Aufbaustudium Informatik

Technische Universität München

Final Thesis

Deep Intrinsic Image Decomposition

Chair for Computer Aided Medical Procedures & Augmented Reality

Author:	Udo Dehm
Examiner:	Prof. Dr. Nassir Navab
Supervisors:	Christian Rupprecht,
	Iro Laina,
	Dr. Federico Tombari
Thesis handed in on:	January 15, 2018

Abstract

This thesis addresses the problem of decomposing a single RGB image into its intrinsic components – the albedo or reflectance layer and the shading layer. We use state of the art deep learning architectures based on fully convolutional neural networks to predict the albedo and shading images in a regression learning task. Our models are designed in such a way that the predictions can be made directly, without post-processing or similar additional steps.

Since the intrinsic image decomposition problem is an ill-posed problem we have to impose some constraints to achieve a successful learning process. We do this by using labeled datasets in a supervised learning task. Therefore we heavily depend on the availability of these datasets. There are two publicly available: the synthetic MPI Sintel dataset and the Intrinsic Images in the Wild dataset which contains real-world scenes.

Once our models have been trained they can be instantly employed and used for real-time predictions on images or videos.

We compare different network architecture styles and loss functions.

On the Sintel dataset we are able to deliver state-of-the-art results comparable or better than privious work. This approach shows that CNN-based models can be successfully trained on this synthetic dataset.

On the Intrinsic Images in the Wild dataset we get less significant results. This seems to be because of the purely data-driven approach and the special construction of the loss function required for this dataset.

Contents

1. Introduction

Every image is described by the scene it shows. The physical characteristics of a scene make each image unique. Two of the most important characteristics are the intrinsic reflectance of objects and the shading in a scene. The reflectance of a scene describes how each image pixel reflects light intrinsically. It measures the albedo of objects. The shading structures in a scene are caused by lightning. Shading basically describes the interaction of surfaces and illumination. To interpret an image correctly, it is important to be able to separate each pixel by its albedo and shading component (see also figure 1.1). Humans can do this task remarkably well. Based on experience, the human visual system can decompose confuse real-world visual information into its components. Even by looking at a single image, humans are able to separate effects of surface reflecatance and scene illumination.

In computer vision, the decomposition of an image into reflectance (albedo[1]) and shading layer is referred to as *intrinsic image decomposition*. The intrinsic image decomposition model assumes, the color image I is the pixel-wise product of the albedo layer A and shading layer S:

$$I = A \cdot S. \tag{1.1}$$

If one assumes that a scene is illuminated by white light (light source that emits nearly all frequencies), the shading layer S is defined as a greyscale image. On the other hand, if a scene is illuminated by a color-dominated light (i.e. a specific frequency band dominates), the shading layer S is often defined as a color image, to account for the different shading contributions in each (RGB) color channel.

The intrinsic image decomposition is a challenging task because it is an ill-posed and under-constrained problem. There are infinite many possibilities to decompose images into its intrinsic components. Besides, it is difficult to generate data, from which the intrinsic image decomposition can be inferred in a supervised learning process (see section 4.1). Therefore, there often have been introduced additional constraints (priors) or additional information in the past to tackle this problem. Usually these methods are limited to a certain field of application and fail if assumptions or requirements do not hold (see chapter 3).

If the decompostion of an image is known, the interpretation of a scene becomes a simpler task. For example, the extraction of the geometry of an object in a scene gets straightforward if the shading of each pixel is isolated. Analogously a segmentation task gets quiet simpler if the reflectance layer is known. Image decomposition can also be usefull for resurfacing objects, material recognition or the extraction of intrinsic color which could be of interest in 3D printing. A follow-up interesting task might be the relighting of scenes.

This work tries to solve the intrinsic image decomposition problem as a purely data driven approach using deep learning. For this, the basics are outlined in chapter 2. Chapter 4

[1] In this thesis, in context of the intrinsic image decomposition task, the terms reflectance layer and albedo layer are used synonymously.

Fig. 1.1.: Intrinsic image decomposition task: An image (left side, [6]) is decomposed into its intrinsic albedo and shading layers. The two pixels X and Y located on the red mini-sofa have clearly the same intrinsic reflectance, because they are made of the same material. However, because of the illumination of the scene and the resulting shades on top of pixel Y their pixel values differ. After decomposing the image [46] into its reflectance and shading layer, it looks like pixels X and Y have (almost) the same values. The residual of X and Y is encoded in the shading layer. The intrinsic image decomposition task defines the input image as the pixel-wise poduct of the albedo and shading layer.

describes the used model architectures and parameters. Finally, in chapter 5 results are presented.

2. Fundamentals

This chapter outlines the basic methods used in this thesis. The main focus is on techniques used in subsequent chapters. Other methods are just mentioned briefly for the sake of completeness.

First, the general principles of artificial neural networks ((A)NNs)[1] are explained. Then the concept of convolutional neural networks (CNNs) is described.

2.1. Artificial Neural Networks

Artifical neural networks are systems of interconnected nodes, also called neurons. They are (loosely) inspired by biological neural networks found in brains of mammals. They basically consist of consecutive layers of neurons connected by weights. In the input layer data is fed into the network. Each data point is fed-in individually and passed through the network layer by layer. If the network has more than one hidden layer it is often called a multilayer perceptron (MLP) and it is referred to as deep learning. A mathematical illustration and explanation of such a feedforward neural network is depicted in figure 2.1.

In a supervised learning task neural networks are trained by inserting training data into the network and comparing the output with labeled ground-truth data. The labeled data is divided into training (used for training different machine learning models), validation (used for choosing the best model, e.g. selecting best parameters) and testing (testing final machine learning model) set. Each input vector $\boldsymbol{x}^n$ is passed through the network. It passes the hidden layers until it reaches the output layer. At each node in the network the weighted sum $a_{j_l}^{(l)}$ is calculated (activation). A (most often) non-linear activation function $z_{j_l}^{(l)} = \phi^{(l)}\left(a_{j_l}^{(l)}\right)$ is applied to this weighted sum.

[1] This work always refers to artificial neural networks, even if this is not explicitly stated. Therefore the term 'artificial' is often omitted.

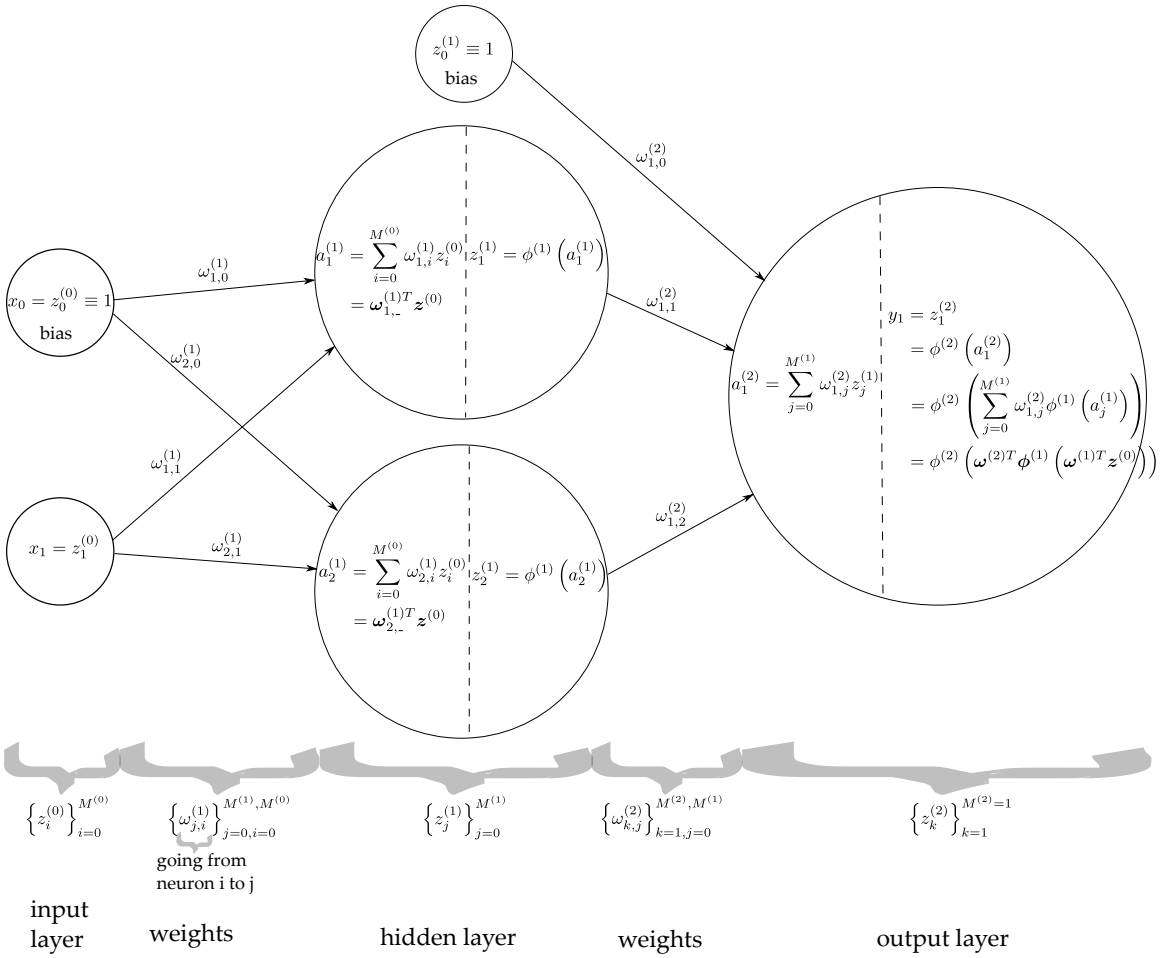

Fig. 2.1.: Mathematically detailed illustration of a simple feedforward neural network with scalar input x_1, one hidden layer and scalar output y_1. Consecutive layers are fully connected, i.e. each neuron $z_i^{(l-1)}$ in layer $l-1$ is connected to all neurons $z_j^{(l)}$ in the subsequent layer l by weights $\omega_{j,i}^{l}$. At each neuron (except neurons in the input layer) the activations $a_j^{(l)}$ are calculated by summing over all weighted neuron outputs of the previous layer. Then a (typically non-linear) activation function $\phi_j^{(l)}$ is applied to the result. In one layer the used activation function is the same. Together with the bias it builds a basis in the function space. Each layer might have a different activation function. In the last layer (output layer) the activation function usually depends on the task which has to be solved: sigmoid activation functions for binary classification tasks, softmax activation functions for multiclass classification problems and linear activation functions for regression tasks.

There are different commonly used activation functions, like sigmoid $\Phi(a) = \frac{1}{1+\exp(-a)}$, rectifier linear units (ReLU, see also [24])

$$\Phi(a) = \max(0, a) = \begin{cases} 0 & \text{for} \quad a < 0 \\ a & \text{for} \quad a \geq 0 \end{cases},$$

tangens hyperbolicus $\Phi(a) = \tanh(a)$ or linear activation functions $\Phi(a) = a$.

In the output layer the predicted output $\boldsymbol{y}^n$ of the neural network is compared to the

(true) label $\boldsymbol{z}^n$. A problem specific loss/cost function gives the residual between the predicted output and the true label. It is calculated from the negative log-likelihood: $E(\boldsymbol{\omega}) = -\ell(\boldsymbol{\omega}; X) = -\ln p_{\boldsymbol{\omega}}(Z|X)$ with underlying model $p_{\boldsymbol{\omega}}(Z|X)$. Depending on the task there are different important loss functions:

- square loss:

$$E(\boldsymbol{\omega}) = \frac{1}{2} \underbrace{\sum_{n=1}^{N}}_{\substack{\text{sum over all} \\ \text{input training} \\ \text{samples}}} (z^n - y^n)^2$$

  derived from regression tasks with the assumption that the true label $\boldsymbol{z}^n$ is a Gaussian distributed random variable with mean $\boldsymbol{y}^n$.

- cross-entropy loss:

$$E(\boldsymbol{\omega}) = -\sum_{n=1}^{N} \underbrace{z^n}_{\in\{0,1\}} \ln \underbrace{y^n}_{\in[0,1]} + (1 - z^n) \ln(1 - y^n)$$

  derived from binary classification tasks with a Bernoulli distribution of the predicted output. The predicted output is mapped (usually with a sigmoid function) on the interval $y^n \in [0, 1]$ so that it represents a voting for a certain class.

- entropy loss:

$$E(\boldsymbol{\omega}) = -\sum_{n=1}^{N} \underbrace{\sum_{k=1}^{K}}_{\substack{\text{sum over} \\ \text{all classes}}} \underbrace{z_k^n}_{\in\{0,1\}} \ln \underbrace{y_k^n}_{\in[0,1]}$$

  derived from standard multi-class classification tasks where a softmax function (normalized exponential function) is used for mapping the predicted outputs to a probability that votes for a class ($\rightarrow y_k^n \in [0, 1]$). Here the true label $\boldsymbol{z}^n$ is given in one-hot encoding.

Now that the difference in the prediction and true label is known, the goal is to minimize this residual. This is equivalent to minimizing the loss function. Since it depends only on the network weights and biases, these parameters have to be optimized. This is the actual part where the network is 'learning'. Typically data is fed into the network in (mini) batches. After all samples of one batch are passed through the NN the weights are updated. In this mini-batch training method we sum the individual sample loss functions over one batch. This method is more memory size efficient than running all data through the network and then updating all parameters and more performance efficient than updating the parameters after a single sample.

The updating of the parameters is done by applying a gradient descent method. All parameters are updated by

$$\boldsymbol{\omega}_{i+1} = \boldsymbol{\omega}_i - \alpha \nabla_{\boldsymbol{\omega}_i} E \tag{2.1}$$

where $-\nabla_{\boldsymbol{\omega}_i} E$ is the summed negative gradient of all the samples in one batch, pointing towards the steepest descent. The learning rate α determines the 'step size' in this direction.

This means one initializes the parameters (weights and biases) of the neural network. This is typically done by generating small random numbers (e.g. random truncated Gaussian distributed numbers) or using parameter values of pre-trained networks. The gradient of E is calculated repeatedly until the weights $\boldsymbol{\omega}_{i+1}$ converge or a termination condition is reached. In detail, the partial derivatives are given by

$$\underbrace{\frac{\partial E}{\partial \omega_{j_l,j_{l-1}}^{(l)}}}_{\substack{\text{derivation with respect to}\\ \text{the parameter that connects}\\ \text{start node } j_{l-1}\\ \text{and destination node } j_l}} \overset{\text{chain rule}}{=} \ldots = \underbrace{z_{j_{l-1}}^{(l-1)}}_{\text{start node}} \cdot \underbrace{\phi^{(l)\prime}\left(a_{j_l}^{(l)}\right)}_{\substack{\text{derivation of the destination}\\ \text{node with respect to its activation}}} \underbrace{\sum_{j_{l+1}=1}^{M^{(l+1)}} \omega_{j_{l+1},j_l}^{(l+1)} \delta_{j_{l+1}}^{(l+1)}}_{\substack{\text{weighted sum over all}\\ \text{errors of next } (l+1) \text{ layer}}}$$

As we can see the errors $\delta_{j_l}^{(l)}$ have to be calculated in backward direction, i.e. starting from the output layer to the input layer. This is because the error in the output layer is known:

$$\delta_{j_L}^{(L)} := \frac{\partial E}{\partial a_{j_L}^{(L)}} \underbrace{=}_{\text{e.g. square loss}} \sum_n |y_{j_L}^n - z_{j_L}^n|.$$

Because of this backward error propagation, one also speaks of error *backpropagation*. A graphical visualization is given in figure 2.2.

Typically the training data has to fed several times to the network. Each forward and backward pass of all the training samples is called one epoch.

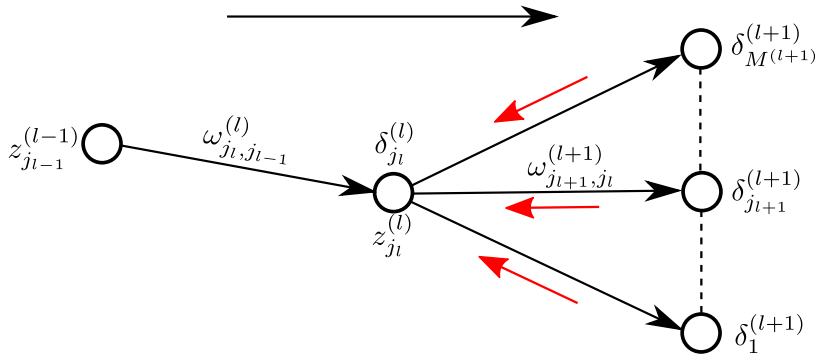

Fig. 2.2.: Illustration of error backpropagation. The error $\delta_{j_l}^{(l)}$ for unit j_l in the hidden layer l is calculated by backpropagation of errors $\delta_{j_{l+1}}^{(l+1)}$ in layer $l+1$. The black arrows donate the direction of information flow during forward propagation, and the red arrows indicate the backward propagation of error information. [8, p. 244]

Nowadays advanced variants of the gradient descent method are used to control the learning rate. For this moving averages of the parameters (momentum) are used. One commonly used optimizer is the Adam optimizer [23].

Neural networks can only learn as long as parameters can be updated. Deep networks are prone to the vanishing gradients. This can be prevented by using regularization methods like dropout, enough labeled data, initializing weights more carefully if possible or standardizing the input.

2.2. Convolutional Neural Networks

Convolutional neural networks (CNNs) are a special kind of neural network. They are designed to efficiently extract information of grid-like inputs like images. For this they exploit properties of images like local contextual interlinkage of pixels located in nearby regions or translational invariance. The architecture of a CNN follows its grid-like input. Neurons in CNNs are arranged in 3D grids (height, width and depth). Otherwise, the basic operation of CNNs is similar to conventional NNs with forward-pass of inputs, applying loss functions and backpropagation of errors by gradient descent optimization methods.

A typical CNN is stacked with *convolutional layers* which are the basic layers in this type of network. Each convolutional layer consists of a number of $F_H \times F_W$ filters (or kernels) which are convolved over the spatial dimensions of the input layer. At each position of the filter the corresponding input elements (pixels) are multiplied with the filter weights in the according depth dimension (e.g. depth 1 filter is applied to depth 1 input). All multiplied pixel-weight pairs are summed up and the bias is added. This results in the corresponding output pixel value. By moving each filter along the spatial dimensions the complete input is scanned. The stride S determines the step size of each filter shift. To have control over the spatial output dimensions one can optionally pad the inputs with zeros. This process is called padding P. The depth of the neuron grid in the output layer is specified by the number of filters. Each filter is always applied over the complete depth D of the previous (input) layer (e.g. the three RGB channels in the input layer). Each filter has $F_H \times F_W \times D$ weights plus 1 bias. Figure 2.3 visualizes the implementation of a convolutional layer.

Another important layer in CNNs is the pooling layer. This layer is typically stacked in between convolutional layers. It reduces the spatial dimensionalities by applying a down-sampling method. Therefore this layer is crucial for the compression of the information space. Again a filter of size $F_H \times F_W$ is moved over the input layer. At each position either the maximum input value within the filter is kept (max-pooling) or the average value (average-pooling). All other values are deleted (which results in a reduction/loss of information). Most commonly the max-pooling operation is used.

After each convolutional layer an activation function is typically applied. This introduces a non-linearity. In CNNs most often ReLUs are used as activation functions.

To train CNNs more efficiently one can use a regularization method called *batch normalization* [21]. Batch normalization implements the normalization directly into the network architecture. For each training mini-batch the distribution of each layer's inputs is normalized. This allows the usage of much higher learning rates and more carelessness with weight initialization.

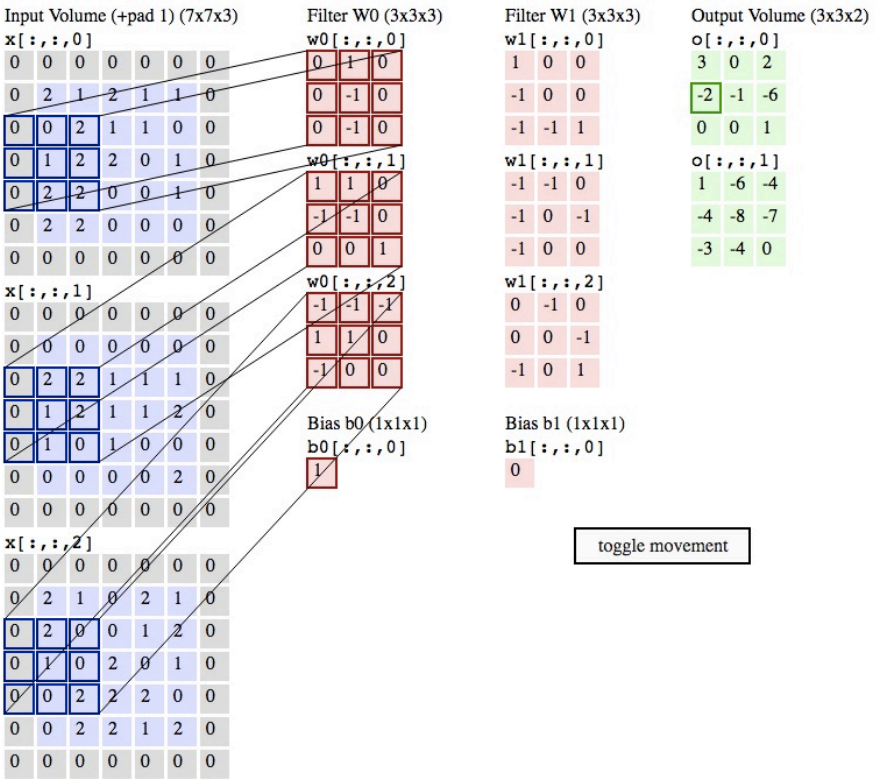

Fig. 2.3.: Illustration of how to apply a convolutional filter. This example visualizes two filters: $K = 2$. Each filter has to have depth $D = 3$ like the inputs. The spatial dimensions of the filters are chosen to be $F_H \times F_W = 3 \times 3$. This totals in 56 parameters for both filters (including the two biases). The output (next layer) has dimensions $H_{\text{new}} = (H+2P-F_H)/S+1$ in height and analogously $W_{\text{new}} = (W+2P-F_W)/S+1$ in width and $D_{\text{new}} = K$ in depth (source: `http://cs231n.github.io/convolutional-networks/`).

In regression tasks (like in the thesis at hand) it is often desirable for the output of the network to have the same size in all dimensions as the input. Since CNNs are typically used to compress these dimensions (this process is called encoding), some methods have been proposed to up-scale the dimensions again (this process is called decoding):

The *unpooling layer* [44] is basically the reverse operation of a pooling layer. On each spatial layer input 'pixel' a filter of size $F_W \times F_H$ is applied. Each pixel is up-scaled to $F_W \cdot F_H$ pixels with all pixels set to zero except one pixel which takes the value of this input pixel.

The *transposed convolutional layer* is related to a convolutional layer. It is often refered to as *deconvolutional layer*, but this is not an appropriate description of its behaviour because it does not invert a convolution operation. A transposed convolutional layer rather reverses the effect of spatial dimension reduction in convolutional layers. This is achieved by applying convolutional operations in combination with some fancy padding. Often zeros are padded around each spatial input pixels (opposed to padding around the complete input like in conventional convolutional layers) which results in increasing spatial dimensions. This makes transposed convolutions a suitable tool for decoders.

Another up-scaling method was introduced by Laina et al. [27]. Their goal was to make the up-scaling process more efficient than using unpooling layers. In unpooling layers a lot of computing time is wasted for matrix multiplications with elements which are equal to zero. Their *up-convolution* and *up-projection* blocks are explained and visualized in figure 2.4.

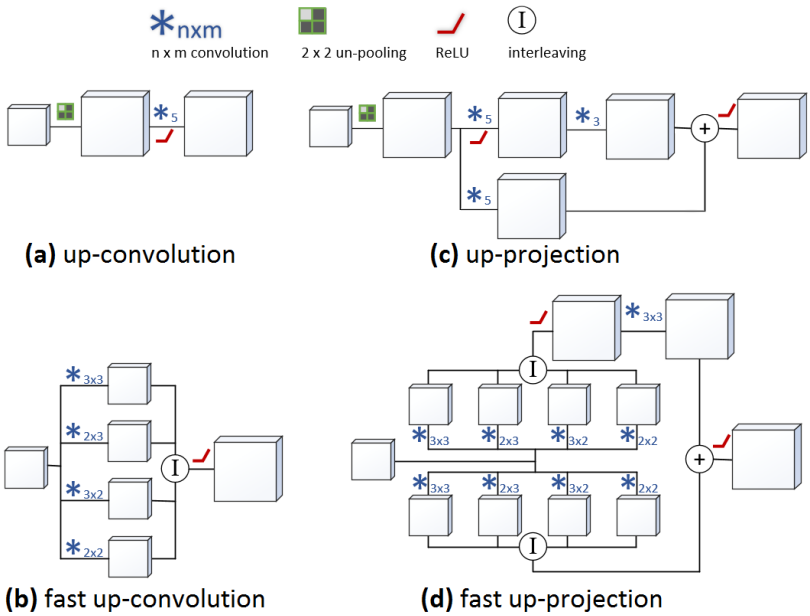

Fig. 2.4.: Up-convolutions and up-projections: (a) Standard up-convolution with unpooling layer followed by a convolutional layer and ReLU activation function. (b) The equivalent but faster up-convolution implementation which consists of carefully designed convolutional layers which are connected in parallel. The elements of the four resulting feature maps are then interleaved. (c) The up-projection block is an extension of the up-convolution block (a). A 3×3 convolution is applied after the up-convolution and a projection connection from the lower resolution feature map to the result is introduced. This architecture allows information to be more efficiently passed forward in the network. (d) The faster equivalent version of (c) replacing the unpooling layer. [27]

3. Related Work

One of the earliest work regarding image decomposition was done by Land and McCann in 1971 [28]. They formulated their so called Retinex algorithm, which stated that large discontinuities in image intensity are due to changes in reflectance, while all other variations are mostly due to changes in shading. This is basically a prior which is imposed on a given scene. Since then there has been done quite some work using different, mostly physical or statistical inspired priors:

Barron and Malik introduced over several studies [3, 2, 5] a piece-wise constant prior of albedo which accounts for the assumption that albedo and material changes are correlated. Garces et al. [13] investigated a clustering of similar reflectance areas. This lead to a chromaticity segmentation which was used to segregate these areas using a linear system. A similar idea was pursued by assuming the same reflectance if chromaticities of neighbouring pixels are about the same [39]. Another method tried to model reflectance values as being drawn from a sparse set of basis colors [35, 37]. Others trained classifiers on local grayscale patterns and derived information about albedo and shading layers [43, 42]. Further methods include introducing priors on texture statistics [34, 30] and non-local texture constraints [40, 45]. Bell et al. [6] used dense conditional random fields to consider long-range material relations. The idea behind this is that objects in one picture often share the same reflectance properties like a painted wall spanning an entire image.

In general, methods based on priors work well in ranges where the assumed priors apply. However, beyond of thes ranges the decomposition usually becomes less accurate or fails completely.

To overcome this disadvantage some studies used additional information to achieve an accurate image decomposition. Hauagge et al. [18] for example used additional input images of the same scene with varying, unknown illumination, while other studies used additional input images from different angles [17, 26, 25]. Bousseau et al. [9] tried to interact with users and imposed assumptions on local reflectance distributions. RGB-D imagery is another method to find significant intrinsic image decompositions [4, 11].

Obviously, these methods involve some effort and are often rather impractical.

In recent years, with the raise of deep learning, purely data-driven methods are increasingly used. The idea is to use labeled datasets with some kind of ground truth intrisic image decompositions to learn the task of decomposing images into albedo and shading layers. Section 4.1 deals with the description and availability of such datasets in more detail. Here we just mention relevant publications that have used these datasets for image decomposition:

Zhou et al. [46] used the Intrinsic Images in the Wild (IIW) dataset (see section 4.1.2) and a CNN to learn a data-driven prior. With that prior they were able to predict relative reflectance ordering between image patches and integrated this into energy minimization frameworks to get image decompositions.

Narihira et al. [33] performed a multiscale CNN regression with a coarse and a fine-tuned scale on the MPI Sintel Dataset (see section 4.1.1). This network architecture is based on Eigen et al. [12]. Since we use a model based on this architecture in this thesis we illustrate it in figure 3.1.

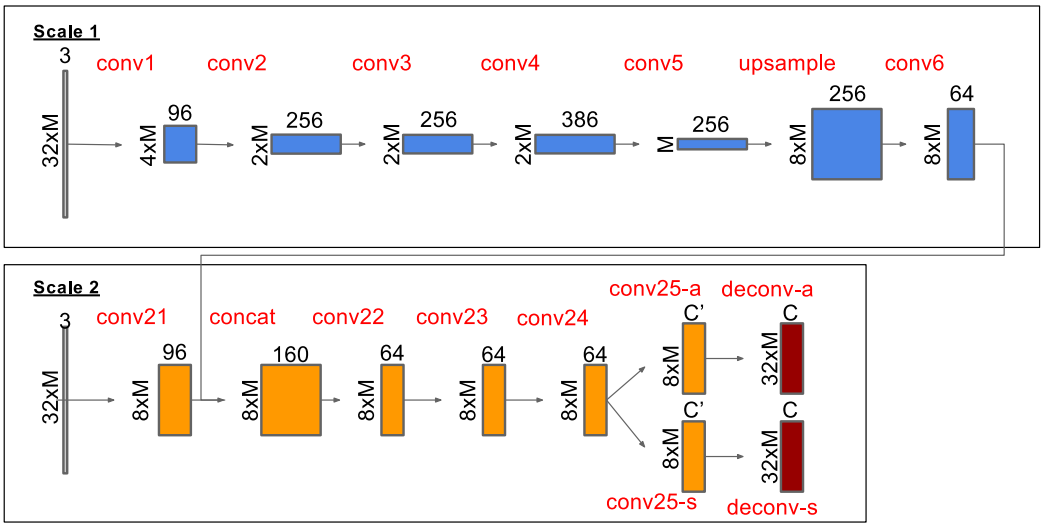

Fig. 3.1.: Network architecture of Narihira et al. [33, p. 2994]. They use as input multiples of 32 and train a coarse network that extracts global information (scale 1). This coarse network (encoder) has the architecture of an AlexNet [24]. They use the output of this subnetwork as an additional input to a finer-tuned network (scale 2). Using deconvolutional layers up-scaling is performed until the dimensions of the albedo and shading outputs match the input image dimensions.

Narihira et al. [32] examined in a subsequent study a similar task. They used the IIW dataset to infer lightness and the perceived reflectance of surfaces using CNNs with fully connected layers. They basically tried to predict the human-labeled relative reflectances.

Finally, there are some studies that use deep learning methods to predominantly learn depth maps. As a byproduct they were able to extract some kind of intrinsic image decompositions [22, 38].

4. Methodology

This chapter describes the models used for intrinsic image decomposition predictions from a single RGB image. Since we formulate the image decomposition task in a purely data-driven supervised learning approach we first take a look at the datasets that are used for training the networks, and their unique features. Then we consider the employed network architectures and loss functions that are to some extent unique and adapted to these datasets.

4.1. Datasets

The problem with the task of decomposing images into their instrinsic albedo and shading components is the availability of meaningful datasets. For a supervised learning approach we need datasets that provide some kind of ground truth reflectance and/or shading labels. Since it is very complicated to provide these labels especially for complex and preferably real-world scenes, there are only two datasets available which are suitable for this task. Furthermore the cardinality of samples has to be large enough in order to train CNN-based models.

4.1.1. MPI Sintel Dataset

The MPI Sintel Dataset [10] is a synthetic dataset generated by the Max Planck Institute for Intelligent Systems in Tübingen. It is based on the open source animated short film Sintel. The dataset provides rendered images and corresponding dense albedo and shading labels which are derived from underlying 3D models (see figure 4.1). The advantage of having dense ground truth albedo and shading labels comes with the cost of not having real-world images.

Overall the dataset contains 840 images from 23 scenes with $20 - 50$ frames each that are usable for training our CNN models. Each RGB image (and its corresponding albedo and shading labels) has dimensions $(436, 1024, 3)$.

Fig. 4.1.: Example of the MPI Sintel Dataset [10]. The dataset provides images and the ground truth albedo and shading labels.

4.1.2. Intrinsic Images in the Wild Dataset

Intrinsic Images in the Wild (IIW) [6] is a large-scale, publicly available dataset. It emerged from the OpenSourfaces dataset [7] and was specifically designed for the task of intrinsic image decomposition. Each image contains crowdsourced annotations of relative reflectance comparisons between pixels. These human based judgements represent a sparse comparison of material properties. Human collaborators have been asked to compare two points. They should determine which point has a darker reflectance. The possible responses were: Point 1 is darker as point 2, point 2 is darker as point 1 and both points have roughly the same surface reflectance (darkness). Furthermore, the confidence of each judgement was monitored. Figure 4.2 illustrates these judgements in more detail.

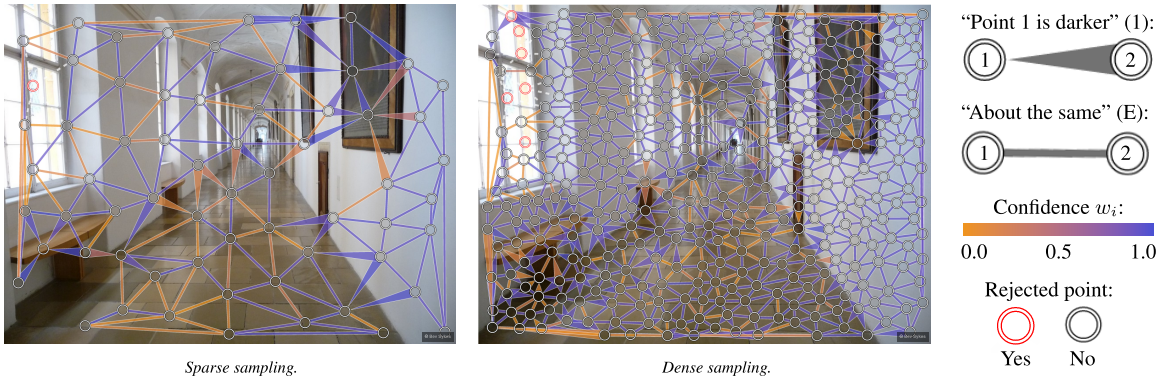

Fig. 4.2.: Sample of the Intrinsic Images in the Wild Dataset [6]. The dataset contains indoor scenes that have pixelwise reflectance comparisons. There are images with sparse sampling and images with a more dense sampling. In order to get a meaningful labeled dataset the task of pixelwise reflectance and accordingly material properties comparison has to be set in a clear unambiguous way. According to the authors, humans can most easily distinguish which point is darker. That is way they ask the question "Which point is darker?" at each comparison. In addition, points located on transparent and opaque surfaces like windows or mirrors are rejected and not used for comparison. The human judgements along with their confidences are visualized by a graph consisting of nodes and edges.

The advantage of the IIW dataset is the availability of real-world images. However, its ground truth is not in the form of actual decompositions, but only relative reflectance judgements over a sparse set of point pairs.

The dataset contains 5230 images of real-world indoor scenes with millions of individual reflectance comparisons. The spatial dimensions of available images vary.

4.2. Network Architectures

In general, the purely data-driven intrinsic image decomposition problem is a standard regression task. This ideal case can be achieved if the appropriate ground truth labels are available. The Sintel dataset provides such labels. It has dense albedo and shading ground truth images which have the same dimensions as the input image. Figure 4.3 shows the basic layout of the network:

The input image is fed into a CNN encoder. It compresses the information space from a

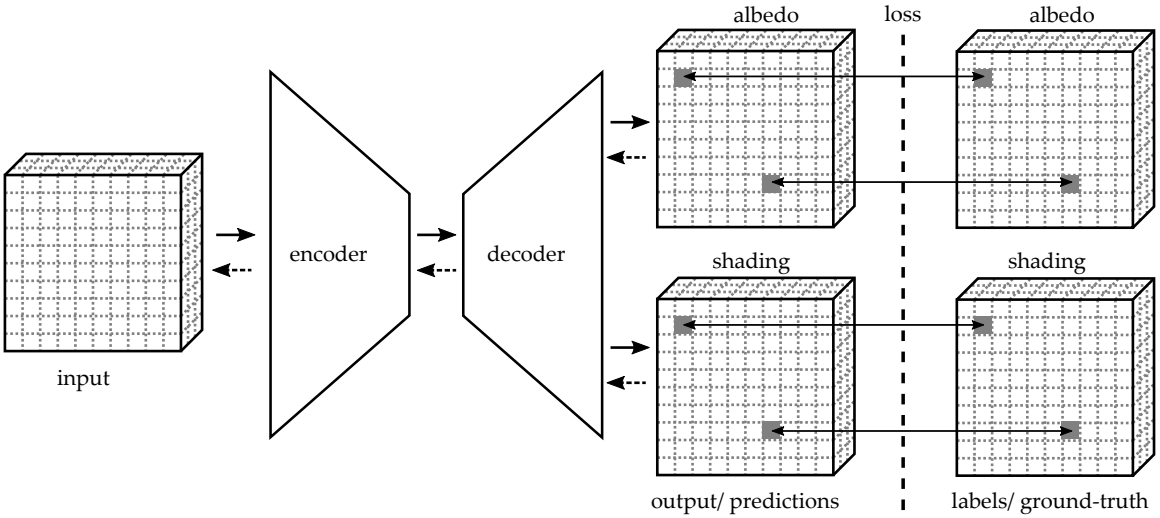

Fig. 4.3.: Basic network architecture for a standard regression task. This architecture style can be employed if dense labels are available, like in the case of the Sintel dataset. A RGB input image is fed into the CNN. It passes the encoder and decoder in the forward pass (solid arrows) and produces the output albedo and shading predictions. A loss function (see section 4.3) compares the predictions and labels pixel by pixel and propagates the error back through the network (dashed arrows). In this process the network is trained by updating its parameters (weights and biases).

complete pixel image with three color channels to a smaller feature space. Information is stored rather in depth (using up to 2048 filters) than in spatial components. The encoder is not a problem specific construct. We can employ commonly used network architectures like AlexNet [24], ResNET [19] or VGG [41]. Since these networks where introduced for image classification tasks and therefore have fully connected layers at the end of a forward pass in their standard versions, we get rid of these layers.

We are interested in a model output that consists of two images – the albedo and the shading image – with the same dimensions as the input image respectively. This means we need to scale our compressed information space up. This is done by the decoder. A decoder typically consists of convolutional layers and some upscaling specific layers like deconvolution layers (i.e. transposed convolution layers), unpooling layers (see chapter 2) or up-projection layers (see figure 2.4). We apply these layers in a way so that we can assure that the input and output images have the same dimensions. All three used decoder architectures are illustrated in figures 4.4 - 4.6.

For learning we apply an appropriate loss function that compares our predicted albedo and shading outputs pixel by pixel. For further details see section 4.3. The weights are updated with a (mini) batch gradient descent method (for the exact implementation details see section 5.1).

In case of the Intrinsic Images in the Wild (IIW) dataset, things get a bit more complicated. Since we want to get the same outputs (albedo and shading image) as the Sintel dataset models, we keep the basic model architecture with its encoder and decoder. In fact, we always train the synthetic Sintel dataset models first and use its trained parameters (weights and biases) for initialization. However, since we do not have dense labels we have

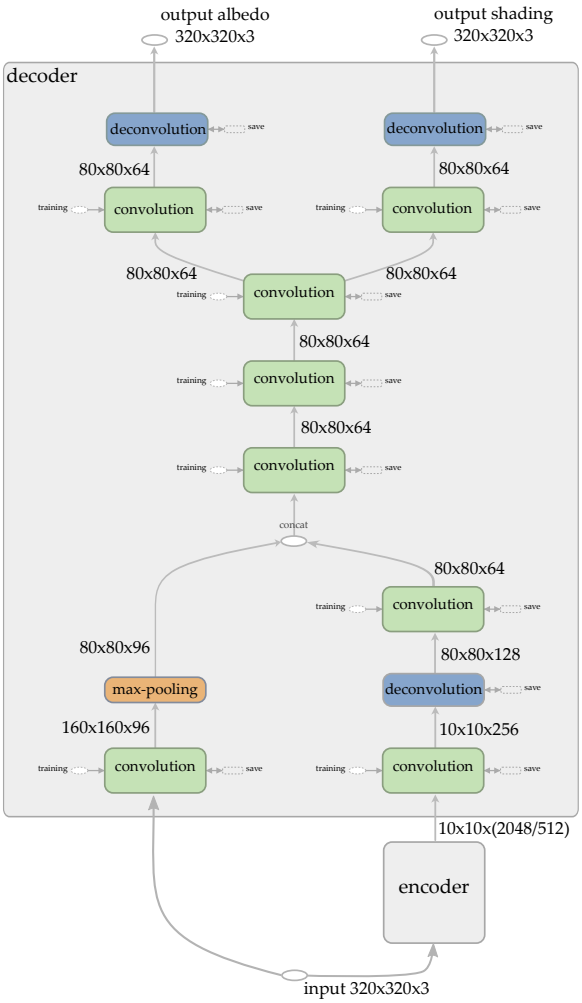

Fig. 4.4.: Architecture of a two-scale decoder. We call this decoder *2scale* decoder. The idea of this architecture style is to extract global contextual information in a sub-network (right path of the network, following the input and encoder trace) and feed its output in a sub-network (left path of the network) that is responsible for fine-tuning. The input and output images have dimensions $320 \times 320 \times 3$. For the encoder architecture we use a ResNET or VGG architecture (see appendix A). The ResNET encoder outputs 2048 feature maps while the VGG encoder outputs 512 ones. The color coding represents same operations. Each convolutional layer node contains the operations convolution, batch normalization and ReLU activation. Deconvolution is used for up-scaling.

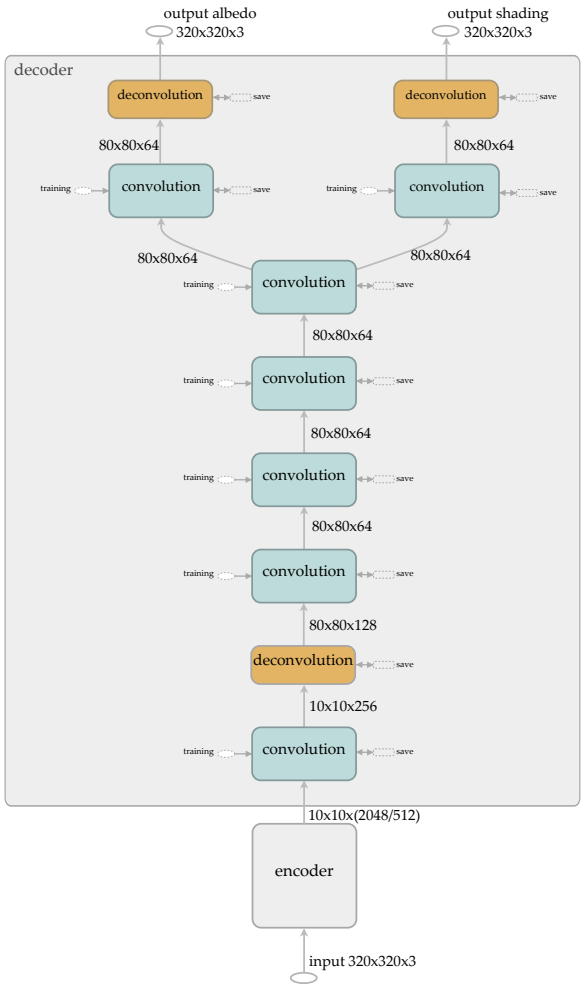

Fig. 4.5.: Architecture of the *deconv* decoder (it uses exclusively deconvolutional (transposed convolutional) layers for up-scaling). This architecture is very similar to the *2scale* decoder (see figure 4.4 for more details) but it uses only one scale for simplicity.

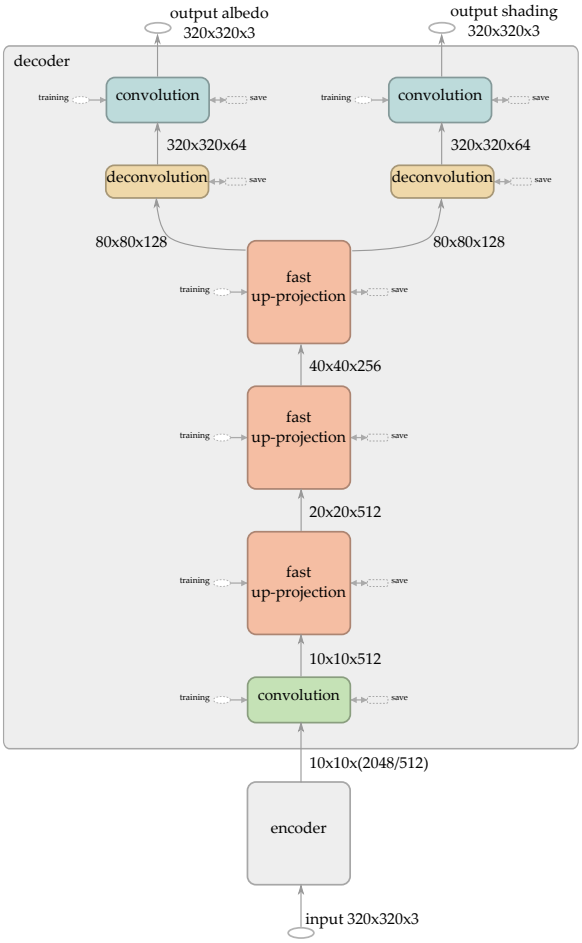

Fig. 4.6.: Architecture of the *upproject* decoder. This decoder predominantly uses fast up-projection layers for up-scaling. After splitting into the output paths we use deconvolutional layers for further up-scaling. All convolutional layers use batch normalization. The green convolutional layer node applies a ReLU activation function while the blue nodes do not.

to adjust the loss function accordingly (see section 4.3). The basic architecture is illustrated in detail in figure 4.7.

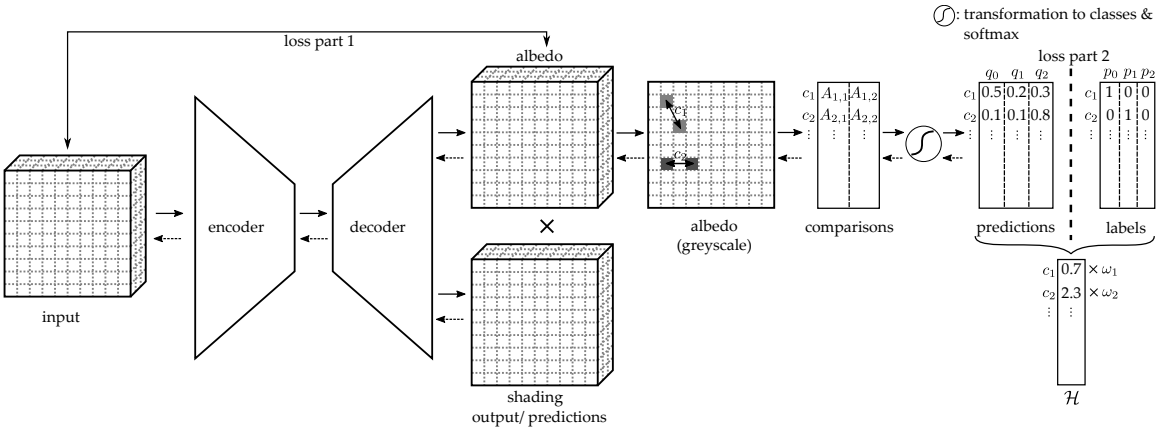

Fig. 4.7.: Basic network architecture for the IIW dataset with its sparse human judgement labels. From input to output the network architecture is the same as in the pure regression task of the Sintel dataset (see figure 4.3). We therefore have the same parameters (weights and biases) in both networks which allows us to train a network on both datasets. The loss function is now composed of two parts. One regression part (loss part 1) and one classification part (loss part 2). The latter is related to the sparse labels in form of pixel comparisons. We have to reshape the albedo output to be able to calculate this part of the loss function. First we have to take the mean over all channels in the albedo prediction. Then we identifiy the pixel which belong to an available label comparison c_i. We transform the comparisons with equations (4.12)-(4.14) and apply the softmax function (4.15). From this we get the prediction for each comparison in a probabilistic way: q_1 is the probability for point 1 beeing darker than point 2, q_2 for point 2 beeing darker than point 1 and q_0 for both points having about the same darkness in the albedo prediction. We compare these predictions with the human judgement labels in OHE form (p_1, p_2, p_0) and calculate the partial loss function (cross-entropy) $\mathcal{H}$. For a detailed analysis see section 4.3. Using backpropagation the error is passed through the network (dashed arrows). In this process the network is trained by updating its parameters (weights and biases).

4.3. Loss Functions

An adequate loss function is essential for a supervised learning task using CNNs. Its correct choice is crucial for a learning success. The given datasets with their ground truths and the learning goal define the form of the loss function. All loss functions are generally calculated over (mini) batches.

Sintel Dataset: The advantage of the synthetic Sintel dataset is that it provides dense albedo and shading labels. This means the ground truth images have exactly the shape of our prediction outputs. That is why we can interpret the intrinsic image decomposition task as a pure regression problem. Let Y_p be the predicted output image (matrix) where Y_p is a placeholder for the albedo prediction A_p or shading prediction S_p, and correspondingly the labels Y_l. Furthermore, we define $Y = Y_l - Y_p$ as the pixelwise difference and $y_{i,j,c}$ as the residual of a single pixel with spatial coordinates (i, j) and channel c. The indices (i, j, c) run over all images in the available batch. All images have the same number of pixels. There are N pixels in a batch. We define three loss functions which we will use for training: First of all we deploy the $\mathcal{L}_1$ loss which is given by

$$\mathcal{L}_1(Y) = |Y| = \frac{1}{N} \sum_{i,j,c} y_{i,j,c} \tag{4.1}$$

Second the $\mathcal{L}_2$ loss is given by the squared euclidean norm between labels Y_l and predictions Y_p. for this norm we follow Narihira et al. [33] and Eigen et al. [12] and optionally apply a regularization term which makes the $\mathcal{L}_2$ loss scale invariant:

$$\begin{aligned} \mathcal{L}_{2,\lambda}(Y) &= ||Y||_2^2 - \lambda \left(\mathcal{L}_1(Y)\right)^2 \\ &= \frac{1}{N} \sum_{i,j,c} y_{i,j,c}^2 - \lambda \frac{1}{N^2} \left(\sum_{i,j,c} y_{i,j,c} \right)^2 \end{aligned} \tag{4.2}$$

With λ we can regulate the scale invariant term. We have a pure $\mathcal{L}_2$ loss for $\lambda = 0$, scale invariant loss when $\lambda = 1$ and a mixture of both when $\lambda = 0.5$.

We also apply a third loss function which is a combination of both the $\mathcal{L}_1$ and $\mathcal{L}_2$ loss – the reverse Huber (BerHu) loss $\mathcal{B}$ [27, 47, 36]. It is given by

$$\mathcal{B}(Y) = \begin{cases} \mathcal{L}_1(Y) & \text{if } |Y| \leq \eta, \\ \frac{Y^2 + \eta^2 J}{2\eta J} & \text{if } |Y| > \eta \end{cases} \tag{4.3}$$

The BerHu loss equals the $\mathcal{L}_1$ loss when $|Y| \leq \eta$ and $\mathcal{L}_2$ when $|Y| > \eta$. It is noteworthy that the BerHu loss is continuous and first order differentiable at η. The point η is defined as the transition where the $\mathcal{L}_1$ loss turns into the $\mathcal{L}_2$ loss. We assign a value $\eta = 0.2 \cdot \max_{i,j,c}(y_{i,j,c})$ i.e. η is defined as 20 % of the maximum residual of the batch. J is an all-ones matrix.

The $\mathcal{L}_2$ loss emphasizes high residuals. In fact, it 'weights' high residuals stronger than smaller ones, while the $\mathcal{L}_1$ loss assures that all residuals are 'weighted' equally and thus smaller ones have a greater impact in $\mathcal{L}_1$ than in $\mathcal{L}_2$ loss.

Of course each final loss function results in the summation of the albedo and shading loss:

$$\mathcal{L}(A, S) = \mathcal{L}(A) + \mathcal{L}(S). \tag{4.4}$$

Here $\mathcal{L}$ is a placeholder for the $\mathcal{L}_1$, $\mathcal{L}_2$ or BerHu loss $\mathcal{B}$.

Which function delivers the best results is discussed in chapter 5.

IIW Dataset: The Intrinsic Images in the Wild dataset misses dense labels. Therefore we have to define a loss function that fits to this dataset and its available pairwise reflectance labels.

We define a loss function that consists of two parts (two summands). The first summand describes the regression problem. Since we do not have albedo and/or shading ground truths, we have to use the definition of the image decomposition (see the introduction, chapter 1, equation (1.1)). We want to calculate the discrepancy between the input image I and the predicted output image given by $A_p \cdot S_p$. For this we will use either the $\mathcal{L}_1$ or $\mathcal{L}_2$ loss:[1]

$$\mathcal{L}_1 = |I - A_p \cdot S_p| \tag{4.5}$$

$$\mathcal{L}_2 = ||I - A_p \cdot S_p||_2^2 \,. \tag{4.6}$$

Using just the $\mathcal{L}_1$ or $\mathcal{L}_2$ loss would not be sufficient because there are infinitely many possibilities of decomposing an image into its intrinsic components. We need to impose some constraints on the loss function. That is where the second summand enters. This part of the loss function considers the human-based pixel comparison labels of the IIW dataset. In the following we explain how we have to extend the loss function:

We start with the "weighted human disagreement rate" (WHDR) introduced in [6] by the authors of the dataset as the basis for our loss function. To get a valid loss function we have to modify the WHDR metric in such a way that it supports backpropagation i.e. it has to be differentiable on its complete domain. We are going to achieve this by reformulating the WHDR metric and transforming it into a classification task. In the following we describe in more detail how we accomplish this.

First, let's have a closer look at the WHDR metric: The WHDR metric presents a numerical way to connect the sparse human judgements J_i to the albedo layer A predicted by our CNN models. Hence the quality of the predicted decomposition is now only measured on the albedo output prediction A and not on the shading predictions S, because we only have ground truth labels for the albedo output in this real-world scenes dataset. In more detail, the WHDR metric measures the percentage of human judgements that a model incorrectly predicts, weighted by the confidence ω_i of each judgement (see also figure 4.2). This means the lower the value of the WHDR metric, the better the decomposition predictions. It is calculated by

$$\mathrm{WHDR}_\delta(J, A) = \frac{\sum_i \omega_i \cdot \mathbf{1}(J_i \neq \widehat{J}_{i,\delta}(\widetilde{A}))}{\sum_i \omega_i}. \tag{4.7}$$

We compare the albedo prediction with human labels. Since humans cannot resolve channels we have to take the mean over all channels to compare the albedo predictions with the judgements. Therefore A represents the mean over the channels of the albedo prediction in the following (see albedo (greyscale) in figure 4.7). The domain of the human judgements J_i is given by $1, 2$ or 0, where 1 means that point 1 is darker, 2 means that point 2 is darker and

[1] For both datasets, Sintel and IIW, we refer the loss functions as $\mathcal{L}_1$ and $\mathcal{L}_2$, even if they are not exactly the same. However, the content should make clear at all times as to which loss function is meant.

0 means that both points are approximately the same color. Because it is relatively difficult for humans to classify two points as the same darkness, the WHDR metric accounts for this by introducing the relative difference δ between two surface reflectances where humans just begin to switch between saying "they are about the same" to "one point is darker". Like suggested in the originial publication [6], the value is set to $\delta = 10\,\%$ in this thesis. The domain of the comparison of the corresponding two pixels in the predicted albedo layer is of course the same. It is calculated by

$$\widehat{J}_{i,\delta}(A) = \begin{cases} 1 & \text{if } A_{2,i}/A_{1,i} > 1+\delta, \\ 2 & \text{if } A_{1,i}/A_{2,i} > 1+\delta, \\ 0 & \text{else,} \end{cases} \tag{4.8}$$

where $A_{1,i}$ is the reflectance at point 1 belonging to the i-th judgement (and accordingly point 2).

If the human judgements J_i disagree with the predicted reflectance comparisons $\widehat{J}_{i,\delta}(A)$, the unit indicator function $\mathbf{1}(\cdot)$ is calculated to 1. The total $\mathrm{WHDR}_\delta(J, A)$ metric is given by the sum over all weights ω_i where the human and predicted judgements disagree divided by all weighted judgements available in an image.

Since this metric is not differentiable, we cannot use it in its current form as a loss function. We have to reformulate it and transform it into a classification problem with classes 1 (point 1 is darker), 2 (point 2 is darker) and 0 (both points are about the same). See also figure 4.7 for a graphical representation of this approach.

First, we rephrase equation (4.8). The goal is to get some kind of probabilities that vote for each class. To get a vote for each class (per judgement i) we need to have suitable functions which give us voting results. We start with the condition for class 1 in equation (4.8) and rewrite it in the following way:

$$A_{2,i} - A_{1,i} > \delta \cdot A_{1,i}$$

Now, we symmetrize the right side of this equation with respect to the participating pixels $A_{1,i}$ and $A_{2,i}$ by replacing $A_{1,i}$ with the mean:

$$\Rightarrow A_{2,i} - A_{1,i} - \delta \cdot \frac{A_{1,i} + A_{2,i}}{2} > 0 \tag{4.9}$$

This transformed inequality differs slightly from the above, but the important part is that it keeps its informative value. Also δ can still be interpreted as the limit where humans say "two points are about the same (darkness)". We do the same thing analogously for class 2:

$$\begin{aligned} A_{1,i} - A_{2,i} &> \delta \cdot A_{2,i} \\ \Rightarrow A_{1,i} - A_{2,i} - \delta \cdot \frac{A_{1,i} + A_{2,i}}{2} &> 0 \end{aligned} \tag{4.10}$$

The advantage of using these equations is that we can formulate an equation for class 0, too. Since we can summarize equations (4.9) and (4.10) in one equation with

$$|A_{1,i} - A_{2,i}| - \delta \cdot \frac{A_{1,i} + A_{2,i}}{2} > 0$$

we immediately know that

$$-|A_{1,i} - A_{2,i}| + \delta \cdot \frac{A_{1,i} + A_{2,i}}{2} > 0 \tag{4.11}$$

is true if class 0 (both points are about the same color) is given. The special thing about equations (4.9)-(4.11) is that they cover the complete domain of two pixel comparisons and only one equation can be greater than zero for a specific input comparison $A_{1,i}, A_{2,i}$.

In summary, we now can write the conditions for each class in the following way:

$$\sigma_{1,i} = A_{2,i} - A_{1,i} - \delta \cdot \frac{A_{1,i} + A_{2,i}}{2} \tag{4.12}$$

$$\sigma_{2,i} = A_{1,i} - A_{2,i} - \delta \cdot \frac{A_{1,i} + A_{2,i}}{2} \tag{4.13}$$

$$\sigma_{0,i} = -|A_{1,i} - A_{2,i}| + \delta \cdot \frac{A_{1,i} + A_{2,i}}{2} \tag{4.14}$$

Each equation can be interpreted as a vote for the corresponding class. Since only one equation can be greater zero, this vote is by far the strongest. In other words, equations (4.12)-(4.14) can be understood as unnormalized log probabilities.

To obtain probabilities (in range $[0, 1]$) which we can compare with the human judgement labels we apply the softmax function to each unnormalized log probability:

$$q_{j,i} = \frac{e^{\sigma_{j,i}}}{\sum_{k=0}^{2} e^{\sigma_{k,i}}}, \quad j \in \{0, 1, 2\}. \tag{4.15}$$

Subsequently, we write the human judgements J_i in One-Hot-Encoding and get the probability for each class $(p_{0,i}, p_{1,i}, p_{2,i})$[2]. With that we can finally calculate the error for each comparison i by calculating the cross-entropy:

$$\mathcal{H}_i = -\sum_{j=0}^{2} p_{j,i} \cdot \log q_{j,i} \tag{4.16}$$

The greater the value of $\mathcal{H}_i$, the higher is the discrepancy between prediction and label.

Finally we take the mean over all judgements in one batch and get the second part of our loss function:

$$\mathcal{H} = \frac{1}{N} \sum_i \mathcal{H}_i \tag{4.17}$$

Alternatively we can weight each cross entropy with the confidence ω_i of each judgement

$$\mathcal{H}_\omega = \frac{1}{N} \sum_i \mathcal{H}_i \cdot \omega_i \tag{4.18}$$

For the total losses we get

$$\mathcal{L}_{1,\lambda,(\omega)} = |I - A_p \cdot S_p| + \lambda \cdot \mathcal{H}_{(\omega)} \tag{4.19}$$

$$\mathcal{L}_{2,\lambda,(\omega)} = ||I - A_p \cdot S_p||_2^2 + \lambda \cdot \mathcal{H}_{(\omega)} \tag{4.20}$$

[2]Notice, since we have distinct labels we always have that $p_{j,i} = 1$ for $j \in \{0, 1, 2\}$ while $p_{k,i} = 0$ for $k \in \{0, 1, 2\}$ and $k \neq j$. Furthermore, it is $p_{0,i} + p_{1,i} + p_{2,i} = 1$.

The regularization coefficient λ is balances the impact of the cross-entropy error relative to the regression error. Appropriate values for λ which deliver the best results are discussed in chapter 5. $\mathcal{H}_{(\omega)}$ with the parenthesized confidence weight ω indicates the two possible used cross-entropy losses. For further analysis of the results see also chapter 5.

5. Experimental Results

In this chapter we give a comprehensive analysis of the results:

5.1. Experimental Setup

5.1.1. Implementation Details

For the implementation of all networks we use the TensorFlow [1] framework. Training is done on a single NVIDIA GeForce GTX TITAN with 12 GB of GPU memory.

We tried to train our models from scratch, but we were not able to get the models to learn. This is most probably due to too small datasets. Therefore we use pre-trained models like ResNet-50 [19] or VGG-16 [41] as encoders (see appendix A). Since both models have been introduced for image classification tasks we adjust them to our problem by ignoring the fully connected layers at the end of each network. For both pre-trained encoders we use the TensorFlow-Slim [1] implementation. The parameters of the newly added decoders are initialized by the "MSRA" weight initialization method [20]. This method is an enhancements of the "Xavier" initialization method [14]. Both names are sometimes used interchangeably in literature. Weights are sampled from a normal distribution with zero mean and a variance which depends on the number of neurons in the layer before the weights n_{in} (fan-in) and the layer after weights n_{out} (fan-out):

$$\omega_{ij} \sim \mathcal{N}\left(0, \frac{4.0}{n_{\text{in}} + n_{\text{out}}}\right) \tag{5.1}$$

This initialization method is suitable for convolutional layers with non-linear activations (like ReLU).

For training we feed the CNN models with images in batches of 16 samples.

We use Adam optimization [23] with an initial learning rate between $5.0{\cdot}10^{-5}$ and $5.0{\cdot}10^{-4}$ depending on the model architecture.

5.1.2. Image Preprocessing

For preprocessing there are some dataset specific operations which are performed to get better, more accurate model predictions. All images and ground truth image labels (if available) are divided pixel by pixel by 256 to normalizing the image pixel values to range $[0, 1]$. This has shown to give the best model training results.

All networks are trained on RGB image patches of size $320 \times 320 \times 3$. Of course the albedo and shading outputs have the same dimensions respectively.

MPI Sintel Dataset: For training we use the provided 'clean pass' images instead of the 'final' images because we do not want any effects like depth of field, motion blur or

fog distract from our application. Some images contain defect pixels due to the image rendering process. These pixels are encoded in invalid pixel masks which are also available in the dataset. Nevertheless, in order to maintain a large dataset we even train our models on images with defect pixels but we do not consider these pixels when calculating the loss functions (see section 4.3).

One disadvantage of the Sintel Dataset is its limited number and variation in training images. Since frames in one scene are quite similar we have to make sure that we shuffle the images before feeding them into our networks. Moreover we split the dataset based on entire scenes into training, validation and testing set. We use specific scenes exclusively either for the training set or the validation/testing set. This ensures that our network does not see similar images during the training and validation/testing process. Only then we are able to calculate meaningful errors on the validation and testing datasets.

We use 690 (82 %) images of 20 scenes[1] for training, 102 (12 %) images for validation and 48 (6 %) images for testing. The validation and testing images are sampled from the remaining three scenes[2].

Furthermore we perform data augmentation on the training dataset to enhance the variation between training images further. This includes randomly mirroring the images horizontally, cropping images at random spatial positions down to the chosen network input size, scaling images by a random factor between $[0.9, 1.1]$ and rotating the images spatially by a random angle in the range of $[-15°, +15°]$. All augmentation processes are performed during training on all images and its corresponding albedo and shading ground truths.

Intrinsic Images in the Wild Dataset: Since the IIW dataset contains only sparse ground truth albedo and shading labels we train the neural networks on rather spatially large images. Only then we can be sure to get a significant comparison between predictions and labels. Therefore we select images for the datasets that have spatial dimensions of at least $(320, 320)$. All smaller 346 images are deleted. This leads to a splitting of the dataset which results in 3908 (80 %) training samples, 488 (10 %) validation samples and 488 (10 %) testing samples.

Appart from cropping images at random spatial positions down to the chosen network input size we do not apply any other data augmentation technique. This is also due to the distinctive form of the IIW dataset with its sparse labels. The sparse labels are adapted in accordance with the image crops.

[1] The Sintel Dataset scenes used in the training set are: *alley_1, alley_2, ambush_2, ambush_4, ambush_5, ambush_6, ambush_7, bamboo_1, bandage_1, bandage_2, cave_2, cave_4, market_2, market_6, shaman_2, shaman_3, sleeping_1, sleeping_2, temple_2, temple_3.*

[2] The Sintel Dataset scenes used in the validation/testing set are: *bamboo_2, market_5, mountain_1.*

5.2. Results

5.2.1. Training on MPI Sintel Dataset

As mentioned before we start by training our networks on the Sintel datasets. We train five different models:

- *ResNet50-2scale*: A network consisting of a ResNet-50 encoder (see figure A.2) and the *2scale* decoder (see figure 4.4).
- *ResNet50-deconv*: A network consisting of a ResNet-50 encoder (see figure A.2) and the *deconv* decoder (see figure 4.5).
- *ResNet50-upproject*: A network consisting of a ResNet-50 encoder (see figure A.2) and the *upproject* decoder (see figure 4.6).
- *VGG16-2scale*: A network consisting of a VGG-16 encoder (see figure A.1) and the *2scale* decoder (see figure 4.4).
- *VGG16-deconv*: A network consisting of a VGG-16 encoder (see figure A.1) and the *deconv* decoder (see figure 4.5).

We trained these models with different loss functions: the $\mathcal{L}_2$ loss, the invariant $\mathcal{L}_2$ loss ($\mathcal{L}_{2,\lambda=1}$), the average $\mathcal{L}_2$ loss ($\mathcal{L}_{2,\lambda=0.5}$), the $\mathcal{L}_1$ loss and the BerHu $\mathcal{B}$ loss. During training we obtained best results using the $\mathcal{L}_1$ loss. BerHu loss delivered similar results. Using the $\mathcal{L}_2$ losses a noticible learning progress was much more difficult to achieve. That is why we used the $\mathcal{L}_1$ loss for training. A typical learning progress using this loss is shown in figure 5.1.

Fig. 5.1.: Examples of typical training processes. $\mathcal{L}_1$ loss in dependence of training step for the *ResNet50-2scale* (left side) and *ResNet50-upproject* (right side) model. The *ResNet50-2scale* model has a much smoother training progress. The *ResNet50-upproject* on the other hand exhibits more outliers in the validation loss. The dashed black line indicates the best model according to validation loss.

	$\mathcal{L}_2$				$\mathcal{L}_{2,\lambda=1}$				$\mathcal{L}_{2,\lambda=0.5}$			
	albedo	shading	avg	sum	albedo	shading	avg	sum	albedo	shading	avg	sum
ResNet50-deconv	0.0463	0.0209	0.0336	0.0671	0.0173	0.0090	0.0132	0.0263	0.0318	0.0149	0.0234	0.0467
ResNet50-upproject	0.0389	0.0225	0.0307	0.0614	0.0141	0.0098	0.0120	0.0239	0.0265	0.0161	0.0213	0.0426
ResNet50-2scale	**0.0364**	**0.0181**	**0.0273**	**0.0545**	0.0128	0.0080	0.0104	0.0208	0.0246	0.0130	0.0188	0.0377
VGG16-2scale	0.0479	0.0244	0.0362	0.0723	0.0185	0.0107	0.0146	0.0291	0.0332	0.0175	0.0254	0.0507
VGG16-deconv	0.0506	0.0228	0.0367	0.0734	0.0210	0.0087	0.0148	0.0297	0.0358	0.0157	0.0258	0.0515

	$\mathcal{L}_1$				$\mathcal{B}$			
	albedo	shading	avg	sum	albedo	shading	avg	sum
ResNet50-deconv	0.1702	0.1089	0.1396	0.2791	0.1904	0.1160	0.1532	0.3064
ResNet50-upproject	0.1573	0.1128	0.1350	0.2701	0.1573	0.1128	0.1351	0.2701
ResNet50-2scale	**0.1539**	**0.1001**	**0.1270**	**0.2540**	0.1702	0.1039	0.1370	0.2740
VGG16-2scale	0.1717	0.1172	0.1444	0.2889	0.1904	0.1206	0.1555	0.3110
VGG16-deconv	0.1720	0.1188	0.1454	0.2908	0.2024	0.1233	0.1629	0.3258

Tab. 5.1.: Comparison of all trained models and estimated losses ($\mathcal{L}_2$, invariant $\mathcal{L}_{2,\lambda=1}$, average $\mathcal{L}_{2,\lambda=0.5}$, $\mathcal{L}_1$ and BerHu $\mathcal{B}$ loss) on the Sintel test dataset. For training the $\mathcal{L}_1$ loss is used. The best model (see bold values) is the *ResNet50-2scale* model which consists of a ResNet-50 encoder and a two-scale decoder. For a visual comparison of these models see figure 5.2. Some further intrinsic image decompositions of our best model *ResNet50-2scale* are shown in figure 5.3.

In table 5.1 we compare all losses of all trained models. All losses are calculated on the test dataset and represent the minimal test loss. It can be seen that models using the ResNet-50 encoder have a more accurate decomposition (lower loss) than models using VGG-16. In case of the decoders, the *2scale* architecture is superior. The best performing model according to the test loss is the *ResNet50-2scale* model. In the following we will analyse this model in more detail and refer to it as best model.

Figure 5.2 shows a frame of the MPI Sintel dataset (test set) with input and ground truth images and visualizes the albedo-shading decomposition quality of all models. This illustrated model comparison highlights the best performance of the *ResNet50-2scale* model, too. One can see clearly how reflectances and shadows are sepearated.

In figure 5.3 we show the intrinsic image decomposition results of our best model *ResNet50-2scale*. On each scene the decompositions appear to be plausible and comprehensible.

Fig. 5.2.: Visualization of the performance of the different models on a Sintel scene from the testing dataset. The best model *ResNet50-2scale* delivers the most accurate decomposition. In generall, the *2scale* decoder architecture performs best.

Of course we also want to compare our results to results from the literature. For this we consider two trivial decomposition baselines where either shading or albedo is assumed

Fig. 5.3.: Intrinsic image decomposition: Further examples of the performance of the best model *ResNet50-2scale*. All frames are sampled from the Sintel validation or testing dataset.

uniform grey (calcualted by [33]) and the historic Retinex algorithm (version of [16]). Besides that we employ models of Lee et al. [29], Barron and Malik [5] and Chen and Koltun [11]. All of these publications use additional input for training (RGB-D images). Finally, Narihira et al. [33] use a similar approach to our models. The results are shown in table 5.2 and figure 5.4. The comparison with the literature shows that our intrinsic image decomposition results are among the more superior ones. We get comparable results to Narihira et al. [33] and outperform all other models mentioned above.

	albedo	shading	avg	sum
Baseline: Shading Constant	0.0531	0.0488	0.0510	0.1019
Baseline: Albedo Constant	0.0369	0.0378	0.0374	0.0747
Retinex [16]	0.0606	0.0727	0.0667	0.1333
Lee et al. [29]	0.0463	0.0507	0.0485	0.0970
Barron and Malik [5]	0.0420	0.0436	0.0428	0.0856
Chen and Koltun [11]	0.0307	0.0277	0.0292	0.0584
Narihira et al. [33]	0.0201	0.0224	0.0212	0.0425
ours	0.0369	0.0187	0.0278	0.0556

Tab. 5.2.: Comparison of $\mathcal{L}_2$ losses (standard MSE metric) of our best performing model *ResNet50-2scale* (ours) with literature. The values are comparable only to a certain degree because they are not calculated on the same test datases. All models but Narihira et al. [33] and ours are trained on a train/test-split of the Sintel dataseta where frames are randomly assigned to the train or test set. This limits the significance of the test loss because scenes are too similar to scenes in the train set. Considering this fact, our model significantly outperforms these models. Compared to Narihira et al. [33] (who also trained their models on a train/test-split by scenes) our model is superior in predicting the shading layer but inferior in the albedo layer prediction. It can be said that both models have a comparable performance. Figure 5.4 shows the intrinisc decomposition results on an example scene.

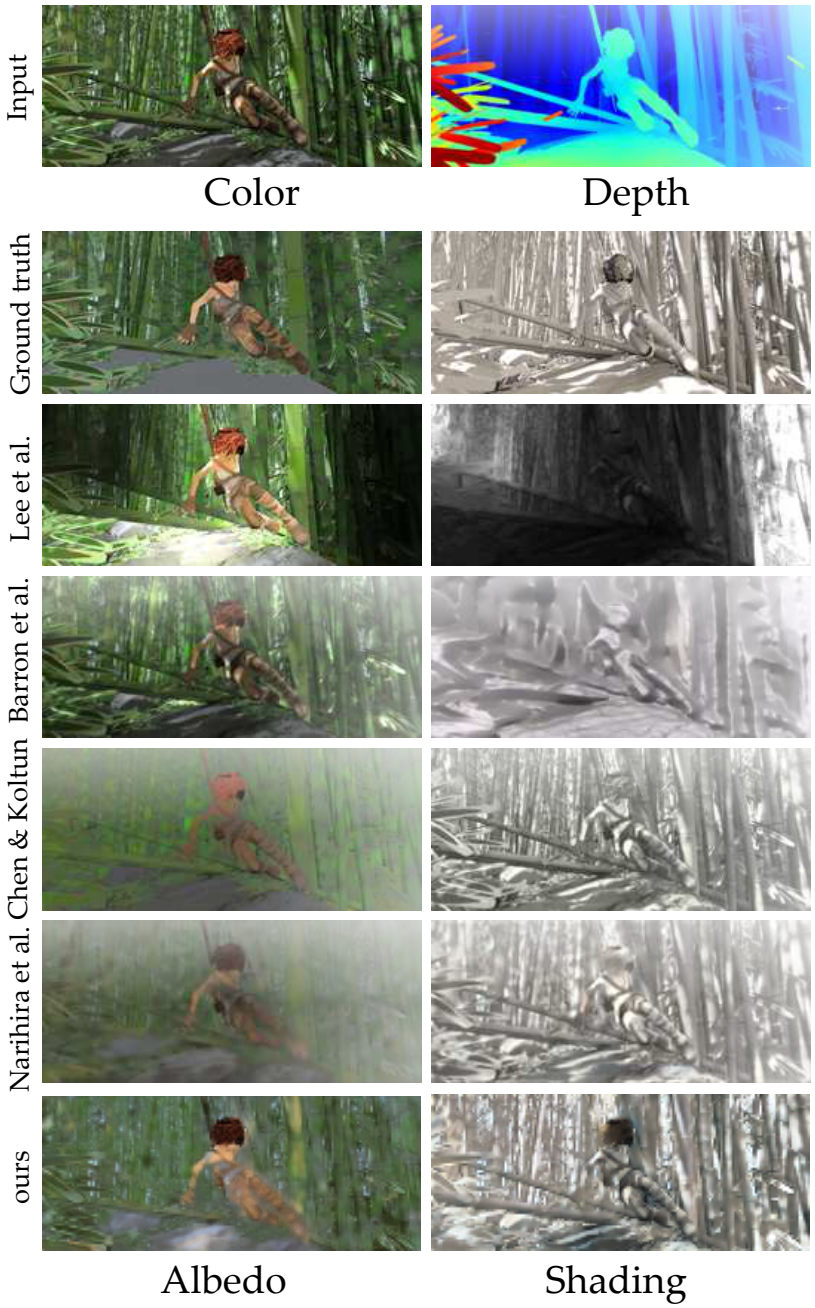

Fig. 5.4.: Comparison of our best performing model *ResNet50-2scale* compared to literature on the MPI Sintel dataset. Lee et al. [29], Barron and Malik [5], Chen and Koltun [11] used for training additional input (RGB-D images). The depth image is shown in the top row. Narihira et al. [33] and our model are only trained on RGB images. It can clearly be seen that we produced comparable results to Narihira et al. [33] and superior decompositions to the other ones (see also table 5.2 for a quantitative analysis).

Finally, we take a look how our best model *ResNet50-2scale* trained on the synthedic MPI Sintel dataset performs on real world scenes from the IIW dataset (see figure 5.5).

Here too, a comprehensible result can be seen.

Fig. 5.5.: Intrinsic image decomposition of real-world scenes from the IIW dataset. These results are generated by our best model *ResNet50-2scale* trained exclusively on the synthetic MPI Sintel dataset. We obtain albedo and shading images that are quite reasonable, but not perfect. We calculate the WHDR metric on the albedo prediction using equation (4.7) for each decomposition: 0.1879 (top row), 0.6119 (middle row), 0.1214 (bottom row). According to the WHDR metric (human reflectance judgements) we get a pretty accurate decomposition of scenes in the top and especially bottem row (low WDHR). However, the decomposition results of the image in the middle are satisfying as reported by the WHDR metric.

5.2.2. Training on IIW Dataset

Now we want to optimize the results from the synthetic MPI Sintel dataset for real-world scenes. For this we use the parameters of our best Sintel model *ResNet50-2scale* and feed them into the network structure illustrated in figure 4.7. We train this network with the ResNet-50 encoder and the *2scale* decoder on the real-world IIW dataset.

We train models with different loss functions. We use the IIW dataset specific loss functions discribed by equations (4.19) and (4.20). In more detail, we us the following loss functions: $\mathcal{L}_{1,\lambda=0.1}$, $\mathcal{L}_{1,\lambda=1.0}$, $\mathcal{L}_{1,\lambda=0.05,\omega}$, $\mathcal{L}_{1,\lambda=0.1,\omega}$ and $\mathcal{L}_{2,\lambda=0.01,\omega}$. Table 5.3 compares all used losses. It turns out that the $\mathcal{L}_2$ loss performs best.

	MSE	WHDR
$\mathcal{L}_{1,\lambda=0.05,\omega}$	0.0020	0.3200
$\mathcal{L}_{1,\lambda=0.1,\omega}$	0.0030	0.3125
$\mathcal{L}_{1,\lambda=0.1}$	0.0041	0.3249
$\mathcal{L}_{1,\lambda=1.0}$	0.0034	0.3251
$\mathcal{L}_{2,\lambda=0.01,\omega}$	0.0019	0.3064

Tab. 5.3.: Comparison of the different models on the IIW test set. We use two metrics that are valid for all models und thus make them comparable: The general mean squared error using the definition of image decomposition (see equation (1.1)) given by MSE $= ||I - A \cdot S||_2^2$ and the WHDR metric (see equation (4.7)). The more meaningful metric in this case is certainly the IIW dataset specific WHDR metric. The best performing model is the *ResNet50-2scale* model using the $\mathcal{L}_{2,\lambda=0.01,\omega}$ loss.

For further analysis we visualize the best performing model with the $\mathcal{L}_{2,\lambda=0.01,\omega}$ loss in figure 5.6. For comparison, we show the same decomposition as in figure 5.5.

Considerting figure 5.6 it is obvious that our intrinsic image decomposition models trained on the IIW dataset do not produce accurate results. If we compare our test set WHDR metric of 0.3064 (see table 5.3) with some literature WHDR values like 0.210 from Bell et al. [6] or 0.1913 from Zhou et al. [46] this impression is intensified. It seems like the loss functions (see equations (4.19) and (4.20)) are the problem of this. During the training process the network tends to generate a decomposition that wants to replicate the input image in the shading output layer. This means it pushes the output pixels of the albedo prediction towards 1. The first summand in equations (4.19) and (4.20) $|I - A \cdot S|$ with $A \to \mathbf{1}$ and $S \to I$ is thereby minimized which leads to a minimization of the complete loss function. Actually, we wanted to prevent this by using the pre-trained Sintel model as original source of parameters. We tried to train the IIW network with lower initial learning rate to prevent the model to go in this direction, but this did not result in better decompositions. The problem seems to be the assymetric loss function which applies heavier constraints on the albedo output. A solution to this problem might be to introduce some constraints on the shading output, too. Another possibility would be to define a loss function that does not depend on the shading S. This could be achieved by defining a loss function that gets rid of the first summand in equations (4.19) and (4.20) and only considers the classification task on the albedo layer. We tried this to a certain extend and noticed that this leads to a too weak loss because then only a few pixel of the albedo layer (because of the sparse available labels in the IIW dataset) contribute to the complete loss function.

Fig. 5.6.: Intrinsic image decomposition of real-world scenes from the IIW dataset. These results are generated by our best model using the network architecture *ResNet50-2scale* and $\mathcal{L}_{2,\lambda=0.01,\omega}$ loss.

6. Conclusion

We were able to produce meaningful results of albedo and shading images on the Sintel dataset. Although it is a synthetic dataset its results can be transfered to real world applications.

Regarding the IIW dataset, we encountered some difficulties with the construction of our dataset specific loss function. However, learning from this dataset in a purely data-driven approach should be possible by tuning the loss function.

In general, there are some limitations considering the task of intrinsic image decomposition:

First of all we consider the definition of the decomposition task $I = A \cdot S$. This is just an approximation and describes the

Second, the availablility of ground truth data (training data) is the most difficult challenge to overcome. The MPI Sintel dataset is synthetic and can describe real-world scenes only to a certain degree. The IIW dataset contains exclusively sparse human judgements rather than physical properties. They still may be insufficient for this task overall. In conclusion, it cannot be said if the Sintel albedo/shading decomposition describes exactly the same physical properties as the sparse human reflectance comparisions of the IIW dataset. Nevertheless, a connection between both should be possible between both to a certain extend. In fact, with our approach we showed that it is possible to use both datasets together in a purely data driven learning task.

Nevertheless, further training is needed, especially on the IIW dataset. With some tuning of the loss function one should be able to achieve better results on the IIW dataset.

Appendix

Appendix

A. Encoder Architectures

In our networks we used state of the art CNN models. For the encoders we adopted well-approved architectures, i.e. VGG-16 and ResNet-50. Both networks have been introduced for image classification tasks. We are generally only interested in the CNN layers of these networks and get rid of fully connected layers at the end of the networks. In practice we use the pre-trained TensorFlow-Slim [1] implementations of both models. During training we fine-tune these parameters to adjust them to our problem. In the following we visualize both used encoders:

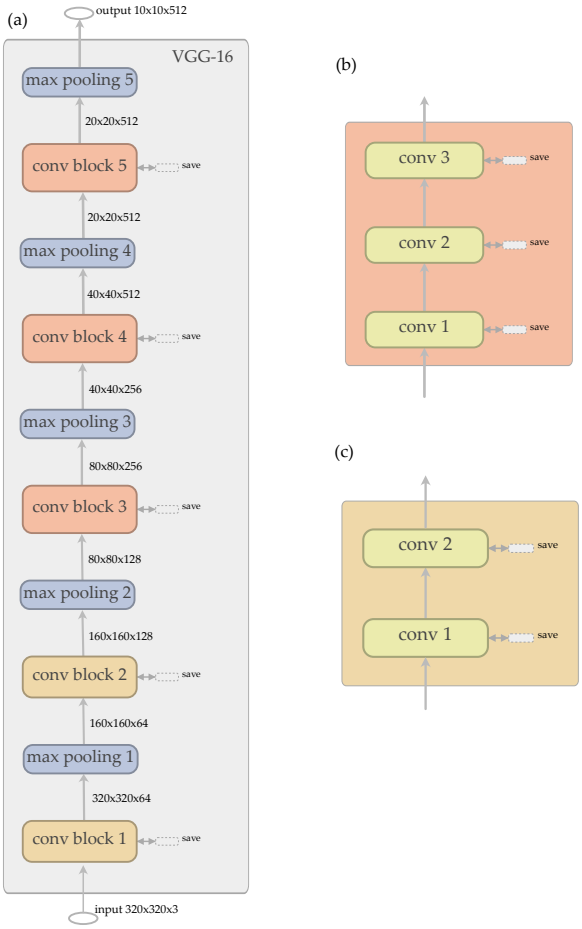

Fig. A.1.: (a) Architecture of the VGG-16 model excluding the final fully connected layers. This network is inspired by [41]. Convolutional layer blocks are followed by max-pooling layers. (b) and (c) show the two different convolution blocks (color-coded). Our input images have always size $320 \times 320 \times 3$. The encoder outputs a feature map with spatial dimensions 10×10 which has to be up-scaled afterwards by the decoder (see section 4.2). This VGG-16 network has $14,714,688$ parameters. It does not use any regularization method like batch normalization.

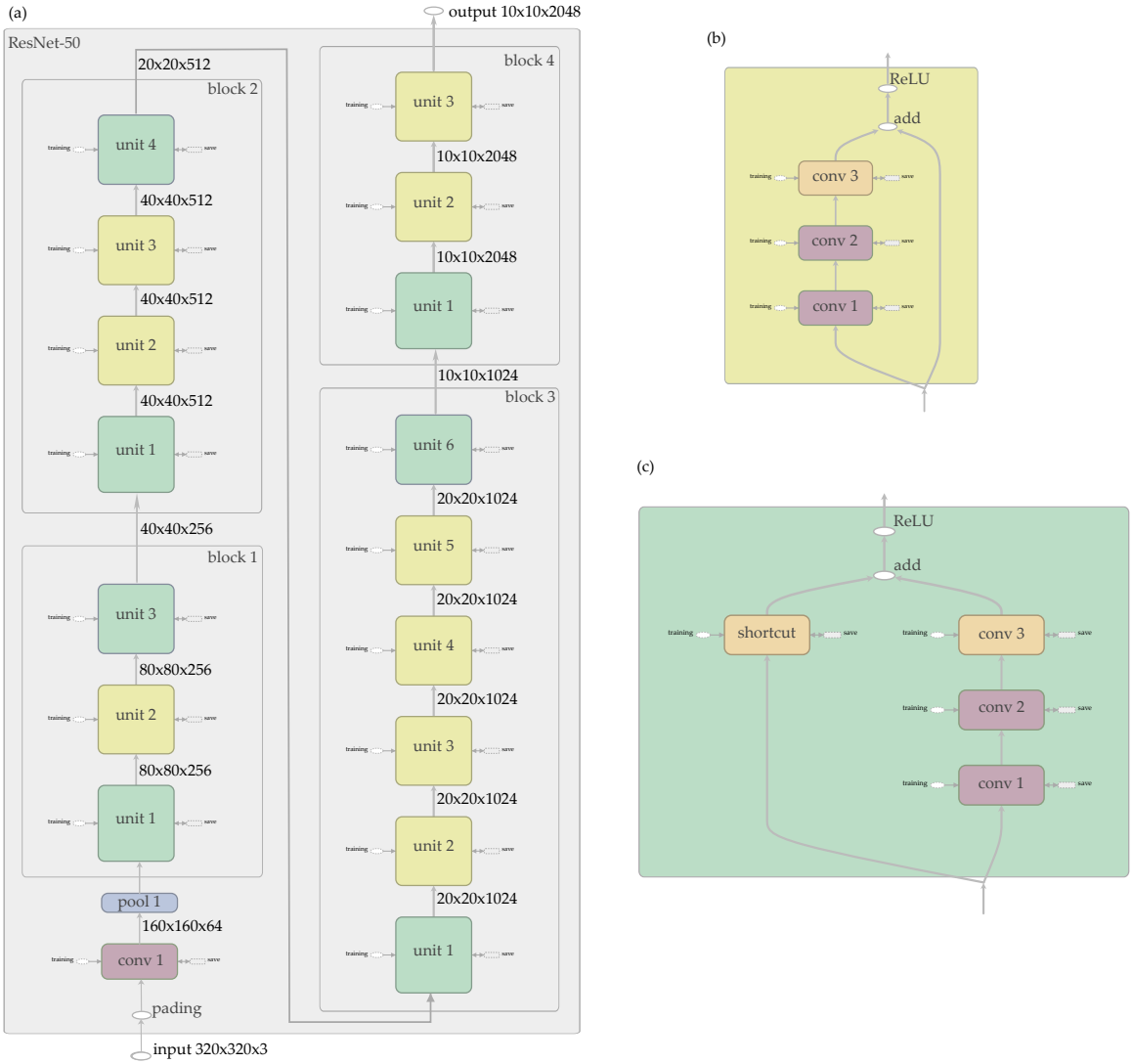

Fig. A.2.: (a) Architecture of the ResNet-50 model excluding the final fully connected layers. This network is inspired by [19]. The network is structured in four blocks consisting of several units of convolutional layers. The color-coded units are visualized in more detail in (b) and (c). Each convolutional layer in each unit is accompanied by batch normalization. Our input images have always size $320 \times 320 \times 3$. The encoder outputs a feature map with spatial dimensions 10×10 which has to be up-scaled afterwards by the decoder (see section 4.2). This ResNet-50 network has $23,508,032$ parameters.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Jonathan T Barron and Jitendra Malik. Color constancy, intrinsic images, and shape estimation. In *European Conference on Computer Vision*, pages 57–70. Springer, 2012.

[3] Jonathan T Barron and Jitendra Malik. Shape, albedo, and illumination from a single image of an unknown object. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 334–341. IEEE, 2012.

[4] Jonathan T Barron and Jitendra Malik. Intrinsic scene properties from a single rgb-d image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 17–24, 2013.

[5] Jonathan T Barron and Jitendra Malik. Shape, illumination, and reflectance from shading. *IEEE transactions on pattern analysis and machine intelligence*, 37(8):1670–1687, 2015.

[6] Sean Bell, Kavita Bala, and Noah Snavely. Intrinsic images in the wild. *ACM Transactions on Graphics (SIGGRAPH 2014)*, 33(4), 2014.

[7] Sean Bell, Paul Upchurch, Noah Snavely, and Kavita Bala. Opensurfaces: A richly annotated catalog of surface appearance. *ACM Transactions on Graphics (TOG)*, 32(4):111, 2013.

[8] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[9] Adrien Bousseau, Sylvain Paris, and Frédo Durand. User-assisted intrinsic images. In *ACM Transactions on Graphics (TOG)*, volume 28, page 130. ACM, 2009.

[10] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. A naturalistic open source movie for optical flow evaluation. In A. Fitzgibbon et al. (Eds.), editor, *European Conf. on Computer Vision (ECCV)*, Part IV, LNCS 7577, pages 611–625. Springer-Verlag, October 2012.

[11] Qifeng Chen and Vladlen Koltun. A simple model for intrinsic image decomposition with depth cues. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2013.

[12] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. In *Advances in neural information processing systems*, pages 2366–2374, 2014.

[13] Elena Garces, Adolfo Munoz, Jorge Lopez-Moreno, and Diego Gutierrez. Intrinsic images by clustering. In *Computer graphics forum*, volume 31, pages 1415–1424. Wiley Online Library, 2012.

[14] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

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

[16] R. Grosse, M. K. Johnson, E. H. Adelson, and W. T. Freeman. Ground truth dataset and baseline evaluations for intrinsic image algorithms. In *2009 IEEE 12th International Conference on Computer Vision*, pages 2335–2342, Sept 2009.

[17] Tom Haber, Christian Fuchs, Philippe Bekaer, Hans-Peter Seidel, Michael Goesele, and Hendrik PA Lensch. Relighting objects from image collections. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 627–634. IEEE, 2009.

[18] Daniel Hauagge, Scott Wehrwein, Kavita Bala, and Noah Snavely. Photometric ambient occlusion. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 2515–2522. IEEE, 2013.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[21] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[22] Seungryong Kim, Kihong Park, Kwanghoon Sohn, and Stephen Lin. Unified depth prediction and intrinsic image decomposition from a single image via joint convolutional neural fields. *CoRR*, abs/1603.06359, 2016.

[23] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

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

[25] Pierre-Yves Laffont, Adrien Bousseau, and George Drettakis. Rich intrinsic image decomposition of outdoor scenes from multiple views. *IEEE transactions on visualization and computer graphics*, 19(2):210–224, 2013.

[26] Pierre-Yves Laffont, Adrien Bousseau, Sylvain Paris, Frédo Durand, and George Drettakis. Coherent intrinsic images from photo collections. *ACM Transactions on Graphics*, 31(6), 2012.

[27] Iro Laina, Christian Rupprecht, Vasileios Belagiannis, Federico Tombari, and Nassir Navab. Deeper depth prediction with fully convolutional residual networks. In *3D Vision (3DV), 2016 Fourth International Conference on*, pages 239–248. IEEE, 2016.

[28] Edwin H Land and John J McCann. Lightness and retinex theory. *JOSA*, 61(1):1–11, 1971.

[29] Kyong Joon Lee, Qi Zhao, Xin Tong, Minmin Gong, Shahram Izadi, Sang Uk Lee, Ping Tan, and Stephen Lin. Estimation of intrinsic image sequences from image+ depth video. In *European Conference on Computer Vision*, pages 327–340. Springer, 2012.

[30] Xiaopei Liu, Lei Jiang, Tien-Tsin Wong, and Chi-Wing Fu. Statistical invariance for texture synthesis. *IEEE transactions on visualization and computer graphics*, 18(11):1836–1848, 2012.

[31] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[32] Takuya Narihira, Michael Maire, and X Yu Stella. Learning lightness from human judgement on relative reflectance. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2965–2973. IEEE, 2015.

[33] Takuya Narihira, Michael Maire, and Stella X. Yu. Direct intrinsics: Learning albedo-shading decomposition by convolutional regression. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[34] Byong Mok Oh, Max Chen, Julie Dorsey, and Frédo Durand. Image-based modeling and photo editing. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 433–442. ACM, 2001.

[35] Ido Omer and Michael Werman. Color lines: Image specific color representation. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–946. IEEE, 2004.

[36] Art B Owen. A robust hybrid of lasso and ridge regression. *Contemporary Mathematics*, 443:59–72, 2007.

[37] Carsten Rother, Martin Kiefel, Lumin Zhang, Bernhard Schölkopf, and Peter V Gehler. Recovering intrinsic images with a global sparsity prior on reflectance. In *Advances in neural information processing systems*, pages 765–773, 2011.

[38] E. Shelhamer, J. T. Barron, and T. Darrell. Scene intrinsics and depth from a single image. In *2015 IEEE International Conference on Computer Vision Workshop (ICCVW)*, pages 235–242, Dec 2015.

[39] L. Shen and C. Yeo. Intrinsic images decomposition using a local and global sparse representation of reflectance. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 697–704, June 2011.

[40] Li Shen, Ping Tan, and S. Lin. Intrinsic image decomposition with non-local texture cues. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–7, June 2008.

[41] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[42] Marshall F Tappen, Edward H Adelson, and William T Freeman. Estimating intrinsic component images using non-linear regression. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1992–1999. IEEE, 2006.

[43] Marshall F Tappen, William T Freeman, and Edward H Adelson. Recovering intrinsic images from a single image. *IEEE transactions on pattern analysis and machine intelligence*, 27(9):1459–1472, 2005.

[44] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[45] Qi Zhao, Ping Tan, Qiang Dai, Li Shen, Enhua Wu, and Stephen Lin. A closed-form solution to retinex with nonlocal texture constraints. *IEEE transactions on pattern analysis and machine intelligence*, 34(7):1437–1444, 2012.

[46] Tinghui Zhou, Philipp Krahenbuhl, and Alexei A Efros. Learning data-driven reflectance priors for intrinsic image decomposition. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3469–3477, 2015.

[47] Laurent Zwald and Sophie Lambert-Lacroix. The berhu penalty and the grouped effect. *arXiv preprint arXiv:1207.6868*, 2012.