Machine Learning Based Reconstruction of Non-Regularly Sampled Raw Images

Anton Stötzer

Bachelor Thesis

Supervised by
Prof. Dr. Johannes Maucher
Prof. Dr. Jan Fröhlich

September 18, 2023

HOCHSCHULE DER MEDIEN

Computer Science and Media
Hochschule der Medien Stuttgart

Abstract

Today's digital cameras use a mosaic of red, green, and blue color filters to capture images in three color channels on a single sensor plane. This thesis investigates the use of convolutional neural networks (CNNs) for demosaicing – the process of reconstructing full-color images from raw mosaic sensor data. While there are existing CNNs for demosaicing raw images from the well-established regular Bayer color filter array (CFA), this thesis focuses on how they perform on alternative non-regular sampling patterns that produce less aliasing artifacts, namely the stochastic Gaussian- and the RandomQuarter sampling pattern (Backes and Fröhlich, 2020).

A basic UNet (Ronneberger et al., 2015) and the spatially adaptive SANet (T. Zhang et al., 2022) are implemented in a supervised training pipeline based on the PixelShift200 image dataset (Qian et al., 2021) to investigate their suitability for the irregular demosaicing task. The experiments indicate that the basic UNet encounters difficulties in restoring the missing color values, whereas the spatially adaptive convolutional layers help in processing the irregularly sampled raw images.

In addition, this thesis enhances SANet effectiveness by employing an alternative residual branch based on a CFA-normalized Gaussian filter, as well as a tileable modification to the Gaussian CFA pattern. The modified SANet is shown to outperform the conventional dFSR algorithm (Backes & Fröhlich, 2020) in terms of peak signal to noise ratio (PSNR) and structural similarity index measure (SSIM).

Kurzfassung

Moderne Digitalkameras nutzen ein Pixel-Mosaik aus roten, grünen und blauen Farbfiltern auf dem Kamerasensor (auch *Color-Filter-Array*, *CFA* genannt), um Bilddaten in drei Farbkanälen auf einer einzigen Sensorschicht zu erfassen. In dieser Bachelorarbeit wird der Einsatz von Convolutional-Neural-Networks (CNNs) für das Demosaicing untersucht, also für die Rekonstruktion von Vollfarbbildern aus Rohdaten von Kamerasensoren mit Farbfiltermosaik. Während bereits CNNs für das Demosaicing von regelmäßig gesampleten Rohbildern des Bayer-Sensors existieren, konzentriert sich diese Arbeit darauf, welche CNNs sich für alternative, unregelmäßige Abtastmuster eignen, die weniger Aliasing-Artefakte erzeugen. Konkret werden das stochastische Gauss- und das RandomQuarter-Abtastmuster (Backes & Fröhlich, 2020) untersucht.
Ein einfaches UNet (Ronneberger et al., 2015) und das räumlich adaptive SANet (T. Zhang et al., 2022) werden in einer überwachten Trainingspipeline basierend auf dem PixelShift200-Bilddatensatz (Qian et al., 2021) implementiert, um ihre Eignung für die Aufgabe des irregulären Demosaicing zu untersuchen. Die Experimente zeigen, dass das einfache UNet auf Schwierigkeiten bei der Wiederherstellung der fehlenden Farbwerte stößt, während die räumlich adaptiven (*spatially-adaptive*) Convolution-Layer die Rekonstruktion der unregelmäßig abgetasteten Rohbilder verbessern.
Zusätzlich wird in dieser Arbeit die Leistungsfähigkeit des SANets durch einen alternativen Residual-Zweig auf Basis eines CFA-normalisierten Gaussfilters sowie durch eine kachelbare Modifikation des Gauss-CFAs erhöht. Das modifizierte SANet übertrifft den herkömmlichen dFSR-Algorithmus (Backes & Fröhlich, 2020) in Bezug auf Peak-Signal-to-Noise-Ratio (PSNR) und Structural-Similarity-Index-Measure (SSIM).

Ehrenwörtliche Erklärung

Hiermit versichere ich, Anton Stötzer, ehrenwörtlich, dass ich die vorliegende Bachelorarbeit mit dem Titel „Machine Learning Based Reconstruction of Non-Regularly Sampled Raw Images“ selbstständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden. Ich habe die Bedeutung der ehrenwörtlichen Versicherung und die prüfungsrechtlichen Folgen (§ 24 Abs. 2 Bachelor-SPO (7 Semester) der HdM) einer unrichtigen oder unvollständigen ehrenwörtlichen Versicherung zur Kenntnis genommen.

Anton Stötzer,
Stuttgart, 18. September 2023

Contents

Acronyms

ACUDE	adaptive chrominance-based universal demosaicing
CFA	color filter array
CNN	convolutional neural network
dFSR	demosaicing with frequency selective reconstruction
FLOPs	floating point operations
FSR	frequency selective reconstruction
GPU	graphical processing unit
ISP	image signal processing
MLP	multi layer perceptron
MS-SSIM	multiscale structural similarity index measure
MSE	mean squared error
OLPF	optical low-pass filter
PSNR	peak signal to noise ratio
RGB	red, green, and blue
SLP	single layer perceptron
SSIM	structural similarity index measure

1. Introduction

1.1. Motivation

In the past, a lot of research has gone into the development of debayering algorithms to overcome negative side effects like aliasing artifacts that arise from the highly repetitive Bayer pattern. In contrast, literature says that irregular sampling patterns produce less aliasing artifacts (see Dippé and Wold, 1985; Grosche et al., 2018a; Hennenfent and Herrmann, 2007). Backes and Fröhlich (2020) propose the use of the demosaicing with frequency selective reconstruction (dFSR) algorithm for universal demosaicing of non-regular, pseudo-random color filter arrays.
However, during the last few years, the evolving machine learning technology has found its way into the classical debayering process (see Buades et al., 2022; T. Zhang et al., 2022; Zhou et al., 2018). In this context it seems worth investigating, whether convolutional neural networks can also be used to demosaic pseudo-stochastic patterns and how they compare to dFSR.

1.2. Hypotheses

The human eye has day vision receptors for red, green, and blue that are randomly distributed across the retina. Neural networks could be assumed to be better than traditional algorithms in dealing with irregularly sampled image data as they share more similarities with the human brain. The CNN approach therefore is expected to produce better results compared to the dFSRalgorithm (Subsection 2.2.3). Furthermore it is assumed, that the CNN should be able to augment missing information with experience learned during the training phase, which makes it superior to dFSR algorithm.
The dFSR algorithm reconstructs the green channel first, in order to enhance the red and blue channels. The CNN is expected to leverage inter-channel correlations so that the green channel also benefits from the red and blue samples. Also, the reconstruction of the green channel through the CNN is expected to be superior when the entire RGB sensor input is fed into the network, compared to solely the green sensor data.
The way CNNs are accelerated by parallel GPU computations should make the demosaicing process much faster as with the conventional dFSR algorithm, which processes the image

in many iterations.

1.3. Research Design

A supervised training pipeline based on the PixelShift200 image dataset (Qian et al., 2021) is implemented to allow training and evaluation of different CNN architectures. The main focus of this thesis is on modifying the SANet architecture (T. Zhang et al., 2022), originally designed for classical Bayer patterns, to operate on (pseudo-)random color filter patterns as studied by Backes and Fröhlich (2020). The implementation will use Python and PyTorch (Paszke et al., 2019).
To prepare the image dataset and simulate the color filter array, a similar workflow to Backes and Fröhlich (2020) is utilized in this thesis. To compare the demosaicing quality of the machine learning approach presented in this thesis with their dFSR algorithm, the same validation images are processed using the dFSR implemention from their repository in MATLAB. Objective similarity and quality metrics including PSNR and SSIM are chosen in accordance with Backes (2019) for the evaluation of the results.

1.4. Scope and Limitations

The experiments will be restricted to the two most promising irregular CFA patterns, RandomQuarter and Gauss, investigated by Backes and Fröhlich (2020). The goal is not universal demosaicing in the sense that a single CNN model can handle different CFAs. Instead, a separate model is trained for each pattern.
As the training process and CNN architectures in this thesis are customized for irregular CFAs, a general comparison of the results with classical Bayer patterns and their different demosaicing algorithms is beyond the scope of this study.
Furthermore, this work focuses on processing individual images and does not deal with the additional challenges of continuous video sequences. To focus on the irregularity of the CFA, the demosaicing of noisy images is not examined, although this would be a logical next step. The experiments do not examine the effects of an optical low-pass filter (OLPF) either, as modern high-resolution cameras do not incorporate them.

1.5. Overview of the Thesis

Chapter 2 and Chapter 3 provide the necessary insight into the two disciplines Raw Image Processing and Convolutional Neural Networks that are combined in this thesis. The explanations focus on the aspects that are particularly relevant to the research questions of this thesis.

Chapter 4 describes the experimental setup in detail. It explains, how the training image set was build and which CNN architectures were used. It gives an overview on the comparison metrics used for evaluation of the results. Further this section documents the modifications made to the network architecture for optimization.
Chapter 5 then evaluates the results of the experiments by comparing the different CFA patterns, CNN architectures, and optimizations applied. Benchmarks with the dFSR algorithm will be included.
Lastly, Chapter 6 concludes the thesis and suggests further research.

2. RAW Image Signal Processing (ISP)

This chapter provides the necessary insight into Raw Image Processing, starting from the camera's RAW files, the color filter arrays (CFAs) explain the need of demosaicing. The regular Bayer pattern, as well as alternative non-regular sampling patterns will be presented. Finally the section about the sRGB color space will lead to a deeper understanding of the data that will later be used for training the convolutional neural network (CNN) models.

2.1. From Linear Camera RAW Files to Viewable sRGB Images

Inside a digital camera, a series of image signal processing (ISP) steps take place before a photo is typically stored as an 8-bit JPEG file on the camera's memory card. Depending on manufacturer and model, many cameras provide the option to output images as RAW files, containing more or less unprocessed sensor data as well as metadata on the image settings and exposure measurements. These files usually have a larger file size as they contain image data in a higher bit depth of typically 10, 12 or 14 bits to cover the original dynamic range of the camera sensor. (Sumner, 2014)
Photographers can use specialized RAW Utility software to process RAW files into final photos, allowing for adjustment of exposure and white balance even after the image has been captured. This can be particularly useful for recovering highlight structures that may have been lost in the camera's own JPEG file. Since this procedure is reminiscent of developing photos in a darkroom using analogue film, RAW files are occasionally referred to as digital photo negatives. (Sumner, 2014)
For this thesis, RAW files provide a way to research the internal signal processing of digital cameras without having to unscrew a camera or require exclusive manufacturer data. RAW files can also be decoded manually using a program library such as dcraw, which provides a common interface for the RAW formats of various camera models and allows to build a custom RAW image processing pipeline e.g. in a programming environment such as MATLAB or Python.

2.1.1. Raw Image Processing Pipeline

To receive a viewable output image, several steps are typically applied to the raw sensor data, that form the raw image processing pipeline, as shown in Figure 1. Some cameras

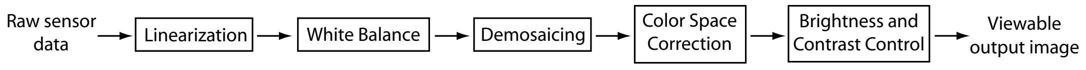

Figure 1: Raw Image Processing Pipeline. Adapted from Sumner, 2014, Fig. 3

use a custom color encoding in their RAW files, defined by a lookup table. As a first step, these values need to be transformed back to the linear color domain.
The colors of a scene are influenced by the light that illuminates it. The human eye naturally adjusts to different colors of light sources. However, when looking at an image on a screen, the difference in lighting to the surrounding room becomes obvious. To compensate for this difference, the images are adjusted to an ideal environment. This operation is called *white-balance* and is merely the process of scaling the red and the blue channel of the image to accomplish the optimal relation between red, green and blue colors. Often, the right values can be extracted from the RAW image file's metadata, where the camera stores its illumination measurements.

The next step in the pipeline is *demosaicing*, which is the process of reconstructing a fully sampled RGB image from the mosaic sensor image, which will be explained in Section 2.2. Finally, the image needs to be transformed to the color space, such as sRGB, which corresponds to the output screen. Further brightness and contrast adjustments might be needed to produce the right image impression. (see Sumner, 2014)
In order to focus on the challenges that arise with (pseudo-)stochastical CFAs this bachelor thesis does not cover the topic of noise reduction and all experiments are conducted with images that are assumed to be noise-free. However, it should be mentioned at this point that noise reduction is an important step to be considered in the raw image processing pipeline when it comes to images captured in non-ideal conditions. Noise can also have a significant impact on the result of the demosaicing process. Depending on the algorithm, noise can be amplified by demosaicing and lead to color artifacts (see Hasinoff, 2014). For this reason noise reduction is best handled before or together with demosaicing (see Gharbi et al., 2016; W. Xing and Egiazarian, 2021).

2.1.2. sRGB – a Gamma 2.2 Transformed Color Space

The sRGB color space was initially proposed by Hewlett-Packard and Microsoft as the standard color space for the Internet (Stokes et al., 1996). After being widely adopted, it was specified by the IEC 61966-2-1 standard (Commission et al., 2003). It has a built-in gamma 2.2 transform, which matches the typical intensity-to-voltage response of cathode ray tube (CRT) displays of that era (Gonzalez & Woods, 2008, p. 111). CRT screens have since been replaced by newer screen technologies such as LCD, DLP, LCoS, or

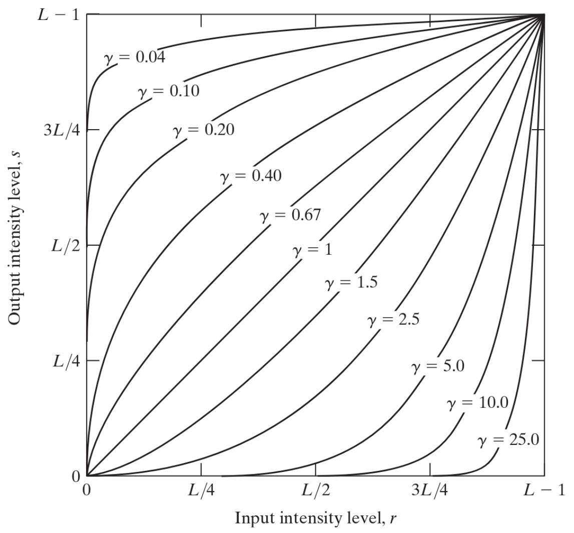

Figure 2: Power-Law Transform Curves for different Gamma values and $c = 1$. Adapted from Gonzalez and Woods, 2008, p. 111.

OLED/AMOLED, which no longer have a physical intensity-to-voltage response like the old CRT (Poynton, 2018, p. 26). These newer screens require digital compensation for the transformation before displaying an sRGB signal.
Despite this, the Gamma 2.2 domain continues to play a significant role due to its close correlation with human brightness perception. Since this aspect of the sRGB color space is relevant to the experiments presented in this thesis, it will be discussed in detail below, after an explanation of how power-law (gamma) transformations and sRGB work in general.
A power-law (Gamma) transform can be basically described as

$$s = c \cdot r^{\gamma}, \tag{2.1}$$

with r being the input and s being the output intensity level, c and γ being positive constants (Gonzalez and Woods, 2008, p. 111).
The transform can be compensated with the inverse gamma transform

$$s' = \frac{r'}{c}^{\frac{1}{\gamma}}, \tag{2.2}$$

with r' and s' being the in- and output intensity levels. To prepare an image for a display device that has a specific gamma characteristic, the inverse transform must first be applied to the image (Gamma correction) to obtain the desired appearance. Figure 2 visualizes the transform function for various γ values and $c = 1$.

The sRGB transform function and it's inverse are defined by a slightly more complex composition of the basic power-law transform. It contains a small linear part to reduce noise and to avoid infinite slopes at position 0. Due to the additional offset and scaling, the exponent $\gamma = 2.4$ is equivalent to an effective $\gamma \approx 2.2$ (see Burger and Burge, 2022, p. 436).
According to the IEC 61966-2-1 standard (Commission et al., 2003), the conversion of an sRGB intensity value x to the linear color domain is defined by

$$f_{sRGB \to lin}(x) = \begin{cases} 12.92 \cdot x, & \text{for } x \leq 0.0031308, \\ 1.055 \cdot x^{1/2.4} - 0.055, & \text{for } x > 0.0031308. \end{cases} \tag{2.3}$$

The conversion linear color intensity value x' to sRGB color space is defined as

$$f_{lin \to sRGB}(x') = \begin{cases} \frac{x'}{12.92}, & \text{for } x' \leq 0.04045, \\ \left(\frac{x'+0.055}{1.055}\right)^{2.4}, & \text{for } x' > 0.04045. \end{cases} \tag{2.4}$$

To convert an RGB image from linear color space to sRGB and back, the intensity values from each color channel are transformed separately using the above functions.

2.1.3. Non-Linear Human Brightness Perception and Perceptual Uniformity

In terms of human brightness perception, the sRGB color space has an advantage compared to linear RGB encoding: The sRGB color space is very close to the inverse of human sensitivity to brightness, allowing an efficient use of the available bit depth. This is especially relevant for 24-bit RGB encoding, where only $2^8 = 255$ of discrete intensity values are available per channel. This *perceptually uniform* Gamma encoding results in more brightness sampling in the dark areas, where the human eye can also distinguish more levels than in the brighter intensity areas (Poynton, 2018, p. 25).

2.1.4. RGB Sensor Sensitivity

The photoelectric elements on the sensor plane convert incoming light photons to electric current, which is digitally measured. In order to capture colored images, they are captured in three light components, red, green, and blue. Color filters are applied on top of the sensor pixels, so that parts of the spectrum of visible light get filtered. This makes the pixel mainly sensitive for either red, green, or blue light components. However, similar to human vision, they measure a range of light frequencies rather than only one specific red, green, or blue frequencies. In fact the color sensitivity ranges overlap, as it is shown in Figure 3. For the demosaicing task this is an important fact, since it means that there exists RGB cross-channel correlations.

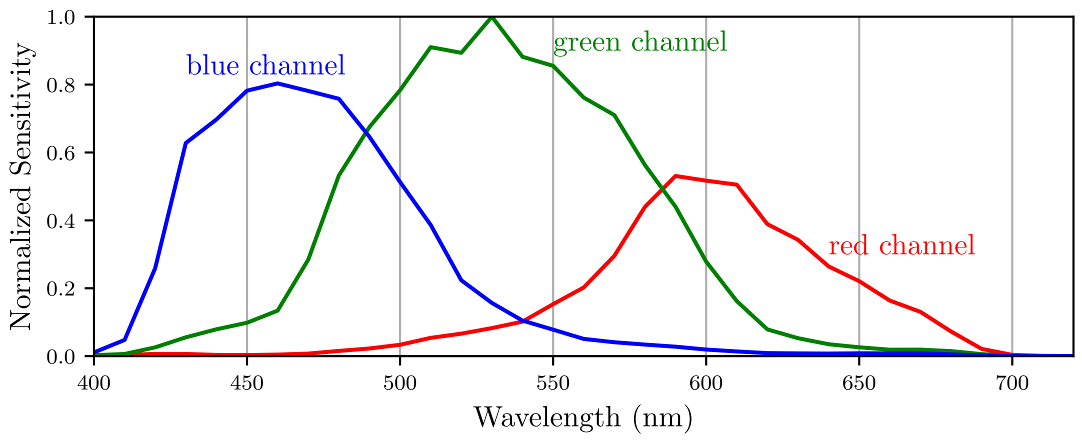

Figure 3: Spectral response of a typical digital CMOS RGB camera sensor. (Own visualization based on the measurements of a Canon 600d from the Camera Spectral Sensitivity Database, Jiang et al., 2013, Creative Commons BY-NC-SA 4.0)

2.2. Color Filter Arrays and Demosaicing

Typical camera sensors can only capture a single light intensity value per pixel[1]. In order to be able to capture the three channels of colored images on a single sensor plane, a spatially alternating color filter array (CFA) is used, as shown in Figure 4: Each sensor pixel has a single color filter applied on top making it sensitive for either red-, green- or blue light components. The sensor's output is not yet a full RGB image but a single channel image containing a mosaic of single color component values.

Most today's (photo- or video-)cameras contain a so-called Bayer sensor, patented in 1976 by its inventor Bryce Edward Bayer and Eastman Kodak Company (Bayer, 1976). The Bayer CFA is characterized by its repetitive two-by-two, red-green-green-blue pixel filter pattern. The captured pixel values can be decomposed into a three-channeled red, green,

[1]An exception to this is a stacked sensor design currently in development where multiple color frequencies can be measured in a single pixel (see Sigma Corporation, 2022).

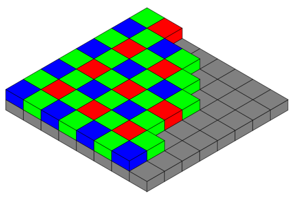

Figure 4: Photo Sensor with Bayer CFA Layout: Spatially alternating color filters allow to capture three color channels on a single sensor plane. Adapted from Sumner, 2014, Fig. 2

and blue (RGB) image with each channel lacking a proportion of pixels in between.
The process of reconstructing these gaps by interpolating the neighboring pixels is referred to as demosaicing or debayering when dealing with the Bayer pattern in particular. Demosaicing is an under-determined mathematical problem, which means that the missing information cannot be fully recovered when reconstructing a fully sampled RGB image from a mosaic sensor (see Gunturk et al., 2005).
The regularity of the Bayer sampling grid can interfere with thin or repetitive image structures that occur at similar spatial frequencies. These undersampling errors cause zipper artifacts or color moiré during demosaicing, which are visually distracting and difficult to remove afterwards. Improving existing demosaicing approaches is an ongoing research topic. The objective is to extract maximum information from the mosaic image to enhance the perceived image quality while avoiding amplification of noise or introduction of moiré or zipper artifacts.

2.2.1. Regular Debayering Algorithms

According to Gunturk et al. (2005), demosaicing approaches can be divided into three groups. The first group can be summarized as "heuristic approaches" as they do not provide an exact solution to a mathematically formulated problem but rather computational efficient approximations based on different assumptions about digital camera sensors and color images.
Li et al. (2008)
The simplest heuristic debayering algorithms reconstruct each channel individually without taking inter-channel-correlations into account (see Li et al., 2008). They define reconstruction rules to interpolate the missing values inside a color channel depending on the arrangement of the surrounding available samples. An example of this is linear debayering (see Figure 5) which fills the gaps by linear averaging four neighboring samples (for green channel) and two or four neighboring samples (for red and blue) depending on the location in the Bayer pattern. The border pixels of the image are either cropped or handled with additional rules based on one or two neighboring samples. (see Gunturk et al., 2005; Li et al., 2008)
More advanced heuristic debayering approaches rely on assumptions about image characteristics and interchannel correlations to regularize the demosaicing problem. For example, the assumption of spatial smoothness and constant hue between adjacent samples allows the application of various mathematical solutions. (see Gunturk et al., 2005) Most of them rely heavily on the green channel due to it's higher sampling density to reconstruct a high quality luminance channel. The remaining color channels are then interpolated e.g. in a green difference domain. (see Li et al., 2008) This technique is also applied by Backes and

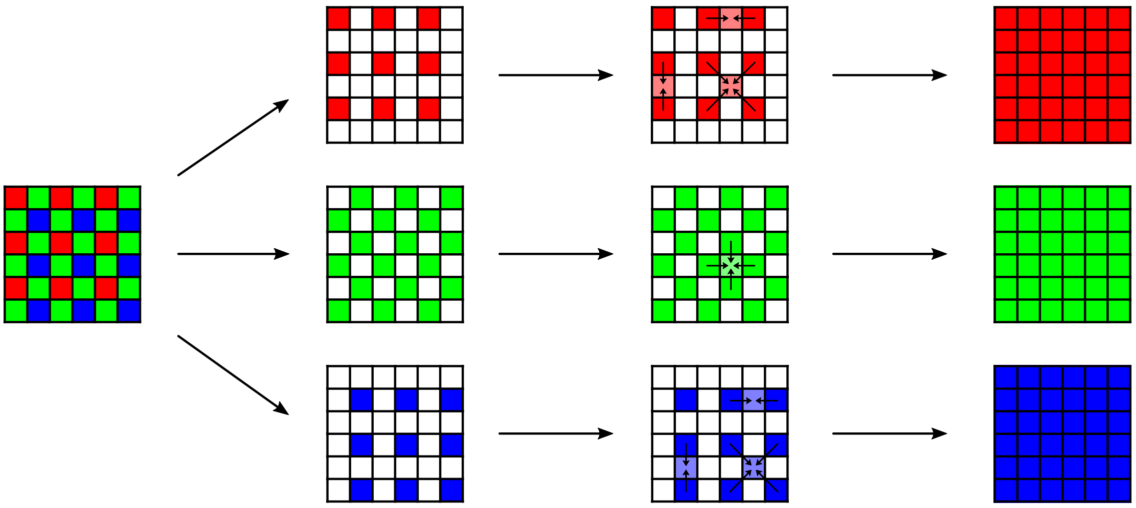

Figure 5: Linear debayering: The smaller arrows demonstrate how the algorithm approximates missing color values in various sample constellations by averaging two or four neighboring samples of the same channel.

Fröhlich (2020) for their dFSR algorithm described in Subsection 2.2.3.
In the recent years, machine learning models for debayering have shown to outperform existing approaches (cf. Kumar et al., 2023; Liu et al., 2020; Qian et al., 2021; W. Xing and Egiazarian, 2021). CNNs for debayering are more data driven: they rely on learned data characteristics from a training image set, rather than prior assumptions (see Li et al., 2008). Chapter 3 will elaborate on CNNs for demosaicing.

2.2.2. Irregular CFA Patterns

In the past, extensive research has been conducted on the development of debayering algorithms to overcome negative side effects such as aliasing artifacts that result from the highly repetitive Bayer pattern. However, some literature suggests resorting to alternative irregular CFA patterns for less aliasing errors (see Dippé and Wold, 1985; Grosche et al., 2018a; Hennenfent and Herrmann, 2007). Non-regular color filter arrays (CFAs) however are incompatible with conventional debayering and need universal demosaicing approaches, that will be part of Subsection 2.2.3.
When designing pseudo-random sampling patterns, it is important to strike a balance between randomness and uniformity. The aim is to distribute color samples randomly to prevent aliasing effects; however, each channel's sampling density should be uniform without significant gaps to achieve a location-invariant reconstruction. (see Backes and Fröhlich, 2020; Grosche et al., 2018a)
Figure 6 displays several non-regular sampling patterns that were examined by Backes and Fröhlich (2020), as well as the regular Bayer pattern. This section provides a brief

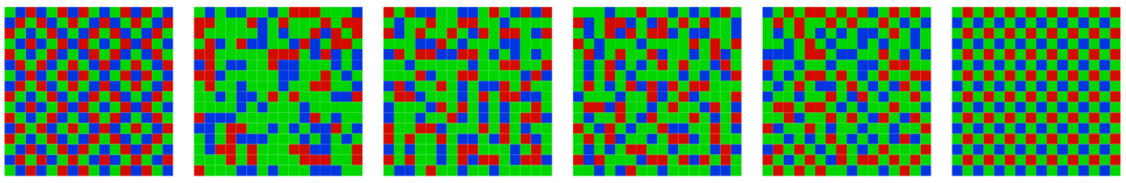

Figure 6: RGB Sampling Patterns. From left to right: Condat, Random, RandomQuarter, RandomICIPrgb, Gauss, and the Bayer pattern for comparison. Adapted from Backes and Fröhlich, 2020

summary of the different (pseudo-)stochastic approaches, before discussing two of them (RandomQuarter and Gauss patterns) as part of the experimental setup in Subsection 4.1.3.
The *Random* sampling pattern is a uniform discrete distribution of the RGB samples with no additional constraints.
The *RandandomQuarter* pattern, based on Schöberl et al. (2011), consists of a random permutation of the four colors red, green, green, and blue within each 2×2 pixel block. It will be further discussed in Subsection 4.1.3.
RandomICIP is a RandomQuarter variation by Grosche et al. (2018b) that reduces larger voids in the color channels and regularity.
The *Gauss* pattern distributes the color samples based on a Gaussian probability to maximize the distance between same colored samples, as described in Grosche et al. (2018a). It will be further discussed in Subsection 4.1.3 and Subsection 4.4.4.
Finally, the *Condat* pattern (Condat, 2010) is systematically built from 3×1 pixel blocks to avoid same-colored neighbors. The Condat pattern consists of an equal number of samples from each color, whereas all other patterns follow the Bayer color ratio of 25% red, 50% green, and 25% blue samples. (see Backes and Fröhlich, 2020)

2.2.3. Universal Demosaicing with dFSR

Backes and Fröhlich (2020) propose demosaicing with frequency selective reconstruction (dFSR), an algorithm for universal demosaicing that can also handle non-regular CFA patterns. Based on a signal processing technique called frequency-selective reconstruction, it attempts to infer a spatial frequency distribution from the available samples in each channel of the image. Once the image is represented in it's spatial frequencies, it can be transformed back into a fully sampled RGB image.
The dFSR algorithm processes the image in small overlapping blocks to produce a seamless uniform output. For each block a model of weighted Fourier basis functions is built iteratively to approximate the available color samples. This makes image reconstruction a rather computationally heavy task compared to simpler debayering algorithms. The reconstruction of a 1920×1080 pixel (FullHD) frame takes the authors about 40 minutes

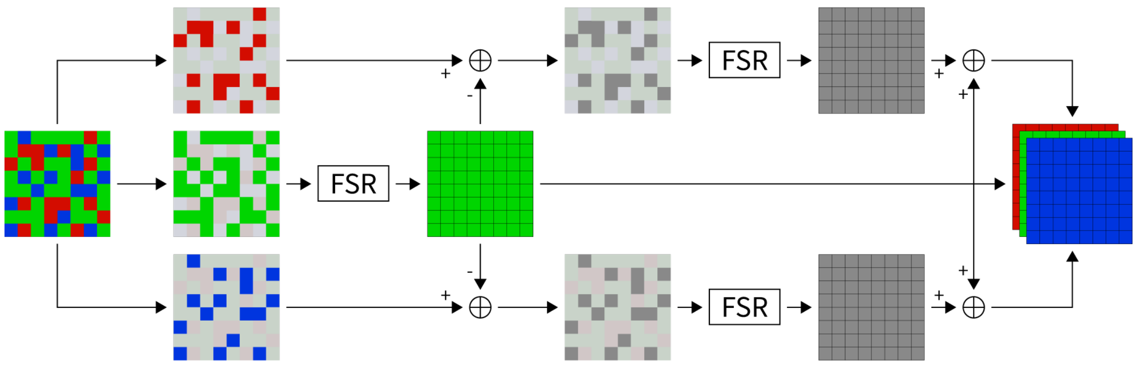

Figure 7: dFSR with green color difference domain. Adapted from Backes, 2019, Fig. 3.11

on a standard computer, making it unsuitable for in-camera or realtime purposes. However, the dFSR algorithm demonstrates, that irregular CFAs effectively reduce aliasing while slightly increasing noise and image smoothness. (see Backes, 2019)

The dFSR reconstructs each image channel red, green, and blue individually. However, Backes and Fröhlich (2020) achieve an improved demosaicing quality by reconstructing the green channel first, which has the highest sample density and use it as a luminance reference. In a second step, the red and blue channel are reconstructed in a green color difference domain, as depicted in Figure 7. This way, the red and blue channel benefits from high quality luminance details from the green channel. This technique, based on assumptions about the color-channel correlations, is also used for regular debayering algorithms, as described in Subsection 2.2.1.

Another alternative universal demosaicing algorithm is adaptive chrominance-based universal demosaicing (ACUDE) (see C. Zhang et al., 2016). According to Backes (2019) it does not perform as good as dFSR on irregular sampling patterns. Therefore, only the dFSR algorithm is used for comparison in this thesis.

3. Convolutional Neural Networks (CNNs) for Demosaicing

Convolutional Neural Networks (CNNs) can be categorized as a supervised machine learning approach which is a form of artificial intelligence, as depicted in Figure 8. Currently, they find application in diverse fields, including demosaicing of camera data.
This chapter first describes what supervised training is, and why it is particularly suitable for the demosaicing task. Following this, the chapter describes the discrete convolution operation, which enables efficient processing of image data and is the most important basic building block of any CNN. Building upon that, the concept of feature extraction inside basic CNNs is described. Further, the UNet architecture is described as an example of a deep neural network Figure 8 for image processing tasks. Regarding the main research topic of this thesis, demosaicing irregularly sampled images, the limitations of classical convolution are highlighted. Spatially adaptive convolution and the SANet architecture (T. Zhang et al., 2022), is offered as a potential solution, which will be tested later in the experiments.

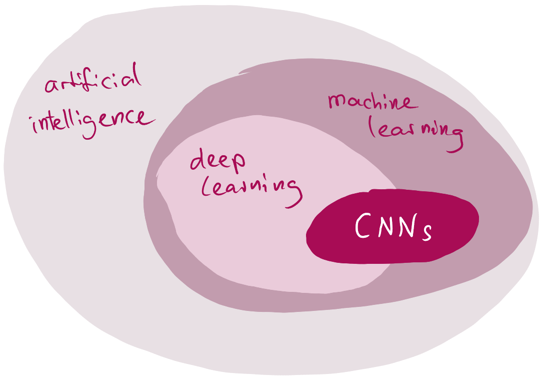

Figure 8: CNNs in the context of artificial intelligence

3.1. Convolution – the Base of Image Feature Extraction

Before introducing spatially adaptive convolution in Subsection 3.3.3 as a specialized form of convolution, it is important to develop a deeper understanding of how classical convolution works and how it is implemented for efficient graphical processing unit (GPU) processing.

3.1.1. 2D Discrete Convolution for Image Filtering

Convolution is a mathematical concept that existed long before the neural networks were invented and is used in image processing and general signal processing. Depending on the values of the filter kernel a convolutional operation can apply very different kind of filtering operations to the input. Figure 9 shows various filter kernels. E.g. a Gaussian kernel produces a blurred version of the input image, a Sobel kernel detects edges, a diamond kernel acts as a sharpening filter. The last Sobel example shows how the horizontal and vertical Sobel filter gradient results G_x and G_y can be combined to detect edges in all directions. This is already an example of how more complex feature detectors can be built by combining multiple simple convolutional filter operations.
Figure 10 shows the 2d discrete convolution of a 5×5 pixels image layer with a 3×3 kernel matrix to produce a 3×3 pixels image result. The kernel is positioned on top of the input image and applied in a sliding window fashion. For each sliding position the values of kernel are multiplied by the values of the image region under the kernel and summed up (scalar product) to result in one pixel of the output image. (see Dumoulin and Visin, 2018)
Most machine learning frameworks, including PyTorch, actually implement cross-correlation instead of convolution (see "Conv2d", 2023), although the two operations have a subtle difference. Cross-correlation applies the kernel without reflection, while convolution applies the kernel reflected about both axes, equivalent to a kernel rotated by 180-degree (Khan et al., 2018, p. 46). In the context of machine learning, this difference is indeed negligible as long as the same operation is used consistently. This thesis explains the cross-correlation operation while referring to it as *convolution* in that broader sense.
In general, the image output $\mathbf{I}^O$ of basic 2d discrete convolution is obtained with

$$\mathbf{I}^O[u,v] = \sum_{i=0}^{K-1}\sum_{j=0}^{K-1} \mathbf{I}^I[u+i, v+j] \cdot \mathcal{H}[i,j], \tag{3.1}$$

where $\mathbf{I}^I$ is the input image and $\mathcal{H} \in \mathbb{R}^{K \times K}$ is the filter kernel. Throughout this thesis, images and kernels are indexed with the spatial origin $[0,0]$ being at the top left position. (cf. Burger and Burge, 2022, p. 96)
Convolution can be run efficiently on GPUs and SIMD-enabled processors because it can

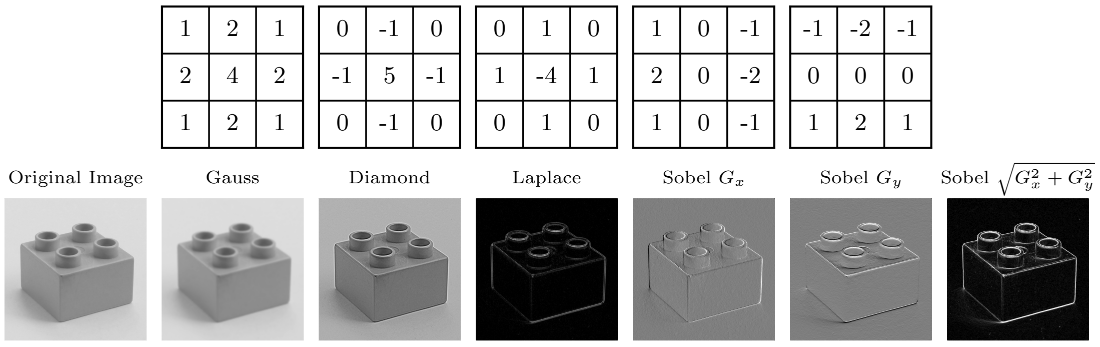

Figure 9: Different filter kernels applied to an image with 2d discrete convolution. The zero point is represented as medium gray for the Sobel components, as the filter output contains negative values.

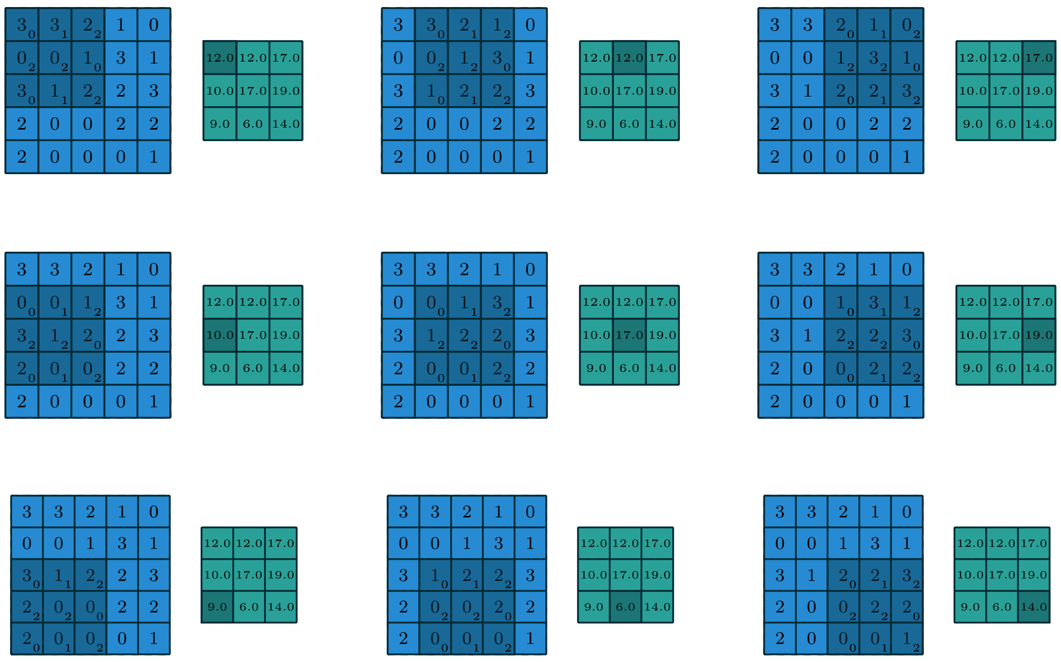

Figure 10: The nine convolution steps of a 5×5 image (blue) with a 3×3 kernel (shaded overlay) lead to a 3×3 output image (green). Adapted from Dumoulin and Visin, 2018

be represented as matrix multiplication.

3.1.2. Convolutional Layers for Neural Networks

When convolution is used inside neural networks, however, the filter kernels are not predefined constants. Instead, they usually are part of the neural network's weights that are *learned* during the training phase of the neural network e.g. using supervised learning. It's values are typically initiated with random numbers before they get adjusted iteratively during training. A back propagation algorithm like the Adam optimizer is able to adjust the weights in a way that minimizes the output error of the network, that is measured by a custom defined loss-function. (see Khan et al., 2018, p. 46)
The convolutional layer is the most basic processing block of every CNNs. It extends the two-dimensional convolution operation by the ability to process images with multiple channels. This can be RGB images in their three channels red, green, and blue, which are input into the first layer of a CNN – or output at the last layer as the prediction result of a CNN. However, in the hidden layers of a CNN much higher channel counts are used to represent abstract image features, which is why the image channels are also referred to as *feature maps.*
For the input $\mathbf{F^I} \in \mathbb{R}^{H \times W \times C_i}$ a convolutional layer produces the output $\mathbf{F^O} \in \mathbb{R}^{C_o \times K \times K}$ by applying a $\mathcal{W} \in \mathbb{R}^{C_o \times C_i \times K \times K}$ filter kernel matrix in a sliding window to the image.
An important aspect of convolutional layers is, that the number of input channels C_i and the number of output channels C_o can be chosen independently. This way, a convolutional layer can remix the input to a higher number of feature maps. In conjunction with a decrease of spatial resolution, this transform often is applied as part of feature extraction, which will be covered in Subsection 3.2.1. On the other hand, reducing the number of feature maps can be useful at the end of a CNN, to flatten a higher dimensional representation to a lower dimensional output, as it will be described in Section 3.2.
Given the example of $C_i = 3$ and $C_o = 1$, the kernel is in fact a set of three $K \times K$ filters, each operating on one of the input feature maps. For each sliding position, each filter is multiplied with the values of the receptive field on the corresponding input feature map. The summation of all three filter results yields one output pixel of $\mathbf{F^O}$.
For an output $\mathbf{F^O}$ with multiple channels, this kind of $C_i \times K \times K$ filter set is needed for each of the output channels, adding another dimension to the kernel. To convolve an input $\mathbf{F^I} \in \mathbb{R}^{H \times W \times C_i}$ to an output $\mathbf{F^O} \in \mathbb{R}^{C_o \times K \times K}$, therefore a kernel $\mathcal{W} \in \mathbb{R}^{C_o \times C_i \times K \times K}$ is needed. (see Khan et al., 2018, p. 46).

3.1.3. Convolution Variants and Pooling Layers

This subsection describes how the output size of a convolutional layer can be controlled by the use of zero padding and strided convolution. After that, the concepts of transposed convolution and max-pooling are presented.

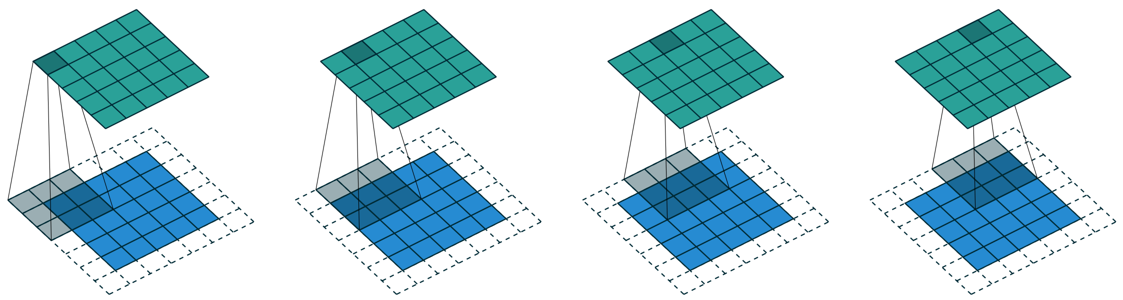
Figure 11: Same padding convolution: zero-padding applied before convolution maintains the original image size at the output (adapted from Dumoulin and Visin, 2018).

Convolution usually reduces the both image's width and height by $K-1$, as the kernel stays inside the image during sliding. To compensate for the size reduction, an additional pixel padding can be applied around the input before convolution, as visualized in Figure 11. This padding can be filled with zeros or the values of the nearest border pixels of the image.

To build a convolution layer that outputs feature maps which have the same size as the input maps a stride of 1 is used together with a padding value of $(kernelsize-1)/2$. This allows the kernel's center to step onto every pixel of the input. (Khan et al., 2018, p. 49)

Strided convolution is a way of reducing the spatial dimensions of the output $\mathbf{F^O}$. The *stride* parameter controls the step size by which the kernel moves on the input $\mathbf{F^I}$. It has a direct impact on the output size as each step position of the kernel convolves to one output pixel. For *stride*=1, the kernel is translated by one unit on the input for each convolution step, which is also referred to as *unit stride*. Figure 12 shows how a stride of 2 leads to an output with reduced spatial dimensions. If no padding is employed, the output width W_o is related to the input width W_i, the kernel size K, and the stride S by

$$W_o = \lfloor \frac{W_i - K}{S} \rfloor + 1. \tag{3.2}$$

The output height depends on the input height respectively. (Dumoulin & Visin, 2018)

Strided convolution can be used as an alternative to max-pooling (cf. GoogLeNet, Szegedy et al., 2014). Max-pooling will be described in the next section.

Instead of an input pixel region of the kernel's size affecting a single output pixel for each kernel step, transposed convolution works the opposite way: for each kernel step, a single input pixel affects output pixels in a region of the kernel's size. This way a larger output image than the input can be achieved, which is not possible with classical convolution. The transposed convolutional layer is used in encoder-decoder architectures mainly as a counterpart to the classical convolutional layer to undo its spatial reduction. (see Khan et al., 2018, p. 57)

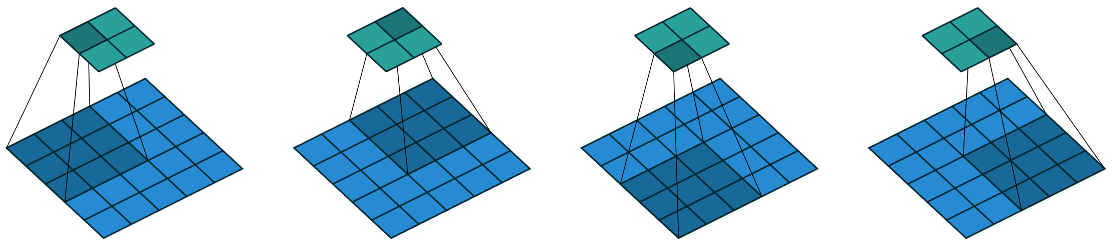
Figure 12: Convolving a 3 × 3 kernel over a 4 × 4 input with a stride of 1 unit (adapted from Dumoulin and Visin, 2018).

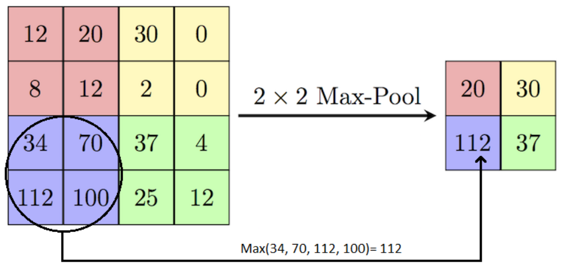

Figure 13: 2 × 2 Max Pooling with stride=2. (Adapted from Sultana et al., 2018, Fig. 3).

The details of transposed convolution will not be discussed in detail, as for image processing it is mostly replaced by a fixed bilinear upscale filter followed by a classical convolution layer (cf. (Ronneberger et al., 2015; T. Zhang et al., 2022)).
The max-pooling layer uses a max-operation kernel instead of kernel multiplication with summation: The largest pixel of a kernel's receptive field becomes the output pixel. The other pixels are discarded. In a pooling layer, the kernel is usually moved without overlap, i.e. with a step size of $stride = kernelsize$. This reduces the width and height of the image by half for a 2×2 kernel, as shown in Figure 13. When the input consists of multiple feature maps, max pooling is normally applied to each feature map individually.

3.2. CNNs for Image Processing

3.2.1. Feature Extraction Inside CNNs

Convolutional neural networks (CNNs) like the AlexNet (Krizhevsky et al., 2017) leverage the fact that multiple convolution operations chained together can build even more complex pattern matching and image transformation filters. Today, CNN models are often superior to conventional algorithms for image processing tasks such as demosaicing, denoising, and upscaling because they take into account learned experience about image content during reconstruction.

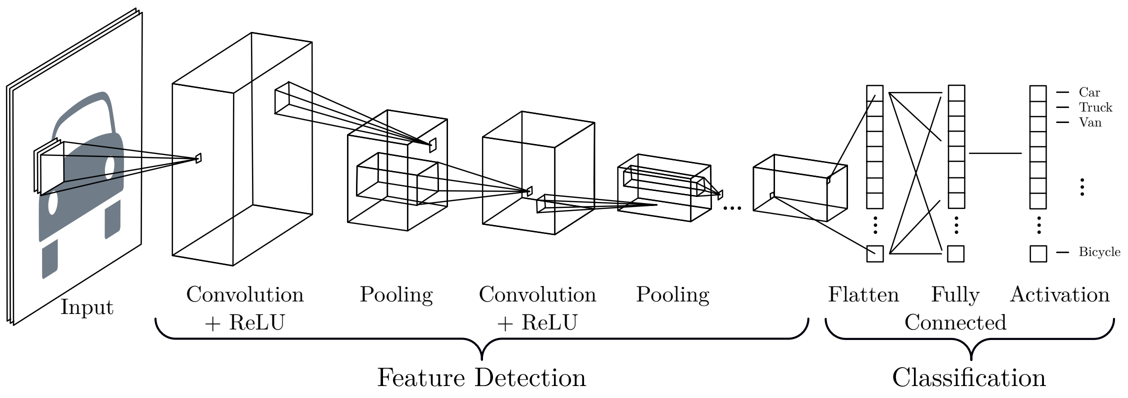

Figure 14: Typical CNN architecture of an image classifier consisting of general purpose feature extraction part and a classification part. (Figure based on Mathworks Inc, 2021, p. 8)

A basic CNN (or ConvNet) combines multiple pairs of convolution and pooling layers to form the feature extractor. While a single convolution operation is only able to detect basic local features like gradients or edges, this group of layers is able to construct more complex feature representations by feature combinations of the previous layers. Layer by layer the information about *what* is in the image is enhanced while information about the *where* is lost. In Figure 14, three-dimensional boxes indicate how the feature representation changes with each layer of the CNN. Each convolution and pooling iteration decreases the spatial dimensions of the features, while increasing the number of feature maps as indicated by the horizontal length of the boxes.
The feature detection starts with the full resolution image input of a car. The three image channels, red, green, and blue can be seen as three initial feature maps. The line beams indicate how the kernel's receptive field on the input forms one pixel of the convolution output. With the convolution layers, the number of feature maps increases, indicated by the box becoming longer. The Max-Pooling steps reduce spatial resolution without influencing the feature dimension.
After feature extraction layers, further layers follow depending on the usecase. For the classification task, as shown in Figure 14, the extracted features are flattened and serialized using a fully connected layer and an activation function to produce classification probabilities for each class label.
As depicted in Figure 14,a non-linear activation function, such as ReLU, is applied after convolution as illustrated in Figure 14. For image classification tasks, the relationship between the image and its target class is non-linear. The required flexibility to model non-linearity is achieved through the use of activation functions. A CNN lacking non-linear operations would be limited to learning linear relations. (see Khan et al., 2018, p. 54)

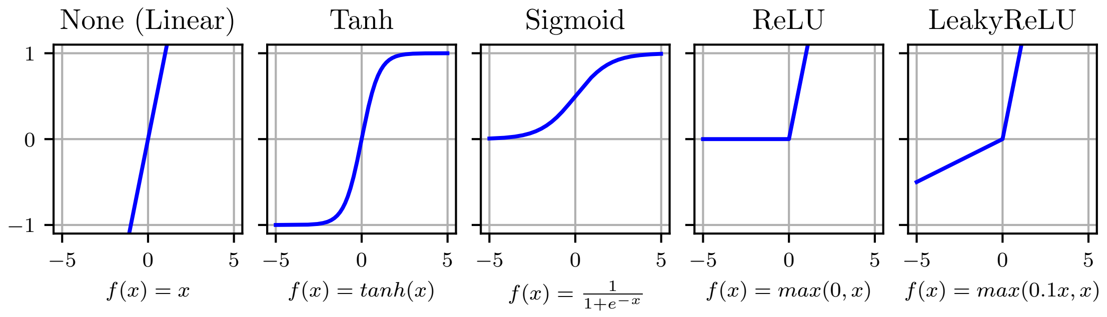

Figure 15: Popular activation functions

Figure 15 provides an overview of commonly used activation functions, including hyperbolic tangent and sigmoidal functions, rectified linear units (ReLU), and leakyReLU.
In earlier CNN architectures, hyperbolic tangent and sigmoidal activations were utilized. However, it was discovered that the newer ReLU function allows for faster training while also mitigating the vanishing gradient problem of hyperbolic tangent and sigmoidal activations (Murphy, 2016). The ReLU function cuts off values below zero by mapping them to zero. To prevent weights from no longer adapting during training, LeakyReLU preserves negative values by damping them by a factor of 0.1. (see Khan et al., 2018, p. 54)
Unlike convolution and pooling operations, activation functions do not alter the data's shape because they are applied element-wise.

3.2.2. From Image Classification to General Image Processing

In contrast to the CNN described in Figure 14, the UNet (Figure 16) is able to preserve the spatial information about the image features. It was initially proposed by Ronneberger et al. (2015) for biomedical image segmentation. Image segmentation is a classification task on the pixel level: While general image classification networks typically predict the assignment of an entire image to a specific class as a single probability value, the UNet is designed to predict an assignment probability for each pixel. As the prediction output has the same spatial dimensions as the input image it can be treated as an image with multiple grayscale channels representing segmentation masks of different classes.
The ability of outputting another image is what makes the UNet architecture applicable far beyond biomedical image segmentation and predicting class assignment probabilities. In fact UNet has been adopted to a wide range of image reconstruction tasks like for example high dynamic range reconstruction (Eisemann et al., 2020) or denoising (Gurrola-Ramos et al., 2021).

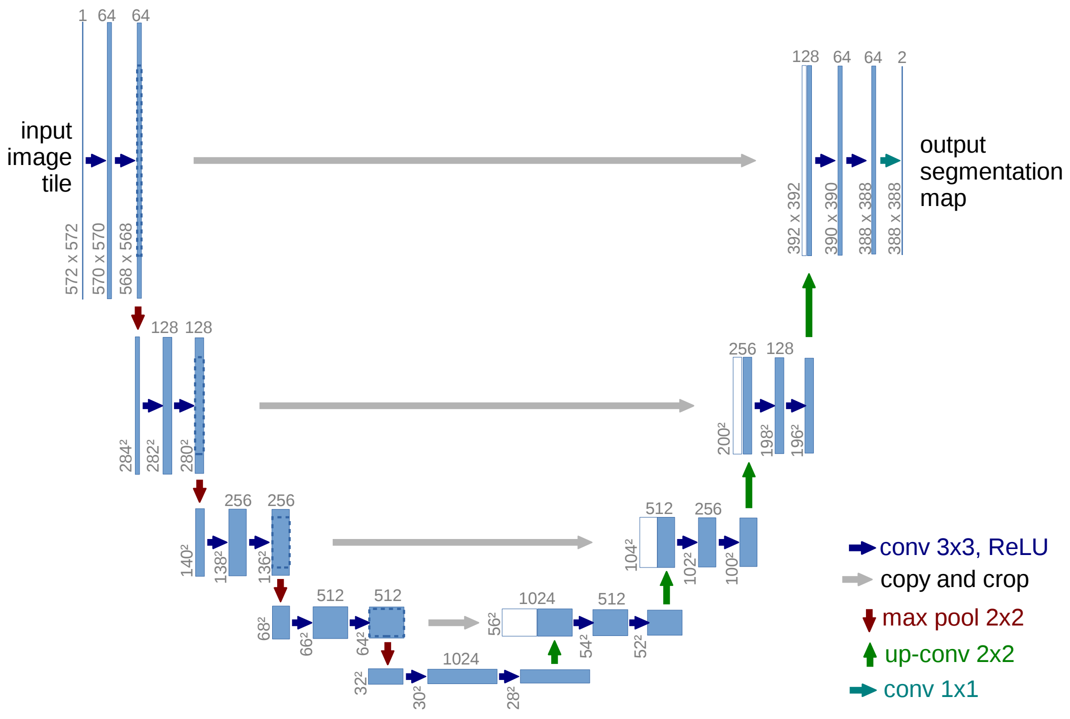

Figure 16: UNet Architecture (Adapted from Ronneberger et al., 2015, Fig. 1).

3.2.3. The UNet's U-Shape or the Encoder-Decoder Principle

The UNet's architecture (Figure 16) can be described in two parts: The left side is the "contracting path" which reduces the spatial resolution with each level down while increasing the amount of feature maps. The right side is the "expansion path" which increases spatial resolution again while reducing the feature dimension. Both paths together form the eponymous U-Shape which can be seen as a bottleneck: The reduction of the spatial dimensions is a constraint that forces the neural network to find a lightweight abstract representation during training that is optimal in preserving the relevant information. Reminiscent of image compression algorithms like JPEG, the two parts of the UNet architecture are also referred to as *encoder* and *decoder* (e.g. Y. Xing et al., 2020) as the spatial image samples are compressed into an abstract feature representation and then decoded back to the original spatial resolution.

3.2.4. UNet Optimizations

It can be seen in Figure 16 that the input dimensions are bigger than the output because the original UNet handles arbitrarily large images by breaking up the input into separate, overlapping tiles. The overlapping padding around each tile offers additional context for the border regions to guarantee that the processed tiles can be seamlessly reassembled.

The padding is cropped during the process, represented by the dashed lines ontop of the blue boxes in Figure 16.
Skip connections are used to allow information to bypass the “bottleneck” on the different levels. This is done by copying the output feature maps from the left side and concatenating them with the corresponding inputs on the right side. Ronneberger et al. (2015) show that these skip connections can improve performance without significantly increasing the complexity of the model as they make it easier for the model to preserve pixel-level high resolution details. Recent experiments by Eisemann et al. (2020) confirm that image reconstruction tasks can benefit from skip connections.

3.3. Universal Demosaicing with Convolutional Neural Networks

3.3.1. Training Demosaic Models With Supervised Learning

To teach a CNN the demosaicing process, the supervised learning approach is used because the desired output is information that can be provided during training. The required training data consists of pairs of mosaic image inputs and their corresponding optimal demosaiced targets, which are the *true labels*, the desired outputs of the CNN. Section 4.1 within the Experimental Setup chapter therefore deals in detail with compiling such a training set by simulating mosaic images from fully sampled RGB ground truth images.

3.3.2. Handling Regular vs. Irregular Image Samples

This chapter is about CNN architectures that can be trained for demosaicing of arbitrary CFA patterns. Most CNN architectures designed by the scientific community for demosaicing assume Bayer CFAs (cf. Ignatov et al., 2020; Liu et al., 2020; Qian et al., 2021; W. Xing and Egiazarian, 2021; T. Zhang et al., 2022; Zhou et al., 2018). However, it can be assumed that at least some of them can be trained on alternative random and pseudo-stochastic patterns, even if this is not explicitly intended by the respective authors. This becomes difficult with architectures that are too optimized for the Bayer pattern, or even contain hard-coded preprocessing steps that make use of the repetitive Bayer layout. More promising are those approaches that provide a separate input for the CFA mask and only specialize in the spatial color encoding of the input through training. In the following, we discuss the requirements for this and describe spatially-adaptive convolution, as implemented in the SANet architecture, as a possible solution for universal demosaicing.
Most CNNs for debayering use the regularity of the pattern to first group the mosaic sensor pixels into four separate color planes by the four positions R,G,G,B of the Bayer pattern. These color channels have only half the pixel height and width and have to be upsampled later, but they are spatially homogeneously sampled, i.e. they do not contain

any missing pixel information.
Irregular CFA patterns do not allow such a decomposition of the mosaic sensor image into four consistent layers. Instead, by knowing the CFA mask, the mosaic sensor image can be decomposed into the three layers for the colors red, green, and blue. However, since each pixel position is defined in only one color channel, these channels contain a portion of undefined values at irregular positions, denoted by zeros, for example. These random holes impose an additional structure overlaying the image information that interferes with pattern recognition within the image.

3.3.3. AdaConv – a Spatially Adaptive Convolutional Layer for Demosaicing

T. Zhang et al. (2022) proposes an alternative method to apply convolution directly to mosaic sensor images: a modified, spatially adaptive convolution layer, *AdaConv*, for CNNs. Instead of a static kernel that is applied to each location of an image in a sliding window fashion, its convolution kernel is adjusted differently for each location depending on the CFA pattern that must be provided as additional input. While T. Zhang et al. (2022), like most demosaicing research, designed their SANet architecture for regular Bayer CFAs, it is subject of this thesis to show that it is indeed applicable for universal demosaicing.
This section first explains the difference between classical and spatially adaptive convolution and shows how it can be made more efficient by using the kernel decomposition technique. Afterwards, this section illustrates the design of the spatially adaptive convolution block AdaConv for demosaicing purposes (T. Zhang et al., 2022).
Convolution inside a classical convolution layer can be described as

$$\mathbf{F}^I * \mathcal{W}_1 = \mathbf{F}^O, \tag{3.3}$$

with the filter kernel $\mathcal{W}_1 \in \mathbb{R}^{C_o \times C_i \times K \times K}$. Here, C_o represents the number of output channels, C_i represents the number of input channels, and K represents the kernel size.
Spatially adaptive convolution,

$$\mathbf{F}^I ** \mathcal{H} = \mathbf{F}^O, \tag{3.4}$$

in its pure form would mean that a separate $K \times K$ filter would be applied *for each sliding position on the input image*. Performing spatially adaptive convolution on an input image $\mathbf{F}^I$ with same padding and a stride of 1 needs a filter kernel $\mathcal{H}$ of size $C_o \times C_i \times H \times W \times K \times K$, which is $H \cdot W$ times larger than a classical convolution kernel. This would mean a massive increase in memory usage and number of parameters to be trained, particularly when processing larger images. (see T. Zhang et al., 2022)
The authors of SANet use kernel decomposition to reduce memory consumption to only 1.8 times that of classical convolution (see T. Zhang et al., 2022). Instead of a single

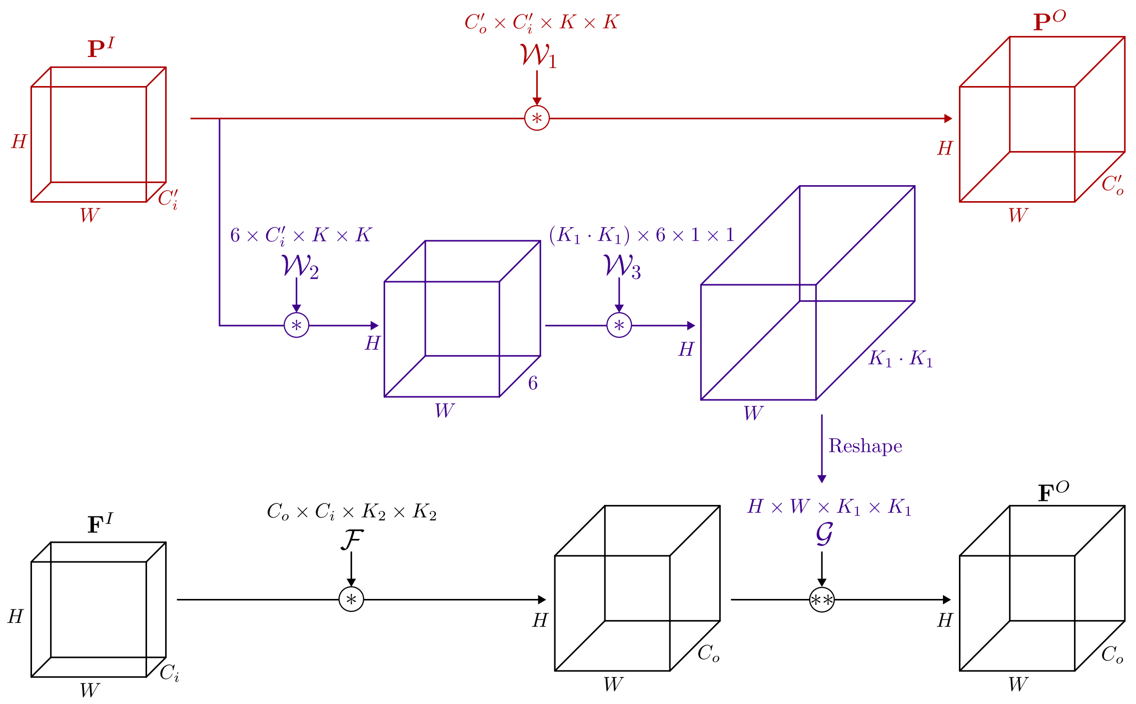

Figure 17: Spatially adaptive Convolution Layer (AdaConv) with separate processing of the sampling pattern information (red) and image information (black). The kernel extraction branch (purple) derives the spatially adaptive kernel $\mathcal{G}$ from CFA pattern information. Own illustration based on the SANet source code (T. Zhang, 2022, October 2/2023)

adaptive convolution operation with the large adaptive kernel $\mathcal{H}$, they combine a classical convolution with a spatially adaptive convolution in a row, with

$$\left(\mathbf{F}^I * \mathcal{F}\right) ** \mathcal{G} = \mathbf{F}^O. \tag{3.5}$$

Here, $\mathcal{F} \in \mathbb{R}^{C_o \times C_i \times H \times W \times K_2 \times K_2}$ represents the classical convolution kernel and $\mathcal{G} \in \mathbb{R}^{H \times W \times K_2 \times K_2}$ the spatially adaptive kernel. K_2 and K_1 are chosen so that

$$K_2 + K_1 - 1 = K, \tag{3.6}$$

to maintain the total receptive field of $\mathcal{H}$. The conversion of the number of input feature maps C_i to the number of output features C_o is accomplished with $\mathcal{F}$ while kernel G is shared across the feature dimension. (T. Zhang et al., 2022)

The complete structure of the spatially adaptive convolutional layer, *AdaConv*, is depicted in Figure 17. It includes separate inputs for CFA pattern $\mathbf{P}^I$ and image information $\mathbf{F}^I$. The bottom row (black) shows how $\mathcal{F}$ and $\mathcal{G}$ process the image information from input $\mathbf{F}^I$ to produce the output image feature maps $\mathbf{F}^O$.

The spatially adaptive kernel $\mathcal{G}$ differs from the other kernels, as it is not directly learned as a weight, but rather derived from pattern information $\mathbf{P}^I$. The weights $\mathcal{W}_2$ and $\mathcal{W}_2$ learn a kernel extraction function (purple), that ouputs $\mathcal{G}$ in the appropriate shape.
Finally, the top row of Figure 17 (red) shows, how the pattern information $\mathbf{P}^I$ is convolved with a simple $K \times K$ kernel $\mathcal{W}_1$, resulting in the separate pattern information output $\mathbf{P}^O$. This allows further parallel processing of the pattern information in architectures like the UNet, as explained in the next section. (see T. Zhang et al., 2022)
The official PyTorch implementation of the spatially AdaConv layer (T. Zhang, 2022, October 2/2023) has the similar parameters to those known from a standard PyTorch conv layer to control stride, kernel size as well as the number of input and output features. This makes it applicable as a replacement for classical convolutional layers in any convolutional network architecture.
The idea behind the approach by T. Zhang et al. (2022) is that each pixel position requires a different interpolation method, depending on the CFA pattern at that position. This was already illustrated in Subsection 2.2.1 for linear debayering and is even more applicable to irregular CFA patterns, where an even larger variety of local sample constellations occur. Simple convolution applies the same filtering kernel to all pixel positions, which means that only a single interpolation is possible per learned feature. Only through the combination of multiple convolution and pooling layers can adaptive structures be developed. Spatially adaptive convolution layers provide the required flexibility at the feature level and are superior to classical convolution in demosaicing, as the investigations of T. Zhang et al. (2022) show.

3.3.4. SANet – Spatially Adaptive Convolution Inside the UNet

The SANet can be understood as two interlinked UNets - one for the mosaic image input and one for the CFA pattern (see Figure 18, left). The first UNet is responsible for image reconstruction, with pattern information from the second UNet injected at each level. In their official PyTorch implementation, the authors achieve this duality within a single UNet, with every network layer featuring separate inputs and outputs for image and pattern information. Adaptive Convolution (*AdaConv*) Layers are the points where pattern information is combined with image information, and replace the convolution blocks of classical UNets. Instead of employing a singular AdaConv per Unet Level, they always appear as a pair of two, forming a *Resblock*, as depicted in Figure 18 on the righthand side. Using two convolutional layers in sequence is an efficient way to increase the receptive field compared to a single convolution with a larger kernel size (cf. Zhou et al., 2018). The ResBlock includes a local residual shortcut that goes from the input to the output through a simple 1×1 convolution to convert between different numbers of input and

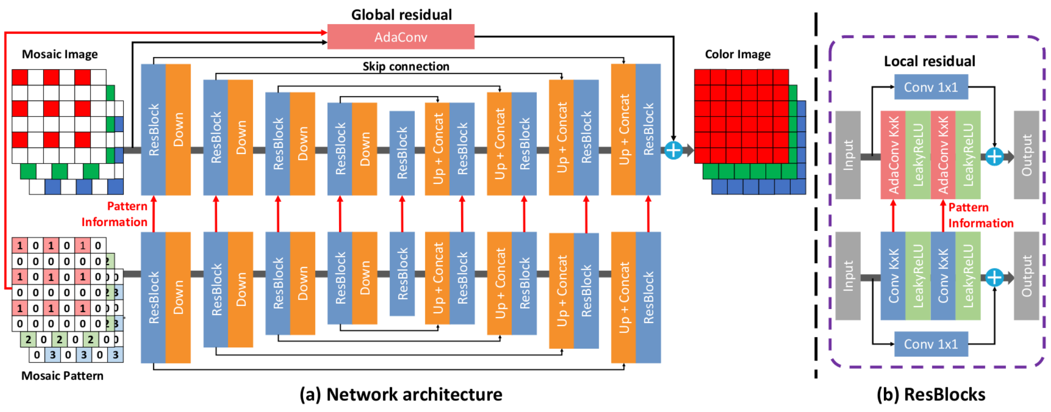

Figure 18: SANet architecture (left) and ResBlock (right), adapted from T. Zhang et al., 2022

output features.

4. Experimental Setup

This chapter documents the experiments that are conducted during this thesis. The first thing is building the training set by simulating labelled data from an existing image set.

4.1. Building the Mosaic Image Training Set

Since this thesis utilizes supervised learning, so-called labeled data is needed to train the CNN. In case of demosaicing this means pairs of raw CFA sensor images (CNN input) and their corresponding optimal reconstruction (ground truth). The optimal reconstruction would be the one identical to the image of a hypothetical camera that captures full rgb samples for every pixel.
Cameras with irregular CFA filters, as they are examined in this thesis, are not yet developed. Therefore, actual raw mosaic sensor images cannot be used as training input for the neural network. Even if this type of data existed, it would be challenging to employ in supervised learning since the corresponding perfect reconstruction, required as target labels for training, is not obtainable. However, Backes and Fröhlich (2020) have shown that mosaic raw images from any CFA can be realistically simulated from fully sampled RGB ground truth images. The remaining obstacle is searching for full-color RGB raw images that are appropriate for use as ground-truth data, given that the majority of current cameras use a color filter array. Therefore, we will initially assess existing datasets with regard to their suitability as ground truth data and decide on the PixelShift200 image dataset.

4.1.1. Choosing PixelShift200 from Existing Image Datasets

In this section, four image sets are discussed that are commonly used in the context of demosaicing: the Kodak, Arri, PixelShift200, and the Moiré image set. While the first two of them turn out to be most suitable for benchmarking, the latter two are specifically designed for training neural networks. After an overview of the image sets, PixelShift200 will be argued to be most suitable for building the training set for this thesis.

The Kodak Image Set

The Kodak image set, which was released back in 1991, consists of 24 analog photos that are scanned from negatives and provided as 8bit sRGB. These images have been fully RGB sampled without introducing demosaicing artifacts, since analog cameras do not involve a CFA. The original Kodak image set existed in a "Full Detail" resolution of 3072×2048 pixels, today, researchers mainly use the downsized "TV-comparable" version of the set with a resolution of 768×512 pixels to benchmark image compression and demosaicing against legacy algorithms (Andriani et al., 2013).
However, none of the variants is optimally suited for training a CNN because the spatial frequency characteristics as well as the color representation of analog film and the involved scanning process may differ from digital sensors and raw image processing pipelines. Another limitation of the Kodak image set is the small number of images included in the set, which may not sufficiently capture the wide array of digital photography. This would however be beneficial for accurate inference by a CNN. Therefore, it can be concluded that although the Kodak image set is a recognized benchmark, it does not satisfy the training pipeline requirements of this thesis.

The Arri Image Set

The Arri image set, published by Andriani et al. (2013), was created as an up-to-date response to the Kodak image set. It will be covered in more detail here as it was utilized in the experiments of Backes and Fröhlich (2020), which this thesis is based on. The set consists of twelve images, as depicted in Figure 19, which are individual frames extracted from video footage captured with an Arri Alexa cinematic camera. The images preserve a high dynamic range and are provided in a raw 16bit-TIFF format.
The last one of the twelve images (see Figure 19) uses a special modified camera body that does not have color filters on the sensor at all. The image is captured using a color filter wheel in front of the camera lens to capture the red, green and blue channel of an image separately. These are combined to a high resolution fully sampled raw image which is optimal for the purpose of simulating other color filter arrays. The results of the color-wheel-technique are comparable to the images from the PixelShift200 database described later.
However, the other eleven images from the set are shot with an Arri Alexa cinematic camera that uses a normal Bayer sensor, which means that the raw images are not fully RGB subsampled, but only capture either the red, green, or blue light intensity per pixel. This kind of raw Bayer images could only be used in this thesis if a preprocessing step is applied: Backes and Fröhlich (2020) show that raw image data from a Bayer sensor can be used to simulate other CFA patterns by first demosaicing and downscaling the original

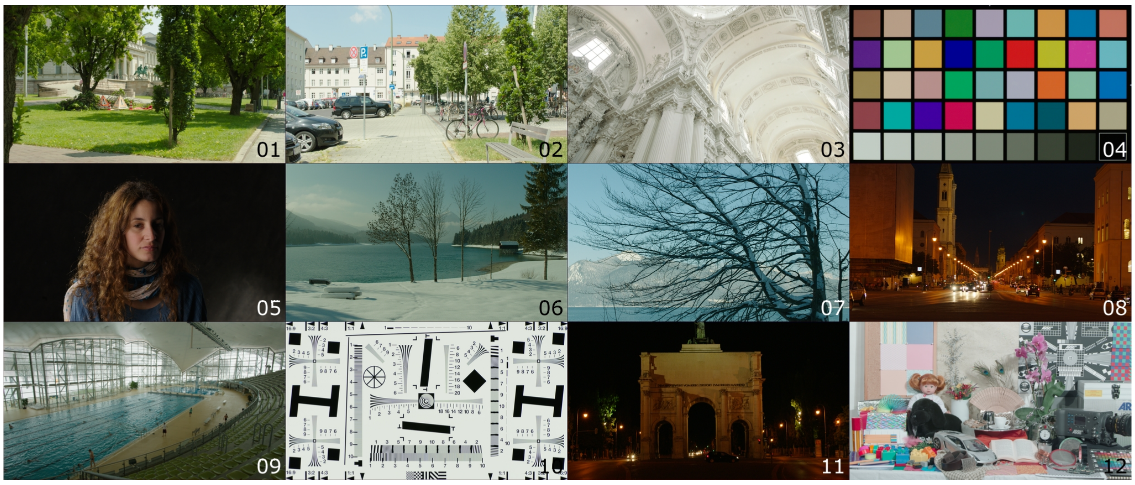

Figure 19: Overview of the Arri image set. Adapted from Andriani et al., 2013

raw image data to get a fully sampled RGB ground truth image. However, this workflow seems suboptimal in comparison to the full RGB raw image data from the color-wheel or the PixelShift technology.

In the work of Backes and Fröhlich (2020), this image set was used to benchmark their demosaicing algorithm (dFSR) on different pseudo-random CFA patterns. Since this thesis builds on the CFA patterns from their research, one could argue for the use of this image set to allow a direct comparison. However, for tasks such as image classification networks, typically large datasets of images, such as ImageNet (Deng et al., 2009), are used. These datasets are compiled from various internet sources to cover a wide range of subjects, locations, times of day, and camera models. In this context, using only twelve images from the Arri dataset may be insufficient for learning demosaicing and lead to poorer inference when the network is presented with real camera data. Although multiple training samples could be extracted from each of the twelve Arri images for a 256 × 256 pixel training pipeline, the image material only covers a limited number of subjects overall. Additionally, partitioning the dataset into images for training and validation further reduces the available training data. Especially, if the aim is for the neural network to consider learned knowledge about typical image content during image reconstruction, a larger variety of image subjects would be beneficial.

The PixelShift200 Dataset

The PixelShift200 image dataset was designed by Qian et al. (2021) specifically for use as training data for demosaicing and denoising CNNs. While their CNN architecture "TENet" presented in conjunction with the data set is too much optimized for Bayer sensors to be applicable in the context of this thesis, the PixelShift200 image set is suitable for simulating any CFAs since it consists of fully sampled raw RGB data acquired using PixelShift technology. The set comprises 200 training and 10 validation images in 4K resolution. A selection of the images is previewed in Figure 20.

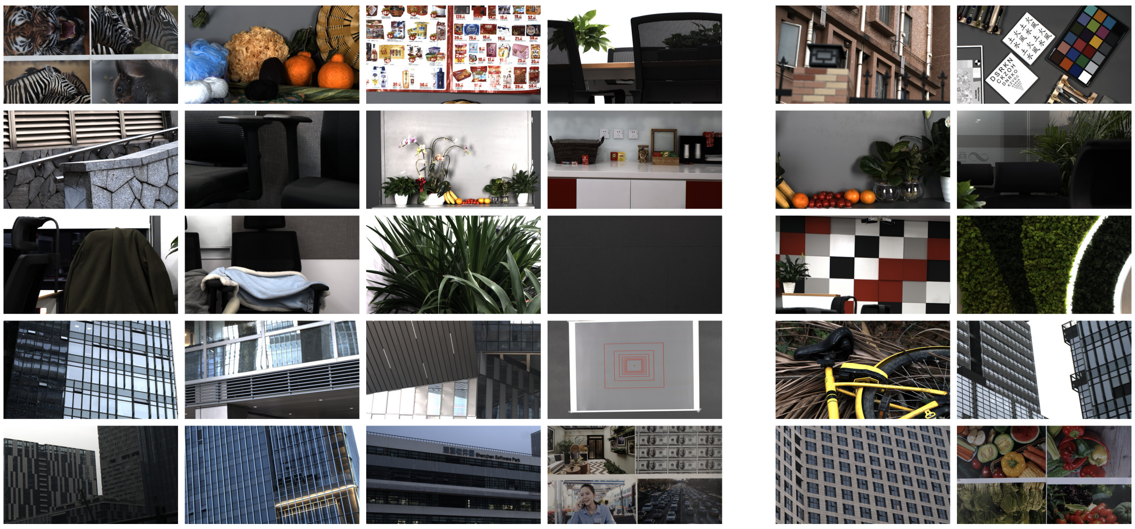

Figure 20: Overview of the PixelShift200 image set (Qian et al., 2021): Preview of random 20 of 200 training images (left) and all 10 validation images (right)

The pictures of the dataset were recorded with a SONY α7R III full frame camera. This camera is equipped with a standard Bayer CFA but uses a sensor shifting technology to capture a fully sampled RGB image. Four images are taken shortly after each other, with the camera's Bayer sensor shifted horizontally or vertically by one pixel for each image. Across the four shots, each pixel is sampled once by each of the four Bayer quadrants – red, green, green and blue. How the four separate images can be combined into a fully sampled RGB image is shown in Figure 21.
The fully sampled RGB PixelShift images can be used to simulate any CFA pattern without prior debayering or downscaling of the raw images as required with the Arri image set. This eliminates the risk of introducing artifacts into the training data. PixelShift images can be used for simulation at full 1:1 pixel resolution, preserving the original sensor characteristics. Provided that the sensor shift mechanism moves accurately by one pixel, the CFA simulation should be very close to a camera that actually has the CFA pattern physically implemented.
PixelShift technology can produce artifacts when capturing moving subjects due to its temporally distributed capturing process. It appears that the PixelShift200 dataset avoids including subjects with motion. Considering that moving subjects, longer exposure times, and motion blur are inherent aspects of photography, it would be advantageous to incorporate such images into the training data. Because the images are otherwise

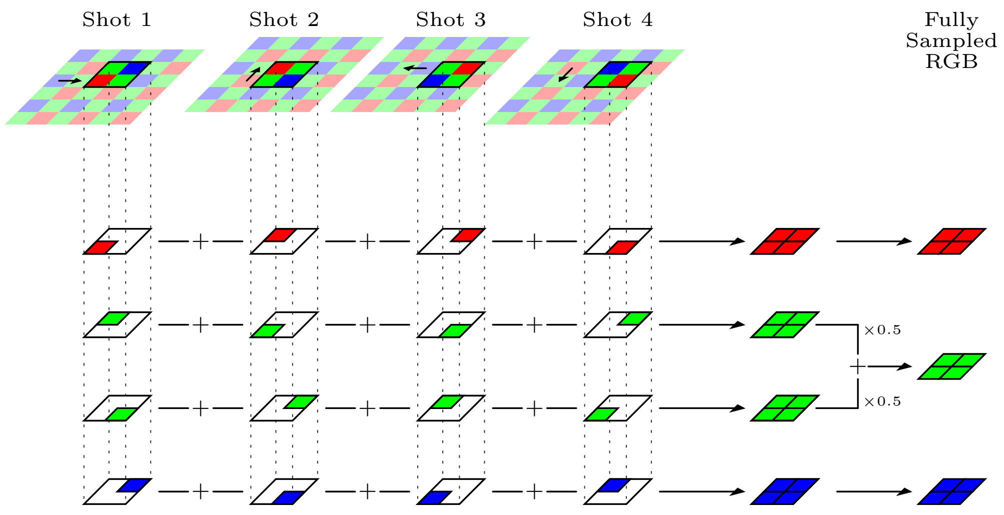

Figure 21: An exemplary 2 × 2 pixel image area gets fully sampled with PixelShift Technology. The camera's Bayer sensor is shifted horizontally and vertically while the camera takes four images so every pixel position is captured with a red-, green- and a blue-pass filter. (Own illustration inspired by Qian et al., 2021, Fig. 4)

well-suited for CFA simulation, this drawback is being accepted.
The images have very low noise levels. If these images were used to train a denoiser, the noise would have to be simulated. However, since this thesis does not address denoising, the PixelShift200 dataset remains a reasonable choice.

The Moiré Dataset

Lastly, even though not applied in this thesis, the Moiré image set shall be at least mentioned here, as it is based on an idea that could still be relevant to this thesis. While the other image sets are manually compiled by capturing or collecting images, the Moiré dataset relies on a data-driven approach to collect image patches that are prone to producing aliasing effects like moiré or zipper artifacts during demosaicing. To build the Moiré dataset, images from various internet sources where first resampled with the Bayer pattern and then reconstructed with a demosaicing CNN. The parts, where the reconstruction differs most from the original images, were extracted to form the Moiré image set. Using this dataset to train a demosaicing CNN is equivalent to re-weighting the loss function towards challenging structures. (Gharbi et al., 2016)
While the Moiré dataset is an interesting concept, it was not applied in this thesis, as the patches are selected with the Bayer pattern in mind. When reconstructing images from

non-regular sampling patterns, different kind of image structures may be challenging to reconstruct. Further investigations would be needed to verify that training would benefit from the Moiré image set in our case. It could then also be used to fine-tune a model pretrained with PixelShift200.

4.1.2. Image Preparation and Data Augmentation

As shown in Figure 22 on the left, the PixelShift200 image set is first extracted form the RAW images by using the original code provided by the authors. After that, five 512 × 512 pixel crops are extracted from each original 4K PixelShift200 image, as explained in on the right side of Figure 22. These crops are then stored as 16bit .tif-files. Figure 23 visualizes, how the cropping operation works geometrically.
As shown in Figure 24, inside the PyTorch Dataloader, different random batches of images are selected for each epoch of training, while a fixed batch order is used for validation. The image batches are converted to PyTorch floating point tensors, before they are linearly downscaled by factor 2. The geometric augmentation consists of a random rotation, that is only applied during training. After augmentation, the CFA is simulated, by multiplying the image with the RGB pattern mask. The resulting training and validation inputs and ground truth targets have a resolution of 256 × 256 pixels. The training set comprises 1000 image samples, while the validation comprises 50.
Although the fully sampled PixelShift200 images allow for sensor simulation at full 1:1 pixel resolution, a factor 2 downscale is applied, as shown in Figure 22. The reason for this decision is that the PixelShift200 images appear somewhat blurry at full resolution compared to the Arri Image Set used in Backes (2019). However, the primary challenge of demosaicing is to reconstruct sharp details on a pixel scale as this is where undersampling can lead to reconstruction artifacts such as Moiré or aliasing. The downscaling increases the difficulty of the training set in this regard. This allows a proper judgement of the actual reconstruction performance and brings it closer to the Arri Image Set. Another issue resolved with the downsampling is that a small amount of the PixelShift200 images contain zipper artifacts, as shown in Figure 25. These may originate either from the camera being moved during exposure or from the sensor shifting mechanism not working

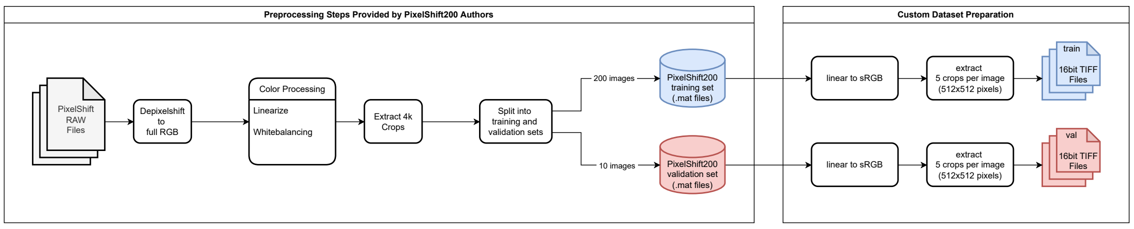
Preprocessing Steps Provided by PixelShift200 Authors
PixelShift RAW Files
Depixelshift to full RGB
Color Processing
Linearize
Whitebalancing
Extract 4k Crops
Split into training and validation sets
200 images
10 images
PixelShift200 training set (.mat files)
PixelShift200 validation set (.mat files)
Custom Dataset Preparation
linear to sRGB
extract 5 crops per image (512x512 pixels)
train
16bit TIFF Files
linear to sRGB
extract 5 crops per image (512x512 pixels)
val
16bit TIFF Files

Figure 22: Building a training and validation set from the PixelShift200 image dataset

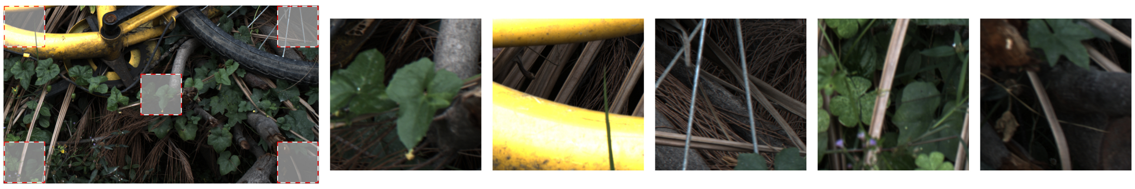
Figure 23: Extracting 5 crops per PixelShift200 image

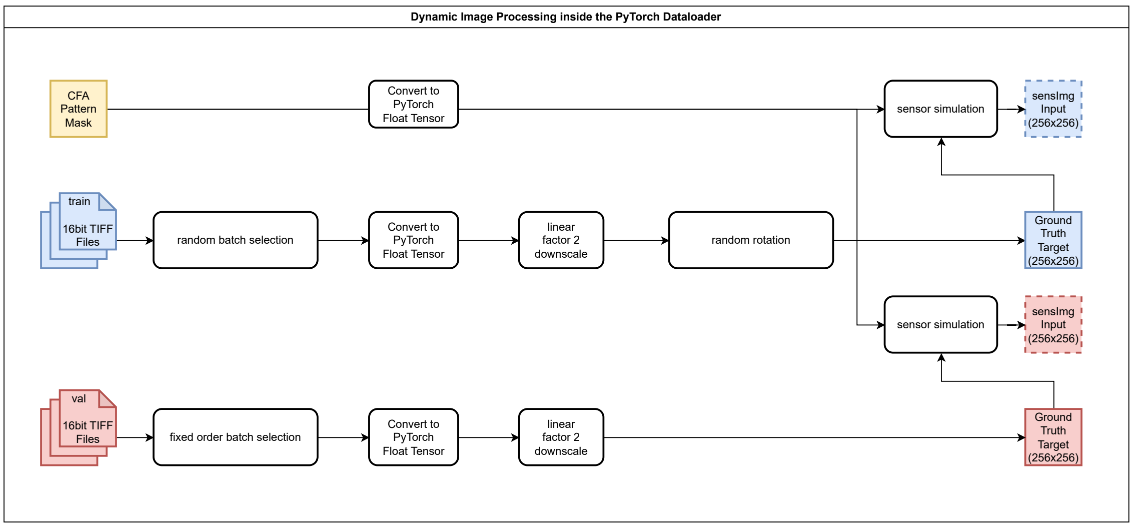
Dynamic Image Processing inside the PyTorch Dataloader
CFA Pattern Mask
Convert to PyTorch Float Tensor
sensor simulation
sensImg Input (256x256)
train
16bit TIFF Files
random batch selection
Convert to PyTorch Float Tensor
linear factor 2 downscale
random rotation
Ground Truth Target (256x256)
sensor simulation
sensImg Input (256x256)
val
16bit TIFF Files
fixed order batch selection
Convert to PyTorch Float Tensor
linear factor 2 downscale
Ground Truth Target (256x256)

Figure 24: Dynamic image processing inside the PyTorch DataLoader. The figure shows how a batch of mosaic sensor images and their corresponding ground truth is loaded during training (blue) and validation phase (red).

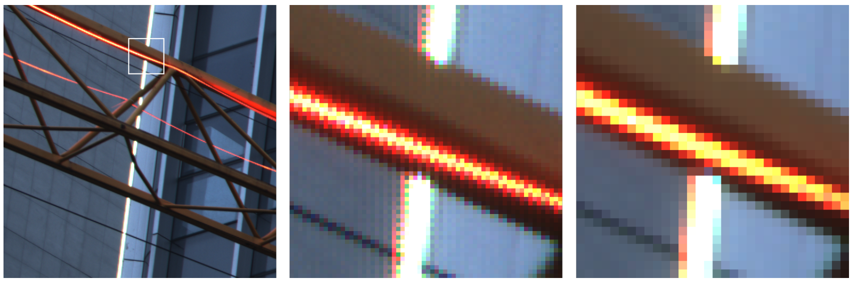

Figure 25: PixelShift: Moving objects cause Bayer artifacts. Training image no. 177 (left) has Bayer artifacts visible in the magnified extract (center). These are resolved by factor 2 linear downsampling (right).

acurately.

The color space utilized in the training image set impacts the evaluation metrics SSIM, MS-SSIM, and PSNR, as well as the loss function. Due to the nonlinear properties of Gamma 2.2, training and evaluation in sRGB gives more weight to the darker intensity values than to the lighter ones, which corresponds to human perception. While the authors of SSIM do not define a specific input color space for the metric, they also employ sRGB encoded images in their examples (see Nilsson and Akenine-Möller, 2020). Preliminary testing also indicated that the range of values of uncorrected RAW images varies greatly, and that the reconstruction errors are weighted differently by the loss function depending on the brightness of the image. In contrast, the sRGB color space, in combination with the previous processing steps of the RAW pipeline (white balance, brightness and contrast adjustment), acts as a normalization of the training data, which is advantageous for training. For this reason, sRGB was chosen as the color domain for training the demosaicing models in this thesis.

The training data is augmented with a random geometric transform that varies for each epoch of training. The primary objective is to ensure transformation-invariant reconstruction quality. For this purpose, random rotation is employed to allow the camera to be oriented in any direction while maintaining the same reconstruction quality. Figure 26 previews how an original training image is transformed differently during four epochs of training. The rotation operation uses mirrored border extension to fill the undefined triangles that result from angles not divisible by 90 degrees. These geometric augmentations contribute to a consistent comprehension of image structures across different non-regular RGB sample constellations. As the CFA remains fixed during training, the random transformation of the underlying ground truth images results in raw image simulations that are differently sampled. This can effectively prevent overfitting and improve the generalization of the trained model.

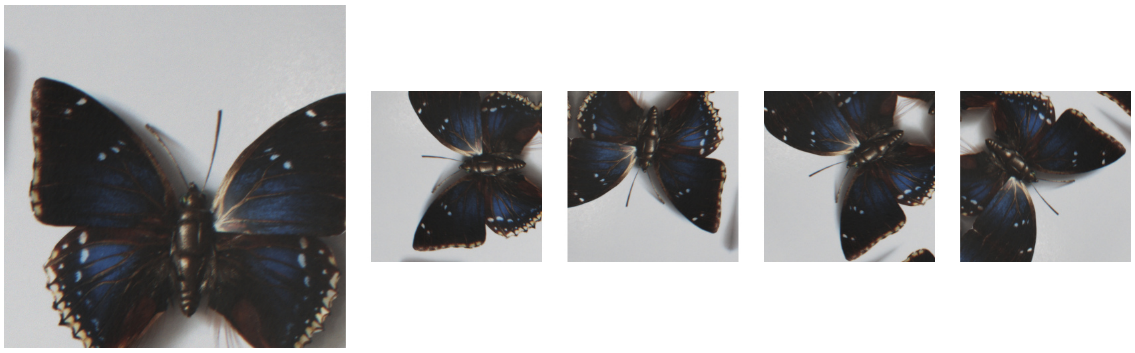

Figure 26: Dynamic geometric augmentation inside the DataLoader: An original 512×512 pixel training sample (left) and four examples of randomly rotation and factor 2 linear downsampling (right).

4.1.3. Choosing Gauss and RandomQuarter as CFA

As mentioned in Subsection 2.2.2, Backes (2019) has investigated different methods to construct the optimal pseudo-stochastic CFA, as shown in Figure 27, and provides the code of the pattern generators implemented in MATLAB. This serves as a starting point for generating the CFA for this thesis. Since the RandomQuarter and the Gauss patterns gave promising results in their experiments, this pattern is chosen for the training of the CNNs.
The RandomQuarter sampling pattern can be obtained by randomly mutating every two-by-two pixel block of the Bayer pattern, while the Gauss pattern is generated by placing single pixels iteratively onto an empty frame using a random gauss distribution. The algorithm will be further discussed in Subsection 4.4.4. In accordance with the Bayer pattern, both RandomQuarter and Gauss contain twice as many green samples, resulting in 25% red, 50% green, and 25% blue to account for the human eye's higher sensitivity to green light.

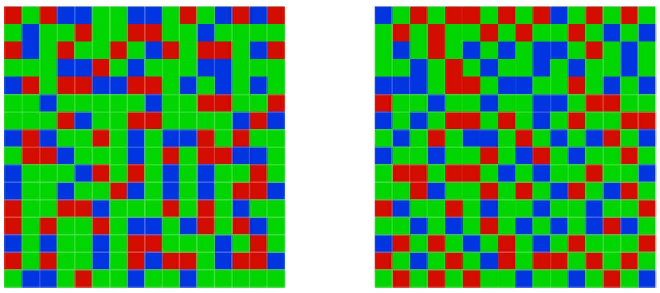

Figure 27: RGB RandomQuarter and Gauss50 Sampling Patterns. Adapted from Backes and Fröhlich, 2020

4.1.4. Simulating Mosaic Sensor Images (CNN Inputs)

As mentioned earlier, when training the CNN with supervised learning a set of labelled data is needed. In case of demosaicing, this means pairs of mosaic sensor input images and their corresponding desired optimal fully sampled RGB outputs. Following the Backes (2019) approach, a set of fully sampled RGB images is first assembled to serve as *true labels* during training and as *ground truth* for the simulated mosaic sensor images.
While demosaicing is a difficult task, simulating a mosaic sensor image from the RGB ground truth is as simple as masking the corresponding pixel positions by the color channels of the CFA-Pattern. This can be achieved by the following element-wise matrix multiplication

$$I_{sensSim} = I_{gt} \times M_{cfa}, \tag{4.1}$$

with $I_{gt} \in \mathbb{R}^{3\times H\times W}$ being the fully sampled RGB ground truth image and $M_{cfa} \in \{0,1\}^{3\times H\times W}$ being the CFA mask image. The resulting mosaic sensor image $I_{sensSim} \in \mathbb{R}^{3\times H\times W}$ still has three RGB-channels with only one of the colors defined per pixel and the others being zero. It can also be flattened to a single channel image by reducing the channel dimension with a sum operation.
As today's digital cameras involve a (Bayer-) color filter array, most image datasets provided by the science community are actually demosaiced with some kind of debayering algorithm. Raw images demosaiced with another debayering algorithm are not appropriate ground truth images as they include just the artifacts that we aim to prevent. The PixelShift technology, as discussed in Subsection 4.1.1, however, provides a good way to recomposite a fully sampled RGB raw image from four raw Bayer images without involving debayering.

4.2. Setup and Implementation of the Convolutional Neural Networks

4.2.1. From UNet to SANet Architecture

Throughout this thesis, different CNN architectures are tested using the training pipeline described above. These include a simple UNet, roughly based on Ronneberger et al. (2015), and the RDUnet which relies on the code provided by Gurrola-Ramos et al. (2021). In addition an own idea of an Adaptive Pattern Interpolation Net (APINet) is prototyped. However, it was only with the SANet architecture that the reconstruction quality met expectations. Therefore, the focus of this thesis is on the customization of the SANet architecture for the reconstruction of irregularly sampled images. However, the experiments with UNet and APINet are referenced at various points for comparison. The architecture

of both models is provided in the appendix.
The PyTorch training loop is initially setup with the help of the guide by Aladdin Persson (2021), which is then extended to allow model switching and evaluation of the desired metrics. The TensorBoard plugin (“TensorBoard”, 2017, May 15/2023) aids in organizing and visualizing the different training runs. A customized logger is implemented to preview the reconstructed validation images, export the model state, and log the evaluation metrics of each training in a JSON file.
For the experiments involving the SANet architecture, specific components of the authors' official PyTorch implementation (T. Zhang, 2022, October 2/2023) are used, including the spatially adaptive convolution modules and the SANet model, but not the training loop or data loaders. All locations where foreign code is used are documented in the source code.

4.2.2. The Appropriate Output Activation Function for Image Reconstruction

The activation function of the last layer should be chosen separately from the inner layers and the output layer of a CNN, as it limits and shapes the values that can be output. For image processing task mostly ReLu Activation and it's derivates PReLu, LeakyReLu are used to prevent the problem of vanishing gradients (cf. Kumar et al., 2023; Zhou et al., 2018). However, at first glance, Sigmoid activation seems to be a good fit for image reconstruction as well (Eisemann et al., 2020, cf.). Mainly designed for classification probabilities, it ensures that the output image values are floating point values between 0 and 1, which can be mapped to the minimal and maximal values of the desired output image format. However, sigmoid was not used as output activation for the following reason: The brightness values of the image reconstruction (CNN output) should remain the same compared to the mosaic sensor image (CNN input). The nonlinearity of the Sigmoid function forces the CNN to generate the output in an inverse Sigmoid domain during training. During initial training experiments with a Sigmoid activation function, the UNet fails to accurately compensate for the s-shape of the Sigmoid activation: Image regions that are nearly black in the input turn out brighter in the reconstruction, with correspondingly worse loss values. A simple ReLu activation seems to be the more obvious choice for the output layer because of its linearity.
The net will then learn through training to output pixel values between 0 and 1, but might predict values outside this range. Therefore, an additional clamping operation is applied to the CNNs output just before saving the reconstruction to an image file and before measuring PSNR and SSIM. No clamping is applied before the loss-function during training to not disturb the back propagation process.

4.2.3. Choosing L^{l_2} Loss Function for Image Reconstruction

Two commonly used loss functions in image reconstruction nets are L^{l_1} and L^{l_2} loss. L^{l_2} loss is also referred to as mean squared error (MSE) or Euclidean loss (see Khan et al., 2018, p. 67). It is utilized as a loss function for demosaicing e.g. in Cui et al. (2021), Gharbi et al. (2016), Qian et al. (2021), and Syu et al. (2018). L^{l_2} is closely connected to the peak signal to noise ratio (PSNR) metric described in Subsection 4.3.1
The L^{l_1} loss is the mean absolute error, which is also employed for demosaicing, as seen in T. Zhang et al. (2022). According to a study on various loss functions by Zhao et al. (2018), the L^{l_1} loss performs better in image restoration compared to the L^{l_1} loss function because of it's stronger correlation with human-perceived image quality. For this reason, L^{l_1} was selected for the experiments carried out in this thesis.
To measure the error of the predicted Image $I_{pred} \in \mathbb{R}^{C \times H \times W}$, L^{l_1} loss can be computed with

$$L^{l_1}(I_{pred}, I_{gt}) = \frac{1}{C \cdot H \cdot W} \sum_{c=0}^{C-1} \sum_{j=0}^{H-1} \sum_{i=0}^{W-1} \left| I_{pred}\left[c, j, i\right] - I_{gt}\left[c, j, i\right] \right|. \tag{4.2}$$

The summed absolute pixel error is normalized with the total number of pixels $H \cdot W \cdot C$ across all image channels, with $C = 3$ for an RGB color image.
For achieving better perceived reconstruction quality of detailed structures, a mixed loss function consisting of Multiscale structural similarity index measure (MS-SSIM), Structural similarity index measure (SSIM), and L^{l_1} loss is proposed by Zhao et al. (2018). However, initial efforts to use various PyTorch implementations of this mixed loss function did not bring the desired success: The duration of a training epoch was four times longer than with L^{l_1}. Focussing on the research questions, it appeared more beneficial to stick to the simpler L^{l_1} loss for training and keep SSIM and MS-SSIM only as separate metrics for independent evaluation of the results, as described in Section 4.3.

4.2.4. Input Layer Adjustments for Processing Image and Pattern Information

The mosaic sensor image is a single channel, grayscale image holding spatially encoded color values. Passing this directly into the network during the training of the network would involve guessing the locations of the red, green and blue pixels. While this is theoretically possible it would not make sense to complicate training to learn something that is already known. It would be unnecessary to let the network learn the positional color encoding of the mosaic sensor image by itself as the design of the CFA is a known parameter. For sake of efficiency all relevant prior knowledge should be provided to the network as an input.
Therefore, the mosaic sensor image pixels are separated into a three-channeled RGB image first. As each pixel position only has a single value for either red, green or blue, the other

color channels contain zeroes where there is no information.
While this input format is already an improvement in contrast to the raw image, it introduces a new ambiguity: A channel value of zero could be either an extremely dark color value or denoting a missing pixel in this encoding.
To avoid this ambiguity, the pattern information is input as a dedicated three-channeled bit mask. The channels correspond to the three colors red, green and blue and each bit denotes if there is information on this pixel (value of one) or not (value of zero). T. Zhang et al. (2022) make use of the exact same input format. Providing the CFA as an additional input also allows processing the pattern information in a separate chain inside the network like with the SANet architecture (see Subsection 3.3.4).

4.2.5. Separate Experiments for Single- vs. Multi-Channel Reconstruction

In order to find out how the different architectures are able to leverage inter-channel correlations the training and evaluation is repeated in three different setups: $g{\to}g$, $rgb{\to}g$ and $rgb{\to}rgb$.
In the first setup the problem is simplified to only reconstructing a single color channel from its available green color samples. This stage will be called $g{\to}g$. The green channel was chosen as it has the double amount of samples: It only lacks fifty percent of the pixel data whereas red and green channel lack 75% of their pixel data in the CFA patterns used in this thesis.
The second setup will be referred to as $rgb{\to}g$ where still only the green channel gets reconstructed but the red and blue channels are fed into the CNN as well giving additional context. It is assumed that the reconstruction quality will improve if the CNN learns how to extract cross-channel correlations from the other channels to contribute to the estimation of the missing pixel values in the green channel.
The third setup $rgb{\to}rgb$ is to reconstruct the full rgb channels from the rgb input.
With regard to their architecture of the networks, the three setups differ mainly in the number of input and output features. The number of input feature maps equals twice the number of color channels entered. Each color channel corresponds to two input feature maps – one for the actual image data and one for the CFA mask of this channel. Therefore, $g{\to}g$ has two input feature maps while $rgb{\to}g$ and $rgb{\to}rgb$ both have six input feature maps.

4.3. Reconstruction Quality and Model Complexity Metrics

To evaluate the quality of demosaicing, the reconstructed image is compared to the ground truth target utilizing the image comparison metrics PSNR, SSIM, and MS-SSIM. Model complexity is judged by the number of trainable parameters. This section provides a brief explanation of these metrics.

4.3.1. Peak Signal to Noise Ratio (PSNR)

The PSNR between the predicted Image $I_{pred} \in \mathbb{R}^{C\times H\times W}$, and the ground truth image $I_{gt} \in \mathbb{R}^{C\times H\times W}$ can be defined as

$$q_{PSNR} = 10\log_{10}\left(\frac{maxval^2}{MSE}\right), \tag{4.3}$$

where mean squared error (MSE) is computed with

$$q_{MSE}(I_{pred}, I_{gt}) = \frac{1}{C\cdot H\cdot W}\sum_{c=0}^{C-1}\sum_{y=0}^{H-1}\sum_{x=0}^{W-1}\left(I_{pred}\left[c,y,x\right] - I_{gt}\left[c,y,x\right]\right)^2. \tag{4.4}$$

(see Kumar et al., 2023)
In our case, $maxval$ is $= 1.0$, since that corresponds to the maximal possible pixel intensity value. Clipping is applied before computing the evaluation metrics, as mentioned in Subsection 4.2.2.
PSNR values are measured on a decibel scale, where higher values indicate higher quality image reconstruction.

4.3.2. Structural Image Comparison with SSIM & MS-SSIM

The structural similarity index measure (SSIM) metric is used in thousands of research papers to objectively compare image quality. It aims to be closely related to human image perception and takes into account the luminance, contrast, and structure of the images when comparing two images (see Nilsson and Akenine-Möller, 2020). Comparable to PSNR, the overall quality score of two images is determined by averaging the SSIM scores computed for every individual pixel position. However, SSIM also considers the neighboring pixels by internally applying an 11×11 pixel Gaussian filter kernel with $\sigma = 1.5$ within the components. For a full definition of SSIM, refer to Nilsson and Akenine-Möller (2020).
The multiscale structural similarity index measure (MS-SSIM) extends SSIM by accounting for various levels of detail. Although SSIM permits adjusting parameters to emphasize different levels of detail, MS-SSIM, in conjunction with the standard scaling weights presented by Wang et al. (2003), seems to be a ready-made solution for many applications. MS-SSIM provides a comprehensive assessment of the perceived quality of the reconstruction by integrating SSIM at multiple levels of detail (see Nilsson and Akenine-Möller, 2020). Both SSIM and MS-SSIM yield scores that range from 0 (lowest similarity) to 1

(highest similarity).
For the experiments in this thesis the MS-SSIM implementation from the Python *piqa* package was used together with the default scale weights described by Wang et al. (2003).

4.3.3. Number of Trained Parameters

The total number of trained parameters provides an estimate of the neural network's complexity. A greater quantity of parameters suggests a more computationally demanding training process. It also indicates that more floating-point operations may be involved in the forward pass, which means that the CNN will require more hardware resources during inference or that reconstruction will take longer.

4.4. Training and Optimization of the SANet Architecture

This section describes various optimizations to the training pipeline and the architecture of the SANet. Here, the training performance will be assessed only by the validation loss, while the Chapter 5 will focus on image quality comparision using PSNR and MS-SSIM.

4.4.1. Increasing the Size of the Training Set

Experiments were performed with two different sizes of the data set. Since the images used for training are small extracts from the original PixelShift200 images, the size of the data set can be increased by the number of extracts. For the smaller training set, only one cropped area from the center of each of the original 200 images was used. For the larger one, five clippings were extracted from each image: One from the center and four from the corners. Here, the resulting size is $5 \cdot 200 = 1000$ training images. In both cases, the same augmentations were applied. The validation set was always left at five crops per image to allow comparison. It consists of 50 images.
As it can be seen in Figure 28, the larger size leads to a much faster convergence per epoch. However, it has to be taken into account that epochs become longer as the number of weight updates increases with the size of the dataset. Therefore, Figure 29 compares the loss per elapsed training time. After compensating for the different epoch length there is no significant difference between the larger and the smaller training set.
A possible explanation could be that demosaicing is more about the very basic image features like gradients, edges or basic shapes at the scale of a few pixels, as this is where the undersampling errors occur. This could mean that the relevant features were already sufficiently represented in the smaller training set. Also, the geometric augmentation inside the train DataLoader could influence the result, as it already enhances the training data. It was however applied in both training runs.

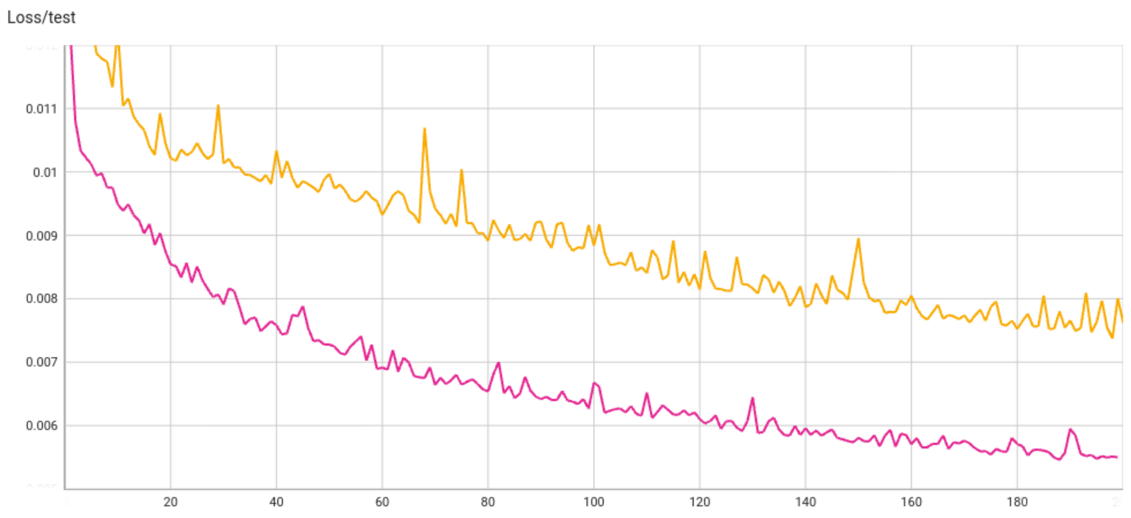

Figure 28: Validation loss per epoch for SANet with large training set and small training set

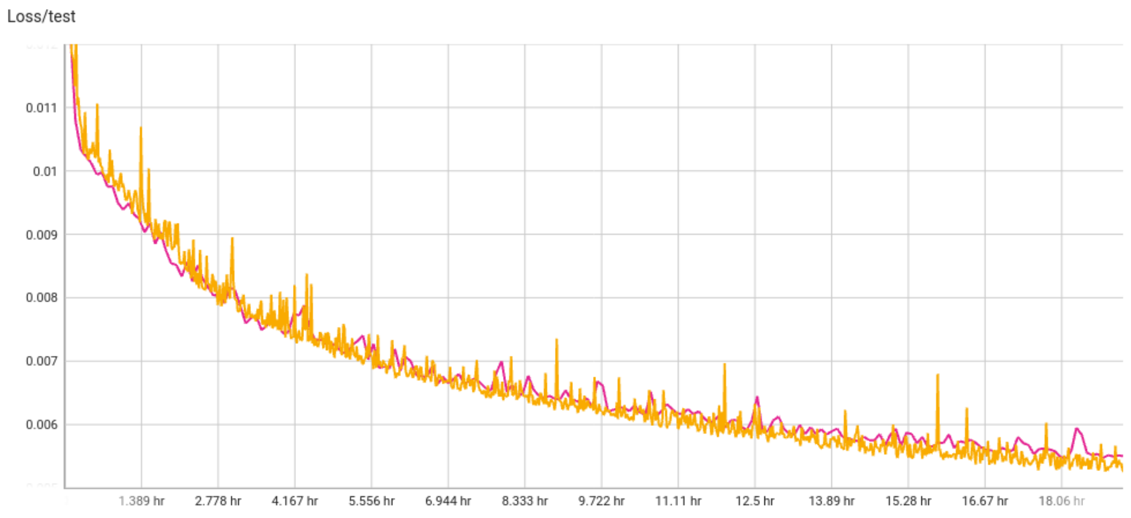

Figure 29: Validation loss per training time for SANet with large trainingset and small trainingset

To find out the optimal size of the image data set, ideally a series of training runs would be conducted with e.g. linearly increasing size of the data set. Plotting the validation loss against the size of the data set could reveil if there exists an optimal size and how much the dataset size can be reduced without decrease of reconstruction performance.

4.4.2. Learning Rate Decay

The original SANet implementation (T. Zhang, 2022, October 2/2023) use a global learning rate multiplier to employ a learning rate warmup and decay scheduled by the training epoch. Figure 30 shows, that the maximum learning rate of $1 \cdot 10^{-4}$ is reached after three epochs of warmup. From then, a learning rate decay is achieved by a cosine annealing function. The warmup is probably intended to prevent overshooting in the initial epochs, while the decay is probably intended to allow the backpropagation descend into smaller minima to achieve a better model fit.
To evaluate the effect of the scheduled learning rate multiplier, the validation loss per epoch is compared for two models of SANet, as shown in Figure 31. One is trained with scheduled learning rate (green) and the other one is trained with a constant learning rate of $1 \cdot 10^{-4}$ (orange). The constant learning rate acheived better results in the validation loss. The same difference shows in the metrics PSNR and MS-SSIM.
The Adam stochastic gradient descent optimizer used in the experiments already dynamically adjusts the learning rate based on gradient updates (Kingma and Ba, 2017). Therefore, it could be argued, that an additional learning rate multiplier redundant. However, according to Loshchilov and Hutter (2019), an additional global learning rate multiplier, scheduled by the epoch, such as cosine annealing, can improve the training success of the Adam algorithm even further. It is however usually implemented as a periodic function that alternates between higher and lower learning rates during training. In this context it would be interesting to spend more investigations on learning rate

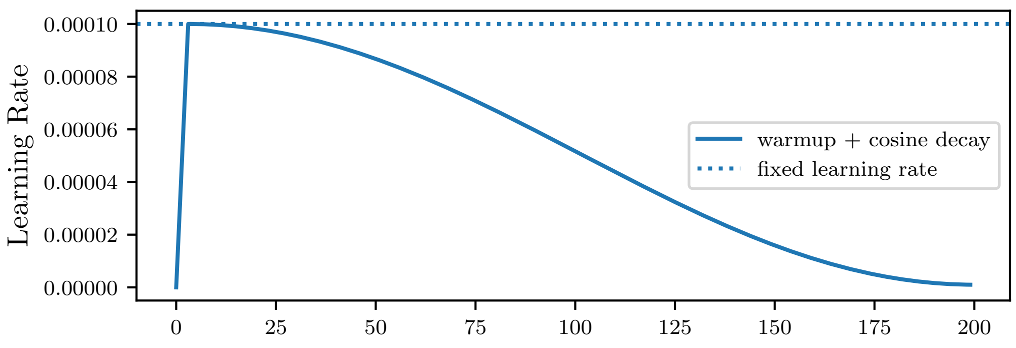

Figure 30: Scheduled Learning Rate Multiplicator

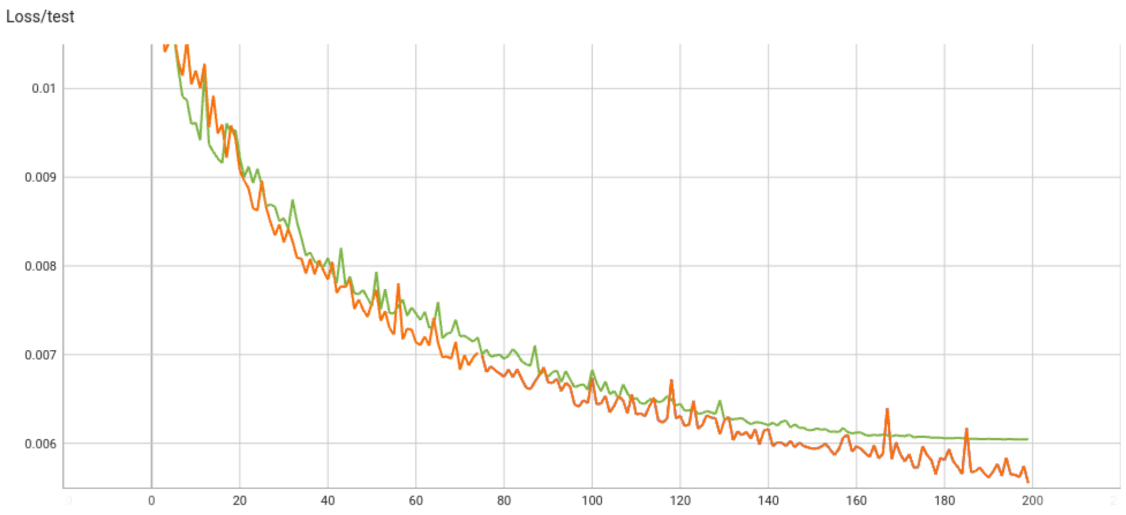

Figure 31: Constant vs. scheduled learning rate multiplier

schedulers and e.g repeat the experiments with an iterative learning rate decay and cosine annealing.

4.4.3. Sparse Residual- and Difference-Learning

The SANet architecture uses global residual learning to increase the efficiency of training or the quality of reconstruction: In a separate branch, a single spatially adaptive convolutional layer is applied to the mosaic input to produce an initial rough full RGB reconstruction (cf. T. Zhang et al., 2022). Adding this to the output of the UNet means that the UNet is trained to predict only the difference between the ground truth and this partial reconstruction. However, the fact that a single adaptive convolution layer can perform a coarse debayering task seems to be true only for the Bayer-CFA for which SANet was originally developed. For irregular patterns such as the random quarter or Gaussian patterns of Backes and Fröhlich (2020), a single adaptive convolution layer seems not able to get a clear separation of the RGB channels. While this original global residual learning design of T. Zhang et al. (2022) still performs better than without residuals, the following experiments show that the reconstruction result can be greatly improved with an alternative residual branch tailored to the nature of irregular CFAs.
As a replacement for the global residual method by T. Zhang et al. (2022), this thesis proposes an alternative residual branch consisting of a CFA-dependent Gaussian filter that works better on irregularly sampled sensor data. Its basic idea is to close the gaps of missing pixel values in the color channels by applying a Gaussian blur filter per color channel. The smoothed result represents a rough reconstruction of the image, which is to be refined by the Encoder-Decoder structure of the SANet. To prevent variations in

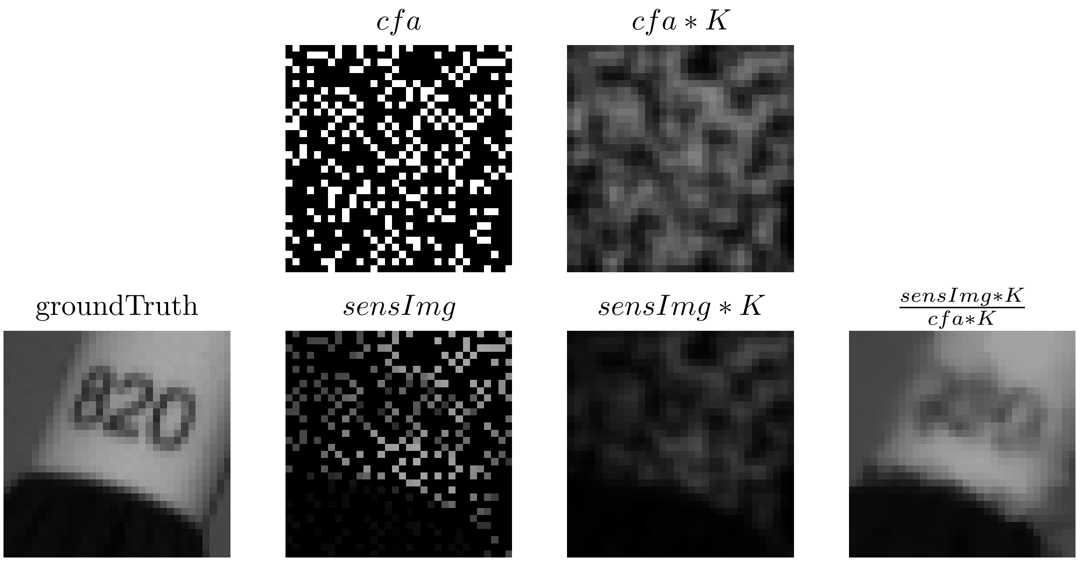

Figure 32: CFA-normalized convolution as residual branch

sample density from affecting the result as local brightness variations, a CFA-weighted sample-averaging is employed. It is achieved via pixel-by-pixel division by the CFA filtered with the same kernel.
A kernel size of 7×7 pixels was chosen, since this corresponds to the maximum sample distance in the stochastic Gaussian CFA. In this way, during convolution, it is ensured that for each kernel position within the receptive field, at least one sample of each channel occurs. This avoids undefined pixel values (holes) in the output, which lead to problems with the described CFA-dependent normalization method (divide by zero). The Gaussian filter kernel is generated with the function 'transforms.functional.gaussian_blur' from the *torchvision* package and set as static, meaning it is not part of the trainable weights of the CNN. The standard deviation σ of the Gaussian distribution is set to 1.0, keeping the blur effect at a minimum. The neighboring pixels of the central pixel in the receptive field have only a minor impact on the convolution output. However, when there are no nearby samples, more distant neighbors receive more weight due to the CFA-dependent normalization process.
The result $\mathbf{I}^O \in \mathbb{R}^{H \times W}$ of the simple convolution $\mathbf{I}^I * \mathcal{H}$ without normalization can now be described as

$$\mathbf{I}^O[u,v] = \sum_{i=0}^{K-1} \sum_{j=0}^{K-1} \mathbf{I}^I[u+i, v+j] \cdot \mathcal{H}[i,j], \tag{4.5}$$

where the convolution kernel of size $K \times K$ is defined as $\mathcal{H} \in \mathbb{R}^{K \times K}$. This kernel is applied to the input $I \in \mathbb{R}^{H \times W}$ in a sliding window. (cf. Burger and Burge, 2022, p. 96)
In order to preserve the brightness of the original image after applying a convolution filter,

the result is typically divided by the sum of the kernel, or a normalized kernel with sum = 1 is used in the first place. When dealing with irregularly sampled color channels a spatially adaptive normalization per pixel is required, as the total weight of the kernel at convolution depends on the available samples in the CFA depending on the position.
However, for irregularly sampled color channels, normalization must be spatially adaptive per pixel, since the total weight of the kernel at convolution depends on the particular sample constellation within the sliding window. This type of Gaussian-weighted averaging avoids local accumulations of samples manifesting as brightness variation artifacts in the resulting image.
The effective total weight of the kernel for each pixel position can be calculated efficiently by applying the same convolution to the CFA bit-mask of a color channel.
The result $\mathbf{I}_c^O[p]$ of a spatially normalized convolution of an image channel c of the sensor image $\mathbf{sensImg} \in \mathbb{R}^{H\times W\times C}$ and the CFA bit-mask $\mathbf{CFAmask} \in \{0,1\}^{H\times W\times C}$ with the fixed kernel $\mathcal{H}$ can be described as

$$\mathbf{I}_c^O[p] = \frac{\sum_{i=0}^{K-1}\sum_{j=0}^{K-1}\mathbf{sensImg}_c\left[u+i,v+j\right]\cdot\mathcal{H}\left[i,j\right]}{\sum_{i=0}^{K-1}\sum_{j=0}^{K-1}\mathbf{CFAmask}_c\left[u+i,v+j\right]\cdot\mathcal{H}\left[i,j\right]}. \tag{4.6}$$

Figure 33 compares the validation loss per epoch of training for the original residual branch (gray), no residual branch (green), and the alternative residual branch (orange). The effectiveness of the alternative branch is clearly superior to the other two variants. Especially during the first epochs of training it can be seen that convergence is much faster. By utilizing the alternative residual, SANet reaches the same validation loss after about 60 epochs, that the original residual branch reaches after 200 epochs of training.

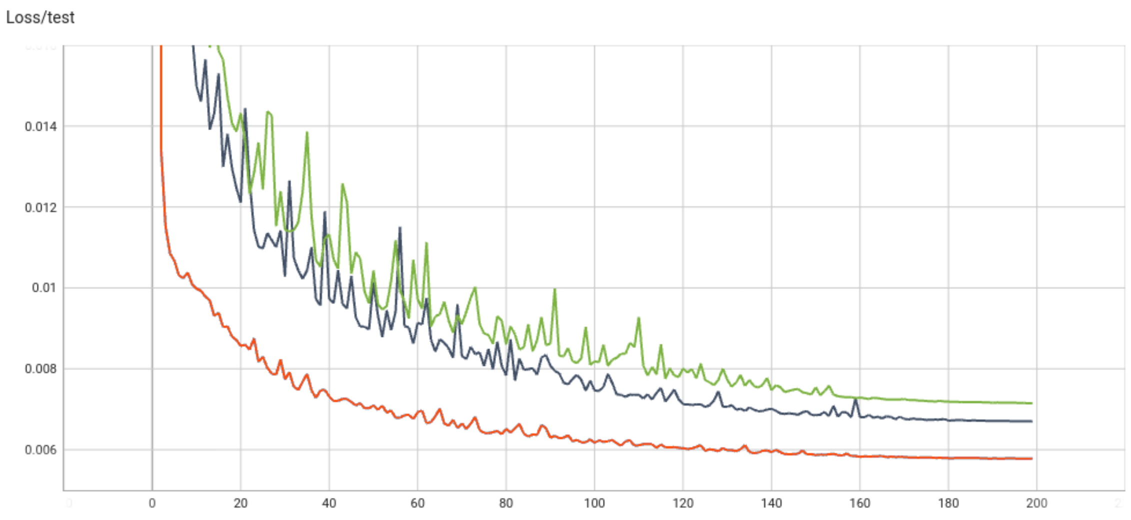

Figure 33: Validation loss of no residual branch (green), original residual branch (gray), alternative residual branch (orange)

4.4.4. Using a Periodic Stochastic Gauss Pattern

The idea for this optimization is, that the full 256×256 pixel stochastically generated Gauss kernel is harder to demosaic for SANet than a smaller 32×32 pixel Gauss kernel that is repeated to fill the 256×256 image. In this section, the Gauss pattern generation algorithm is first adjusted to produce a seamlessly tileable subpattern. Then the SANet is trained with both variants to evaluate the impact on the reconstruction performance.
In the implementation by Backes (2019), smaller Gauss CFA pattern of 128×128 pixels was generated because the iterative generation process takes very long. The generated 128×128 pattern was then repeated in x and y direction to fill the simulated image size 256×256 pixels of their experiments. In regard of the image size of actual image sensors and the manufacturing process it makes sense to have a repeated CFA pattern at this larger scale.
However, the pattern generation code by Backes (2019) does not take seamless tileability of the pattern into account, as the Gaussian probability distribution constraint does not wrap around the edges. This leads to clearly visible accumulation of same colored samples at the seams of the adjacent tile borders, as shown in Figure 34 on the left.
For generating the Gauss sampling, Backes (2019, p. 34) places new color samples iteratively on an empty grid until every pixel position is filled. He uses a (pseudo-)distance function,

$$d = \frac{(x - x_i)^2 + (y - y_i)^2}{4}, \tag{4.7}$$

to calculate how far a new color sample position is away from all existing same-colored

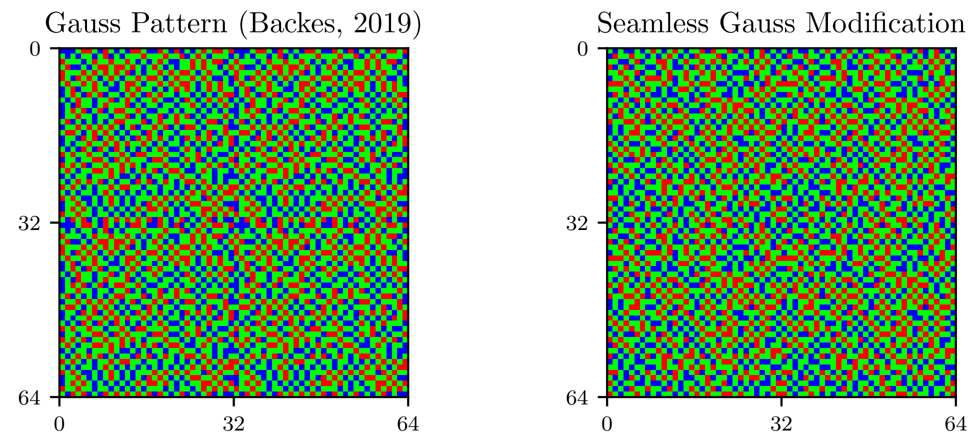

Figure 34: Tiled examples of generated 32 × 32 RGB Gauss patterns. With the original implementation by Backes (2019) (left), the horizontal and vertical seams stand out because of their inhomogeneous distribution. The proposed modification (right) ensures an even distribution across the seams at 32 pixels (Please zoom in for a pixel-perfect view).

samples. The distance function is part of a Gaussian probability distribution, that influences the placement of the next color sample.
To receive a seamlessly tileable CFA pattern, the distance function was altered to

$$d = \frac{min\left(|x - x_i|\,, w - |x - x_i|\right)^2 + min\left(|y - y_i|\,, h - |y - y_i|\right)^2}{4}. \tag{4.8}$$

For this, the original MATLAB implementation was ported to Python. Figure 34 shows two examples of a 32 × 32 pixel Gauss pattern both repeated in x- and y-direction to cover a 64 × 64 CFA. On the left, the original algorithm implemented by Backes (2019) leads to non-uniform clustering of same-colored samples at the borders, visible as horizontal and vertical seams at positions 32. In addition, the border regions of the tile seem to deviate from the desired proportions of 50% green, 25% red, and 25% blue pixels. They tend to have fewer green pixels. In the right image, thanks to a modified distance function, no seams are visible between the repeated tiles, as the Gaussian distribution wraps around the edges.
Figure 35 shows that the 32 × 32 pixel periodic Gauss pattern perform leads to a smaller validation loss (gray) than the full 256 × 256 Gauss pattern. This could be explained with the reduced complexity of the sampling pattern, as it repeats many times to fill the image size of 256 × 256. It can be assumed that the effect would be even greater for larger input images.

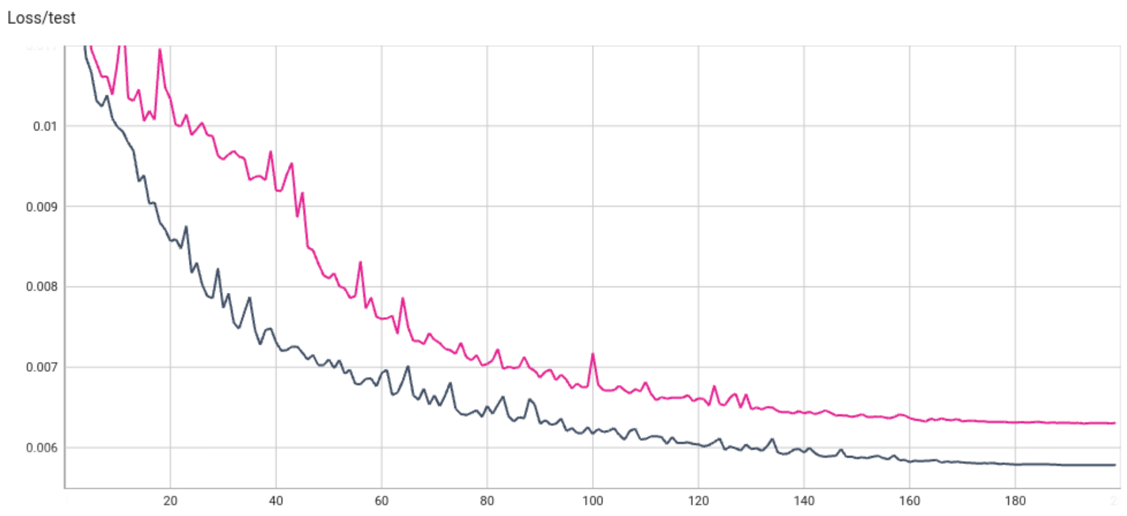

Figure 35: SANet validation loss per training epoch for the original Gauss pattern (pink) and the periodic Gauss32 seamless pattern

5. Results

5.1. Comparision of the Base Architectures

Architectures	UNet	SANet	APINet
PSNR	34.55	**40.20**	34.77
MS-SSIM	0.9754	**0.9954**	0.9866
Params(M)	1926531	10783810	**19779**

Table 2: Full RGB reconstructions from UNet, SANet, and APINet (RandomQuarter CFA; 100 epochs of training)

5.1.1. Reconstruction Quality

Table 2 shows a comparison of three CNN base architectures investigated in this thesis. As the goal of this comparison is to point out the most appropriate base architecture, they are trained in their most basic form without further optimizations.
The comparison features the SANet architecture (see Subsection 3.3.4) before applying the optimizations presented in this thesis. Secondly, a basic UNet architecture is listed, which is included in the attachments of this thesis. Further, the APINet prototype is added to the comparison, which was newly developed during this thesis. It can also be found in the attachments. For this comparison, APINet is applied to the three channels independently without further heuristical optimizations such as green difference reconstruction.
The differences of the MS-SSIM may not seem huge when considering the value range of MS-SSIM, that is from 0 to 1. However, the difference is clearly visible in the reconstruction images (see Figure 36). Given demosaicing is about the recovery of 66% missing pixel values while 33% are already there, higher metric values don't come as a surprise. Furthermore, image structures, that are challenging to demosaic, only appear in parts of the images, while huge parts of the images are easier to reconstruct, e.g. because they are blurred or lack tiny structures. Therefore, MS-SSIM value differences above 0.9 are where the crucial improvements in reconstruction quality take place.
The UNet model produces acceptable results for grayscale images. However, as shown in Figure 36, in areas with minimal saturation, the red, green, and blue channels of an image become identical, making demosaicing an easy task. The example of a saturated image

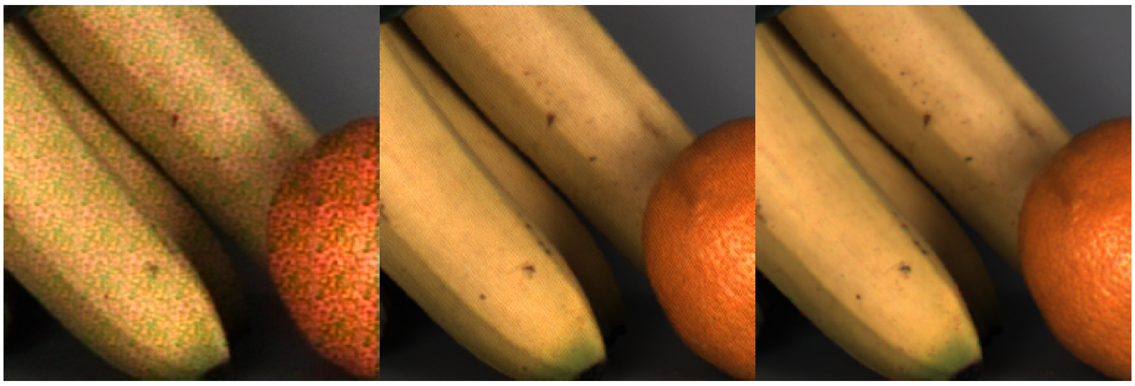

Figure 36: Selected Reconstructions from UNet, SANet, and APINet

shows that the UNet does not separate the colors in the details sufficiently, resulting in a coarse color noise that seems to be the sampling pattern manifested in the output.
APINet produces clear and highly detailed images at first glance. However, slight pixel distortions and directed blurs at the pixel level, that are not noticeable in undefined or blurred areas, lead to visible false colors in sharp boundaries and color transitions. This can be explained with APINet not considering the image content, but only the sample constellations for applying the interpolation filters.
The basic SANet model provides a much better reconstruction than APINet. This may be thanks to the UNet shape it combines with spatially adaptive convolution, which allows it to recognize structures even though they are masked by an irregular sampling pattern. Nevertheless, the color separation and detail reconstruction of the original SANet remain suboptimal. However, the modifications and optimizations presented in this thesis significantly improve the demosaicing quality, which are summarized in Subsection 5.1.3.

5.1.2. Model Complexity

As shown in Table 2, SANet has five times the number of trainable parameters of the simple UNet tested in this thesis, which can be justified by its significant quality advantage. The very light APINet questions the suitability of the simple UNet model since it produces significantly better results with over one hundred times fewer trainable parameters. Compared to SANet, it differs by a factor of a thousand. The memory consumption of the forward pass was not measured. However, the difference in memory usage can be expected to be somewhat smaller, since the Encoder-Decoder architecture of UNet and SANet efficiently reduces the spatial feature resolution. On the other hand, APINet operates with feature maps in full resolution, although only one layer deep.
APINet, described in Appendix A, could maybe be adopted in environments where hardware resources are limited, like for in-camera preview. It could be further simplified

for this purpose to allow real-time image processing.

5.1.3. Evaluate SANet Optimizations

Optimizations	Cases		
	1	2	3
SANet	✓	✓	✓
Alternative Residual	✓	✓	✓
Tiled CFA		✓	✓
Extra Long Training			✓
PSNR	40.03	40.56	42.04
MS_SSIM	0.9951	0.9960	0.9969
Trainable Parameters	10783131	10783131	10783131

Table 3: Performance of SANet with different optimizations applied.

As shown in Table 3 the optimizations enhance the performance of the SANet for demosaicing non-regularly sampled images.

5.1.4. RandomQuarter vs. Gauss vs. Periodic Gauss Pattern

Architectures	RandomQuarter	Gauss	Gauss32 seamless
PSNR	**40.71**	40.03	40.56
MS-SSIM	**0.9961**	0.9951	0.9960
Params(M)	10783131	10783131	10783131

Table 4: Comparision of the CFA patterns RandomQuarter, Gauss and Gauss32 seamless (SANet; 200 epochs of training).

Table 4 shows the SANet trained three times with only the CFA pattern changed. RandomQuarter appears to perform best with SANet. As expected the periodic Gauss32 pattern is superior to the original Gauss pattern. It is also very close to RandomQuarter when regarding MS-SSIM, but for the PSNR values the distance is greater in relation to the original Gauss pattern.
The result could again be explained with the complexity of the patterns. RandomQuarter consists of red-green-green-blue mutations in each 2×2 block. As there is a limited number of such mutations, they recur many times in the CFA. This probably allows the SANet to use the same filter for recurring subpatterns. Gaus32 recurs at a 32×32 pixel level thus is more complex to process. The original Gauss pattern has the highest complexity and is hardest to demosaic.

5.2. Benchmarking SANet vs. dFSR

To benchmark the performance of the CNN architectures studied in this thesis, their results are compared with the dFSR universal demosaicing algorithm. For this purpose, the MATLAB code provided by Backes and Fröhlich (2020) is integrated into the PyTorch training and evaluating pipeline implemented for this thesis. To be able to use the same image loaders and evaluation metrics as for the other CNN models, a fake PyTorch 'torch.nn.model' class is created that calls the dFSR script from within it's 'model.forward()' method using the MATLAB Engine for Python. Of course, this model is not actually trainable as it is just a dummy to be able to use the same image loaders and evaluation metrics as for the other CNN models.

5.2.1. Optimized SANet vs. dFSR

Metrics	Cases	
	dFSR	SANet
PSNR	41.38	**42.04**
MS_SSIM	0.9950	**0.9969**

Table 5: Optimized SANet vs. dFSR with periodic Gauss32 CFA. Evaluated using the PixelShift200 validation set.

The optimized SANet manages to surpass the dFSR algorithm in both PSNR and MS-SSIM, as indicated in **??**. The periodic Gauss32 pattern is used.However, it has to be pointed out, that these are the average metrics across all images in the validation set. The next section shows that this holds not necessarily true for all validation images.

5.2.2. Comparison of PSNR and MS-SSIM per Image

Figure 37 shows the metrics PSNR and MS-SSIM per validation image for the best optimized SANet model and the dFSR.
Figure 38 depicts selected validation image no. 14, 15, 06 from dFSR (left), SANet (middle), Ground Truth (right). These images were selected, as they contain detailed structures like letters as well as colored objects. By zooming in, subtle differences can be seen. dFSR produces a sharper overall image impression, while it also produces false colored blurred splashes at the white and black letters, as well as dark structures on the orange. SANet produces an overall smoother output and much cleaner and sharper letters- However at the chinese letter the edge from black to white appears a bit noisy. One important difference is, that dFSR took about two minutes to reconstruct a single 256×256 pixel validation image, while SANet produces a batch of four in under a second on a GTX 1050 Ti.

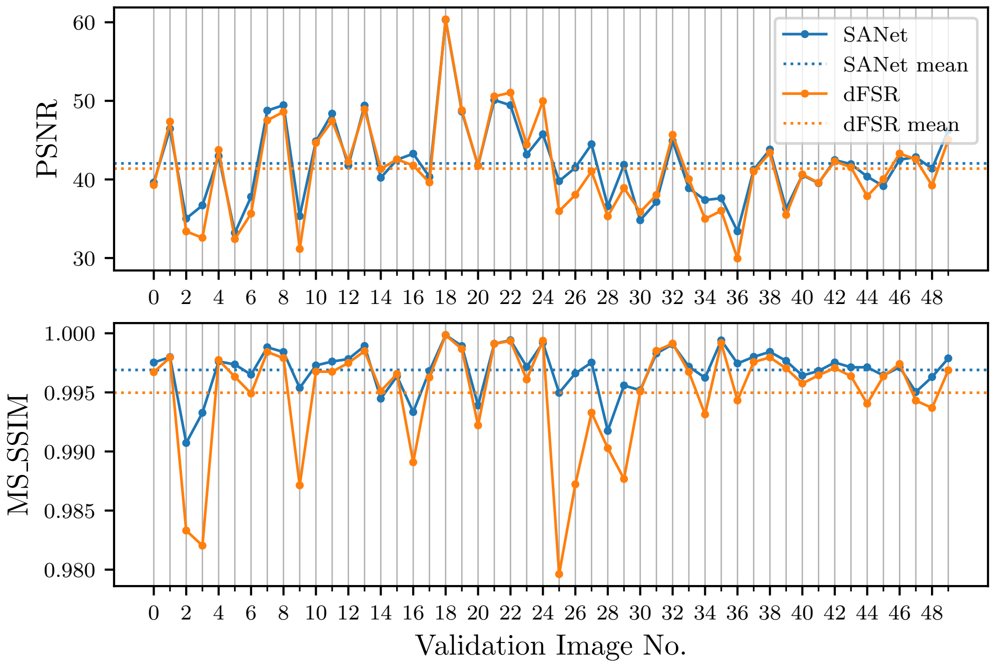

Figure 37: dFSR vs. SANet seamless Gauss32 evaluation per image

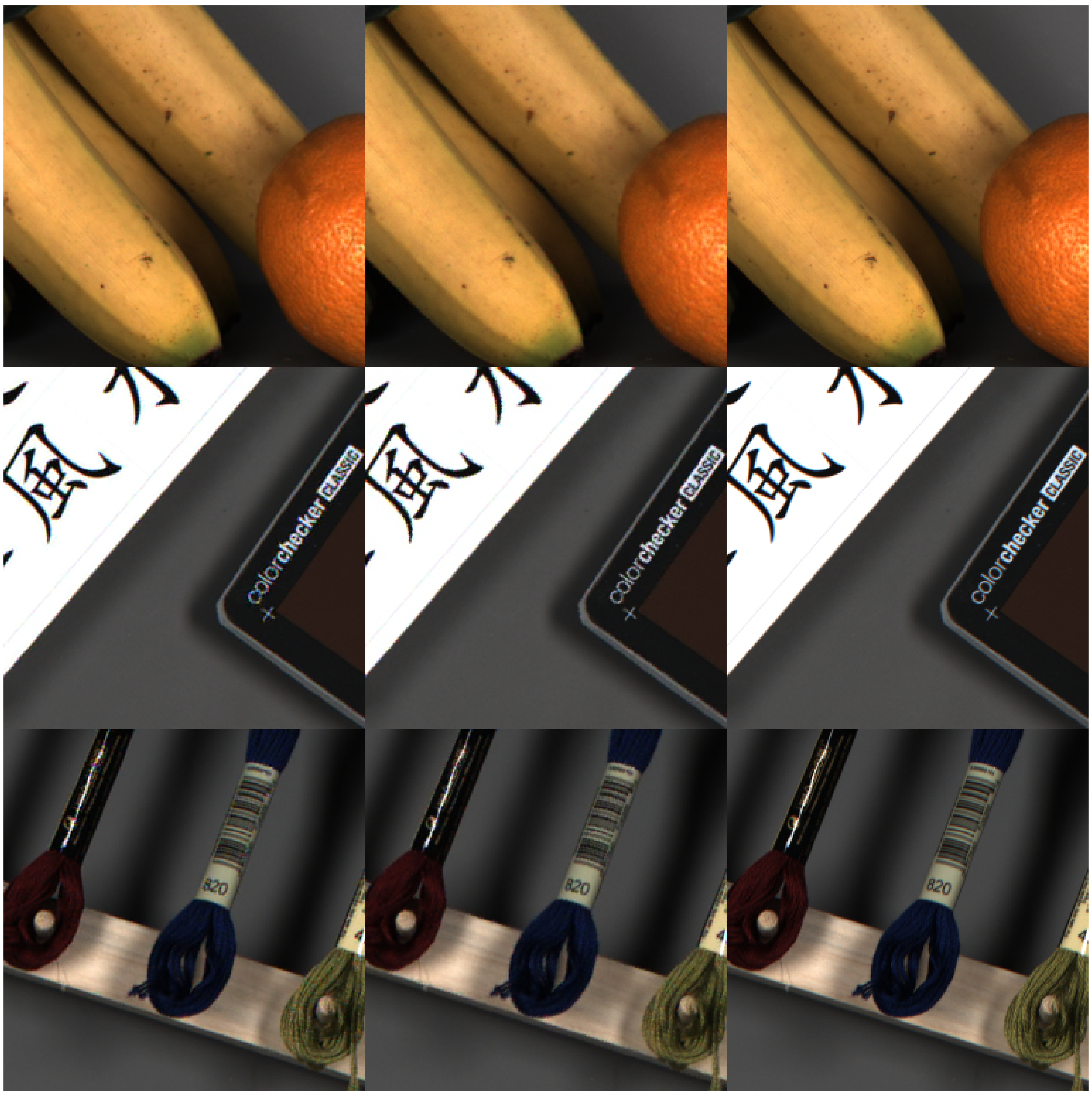

Figure 38: Comparision of the validation images no. 14, 15, 06 for dFSR (left), SANet (middle), ground truth (right)

5.2.3. Evaluate how Green Channel Benefits from the Other Channels

For this experiment, SANet is trained and evaluated twice to exclusively reconstruct the green channel. First, only the green sensor samples are provided as input. For the second model, the input comprises the complete red, green, and blue mosaic image samples. Figure 39 demonstrates that the PSNR and MS-SSIM for the green channel improve when incorporating red and blue sensor data. This demonstrates that the model utilizes cross-channel correlations to improve the green channel, providing an advantage over the dFSR algorithm, which only heuristically enhances the red and blue channels. It is reasonable to assume that in SANet, the green sensor data also contributes to the reconstruction of the red and blue channels.

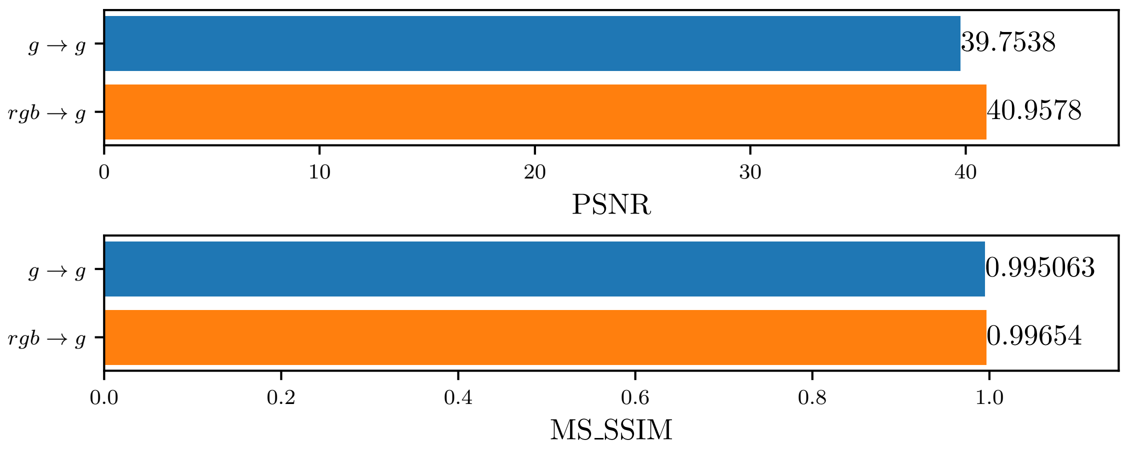

Figure 39: SANet vs. dFSR – Is the green channel by red and blue?

6. Conclusions

6.1. Summary

This thesis combines the fields of RAW image processing and machine learning. Chapter 2 introduces technical details about digital camera sensors, RAW images, color filter arrays, and demosaicing algorithms. Chapter 3 presents the necessary basis of convolutional neural networks, including the concepts of convolution, supervised training, and network architectures. The description of a supervised training pipeline's setup and implementation is presented in Chapter 4. An image training set is created by simulating mosaic sensor images from the Pixelshift200 dataset. Different experiments are carried out, which cover a straightforward UNet architecture, a personally designed APINet, and the spatially adaptive SANet viewed as the most successful. As opposed to the basic UNet's classical convolution, spatially adaptive convolution is identified as being more appropriate for non-uniform samples. Additionally, this thesis presents a new residual branch to improve the demosaicing performance of the SANet on irregularly sampled images. Moreover, two pseudorandom sampling patterns, Gauss and RandomQuarter, are investigated and a seamless Gauss modification is shown to perform better with SANet. The final evaluation, based on the validation images, indicates that the modified SANet CNN outperforms the dFSR algorithm in terms of PSNR and MS-SSIM.

6.2. Concluding Remarks

The human eye contains day-vision receptors for red, green, and blue that are randomly distributed across the retina. Neural networks were thought to be superior to traditional algorithms in handling irregularly sampled image data due to their similarities with the human brain. However, this assumption does not necessarily hold true for convolutional neural networks. The convolutional layer as the fundamental building block of any CNN takes advantage of the spatial uniformity of image data, which is lacking in irregularly sampled image data. The experiments show that a UNet, which uses traditional convolution with spatially consistent kernels, struggles to extract the color information from the irregular mosaic image input and produce a homogeneous reconstruction. Only with the use of spatially adaptive convolution could the experiments generate compelling image outputs

from irregular sensor data.
The experiments with SANet, APINet, and CFA-normalized convolution represent only first efforts to exploit the full potential of CNNs for demosaicing irregularly sampled images. From this point, it might be worthwhile to conduct extensive literature research on machine learning approaches for processing non-image irregularly sampled data that may be applicable to the demosaicing problem.
Developing a CNN to fulfill a specific purpose generally can be approached in two different ways, the pure deep learning and the manual approach:
The first one is trying to let the network deep-learn everything by itself, relying completely on the principles of Supervised Learning and Back Propagation in order to adapt to the data and provide the best possible solution. This approach can be argued to be superior as it outsources the problem-solving completely to the machine. (An extreme example of this could be the tendency to train multiple image-processing steps end-to-end in a single CNN, as with Joint Denoising and Multi-Exposure-Fusion, Buades et al. (2022))
The manual approach would be to analyze the structure of the data in order to handcraft prior knowledge into the architecture of the CNN to constrain and lead the deep learning process into the right direction. This can be in form of custom data preprocessing, separately supervised branches, custom weights. This approach resembles the development of conventional procedural algorithms (e.g. traditional debayering algorithms or the research by Backes and Fröhlich (2020)) and tends to require greater domain knowledge and more human work.
However, in reality manual adjustments are often needed if the pure deep learning approach does not produce satisfying results, as during the early experiments of this thesis. To justify manual adjustments to the CNN architecture, one could also argue that even the very generic UNet architecture contains certain constraints that rely on assumed data characteristics: Convolution layers can be seen as optimized versions of fully connected layers that exploit the prior knowledge of image data being spatially homogeneous. In our case of irregularly sampled image data, this constraint enforced by spatially shared kernels inside convolutional layers is hindering finding the right solution. This example shows that modifications to the mechanics of a CNN may be justified by the characteristics of the data, in order to achieve good convergence or to increase training efficiency.

6.3. Suggested Future Research

CFA-normalized convolution operation was presented in Subsection 4.4.3 and used with a fixed Gaussian kernel to replace the global residual branch of the SANet. During the work on this thesis experiments were done to transfer this concept into a Conv layer with a trainable kernel. This Conv layer would have the same parameters as a classical conv

layer, except that it would have additional input and output channels for the CFA just like in Subsection 4.4.3. This would then allow to employ CFA-normed convolution inside a UNet architecture similar to spatially adaptive convolution inside SANet. While this idea could not be further developed during this thesis, it would be interesting to find out how it could be implemented and if it performs better on irregularly sampled image data.
The pseudo-random CFA patterns used by Backes and Fröhlich (2020) contain two times more green pixels than red or blue, derived from the Bayer pattern. However, Alleysson et al. (2005) suggests that the Bayer pattern should be changed to have two times more blue than red or green pixels to allow "[...]estimating achromatic spatial acuity to higher frequencies." It could be interesting to examine whether other proportions of the colors in the CFA would also lead to finer luminance details for pseudo-random CFAs. In this context the works of Syu et al. (2018) and Henz et al. (2018) are interesting as they approach the question of the optimal CFA with machine learning: They use a detachable Encoder-Decoder architecture to develop the optimal color filter pattern and the corresponding demosaicing method at the same time. The found CFA patterns interestingly consists of color filters other than red, green, and blue.
In the experiments conducted during this thesis cropped and resized images of size 256×256 pixels were used. However, real raw images have resolutions of 4k to 8k pixels. While in theory UNet should be able to handle different input resolutions, further research would be needed to confirm that SANet implementation could handle the additional complexity of larger images and the correspondingly larger random CFAs.
In regard of the SANet architecture (T. Zhang et al., 2022) memory consumption of spatially adaptive convolution greatly increases with image size. This might make an overlapping tiled image processing approach necessary as proposed by Ronneberger et al. (2015) (see Figure 16).
Most CNN architectures for classic debayering, which also address noise reduction, add synthetic noise on top of their clean training images (c.f. Gharbi et al., 2016; Qian et al., 2021). Most of them use a simplified signal-independent white Gaussian noise which is overlaid in an additive fashion. On the other side some research uses specially captured pairs of noisy and clean images to gain more realistic training data (see T. Zhang et al., 2022). Further research would be needed to find out how the CNN architectures examined in this thesis perform on noisy image data in terms of noise reduction.
During this thesis it was discussed that demosaicing is most challenging when reconstructing high frequency details, especially if they are strongly saturated. These structures can interfere with the CFA patterns and create artifacts that are perceived as disturbing during reconstruction. Gharbi et al. (2016) proposes an interesting approach to make training more efficient by using a special dataset that contains mainly such challenging structures. The Moiré Dataset published with the paper, contains image extracts that are particularly

poorly reconstructable when sampled with the Bayer pattern (see subsubsection 4.1.1). It would be interesting to investigate whether this set also provides more efficient training and thus better reconstructions for demosaicing irregular CFAs or whether it would need a dedicated set.

Bibliography

Aladdin Persson. (2021, February 2). *PyTorch Image Segmentation Tutorial with U-NET: Everything from scratch baby*. YouTube. Retrieved August 22, 2023, from https://www.youtube.com/watch?v=IHq1t7NxS8k

Alleysson, D., Susstrunk, S., & Herault, J. (2005). Linear demosaicing inspired by the human visual system. *IEEE Transactions on Image Processing*, *14*(4), 439–449. https://doi.org/10.1109/TIP.2004.841200

Andriani, S., Brendel, H., Seybold, T., & Goldstone, J. (2013). Beyond the Kodak image set: A new reference set of color image sequences. *2013 IEEE International Conference on Image Processing*, 2289–2293. https://doi.org/10.1109/ICIP.2013.6738472

Backes, P. (2019, December 5). *Bildsensoren mit unregelmäßiger Farbabtastung und universelle Demosaicking-Methoden* (Master's thesis). Hochschule der Medien. Stuttgart.

Backes, P., & Fröhlich, J. (2020). A practical approach on non-regular sampling and universal demosaicing of raw image sensor data. *London Imaging Meeting*, *1*(1), 91–95. https://doi.org/10.2352/issn.2694-118X.2020.LIM-17

Bayer, B. E. (1976, July 20). *Color imaging array* (U.S. pat. 3971065A). Retrieved February 22, 2023, from https://patents.google.com/patent/US3971065A/en

Buades, A., Martorell, O., & Sánchez-Beeckman, M. (2022, January 18). *Joint denoising and HDR for RAW video sequences*. arXiv: 2201.07066 `[cs, eess]`. Retrieved January 22, 2023, from http://arxiv.org/abs/2201.07066

Burger, W., & Burge, M. J. (2022). *Digital Image Processing: An Algorithmic Introduction*. Springer International Publishing. https://doi.org/10.1007/978-3-031-05744-1

Commission, I. E., et al. (2003). *IEC 61966-2-1: 1999/AMD1: 2003 Amendment 1-Multimedia systems and equipment- Colour measurement and management- Part 2-1: Colour management- Default RGB colour space- sRGB*.

Condat, L. (2010). Color filter array design using random patterns with blue noise chromatic spectra. *Image and Vision Computing*, *28*(8), 1196–1202. https://doi.org/10.1016/j.imavis.2009.12.004

Conv2d. (2023). PyTorch 2.0 Documentation. Retrieved September 13, 2023, from https://pytorch.org/docs/2.0/generated/torch.nn.Conv2d.html#conv2d

Cui, K., Boev, A., Alshina, E., & Steinbach, E. (2021). Color Image Restoration Exploiting Inter-Channel Correlation With a 3-Stage CNN. *IEEE Journal of Selected Topics in Signal Processing*, *15*(2), 174–189. https://doi.org/10.1109/JSTSP.2020.3043148

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 248–255. https://doi.org/10.1109/CVPR.2009.5206848

Dippé, M. A., & Wold, E. H. (1985). Antialiasing through stochastic sampling. *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, 69–78.

Dumoulin, V., & Visin, F. (2018, January 11). *A guide to convolution arithmetic for deep learning*. arXiv: 1603.07285 `[cs, stat]`. https://doi.org/10.48550/arXiv.1603.07285

Eisemann, L., Froehlich, J., Hartz, A., & Maucher, J. (2020). Expanding dynamic range in a single-shot image through a sparse grid of low exposure pixels. *Electronic Imaging*, *32*(7), 229-1-229–7. https://doi.org/10.2352/ISSN.2470-1173.2020.7.ISS-229

Gharbi, M., Chaurasia, G., Paris, S., & Durand, F. (2016). Deep joint demosaicking and denoising. *ACM Transactions on Graphics*, *35*(6), 191:1–191:12. https://doi.org/10.1145/2980179.2982399

Gonzalez, R. C., & Woods, R. E. (2008). *Digital image processing* (3rd ed). Prentice Hall.

Grosche, S., Seiler, J., & Kaup, A. (2018a). Design Techniques for Incremental Non-Regular Image Sampling Patterns. *2018 IEEE International Conference on Imaging Systems and Techniques (IST)*, 1–6. https://doi.org/10.1109/IST.2018.8577090

Grosche, S., Seiler, J., & Kaup, A. (2018b). Iterative Optimization of Quarter Sampling Masks for Non-Regular Sampling Sensors. *2018 25th IEEE International Conference on Image Processing (ICIP)*, 26–30. https://doi.org/10.1109/ICIP.2018.8451658

Gunturk, B., Glotzbach, J., Altunbasak, Y., Schafer, R., & Mersereau, R. (2005). Demosaicking: Color filter array interpolation. *IEEE Signal Processing Magazine*, *22*(1), 44–54. https://doi.org/10.1109/MSP.2005.1407714

Gurrola-Ramos, J., Dalmau, O., & Alarcon, T. E. (2021). A Residual Dense U-Net Neural Network for Image Denoising. *IEEE Access*, *9*, 31742–31754. https://doi.org/10.1109/ACCESS.2021.3061062

Hasinoff, S. W. (2014). Photon, Poisson Noise. In K. Ikeuchi (Ed.), *Computer Vision: A Reference Guide* (pp. 608–610). Springer US. https://doi.org/10.1007/978-0-387-31439-6_482

Hennenfent, G., & Herrmann, F. J. (2007). Irregular Sampling – From Aliasing to Noise, cp. https://doi.org/10.3997/2214-4609.201401481

Henz, B., Gastal, E. S. L., & Oliveira, M. M. (2018). Deep Joint Design of Color Filter Arrays and Demosaicing. *Computer Graphics Forum*, *37*(2), 389–399. https://doi.org/10.1111/cgf.13370

Ignatov, A., Van Gool, L., & Timofte, R. (2020, February 13). *Replacing Mobile Camera ISP with a Single Deep Learning Model*. arXiv: 2002.05509 `[cs, eess]`. Retrieved December 13, 2022, from http://arxiv.org/abs/2002.05509

Jiang, J., Liu, D., Gu, J., & Susstrunk, S. (2013). What is the space of spectral sensitivity functions for digital color cameras? *2013 IEEE Workshop on Applications of Computer Vision (WACV)*, 168–179. https://doi.org/10.1109/WACV.2013.6475015

Khan, S., Rahmani, H., Shah, S. A. A., & Bennamoun, M. (2018). *A Guide to Convolutional Neural Networks for Computer Vision*. Springer International Publishing. https://doi.org/10.1007/978-3-031-01821-3

Kingma, D. P., & Ba, J. (2017, January 29). *Adam: A Method for Stochastic Optimization*. arXiv: 1412.6980 `[cs]`. Retrieved April 11, 2023, from http://arxiv.org/abs/1412.6980

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, *60*(6), 84–90. https://doi.org/10.1145/3065386

Kumar, S. P., Peter, K. J., & Kingsly, C. S. (2023). De-noising and Demosaicking of Bayer image using deep convolutional attention residual learning. *Multimedia Tools and Applications*, *82*(13), 20323–20342. https://doi.org/10.1007/s11042-023-14334-z

Li, X., Gunturk, B., & Zhang, L. (2008, January 27). Image demosaicing: A systematic survey. In W. A. Pearlman, J. W. Woods, & L. Lu (Eds.). https://doi.org/10.1117/12.766768

Liu, L., Jia, X., Liu, J., & Tian, Q. (2020). Joint Demosaicing and Denoising With Self Guidance, 2240–2249. Retrieved February 6, 2023, from https://openaccess.thecvf.com/content_CVPR_2020/html/Liu_Joint_Demosaicing_and_Denoising_With_Self_Guidance_CVPR_2020_paper.html

Loshchilov, I., & Hutter, F. (2019, January 4). *Decoupled Weight Decay Regularization*. arXiv: 1711.05101 `[cs, math]`. https://doi.org/10.48550/arXiv.1711.05101

Mathworks Inc. (2021, September 2). *Introducing Deep Learning with MATLAB*. Retrieved September 14, 2023, from https://www.mathworks.com/content/dam/mathworks/ebook/gated/80879v00_Deep_Learning_ebook.pdf

Murphy, J. (2016). An overview of convolutional neural network architectures for deep learning. *Microway Inc*, 1–22.

Nilsson, J., & Akenine-Möller, T. (2020, June 29). *Understanding SSIM*. arXiv: 2006.13846 `[cs, eess]`. Retrieved May 13, 2023, from http://arxiv.org/abs/2006.13846

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., . . . Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in neural information processing systems*, *32*.

Poynton, C. (2018, July 30). *Colour Appearance Issues in Digital Video, HD/UHD, and D-cinema* (Doctoral dissertation). Simon Fraser University. Retrieved May 15, 2023, from https://summit.sfu.ca/item/19854

Qian, G., Wang, Y., Dong, C., Ren, J. S., Heidrich, W., Ghanem, B., & Gu, J. (2021, March 29). *Rethinking the Pipeline of Demosaicing, Denoising and Super-Resolution.* arXiv: 1905.02538 `[cs, eess]`. https://doi.org/10.48550/arXiv.1905.02538

Ronneberger, O., Fischer, P., & Brox, T. (2015, May 18). *U-Net: Convolutional Networks for Biomedical Image Segmentation.* arXiv: 1505.04597 `[cs]`. https://doi.org/10.48550/arXiv.1505.04597

Schöberl, M., Seiler, J., Foessel, S., & Kaup, A. (2011). Increasing imaging resolution by covering your sensor. *2011 18th IEEE International Conference on Image Processing*, 1897–1900. https://doi.org/10.1109/ICIP.2011.6115839

Sigma Corporation. (2022, February 21). *Development status of the three-layer image sensor | Information | News.* SIGMA Corporation. Retrieved February 8, 2023, from https://www.sigma-global.com/en/news/2022/02/21/17697/

Stokes, M., Anderson, M., Chandrasekar, S., & Motta, R. (1996, November 5). *A Standard Default Color Space for the Internet - sRGB.* Retrieved May 10, 2023, from https://www.w3.org/Graphics/Color/sRGB.html

Sultana, F., Sufian, A., & Dutta, P. (2018). Advancements in Image Classification using Convolutional Neural Network. *2018 Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)*, 122–129. https://doi.org/10.1109/ICRCICN.2018.8718718

Sumner, R. (2014). Processing RAW Images in MATLAB. http://www.rcsumner.net/raw_guide/RAWguide.pdf

Syu, N.-S., Chen, Y.-S., & Chuang, Y.-Y. (2018, February 11). *Learning Deep Convolutional Networks for Demosaicing.* arXiv: 1802.03769 `[cs]`. https://doi.org/10.48550/arXiv.1802.03769

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2014, September 16). *Going Deeper with Convolutions.* arXiv: 1409.4842 `[cs]`. https://doi.org/10.48550/arXiv.1409.4842

TensorBoard. (2023, August 22). Retrieved August 22, 2023, from https://github.com/tensorflow/tensorboard

Wang, Z., Simoncelli, E., & Bovik, A. (2003). Multiscale structural similarity for image quality assessment. *2*, 1398–1402 Vol.2. https://doi.org/10.1109/ACSSC.2003.1292216

Xing, W., & Egiazarian, K. (2021). End-to-End Learning for Joint Image Demosaicing, Denoising and Super-Resolution, 3507–3516. Retrieved February 6, 2023, from https://openaccess.thecvf.com/content/CVPR2021/html/Xing_End-to-End_Learning_for_Joint_Image_Demosaicing_Denoising_and_Super-Resolution_CVPR_2021_paper.html?ref=https://githubhelp.com

Xing, Y., Zhong, L., & Zhong, X. (2020). An Encoder-Decoder Network Based FCN Architecture for Semantic Segmentation. *Wireless Communications and Mobile Computing*, *2020*, 1–9. https://doi.org/10.1155/2020/8861886

Zhang, C., Li, Y., Wang, J., & Hao, P. (2016). Universal Demosaicking of Color Filter Arrays. *IEEE Transactions on Image Processing*, *25*(11), 5173–5186. https://doi.org/10.1109/TIP.2016.2601266

Zhang, T. (2023, January 31). *Spatial Adaptive Network for Real Image Demosaicing*. Retrieved February 16, 2023, from https://github.com/ColinTaoZhang/SANet

Zhang, T., Fu, Y., & Li, C. (2022). Deep Spatial Adaptive Network for Real Image Demosaicing. *Proceedings of the AAAI Conference on Artificial Intelligence*, *36*(3), 3326–3334. https://doi.org/10.1609/aaai.v36i3.20242

Zhao, H., Gallo, O., Frosio, I., & Kautz, J. (2018, April 20). *Loss Functions for Neural Networks for Image Processing*. arXiv: 1511.08861 [cs]. Retrieved March 22, 2023, from http://arxiv.org/abs/1511.08861

Zhou, R., Achanta, R., & Süsstrunk, S. (2018, February 19). *Deep Residual Network for Joint Demosaicing and Super-Resolution*. arXiv: 1802.06573 [cs]. https://doi.org/10.48550/arXiv.1802.06573

List of Figures

List of Tables

Glossary

Debayering

Demosaicing of Bayer images

Demosaicing, Demosaicking

Process of reconstructing a full colored RGB image from the color mosaic encoded into the raw sensor image.

MATLAB

The commercial software and programming language by the MathWorks, Inc. can be used to solve mathematical problems and work with matrices.

MATLAB Engine

A small Python Library that provides Bindings to call MATLAB functions from Python by using an installed MATLAB instance.

Python

Programming language commonly used in data science and machine learning.

PyTorch

Python based machine learning framework, initially presented by Paszke et al., 2019.

Appendix A.

Further Experiments with (Own) APINet Architecture

During the course of this thesis, experimentation was conducted on a prototype CNN model for adaptive pattern interpolation referred to as APINet. The basics of its design will be explained here. APINet shares similarities with SANet as it also learns to adapt to the CFA pattern, but in a much more constrained way. The basic APINet comprises only two convolutional layers per image channel and does not utilize an Encoder-Decoder architecture. However, it is mentioned here to showcase the limitations of spatially consistent convolution in a traditional UNet, as it achieved better metrics after only a few epochs of training compared to the simple UNet's extended training. Additionally, the APINet is computationally efficient, making it suitable for real-time preview purposes. However, the design of the APINet only allows it to adapt to different sample constellations and does not take the image content into account. It can be compared to the basic linear debayering algorithm described in Subsection 2.2.1 as it applies specific interpolations for different sample constellations.
Figure 40 illustrates the interpolation of a single color channel using a set of learned filter kernels, weighted by features acquired from the sampling pattern. The most basic variant of APINet reconstructs all three color channels independently. Alternatively, heuristic optimizations known from traditional debayering algorithms can be applied to exploit cross-channel correlations during reconstruction.

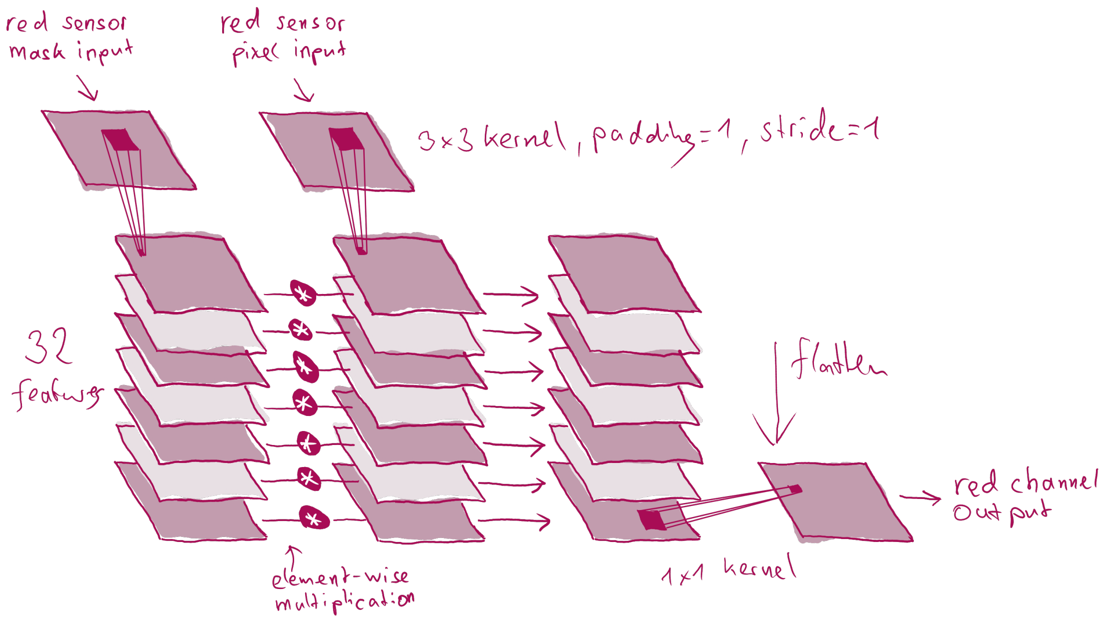

Figure 40: Prototype of an Adaptive Pattern Interpolation Network (APINet): Reconstruction of the red channel

Appendix B.

Source Code Repository

The PyTorch implementation of the experiments conducted for this thesis is available at https://gitlab.com/antonstoetzer/irregular-demosaicing-cnn.