken.

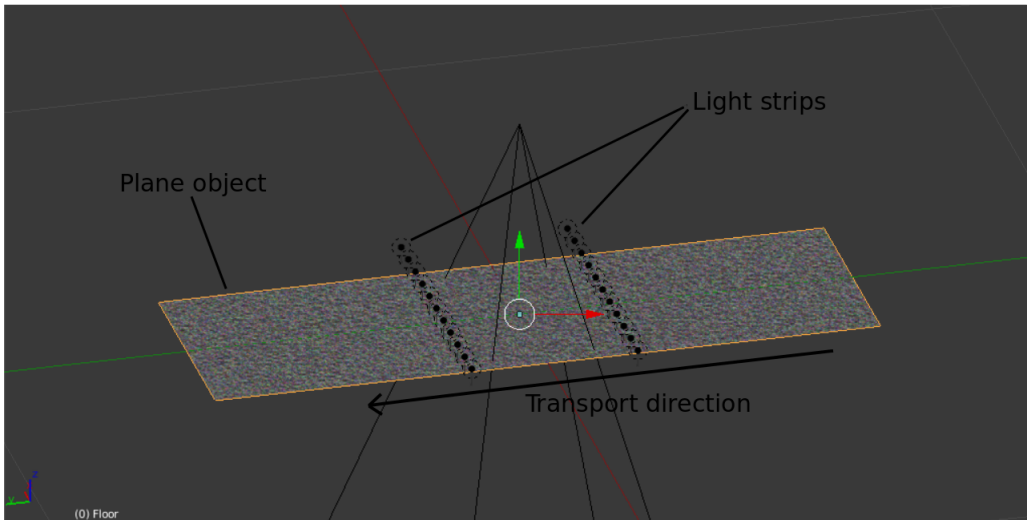

Figure 4.3: Blender scene of recreated multi-line scan camera setup. A random bump map is applied to the plane in order to create a 3D structure. In every acquisition step the object is moved exactly one pixel in the transport direction

of the light field. Each image plane is then shifted to the left in the following manner:

$$\forall x, y, i : I'_i(x, y) = I_i(x - ki, y) \tag{4.3}$$

where $i \in [0 \dots 12]$ denotes the index of the image in the 3D light field structure, $I_i \in \mathbb{R}^{width \times height \times 3}$ is the spatial image domain of the i-th view and I'_i denotes the new translated image, k denotes the number of inactive lines in between views. Since the disparity (i.e. the gap between active lines on the camera sensor) is 40 pixels it was used as the shifting constant. The resulting overlap (at most 12×40 in the last view) is then cropped.

This is done so that the EPI-lines are vertical with no slope, as they would be with an object with true 3D geometry.

For the creation of the dataset, 13 views (or active lines) with a disparity of 40 pixels (inactive lines in between each active lines, i.e. $k = 40$) was chosen. This resulted in around $\sim$ 500 000 pixels per material type or around $\sim$ 3 000 000 pixels in total.

4.2 Prediction Of The Surface Normal

In order to learn a mapping between the intensity vectors of individual points (here pixels) to object geometry, a regression artificial *feed-forward neural network* approach was chosen(see Sec. 1.11). In Sec. 4.3 it was already mentioned that in this setup, illumination responses are the strongest along the transport direction and weaker and very noisy orthogonal to it. This is why in this chapter only object geometry in transport direction will be learned. However, there is a possibility of getting more than just noise for the orthogonal dimension through iterative refinement and using the estimation of the surface normals in transport direction as a-priori knowledge. This was explored in the paper by D. Antensteiner et. al in [2]. Furthermore the gradient $\nabla x = \frac{Nx}{Nz}$, or more accurately $\nabla x = \frac{aNx}{aNz}$ was regressed rather than individual surface normal components. Training a network for ∇x proofed to be more stable, as training for Nx and Nz decreased the accuracy slightly on the experimental results, very likely due to floating point precision errors in the division of $\frac{Nx}{Nz}$. Also as the estimation of Nx and Nz can get very small, the gradient ∇x can "explode" during the training phase. Introducing an estimate of the albedo also does not help to make this more stable. The data used for all the experiments are the six computer-graphic generated datasets as described in Sec. 4.1.

4.2.1 Network Parameters Evaluation

For the optimal performance of a neural network some parameter evaluation and tuning, such as changing the number of hidden neurons, using different activation functions or cost functions, is needed. In our evaluation, we looked at 3 different activation functions, namely *linear*, *sigmoid* and *rectified linear* unit (RELU).

The input layer, which consists of 39 neurons (because one data sample consists of the three color channels information in all views, i.e. 3 channels $\times$ 13 views) is fully connected with the hidden layer. We tried different numbers of neurons in the hidden layer for each evaluated activation function respectively. The results of these experiments can be seen in Table 4.1 for the trainings set and in Table 4.2 for the testing set. For the read-out of the output layer, which consists of only one neuron as we only regress to the gradient in the transport direction ∇x, a linear activation function was used.

Given the problem we want to solve and our material properties, one can expect that a low number of hidden neurons will suffice and already give a good performance, as the Lambertian reflectance function is low dimensional. The low dimensionality of

a Lambertian reflectance has been proven and explored in [3]. Having a less complex network architecture can be beneficial for both the runtime as well as the generalization of the network. Here 1, 3, 10 and 20 neurons of one hidden layer was tried and evaluated.

The cost function used to measure the quality of the regressed prediction (therefore also the value used for the optimization of the network) was the mean square error (MSE):

$$MSE = \frac{1}{N}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{4.4}$$

For optimization the batch based standard gradient descent (no momentum term) algorithm with a learning rate of $\eta = 0.001$ was used. The dataset was split into 80% training set and 20% testing set, as proposed by the *Pareto Principle* by J.M. Juran [37]. The network was always trained for 100 epochs and no early stopping or weight decay was used, as the network did not seem to overfit.

Training set MSE

# hidden neurons	1	3	10	20	avg
act. fct.					
linear	0.05903	0.05988	0.05760	0.05857	0.05877
Sigmoid	0.05429	0.05285	0.05263	**0.04792**	**0.05192**
RELU	0.05605	0.05543	0.05283	0.05150	0.05395

Table 4.1: Training MSE with different numbers of neurons and activation functions.

Testing set MSE

# hidden neurons	1	3	10	20	avg
act. fct.					
linear	0.05902	0.05972	0.05777	0.05855	0.05877
Sigmoid	0.05402	0.05276	0.05266	**0.04768**	**0.05178**
RELU	0.05608	0.05571	0.05312	0.05147	0.054095

Table 4.2: Testing MSE with different numbers of neurons and activation functions.

Table 4.1 and Table 4.2 show that using the Sigmoid activation function in the hidden layer has both the best overall, as well as the best performance in a single run with 20 neurons in the hidden layer. As the aforementioned experiments were performed only to show the overall tendency and convergence of the network structure, a small learning rate η was used for all the experiments. However, [34] shows that exploring this parameter further is important for the overall network accuracy. For this task we found that a learning rate of $\eta = 0.2$ works best which improved the overall accuracy of the network to $MSE_{train} = 0.020464$ and $MSE_{test} = 0.02052$ when trained for 100 epochs.

4.2.2 Network Performance

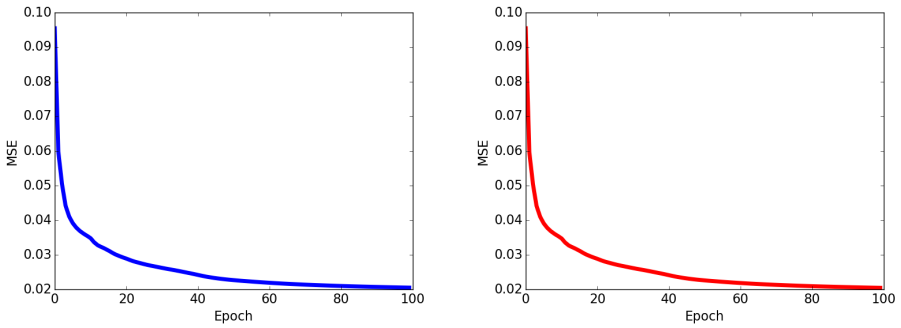

Figure 4.4: Evolution of the mean square error over epochs with a learning rate of $\eta = 0.2$. Left: Training set (80% randomly chosen from all sets), right: Testing set (20% randomly chosen from all sets).

Fig. 4.4 shows the convergence of the overall accuracy on the training and test set, combining and shuffling all six created datasets. This was done in order to generalize the network as much as possible regarding the material type (matte, semi-glossy or glossy). Once the network was learned it was applied to each material type individually and the accuracy of the prediction on the whole set was reported. For simplicity we use acronyms for each created dataset, which are as follows:

- Lambertian (gloss:0,roughness:0)
- g025r025 (gloss:0.25,roughness:0.25)
- g025r075 (gloss:0.25,roughness:0.75)
- g075r025 (gloss:0.75,roughness:0.25)
- g075r075 (gloss:0.75,roughness:0.75)
- glossy (gloss:1,roughness:0.2)

Table 4.3 shows the accuracy of each individual dataset. For the sake of simplicity we took the liberty of reporting the error on the whole dataset (data points used for training and testing combined). As the errors on the training and on the testing set are very close together and there is no sign of overfitting the network, this liberty can be taken without distorting the results. The best performance was achieved on the semi-glossy datasets. The larger error on the glossy dataset is due to the fact that the sign of the surface normal is sometimes predicted wrong if the specular lobe is narrow and outside of the observed range. Fig. 4.5 shows a more qualitative result of network prediction in the form of a colored gradient map for some datasets. In this figure the aforementioned behaviour can

Dataset	MSE
Lambertian	0.01637
g025r025	0.01537
g025r075	0.01835
g075r025	0.01760
g075r075	0.01795
glossy	0.03722

Table 4.3: MSE of each individual whole dataset applied to the network

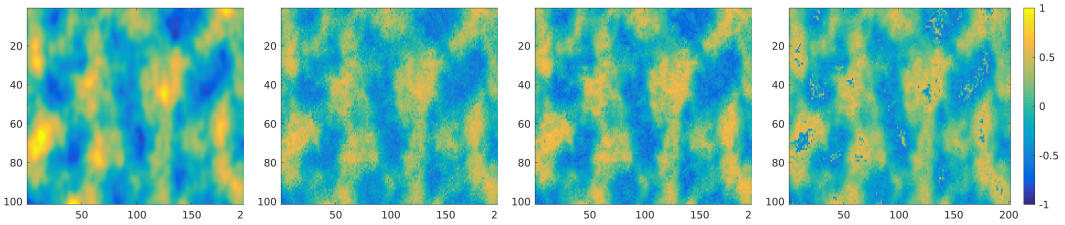

Figure 4.5: from left to right: ground truth map of ∇x, surface normal gradient in transport direction of a Lambertian material learned by the network, surface normal gradient in transport direction of a semi-glossy (gloss = 0.25, roughness = 0.75, see Fig. 4.2) material learned by the network, surface normal gradient in transport direction of a very glossy material learned by the network. The properties of the different materials can be seen in Fig. 4.2.

be observed in the right-most figure, were one can observe sign flips in the highest or lowest regions.

As described in Section 1.3 for the sake of comparability also a slightly changed traditional closed-form solution for photometric stereo was implemented. Because of the synthetically generated data, the unknowns were rather the light positions (or light matrix) than the surface normals (which can be read-out from the rendering process). So the general equation (see Sec. 1.3):

$$\mathbf{n}a = \mathbf{L}^{+}\mathbf{e}, \tag{4.5}$$

has to be slightly changed, as the normals are known (from the renderer). Instead of solving for $\mathbf{n}$ we solve for the unknown light matrix, were the albedo is given as a diagonal matrix.

$$\begin{bmatrix} e_1^1 & \dots & e_1^N \\ e_2^1 & \dots & e_2^N \\ \vdots & \ddots & \\ e_V^1 & \dots & e_V^N \end{bmatrix}_{V\times N} = \begin{bmatrix} L_1^1 & \dots & L_1^3 \\ L_1^2 & \dots & L_3^2 \\ \vdots & \ddots & \\ L_1^V & \dots & L_3^V \end{bmatrix}_{V\times 3} \begin{bmatrix} N_x^1 & \dots & N_x^N \\ N_y^1 & \dots & N_y^N \\ N_z^1 & \dots & L_z^N \end{bmatrix}_{3\times N} \begin{bmatrix} a_1^1 & \dots & \emptyset \\ \vdots & a_2^2 & \vdots \\ \emptyset & \dots & a_N^N \end{bmatrix}_{N\times N} \tag{4.6}$$

In Eq. 4.6 N stands for the number of samples used to solve for the light matrix L and V stands for the number of views. Then to get to re-written formulation, by getting rid

of $\mathbf{n}$ and a on the right-hand side of the equation we end up with:

$$\mathbf{e}\ diag(1/a)\ \mathbf{n}^{-1} = \mathbf{L}. \tag{4.7}$$

Note that Eq. 4.6 shows the general form for this equation. In the concrete implementation in this project, the number of views V was 11 and also (as previously explained) the y-coordinate of the surface normal was omitted, which lead to slightly different dimensions for $\mathbf{L}^{V\times 2}$ and $\mathbf{n}^{2\times N}$.

This optimized $\mathbf{L}$ matrix was than used as seen in Eq. 1.3 for the surface normal prediction.

Here two different approaches were taken, one with Woodham's original assumption of Lambertian reflectance, which was named *Lambertian model with Lambertian dataset*, or abbreviated L.m.L. Here all the available samples of the Lambertian dataset were taken for the prediction of L. However also all material reflectance datasets were tried for solving the overdetermined linear equation system. This approach was named *Lambertian model with all datasets*, or abbreviated L.m.a. Here the exact same set of data points that were also used to train the network was taken. The accuracy of all three approaches (L.m.L, L.m.a and neural network) on all individual datasets as well as the average performance can be seen in the barplot in Fig. 4.6.

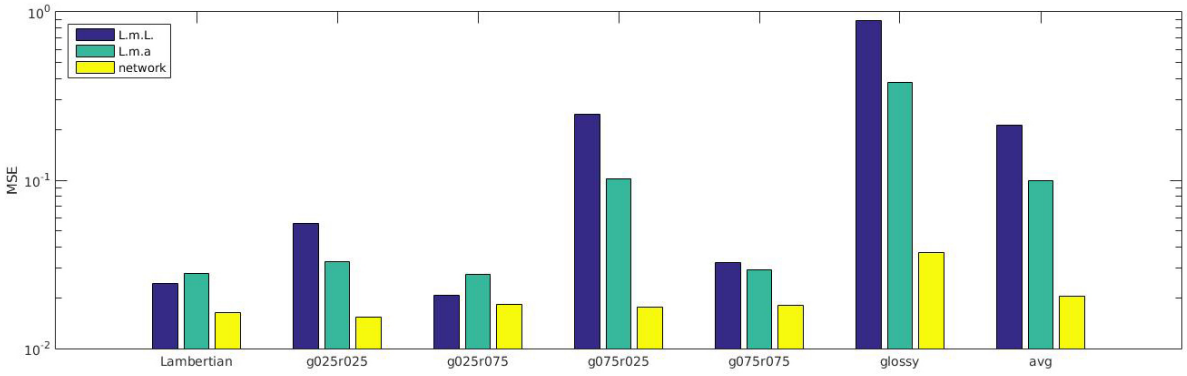

Figure 4.6: Barplots showing the accuracy of the three different approaches on each individual material dataset.

Fig. 4.6 shows that on average the L.m.a. model is better then the L.m.L model, in fact it is only worse for the two very diffuse materials *Lambertian* and *g025r075*. Appendix A shows some visual qualitative results of the prediction of the gradient map of every tested model for every material type, as well as an error map (absolute distance between ground truth and prediction) and a correlation plot between labels and predictions.

4.2.3 Experimental Results On Real Light Field Data

After training the network it was applied to real light field data acquired by the multi-line scan camera. These results should be seen as experimental, as it was an uncalibrated system. Also the real light field data has less number of views (9-11) than the synthetic data with unknown disparities between the views. Rather than decreasing the number of views in the synthetic data, the light field data acquired by the real setup was changed so to fit into the network. This was done by adding additional views on top and bottom of the image stack (i.e. before the first and after the last view in the z-Dimension of the 3D data). In order to generate this missing views, a cubic spline extrapolation for every pixel and color channel was used. This of course introduced new additional noise (especially in the green channel).

In the future one could also re-train the neural network with the same amount of views as this datasets. This could be done by deleting the first and the last view of the synthetic data altogether, however the further away the illumination angles are from the angle orthogonal to the object, the better the shading information is. This could therefore lead to a less good predictions. Deleting the middle views (i.e. the one closest to axial) would lead to a non-equivalent disparity within active lines and to a huge gap of inactive lines in between the two new middle views. Therefore the probably best approach for the future would be to re-create the synthetic data, using 9, 10 or 11 views and choosing the number of inactive lines accordingly in order to have equivalent disparity between active views.

The inference results of the three methods on a real light-field coin object can be seen in Fig. 4.7.

Fig. 4.7 shows that although the neural network performs better on the synthetic data, the experimental results on real light field data seems to have degraded gradients.

To further explore this, several new neural networks were trained and afterwards inference was done on the same coin dataset as shown in Fig. 4.7. The results of this can be seen in Fig. 4.8

Fig. 4.8 shows that the networks only trained with the Lambertian or the r025g025 give the best results on this specific dataset. This shows that these datasets probably resemble this light-field data the closest in respect to the intensity profiles. One can also see that the network on the neural network trained on the glossy dataset degrades the gradients the most.

This can be explained by looking at the intensity profile of this rather shiny material dataset. As the reflectivity is much higher in the glossy case, this also means that for high or low gradients the observed specular peak is much higher, meaning that it only observes (and in fact learns) high gradients with high peaks. As this real light-field dataset has not such a high specular response for the high and low gradients, the network will interpret this as flatter areas (or lower gradient).

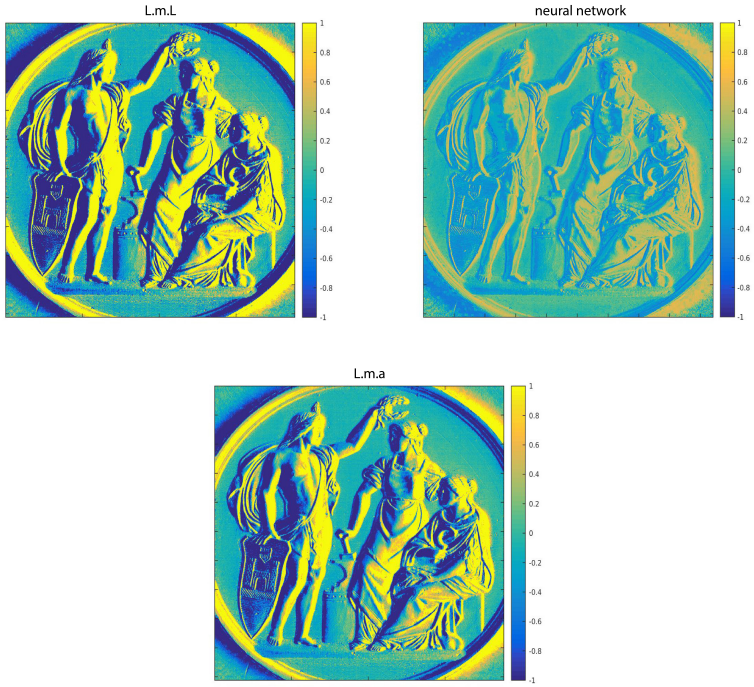

Figure 4.7: Predictions of a gradient map from light-field data acquired by the real setup. From left to right top to bottom: L.m.L, neural network, L.m.a

4.2.4 Experiments With New Dataset

The previous experiments have shown that the regression task of learning a mapping between an intensity profile of one object location to the ∇x component of the surface normal works well and has advantages over a linear model. However applying the trained network to real light-field data acquired by the demonstrator (described in Section 2) yields degraded results. Fig. 4.8 has shown that this is likely due to the fact that the shown trainings data of the mixed synthetically created BRDF's does not represent the distribution of the real data well enough.

For this reason a new approach was taken to create better training data for the network using real light-field acquisitions and some assumed geometry. The same requirements on the trainings data as mentioned before applies here, namely it should cover as many different surface normals orientation as possible with different BRDF's. A good geometric shape for this requirements is a sphere (or more correct a half-sphere) as it covers most possible normal orientations. Therefore for the new dataset precision balls of three different material types (steel, ceramic and plastic) where used. These balls were placed on a metal background and fixed in place with an adhesive tape. Fig. 4.11 shows an example of one ball of each material type

In order to find and select the balls, a *Circle Hough Transform* [35] was used for each of

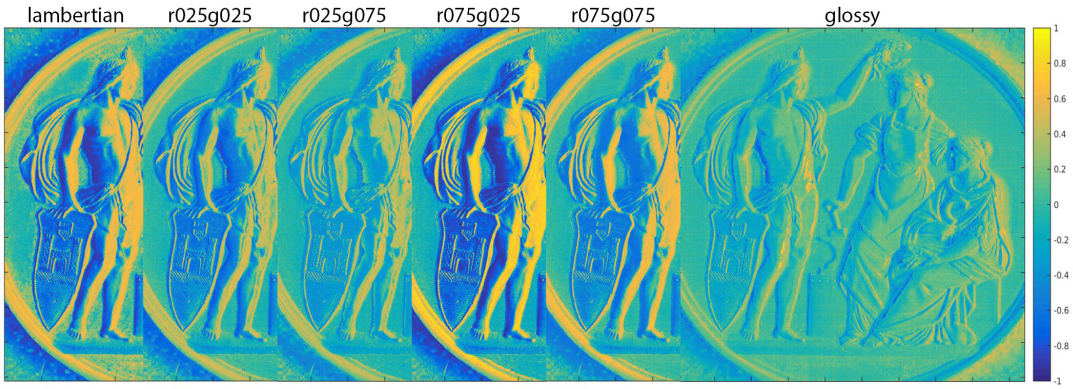

Figure 4.8: Inference results of the different networks trained solely on (from left to right): Lambertian, r025g025, r025g075, r075g025, r075g075, glossy synthetic dataset

the 11 views individually. For this purpose a Matlab implementation of the algorithm was used. The only thing needed for this to run was the radius of the spheres to be detected, which in this case was 25 pixels. After locating every sphere in every view like this the corresponding patches with the balls were once again concatenated to a 3-dimensional image stack (images stacked in the third dimension). As the standard implementation of Matlab did not always find the pixel-perfect centers and contours of each ball (especially in the case of the plastic balls), some manual post-processing was needed. This was done using visual marker (one manually colored pixel in the exact same position in every view). The circle detection could probably be improved, for example by using some kind of better voting scheme in an iterative fashion. However due to time constraints and because it was only needed once for the creation of the calibrated dataset the manual approach was more efficient.

After this step the EPI-lines of the 3D light-field structure of each individual ball were still sloped (or not orthogonal) as expected of real 3D structures. In other works, these EPI-slopes are used in order to calculate the geometry of the object [24][43][42]. Fig. 4.9 shows an illustration of such a sloped EPI-line of one point over the views. As the intensity vector used as input for the learning task is a slice in the third dimension through the light-field (as seen in Fig. 2.3) the image stack needs to be shifted so that corresponding pixels are orthogonal.

Under the assumption that the distance between each view is constant (see dashed line in Fig. 4.9) each view was shifted using the following equation:

$$shift = \frac{\#view}{tan(\alpha)} \tag{4.8}$$

This scalar was then used in order to shift each view individually with sub pixel

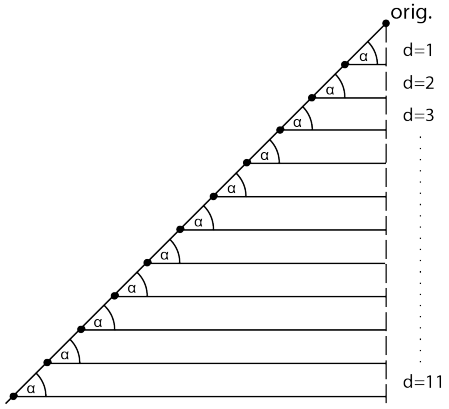

Figure 4.9: Schematic representation of a sloped EPI-line. The dark dots represent the same pixel in each view. The horizontal lines represent the different views and the hypotenuse represents the sloped EPI-line as it would appear in the slice of the light-field.

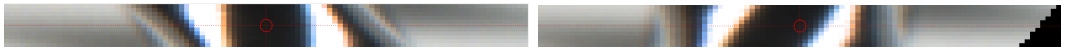

Figure 4.10: left: EPI-line profile of one pixel row before shifting, right: EPI-line of one pixel row after shifting (from top to bottom: first view to last view of same row)

precision using Matlab's *imtranslate* function using linear interpolation. The parameter α was chosen through trial and error for each ball. An example of how this looks for one line of a steel ball before and after the image stack is shifted can be seen in Fig. 4.10.

As the circles are stored in a square matrix there are still some pixels that are element of the background. In order to set those to zero a binary mask was created using the knowledge that the center of the sphere is the mid element of the square matrix and the radius is exactly 25 pixels. This mask was then used as a filter to set the pixels that are element of the background to zero.

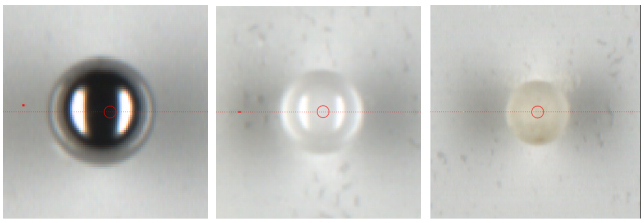

Figure 4.11: Balls with the different material types. left: steel, middle: ceramic, right: plastic

Under the assumption of a perfect sphere, the surface normals can be calculated as follows:

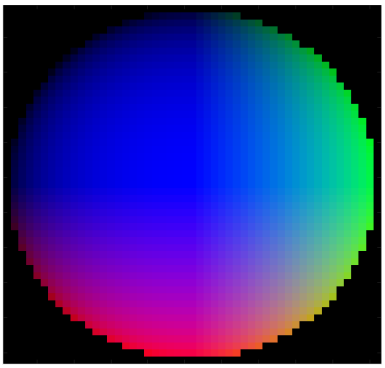

Figure 4.12: Calculated surface normals in normal mapping color scheme

$$x' = \frac{x - r}{r} \tag{4.9}$$

$$y' = \frac{y - r}{r} \tag{4.10}$$

$$z' = \sqrt{1 - x'^2 - y'^2} \tag{4.11}$$

Here, x is the spatial coordinate of the sphere in x-dimension, y is the spatial coordinate of the sphere in y-dimension and r is the radius of the sphere. (x', y', z') are the calculated x, y and z component of the surface normal. The $(x - r)$ in the numerator is so to get the position relative to the spheres origin. if the center of the sphere would be at the origin $(0, 0)$ this would reduce to:

$$x' = \frac{x}{r} \tag{4.12}$$

$$y' = \frac{y}{r} \tag{4.13}$$

$$\tag{4.14}$$

The result of the so created surface normals in normal mapping color scheme can be seen in Fig. 4.12.

With this approach however two problems regarding the target labels (i.e. the surface gradients) are, that there are far more data points that are close to flat, i.e. normal to the fronto parallel plane (around the center) than steep gradients (around the edges of the half-sphere). In order to circumvent this as good as possible, here the following approach was chosen:

A binary mask only selecting the pixels on the outer ring of the sphere (with a width of 4 pixels) was created and used for every ball. These points were used to doubled the amount of data samples with steep gradients which lead to better result. This lead to

around 43K data samples in whole (training and test set combined), which is a lot less than the amount of data samples created synthetically.

The second problem that arises originates from the fact that not every pixel measurement of the sphere is useful for the trainings task. Fig. 4.10 shows that especially for the case of the glossy steel ball (which is the most reflective surface) some pixel around the center never change their brightness and stay mostly black. As these points on the sphere do not show the behaviour that should be learned (i.e. change in intensity profile over views to learn the surface orientation) these points were excluded from the dataset. This further decreased the dataset a little. It should be said however, that these points might as well also be kept in the dataset, as a neural network is expected to be able to deal with a bit of noise in the data.

Another experiment that was tried was to enrich the so created dataset by changing the brightness of some data points. In order to do so, a smaller set of pixels from the original data set were chosen. For each of this data points a random value within some margin (here $[0.6, 1.6]$) was chosen and then multiplied to the intensity vector (constant over the three color channels and 11 views). These modified points were then added to the dataset. However this approach did not show any improvement of the overall accuracy of the inference.

Another slight improvement over the previous experiments was obtained using a sigmoidal activation function (which has a range of $[0, 1]$) for the second output neuron (corresponding to the Z gradient component) to enforce that the Z normal is within the right range. This only very slightly improved the final prediction, as also with the linear read-out the Z gradient of the output stayed in range for the most part. However this change was kept as it is a more correct way of setting up the network. The final prediction of the network trained with the new training data on the coin dataset can be seen in Fig. 4.13.

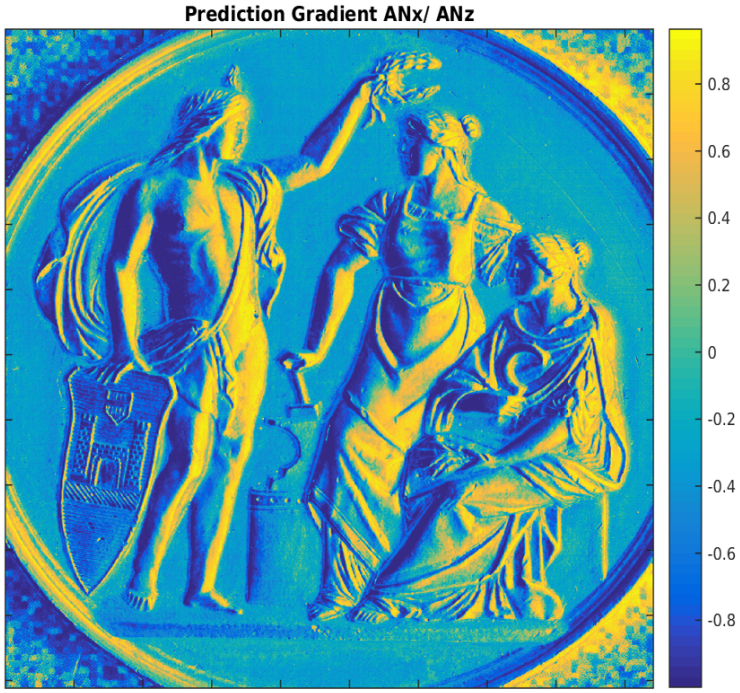

Figure 4.13: Final result of the trained network using the Coin-figural dataset acquired by the real demonstrator setup

4.3 Classification Of The Material

In order to classify our created datasets the network had to be changed slightly. The data samples used for class inference are the same as used in Chapter 4.2. Instead of using MSE (Equation 4.4) as the cost function, for a classification task one should use Cross-Entropy (CE):

$$CE = -\sum_{j=1}^{N}(y^j \ log(\hat{y}^j)) \tag{4.15}$$

More exact, for a batch base approach, the average cross-entropy (ACE), which means the average CE score over the samples in the mini-batch, should be used for the back propagation step.

$$ACE = -\frac{1}{N}\sum_{j=1}^{N}(y^j \ log(\hat{y}^j)) \tag{4.16}$$

While the mean square error loss function by no means fails for the task of multi-classification, it has some inherit disadvantages over the average cross entropy score. For one, MSE gives a lot of emphasis on incorrect outputs (i.e. trying to minimize the wrong

positives) and on the other hand, as the true positive values get larger, the MSE gets very small, leading to small weight changes in the back propagation step.

Since six different datasets where created (i.e. materials, see Figure 4.1) we have six different classes $C = \{C_0, C_1 \dots C_5\}$. For a binary classification task one output neuron would suffice, e.g. by creating a threshold were $f(\mathbf{x}) < 0.5$ is set to be the first class and $f(\mathbf{x}) \geq 0.5$ is set to be the second class. For a multiclass classification one needs an output layer with as many output neurons as there are classes, so $\#classes = \#neurons_{output}$. Each of these output neurons corresponds to the probability that a certain data point is of the corresponding class. The probabilities of all classes for one specific sample add up to one like

$$\forall \mathbf{x}^j : \sum_i P(\mathbf{x}^j = C_i) = 1. \tag{4.17}$$

In this case the best practice is to use a softmax activation function for the output neurons. The class prediction (denoted with $\hat{y}$) is the class with the highest probability

$$\hat{y} = \arg\max_{\mathbf{x}} f(\mathbf{x}). \tag{4.18}$$

In order to deal easier with the error calculation (i.e. the cost function), the class labels of the datasets were encoded using the *one-hot* encoding. One-hot encoding is a binary code, where the lenght of the code l_c is equal to the number of attributes, here classes. The active, or "hot" attributes are encoded with 1, while the rest is 0. Since in our case every sample can only belong to exactly one class, one entry in the one-hot encoded vector is always one, while the rest is 0.

Lambertian	Glossy	r025g025	r025g075	r075g025	r075g075
1	0	0	0	0	0
0	1	0	0	0	0
0	0	1	0	0	0
0	0	0	1	0	0
0	0	0	0	1	0
0	0	0	0	0	1

Table 4.4: One-hot encoding of each class label

For reporting the accuracy of the network, the misclassification rate (MCR) was used.

$$MCR = \frac{1}{N} \sum_{j=1}^{N} C(\mathbf{x}^j)_{pred} \neq C(\mathbf{x}^j)_{label} \tag{4.19}$$

A good way to present the classification accuracy of the network, other than the MCR is the *confusion matrix*. In this matrix each row corresponds to the total amount of classified samples of a specific class. The columns represent the different classes that one sample can be classified as. So the confusion matrix has the dimension of $ConfMat : R^{c \times c}$,

where $c = \#classes$ and is therefore a square matrix dependent on the number of classes. The diagonal elements are the *true positives*, meaning the samples of the given class that were classified as this class. The other row elements are the samples of the class that were classified as a different class (corresponding to the row).

Let's consider a small example of such a confusion matrix: Let's assume we want to distinguish between three different kinds of animals, namely *cats*, *dogs* and *rabbits* given their set of features. Let's further say that for each class we have data of 100 individuals. After training (for instance a neural network), a confusion matrix may look like:

	cat	dog	rabbit
cat	85	10	5
dog	12	80	8
rabbit	33	3	75

Table 4.5: Confusion matrix of example

The confusion matrix not only shows the accuracy of the prediction, it also shows which classes get confused more easily. For instance in this example one could see that cats are twice as likely to be wrongly classified as a dog than as a rabbit. Rabbits on the other hand are almost always wrongly classified as cats and so on. If you divide each row by its sum (i.e. number of samples in the class), you get the *normalized confusion matrix*, which shows the accuracy not in absolute but in probabilistic terms (i.e. percentage).

In this example the normalized confusion matrix would look like:

	cat	dog	rabbit
cat	0.85	0.1	0.05
dog	0.12	0.8	0.08
rabbit	0.33	0.03	0.75

Table 4.6: Normalized confusion matrix of example

4.3.1 Network Parameters Evaluation

Since the learning task has changed, some new parameter evaluations has been conducted. Since it is a classification task, the misclassification rate (percentage of wrongly classified samples) has been reported rather than MSE. First the exact same network structure that was the best fit in Section 4.2 was taken and tried with the Sigmoid (Equation 1.17) and RELU (Equation 1.16) activation function. The misclassification rate when trained for 100 epochs can be seen in Table 4.7

As the RELU activation function yields better results, it will be taken from here on out. However the network performs very poorly, as about 78% are wrongly classified. The obvious reason for this would be that the network complexity is too low (i.e. not enough neurons), however empirical studies with this dataset have shown that using more

	MCR Train	MCR Test
Sigmoid	0.79669	0.79707
RELU	0.74118	0.74137

Table 4.7: Misclassification rate of Sigmoid and RELU activation function

neurons per layer, or adding more layers does not improve the accuracy of the network. The experiments have been conducted with up to 60 neurons per layer and up to 6 layers deep. Even the deepest network structure did not improve the MCE alot after training.

An improvement to the overall accuracy of the network has been to change the cost function from simple gradient descent to the now widely used Adam optimizer [25]. The name of this cost function is derived from the fact that it uses ***ad****aptive* ***m****omentum*. This means that rather than keeping the learning rate η fixed for every step of the descent, it is changeable (adaptive) for each iteration. In contrast to AdaGrad [11], Adam uses two parameters to change the momentum, namely the first and second momentum of the gradients. This improved the overall classification error to about 50%. The Adam optimizer has shown to work well for many applications and often outperforms other functions.

4.3.2 Network Performance

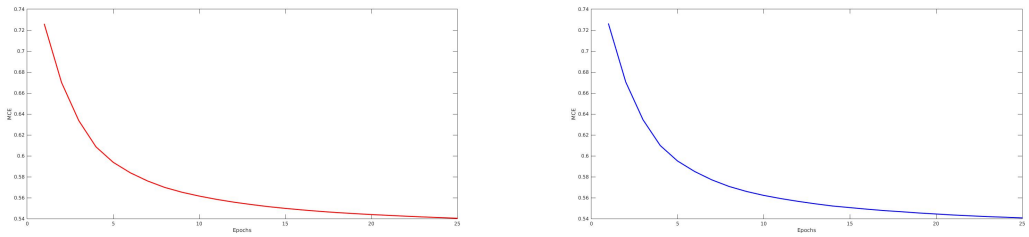

Figure 4.14: Evolution of the mean classification error over epochs. Left: Training set (80% randomly chosen from all sets), right: Testing set (20% randomly chosen from all sets).

Even after some parameter optimization, the overall accuracy of the network remained poor and as is, can not be used for most real application.

There can be multiple reasons for this. For one, the intensity profile of one individual pixel is quite a weak classifier. Another reason lies within the fact that the classes (here materials with different reflection profiles) may not be very well distinguishable. Or differently put: The inter-class variation is too low. However, Fig. 4.14 shows that the training and test error of the network converges.

One interesting behavior that was observed during the experiments, was that the MCE can be misguiding in respect to the actual prediction of the network. Sometimes during training, the MCE could stagnate (i.e. be the same range) for many iterations or at time

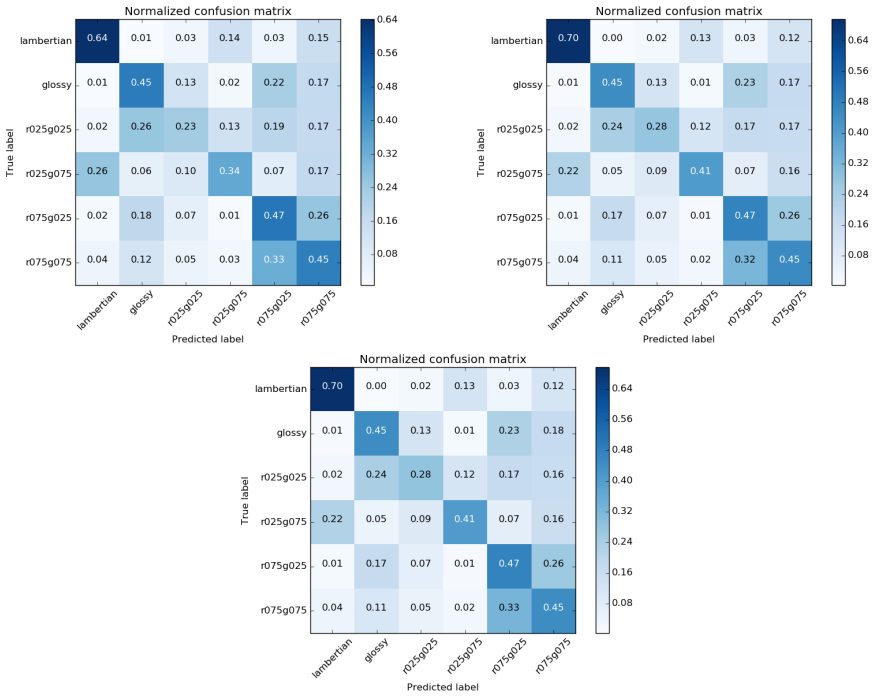

Figure 4.15: Confusion matrix (from top to bottom, left to right): Epoch 10 training set, Epoch 25 training set, Epoch 25 testing set

even get slightly worse in the next iteration. This would suggest that the network is already converged and starts to overfit, or is stuck at a local minimum. However, in order to get a scalar representation of the accuracy, the MCE reported in this experiment was the average misclassification rate over all classes.

Therefore, looking at the (normalized) confusion matrix gives a better idea on how well of a job the network does. Plotting the confusion matrix after every iteration, one can see that while the overall error may not improve, the diagonal entries become stronger.

For example during iteration 10 to 11 (one iteration again being all mini-batches being used to train once), the trainings MCE went from 0.50 to 0.52, however looking at the confusion matrices shows that the true positives became better.

Fig. 4.15 shows the confusion matrix on the training set after 10 epochs as well as the confusion matrix on both the training set and test set after the final epoch. Not surprisingly the best classification was achieved on the Lambertian dataset, which was classified right 70% of the time. The worst classification by far was achieved by the r025g025 dataset which, excluding glossy material, seems close to a coin flip for the classification.

As the result of this experiment was not satisfactory, the results on real data was omitted. To improve this experiment one could think about better material types (or in fact, have more realistic and distinct BSDF's) that are more easily separable. Also for type-classification neighbor information, such as used in a convolution neural network, would be advantageous and quite possibly improve the accuracy of the network.

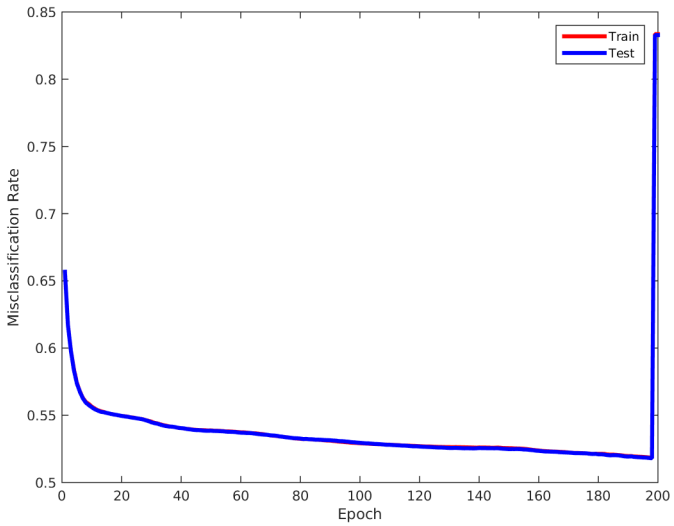

Figure 4.16: After 200 epochs the neurons died and outputting only 0

4.3.3 Dying RELUs

While the *rectified linear unit* activation function has some advantageous properties, like boosting the accuracy of the trained networks, or generally being very fast to compute, it also has its negative properties. One problem with the RELU activation function is that neurons can "die" during the training phase. That means from a certain point on the neuron only outputs zero and doesn't recover from this state (i.e. "died"). This problem stems from the form of the activation function itself and the back propagation algorithm and happened in practice in this particular classification task quite often. When the input passed to a RELU activation function is negative, it outputs 0 or else it outputs the identity function. A RELU neuron "dies" once it becomes zero. Since the derivative of the function remains zero, once a neuron is in this state, a dead RELU cannot leave this state, as the backwards pass will not update the weights and therefore it remains dead. The *leaky rectified linear unit* was build in order to prevent this RELU unit dying, however as of the time this thesis was written, tensorflow does not support this activation function. Since one of the categories used in this classification task was coded with 0 and categories have the exact same number of samples, exactly 16% or 1/6 of the samples where correctly classified with 0. Fig. 4.16 shows that after the 200th epoch neurons are dying, leading to the jump in the MCR.

4.4 Albedo or ColorMap Estimation

The goal of this experiment was to learn a mapping between the observed intensity vectors to the true RGB values of pixels (in some applications also referred to as albedo, as

explained in Sec. 1.2). This can be advantageous for pixels that are highly reflective (or specular), as such pixels can pose difficulties if one just approximates the true color using e.g. the mean value. The training and test data is the same as in all previous networks (described in Sec. 4.1). The labels of the pixel are the corresponding RGB values of the texture that was applied to the object.

The network consists of 39 input neurons (13 views for each color channel stacked on top of each other) and 20 neurons in the hidden layer with sigmoid activation functions. The output layer has 3 neurons with linear read-out corresponding to the estimation of the RGB value of the pixel. Standard gradient descent with a learning rate of $\eta = 0.2$ was used as an optimizer function and the mean-squared error as a cost function (the mean of the error of each individual channel was used for the backwards pass). This network was trained for 500 epochs.

4.4.1 Network performance

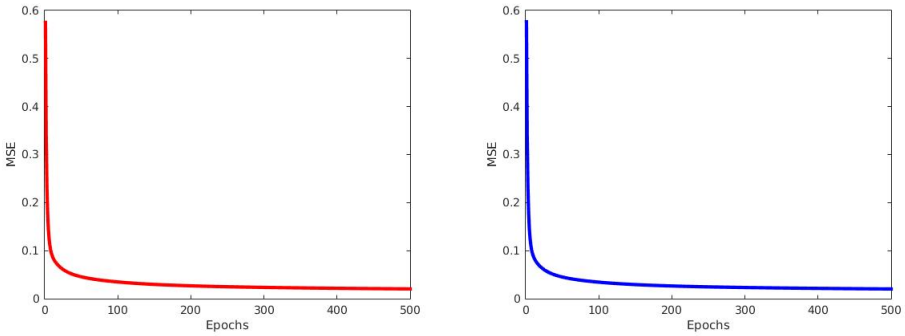

Figure 4.17: Evolution of the mean square error over epochs with a learning rate of $\eta = 0.2$. Left: Training set (80% randomly chosen from all sets), right: Testing set (20% randomly chosen from all sets).

Figure 4.17 shows the convergence of the network trained on all materials over the epochs. For this experiment, in the random pooling process for the batch based training approach an equal amount of samples from each material type was pooled. The performance of the network after the training process for each material type individually can be seen in Table 4.8.

A portion of the so learned mapping between intensity vectors and RGB values for each material type can be seen in Fig. 4.18.

4.4.2 Experimental Results On Real Light Field Data

As there is no label information of the true color or albedo (i.e. color without any shading components), the label, or desired output, for the light field data was approximated by averaging the color intensities over the views.

Dataset	MSE
Lambertian	0.0685
g025r025	0.0662
g025r075	0.0656
g075r025	0.0626
g075r075	0.0610
glossy	0.0688

Table 4.8: MSE of each individual whole dataset applied to the network

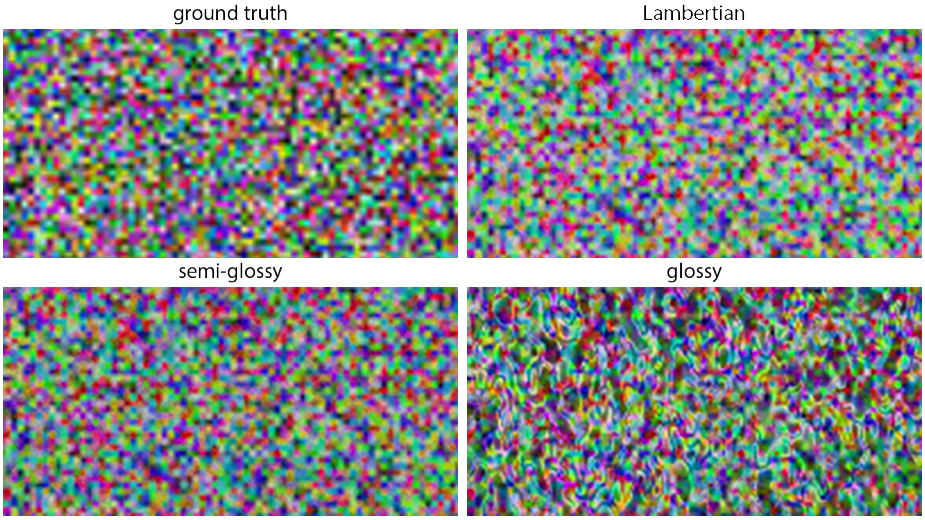

Figure 4.18: From left to right top to down: ground truth colormap, estimated colormap of the Lambertian material, estimated colormap of a semi-glossy material (r075g025) and estimated colormap of the glossy material.

Figure 4.19 shows the approximated color label and the network prediction of two different datasets side-by-side. It shows that while the overall prediction of the network does not fail, the colors (especially for the coin dataset) look washed out, or too bright.

Again through empirical studies it has been shown that the color of the synthetically created training dataset (uniformly random sampling over the color space ,as described in Section 4.1) is probably not the best approach for this specific task. Generating the label data in this fashion leads to many colors that are not very likely to appear in real life objects, while naturally more often appearing color information (white, black, gray, silver etc.) are under-represented. Figure 4.18 shows that there are alot of “vibrant” colors (like pink and violet) while there are not that many darker colors. This is also the reason why the second shown dataset (braille packaging) has a better prediction compared to the coin dataset, as its colors is closer to the random uniform texture pattern of the synthetic data (both qualitatively and quantitatively). Table 4.9 shows the MSE of the prediction in respect to the average intensities of both datasets.

One more experiment that was conducted in this context was to decrease the whole

Figure 4.19: f.l.t.r: average intensities over views of the coin-figural data, network prediction of the coin-figural data, average intensities over views of the braille data, network prediction of the braille data

Dataset	MSE
Coin-figural	0.0421
Packaging-Braille	0.0161

Table 4.9: MSE of each individual whole dataset applied to the network

brightness of the albedo label data as seen in Eq. 4.20.

$$Albedo\_new = Albedo\_old \cdot 0.7 \tag{4.20}$$

This experiment was an attempt to see if the washed out colors of the prediction are due to the average brightness of the pixels presented to the network in the training phase. This new albedo label was used to re-train the network and afterwards the same datasets were applied to the network. The result can be seen in Fig. 4.20

Not surprisingly, this improved the result on the coin-figural dataset while decreasing the result on the Packaging-Braille dataset.

Figure 4.20: f.l.t.r: average intensities over views of the coin-figural data, network prediction of the coin-figural data using darker labels, average intensities over views of the braille data, network prediction of the braille data using darker labels

Dataset	MSE
Coin-figural	0.0135
Packaging-Braille	0.0746

Table 4.10: MSE of each individual whole dataset applied to the network

As this was just thought to be a quick try-out, or possible further use-case of the created data, no more experiments were conducted. This proof of concept implementation has shown that the network converges and using the approximated color of the real light-field datasets, that the prediction is not bad as is. However for a performance that maybe is even better than the average over the views, one would have to think more carefully about the training dataset in regards to color information and brightness (and maybe even some more aspects such as gamma etc.).

5 Conclusion and Future Work

This project showed a neural network based machine learning approach in order to learn a mapping between intensity vectors (i.e different illumination angles) of points with different reflectance properties to a surface normal gradient. It was shown that in the presented approach it was not needed to know the position and direction of the light source as well as not needing any spatial information while still having better results for most material types than the reference implementation of the standard photometric stereo approach (L.m.L. and L.m.a.). The network was tested on synthetically generated data and showed that the implementation presented in this project works well even for very glossy surface properties. In the conducted simulations the train error converges very fast which suggests that the network did not yet reach the absolute best accuracy possible and increasing the number of features as well as training the network for longer may still increase the overall prediction of the multilayer perceptron.

However as the qualitative result on real light field data of the network prediction seemed to have degraded gradients, a new training data set was created using the acquisition of the multi-line scan light field camera with assumed geometry (geometry of a perfect sphere). This lead to a quite smaller dataset than with the synthetic approach (around 47K data samples versus around 3 million data samples), however it still improved the prediction result for real light field data.

Herein probably lies the biggest possibility for improvement by creating a larger and more diverse (in respect to material types) dataset for training.

Another improvement that could in the future be tried is the use of convolutional layers instead of the single pixel pooling approach. The neighbourhood information may prove to be advantageous for this task.

Another tried experiment was to do per-pixel classification of the materials. This proved to not work satisfactory, most probably because the intensity profile of one pixel is an extremely weak classifier especially since the intensity profile of some material types are very similar. A convolutional neural network (CNN) would give more information and could work better for this task. However as objects can consist of more than one material

type, one would need to carefully think about the network structure.

The last experiment that was to learn a mapping of the intensity profiles to the albedo or colormap (color without reflective properties). This worked well however there seemed to be a bias in the albedo trainings data, as colors seemed washed out in the prediction. This improved by changing the brightness of the training labels and re-training the network. Again in the future one may think more carefully about how to build a proper training dataset for this task.

The mean absolute error (abbreviation MAE) can be advantageous as it is more robust against outliers [21], however since strong outliers were excluded manually in the datasets beforehand MAE was not tried in this project. However exploring this cost function in the future could also prove advantageous.

A

Figures

This chapter contains figures of the results of the three different approaches (L.m.L., L.m.a. and neural network) trained on synthetic data and afterwards tested on each of the six material types individually.

Fig. A.1 shows the ∇x prediction of the L.m.L approach (as described in Sec. 4.2.2) of the same area of the synthetic datasets as in Fig. 4.5. Fig. A.4 shows the same as Fig. A.1, only using the L.m.a. approach (as described in Sec. 4.2.2) instead of L.m.L. and finally Fig. A.7 shows the result of the neural network. Fig. A.2 shows the absolute error between the ground truth and the L.m.L predictions of each material type individually. Again Fig. A.5 shows the same only using the L.m.a approach and finally Fig. A.8 shows the absolute error of the neural network. Fig. A.3 shows a correlation plot of 1000 randomly sampled points between the predicted label by the L.m.L. approach and the ground truth label. Fig. A.6 shows the correlation plot of the same 1000 samples using the L.m.a. approach and Fig. A.9 of the neural network.

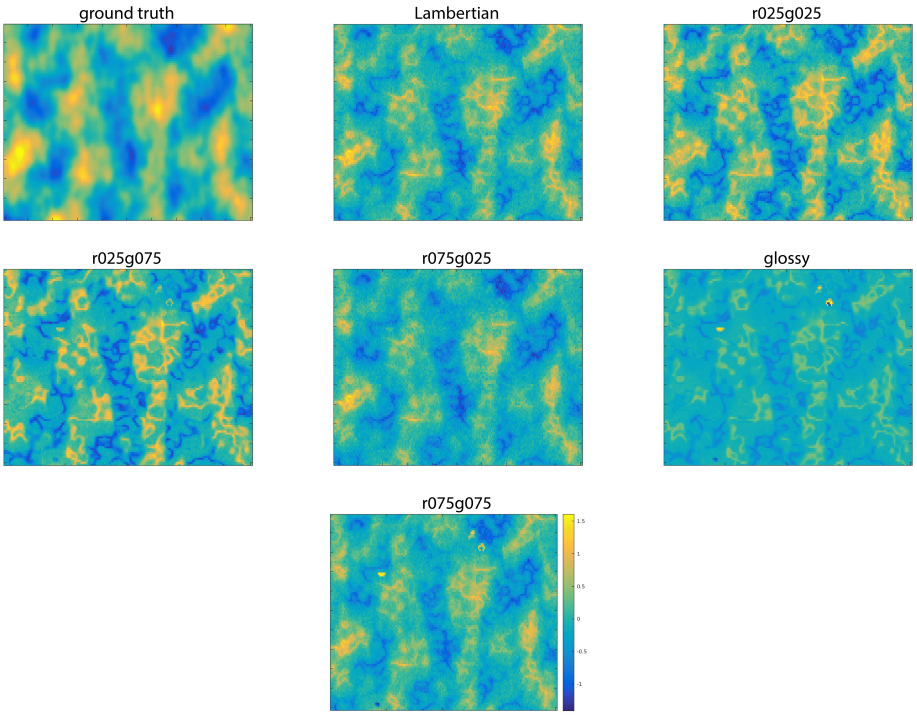

Figure A.1: From left to right: ground truth, predicted ∇x using the L.m.L. approach for: Lambertian, r025g025, r025g075, r075g025, glossy, r075g075 dataset

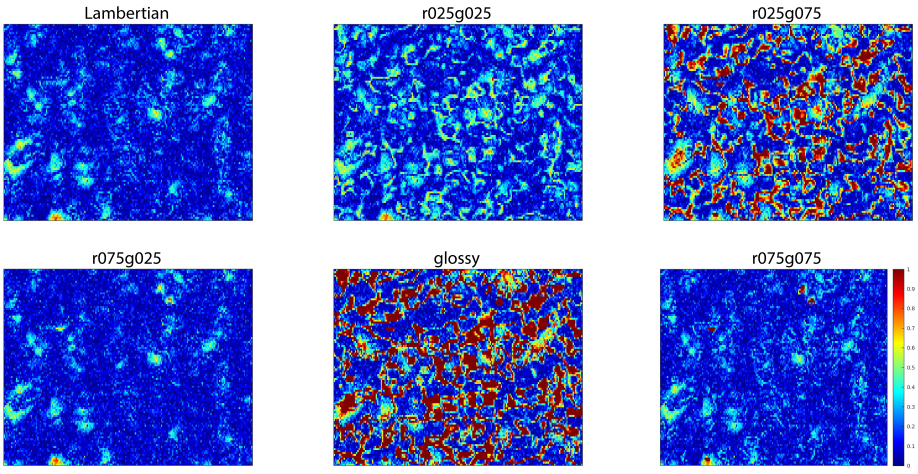

Figure A.2: From left to right, absolute error between ∇x of the prediction using the L.m.L approach and the ground truth label for: Lambertian, r025g025, r025g075, r075g025, glossy, r075g075

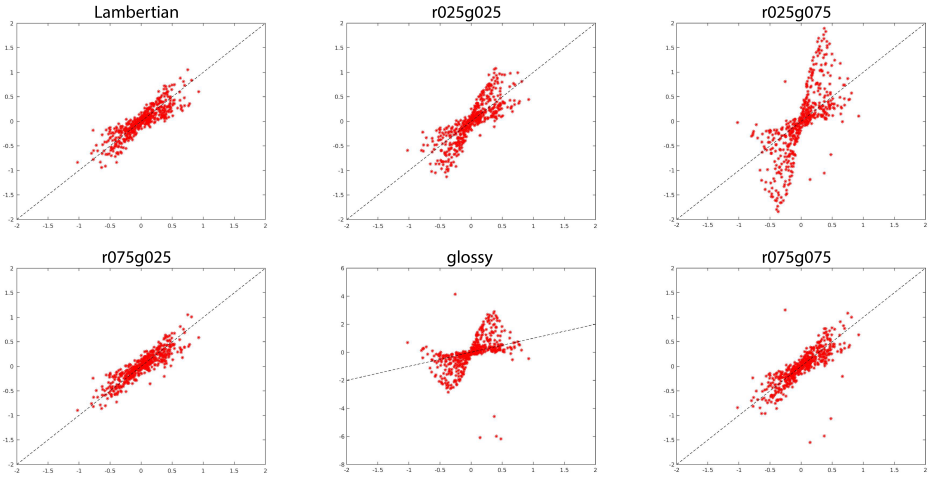

Figure A.3: From left to right, correlation plot between predicted ∇x of 1000 randomly sampled points using the L.m.L. approach and the ground truth for: Lambertian, r025g025, r025g075, r075g025, glossy, r075g075 dataset

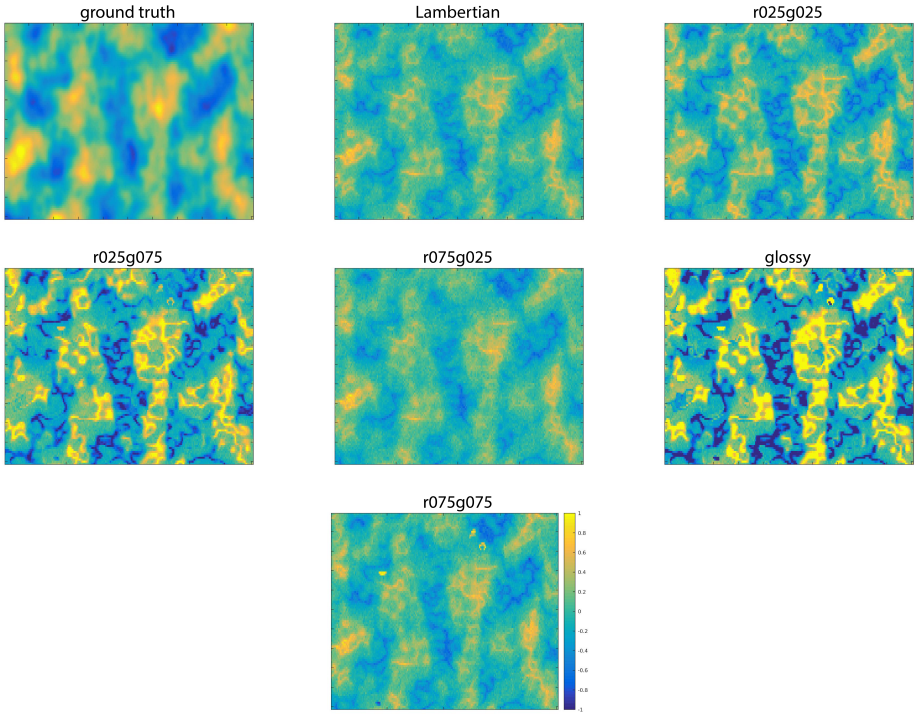

Figure A.4: From left to right, ground truth, predicted ∇x using the L.m.a. approach for: Lambertian, r025g025, r025g075, r075g025, glossy, r075g075 of the L.m.a. approach

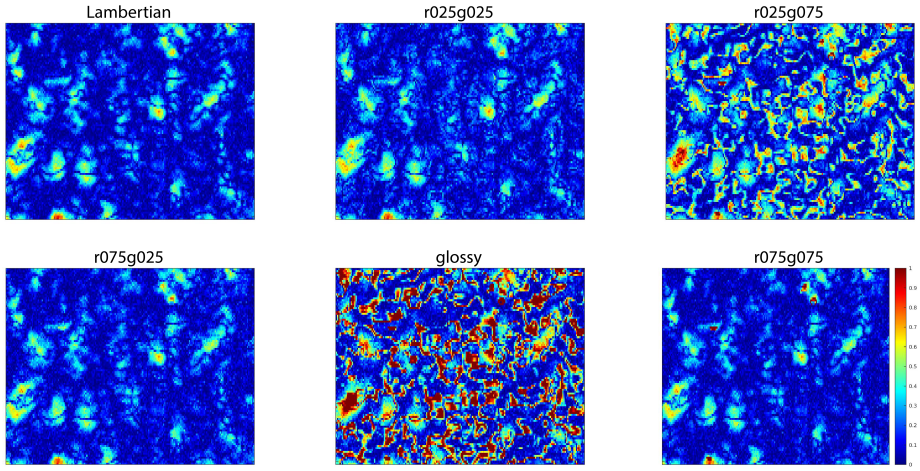

Figure A.5: From left to right, absolute error between ∇x of the prediction using the L.m.a approach and the ground truth for: Lambertian, r025g025, r025g075, r075g025, glossy, r075g075

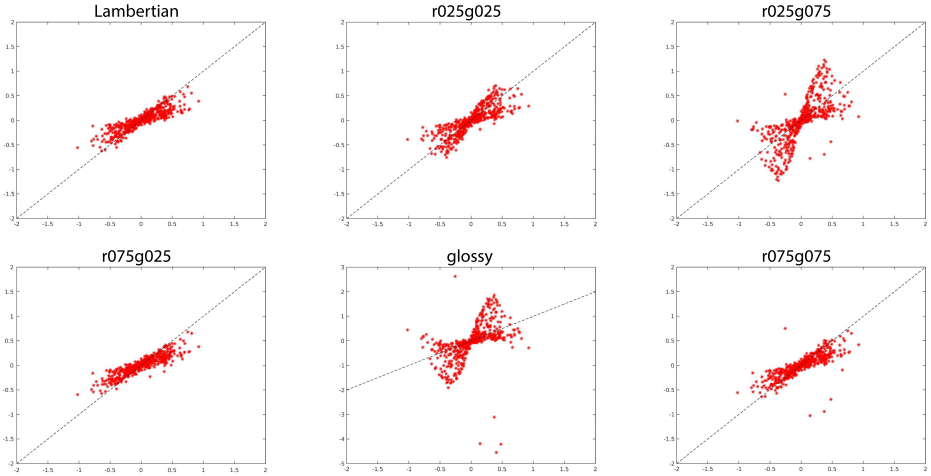

Figure A.6: From left to right, correlation plot between predicted ∇x of 1000 randomly sampled points using the L.m.a. approach and the ground truth for: Lambertian, r025g025, r025g075, r075g025, glossy, r075g075

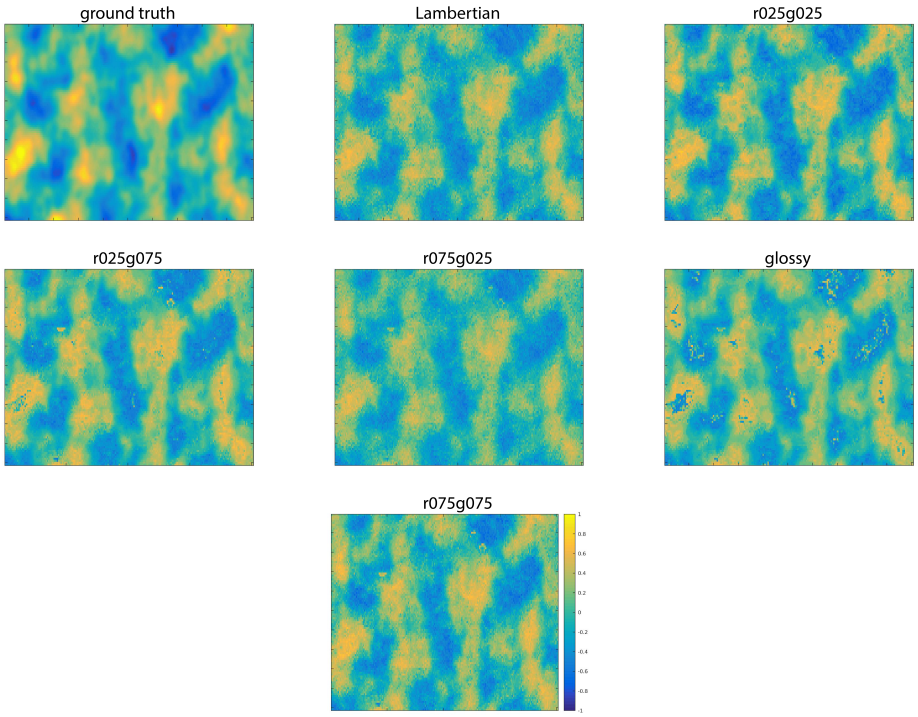

Figure A.7: From left to right, ground truth, predicted ∇x using the trained neural network for: Lambertian, r025g025, r025g075, r075g025, glossy, r075g075

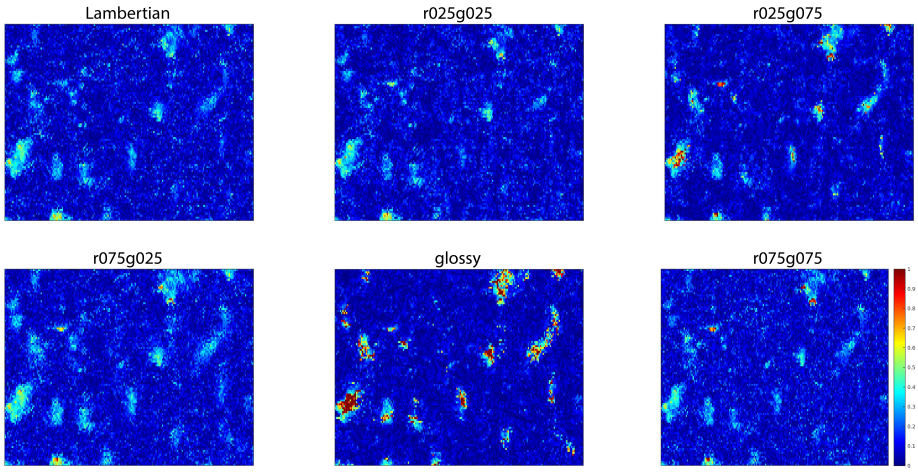

Figure A.8: From left to right, absolute error between ∇x of the prediction of the trained neural network for: Lambertian, r025g025, r025g075, r075g025, glossy, r075g075

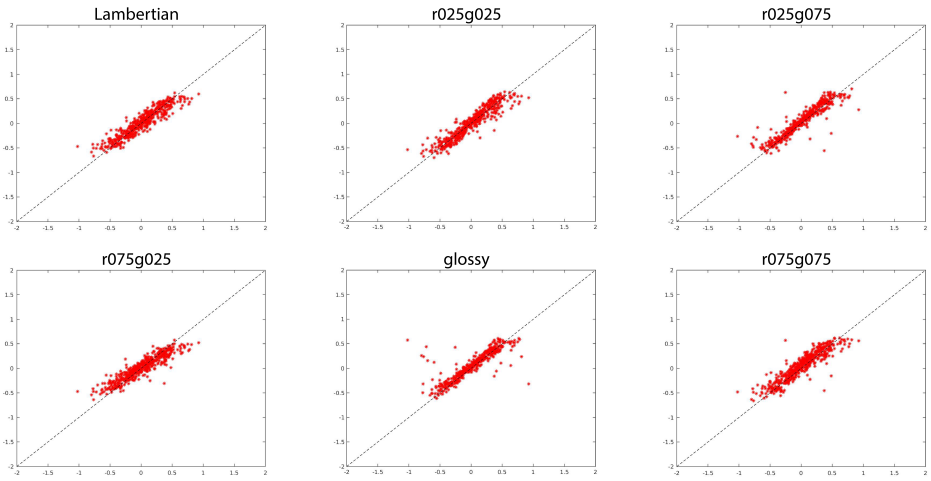

Figure A.9: From left to right, correlation plot between predicted ∇x of the trained neural network of 1000 randomly sampled points for: Lambertian, r025g025, r025g075, r075g025, glossy, r075g075

B

List of Publications

My work at the Institute for Computer Graphics and Vision and AIT led to the following peer-reviewed publications. For the sake of completeness of this Thesis, they are listed in chronological order along with the respective abstracts.

B.1 2017

Photometric Stereo in Multi-Line Scan Framework under Complex Illumination via Simulation and Learning

Dominik Hirner, Svorad Štolc and Thomas Pock
In: *Proceedings of OAGM-ARW Joint Workshop*
May 10-12, 2017, Vienna, Austria
(Accepted for a presentation)

Abstract: This paper presents a neural network implementation of photometric stereo formulated as a regression task. Photometric stereo estimates the surface normals by measuring the irradiance of any visible given point under different lighting angles. Instead of the traditional setup, where the object has a fixed position and the illumination angles changes around the object, we use two constant light sources. In order to produce different illumination geometries, the object is moved under a multi-line scan camera. In this paper we show an approach where we present a multi-layer perceptron with a number of intensity vectors (i.e. points with constant albedo under different illumination angles) from randomly chosen pixels of six materials with different reflectance properties. We train it to estimate the gradient of the surface normal along the transport direction of the given point. This completely eliminates the need of knowing the light source configuration while still remaining a competitive accuracy even when presented with materials which have non-Lambertian surface properties. Due to the random pooling of the pixels our implementation is also independent from spatial information.

Bibliography

[1] Antensteiner, D., Štolc, S., and Huber-Mörk, R. (2016). Depth estimation using light fields and photometric stereo with a multi-line-scan framework. *Proceedings of Austrian Association for Pattern Recognition (AAPR)*, 24(3). (page 31)

[2] Antensteiner, D., Štolc, S., Valentín, K., Blaschitz, B., Huber-Mörk, R., and Pock, T. (2017). High-precision 3d sensing with hybrid light field & photometric stereo approach in multi-line scan framework. (page 31, 38)

[3] Basri, R. and Jacobs, D. W. (2003). Lambertian reflectance and linear subspaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(2):218–233. (page 39)

[4] Blender Online Community (*Blender - a 3D modelling and rendering package*. Blender Foundation, Blender Institute, Amsterdam. (page 35)

[5] Bolles, R. C., Baker, H. H., and Marimont, D. H. (1987). Epipolar-plane image analysis: An approach to determining structure from motion. *International Journal of Computer Vision*, 1(1):7–55. (page 34)

[6] Butler, D. J., Wulff, J., Stanley, G. B., and Black, M. J. (2012). A naturalistic open source movie for optical flow evaluation. In *European Conference on Computer Vision*, pages 611–625. Springer. (page 6)

[7] Cheney, N., Clune, J., and Lipson, H. (2014). Evolved electrophysiological soft robots. In *ALIFE*, volume 14, pages 222–229. (page 11)

[8] Cheng, W.-C. (2006). Neural-network-based photometric stereo for 3d surface reconstruction. In *Neural Networks, 2006. IJCNN'06. International Joint Conference on*, pages 404–410. IEEE. (page 34)

[9] Cho, S.-Y. and Chow, T. W. (1999). Shape recovery from shading by a new neural-based reflectance model. *IEEE Transactions on Neural Networks*, 10(6):1536–1541. (page 34)

[10] Corucci, F., Cheney, N., Lipson, H., Laschi, C., and Bongard, J. (2016). Evolving swimming soft-bodied creatures. In *ALIFE XV, The Fifteenth International Conference on the Synthesis and Simulation of Living Systems, Late Breaking Proceedings*, page 6. (page 11)

[11] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159. (page 52)

[12] Gastaldo, F. (2012). How to make your own physically correct shading. Blender Converence. (page 36)

[13] Gershun, A. (1936). The light field. moscow. *Journal of Mathematics and Physics*, 18. (page 7)

[14] Ghaheri, A., Shoar, S., Naderan, M., and Hoseini, S. S. (2015). The applications of genetic algorithms in medicine. *Oman medical journal*, 30(6):406. (page 11)

[15] GmbH, R. (2018). Raytrix- 3d light field camera technology. (page 33)

[16] Goldman, D. B., Curless, B., Hertzmann, A., and Seitz, S. M. (2010). Shape and spatially-varying brdfs from photometric stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(6):1060–1071. (page 32)

[17] Hayakawa, H. (1994). Photometric stereo under a light source with arbitrary motion. *JOSA A*, 11(11):3079–3089. (page 34)

[18] Heath, T. L. (1921). *A history of Greek mathematics*, volume 1. Clarendon. (page 1)

[19] Hinton, G. E., Sejnowski, T. J., and Ackley, D. H. (1984). *Boltzmann machines: Constraint satisfaction networks that learn*. Carnegie-Mellon University, Department of Computer Science Pittsburgh, PA. (page 13)

[20] Horn, B. K. and Brooks, M. J. (1989). *Shape from shading*. MIT press. (page 34)

[21] Hyndman, R. J. and Koehler, A. B. (2006). Another look at measures of forecast accuracy. *International journal of forecasting*, 22(4):679–688. (page 60)

[22] Im, S., Jeon, H.-G., Ha, H., and Kweon, I. S. (2015). Depth estimation from light field cameras. In *Ubiquitous Robots and Ambient Intelligence (URAI), 2015 12th International Conference on*, pages 190–191. IEEE. (page 32)

[23] Janoch, A., Karayev, S., Jia, Y., Barron, J. T., Fritz, M., Saenko, K., and Darrell, T. (2013). A category-level 3d object dataset: Putting the kinect to work. In *Consumer Depth Cameras for Computer Vision*, pages 141–165. Springer. (page 33)

[24] Kimura, M. and Saito, H. (2001). 3d reconstruction based on epipolar geometry. *IEICE TRANSACTIONS on Information and Systems*, 84(12):1690–1697. (page 45)

[25] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980. (page 52)

[26] Lambert, J. (1760). *Photometria sive De mensura et gradibus luminis, colorum et umbrae*. Sumptibus viduae Eberhardi Klett, typis Christophori Petri Detleffsen. (page 2, 4, 6)

[27] Lu, F., He, L., You, S., Chen, X., and Hao, Z. (2017). Identifying surface brdf from a single 4-d light field image via deep neural network. *IEEE Journal of Selected Topics in Signal Processing*, 11(7):1047–1057. (page 33)

[28] Mazur, M. (2015). A step by step backpropagation example. `https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/`. (page 18)

[29] Narihira, T., Maire, M., and Yu, S. X. (2015). Direct intrinsics: Learning albedo-shading decomposition by convolutional regression. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2992–2992. (page xvii, 5)

[30] Nehab, D., Rusinkiewicz, S., Davis, J., and Ramamoorthi, R. (2005). Efficiently combining positions and normals for precise 3d geometry. In *ACM transactions on graphics (TOG)*, volume 24, pages 536–543. ACM. (page 34)

[31] Ng, R., Levoy, M., Brédif, M., Duval, G., Horowitz, M., and Hanrahan, P. (2005). Light Field Photography with a Hand-Held Plenoptic Camera. Technical report. (page 8)

[32] Nicodemus, F. E., Richmond, J., Hsia, J., Ginsberg, I., and Limperis, T. (1992). Geometrical considerations and nomenclature for reflectance. In *Radiometry*, pages 94–145. Jones and Bartlett Publishers, Inc. (page 4)

[33] Oren, M. and Nayar, S. K. (1994). Generalization of lambert's reflectance model. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 239–246. ACM. (page 36)

[34] Peace, I. C., Uzoma, A. O., and Ita, S. A. (2015). Effect of learning rate on artificial neural network in machine learning. In *International Journal of Engineering Research and Technology*, volume 4. IJERT. (page 39)

[35] Pedersen, S. J. K. (2007). Circular hough transform. *Aalborg University, Vision, Graphics, and Interactive Systems*, 123:123. (page 44)

[36] Rajaram, K., Parthasarathy, G., and Faruqi, M. (1995). A neural network approach to photometric stereo inversion of real-world reflectance maps for extracting 3-d shapes of objects. *IEEE transactions on systems, man, and cybernetics*, 25(9):1289–1300. (page 34)

[37] Reh, F. J. (2005). Pareto's principle-the 80-20 rule. *BUSINESS CREDIT-NEW YORK THEN COLUMBIA MD-*, 107(7):76. (page 39)

[38] Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386. (page 13)

[39] Shekhovtsov, A., Reinbacher, C., Graber, G., and Pock, T. (2016). Solving dense image matching in real-time using discrete-continuous optimization. *arXiv preprint arXiv:1601.06274*. (page 32)

[40] Štolc, S., Huber-Mörk, R., Holländer, B., and Soukup, D. (2014a). Depth and all-in-focus images obtained by multi-line-scan light-field approach. In *IS&T/SPIE Electronic Imaging*, pages 902407–902407. International Society for Optics and Photonics. (page 27)

[41] Štolc, S., Huber-Mörk, R., Holländer, B., and Soukup, D. (2014b). Depth and all-in-focus images obtained by multi-line-scan light-field approach. In *IS&T/SPIE Electronic Imaging*, pages 902407–902407. International Society for Optics and Photonics. (page 31)

[42] Takimoto, R. Y., de Castro Martins, T., Takase, F. K., and Tsuzuki, M. d. S. G. (2012a). Epipolar geometry estimation, metric reconstruction and error analysis from two images. *IFAC Proceedings Volumes*, 45(6):1739–1744. (page 45)

[43] Takimoto, R. Y., Neves, A. C., de Castro Martins, T., Takase, F. K., and Tsuzuki, M. d. S. G. (2012b). Three dimensional scene reconstruction using epipolar geometry. (page 45)

[44] Tao, M. W., Hadap, S., Malik, J., and Ramamoorthi, R. (2013). Depth from combining defocus and correspondence using light-field cameras. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 673–680. IEEE. (page 33)

[45] Tao, M. W., Srinivasan, P. P., Malik, J., Rusinkiewicz, S., and Ramamoorthi, R. (2015). Depth from shading, defocus, and correspondence using light-field angular coherence. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1940–1948. (page 34)

[46] Tomasi, C. and Kanade, T. (1991). Detection and tracking of point features. (page 32)

[47] Valentın, K., Štolc, S., and Huber-Mörk, R. (2015). Improved cost computation with local binary features in a multi-view block matching framework. In *Proceedings of 10th International Conference on Measurement*, pages 21–24. (page 32)

[48] Walter, B., Marschner, S. R., Li, H., and Torrance, K. E. (2007). Microfacet models for refraction through rough surfaces. In *Proceedings of the 18th Eurographics Conference on Rendering Techniques*, EGSR'07, pages 195–206, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association. (page 36)

[49] Wilburn, B., Joshi, N., Vaish, V., Talvala, E.-V., Antunez, E., Barth, A., Adams, A., Horowitz, M., and Levoy, M. (2005). High performance imaging using large camera arrays. *ACM Trans. Graph.*, 24(3):765–776. (page 8)

[50] Wolff, L. B. (1992). Diffuse reflection (intensity reflectance model). In *Computer Vision and Pattern Recognition, 1992. Proceedings CVPR'92., 1992 IEEE Computer Society Conference on*, pages 472–478. IEEE. (page 4)

[51] Woodham, R. J. (1980). Photometric method for determining surface orientation from multiple images. *Optical Engineering*, 19(1):191139–191139–. (page 6, 31)