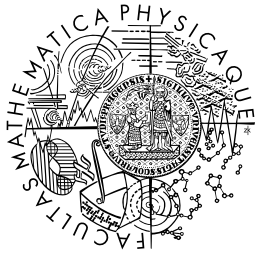


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

**MASTER THESIS**

Bc. Martin Mirbauer

# Evaluation of Dynamic Range Reconstruction Approaches and a Mobile Application for HDR Photo Capture

Department of Software and Computer Science Education

Supervisor of the master thesis: doc. Ing. Jaroslav Křivánek, Ph.D.
Study programme: Computer Science (N1801)
Study branch: IPGVPH (1801T053)

Prague 2018

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date .....  signature of the author

I would like to thank my thesis advisor for help with important decisions and my family for their support and patience.

Thank you!

Title: Evaluation of Dynamic Range Reconstruction Approaches and a Mobile Application for HDR Photo Capture

Author: Bc. Martin Mirbauer

Department: Department of Software and Computer Science Education

Supervisor: doc. Ing. Jaroslav Křivánek, Ph.D., Department of Software and Computer Science Education

Abstract: Digital photography became widespread with the global use of smartphones. However, most of the captured images do not fully use the camera capabilities by storing the captured photos in a format with limited dynamic range. The subject of dynamic range expansion and reconstruction has been researched since early 2000s and recently gave rise to several new reconstruction methods using convolutional neural networks (CNNs), whose performance has not yet been comprehensively compared.

By implementing and using our dynamic range reconstruction evaluation framework we compare the reconstruction quality of individual CNN-based approaches. We also implement a mobile HDR camera application and evaluate the feasibility of running "the best-performing reconstruction method" directly on a mobile device.

Keywords: high dynamic range, inverse tone mapping, deep learning, mobile application development

# Contents

# Introduction

## Background and Motivation

Digital cameras have become ubiquitous during the past few years, yet the majority of taken photographs does not fully use the camera capabilities. While the resolution and color accuracy reproduction have been improving, the captured image brightness still lacks the dynamic range of the original scene. This distortion is caused by hardware limitations such as sensor sensitivity and saturation and more importantly by the camera software processing the image data from the sensor and transforming the measured pixel values into low dynamic range (8 bits per channel) and usually storing the result in a lossy format such as JPEG File Interchange Format. The process of reduction of dynamic range is called tone mapping; the complete transformation, which maps incoming luminance range to 8 bits to be stored, is called the camera response function. It clips low-luminance values (not measurable by the sensor) to zero and high-luminance values (i.e. when sensor saturation occurs) to 255. The mapping of the intermediary luminance values can be any monotonically increasing function of luminance - determined by the camera manufacturer.

High dynamic range (or HDR) images contain more per-pixel information – pixel values are stored with higher precision and are from higher range of values, usually linearly representing the incoming luminance. This additional information can be used in image post-production, allowing exposure fine-tuning or emphasizing details in shadowy areas, and even for viewing, as high dynamic range displays become widely available. Apart from normal photography producing "framed rectangle" images, the need for HDR images extends to spherical panoramas, which capture the whole 360 deg of azimuth and 180 deg of elevation, used as environment maps for computer games or image-based lighting for offline rendering.

High dynamic range images can be captured with regular low dynamic range (or LDR) camera by taking multiple photos with different exposures, e.g. by varying the exposure time, where each exposure captures part of the scene's dynamic range, which contains details in scene's highlights or shadows, and merging the resulting images. While many cameras support taking an "HDR" image using multiple exposures, especially in smartphones the software often stores the merged image in an LDR format, reducing the captured dynamic range. Although many alternative camera applications offering true HDR image capture are available in the Apple App Store, none of them seems to be open-sourced, which makes mobile HDR photography inaccessible to researchers without previous experience with iOS application development.

Recently several advanced approaches for HDR image reconstruction from a single exposure LDR image have been published. These approaches use deep convolutional neural network to estimate the image areas, which were distorted when capturing the LDR photo. The performance of these approaches needs to be measured and suitability for practical applications needs to be evaluated.

## Goals

In this thesis, we focus on evaluation of methods for expanding the dynamic range of LDR images using deep convolutional neural networks with emphasis on spherical panorama images. We implement a pipeline to automatically reduce the dynamic range of an original (ground truth) HDR image, run several HDR reconstruction algorithms, measure error metrics comparing the reconstruction to the ground truth image and to present the results in both user-interactive and machine-readable formats.

We also present an implementation of a mobile application for capturing the highest possible dynamic range of a scene and evaluate the feasibility of using a dynamic range enhancement algorithm to recover details in the image areas, which cannot be captured by the camera, directly on the device.

This application's source code will be released along with the HDR reconstruction evaluation pipeline code under a permissive license, allowing easy experimentation with HDR image capture and reconstruction algorithms evaluation.

## Results (preliminary)

See Figure 1 for an example output of the HDR reconstruction evaluation pipeline and the implemented application's GUI on Figure 2.

## Thesis outline

In the first chapter (Related work) we explore existing approaches to HDR image capture and reconstruction.

The second chapter (High dynamic range reconstruction evaluation framework) describes the design and implementation of the HDR reconstruction evaluation pipeline.

The third chapter (HDR camera application) presents the design of an HDR capture pipeline with optional artificial dynamic range enhancement and describes the obstacles that needed to be overcome for a working implementation on a mobile device.

The fourth chapter (Results) compares the performance of three HDR reconstruction approaches based on deep learning using the HDR reconstruction evaluation pipeline and compares photos taken with the implemented HDR camera application with other camera applications' results.

The last chapter (Conclusion and future work) summarizes the achieved results and presents possible future expansion of this work.

| Ground truth image | Reconstruction image | Label | Saturation | L2 error | L2 error (saturated pixels) | L2 error (un-saturated pixels) | SSIM | Mapping |
|---|---|---|---|---|---|---|---|---|
| 04-16_Sky.exr | 04-16_Sky.exr.jpg_hdrcnn.exr | 04-16_Sky.exr reconstruction hdrcnn | 0.0013 | 0.13548365 | 3.73722496 | 0.00104895 | 0.999184028687 | Luminance Color |
| 04-16_Sky.exr | 04-16_Sky.exr.jpg | 04-16_Sky.exr LDR | 0.0013 | 0.14575489 | 4.02059151 | 0.00091467 | 0.998871532189 | Luminance Color |
| 04-16_Sky.exr | 04-16_Sky.exr.jpg_drtmo.exr | 04-16_Sky.exr reconstruction drtmo | 0.0013 | 0.15259209 | 4.20087841 | 0.00964005 | 0.997723076046 | Luminance Color |
| 04-16_Sky.exr | 04-16_Sky.exr.jpg_expandnet.exr | 04-16_Sky.exr reconstruction expandnet | 0.0013 | 0.15603888 | 4.26341609 | 0.02148408 | 0.993623129662 | Luminance Color |
| 04-17_Cloudy.exr | 04-17_Cloudy.exr.jpg | 04-17_Cloudy.exr LDR | 0.2026 | 0.00509132 | 0.01072722 | 0.00180702 | 0.986771312445 | Luminance Color |
| 04-17_Cloudy.exr | 04-17_Cloudy.exr.jpg_hdrcnn.exr | 04-17_Cloudy.exr reconstruction hdrcnn | 0.2026 | 0.01947942 | 0.04105619 | 0.00689289 | 0.988505361364 | Luminance Color |
| 04-17_Cloudy.exr | 04-17_Cloudy.exr.jpg_drtmo.exr | 04-17_Cloudy.exr reconstruction drtmo | 0.2026 | 0.02029011 | 0.01619066 | 0.02120599 | 0.894827024909 | Luminance Color |
| 04-17_Cloudy.exr | 04-17_Cloudy.exr.jpg_expandnet.exr | 04-17_Cloudy.exr reconstruction expandnet | 0.2026 | 0.04758943 | 0.07920407 | 0.03529963 | 0.868872542217 | Luminance Color |
| 04-18_Cloudy.exr | 04-18_Cloudy.exr.jpg_hdrcnn.exr | 04-18_Cloudy.exr reconstruction hdrcnn | 0.0013 | 0.04963048 | 1.35883556 | 0.00120543 | 0.999521416224 | Luminance Color |

Figure 1: Preview of the HDR reconstruction evaluation pipeline results: RMSE and SSIM error metrics for each reconstruction method.

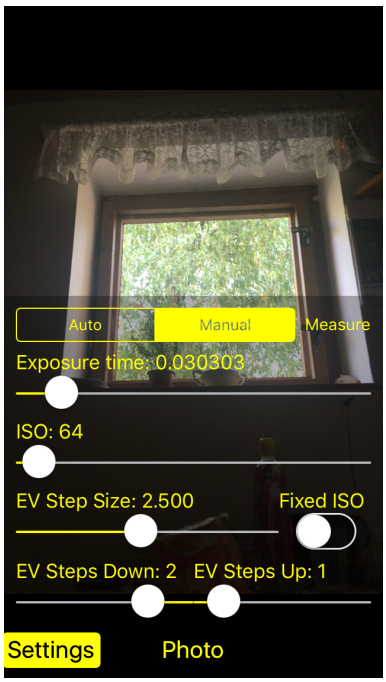



Figure 2: Camera GUI: capture settings.

# 1. Related work

## 1.1 HDR image reconstruction

The HDR to LDR transformation (tone mapping) loses image data by reducing the range and precision of pixel values, therefore there is no "correct" inverse transformation. But there are approaches to approximate or simulate the original luminance values, essentially attempting to find an inverse tone mapping function by making assumptions about the tone mapping function.

### 1.1.1 Global expansion methods

Global expansion methods do not attempt to reconstruct any missing data, they only expand the luminance range by applying a global (spatially constant) per-pixel function.

These approaches are based on relatively simple ideas, which are easy to implement and computationally non-demanding to execute. This makes them suitable for hardware with limited computation power and when low latency is desirable, such as in HDR displays with LDR inputs.

The simple global dynamic range expansion introduces/amplifies banding or quantization artifacts, which some of the approaches try to reduce.

One of the first dynamic range expansion approaches is outlined by [Landis, 2002] expanding the dynamic range of an environment map using exponentiation – raising luminance values in selected range to a constant power. While this method may be used to boost the highlights, it disregards any introduced artifacts.

Another method described by [Akyüz et al., 2007] consists of linearization of the pixel values (gamma correction) followed by linear scaling to match the required output range. Despite the simplicity of this approach, according to the authors' psychophysical experiment, this reconstruction method was preferred by most of the test subjects, compared to non-linear dynamic range expansion methods.

[Meylan et al., 2006] focuses on re-rendering of LDR images for displaying on HDR displays. The global inverse tone mapping function is piecewise linear with steeper slope for high input values, leading to greater expansion of specular highlights. To reduce artifacts caused by the non-linearity in the tone mapping function, the image is blurred around the highlights as a post-processing step.

A recent research by [Masia et al., 2017] shows that "gamma correction, then linear expansion" can produce good results even for underexposed or overexposed images when correct $\gamma$ is used. The authors propose a method for finding the $\gamma$ based on statistical properties of the input image.

### 1.1.2 Local ("surrounding-aware") expansion methods

Local expansion methods assume non-constant (spatially varying) tone mapping function, whose parameters change based on the pixel values of the input image to preserve contrast and details in both highlights and shadows. An example of such a tone mapping function is presented by [Reinhard et al., 2002]. The

corresponding inverse tone mapping function therefore needs to treat the image areas with different surrounding luminance differently to reverse this effect.

Local dynamic range expansion approaches benefit from the extra information for generating parameters of the per-pixel inverse tone mapping function. While these approaches are generally more computationally demanding, some describe how the algorithm can be implemented in an efficient way, allowing it to run in real-time.

[Banterle et al., 2006] suggest boosting highlights by computing the expanded luminance value of a pixel as a density estimation of high-value pixels around current pixel. This can be achieved more efficiently by creating an *expand map* – a monochrome image containing ones in highlights (zeros elsewhere), which is then blurred in order to provide smooth transitions. The expand map values are then used as weights for linear interpolation between the original LDR image and an HDR image constructed using a global inverse tone mapping function.

Another approach by [Rempel et al., 2007] aims to prevent "bleeding" of a highlight into less saturated areas, caused by blurring the *expand map*, while managing to achieve high performance and temporal stability of colors to be suitable for expanding the dynamic range of both LDR image and video input on hardware embedded in HDR displays. This approach generates a map of saturated pixels (with value greater than a set threshold), which is then blurred producing a *smooth brightness enhancement* image similar to *expand map* of the preceding approach. In addition to this image, an *edge stopping function* is used to preserve contrast near sharp edges – a flood-fill algorithm is used to find borders of the "near-saturated" areas, which need to be expanded, preventing highlights bleeding over steep gradients. The smooth brightness enhancement map is then multiplied by the *edge stopping* map producing a map describing how the brightness of each input image pixel should be expanded. This approach can be implemented by using multi-resolution image pyramids for efficient filtering and flood fill computation.

A more recent approach by [Huo et al., 2014] is inspired by human vision system and proceeds by modelling human retina response, which locally adapts to a certain luminance level and perceives values in limited dynamic range around this level. The input LDR image is split to luminance and chrominance channels. The luminance channel is then transformed by the modelled inverse tone mapping function, which also takes a filtered version of the image as an input for estimating the local surrounding luminance. Finally, the HDR output is generated by combining the obtained luminance with the original chrominance channel.

### 1.1.3 Clipped data restoration methods

The previous approaches are not able to recover data in the underexposed or overexposed areas – at most they generate a gradient in the luminance channel causing smoother transitions and reducing the quantization artifacts. While the clipped regions can be easily detected, restoration of the missing contents is non-trivial.

[Wang et al., 2007] use inpainting to replace the overexposed and underexposed areas with correctly exposed patches from the input image. While this method convincingly restores lost details in the clipped areas, it requires manual

annotation to select, which areas should be used as source of the texture. Two similar approaches by [Jain et al., 2014] and [Savoy et al., 2014] use Internet image retrieval to find similar images, which are then used for inpainting the clipped areas.

The approach presented by [Rouf et al., 2012] attempts to recover lost color information in highlights caused by pixel value saturation in individual color channels. The hue and "internal texture" (or gradient) of the clipped area are derived from the un-clipped border of the clipped region. The luminance can be also partially reconstructed by maintaining the individual color channels' ratio based on yet-unclipped color channels. The maximum luminance is reached when all color channels are saturated.

### 1.1.4 Machine learning-based methods

During the last year several machine learning-based approaches have been published. These approaches use convolutional neural networks (or CNNs), previously used primarily for object classification tasks.

Thanks to the convolutional neural network being able to learn to recognize high-level objects in images, this information can be used for estimating the relative luminance of the object. For example when an object is classified as a light source such as the Sun or a street lamp, the luminance can be estimated accordingly. The CNNs are generally good at recognizing textures so normal/non-emissive objects and surfaces can be "classified" in similar fashion e.g. causing the luminance expansion on diffuse surfaces to be "smoother" than on glossy surfaces. Thanks to the data-driven training, the neural network approaches derive the dynamic range reconstruction "rules" automatically, but it also makes the reconstruction process less transparent.

Some of the convolutional neural networks designed for image processing tasks such as super-resolution and denoising are based on the autoencoder architecture, which is also suitable for dynamic range reconstruction. An autoencoder network consists of an encoder (a sequence of convolutional layers) followed by a decoder (deconvolutional layers). The encoding stage reduces the dimensionality of the input, creating a bottleneck, which forces the network to learn a compressed internal representation of the input image, leading to higher-level abstraction of the image. This representation is then reconstructed by the decoder part to produce the desired output. In order to preserve high-frequency details of the input image, *skip connections* may be used to pass intermediate outputs of the encoder layers to the decoder layers.

Unless stated otherwise, the following architectures are end-to-end, meaning they take an LDR image as an input and directly produce the resulting HDR image.

The first published CNN-based dynamic range reconstruction method *HDR-CNN* was designed by [Eilertsen et al., 2017]. It uses an autoencoder network to *hallucinate* details in over-exposed parts of the image. The encoder consists of convolutional layers from the VGG-16 architecture [Simonyan and Zisserman, 2014], a then-state-of-the-art classification CNN. By using a verified-working and already pre-trained neural network as a part of the network architecture, only the decoder weights need to be fully trained, saving considerable computational

resources. To reduce the amount of information the network needs to output, only the saturated areas in the input image are reconstructed, keeping the well-exposed pixels unmodified.

Another approach *DrTMO* by [Endo et al., 2017] aims to recover the whole inverse tone mapping of an image, reconstructing lost/clipped details in both highlights and shadows, by using an encoder-decoder architecture. To make training of the network feasible, the network outputs only 8-bit images, while simulating/predicting increased and decreased exposures of the input image. The resulting HDR image is produced by merging these outputs. The up-exposure and down-exposure models share the same network structure, but the weights (trained parameters) differ as the authors assume the up-exposure and down-exposure problems are based on different features.

The next approach by [Zhang and Lalonde, 2017] uses an autoencoder-based network to increase the dynamic range of an outdoor spherical panorama as well as to recover the elevation of the Sun from the "bottleneck" image representation. In addition, this network expects the Sun to be located horizontally in the center of the input image. As the network outputs an HDR image, which increases the complexity of the training, the authors opted to limit the size of the network by fixing the resolution to 128 x 64 pixels, which makes it unsuitable for high-resolution HDR image reconstruction.

The following approach by [Marnerides et al., 2018] attempts to solve the problem with occasional checkerboard artifacts, which are caused by deconvolution layers used in the upscaling decoding stage in autoencoder-based networks. The authors propose the *ExpandNet* network architecture without any upscaling steps – a custom multiscale CNN with 3 branches, each focusing on specific level of detail, which are then merged, producing an HDR image. The authors claim this approach handles ill-exposed input images containing large clipped areas better than other CNN-based dynamic range reconstruction approaches, producing fewer artifacts.

The last of the recently published approaches by [Ning et al., 2018] aims to solve the issue of limited training dataset size by using generative adversarial network. The proposed architecture consists of two neural networks, which are trained simultaneously: an inverse tone mapping network and a discriminator network. The dynamic range expansion network architecture is based on U-Net [Ronneberger et al., 2015] – an autoencoder-like network originally designed for image segmentation; the discriminator evaluates whether the reconstruction is correct by trying to distinguish generated images from real HDR images. The main advantage using the discriminator is that it allows the authors to use just low dynamic range dataset instead of HDR-LDR pairs when training the network.

# 2. High dynamic range reconstruction evaluation framework

This chapter describes the design decisions and implementation of the HDR reconstruction evaluation framework.

## 2.1 Motivation and requirements

The main goal of the evaluation framework is to compare the reconstruction quality of several dynamic range expansion algorithms used on spherical panorama images intended for image-based lighting.

To evaluate the performance of reconstruction algorithms, each algorithm needs to be run on an LDR input image to produce an HDR output, which can then be compared with a correct (or ground truth) output image using an error metric.

In order to obtain corresponding *LDR-HDR (ground truth)* image pairs, LDR image may be generated by a tone mapping operation such as simple dynamic range clipping of the HDR image followed by quantization to 8-bits per channel.

Differently sized input images should not influence the value of error metrics, e.g. the framework should not favor smaller images.

To find cases where individual reconstruction approaches yield unsatisfactory or subpar results, many input images need to be processed. Therefore, batch processing of multiple input images with one command is required.

The framework should also present the results in user-friendly way allowing the user to compare each reconstruction result with the ground truth image as well as with other reconstruction results. Similarly, the user should be able to compare the use of reconstructed spherical panoramas as environment maps in in a simple test scene.

The implementation should allow easy re-running of the pipeline for changed input dataset. It should be also simple to modify, e.g. addition of new reconstruction algorithms and error metrics is expected.

## 2.2 Platform

The main function of the framework is to launch programs implementing algorithms for dynamic range reconstruction and other programs preparing their input data and processing the results. Because compiled programming languages would require an extra compilation step before each run after modifying the program, a *scripting* programming language is more suitable. As Linux compatibility is desirable for running the evaluation pipeline on headless machines such as servers, *POSIX shell* was chosen as a fitting programming language for the framework implementation. *Python 3*, which is used by some of the image

processing "sub-tools", was also considered, but due to author's limited knowledge of the Python language syntax at the time of choosing, *shell* was preferred. The choice of *Python* would simplify data passing between individual parts of the program and the support for data structures such as list/set or dictionary might have simplified the code. On the other hand, launching a subprocesses in *Python* would need marginally more lines of code.

To simplify the installation of image manipulation tools (pfstools), which are not easy to compile, Docker virtualization system is used to provide the required environment in a container.

## 2.3 Stages

The evaluation pipeline consists of the following stages, each of which is implemented by one or more shell scripts. The "main" script all.sh sequentially runs all the stages for an HDR image given as input. When batch processing of all files in a folder is desired, the script all_batch.sh should be used. Paths to individual external programs and other parameters are stored in the file _settings.sh.

Instead of executing all stages by the all.sh script, any script can be run manually when only one step should be executed. The scripts print a short help/usage message when run without valid parameters.

### 2.3.1 Import

The aim of this stage is to convert the input HDR image into a common format and resolution. OpenEXR format was chosen as it provides high compression ratio thanks to wavelet transform. The resolution of the input images is unified by resizing/resampling them to prevent unfair error metric results caused by different image pixel counts.

This stage is implemented in the convert.sh script by executing several utilities from the pfstools package.

### 2.3.2 Degrade

This stage simulates dynamic range and luminance precision degradation by converting the input image into a lossy LDR format. The JPEG compression and sRGB color space is used, as it represents the majority of taken photos. The compression quality (DCT coefficient pre-quantization multiplier) is set to maximum to minimize the effect of compression artifacts on the evaluation metrics.

The stage is implemented in hdr_to_ldr.sh script. In case a different tone mapping algorithm is required, this stage can be modified.

### 2.3.3 Reconstruct

This stage handles dynamic range reconstruction by running each reconstruction algorithm.

Along with each reconstruction result, a non-reconstructed image is passed to the next stage to serve as a baseline for metrics evaluation.

Each reconstruction algorithm is wrapped in a script ldr_to_hdr_[reconstruction name].sh, which handles implementation-specific details of the respective algorithm to produce correctly named output file for the following stage.

### 2.3.4 Evaluate

The reconstruction quality needs to be evaluated, which is the task of this stage. The evaluation consists of comparing the reconstruction result to the ground truth image by error metrics computation and *luminance* and *color mapping* plot generation.

We use two error metrics to "numerically"/objectively evaluate the reconstruction quality: RMSE and SSIM.

RMSE (root-mean-square error or "L2 error") is computed as

$$RMSE = \sqrt{\frac{\sum_i^N \left(L_{GT,i} - L_{R,i}\right)^2}{N}}, \tag{2.1}$$

where $N$ denotes the number of pixels of an image and $L_{GT,i}$ and $L_{R,i}$ denote the luminance of $i$-th pixel of the ground truth or reconstructed image, respectively. As each pixel pair is equally important regardless of position or brightness, this metric does not ignore any kind of error, as perception-based metrics do. The evaluation is split to two image "areas": saturated (pixels having luminance value greater than or equal to 1) and unsaturated pixels, allowing separate measurement of the reconstruction quality of clipped highlights and of the distortion of well-exposed pixels' values.

*SSIM* by [Wang et al., 2004] measures the *structural similarity*, aiming to model the human perception of image differences. While it was originally used for measuring the quality of lossy compression algorithms, it is also suitable for comparing details preservation. This metric is evaluated by computing statistical properties of pixels in a *sliding* window so it cannot be easily used to evaluate only saturated/unsaturated pixels, therefore, unlike the RMSE evaluation, only one numeric result is produced for an input image pair.

Another metric *HDR-VDP-2* by [Mantiuk et al., 2011] was considered as a suitable metric, as it aims to measure perceived differences in HDR images, but it was not integrated for technical reasons, as no implementation, which would not require installation of large bundle of statistical software was found. This metric can be added in the future, but currently a smaller footprint and fewer dependencies of the framework were preferred.

Apart from numeric evaluation of reconstruction results by computing error metrics, two graphs representing luminance and color transformation are generated.

A *luminance mapping plot* is a scatter plot containing a dot for each image pixel – the dot is drawn at the x-position equal to the pixel luminance in the ground truth image and y-position equal to the luminance in the reconstruction image. Along with the image data, a function $f(x) = x$ is plotted to help spot any nonlinearity of the reconstruction algorithm – an optimal reconstruction algorithm should contain points only on this line.

Similarly, by drawing each color channel separately instead of combining the color channel values into luminance, drawing red, green and blue dot for each

pixel, a *color mapping plot* is generated. It shows the distortion of individual color channels allowing the user to inspect whether possible reconstruction errors are color-dependent. These plots may help the user spot any global pixel value distortion or e.g. in a specific case of a photo of a landscape, the *color mapping plot* may help discriminate whether a reconstruction algorithm fails to recover the dynamic range in the sky or grass area.

Example of the mapping graphs is shown in Figure 2.1.

This stage is implemented by the evaluate.sh script running the metrics evaluation and graphs generation.

### 2.3.5 Render

Using the reconstructed HDR image from the previous stage as an environment map, a simple 3D scene is rendered (path-traced) to visualize the effects of using the reconstructed image for image-based lighting.

The 3D scene consists of three glossy spheres with different surface roughness placed on a checkerboard-textured plane. The scene rendering is depicted in Figure 2.2.

High intensity light sources, which occupy a small area on the environment map image, such as Sun, cause crisper shadows and larger highlights on glossy surfaces. The relative intensity of such sources can therefore be estimated by visually inspecting the size of the highlights. Comparing the rendering outputs (at a fixed exposure level) produced from a reconstructed image and the corresponding ground truth image can be used to assess the quality of the reconstruction algorithm. The scene is rendered and stored in high dynamic range to allow simulation of different exposures when viewing.

This step is implemented in render.sh, which runs the *pbrt* renderer, an open-source physically-based renderer [Pbr] to simulate the light transport in the scene.

### 2.3.6 View

This stage prepares a user-viewable version of reconstruction results along with the ground truth image for comparison. The user can set an exposure value and switch between images (individual reconstructions and the ground truth image). Switching between images maintains the exposure value allowing the user to compare specific parts of the image. The HDR image viewer is based on web technologies (HTML5 and JavaScript/ECMAScript) for platform independence and simple result publication. Figure 2.3 shows an example of the viewer graphical user interface.

This stage is implemented by the view.sh script. The single-image viewer code and data is generated by the pfsouthdrhtml tool by [Mantiuk and Heidrich, 2009], which generates multiple simulated exposures to be alpha-blended by precomputed coefficients at runtime to produce a desired exposure value. The image switching functionality is added as a part of the *HDR reconstruction evaluation framework*.

Currently this viewer should be only used as a preview of the images, with important observations being verified in another viewer application, as there is a known issue where two input HDR images with very different maximum pixel

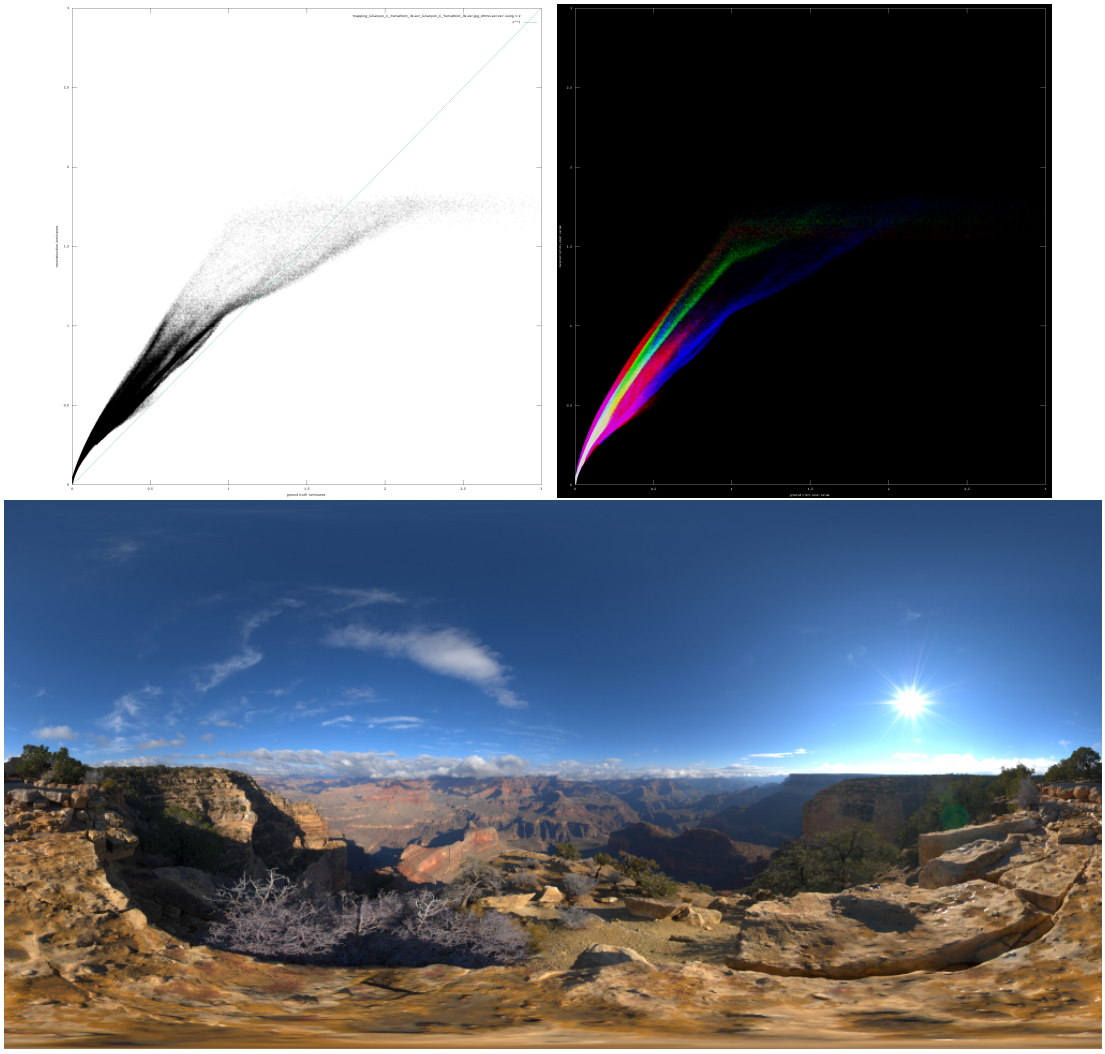


Figure 2.1: Luminance and color mapping graphs for the DrTMO reconstruction method on the GCanyon_C_YumaPoint_3k image (pictured below the mapping graphs).

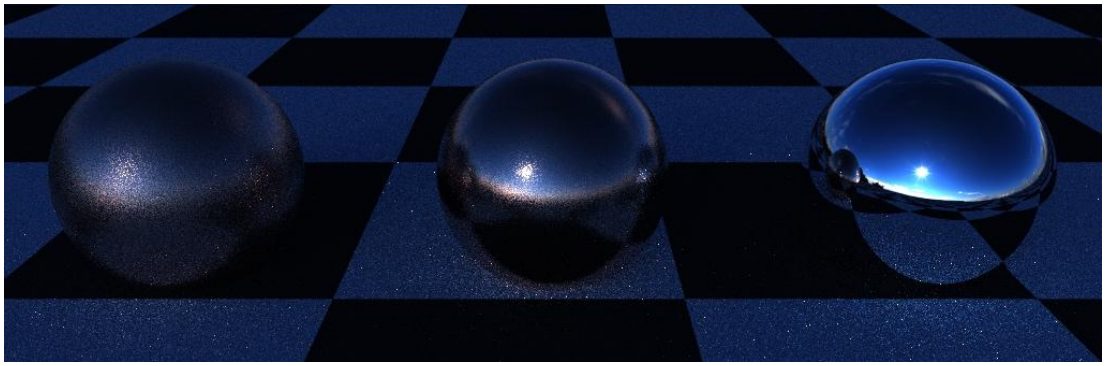


Figure 2.2: Rendering of the testing scene using the reconstructed image applied as an environment map for image-based lighting.

values may appear having a slightly different brightness, although the original HDR pixel values and the requested exposure are the same. It is caused by the "pre-rendered-then-blended exposures" approach as a fixed number of pre-rendered exposures is used to capture any dynamic range. The effects of this issue were reduced by modifying the pfsouthdrhtml tool by increasing the base exposure value precision, but the issue was not fully resolved. Because the issue was discovered relatively late, a complete solution was not implemented. To achieve better preview quality, a different client-side JavaScript HDR viewer e.g. jeri.io [Jer], which was developed concurrently with this work, needs to be integrated into the framework or the HDR images may be compared in a desktop application such as *tev* [Tev].

### 2.3.7 Visualize the evaluation results

The task of this last stage is to present the evaluation result to the user – generate a web-page with an interactive table containing numeric evaluation results along with links to open the HDR image viewer with a given set of images and color/luminance mapping graphs. An example of this page was shown on Figure 1 in the Introduction section.

Clicking the ground truth file name opens the HDR image viewer from the previous stage so individual reconstruction algorithms and their application as an image-based light source can be compared with each other and with the ground truth image. The links in the Mapping column open luminance and color mapping graphs from the *Evaluate* stage.

The table allows sorting by clicking on a column header; secondary sorting criteria can be set by holding the shift key when clicking a header. This allows the user to quickly find best and worst reconstruction results, which can be inspected in more detail. Multi-criteria sorting simplifies the process of finding the ranking of individual reconstruction algorithms by choosing the *Ground truth image* column as a primary key and an error metric as a secondary key. The sorting functionality is provided by the TableSorter library [Tab].

This stage is implemented by the evaluate_view.sh script, which generates the page index.html in current working directory and copies all required resources.

## 2.4 Framework implementation

### 2.4.1 Source code

The HDR reconstruction evaluation framework code is stored in the pipeline folder in the *electronic attachment* of this thesis. The code is also available in the Git repository[1], where the most recent version is available.

### 2.4.2 System requirements

POSIX shell interpreter with standard POSIX.1-2004 tools (cut, sed, find, awk, ...), Python 3 (and 2) (with OpenCV, SciPy and NumPy libraries installed), Docker and imagemagick (compose tool).

[1] `https://bitbucket.org/hdri/pipeline`

The framework has been tested on macOS 10.12.6, but should work on any POSIX-compliant system.

### 2.4.3 Usage

The usage instructions are included in the Attachment A.2.

A high-level example of practical usage of the evaluation framework and interpretation of the evaluation results is shown in the Results chapter.

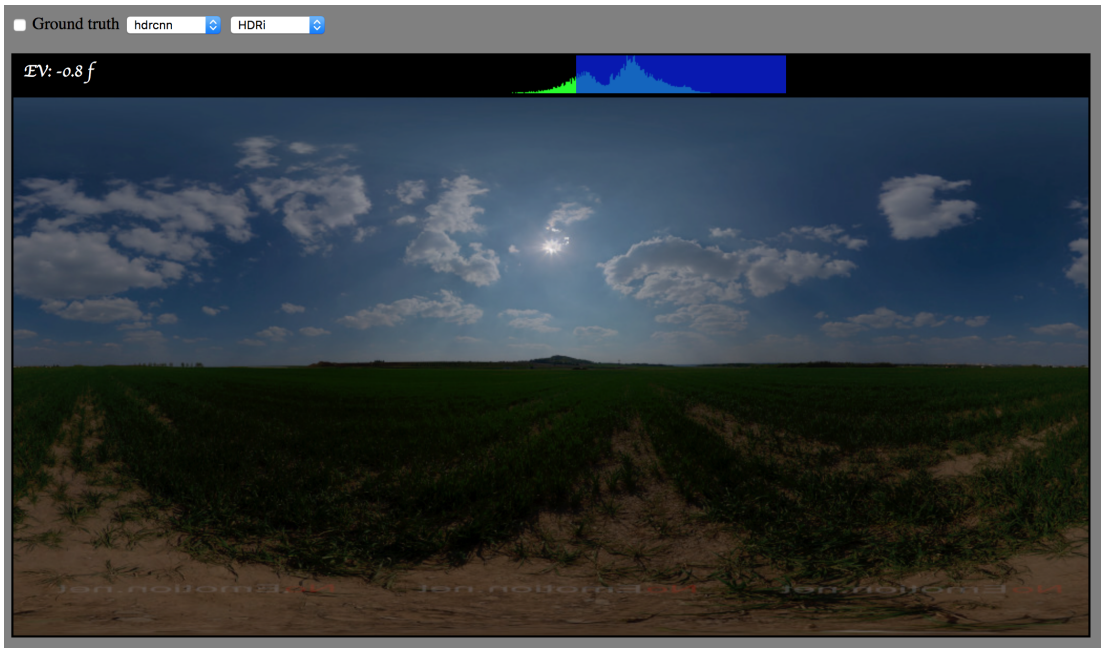



Figure 2.3: Screenshot of the HDR image viewer for visually comparing the output of a chosen reconstruction method with the ground truth environment map.

# 3. HDR camera application

## 3.1 Goal

The application should be able to capture the highest possible dynamic range of a scene, possibly recovering details that could not be captured, and saving the photo in a format, which does not degrade the image quality.

## 3.2 Platform

Apple iOS was chosen as the target platform because of author's previous experience with development for this platform and because the author already owned a compatible smartphone. The application has been developed and tested on an iPhone SE with the iOS 10.2 operating system installed. The device has a dual core 64-bit ARMv8-A CPU clocked at 1.8 GHz, 2 GB of RAM and a 6-core GPU allowing execution of Metal compute shaders.

## 3.3 Method outline

When an image is captured, its dynamic range is limited by the sensor sensitivity and saturation and by the storage format. This limit can be overcome by taking multiple photos with different camera settings a technique called bracketing. Dedicated cameras usually allow manual setting of exposure time, ISO and aperture, which all affect the exposure. In case of smartphone cameras with fixed aperture only exposure time and ISO settings are supported by the hardware.

Bracketed photos can be merged by linearization (applying inverse camera response function), multiplication by their individual exposures and computing the weighted sum, which prioritizes well-exposed pixel values. The detailed process is described by [Debevec and Malik, 1997] or [Robertson et al., 1999] along with an algorithm to find the inverse camera response function.

Dynamic range can be further improved beyond the physical sensor limits (e.g. the shortest exposure time and the lowest supported ISO) by hallucinating the clipped pixels using a CNN-based approach.

## 3.4 Camera overview

The back camera of iPhone SE has a 12 megapixel sensor capturing still images with resolution 4032 by 3024 pixels. The supported exposure time range is 0.000013 s – 0.333333 s and ISO range is 23 – 1840, as described in the documentation [App, b].

The camera API is asynchronous, event-driven: when a capture is requested, a delegate object must be provided – the delegate needs to handle callbacks representing events such as "a photo was taken" e.g. by compressing the image[1]

[1] AVCapturePhotoOutput.jpegPhotoDataRepresentation(forJPEGSampleBuffer: photoSampleBuffer, ...)

and saving the result to a file or the photo library.

An application can request a bracketed capture, the iPhone SE with iOS 10.2 accepts up to 4 bracket settings at once.

Apart from the pre-processed Y'CbCr image an application may request a RAW image capture (introduced in iOS 10). The iPhone SE supports capturing RAW images in the "bgg4" pixel format – a 14-bit Bayer format [Sta, a], which contains 12 bit values[2].

## 3.5 Existing solutions

The most straight-forward option to capture photos is the built-in camera application. This application supports "HDR" mode, which automatically activates if the scene dynamic range significantly exceeds the range, which can be captured with just one exposure. Despite the name of this mode, the resulting image has low dynamic range – the output is tone-mapped and stored with JPEG compression.

While the default camera application does not explicitly allow manual setting of capture parameters, bracketed image capture is possible: when the capture settings are locked by long pressing on the camera preview, relative exposure can be adjusted by swiping up/down, which allows the user to take multiple pictures with varying exposure value. Although this theoretically allows capturing of bracketed images to be merged later using appropriate software, the time between captures is significantly longer than if the pictures were taken automatically, making the capture of dynamic scenes infeasible without additional processing that would compensate for the movement and remove ghosting artifacts. Also the camera movement caused by manually adjusting the exposure value would likely cause additional artifacts in the merged image.

There are third-party camera applications available in the App Store, which allow manual settings of exposure parameters (e.g. Musemage), automatic exposure bracketing (e.g. ProCam) or even bracketed capture with integrated merging of the bracketed photos (e.g. Adobe Lightroom CC). Adobe Lightroom CC takes 3 RAW photos with different exposures, merges them and saves the result to a 32-bit DNG file [Ado].

## 3.6 HDR Capture Pipeline

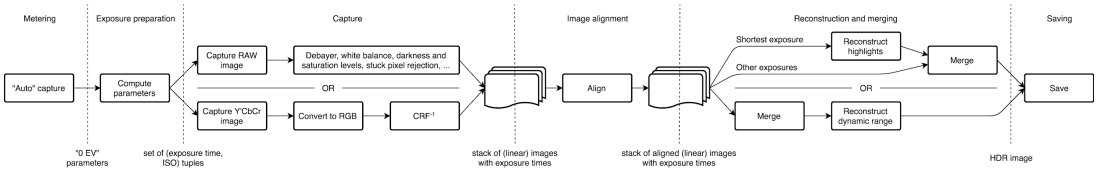



Figure 3.1: Schematic diagram of the HDR capture pipeline.

[2]Measured by taking a RAW photo of a bright scene with clipped pixels and looking at the highest value in dcraw: running 'dcraw -v photo.dng' printed 'Scaling with [...] saturation 4095'.

The HDR capture pipeline is based on the "Stack-Based Algorithms for HDR Capture and Reconstruction" chapter from a book by [Dufaux et al., 2016], adapted to fit the use-case of a mobile application, which would make taking environment maps easier, with an added dynamic range reconstruction step. The pipeline phases and the description of data passed between individual phases are outlined in the Figure 3.1.

### 3.6.1 Metering

The aim of this phase is to find parameters of the base exposure, where most of the pixels values are not clipped. Measuring the dynamic range of the scene also belongs in this phase.

In our application this information may be set by the user by specifying the base exposure parameters (with an option to measure them automatically) as well as setting the desired number of down-/up-exposure steps and the EV step size, describing the dynamic range of the scene.

### 3.6.2 Exposure preparation

This phase's task is to generate the set of exposure parameters – orders for the camera – to capture the requested dynamic range. This can be done by choosing a set of exposure values (EV), which cover the scene's dynamic range.

The exposure value [Wikipedia contributors, 2018a] is defined by the following equation:

$$EV = \log_2 \frac{N^2}{t}, \tag{3.1}$$

where $N$ is relative aperture and $t$ is the exposure time. The relation between ISO and time is following:

$$\frac{N^2}{t} = \frac{LS}{K}, \tag{3.2}$$

where $L$ is the average scene luminance, $S$ is the ISO value and $K$ is a constant.

This means increasing the EV by 1 can be achieved by doubling the exposure time or ISO. The exposure time and ISO are dependent on each other in a way, that when the exposure time is divided by $n$, ISO needs to be multiplied by $n$ to keep the same exposure value.

The camera API offers two modes of operation influencing what kind of exposure parameters is used: "Auto" mode[3], where the application specifies the requested list of EV offsets from the automatically determined "+0 EV" level, and "Manual" mode[4], which takes list of arbitrary (ISO, exposure time) tuples.

The length of the list of exposure parameters is limited by hardware, in case of the iPhone SE only 4 images can be requested to be captured in one request. This limitation can be overcome by requesting multiple captures in succession at the risk of camera movement between captures.

The nontrivial part of the parameter generation process is maintaining balance between the exposure time and the ISO to avoid motion blur and noise caused by high values of these parameters. Usually this is achieved by using heuristics

[3] AVCaptureAutoExposureBracketedStillImageSettings
[4] AVCaptureManualExposureBracketedStillImageSettings

such as an estimated maximum “motion-blur-free” exposure time, which can be used when the camera is held in hand. The iOS 10.2 uses maximum exposure time of 1/17 s when using the “Auto” exposure mode; when a higher exposure value needs to be achieved, ISO is increased.

As the ”Auto” mode is varying both the ISO and the exposure time, we decided to use the ”Manual” mode. While this gives us more control over the capture settings, mainly the ability to take multiple images with the same set of exposure parameters, which is crucial for taking panoramas[5] and simplifies further processing, we have to determine the ”correct” or base exposure parameters ourselves.

To simplify the finding of base exposure parameters we take an image with automatic exposure (or “+0 EV” point of the camera-determined ”correct” exposure) and access the exposure time and ISO metadata when the image is captured. From these exposure parameters we compute additional (exposure time, ISO) tuples to capture the dynamic range requested by the user. Up-exposure (“+x EV”) parameters are computed by increasing time up to 1/17 s, then by increasing the ISO; for down-exposure (“-x EV”) we decrease the ISO until reaching the lower limit in order to reduce the noise, then we decrease the exposure time.

### 3.6.3 Capture

This phase takes care of taking the required exposures of the scene, producing a set of images, also called a stack.

The camera API allows capturing either RAW or processed images. RAW images should provide more information per-picture providing 14 bits of precision for each sensor pixel, while the processed formats provide only 8 bits per channel.

#### RAW

When using the RAW format, capturing the required dynamic range would need taking less photos, but RAW processing would require more work as the conversion to RGB is not trivial. We would need to measure sensor “darkness” and saturation levels, handle stuck pixel rejection, white balance and debayering/demosaicing on our own. Integration of a third party tool or library such as dcraw [Dcr] or LibRaw [Lib] would make the process easier, but still not as easy as using the built-in image processing.

#### Processed

The processed image can be retrieved in the 8-bit planar Y’CbCr format[6] or converted to an BGRA[7] bitmap with 8 bits per channel. After attempting to use the BGRA format to avoid manual conversion to RGB, getting completely black images, although the format was specified as supported by the camera API, we opted to use the Y’CbCr format, which we convert to RGB using the cvtColor function of the OpenCV library [Ope].

[5] Constant exposure parameters make stitching of both HDR and LDR panoramas easier.
[6] kCVPixelFormatType_420YpCbCr8BiPlanarFullRange
[7] kCVPixelFormatType_32BGRA

While the processed format does many steps, which simplify our work, it also involves dynamic range reduction by tone mapping the sensor's 14-bit range to 8 bits. This requires us to linearize the sensor-processing system response by finding and applying an inverse camera response function.

The camera response function is a function $f(\mathbb{R}^+) \rightarrow [0, 255]$, which transforms the incoming radiance to a pixel value. The calibration process outlined by [Debevec and Malik, 1997] or [Robertson et al., 1999] consists of taking multiple images with different exposure values[8] of the same scene without moving the camera and running an optimization algorithm. The optimization process finds the camera response curve by minimizing the error when estimating the incoming radiance using the inverse camera response function while compensating for the known exposure time.

We have chosen to use the processed image format, as the conventional image processing such as bracketed exposure merging is well-documented; RAW capture can be implemented later as an optimization to reduce the number photos of that need to be taken for the same dynamic range.

### 3.6.4 Image alignment

This step prevents incorrect exposure merging results caused by camera movement between captures of individual bracketed images.

We use a photo alignment approach by [Ward, 2003], which aligns the differently exposed images by translating them to compensate for the camera movement. The author claims the translation correction is sufficient as the majority of handheld captured sequences do not require rotational alignment. This algorithm is implemented in the OpenCV library in the AlignMTB class.

### 3.6.5 Dynamic range reconstruction and merging

This phase aims to increase the dynamic range beyond the camera limits.

#### Dynamic range reconstruction approach selection

The first thing to consider is the choice of a suitable dynamic range reconstruction approach.

The most important criterion is the performance and capabilities of each approach. As shown below in the Results chapter, HDRCNN is the best-performing of the compared networks. However, as it only restores clipped highlights without modifying the un-saturated areas with shadows, DrTMO and ExpandNET also need to be considered to be used.

The implementations of the DrTMO and ExpandNet CNNs, which use the Chainer neural network framework [Cha] and PyTorch [Pyt], respectively, are not compatible with iOS, requiring network models to be converted to a supported machine learning framework such as CoreML [Cor]. This leaves HDRCNN, which is based on the TensorFlow library [Ten, a], the only option, which does not require framework change for running on iOS.

[8] When calibrating the camera, the ISO was fixed to the lowest value and only the exposure time was being changed, because varying the ISO would change the noise levels.

System requirements need to be also considered due to the limited memory and processing power of the mobile device. The authors of ExpandNet declare their network to have higher memory footprint than HDRCNN, which was confirmed by our observation. Similarly DrTMO implementation is also very demanding, requiring up to 13.48 GB for the default input image[9]. This would require modifications to the network architectures[10] to reduce the requirements.

Another matter to consider when choosing the reconstruction approach is the license used by each approach implementation. HDRCNN and DrTMO source codes are published under the *BSD 3-Clause* license, allowing even "closed-source" commercial distribution, while ExpandNet code is intended only for scientific purposes.

Because of these reasons we selected HDRCNN as the most applicable approach.

### HDR reconstruction position in the pipeline

There are multiple possibilities when the reconstruction can occur.

The dynamic range can be increased by processing either one or more images. This determines the sequence of the dynamic range reconstruction and bracketed exposure merging.

Single-image reconstruction of the clipped highlights of the lowest-exposure-value image[11] and/or reconstruction of details in the highest-exposure-value image by simulating up-exposure need to be followed by merging the results with the other captured images to produce the HDR image.

Another possibility is to merge the exposures first and then feed the result into a neural network for further dynamic range expansion after scaling the HDR image pixel values to the range expected by the neural network.

However, neither of these options is straightforward: both reconstruction and merging expect or require LDR inputs, therefore either needs to be adapted.

The reconstruction of the lowest-exposure-value image would require merging the reconstructed HDR image with the remaining (unreconstructed) LDR images, requiring a modification of the merging algorithm, while the the merged HDR image passed to a neural network would likely decrease the image quality, as the CNN has not been trained with HDR inputs.

While the latter approach may theoretically produce better results as the network would have access to the full captured dynamic range image, training the CNN to process such inputs would be necessary. Therefore we chose to use the former approach – reconstruction of just the lowest-exposure-value image followed by a merge using a modified merging algorithm described below in the Merging subsubsection.

[9] Forest.png with 1536 by 1024 pixels resolution, provided with the DrTMO code.

[10] E.g. DrTMO may be modified to output less exposures at the same time, which would require the network evaluation to be run multiple times sequentially, increasing the computational demands.

[11] essentially estimating/simulating how even lower-exposure-value images would look

### Dynamic range reconstruction on mobile devices

The easiest approach would be to execute the computationally demanding dynamic range reconstruction algorithm on a higher-performance server in order to speed the process up and avoid possible compatibility issues, there would be tradeoffs in form of hosting costs and relatively high bandwidth requirements[12] also impacting the users of the application. Therefore we have decided to evaluate the feasibility of running the dynamic range reconstruction – and inference of large CNN models in general – directly on a mobile device.

As the HDRCNN code is written in the Python programming language and Python execution is not officially supported on iOS, a native code must be used. Fortunately TensorFlow compilation for iOS is possible and the library allows the model (network structure along with the trained parameters) to be exported from the Python implementation and imported in the native code, requiring only the inputs and outputs to be correctly mapped.

As the TensorFlow versions used for exporting and importing the model must match and because the provided pre-built library in the CocoaPods dependency manager [Coc] was too old, we have built the matching version from the source code[13].

**Model preparation** We have used the TensorFlow Mobile manual and reference [Ten, c] as the outline for the model preparation workflow.

As the HDRCNN code generates the network structure on-the-fly based on the set image resolution, the model needs to be exported using the simple_save function after the network is constructed and the trained weights are loaded[14]. This produces a saved_model.pb file, which needs to be further processed.

As the network input and output are referenced only by Python variables, we need to find names of the corresponding nodes. This can be done using the saved_model_cli tool[15] provided as a part of the TensorFlow library. The tool prints the needed node names – input "Placeholder", which represents the input image, and output "add_7", which is the node computing the addition of the alpha-blended reconstruction to the input image.

Knowing the input and output node names, we may proceed by preparing the model for mobile using the freeze_graph tool[16]. These steps are sufficient to

[12]"Relatively high" – in context of the price of mobile LTE connectivity and data limits. Upload LDR: ˜2-5 MB (lossy JPEG LDR image), download HDR: ˜35-40 MB (Radiance HDR (RGBE) format, better compression might be achieved with wavelet transform used by OpenEXR).

[13]We followed build installation notes [Ten, b]. The version 1.8 was used, which we had to build using the following command: ANDROID_TYPES="-D__ANDROID_TYPES_FULL" tensorflow/contrib/makefile/compile_ios_tensorflow.sh -f "-O3" -h tensorflow/contrib/makefile/downloads/nsync/builds/default.macos.c++11/nsync.a -n tensorflow/contrib/makefile/downloads/nsync/builds/lipo.ios.c++11/nsync.a -a arm64 as the __ANDROID_TYPES_FULL macro enables all operations required by the saved model for multiple data types for experimentation with memory footprint reduction. Note: the build process fails if the working directory path contains a space character.

[14]This can be done by adding "tf.saved_model.simple_save(sess, './export', inputs={"x": x}, outputs={"y": y})" to the hdrcnn_predict.py script just after assign_params is called. A folder called "export" will be created when the script is run, producing a saved model.

[15]saved_model_cli show --dir export/ --tag_set serve --signature_def serving_default

[16]This tool needs to be built from source by running bazel build

produce a model that can be loaded in the native application.

**Running on the device** The TensorFlow library contains code sample project tf_simple_example, which runs a classification network on an image. This code has been used as a base for the experiments.

When the inference was run on the mobile device (using the default resolution of 1024x768), the program crashed when allocating about 1.3 GB of memory, which is the maximum amount of RAM an application is allowed to allocate on a 2GB device [Sta, b].

While the model size is relatively large for running on a mobile device (approximately 117 MB on-disk file size), it easily fits into memory. The issue is the *runtime* memory allocation when running the inference.

The benchmark_model tool[17] reports approximately 4.5 GB memory usage of the frozen_graph.pb model for 1024 by 768 pixels RGB input image size – probably an upper limit assuming all nodes' values are loaded in memory at the same time as only 3.2 GB peak was observed when running the HDRCNN Python script to verify this result. As the memory footprint is beyond the device's limits and no obvious method to overcome it during execution is available[18], we looked at the optimization and network size reduction possibilities.

**Optimization and reduction attempts/options** First we have tried optimizing the model using the TensorFlow-provided tools.

The strip_unused tool[19] removes the nodes, which are not used for evaluation of a specified output node. While this tool would simplify the graph of e.g. a model with multiple outputs when only one output is required, the HDRCNN's architecture does not contain any unnecessary nodes, leading to no reduction in size.

Another optimization we attempted is using the optimize_for_inference tool[20]. This tool attempts to remove unused nodes as well as to use pre-calculated con-

tensorflow/python/tools:freeze_graph, then can be run with bazel-bin/tensorflow/python/tools/freeze_graph --input_saved_model_dir /pwd --output_graph=/pwd/frozen_graph.pb --output_node_names 'add_7', where pwd represents the working directory path – we have run a Docker container in the export folder to reduce the need for dependencies: docker run -it -v "$(pwd)":/pwd tensorflow/tensorflow:latest-devel. A specific container version such as tensorflow/tensorflow:1.8.0-devel may be used instead if needed.

[17] bazel build tensorflow/tools/benchmark:benchmark_model, bazel-bin/tensorflow/tools/benchmark/benchmark_model --graph=/pwd/frozen_graph.pb --show_flops --input_layer=Placeholder --input_layer_type=float --input_layer_shape=1,768,1024,3 --output_layer=add_7

[18] System-level memory swapping/paging is not officially supported on iOS and application-level swapping or hacks such as forgetting and recomputation of network node values when evaluating the feed-forward network by parts would slow down the computation.

[19] bazel build tensorflow/python/tools/strip_unused, bazel-bin/tensorflow/python/tools/strip_unused --input_graph /pwd/frozen_graph.pb --output_graph /pwd/frozen_graph_stripped.pb --input_node_names Placeholder --output_node_names add_7 --input_binary --output_binary

[20] bazel build tensorflow/python/tools:optimize_for_inference, bazel-bin/tensorflow/python/tools/optimize_for_inference --input /pwd/frozen_graph_stripped.pb --output /pwd/frozen_graph_stripped_optimized.pb --input_names Placeholder --output_names add_7

stants more aggressively and to combine multiple nodes into one (e.g. resizing and convolution nodes). While not significantly reducing the model size, we measured approximately 5% speedup of the inference.

Looking at the output of the benchmark_model tool[21], measuring total memory usage by node type, we observed the most demanding node type to be Conv2D using 61 % followed by Conv2DBackpropInput operations using 22 %. Conv2D is a node performing the convolution operation, which is essential, but Conv2DBackpropInput appeared to be related to the training process of the network, unrelated to the inference process. However, because these nodes implement the conv2d_transpose operation[22] used in the "deconvolutional" layers in the decoder part of the HDRCNN, they need to be preserved.

When TensorFlow opens the saved model, it loads the contained trained parameters into memory. This can be avoided by converting the model to a format using the convert_graphdef_memmapped_format tool[23], which allows memory mapping of the parameters. Since only the network structure needs to be loaded into memory, the memory footprint is reduced by the parameters size. While saving approximately 117 MB of memory would help, the main problem with *runtime* memory allocation remains. We leave this optimization as an optional last step when packaging the application.

The CNN internally uses 32-bit floating point numbers. If we could reduce the precision we may be able to reduce the memory footprint.

TensorFlow supports graph parameters quantization[24], reducing values precision to 8 bits, reducing the stored model size to approximately 25 %. Unfortunately this only affects the on-disk format as all weights are "unpacked" or dequantized when loading the graph, only leading to lower parameter precision causing artifacts in the output image.

We also attempted reducing the "runtime" number precision by modifying the HDRCNN code to use float16 and recompiling TensorFlow with support for 16-bit operations, which should theoretically reduce the runtime memory footprint by 50 %. This attempt was unsuccessful[25] since not all the required operations/nodes are implemented for 16-bit operands. When running the precision-reduced HDRCNN Python code on an image with 1024x768 resolution, we saw lower memory

[21] bazel build tensorflow/python/tools:benchmark_model, bazel-bin/tensorflow/tools/benchmark/benchmark_model --graph=/pwd/g.pb --show_flops --input_layer=Placeholder --input_layer_type=float --input_layer_shape=1,768,1024,3 --output_layer=add_7

[22] conv2d_transpose documentation: "*This operation is sometimes called "deconvolution" [...], but is actually the transpose (gradient) of 'conv2d' rather than an actual deconvolution.*"

[23] bazel build tensorflow/contrib/util:convert_graphdef_memmapped_format, bazel-bin/tensorflow/contrib/util/convert_graphdef_memmapped_format --in_graph=/pwd/frozen_graph_stripped_optimized_quanized.pb --out_graph=/pwd/frozen_graph_stripped_optimized_quanized_mmapped.pb --min_conversion_tensor_size=100

[24] bazel build tensorflow/tools/graph_transforms:transform_graph, bazel-bin/tensorflow/tools/graph_transforms/transform_graph --in_graph=/pwd/g.pb --out_graph=/pwd/frozen_graph_stripped_optimized_quanized.pb --inputs=Placeholder --outputs=add_7 --transforms='quantize_weights'

[25] Running in iOS fails when loading the graph: "Invalid argument: Input 0 of node Pow_6 was passed float from Placeholder:0 incompatible with expected half." Maybe there is just a problem with the input image type specification and lack of built-in half-precision floating point numbers in C++.

usage with an observed peak of 1.9 GB. We could not verify the theoretical 50% footprint reduction as TensorFlow's benchmark script does not support 16-bit floating point inputs.

Since we could not achieve the footprint reduction by any other method, we have decided to lower the reconstruction resolution. We chose the resolution 320x320 pixels as these dimensions were used for training the network. This reduced the application memory usage to approximately 750 MB when running the inference, enabling the dynamic range reconstruction to be run on a mobile device, while still leaving free space to other data such as images with different exposure parameters. While dynamic range reconstruction of such small images would not be directly usable, high-resolution images can be split to tiles of the supported size and reconstructed independently sequentially.

To validate the necessity of the tiling approach, we attempted running the reconstruction on a full-resolution image from the iPhone camera using the original HDRCNN code, which allocated more than 14 GB before its process was killed, making the high-resolution image dynamic range reconstruction infeasible with the current network architecture.

The naive tiling introduces tiling artifacts – when a saturated area is overlapped with a tile edge, the predicted luminance value of one tile's reconstruction may differ from the other tile's predicted value, causing visible edges along the tile borders. This can be solved by overlapping the tiles and creating a linear transition between the reconstructed images. The overlap of the tiles would increase the number of tiles needed to cover the image, reducing the reconstruction performance.

While the reconstruction can be run on a mobile device, it is not practical as reconstruction of a each tile takes about 4 seconds using both CPU cores, making the reconstruction of the full-resolution image take approximately 335 seconds. For comparison running the same code (compiled to the x86_64 architecture) on a laptop with i7-4770HQ CPU took 147 s. As optimization with TensorFlow-provided tools does not yield significant speedup and since the reconstruction is using only the CPU, a GPU-accelerated computation needs to be considered.

Because TensorFlow does not support GPU acceleration with Metal compute shaders, other machine learning framework needs to be used. A 40% performance increase is documented [Mat] in case of classification of the MNIST digits dataset.

Converting or reimplementing the HDRCNN architecture in a mobile-GPU-supporting framework is left as a possible future work as well as determining whether increasing the tile size to maximize the memory usage and reduce the tile count would positively impact the performance. Additional performance could be achieved by changes to the network architecture such as reducing the number of convolutional layers, which would impact the reconstruction quality and would require re-training of the network.

Because of the performance issues, the dynamic range reconstruction code has not been merged into the main camera application repository and is kept separately in the tensorflow repository in the tensorflow/examples/ios/simple folder.

### Merging

This step merges multiple LDR images captured with different exposure values into one HDR image.

There are two widely used merging approaches, which also find the inverse camera response curve: Calibrate/*MergeDebevec* [Debevec and Malik, 1997] and Calibrate/*MergeRobertson* [Robertson et al., 1999]. While the merged image computation process is similar[26], different weighting functions, which estimate the variance of incoming light range represented by digitized pixel value, are used: instead of a hat function used by the *MergeDebevec* algorithm, the *MergeRobertson* algorithm uses a Gaussian-like function, generally yielding more accurate results.

We have tested both approaches for finding the inverse camera response function as well as for the merging process. The *MergeRobertson* algorithm generally performed better, therefore it was selected to be used in the camera application.

The merging algorithms are already implemented in libraries/tools such as OpenCV and pfstools [Pfs]. While these implementations (and the original articles) take only exposure times on input, disregarding any other exposure parameters (ISO or aperture), photos with different ISOs can be merged by compensating the exposure time by the ISO value[27]. Scripts, which run the merging process with the two algorithms (also comparing their implementations), are available in the "HDR photo/merge" repository along with a few test images.

Although the implementation in the *pfstools* package[28] offers the weighting function to be chosen from multiple options and supports several camera response curve fitting models, providing more tuning options, we chose the OpenCV implementation, because the library is already pre-compiled for iOS and is easier to integrate than a command line tool.

We have observed noise artifacts in areas, which are not well-exposed in any of the images, which are being merged. This noise amplification is caused by the extremely low weight of always-clipped pixels, giving higher priority to noise pixel values, which are "better-exposed" according to the weighting function, over the correct-clipped values. This could be solved by special handling of pixels with low weight sum value, assigning them the clipped value from the shortest/longest exposure, depending on whether the pixel was saturated or black in all images. Alternatively, all input images may be pre-processed by rounding the almost-clipped values to the corresponding clipped value. Verification of these fixes is left as a future work.

As we need to merge the reconstructed HDR image with other LDR images captured with different exposures, we needed to modify the merging algorithm. The modification consists of modifying 8-bit input assertions and special processing of HDR inputs: we generate multiple LDR images with simulated exposures covering the reconstructed dynamic range, which are merged without applying the found inverse camera response function. The code of the modified algorithm is in the MergeRobertsonUnlimited repository.

[26] Sum of (exposure time and camera response function)-compensated pixel values multiplied weighted by the weighting function, divided by the sum of weights.

[27] Simplest possibility: when the product of the exposure time and the ISO value is passed to the algorithm, the merged result does not show any artifacts.

[28] the pfshdrcalibrate tool

### 3.6.6 Saving

The merged image needs to be saved, therefore a format and location needs to be determined.

The taken photos can be stored either in the system-wide photo library, or manually in a folder owned by the application, such as application's Documents folder, which is accessible from the computer e.g. using iTunes. While the former approach would provide us with on-device viewing of the captured image (at least LDR), the latter approach was chosen as is does not limit the choice of image formats and because we would like to also save all captured LDR images before merging – saving many images in the photo library would "spam" the user with unwanted pictures.

We chose the Radiance HDR (RGBE) image format [Wikipedia contributors, 2018b] for its compatibility and relatively compact data storage.

## 3.7 Application code

The application code is based on AVCam code sample [App, a], on top of which we have implemented automatic base exposure parameters metering, generation of additional exposure parameters and GUI for setting the parameter generation options, image alignment using the OpenCV library and merging using a pre-calibrated inverse camera response curve for iPhone SE using the modified *MergeRobertsonUnlimited* algorithm. The resulting code is available in the CameraApp repository.

## 3.8 Application usage

A short description of how to use the HDR camera application is written in the Attachment A.3.

# 4. Results

In this chapter we present results of the HDR reconstruction methods evaluation and show the capabilities of the HDR camera application comparing it to the built-in camera application and another HDR camera application.

## 4.1 Comparison of existing machine learning-based dynamic range expansion methods

### 4.1.1 Compared approaches

Out of the five deep learning-based high dynamic range reconstruction approaches referenced in the *Related work* chapter only four approaches have a public implementation. As the neural network by [Zhang and Lalonde, 2017] only supports low resolution inputs and outputs, making the output images unsuitable for use as an environment map, it was excluded from the evaluation. The remaining methods, which provide pre-trained neural networks and support high resolution images, are: HDRCNN [Eilertsen et al., 2017], DrTMO [Endo et al., 2017] and ExpandNET [Marnerides et al., 2018]. As the pre-trained models are used, the reconstruction quality depends on the training dataset size and image selection. This issue was disregarded due to too high computational demands to train all the networks and may be addressed in a future work.

### 4.1.2 Dataset

The evaluation dataset consists of HDR spherical panoramas from the following sources: *NoEmotion HDRs – Dayhdr* [Noe] and *HDRLabs sIBL Archive* [Hdr].

This set of HDR images contains a combination of outdoor scenes with direct sunlight and sky with and without clouds as well as indoor images.

### 4.1.3 Results and interpretation

The high dynamic range reconstruction evaluation framework was used to process (import, degrade, reconstruct using each approach and compute error metrics for) each image in the dataset described above to evaluate the reconstruction methods.

#### Average metrics results

The aim of these results is to get an aggregated performance of individual dynamic range reconstruction methods and to spot any global issue with the networks. The results are shown in Table 1 with the best achieved value for each metric highlighted in bold text.

#### Initial observations

The *Average RMSE (unsaturated)* of *LDR (baseline)* is non-zero because of quantization of the HDR image: the size of an 8-bit value step is $1/256 = 0.00390625$,

because rounding to the nearest value is used, the upper bound for per-pixel error is half a step, which is $1/256/2 = 0.001953125$. The actual *Average RMSE (unsaturated)* is less, because not all pixel values are "exactly in the middle between steps" – the values are probably uniformly distributed.

Another observation is that all reconstruction methods introduce an error to well-exposed/unsaturated pixels. The pixel value distortion issue is best visible for the DrTMO reconstruction method – the solution is outlined below.

HDRCNN is the only method, which globally improves the baseline (both based on RMSE and SSIM). This is caused by the luminance (and color) distortion shown in Figure 4.1. This distortion is so large it negates any metric value improvements caused by the dynamic range reconstruction. Before we proceed with further comparison, we need to "linearize" the output of reconstruction methods.

### Linearization

Our goal is to "straighten" the luminance mapping curve so that it is as close to the "$y = x$" line as possible.

The linearization of HDRCNN results is relatively simple – as the luminance mapping curve distortion in the interval <0,1> is constant for each image, this points to a bug in code or incorrect usage of the dynamic range reconstruction algorithm.

The reason for the nonlinearity was an assumption in the HDRCNN code – the non-saturated parts of the input image were transformed by raising the pixel values to the power of 0.5 after running the inference, (probably) assuming the input LDR image to be saved with gamma value of 2. Therefore sRGB linearization followed by simulation of gamma 2.0 straightened the mapping curve in the unsaturated <0,1> range while keeping the reconstruction accuracy in clipped highlights. No post-reconstruction linearization needs to be applied, unlike other CNNs.

The linearization of the results of the remaining two CNNs (DrTMO and ExpandNet) is not as easy since the luminance mapping curve shape varies on per-image basis, requiring different "straightening" function parameters to be applied to each image, minimizing the global error (RMSE). We assume that only the luminance needs to be modified, therefore we try to achieve our linearization goal by uniformly transforming each color channel.

As the linearization needs to be done during the reconstruction, we "are not allowed to" use the ground truth image data. The choice of the function needs to

| **Method** | **Average RMSE** | **Avg. RMSE (saturated)** | **Avg. RMSE (unsaturated)** | **Average SSIM** |
|---|---|---|---|---|
| LDR (baseline) | 0.553395 | 2.273523 | 0.001476 | 0.986624 |
| HDRCNN | 0.496427 | 2.043810 | 0.023766 | 0.988511 |
| DrTMO | 0.618988 | 2.261333 | 0.162090 | 0.824917 |
| ExpandNET | 0.598677 | 2.411441 | 0.077155 | 0.930315 |

Table 4.1: Comparison of "raw" reconstruction methods. LDR (baseline) is the degraded image without any reconstruction.

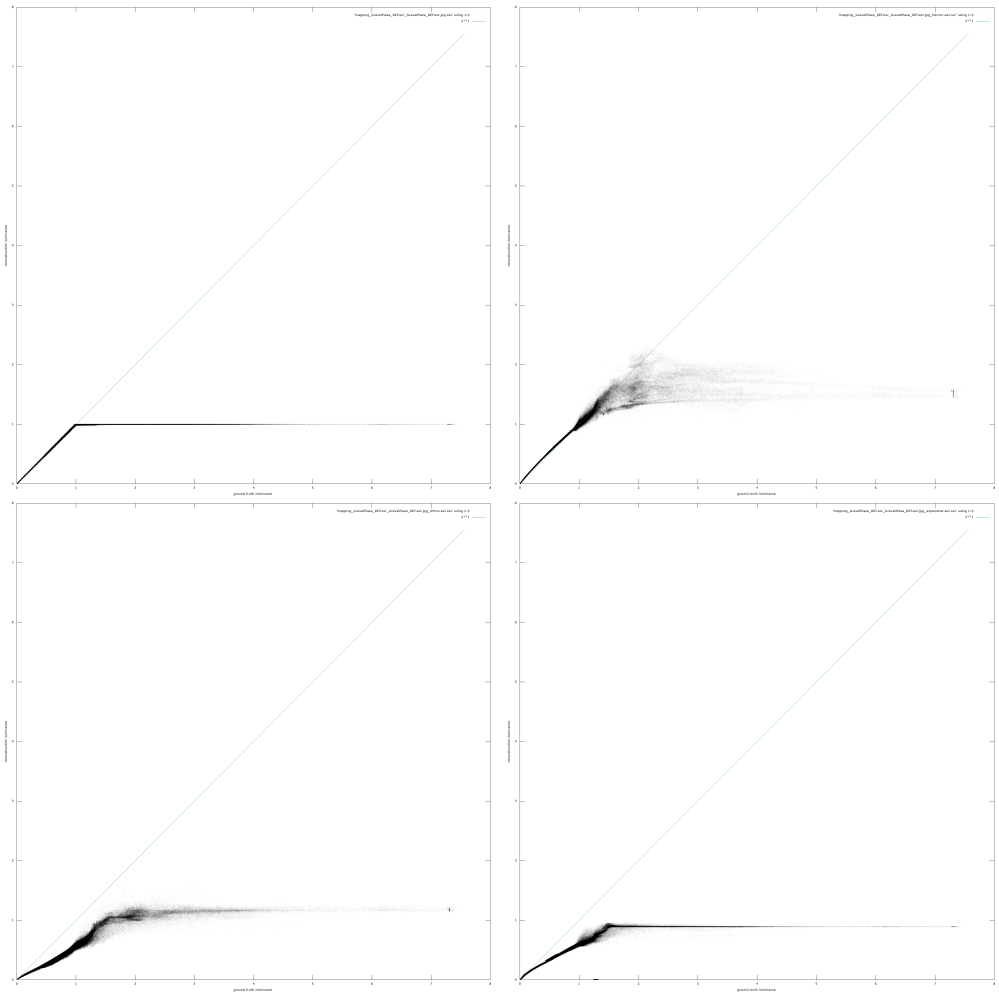


Figure 4.1: Luminance mapping graphs for the image GravelPlaza_REF, top-left: LDR (baseline), top-right: HDRCNN, bottom-left: DrTMO, bottom-right: ExpandNet.

be based on the knowledge of ground truth data, the function is then fixed with only its parameters being changed to the fitted values when running the reconstruction. Therefore, for each CNN we are looking for a function, which would straighten the luminance mapping curve, based on parameters found by fitting this function on the <0,1> domain, where the (quantized) ground truth values are known from the LDR image. The complete list of functions we attempted to fit is available alongside the fitting source code in the color_mapping repository – in the fix_mapping_auto.py file.

In case of the DrTMO CNN, the luminance mapping graph contained visible "stairs" or discontinuities for some images. Attempts to fit a function to the luminance curve failed as none of the test-fitted functions provided significantly lower error than other functions. Later investigation showed that the artifacts were caused by an inaccurate camera response function when merging the bracketed exposures. The effect of the camera response function calibration on the merging process is shown in Figure 4.2

When fitting the functions to the merged image produced with using the calibrated camera response function, the best results were achieved with the linear function ($f(x) = a*x+b$), which corresponds to shifting the exposure and adding an offset. As the fitted value of the offset ($b$) was always almost-zero[1], only multiplication ($f(x) = a * x$) was used for fitting.

The best performing transformation function for ExpandNet is also the multiplication (exposure shift).

Although *luminance mapping* curves for DrTMO nor ExpandNet were linear, the shape could not be reliably fitted using the test-fitted functions as the linearization using more complex functions usually "broke" other parts of the curve. Simple multiplication was generally the most reliable linearization operation.

The improved metrics values after linearization are shown in the Table 4.2

### Average ranking results

The HDRCNN network performed the best even when considering its ranking for both metrics, being the best reconstruction method for almost all images. Although DrTMO and ExpandNet rank on average between second and third place, beating the baseline, they yield significantly worse results than HDRCNN.

DrTMO and ExpandNet rank on average worse than LDR (baseline), based on the SSIM ranking, meaning running no reconstruction at all yields better result.

[1]The values were approximately +-0.01 – probably caused by noise.

| **Method** | **Average RMSE** | **Avg. RMSE (saturated)** | **Avg. RMSE (unsaturated)** | **Average SSIM** |
|---|---|---|---|---|
| LDR (baseline) | 0.553395 | 2.273523 | 0.001476 | 0.986624 |
| HDRCNN | 0.502348 | 2.074193 | 0.003058 | 0.993377 |
| DrTMO | 0.548770 | 2.270904 | 0.019501 | 0.980021 |
| ExpandNet | 0.543759 | 2.260362 | 0.025225 | 0.981952 |

Table 4.2: Average results for each metric and reconstruction approach after output linearization.

This is due to the pixel value distortion, although improved by our imperfect linearization of the <0,1> area, causing a loss of details.

### Observations

All networks seem to have learned that when a pixel has value of 1 in the LDR image, the reconstructed value needs to be at least 1.

The RMSE is sometimes high while the SSIM is near 1.0 for images, where the absolute luminance was recovered incorrectly, even when the incorrectly recovered area is small. The high value is caused by the high difference, which is preserved by the RMSE summing the intermediate results, while the SSIM is less sensitive to the error due to its windowed statistics computation approach and composition of the final similarity score from multiple equally-weighted components: luminance, contrast and structure.

Luminance mapping graphs of some reconstructed images contain a strange discontinuity, as depicted in Figure 4.3. Examples of such images are depicted in Figure 4.4. The corresponding color mapping graphs suggest the error occurs mostly in the blue color channel. The NarrowPath_3k image (as well as other affected images) contains the sky with blue channel values between 2 and 3. The large sky area gets clipped to 1.0 value during the degradation process, making it difficult for the CNNs to guess the original value. This causes the discontinuity in the luminance/color mapping graphs. As there may still be limited amount of information in the yet-unclipped color channels, all CNNs manage to estimate or at least "get closer to" the correct value for some of the clipped pixels. This can be seen on the luminance mapping graph as the vertical spread of points near the discontinuity, suggesting the CNNs manage to reconstruct "almost" the correct value.

We observed another strange artifact when examining ExpandNet luminance mapping graphs: *zeroing* of some values, both outside and inside the <0,1> range, as seen on Figure 4.5, indicating that some pixels have been turned black in the process of the dynamic range reconstruction. It was discovered that the first line of all reconstructed images is very dark, almost black[2]. The issue was not discovered earlier when manually looking at ExpandNet outputs because of the dark background of the image viewer.

This artifact appears to be caused by a bug in the ExpandNet code, possibly by use of incorrect padding option of convolution, causing "bleeding" of non-

[2]The values are usually between 0.001 and 0.003.

| **Method** | **Average RMSE** | **Average SSIM** |
|---|---|---|
| LDR (baseline) | 3.337662 | 2.753246 |
| HDRCNN | 1.233766 | 1.025974 |
| DrTMO | 2.649350 | 2.961038 |
| ExpandNet | 2.779220 | 3.259740 |

Table 4.3: Average ranking of each reconstruction approach for each metric, after reconstruction output linearization.

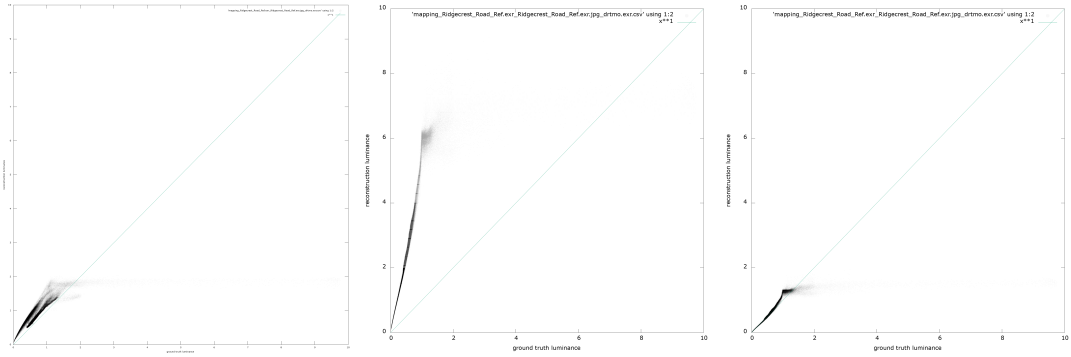



Figure 4.2: Luminance mapping graphs of the DrTMO output for the Ridgecrest_Road_Ref image. From left to right: without camera response function calibration, with CRF calibration and with CRF calibration and linearization.

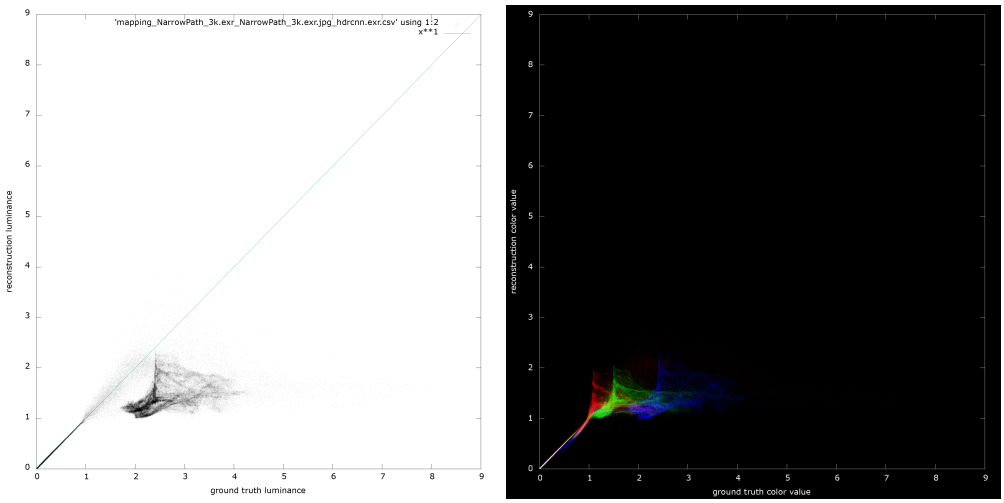



Figure 4.3: Luminance and color mapping graphs showing a discontinuity artifact.

Figure 4.4: LDR preview of the images with incorrectly reconstructed highlights. From left to right: NarrowPath_3k, Brooklyn_Bridge_Planks_2k and 10-Shiodome_Stairs_3k.

existent outside pixels into the image, or by a training data preparation issue, but the cause has not found and fixed yet.

There are several "outlier" images in the dataset, such as Chelsea_Stairs_3k and Circus_Backstage_3k, with very high pixel values in the sky or artificial light sources with a luminance value over 200, while the Sun on other images has much lower values. This is caused by the different base exposure: as the exposure value shift is equivalent to pixel value multiplication, high pixel values can be "legally" achieved by just choosing a low base exposure value and sufficient number of up-exposures when capturing the scene. As the dataset images are not photometrically calibrated, absolute pixel values are not indicative of the real-world light intensity.

Another irregularity can be observed for the Milkyway_small image, where HDRCNN performs significantly worse according to RMSE ranking than other CNNs. The image is synthetic: "Derived from NASA map, spiced up with nebulas." [Hdr]. While other CNNs are more conservative with the luminance value prediction, HDRCNN hallucinates high-value highlights (up to 50) at positions of some stars, which, while being correct, does not correspond to the ground truth image, causing the high measured error.

#### Outcomes

The best-performing dynamic range reconstruction approach is HDRCNN, leading in both RMSE and SSIM error metrics, outperforming all other reconstruction methods for the vast majority on input images. The remaining two metrics, while outperforming the baseline in the RMSE metric, reduce the SSIM metric results below the baseline level.

As the reconstructed images in the HDR camera application are not primarily meant to be seen by the user (they are meant to serve as an input of a merging algorithm), we prefered RMSE over a perception-based SSIM for its "absolute measurement".

HDRCNN provides the lowest distortion in the unsaturated areas, which is desirable for merging the image with other exposures, as well as the best recovery of clipped highlights. DrTMO and ExpandNet outputs perform measurably worse. DrTMO may be beneficial as it could expand the dynamic range not just to highlights, but also simulate increased exposure times recovering details in shadows, but based on the metric results and our intended use-case, we chose the HDRCNN reconstruction method.

## 4.2 HDR Camera Application

To compare our HDR camera application with other existing applications, we have captured a scene with high dynamic range – an interior with a window on a sunny day – using three applications: the built-in default Camera application, Lightroom CC and our HDR camera application. After capturing the scene, the HDR image produced by our HDR camera application needed to be exposure-shifted by -2 EV in order to match the brightness of other applications' "+0 EV" brightness level. The comparison of the captured images can be seen in Figure 4.6.

The default iPhone Camera application does not capture a true HDR image as the image is stored in an 8-bit format, which is visible from the lack of highlights in the "-5 EV" image. Because the "HDR" mode was enabled, the captured dynamic range was slightly improved before the tone-mapping algorithm was executed, which managed to preserve some details in both highlights and shadows – the window nor the interior is completely clipped. The shadows contain noise, as can be seen on the wall below the window when the exposure is shifted by "+3 EV".

In comparison, Lightroom CC produces a true HDR image. While Lightroom CC is limited by taking only 3 exposures, it manages to sufficiently cover the dynamic range of the scene, which is achieved thanks to the RAW image capture and processing. There is, however, visible noise in the shadows, observable in the "+3 EV" shifted exposure. This is caused by the lack of higher-exposure-value captures.

The image produced with Our HDR camera application captures the whole dynamic range of the scene (the lowest-exposure-value image contained no clipped highlights and the highest-exposure-value contained no clipped shadows.) There is, however, slight color inaccuracy when compared to the other two applications. Our HDR application also "suffers" from different exposure scaling, reducing the contrast. Both issues are probably caused by a minor inaccuracy in the camera response curve calibration.

Our HDR camera application is slower when compared to the default camera application. Capturing of a single image takes approximately 2 seconds due to merging being executed even when there is only one exposure. Capturing 4 exposures and merging them takes approximately 10 seconds. When multiple capture batches are required, i.e. when more than 4 different exposure values are ordered in case of the iPhone SE, capturing the 4-picture batches is done in approximately 2-second intervals. The merging process duration depends on the number of captured photos taking approximately 1.6 seconds per image. The interval between capture batches may be decreased by reducing the amount of processing, which is done – currently when a picture is taken, it is immediately compressed to PNG and stored for reference, while another uncompressed copy is passed to the merging algorithm. This optimizarion is, however, left as a future work.

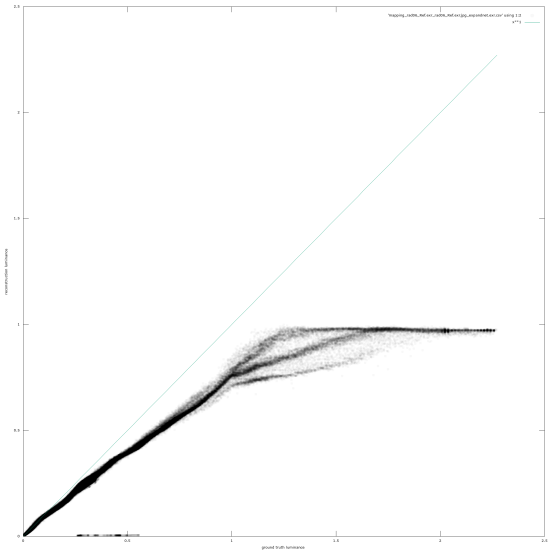
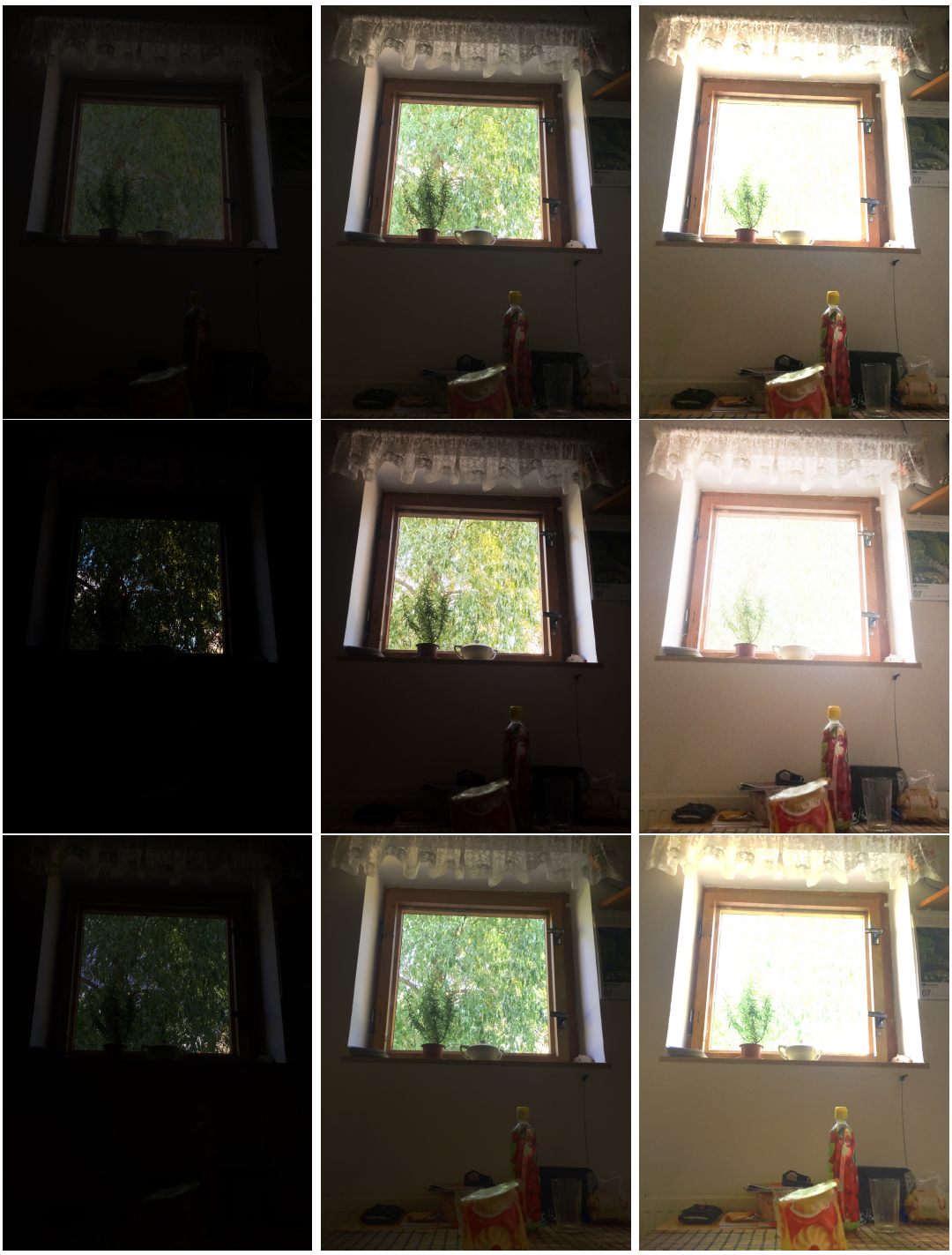


Figure 4.5: Luminance mapping showing ExpandNet artifact: many points with non-zero ground truth luminance, but almost-zero reconstructed luminance – points along the x axis. The non-linearized output version is shown to rule out possible issue in the linearization code.



The first row shows the image taken with the default Camera application using "HDR" mode, the second row shows the image produced by Lightroom CC and the third row is the image produced by our HDR camera application.
Each row contains three columns with shifted exposure values. From left to right: "-5 EV", "+0 EV" and "+3 EV".

Figure 4.6: Comparison of HDR camera applications for iOS.

# Conclusion and future work

In this thesis we have presented an evaluation framework for measuring performance of high dynamic range reconstruction methods, which take single LDR image as an input. We have used the framework for comparing the reconstruction quality of three machine-learning-based HDR reconstruction approaches, discovering difficult reconstruction cases as well as systematic reconstruction errors, which we eliminated by fixing bugs or linearizing the output of all three CNN implementations.

Although we were able to "port" a CNN-based HDR reconstruction method to iOS and run the image processing inference using a large convolutional neural network, we have come to the conclusion that it is currently impractical to do so on a mobile device, because of the performance limits.

We have also implemented an HDR camera application, which is able to capture the highest possible dynamic range, that can be achieved by the hardware.

As the topic of dynamic range reconstruction is so wide, we had to keep a few questions open or leave some tasks, which were out of scope of this thesis, for the future.

As a future work, training all the HDR reconstruction CNNs on the same dataset should be done for a fair comparison of the reconstruction approaches, adapting the reconstruction approaches for further dynamic range reconstruction of already HDR input, GPU-acceleration options for the HDRCNN execution optimization on iOS may be explored, the impact of the tile size on HDR reconstruction speed may be measured and ideas for fixing HDR-merging artifacts caused by the lack of well-exposed pixels may be verified, performance and image quality optimizations of the HDR camera application, such as using RAW image processing, may be implemented as well as features, which would simplify the workflow for capturing HDR spherical panoramas directly from the application.

# Bibliography


Lightroom Journal Lr Mobile Update: Raw HDR capture mode for iOS and Android. http://blogs.adobe.com/lightroomjournal/2017/03/lr-mobile-update-raw-hdr-capture-mode-for-ios-and-android.html#comment-216145. Accessed: 2018-07-20.

AVCam-iOS: Using AVFoundation to Capture Images and Movies. https://developer.apple.com/library/archive/samplecode/AVCam/Introduction/Intro.html, a. Accessed: 2018-07-20.

Apple Documentation Archive: Cameras. https://developer.apple.com/library/archive/documentation/DeviceInformation/Reference/iOSDeviceCompatibility/Cameras/Cameras.html#//apple_ref/doc/uid/TP40013599-CH107-SW26, b. Accessed: 2018-07-20.

Chainer: A flexible framework for neural networks. https://chainer.org. Accessed: 2018-07-20.

CocoaPods.org. https://cocoapods.org. Accessed: 2018-07-20.

Core ML — Apple Developer Documentation. https://developer.apple.com/documentation/coreml. Accessed: 2018-07-20.

dcraw Decoding raw digital photos in Linux. https://www.cybercom.net/~dcoffin/dcraw/. Accessed: 2018-07-20.

sIBL Archive: Free HDRI sets for smart Image-Based Lighting. http://hdrlabs.com/sibl/archive.html. Accessed: 2018-07-20.

JERI: Javascript Extended-Range Image viewer. https://jeri.io. Accessed: 2018-07-20.

LibRaw raw image decoder. https://www.libraw.org. Accessed: 2018-07-20.

Speeding Up TensorFlow with Metal Performance Shaders. http://www.mattrajca.com/2016/11/26/speeding-up-tensorflow-with-metal-performance-shaders.html. Accessed: 2018-07-20.

NoEmotion HDRs: HDRDay images dataset. http://noemotionhdrs.net/hdrday.html. Accessed: 2018-07-20.

OpenCV library. https://opencv.org. Accessed: 2018-07-20.

pbrt, Version 3: Source code for pbrt, the renderer described in the third edition of "Physically Based Rendering: From Theory To Implementation", by Matt Pharr, Wenzel Jakob, and Greg Humphreys. https://github.com/mmp/pbrt-v3. Accessed: 2018-07-20.

pfstools About. http://pfstools.sourceforge.net. Accessed: 2018-07-20.

PyTorch. https://pytorch.org. Accessed: 2018-07-20.

StackOverflow: What is a CVPixelBuffer in iOS? https://stackoverflow.com/a/41825032, a. Accessed: 2018-07-20.

TensorFlow Mobile Overview. https://stackoverflow.com/questions/5887248/ios-app-maximum-memory-budget/15200855#15200855, b. Accessed: 2018-07-20.

tablesorter: Flexible client-side table sorting. http://tablesorter.com/docs/. Accessed: 2018-07-20.

TensorFlow. https://www.tensorflow.org, a. Accessed: 2018-07-20.

GitHub tensorflow/tensorflow repository. https://github.com/tensorflow/tensorflow/tree/r1.8/tensorflow/contrib/makefile#ios, b. Accessed: 2018-07-20.

TensorFlow Mobile Overview. https://www.tensorflow.org/mobile/, c. Accessed: 2018-07-20.

tev – The EXR Viewer: High dynamic range (HDR) image comparison tool for graphics people with an emphasis on OpenEXR images. https://github.com/Tom94/tev. Accessed: 2018-07-20.

Ahmet Oğuz Akyüz, Roland Fleming, Bernhard E Riecke, Erik Reinhard, and Heinrich H Bülthoff. Do hdr displays support ldr content?: a psychophysical evaluation. *ACM Transactions on Graphics (TOG)*, 26(3):38, 2007.

Francesco Banterle, Patrick Ledda, Kurt Debattista, and Alan Chalmers. Inverse tone mapping. In *Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*, pages 349–356. ACM, 2006.

Paul E Debevec and Jitendra Malik. Recovering high dynamic range radiance maps from photographs. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 369–378. ACM Press/Addison-Wesley Publishing Co., 1997.

Frédéric Dufaux, Patrick Le Callet, Rafal Mantiuk, and Marta Mrak. *High dynamic range video: from acquisition, to display and applications*. Academic Press, 2016.

Gabriel Eilertsen, Joel Kronander, Gyorgy Denes, Rafał K Mantiuk, and Jonas Unger. Hdr image reconstruction from a single exposure using deep cnns. *ACM Transactions on Graphics (TOG)*, 36(6):178, 2017.

Yuki Endo, Yoshihiro Kanamori, and Jun Mitani. Deep reverse tone mapping. *ACM Trans. Graph*, 36(6), 2017.

Yongqing Huo, Fan Yang, Le Dong, and Vincent Brost. Physiological inverse tone mapping based on retina response. *The Visual Computer*, 30(5):507–517, 2014.

Garima Jain, Anand Plappally, and Shanmuganathan Raman. Internethdr: Enhancing an ldr image using visually similar internet images. In *Communications (NCC), 2014 Twentieth National Conference on*, pages 1–6. IEEE, 2014.

Hayden Landis. Production-ready global illumination. *Siggraph course notes*, 16 (2002):11, 2002.

Rafał Mantiuk and Wolfgang Heidrich. Visualizing high dynamic range images in a web browser. *Journal of Graphics, GPU, and Game Tools*, 14(1):43–53, 2009.

Rafat Mantiuk, Kil Joong Kim, Allan G Rempel, and Wolfgang Heidrich. Hdr-vdp-2: a calibrated visual metric for visibility and quality predictions in all luminance conditions. In *ACM Transactions on graphics (TOG)*, volume 30, page 40. ACM, 2011.

Demetris Marnerides, Thomas Bashford-Rogers, Jonathan Hatchett, and Kurt Debattista. Expandnet: A deep convolutional neural network for high dynamic range expansion from low dynamic range content. In *Computer Graphics Forum*, volume 37, pages 37–49. Wiley Online Library, 2018.

Belen Masia, Ana Serrano, and Diego Gutierrez. Dynamic range expansion based on image statistics. *Multimedia Tools and Applications*, 76(1):631–648, 2017.

Laurence Meylan, Scott Daly, and Sabine Süsstrunk. The reproduction of specular highlights on high dynamic range displays. In *Color and Imaging Conference*, volume 2006, pages 333–338. Society for Imaging Science and Technology, 2006.

Shiyu Ning, Hongteng Xu, Li Song, Rong Xie, and Wenjun Zhang. Learning an inverse tone mapping network with a generative adversarial regularizer. *arXiv preprint arXiv:1804.07677*, 2018.

Erik Reinhard, Michael Stark, Peter Shirley, and James Ferwerda. Photographic tone reproduction for digital images. *ACM transactions on graphics (TOG)*, 21(3):267–276, 2002.

Allan G Rempel, Matthew Trentacoste, Helge Seetzen, H David Young, Wolfgang Heidrich, Lorne Whitehead, and Greg Ward. Ldr2hdr: on-the-fly reverse tone mapping of legacy video and photographs. In *ACM transactions on graphics (TOG)*, volume 26, page 39. ACM, 2007.

Mark A Robertson, Sean Borman, and Robert L Stevenson. Dynamic range improvement through multiple exposures. In *Image Processing, 1999. ICIP 99. Proceedings. 1999 International Conference on*, volume 3, pages 159–163. IEEE, 1999.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

Mushfiqur Rouf, Cheryl Lau, and Wolfgang Heidrich. Gradient domain color restoration of clipped highlights. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on*, pages 7–14. IEEE, 2012.

Florian M Savoy, Vassilios Vonikakis, Stefan Winkler, and Sabine Süsstrunk. Recovering badly exposed objects from digital photos using internet images. In *Digital Photography X*, volume 9023, page 90230W. International Society for Optics and Photonics, 2014.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Lvdi Wang, Li-Yi Wei, Kun Zhou, Baining Guo, and Heung-Yeung Shum. High dynamic range image hallucination. In *Proceedings of the 18th Eurographics conference on Rendering Techniques*, pages 321–326. Eurographics Association, 2007.

Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

Greg Ward. Fast, robust image registration for compositing high dynamic range photographs from hand-held exposures. *Journal of graphics tools*, 8(2):17–30, 2003.

Wikipedia contributors. Exposure value — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Exposure_value&oldid=831545501`, 2018a. [Online; accessed 20-July-2018].

Wikipedia contributors. Rgbe image format — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=RGBE_image_format&oldid=828887752`, 2018b. [Online; accessed 20-July-2018].

Jinsong Zhang and Jean-François Lalonde. Learning high dynamic range from outdoor panoramas. *arXiv preprint arXiv:1703.10200*, 2017.

# List of Figures

# List of Tables

# A. Attachments

## A.1 *Electronic attachment* contents

- README.txt – document and links to the Git repositories with current version of the programs
- thesis.pdf – this document
- dp_root – root directory/repository containing the the source code of all necessary programs
  - pipeline – HDR reconstruction evaluation pipeline code
  - color_mapping – set of shell and Python scripts for comparing images and generating luminance/color mapping graphs
  - hdr_viewer – HTML/JavaScript HDR image viewer, modified pfstouth-drhtml output allowing switching of the displayed images
  - HDR photo/merge – set of scripts for merging bracketed LDR images with a few example images
  - MergeRobertsonUnlimited – modified HDR merging algorithm also allowing HDR image processing
  - CameraApp – source code of the iOS HDR camera application
  - ExpandNet/hdr-expandnet – source code of the ExpandNet CNN
  - DrTMO/DrTMO – source code of the DrTMO CNN
  - HDRCNN/cnn/hdrcnn – ource code of the HDRCNN CNN
  - pbrt-v3 – source code of the pbrt raytracer
  - dependencies/pfstools – Dockerfile for creating a container with pfstools

## A.2 HDR reconstruction evaluation framework usage

After installing the dependencies by following installation instructions in the pipeline repository, open a shell interpreter and change the working directory to a directory (preferably an empty folder), where the pipeline output should be generated.

Run the script all_batch.sh passing an input directory path as an argument. Each image with .hdr or .exr extension stored in the directory or any of its subdirectories will be processed – supported input formats are OpenEXR and Luminance HDR (RGBE). If processing only one image is desired, run the all.sh script passing only one input image path as an argument.

The evaluation speed on a laptop with Intel Core i7-4770HQ CPU and 16GB RAM is approximately 5 minutes per one input image.

If an error occurs during the pipeline execution, the process is stopped showing the most recently executed commands before the error in case of all.sh or in case of using all_batch.sh the error output can be found in a log file with file name corresponding to the just-processed image.

During the evaluation process the file evaluate.csv is generated, containing error metrics values for each reconstruction method. This table can be used for computer processing of the results such as computation of aggregated statistics.

When the evaluation finishes (and whenever processing of an image is finished), a user-interactive webpage with a sortable table with evaluation results is generated with the filename index.html.

## A.3 HDR Camera application usage

The camera application starts in automatic exposure mode with default relative dynamic range settings, allowing immediate image capture.

The photo(s) can be taken by pressing the Photo button at the bottom of the screen.

The Settings button at the bottom of the screen opens or closes the exposure and dynamic range settings overlay.

When Automatic mode is used, base exposure parameters are automatically determined based on the lighting conditions; in Manual mode the exposure parameters may be changed manually or determined automatically after pressing the Measure button in the settings.

Note that when too many exposures are requested, the device may run out of memory, causing the application to crash. On iPhone SE, the upper limit is approximately 6 or 7 exposures. The limit may differ based on the software running in the background.

When a capture is initiated, all captured exposures are stored in the application's Documents folder along with the HDR image merged from these exposures. The files are named IMG_[serial_number]_[exposure_time]_[ISO].png and IMG_[serial_number]_merged.hdr.

The Documents folder with all taken photos can be accessed from computer using iTunes.

For future reference:

```
dc53a4260d52f0ece5fcd88ff68d4fe96d24de3c67758acb63ebd90602ecce8e,
639086fb876ddb5ba1269a47afca89025cd70e97469093b88f28c7f
08ab0692c5cdbfc9c26148df9736bde90ee77ae109d8ae44200377
2db149e779370374c0e
```