Demosaicing using a Convolutional Neural Network approach

Authors
Karin Dammer* karindammer@live.se
Ronja Grosz† rongr946@student.liu.se

Supervisors
Magnus Oskarsson* magnuso@maths.lth.se
Pierangelo Dell'Acqua† pierangelo.dellacqua@liu.se

Examiners
Anders Heyden* heyden@maths.lth.se
Reiner Lenz† reiner.lenz@liu.se

June 16, 2017

Abstract

This thesis is about investigating the feasibility to use convolutional neural networks as a demosaicing approach. Three loss methods and different layer structures have been evaluated as well as changing different parameters and layers in the convolutional neural network to find which changes are beneficial to make a neural network perform demosaicing well.

The convolutional neural network has been compared to a fully convolutional neural network, a multilayer perceptron and the Hamilton Adams demosaicing algorithm. The prospect of demosaicing raw image sensor data and images with noise was also investigated.

The conclusion is that a convolutional neural network can indeed perform demosaicing with good results, even when using a small and less complex network. The convolutional neural network was also able to demosaic raw images as well as remove noise from images, although with not as good result as when demosaicing artificial data. The convolutional neural network on average performed demosaicing with a peak signal to noise ratio of 34 dB. This compares to the Hamilton Adams method that has a peak signal to noise ratio of 37 dB, although when measured with the structural similarity our method performs better than the Hamilton Adams method.

Index Terms: Convolutional neural networks, fully convolutional neural network, demosaicing, image sensor data, noise reduction.

*Lund University
†Linköping University

Preface

This master thesis was made in cooperation with the Department of Science and Technology at Linköping University, the department of Mathematics at Lund University and Axis Communications AB. It was carried out in Lund at Axis premises during the spring of 2017.

Acknowledgements

We would like to thank our supervisors and examiners Pierangelo Dell'Acqua, Reiner Lenz, Magnus Oskarsson and Anders Heyden at LiU and LTH for enabling us to work together on this thesis and for all the guidance during the thesis. Thanks to all supervisors at Axis for providing guidance and the freedom to shape our thesis freely and thanks to all Core Technology employees for showing such an interest in our thesis. We will also like to thank our friends and family for all the support you have given us.

We would like to dedicate some special thanks to:
Gunnar Dahlgren for an excellent code base to start on.
Andreas Nilsson for providing your famous tool and knowledge about demosaicing and the image processing pipeline.
Per Wilhelmsson for guiding us in the right direction early on in our thesis.
Jakob Grundström for providing guidance in the setup of the environment, your enthusiasm towards machine learning and for providing us with relevant scientific papers and data sets.
Johan Jeppsson for stepping in after Per, driving us to rubber duck our thesis.

Contents

List of Figures

List of Tables

Nomenclature

CCD Charge-coupled device, page 7

CFA Color filter array, page 6

CMOS Complementary metal-oxide-semiconductor, page 7

CNN Convolutional neural network, page 6

FCNN Fully convolutional neural network, page 6

HDR High dynamic range, page 15

Hyperparameter Parameter describing the properties of the network. This parameter is not updated during the training session, page 25

MLP Multilayer perceptron, page 17

MS-SSIM Multi-scale structural similarity index, page 19

MSE Mean square error, page 13

PSNR Peak signal-to-noise ratio, page 13

ReLU Recitifier linear unit, page 11

RGB Additive color model which combines the three primary colors red, green and blue, page 8

SSIM Structural similarity, page 13

VDP Visual difference prediction, page 15

YCbCr Color model using luminance and chrominance measures, page 8

1 Introduction

Most color cameras use a single image sensor overlaid with a color filter array (CFA). Consequently, some form of interpolation, called demosaicing is needed to get a full resolution color image. Demosaicing is usually done through different interpolation algorithms such as bilinear interpolation or the Hamilton Adams method [1]. This master thesis has been about investigating the suitability to use neural networks to demosaic raw image data. This has been done by comparing results from different neural networks, such as a convolutional neural network (CNN) and a fully convolutional neural network (FCNN), to conventional interpolation methods. For the CNN and FCNN different parameters and structures have been analyzed such as loss metric, optimizer, number of epochs and different numbers and types of layers to characterize properties of a successful neural network for the demosaicing problem. The work also aims to investigate and quantify whether a CNN can perform noise reduction simultaneously with the demosaicing.

1.1 Purpose and problem statements

The purpose of this master thesis is to investigate if a convolutional neural network approach can replace the classical interpolation methods for demosaicing color filter arrays. To do so a number of problem statements were formulated.

- Compare a convolutional neural network with a multilayer perceptron on the ability to demosaic a color filter array.
- Compare a method using convolutional neural networks with a method using bilinear interpolation.
- Compare different loss methods as the error metric in the network: L_2, SSIM, PSNR.
- Investigate the benefit of using a residual layer.
- Compare a fully convolutional neural network with a convolutional neural network using fully connected layers on the ability to demosaic a CFA.
- Investigate the possibility to train and validate a convolutional neural network using raw image data from an image sensor.
- Investigate the possibility of a convolutional neural network performing noise reduction.

1.2 Limitations

To answer the problem statements, limitations to the problem have to be set. This thesis has not been about finding the best convolutional network for demosaicing purposes, but more about finding which parts could be beneficial for demosaicing purposes. The investigations have been limited to relatively shallow networks, e.g. networks with a small number of operational layers such as convolutions. This is to limit the complexity of the network and the training time. For the same reasons the kernel width is kept low.

Training and validation data has mostly been limited to artificially produced mosaic data since processed images are easier to find than raw images. The data sets for training and validation are also kept relatively small if comparing to corresponding experiments encountered in the literature study. The number of epochs for the training is also considerably low. The main reason for this is to lower the training and validation time. Therefore the full potential of our method will not be achieved but rather gives a measurement of the potential the neural network could have when applied to demosaicing.

2 Background

This section gives some background to the demosaicing problem and to neural networks and their components.

2.1 Demosaicing

Demosaicing is a digital image process for producing color images from incomplete color data produced by an image sensor overlaid with a color filter array. Three charge-coupled device (CCD) sensors or complementary metal-oxide-semiconductor (CMOS) sensors can both be used to get three full-channel color images directly, although this approach is costly and still needs spectral bandpasses, often in the form of beam splitters with half transparent mirrors. A more cost effective solution, that most digital cameras on the market use, is to put a CFA in front of the sensor to capture one color component per pixel and interpolate the missing color components. CFAs are used to separate the color information, since the typical photo sensor only detects light intensity without information about the wavelength. The CFA consists of spectrally selective filters arranged in certain patterns. The sparsely sampled color channels collected from the color filter array are called CFA images or mosaic images.

2.1.1 Color filter array patterns

The choice of CFA pattern mostly depend on the camera manufacturer but the Bayer pattern [2] is the most commonly used CFA [3]. Demosaicing CFA's may cause image artifacts. A common reason for image artifacts is aliasing, which occurs when a signal is sampled at a less than twice the highest frequency present in the signal. In images, aliasing is caused by a low sampling in the spatial domain. Moiré patterns, which is a large scale interference pattern, occur when sampling a repetitive pattern of high spatial frequency at a low frequency. The interference appears as repetitive patterns, color artifacts or pixels arranged in an unrealistic maze-like pattern. An example of a Moire artifact can be seen in figure 2. Demosaicing may also cause other false color artifacts. These often manifest themselves along edges where the interpolation might have been done across an edge. Zippering artifacts is another artifact that is commonly occuring along edges. Zippering is prevalent when the demosaicing algorithm averages pixel values over an edge, causing the edge to become blurred in a zipper pattern. An example of zippering can be seen in figure 3.

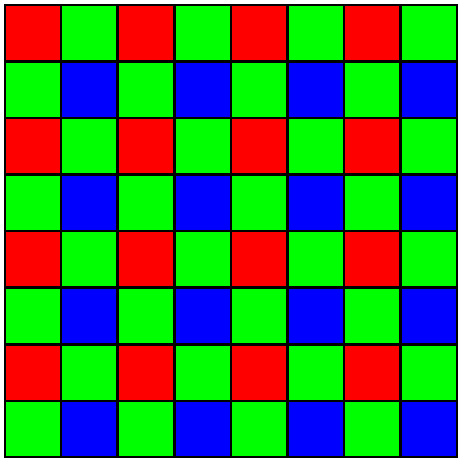


Figure 1: A Bayer color filter pattern. Each 2×2 sub mosaic contains two green, one blue and one red filter, where each filter covers one pixel sensor.

The Bayer pattern is a CFA pattern which measures the green image on a checkerboard grid with half of the image resolution and the red and blue images on rectangular grids with each a quarter of the image resolution, see figure 1. The green channel is thus measured at a higher sampling rate. This is motivated by the fact that the human visual system has

Figure 2: Moire artifacts in the fence.

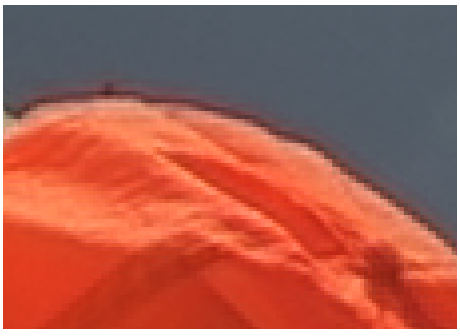


Figure 3: Zippering artifacts along an edge.

its peak sensitivity at the wavelength corresponding to green. This property is used in most color filter arrays beyond the Bayer pattern. Since the Bayer pattern is common to use, most previous work on demosaicing only covers methods for the Bayer pattern.

Two other wide-spread color filter array patterns is Sony's RGBW Bayer pattern and the Aptina Clarity+ pattern. The Sony RGBW pattern has the same structure as the Bayer pattern except every other green filter is replaced with clear filters making the pixels register light of all wavelengths. This gives a pattern structure with all four different filter types in rectangular grids. The Aptina Clarity+ pattern also has the same structure as the Bayer pattern except all green filters are replaced with clear filters.

2.1.2 Color models

Colors are often described and organized through different color coordinate systems. A common color model is the RGB color model where the color is expressed as an additive combination of the three primary colors red, green and blue.

Another widely used representation is the YCbCr model. The colors are decorrelated and separated into a luminance channel and two chrominance channels. The luminance channel is constructed as a linear combination of the red, green and blue channels and the chrominance channels are defined as the weighted difference between the red respectively blue channel and the luminance channel.

2.1.3 Demosaicing methods

Demosacing is commonly done through interpolating the missing color channel information. The following section describes some demosaicing methods operating in the spatial dimension.

Bilinear interpolation

The simplest bilinear (or linear) interpolation method simply estimates the pixel value as the mean value of the nearest surrounding pixels in each color, i.e. to estimate the green pixel value of a non-green pixel, four values can be used whereas a red or blue pixel value only has access to two nearby pixels in the correct color. This simple method does not use the correlation between different color layers and therefore performs poorly and yields blurred images. The method can be expanded to include more of the corresponding pixels, including the pixel value in the current pixel. An example of this is a method by Malvar et al. [4] where 9 pixel values are used to estimate the green channel and 11 values are used to estimate the red and blue channels. This makes better use of the correlation between the different color channels. To calculate the pixel value a weighted sum of the corresponding pixels are calculated according to certain weight patterns.

Sequential demosaicing

The green part of a mosaiced image often contains twice as many pixels (and more information) than the red or blue part. A common approach is to interpolate the green (or luminance) channel first and then use the fully interpolated green channel to estimate the red and blue (or chrominance) channels [3]. The reason for this is to avoid aliasing which is less likely to occur in the green channel as it encodes twice as much information. As described by Li et al. [3] the interpolation is many times done by an initial edge-detection algorithm to find edges to interpolate along to avoid zippering effects along the edges.

The chrominance channel is interpolated by assuming constant hue and estimating the difference between the blue and green hannels, and the red and green channels. The pixel values for the green channel are subtracted and an estimation for the red and blue channels is obtained. The downfall of this method is primarily that errors from the interpolation of the green channel propagate into the red and blue channels which, in the end can create large errors in the demosaiced image.

Another approach is the luminance channel interpolation where the missing data in the green channel is interpolated through heuristic edge-direct rules. The local edge direction is estimated from available data on the green, red or blue channel and the second order gradients of the chrominance channels can be used as a correction term. Local covariance, estimated based on geometric duality, can be used to adapt the interpolation.

Iterative demosaicing

Sequential demosaicing can have a fundamental weakness of error propagation where any errors rendered during the interpolation of the luminance channel inevitably propagate to the chrominance channels. This can be solved through iterative reconstruction. The green, red and blue channels are iteratively refined based on color ratio rules.

Machine learning methods

As machine learning techniques gain more ground in the image processing field a number of different machine learning methods have been tried for demosaicing purposes. Neural networks using both convolutional kernels and multilayer perceptrons have been proven useful and can, if designed properly, be used for both demosaicing and denoising [5].

Super resolution is a method that aims to increase the resolution of a low resolution image by non-linearly interpolating new values between the existing pixels. This approach can be applied channel-wise on a mosaiced image to find the missing values. A machine learning method that has been proven successful for super resolution tasks is Generative

adversarial networks [6]. The generative adversarial network is a network structure with two parallel networks, one generative and one discriminative, that are trained together and challenge each other to improve the performance of the requested task.

Adaptive color plane interpolation

In 1996 Hamilton and Adams proposed a method for demosaicing using adaptive color plane interpolation [1]. The method uses a sequential demosaicing approach which separates the image into RGB color channels. It first interpolates the green channel and uses it to interpolate the red and blue color channels. The interpolation is done using an edge-detection approach that allows the interpolation to be done along edges and not across them.

2.2 Image sensor noise

Data from the image sensor contains unavoidable image noise in the form of variations of brightness and color information that was not present in the pictured scene [7]. The sensor noise is a problem since it creates grainy images. Therefore the noise needs to be removed or limited prior to demosaicing or after. Noise in the sensor data is for the most part caused by photon transfer noise which can originate from several different sources. There are four fundamental noise sources: signal shot noise, Fano noise, fixed pattern noise and read noise. Signal shot noise and Fano noise are two signal deviations that are related to photon interactions and are thus dependent on the signal strength. Fixed pattern noise is associated with pixel to pixel sensitivity irregularities and read noise includes all other noise sources that are not dependent on signal strength. These four noise sources can be approximated through Gaussian distribution, although the shot noise is Poisson distributed.

Signal shot noise is related to how photons arrive on an image sensor. The number of photon interactions per pixel can vary and is described by the standard deviation that is called photon shot noise. The interacting photon shot noise variance can be described as

$$\sigma_{SHOT}(P_I)^2 = P_I \frac{e^{hc/\lambda kT}}{e^{hc/\lambda kT} - 1} \tag{1}$$

where h is the Planck constant, λ is the photon wavelength, k is the Boltzmann's constant, c is the speed of light and T is the absolute temperature in kelvin.

The signal shot noise generated by interacting photons can be described as

$$\sigma_{SHOT} = (\eta_i S)^{1/2} \tag{2}$$

where η_i is the quantum yield gain and S is the signal.

Fano noise is a multiple electron-hole charge generation and is described as

$$\sigma_{FN} = (F_f \eta_i)^{1/2} = (F_f \frac{h_v}{E_{e-h}})^{1/2} \tag{3}$$

where F_f is the Fano factor which is the variance of the number of electrons generated divided by the average number of electrons generated per interacting photon.

The pixel's collection process of the charge from the photoelectrons' interaction is not perfect since some pixels collect charges more efficiently than others. This results in pixel to pixel sensitivity differences which generates fixed pattern noise in an image. Fixed pattern noise is defined as

$$\sigma_{FPN} = P_N S \tag{4}$$

where σ_{FPN} is the fixed pattern noise and P_N is the fixed pattern noise quality factor.

Read noise is defined as any noise source that is not a function of the signal. It is added together with

the signal shot noise, Fano noise and the fixed pattern noise to produce the total noise equation,

$$\sigma_{TOTAL} = (\sigma_{READ}^2 + \sigma_{FN}^2 + \sigma_{SHOT}^2 + \sigma_{FPN}^2)^{1/2}. \quad (5)$$

2.3 Neural networks

Neural networks are inspired by how neurons in the brain work, firing signals only when certain conditions are met, i.e. the combined input in a neuron has to exceed some kind of threshold value (bias) to produce an output. The concept of threshold logic was proposed by McCulloch and Pitts [8] in 1943 and can be described as

$$y = \begin{cases} 1, & \text{if } \sum_{i=1}^{N} w_i x_i \geq b \\ 0, & \text{if } \sum_{i=1}^{N} w_i x_i < b \end{cases} \quad (6)$$

where w_i is a weight, x_i is an input value and b is a threshold value. The neuron is activated at $y = 1$.

Neural networks are suitable for many computing tasks such as classification and clustering and the use of neural networks has increased for various kinds of mathematical and computational applications as they possess desirable characteristics. These include good ability to fit to non-linear data, ability to perform parallel computations and the ability to update and learn [9].

Hidden layers

A neural net consists of layers of neurons where the input of a layer is connected to the output of the previous layer. The simplest neural network consists of an input layer, an output layer and one or more hidden layers. The hidden layers contain learnable weights and biases that are optimized by the backpropagation process, which is a two phase cycle propagation for updating the weights of the neural network. The weights decide how much of each input should be added in the neuron and the bias sets the threshold value. Hidden layers can be fully connected layers connecting all input variables to all output variables. They can also consist of convolutional layers or locally connected layers.

Activation layers

Hidden layers often contain a non-linear activation layer such as a rectifier linear unit (ReLU) or a sigmoid function. The activation function is what makes the network non-linear and makes the network more complex and able to learn more complex functions. If the activation functions were omitted the network could be reduced to a linear regression. Equation 6 shows the simplest activation function in terms of a step function. The ReLU function is defined as

$$f(x) = max(0, x) \quad (7)$$

and is currently the most commonly used activation function for neural networks [10].

Several recent studies such as Krizhevsky et al. [11] have shown that the ReLU function increases the convergence of the neural network in comparison to the sigmoid function. Developments of the ReLU functions are the leaky ReLU and parametric ReLU. The leaky ReLU adds a small slope to the negative part and is defined as

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha x, & \text{if } x \leq 0 \end{cases} \quad (8)$$

where α is a small constant. The parametric ReLU is defined in the same way with the difference that α is a trainable parameter.

The sigmoid function is defined as

$$f(x) = \frac{1}{1 + e^{-x}}. \quad (9)$$

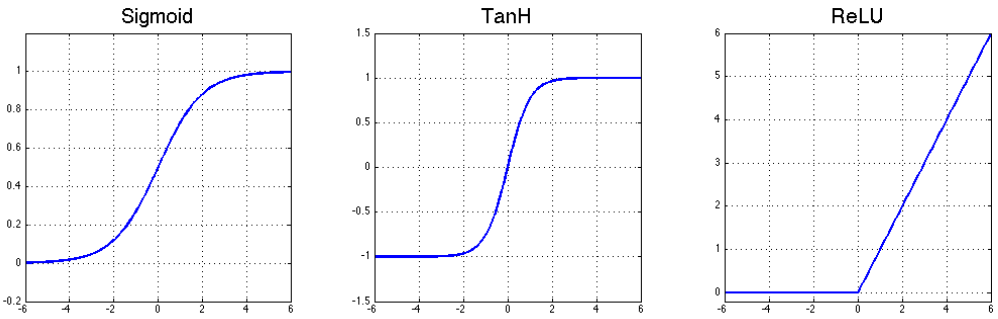


Figure 4: Shapes of the curves for the sigmoid, TanH and ReLU functions, from left to right respectively.

The sigmoid function has a value region of $[0,1]$ which gives it a non-zero mean value.

The tanh function is similar to the sigmoid function and is defined as

$$f(x) = \frac{2}{1+e^{-2x}} - 1. \tag{10}$$

The tanh function has a value region of $[-1,1]$ and a mean value of zero.

The sigmoid, tanh and ReLU function shapes can be seen in figure 4.

Training

To establish the ultimate set of weights and biases for the net it has to be trained to produce an output as similar as possible to a known ground truth image, e.g. the image that the network should learn to reproduce through demosaicing. This is done by an optimization algorithm. The optimization algorithm minimizes the loss function i.e. a quantification of the difference between the result from the net and the ground truth. The gradient of the loss function is computed to establish in what direction the optimization algorithm should update the weights. The process is ideally iterated until the result converges. True convergence, when the produced image is the same as the ground truth image, is hard to achieve and the training is iterated until the loss function yields a sufficiently low value.

Epoch

An epoch is a measurement of when the neural network runs through the whole training data set once. With limited data, a neural network needs to be trained for several epochs to reach convergence. The number of epochs can vary and can either be set to a fixed number, as done by Kapah et al. [12] or vary over iterations as the training continues until a certain termination criteria is met, as done by Gharbi et al. [5].

Regularization

Neural networks can become overfitted, which is when the network performs well on training data but does not achieve good results on new data. To prevent overfitting it is common to regularize the weights. This means that large weights are penalized and the training procedure is encouraged to find smaller values for the weights. A simple method for

regularization can be described as

$$\tilde{E}(\mathbf{w}) = E(\mathbf{w}) + \lambda \cdot \frac{1}{2}\mathbf{w}^2 \tag{11}$$

where E is the loss function, w is the weights and λ is a small constant. When used in neural networks this regularization is referred to as weight decay.

Dropout layers

Another method for avoiding overfitting is to apply a dropout layer to each layer. The dropout layer inhibits a given fraction of the neurons and sets them to zero. Which neurons that are dropped out are randomly distributed and is changed for every batch of training data. It can also be seen as several different networks with different connections between the neurons that are sub-sampled and trained simultaneously [13]. All neurons are used to produce an output, taking the dropped out values into account as zero values. Dropout layers are not used during validation.

2.3.1 Loss functions

The loss function is the quantification of the difference between the output from the net and the ground truth and is used to measure the convergence. Several methods have been proposed throughout the years, both simple methods using norm values and more advanced methods using correlation between the result and ground truth.

Norm based methods

A common loss function to evaluate neural networks is the L_2 loss function [14]. The method simply computes the square of the Euclidean norm of the difference between the result and the ground truth and can be described as

$$L_2(x,y) = \sum_{i=0}^{n} (y_i - x_i)^2 \tag{12}$$

where n is the number of pixels in the image sample. The L_2 method is considered to be the standard method but does not always achieve the best performance [14].

The least absolute deviation, the L_1 loss,

$$L_1(x,y) = \sum_{i=0}^{n} |y_i - x_i| \tag{13}$$

is also commonly used, although it is not as stable as the L_2 loss function.

PSNR

The peak signal-to-noise ratio uses the L_2 norm to compute the mean square error (MSE) and compare it to the maximal image value (255 in an 8-bit representation). The PSNR and MSE are calculated as

$$\begin{aligned} L_{PSNR}(x,y) &= 10\log_{10}(255^2/MSE(x,y)) \\ MSE(x,y) &= \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}(x_{ij} - y_{ij})^2. \end{aligned} \tag{14}$$

The PSNR does not give much more information than the MSE value as it is used in the calculations but the measurements are often presented in the logarithmic decibel scale due to the wide range of results. This way of presenting the error can give a better intuition of the image quality. The logarithmic measure also benefits from being normalized with the maximum value of the data. This gives a metric that is independent of the bit depth in the image.

SSIM

The structural similarity (SSIM) method aims to get a better representation of what metrics the human visual system percepts. As the L_2 and PSNR loss does

not correlate well with the human perception of image quality [15] another method with better estimation for what the human visual system is expecting in terms of image quality is needed. Therefore a method for measuring the structural similarity was proposed by Wang et al. [16]. The properties of the SSIM index includes giving information of how similar adjacent pixels are instead of just a metric of their closeness to the ground truth. The structural similarity index measures the difference in the luminance channel and the contrast between images as well as the correlation between images. The luminance is calculated as

$$l(x,y) = \frac{2\mu_x\mu_y + C}{\mu_x^2 + \mu_y^2 + C} \tag{15}$$

where μ is the mean value for the images and $C = (K_1 L)^2$ is a constant preventing zero division. L is chosen as the maximum pixel value, 255 for 8-bit gray scale images, and $K_1 \ll 1$ is a small constant.

The mean luminance is then extracted from the image as the structural features of the images are independent of the illumination. The standard deviation is defined as

$$\sigma_x = \left(\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)^2\right)^{1/2}. \tag{16}$$

It is natural to compute the contrast as a function of the standard deviation. The contrast comparison function accordingly measures the differences in the standard deviation and is described as

$$c(x,y) = \frac{2\sigma_x\sigma_y + D}{\sigma_x^2 + \sigma_y^2 + D} \tag{17}$$

where $D = (K_2 L)^2$ and $K_2 \ll 1$.

The structural information defined as the information about objects in the image is independent of luminance and contrast and is computed with a normalization of the standard deviation, i.e. the standard deviation is set to one as the expression is divided with its standard deviation. The correlation coefficient is defined as

$$\sigma_{xy} = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)(y_i - \mu_y) \tag{18}$$

and the structure function is calculated according to

$$s(x,y) = \frac{\sigma_{xy} + E}{\sigma_x\sigma_y + E} \tag{19}$$

where $E = (K_3 L)^2$ and $K_3 \ll 1$.

The final value for the SSIM index has values in the range $\in [-1,1]$ and reaches its maximum when $x = y$. It is calculated as

$$L_{SSIM}(x,y) = l(x,y)\cdot c(x,y)\cdot s(x,y). \tag{20}$$

If the constant $E = D/2$ the full expression can be re-written as

$$L_{SSIM} = \frac{(2\mu_x\mu_y + C)(\sigma_{xy} + D)}{(\mu_x^2 + \mu_y^2 + C)(\sigma_x^2 + \sigma_y^2 + D)} \tag{21}$$

as suggested by Wang et al. [16].

As the properties of structural similarity are better examined locally than globally Wang et al. [16] proposed a method where the SSIM index is calculated by an 11×11 circular-symmetric Gaussian weighting kernel $\mathbf{w}$ with a standard deviation of 1.5 and the unit sum being swept pixel by pixel over the image. This makes the estimations of μ_x, σ_x and σ_{xy} able to be expressed as

$$\mu_x = \sum_{i=1}^{n} w_i x_i, \tag{22}$$

$$\sigma_x = \left(\sum_{i=1}^{N} w_i (x_i - \mu_x)^2\right)^{1/2}, \tag{23}$$

$$\sigma_{xy} = \sum_{i=1}^{N} w_i (x_i - \mu_x)(y_i - \mu_y). \tag{24}$$

To get a single value for the SSIM index a mean value is calculated from the local SSIM indices.

The SSIM index only works on one dimensional images and can thereby only be applied to gray scale images.

HDR-VDP2

To further improve error metrics to better represent the human visual system, Mantiuk et al. [17] proposed a method for Visual Difference Predictor for High Dynamic Range images (HDR-VPD). The method compares the ground truth image to a reconstruction of the image and predicts both the visual difference between the images as well as the quality loss from the original image.

The HDR-VDP2 is a progression proposed by Mantiuk et al. [18] adapted to find artifacts salient to the human visual system. It is used by Gharbi et al. [5] to mine patches with luminance artifacts. Due to the complexity of the implementation this is not one of the loss methods examined in this thesis. Among the articles encountered in the literature study a trend was to use a more simple loss function. For example Gharbi et al. [5] used L_2 loss for the training even though they had used more advanced error metrics for their patch mining.

2.3.2 Optimization methods

The essence of neural networks is their ability to learn and adapt in order to find an optimal solution to the problem the network is trying to solve. To do this, an optimization method is needed. The optimization method updates the weights of the neural network towards values that improve the result of the network. This section discusses different optimization methods.

Gradient descent optimization

A common method for optimizing a neural network is the gradient descent optimization method [19]. The gradient descent optimization method calculates, as the name suggests, the gradient of the function containing the optimization parameters. The parameters are then updated in the direction where the magnitude of the gradient descend is the steepest. This is optimally iterated until the magnitude of the gradient is close to zero which means that the parameter space has reached a local minimum. In reality the optimization is terminated when the loss is below a threshold value. The parameter θ_t is updated according to

$$\theta_t = \theta_{t-1} - \mu \nabla f(\theta_{t-1}), \tag{25}$$

where μ is the step size for the optimization and f is the function containing the parameters. The downfall of this method is that the whole data set has to be processed to perform one update of the parameters.

Stochastic gradient descent optimization

To improve the optimization the stochastic gradient descent method is able to update the parameters in every iteration according to

$$\theta_t = \theta_{t-1} - \mu \nabla f(\theta_{t-1}; x_i; y_i). \tag{26}$$

This method could improve the convergence rate as opposed to the gradient descent method, but also make it fluctuate more and it may have trouble finding the exact minimum.

Mini-batch optimization

A compromise of these two methods is the mini-batch optimization method. The mini-batch optimization uses the data samples of a small batch and computes the gradient as an average of the gradients

of each data sample according to

$$\theta_t = \theta_{t-1} - \mu \nabla f(\theta_{t-1}; x_{i:i+n}; y_{i:i+n}), \tag{27}$$

where n is the batch size.

Momentum optimization

The stochastic gradient method may have trouble with oscillations which makes it hard to hit the exact minimum. The momentum optimization method allows a fraction of the previous update vector to be a part of the new update vector [20]. This makes the update path less oscillating and the minimum is reached faster [21]. The weight update is done according to

$$v_t = \gamma v_{t-1} + \mu \nabla f(\theta_{t-1}), \tag{28}$$

$$\theta_t = \theta_{t-1} - v_t, \tag{29}$$

where γ is the momentum constant and v_t is the update vector.

ADAM optimization

The ADAM optimization is a development of the momentum optimization and uses an adaptive momentum estimation [22]. This is done by using two momentum constants $\beta_1 \in [0,1]$ and $\beta_2 \in [0,1]$ and two momentum vectors m_t and v_t. The weight update is done according to

$$g_t = \nabla f(\theta), \tag{30}$$

$$m_t = \beta_1 \cdot m_{t-1} + (1-\beta_1) g_t, \tag{31}$$

$$v_t = \beta_2 \cdot v_{t-1} + (1-\beta_2) g_t^2, \tag{32}$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \tag{33}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}, \tag{34}$$

$$\theta_t = \theta_{t-1} - \frac{\alpha \cdot \hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}, \tag{35}$$

where g_t^2 should be interpreted as the element-wise multiplication of the gradients and β_1^t and β_2^t are the constants β_1 and β_2 to the power of t.

The steps of the ADAM optimization can be explained in the following way:

- m_t is the first order momentum and is calculated as the convex combination of the previous first order momentum and the gradients. This can be compared to the momentum, see (28).

- v_t is the second order momentum and is calculated in a similar way as the first order momentum as a convex combination of the previous second order moment and the element wise multiplicated gradients.

- $\hat{m}_t$ and $\hat{v}_t$ are the bias-corrected momentums. These operations are done to decrease the influence of the bias arisen from the initializations of m_t and v_t. Since $\beta < 1$ the influence of this operation will decrease as the number of iterations increases and $\hat{m}_t$ will converge to m_t and $\hat{v}_t$ converges to v_t.

- The parameters θ are then updated using the bias-corrected momentums and the step size α. The term ε is used to prevent division with zero.

Kingma et al. [22] recommends the following constant values: $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\varepsilon = 10^{-8}$. According to both Kingma et al. [22] and Ruder [19] the ADAM optimization method has the best performance for both ordinary and convolutional neural networks and is the preferred optimization algorithm for the authors of several recent articles in the field such as Gharbi et al. [5].

2.3.3 Convolutional neural network

A convolutional neural network is a type of neural network that has one or more convolutional layers. They are inspired by the mammalian vision system, and are mostly used for image processing, video recognition and natural language processing. A convolution is a specialized kind of linear operation. The convolution can be described as

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \quad (36)$$

where a kernel K of size $i \times j$ sweeps over an input image I of size $m \times n$ [23].

A convolutional layer provides advantages such as sparse interactions, parameter sharing and equivariant representations. The sparse interactions are consequences of local connectivity, i.e. of the kernel being smaller than the input, making the limiting of connections for each output possible. Parameter sharing is achieved by using the same parameter for more than one function in a model. The parameter sharing leads to translation equivariance, which means that if the input changes the output changes in the same way. For example a function $f(x)$ is equivariant to a function g if $f(g(x)) = g(f(x))$.

Receptive field

An important concept when discussing neural networks in general and convolutional neural networks in particular is the receptive field. The receptive field for a layer is the part of the input that a single pixel in the output of that layer depends on. In a convolutional neural network, the receptive field of a layer is the spatial dimensions of the filter kernel, e.g. 3×3. The receptive field for the entire network is the region of the input of the network that affects a single pixel in the network output. By making the network deeper, the receptive field is increased linearly.

The effective receptive field is smaller than the theoretical receptive field. Due to the properties of the forward and backward pass in the loss function the gradient of the central pixels of the receptive field has a larger magnitude [24]. The distribution of the impact of the receptive field is Gaussian and decays rather quickly from the center yielding an effective receptive field that is only a small part of the theoretical receptive field. The receptive field for a multilayer perceptron (MLP) is always the entire input due to the full connectivity in each layer.

2.3.4 Fully convolutional neural networks

A development of convolutional neural networks is fully convolutional neural networks. The FCNN has no fully connected layers and use convolutional layers throughout the entire network. This feature makes the network invariant of the input geometry as the network only has to train the variables in the convolutional kernels, which has predefined sizes, and can exclude the bias variables. Another benefit of changing the fully connected layer to a convolutional layer is that the spatial dimension remains unaltered. Most implementations of the fully connected layers throw away the spatial dimensions to achieve matrix multiplications in two dimensions rather than four while convolutional kernels work independently of the matrix shape [25]. This makes it possible to use the information from the spatial correlation throughout the entire network.

Fully connected layers offer a simple way to alter the dimension of the data. As they consist of matrix multiplications it is trivial to modulate the output shape by assigning a suitable shape to the weight matrix. To achieve the same effect in an FCNN, a common operation to use is the 1×1 convolution or a transposed convolution.

The 1×1 convolutions are often used to reduce the channel dimension to avoid computationally heavy convolutions with larger convolutional kernels and a large number of channels. It can also be seen as

a cross channel parametric pooling as the existing channels are mapped to a single output channel for each convolution. This methodology is used in the Inception modules by Szegedy et al. [26] and in the Network in network modules by Lin et al. [27].

Transposed convolution or "deconvolution" is an operation which can be used to upsample a matrix. The convolution is done by separating the input pixels, applying zero padding between them and letting the convolutional kernel sweep over the padded matrix. This can be compared to a bilinear interpolation using trainable weights. The size of the convolutional kernels decides how many values from the previous layer the interpolation should be taken into account.

The upsampling itself is closely related to the demosaicing problem as they both deal with interpolation of missing data. The FCNN has the property of doing a proper upsampling using the spatial correlations whereas the networks using fully connected layers do an expansion of the data, not necessarily using the information from the spatial correlations.

An advantage of the FCNN is the ability to do fully convolutional training, i.e. training on a set of full images. This is is possible due to the invariance of the size of the training data. A fully connected layer requires that the input size is fixed and patch-wise training is therefore the most common practice [25].

A disadvantage of the FCNN is the connectivity. The FCNN works with local connectivity as they connect the pixels in a layer to the surrounding pixels in the previous layer due to the kernel size. The receptive field of a pixel increases as the number of layers are augmented and the receptive field for a deep net with small patches often exceeds the size of the input. The connection between two consecutive layers remains local. A fully connected layer has on the other hand the ability to have a full connectivity as all pixels in a layer can be connected to all pixels in the previous layer.

Another disadvantage of an FCNN is the difficulty to implement bias variables. As the data samples that are processed in the net have undefined size and the bias variables need to have the same spatial dimensions as the training data, an implementation of such variables is impossible. An implementation of a channel-wise bias, i.e. a constant bias for each channel, is possible as the number of channels is rarely undefined.

2.4 Related work

The following section depicts previous work relevant to the goal of demosaicing using neural networks.

Joint denoising and demosaicing

Gharbi et al. [5] introduce a new approach for jointly demosaicing and denoising images while minimizing artifacts such as Moiré and zippering. A deep neural network is trained to demosaic difficult images with luminance artifacts around thin structures, e.g. zippering, and color Moiré artifacts. The method successfully handles complex patterns and generates artifact-free results. The limitation of the method is that it relies on detecting challenging patches for use in training and if ground-truth images already contain artifacts such as Moiré, the network will learn to make images containing artifacts.

Demosaicing using a multilayer neural network

Another method based on neural networks is the work by Wang [28]. A 4×4 patch based multilayer neural network was used for image demosaicing. Compared to state of the art demosaicing algorithms the multilayer neural network handles abrupt color transitions well, however it fails at recovering high frequency patterns. A hypothesis was that using larger patches would make the network recover

better for high frequency patterns.

Linear interpolation

A high quality linear interpolation technique for demosaicing Bayer patterned color filter arrays was developed by Malvar et al. [4]. The method exploits the correlation among the RGB channels and is based on the assumption that the chrominance component in the luminance/chrominance decomposition does not vary much across pixels. The green channel is interpolated bilinearly and the red and blue channels are interpolated such as to maintain a constant hue. Their method leads to an improvement in PSNR of over 5.5 dB compared to bilinear demosaicing. It also outperforms many nonlinear algorithms.

Loss function for image restoration using neural networks

Zhao et al. [29] investigated the loss function for image restorations using neural networks. The focus was on finding a loss function with a perceptually motivated good result. Critique is aimed at the popular L_2 loss which does not produce image quality that correlates with image quality perceived by a human observer. The alternative error metrics such as L_1, SSIM, multi-scale structural similarity index (MS-SSIM) and a combination of L_1 and MS-SSIM are examined to find an alternative to L_2. The error metrics are evaluated using tasks such as image super resolution, JPEG artifacts removal and joint denoising and demosaicing. They found that their own loss function, a combination of L_1 and MS-SSIM, outperformed other loss function.

Demosaicing using artificial neural networks

Kapah et al. [12] early examined demosaicing with the help of artificial neural networks. The perceptron, the backpropagation model, the selector model and the quadratic perceptron model were compared to each other. It was found that the perceptron was good at demosaicing low frequency regions and failed at high frequency regions which contain saturated colors. Contrariwise the backpropagation model was good at demosaicing high frequency regions and at enhancing color contrast although in low frequency areas it failed to reconstruct the correct colors. The selector model was trained to be able to choose when the perceptron and backpropagation model were going to be used to demosaic the region of an image. Consequently high frequency regions could be demosaiced using the backpropagation model and low frequency regions were demosaiced using the perceptron. The last method that was investigated was the quadratic perceptron model which preformed good in both high and low frequency regions and performed the best among all of the examined methods.

Fully convolutional neural networks for semantic segmentation

An implementations of a fully convolutional neural network for segmentation and classification have been proposed by Long et al. [25]. The network implementation emanated from a number of previous implementations such as AlexNet [11] and GoogLeNet [26]. The precursors' network structure was kept except for the fully connected layers that were changed into convolutional layers and the data process was changed to process data of arbitrary size. This enables the classification to be done using heat maps instead of statistical classifications using an output vector. The heat maps showed where in the image it was the most likely to find the detected objects. The efficiency of fully convolutional training to patch-wise training as well as dense output maps and non-spatial output were compared to each other. Three different implementations were

done where the most accurate one combined three different outputs from different levels of the network. The implementations also used a method of forward passes where coarser data was combined with finer data from previous layers. It was found that the FCNN performed better than its ordinary convolutional equivalent with shorter execution time.

3 Method

Different kinds of neural networks were developed to find the answer for the problem statements. This section describes the architecture, variables, hardware and software used to achieve this.

3.1 Implementation

The training was done using a quad-core Intel i7 CPU and an NVIDIA GeForce GTX 1070 GPU. The implementation was written in Python version 3.4.2 with the Tensorflow machine learning library version 0.12.0.

3.2 Data

The training data was taken from the data set of difficult patches from the work of Gharbi et al. [5] and consists of 500 128×128 RGB JPG patches. These patches have been extracted from a large number of sRGB images chosen by the authors due to their low performance in their original settings for their net. The patches are chosen using two criteria: luminance artifacts such as zippering, and Moiré artifacts. The luminance artifacts arise around edges in the image as the demosaicing algorithm takes an average of the pixels over the edge. To find patches yielding luminance artifacts they measured the HDR-VDP2 error and chose images with low scores. HDR-VPD2 is especially suitable for this task as it is specialized in finding perceptual artifacts such as zippering. To find patches with high probability of yielding Moiré artifacs Gharbi et al. [5] created their own error metric specialized in finding such errors. This was done by taking the Fourier transform of the image to find images with high frequencies. As no neural network method is better than its ground truth data it is important to find data with as little bias from previously used demosaicing methods. To avoid this bias Gharbi et al. [5] downsampled their images used as ground truth by a factor 4 using bicubic interpolation.

For the noise reduction part of the thesis, raw images from the Adobe fiveK data set were used [30]. The offset was removed from the raw images and they were white balanced. To create ground truth images, the raw images were bilinearly demosaiced and applied with a Gaussian blur to get better images when applying a downsampling. The images were downsampled to create an oversampling to limit the occurrence of artifacts when demosaicing. For noisy training data a noise was added onto the image at this stage using a sensor noise model. After the downsampling (and added noise for the noisy images), the images were gamma and color corrected and saved with an 8-bit precision to use as ground truth. To get training data, the gamma and color corrected images with and without nosie were subsampled to obtain the mosaic pattern again and also saved with an 8-bit precision.

For the training the input data was saved in $1 \times 1 \times 3$ RGB triplets as ground truth for the patch to pixel network and $32 \times 32 \times 3$ pixel RGB patches as ground truth for the patch to patch network along with the associated $16 \times 16 \times 4$ pixel Bayer patches as training data. The original shape of the Bayer patches was $32 \times 32 \times 1$, but for this implementation each patch is downsampled by a factor of 2 and the data separated in color layers to ensure translational invariance. Since the green pixels are twice as many as the red or blue pixels, they are sampled into two layers. The patches were originally extracted from the original image with a step length of one and the training data consisted of one training sample per pixel in the original image. The data is then normalized to an interval of $[-0.5, 0.5]$ to avoid saturation in the tanh units in the network and to work in the linear region of the function. The data is throughout the training and validation shaped into 4D matrix with

the shape $batchsize \times height \times width \times channels$, e.g. $64 \times 16 \times 16 \times 4$.

3.3 Architecture of the network

This section explains the architecture of the different neural network approaches.

Convolutional neural network

The standard setting of the convolutional neural networks used consists of four convolutional layers and one fully connected layer. All convolutional layers have the same width except the last one which has a width of three layers to match the data to the output channels. The spatial kernel sizes of the filters are equal for all layers and all convolutional layers use zero-padding to maintain the same size of the same spatial dimensions of the data patch throughout the training. The fully connected layer maps the output from the final convolutional layer to an RGB-triplet of desired spatial size. All convolutional layers use parametric ReLU activations on the output and the fully connected layer uses a tanh activation to map the data back to the normalized region.

Patch to pixel approach

As the data dimensionality is to be reduced in the spatial dimensions for the patch to pixel approach the fully connected layer is implemented as a convolution using a filter kernel with the same spatial dimensions as the data patch and no zero-padding. This resulted in an output of a single RGB-triplet.

Patch to patch approach

The patch to patch approach used a fully connected layer using matrix multiplication for calculating the output. The output was an RGB patch with a spatial dimension of $32 \times 32 \times 3$.

Multilayer perceptron

The MLP uses three fully connected layers. As the fully connected layers operate by matrix multiplication the 4D input data has to be reshaped. The spatial dimensions as well as the information about the number of input channels are discarded as the input image is reshaped to a column and channel stacked image, i.e. the data is reshaped in the shape $batchsize \times height \times width \times channels$. The first layer increases the dimension of the data and the last layer reduces the dimension to match the output channels. The first two layers use a dropout layer to avoid overfitting.

Residual learning

Residual learning was applied between the third and the fourth convolution in the patch to patch CNN. The residual layer is a mix of mosaiced data and data from the previous convolutional layer. The 12 channels from the previous convolution are joined into three separate channels to form the RGB representations, see figure 6. These are concatenated below the three upsampled channels of the RGB mosaic channels. A convolution is applied to the residual layer before resuming the regular structure of the network. A simplification of the structure of the network can be seen in figure 5.

Fully convolutional neural network

To investigate the properties of fully convolutional neural networks the performance of a number of different network setups were investigated. The fully convolutional version of the baseline network shares, as stated, many features with its precursor. The networks are identical until the residual layer with four convolutional layers with 16 channels each. The residual layer is replaced with a deconvolutional layer which serves as an upsampling layer to achieve

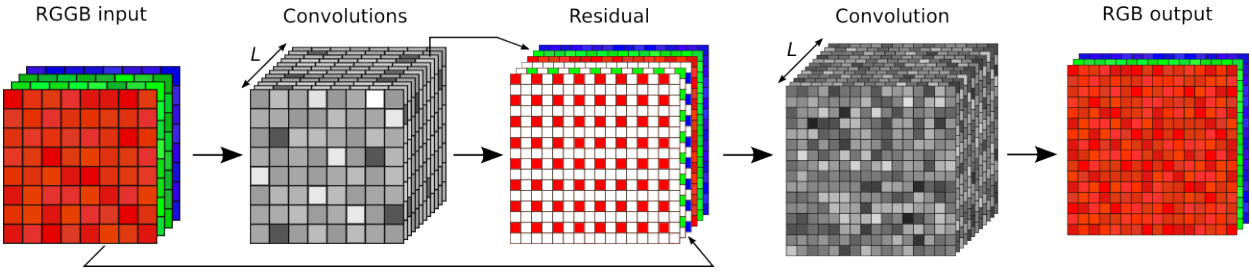


Figure 5: A simplification of the architecture of the patch to pixel neural network. The input is a patch of the mosaiced image, where the different channels have been divided into layers. A set of convolutions are then applied, followed by an optional residual layer. The convolved layers are then arranged into RGB layers and a final full connection results in an RGB pixel. L denotes the number of channels in each convolutional layer.

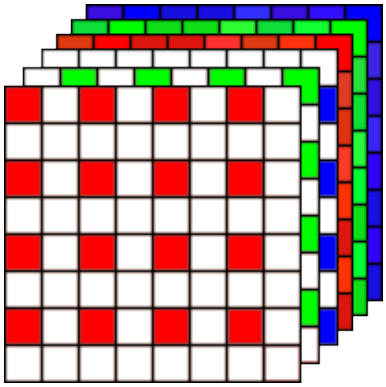


Figure 6: The residual matrix. The first three layers contains the mosaiced RGB patches and the last three layers are the joined color layers from the previous convolutions.

the desired output size as the input is the same downsampled four channel structures. The deconvolutional layer is followed by an additional convolutional layer in full resolution. The output is then produced using a final convolutional layer with a 1×1 convolutional kernel which maps the 16 previous channels to the three output channels.

3.4 Standard setup for the training

The data was processed in batches of 64 samples and trained for 10 epochs. The spatial dimensions for the filters where set to 3×3. All convolutional layers except the last used 16 filter channels. The training was optimized using ADAM optimization with a learning rate with an initial value of $5 \cdot 10^{-4}$. All other variables in the optimization were left as suggested by Kingma et al. [22]. The training aimed to minimize the L_2-loss of the training data and a weight decay of $5 \cdot 10^{-2}$ for the weights of the convolutional layers was used. The learning rate had an exponential decay which decreased with a factor of $0.95^{\frac{x}{s}}$ where x is the batch number and s is the size of the training data. The weights in the convolutional layers

and the fully connected output layer were initiated to be normally distributed with a standard deviation of $\sqrt{2/n} \approx 0.044$, where n is the number of input values per channel (1024), according to He et al. [10]. The biases in both convolutional layers and the fully connected layer were initialized as 0.01 to avoid too many neurons to be throttled in the ReLU activations and thereby excluded in the gradient computations.

3.5 Standard setup for validation

To validate the network an image is preprocessed in the same way as the training data with patch extraction and normalization. The patches are then sent into the network in batches of 64 and the result is saved as an image file. The output from the patch to pixel network is easily saved as the number of pixels in the input image is the same as the number of pixels from the network. In the patch to patch network a Gaussian circular kernel with the same size as the output patch is element-wise multiplied with the output patch. This makes values in the center of the patch more significant in the final evaluation of the image. All patches are then placed in the correct places on top of each other in the resulting image and the sum of all contributing pixel values in every pixel is computed. The quality of the reconstructed image is then quantified by two error metrics, PSNR and SSIM. The validation is done on 5 images for which a mean SSIM index and PSNR value is computed. For all images an error map containing the contrast sensitivity from the SSIM is saved. These error maps highlight parts of the image with salient errors to the human visual system.

4 Results

By comparing different network types and network structures to the baseline network, the performance of different variable and structure changes can be isolated. This section gives an account of these changes and their result. All table measurements are an average out of five validation images. The SSIM measurement in the image texts is for the pictured image, and not the result from the average validation images. The lighthouse image from the Kodak data set, which can be seen in figure 7, is used for comparison since image artifacts are prone to occur in the fence and panels in the image. Comparisons between raw images with and without noise is done using figure 8 from the Adobe fiveK data set.

Figure 7: Kodak lighthouse ground truth image.

Figure 8: Adobe fiveK color corrected ground truth image.

4.1 Establishing properties of the baseline network

To establish the outline of our baseline network we experimented with hyperparameters outside the actual network structure such as output properties and patch size. These investigations were done to be able to discard some of the properties we expected to lower the performance of the network.

Network output

In order to establish which output was most suitable for further investigations a network using pixel output was compared to a network using patch output. To make reasonable comparisons L_2 loss was used for training and validation since it is the only suitable loss for pixel output out of the three examined loss methods. The result can be seen in the table 1 and figure 9. As can be seen the patch output yields better results and therefore further research only uses this approach as the baseline network.

Network output	SSIM	PSNR	Total training time
Patch output	**0.949**	**32.40**	1 h 38 min
Pixel output	0.887	28.85	1 h 22 min

Table 1: Results from using different output structures, measured using different error metrics.

(a) Patch output, SSIM: 0.929 *(b) Pixel output, SSIM: 0.871*

Figure 9: Error maps of demosaiced images using the same network with different output structures.

Network structure

To achieve a comparison of a simpler implementation of a network a four layer perceptron (fully connected network) was used. This was done to confirm that the convolutional neural network structure was more suitable for this kind of task. The comparison to the convolutional neural network can be seen in table 2 and in figure 10. The convolutional neural network yields much better results, therefore the MLP method was discarded.

Network structure	SSIM	PSNR	Total training time
CNN	**0.949**	**32.40**	1 h 38 min
MLP	0.763	25.26	1 h 28 min

Table 2: Results from comparing a convolutional neural network to a multilayer perceptron.

(a) Convolutional neural network, SSIM: 0.929 *(b) Multilayer perceptron, SSIM: 0.763*

Figure 10: Error maps of demosaiced images using different networks structures.

Sampling of training data

The first approach for sampling training patches from the input images used a step length of one, mak-

ing the CFA change over patches, since the Bayer pattern spans over a 2×2 pixel block. To make the Bayer pattern consistent over the patches, a step length of 2 was introduced. This enables the patches to always start at the same color, making the comparison between patches easier. The result from using different step lengths can be seen in table 3 and figure 11.

Sampling step length	SSIM	PSNR	Total training time
1	0.949	32.40	1 h 38 min
2	**0.991**	**34.44**	34 min

Table 3: Results from changing sampling step length.

(a) Sampling step length 1, SSIM: 0.929 *(b) Sampling step length 2, SSIM: 0.990*

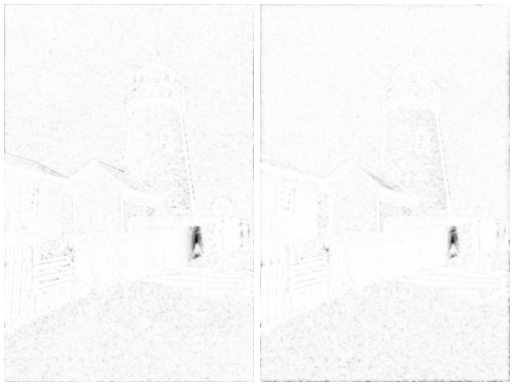
Figure 11: Error maps of demosaiced images using different sampling step lengths.

Input patch size

The first network had 16×16 pixels large patches as input which translates to a $8 \times 8 \times 4$ pixels large input matrix for the network. This approach showed some limitations, the receptive field covers more than the spatial dimensions of the input patch for networks using more than 4 layers (kernel size 3×3) or a kernel size of 4×4 or larger. Because of this the patch size was increased to 32×32 yielding $16 \times 16 \times 4$ pixels large matrices as network input. The comparison can be seen in table 4 and figure 12. The baseline network is a convolutional neural network with overlapping patch output. The training data consists of 32×32 pixels large patches sampled from the original image with a step length of two. Table 4 describes the training time for different network structures using 16×16 and 32×32 patches. This allows to draw some conclusions about the complexity of different model structures.

Patch size	SSIM	PSNR	Total training time
16×16	**0.991**	**34.44**	34 min
32×32	0.989	33.93	1 h 56 min

Table 4: Results from comparing different patch sizes.



(a) 16 × 16 patches, SSIM: 0.990 *(b) 32 × 32 patches, SSIM: 0.988*

Figure 12: Error maps of demosaiced images using different patch sizes.

4.2 Parameters in convolutional neural networks

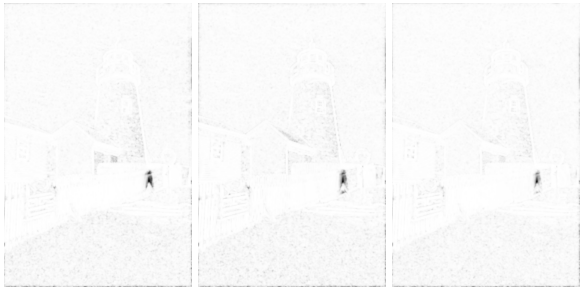
To see what effect different parameter changes had on the result of the network, different parameters were altered, one at a time. The following section depicts the effects of the different parameters choices for the baseline network.

Number of layers

To determine the impact of the number of convolutional layers three different networks were compared using 2, 4 respectively 8 convolutional layers respectively. The networks had 16 channels in each layer. The result can be seen in table 5 and figure 13.

Number of convolutional layers	SSIM	PSNR	Total training time
2	0.989	34.08	1 h 52 min
4	0.989	33.93	1 h 56 min
8	**0.990**	**34.13**	2 h 20 min

Table 5: Results from using different numbers of convolutional layers in the convolutional neural network.



(a) 2 convolutional layers, SSIM: 0.988 *(b) 4 convolutional layers, SSIM: 0.988* *(c) 8 convolutional layers, SSIM: 0.990*

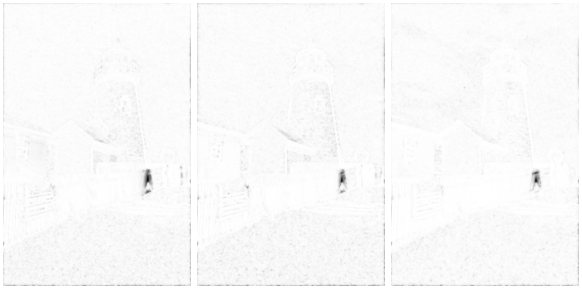
Figure 13: Error maps of demosaiced images using different numbers of layers.

Number of filter channels

To evaluate the impact of the number of filter channels in each layer the result of using 8, 16 respectively 32 channels were compared to each other. The network had 4 convolutional layers. The result can be seen in table 6 and in figure 14. The result shows an improvement by increasing the number of filter channels.

Number of filter channels	SSIM	PSNR	Total training time
8	0.988	33.40	1 h 27 min
16	0.989	33.93	1 h 56 min
32	**0.990**	**34.30**	3 h 3 min

Table 6: Results from using different numbers of filter channels in the convolutional neural network.



(a) 8 filter channels, SSIM: 0.987 *(b) 16 filter channels, SSIM: 0.988* *(c) 32 filter channels, SSIM: 0.989*

Figure 14: Error maps of demosaiced images using different numbers of filter channels.

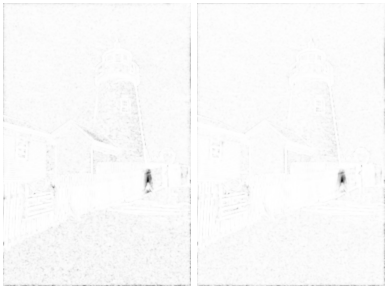
Size of filter kernels

To examine the behavior of the network when the kernel size was altered the network was trained using a 5×5 kernel in all convolutional layers. All filter

kernels used 16 channels. The results can be seen in table 7 and figure 15.

Kernel shape	SSIM	PSNR	Total training time
3×3	0.989	33.93	1 h 56 min
5×5	**0.990**	**33.96**	2 h 3 min

Table 7: Results for a CNN using different kernel sizes.



(a) 3×3 *kernel, SSIM: 0.988* *(b)* 5×5 *kernel, SSIM: 0.990*

Figure 15: Error maps of demoasiced images from a CNN using different kernel sizes.

Further increased receptive field

To further examine the effects of a receptive field that covers more than the spatial dimension of the input patches a network using both 8 convolutional layers and a kernel size of 5×5 was compared to the baseline network, a CNN with a 5×5 kernel and 4 convolutional layers and a CNN using 8 convolutional layers with a 3×3 kernel. The results can be seen in table 8

Conv layers	Kernel shape	SSIM	PSNR	Tot. training time
4	3×3	0.989	33.93	1 h 56 min
	5×5	**0.990**	33.96	2 h 3 min
8	3×3	**0.990**	34.13	2 h 20 min
	5×5	**0.990**	**34.24**	2 h 48 min

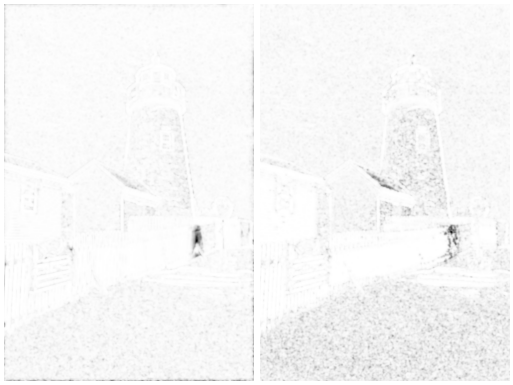
Table 8: Results for a CNN using different receptive fields.

4.3 Classical demosaicing method

As a reference to the neural network method for demosaicing the same validation process was done using the Hamilton Adams method. The results can be seen in table 9 and figure 16. As can be seen the Hamilton Adams method suppresses the Moiré artifact in the fence well but amplifies artifacts in all other areas.

Demosaicing method	SSIM	PSNR	Total training time
CNN	**0.989**	33.93	1 h 56 min
Hamilton Adams	0.979	**37.10**	N/A

Table 9: Results from comparing a convolutional neural network to the Hamilton Adams method.



(a) Convolutional neural network, SSIM: 0.988 *(b) Hamilton Adams, SSIM: 0.979*

Figure 16: Error maps of demosaiced images using different networks structures.

4.4 Loss metrics

To evaluate the impact of the loss metric during training, three different loss metrics were used: L_2, SSIM and PSNR. The result can be seen in table 10, figure 17.

Loss metric	SSIM	PSNR	Total training time
L_2	0.989	33.93	1 h 56 min
SSIM	0.952	31.00	2 h 11 min
PSNR	**0.990**	**34.29**	1 h 57 min

Table 10: Results from using different loss functions for the optimization method.

(a) L_2 loss, SSIM: 0.988 *(b) SSIM loss, SSIM: 0.950* *(c) PSNR loss, SSIM: 0.990*

Figure 17: Error maps of demosaiced images using different loss functions for the optimization method.

4.5 Demosaicing of raw images

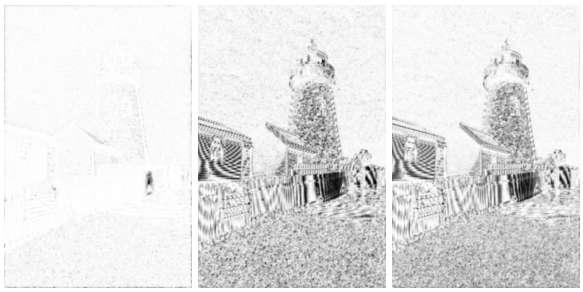
Demosaicing of raw images was investigated in two different ways. It was first compared how well the baseline network, trained on the mined data set, validated using the same five Kodak images versus five raw images and the same five raw images with added noise, see table 11. Secondly, the network was furthermore trained on color and gamma corrected raw images with and without noise and validated using the same procedure. The result can also be seen in table 11. The different images validated on the different networks can be seen in figure 18, 19 and 20. The images were demosaiced using Hamilton Adams as a reference which can be seen in table 12 and figure 21.

Training images	Validation images	SSIM	PSNR	Tot. training time
Mined	Kodak	**0.989**	**33.93**	1 h 56 min
	Raw	0.975	30.3	
	Noisy raw	0.813	27.94	
Raw	Kodak	0.709	23.76	1 h 5 min
	Raw	**0.753**	**24.05**	
	Noisy raw	0.627	23.25	
Noisy raw	Kodak	0.786	**25.86**	1 h 5 min
	Raw	**0.826**	25.77	
	Noisy raw	0.804	25.53	

Table 11: Results from using different validation images to validate a baseline convolutional neural network trained with different images.

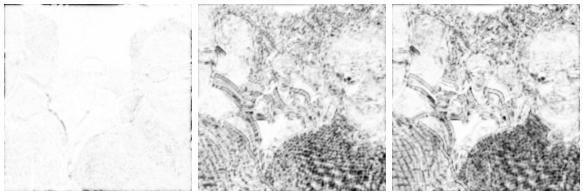
Image	SSIM	PSNR
Kodak lighthouse	**0.990**	34.21
Adobe fiveK raw	0.987	**36.89**
Adobe fiveK noisy raw	0.834	29.37

Table 12: Results from demosaicing images using the Hamilton Adams method.



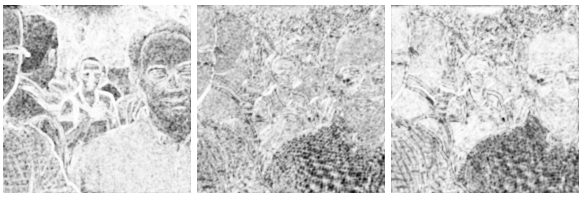
(a) Mined data set, SSIM: 0.988 *(b) Raw data set, SSIM: 0.685* *(c) Noisy raw data, set SSIM: 0.763*

Figure 18: Error maps of the demosaiced Kodak image using networks trained with different image data sets.



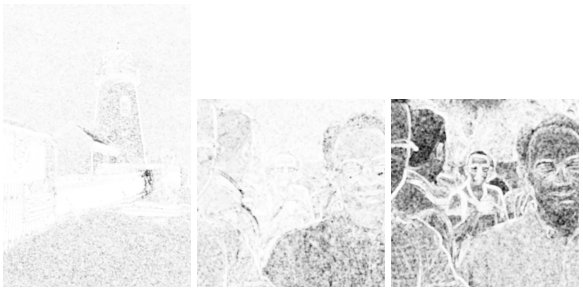
(a) Mined data set, SSIM: 0.987 *(b) Raw data set, SSIM: 0.693* *(c) Noisy raw data set, SSIM: 0.778*

Figure 19: Error maps of the demosaiced raw Adobe fiveK image using networks trained with different image data sets.



(a) Mined data set, SSIM: 0.848 *(b) Raw data set, SSIM: 0.595* *(c) Noisy raw data set, SSIM: 0.757*

Figure 20: Error maps of the demosaiced noisy raw Adobe fiveK image using networks trained with different image data sets.



(a) Kodak lighthouse, SSIM: 0.990 *(b) Adobe fiveK raw, SSIM: 0.987* *(c) Adobe fiveK noisy raw, SSIM: 0.834*

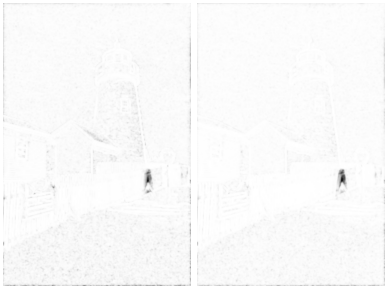
Figure 21: Demosaiced images using Hamilton Adams.

4.6 Residual layer

To investigate the impact of the residual layer a network using the residual layer was compared to the baseline network. The results can be seen in table 13 and figure 22.

Residual layer	SSIM	PSNR	Total training time
Without residual layer	0.989	33.93	1 h 56 min
With residual layer	**0.990**	34.23	5 h 36 min

Table 13: Results from using a CNN with and without a residual layer.



(a) Baseline network, SSIM: 0.988 *(b) Network with residual layer, SSIM: 0.989*

Figure 22: Error maps of demosaiced images using a CNN with and without a residual layer.

4.7 Fully convolutional neural network

To investigate the performance of the fully convolutional neural network the performance of the network was compared to the performance of the original baseline network using a residual layer. The results can be seen in table 14 and figure 23. The following paragraphs presents the results from changing different parameters of the FCNN.

Network structure	SSIM	PSNR	Total training time
CNN	0.989	33.93	1 h 56 min
CNN with residual layer	**0.990**	**34.23**	5 h 36 min
FCNN	0.987	34.00	2 h 45 min

Table 14: Results from comparing an FCNN to other networks.

(a) CNN, SSIM: 0.988 *(b) CNN using residual layer, SSIM: 0.989* *(c) Baseline FCNN*

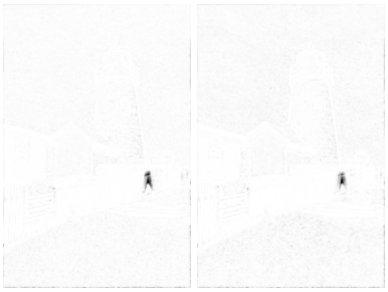
Figure 23: Error maps for a CNN, a CNN with a residual layer and an FCNN.

Size of filter kernels

To examine the behavior of the network when the kernel size was altered the network was trained using a 5×5 kernel in all convolutional layers to compare with the baseline network that has a 3×3 kernel. The results can be seen in table 15 and figure 24.

Kernel shape	SSIM	PSNR	Total training time
3×3	0.987	34.00	2 h 45 min
5×5	**0.989**	**34.17**	2 h 53 min

Table 15: Results from using different kernel sizes for an FCNN.



(a) 3×3 *kernel* *(b)* 5×5 *kernel*

Figure 24: Error maps for an FCNN using different kernel sizes.

Number of filter channels

To examine the performance due to number of filter channels, the width of the network, a version of the network using 8 filter channels and a version using 32 filter channels were used. The results can be seen in table 16 and figure 25.

Channels	SSIM	PSNR	Total training time
8	**0.989**	33.53	2 h 17 min
16	0.987	34.00	2 h 45 min
32	**0.989**	**34.18**	3 h 22 min

Table 16: Results from using different numbers of filter channels.

(a) 8 channels, SSIM: 0.985 *(b) 16 channels, SSIM: 0.990* *(c) 32 channels, SSIM: 0.989*

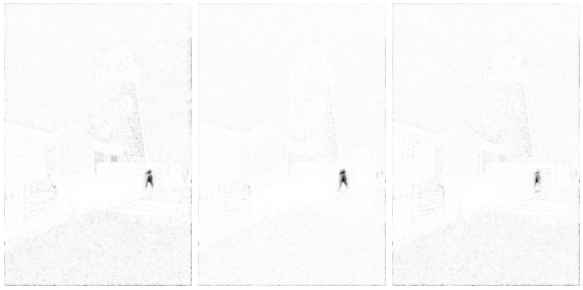
Figure 25: Error maps for an FCNN using different numbers of filter channels.

Number of layers

To examine the effects of the number of layers in the network versions using 2 and 8 convolutional layers were compared to the baseline version. The results can bee seen in table 17 and figure 26.

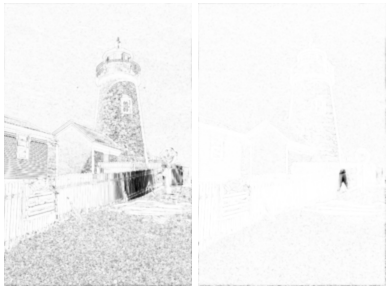
Layers	SSIM	PSNR	Total training time
2	0.986	33.51	2 h 19 min
4	**0.987**	**34.00**	2 h 45 min
8	**0.987**	33.91	3 h 21 min

Table 17: Results from using different numbers of convolutional layers.



(a) 2 layers, SSIM: 0.984 *(b) 4 layers, SSIM: 0.990* *(c) 8 layers, SSIM: 0.989*

Figure 26: Error maps for an FCNN using different number of convolutional layers.



(a) Residual layer *(b) Deconvolutional layer*

Figure 27: Error maps from using different upsampling methods.

Upsampling layer

To investigate the effect of the performance due to the upsampling layer a comparison was made between an FCNN with a deconvolutional layer in place of a residual layer and an FCNN with a residual layer. The results can be seen in table 18 and figure 27.

Upsampling layer	SSIM	PSNR	Total training time
With residual layer	0.914	29.14	3 h 28 min
With deconvolutional layer	**0.987**	**34.00**	2 h 45 min

Table 18: Results from using different upsampling layers for an FCNN.

Placement of upsampling layer

To further examine the performance due to the upsampling layer two networks with different placements of the upsampling layer were compared. The baseline network, which has the upsampling layer preceeding the last convolutional layer (excluding the output layer), was compared to a network where the upsampling layer was succeeding the first convolutional layer, yielding four convolutional layers (excluding output layer) in full resolution. The upsampling was done using deconvolution. The results can be seen in table 19.

Placement of upsampling layer	SSIM	PSNR	Total training time
Early	**0.988**	**34.02**	4 h 0 min
Late	0.987	34.00	2 h 45 min

Table 19: Results for using different placements of the upsampling layer in an FCNN.

Output layers

To examine the effects due to the output layer, three different output layers were tested; a convolutional layer with a 3×3 convolutional kernel, a convolutional layer using a 1×1 kernel (channel wise parametric pooling) and a fully connected layer. The results can be seen in table 20 and figure 28.

Output layer	SSIM	PSNR	Total training time
3×3 convolution	0.986	33.97	3 h 7 min
Fully connected	0.949	31.11	4 h 55 min
1×1 convolution	**0.987**	**34.00**	2 h 45 min

Table 20: Results from using different output layers for an FCNN.

(a) 3×3 *convolution* *(b) Fully connected layer* *(c)* 1×1 *convolution*

Figure 28: Error maps of demosaiced images using different output layers for an FCNN.

5 Discussion

This section discusses the results of the different network structures and configuration changes. It also discusses the possibilities to use neural networks for denoising and the differences in properties for ordinary neural networks and fully convolutional neural networks.

5.1 Property changes in the convolutional network structure

Both outer and inner parameters affected the network in different ways. This section discusses the effect of property changes of the baseline network.

Outer configurations of the network

The patch output format gives a better result than the pixel output. This seems reasonable since every pixel gets multiple suggested output values in the patch approach. The output can draw benefits from using both spatial correlation to estimate the pixel value and the large number of output values for all pixels. As the number of suggested values for every pixel is large (1024) outliers have little impact on the final result. The results from the patch network can also draw benefit from the weighting function as the accuracy of the output pixel value decreases as you approach the border of the patch.

A convolutional neural network produces a better result than a multilayer perceptron. This could also be predicted as the convolutional neural network has a more complex structure than the multilayer perceptron and is thereby able to learn more complex structures, even though the MLP has more trainable weights. It was however an important part of the project to rule out the MLP as a promising approach to continue working with.

The sampling step length is the property that makes the largest difference in the result. This is however not surprising as the validation and training is done on input data with a consistent CFA pattern. This allows the network to draw conclusions using the correlation between between the color channels recognizing specific correlations between specific color channels and using these conclusions in an efficient way in the validation.

A good patch size is a trade-off between an improved result and an increase in the complexity of the calculations. Increasing the input patch size from 16×16 to 32×32 increased the image quality of the demosaiced image considerably when using a sampling step length of 1. When changing the sampling step length to 2, the change was not as dramatic. The input patch size of 16×16 showed a better result according to the SSIM and PSNR measurements, although when comparing the demosaiced images image artifacts appears to be less prominent in the image demosaiced using 32×32 patches. This could be explained by Wang's [28] hypothesis that larger patches would make a multilayer neural network recover better for high frequency patterns, limiting image artifacts. Therefore continued development was done using a patch size of 32×32. The change from a smaller patch size to a larger patch size also made the difference between other parameter changes less prominent as measured by the error metric and on the current validation set.

An increase in patch size increased the training time. As can be seen in table 2, an MLP slows down the training session considerably. This is due to the high numbers of weights in the fully connected layers. This is most prominent when comparing the training time of an MLP using 16×16 patches and the same network training on 32×32 patches. This is expected as the weight matrices need to contain four times as many weights and four times as many multiplications are needed in the case with 32×32 patches compared to the case with 16×16 patches.

When using an FCNN, which does not contain any fully connected layers and thus takes better advantage of parameter sharing, the difference in training time is not prominent at all.

Network depth

The depth of the network (the number of operation layers) is of importance for the result even though the changes are small. As can be seen in table 5 the results for the PSNR measurement increase slightly with the depth of the net while the SSIM remains more or less unchanged. The changes between four and eight layers are small and as the training time increases with 21% this appears to be an unnecessary change. When looking at the images in figure 13 it appears as if the network with 8 convolutional layers gives an image with less artifacts in the fence. The reduction of the aliasing artifact is probably due to the deeper network being able to process the data additionally with its additional layers and can therefore use the increased abstraction level to draw more conclusions. A deep network with a large kernel size makes the receptive field cover more than the spatial dimension of the input patch. This makes the network process data in the padded regions instead of expanding its receptive field. This can result in border effects, which can be seen in figure 13c. The effective receptive field can, however, be expected to not exceed the input patch. To benefit more from a deeper network the patch size needs to be increased.

Network width

The width of the network is also of importance for the result and the results are more apparent than in the case with the number of layers as can be seen in table 6. This is expected as the complexity increases and is well confirmed in the literature such as by Gharbi et al. [5]. The increased number of filter channels allows the network to express more features in each convolutional layer which yields a better result. The execution time is also increased as the complexity of the network increases which is expected. In contrary to the case with network depth the number of filter channels in each layer could be increased to almost any number only limited by the memory capacity for storing the weights and a reasonable training time.

Size of filter kernels

The network using larger filter kernels yields better results than the network using smaller kernels as the network can draw conclusions from larger spatial regions in each step and get a larger receptive field in the whole network. Due to the low number of convolutional layers in the network the receptive field does not enclose more than the spatial dimensions of the input patch even though the kernel size is larger. This allows us to be certain that any edge effects is not due to that the receptive field is exceeding the size of the input patches.

Receptive field

The network seems to benefit from an increased receptive field whether the increase in the result is due to a deeper network, larger kernel size or both. The largest increase in the result seems to be due to a deeper network. As the best result is given from the largest receptive field it is reasonable to assume that the effective receptive field is not exceeding the size of the input patches.

Loss function

The loss function that yields the best result during validation is the PSNR. The small improvement from the L_2 loss function can probably be derived from the logarithmic scale in the PSNR which gives the loss function ability to punish small errors as well

as larger ones. Worth noting is that neither of these functions manage to suppress the aliasing artifact in the fence as both of them only examine pixel values and neglect the correlation between pixels. The SSIM loss, on the other hand, does examine the correlation between pixels. It does however not examine frequencies in the image and can therefore not suppress the aliasing. Another property of the SSIM metric is that it only works on gray scale images. This makes it hard for the loss function to get a good estimate of the color rendering, something that can be said to be of great importance for demosaicing. These effects were very visible when using a step length of one in the sampling due to the dissimilarity in the patch structure. SSIM is however a good metric for validation as a complement to the PSNR metric due to the usage of correlation between pixels.

Residual layer

The result indicates that the convolutional network shows tendencies to benefit from the residual layer. This is due to the additional forward-pass of the input data which allows the net to learn from the residuals of the network.

Reference method

The performance of our method can be considered to be in the same region as the performance of the Hamilton Adams method. The features of the performance however differs. The Hamilton Adams method can more efficiently remove the artifact in the fence due to the contribution from the edge detection algorithm. It does however amplify the artifacts in the images whereas our method successfully reduce them. The Hamilton Adams also have the benefit of a low computational cost in comparison to our neural network method.

5.2 Demosaicing raw images

It was possible to demosaic raw images with the baseline neural network trained on artificially mosaiced mined images. The baseline network was however not able to remove any noise from the noisy raw images since the network was not trained to remove noise by getting noisy input that has a noiseless ground truth.

When demosaicing using a network trained on raw Adobe fiveK images, the validated images ended up with a lot of edge artifacts. This is probably due to the network not being trained on a data set with difficult patches. I.e. the concept of patch mining seems like an important strategy for obtaining good results. This network did not remove any noise from the noisy raw images.

When training and validating on raw images without color and gamma correction the network did not produce any usable results. This is due to the network only being optimized for 8-bit images. Therefor using 12-bit images as input, which the non color and gamma corrected raw images were, would not produce good results.

The network that trained on noisy raw images performed better than the network trained on raw images without noise. The validated images contained less edge artifacts although the images appeared to be smoothed. When validating using the noisy raw images, a lot of noise was reduced. The images did not contain as much edge artifacts which can be a consequence of the network being prone to smoothing the images to remove the noise. The network that trained on noisy raw images was the network that removed the most noise. The Hamilton Adams demosaicing algorithm only amplified the noise.

5.3 Fully convolutional neural networks

The fully convolutional version of our baseline network appears to have worse performance than its ordinary convolutional counterpart. This could imply that the fully connected layer plays an important role in the demosaicing process and that full connectivity for the processed patches at some point in the network is important. Another possible interpretation is that the large number of filter weights in a fully connected layer contribute to enhance the performance. The network shows the same tendencies as the ordinary convolutional network when it comes to number of channels, depth of the network and the kernel size. When examining the differences due to the different upsampling layers it is obvious that the deconvolution layer works much better than the residual layer in the fully convolutional neural network. There is also a huge improvement in the speed of the training. This implies that it might be the upsampling itself rather than the actual operation that enhances the performance of the network. The upsampling allows the final convolution to be done in full resolution which appears to be beneficial. It appears that the placement of the upsampling layer does not affect the performance significantly even though a placement which allows more convolutions to be done in full resolution seems beneficial. Neither is the size of the receptive field altered. The training time is thus increased as more convolutions have to be done in full resolution. The performance is furthermore improved when the final convolutional layer is replaced with a channel-wise parametric pooling layer, implemented as a 1×1 convolutional layer. As can be seen in table 20 the fully connected layer gives an extended training time. The reason for this is the full resolution in the final layer. In comparison to the ordinary neural network the number of filter weights, and there by the number of computations, is increased by a factor 4.

5.4 Demosaicing demosaiced images

A large part of this thesis has been based on training and validating on artificially created mosaiced data from images that have already been demosaiced. It is an easy way to get both training data and ground truth data. The downside is that the neural network can learn to mimic the demosaicing algorithm used in the first place to create the ground truth image. It is also unknown which pixels in the image are from the original sensor data. To lessen the effect of this the ground truth image can be downsampled or blurred.

The Kodak data set is in itself problematic for the demosaicing process since it contains scanned high resolution photos. The Kodak image data set is popular to use when benchmarking demosaicing algorithms, which enables comparisons between research papers. The problem with this is that the algorithms often perform badly when demosaicing real raw image sensor data, making the point of developing demosaicing algorithms void when they do not work on the real world problem.

6 Further work

To further improve the methods stated in this report other associated areas have to be investigated. This section describes possible further work concerning such areas that might be interesting to investigate.

Shape of filter kernels

During the thesis quadratic filter kernels have been used during convolution. Sun et al. [31] suggests that using filter kernels with a hexagonal shape can be beneficial for the image classification task. It can be argued that using filter kernels with different shapes could help remove image artifacts in demosaiced images. It would be interesting to investigate this further.

Lower complexity

Deep neural networks can be computationally heavy, which can be limiting to some areas of usage. Some processors do not have the capacity to handle large numbers of data in a reasonable time frame, like a camera demosaicing an image in real time. To get demosaicing of images with the help of neural networks relevant to low capacity processors the neural network needs to be less complex. Deep neural networks do unfortunately not reach up to this goal. A solution to this could be to train shallow neural networks to be able to achieve similar results as the deeper neural networks. Lei et al. [32] has shown that a shallow network cannot achieve as good results as a deep network when training on the same data. However a shallow network can be trained to mimic the deeper network and therefore achieve the same accuracy as the deeper network.

Color precision on raw images

Training the baseline neural network with raw images without gamma correction was not possible since our implementation of the network is limited to 8-bit color image input. The raw images had a color precision of 12 bits. Since the demosaicing process can occur before the gamma and color correction in the image processing pipeline the neural network could further be improved to handle images with a 12-bit precision to be able to take raw images without gamma and color correction as input.

Customized loss function

As can be seen throughout the results our method causes Moiré artifacts due to aliasing. A reason for this behavior might be that it is not suppressed enough by the loss function in the optimization. To improve the result a loss method that operates in the the frequency domain and suppresses high frequent artifacts would be more adequate. This needs to be combined with a method that ensures a good color rendering in other parts of the image as well. Designing such a method did not fit within the scope of this thesis but would surely be a good development and addition to our method.

Image mining

To increase the accuracy of the network on raw images a future implementation could be to mine challenging raw images. This could make the network able to train and learn from images that are suited for eliminating artifacts and noise that is undesirable in the output image.

Ground truth

To get a network that performs well and does not mimic any previous demosaicing methods present in ground truth images it would be ideal to use ground

truth data that has not been produced by demosaicing. This could be done by using for example three charge-coupled devices or some other method that captures three colors per pixel.

Edge effects

When inspecting the edges of the resulting images closely, salient effects of the convolutional operations are visible in the shape of fading edges. This is a result of the lack of information of the pixel values outside the image. To avoid these effects some kind of filtering could be applied to the edges. Another solution would be to train the network using more edge data allowing the network to learn to process the edges with good results. A simpler solution would of course be to crop the images to remove all edge effects, but this is not recommended.

7 Conclusions

Using convolutional neural networks is a valid method for demosaicing images with good results and it could replace a method using linear interpolation. Our CNN method outperforms the multilayer perceptron by a difference of 7.14 dB in the peak signal to noise ratio. The convolutional neural network performs well when using L_2 and PSNR as loss functions when training the network, however SSIM does not perform as well. Despite the relatively good result the network would benefit from using an error metric that is better at indicating the presence of image artifacts and color errors. The network did not significantly benefit from the residual layer nor a deconvolution layer. The benefit these layers added is believed to have come from their ability to upsample the data. The FCNN did not perform as well as the CNN which implies that fully connected layers are beneficial for the demosaicing process. Even though the demosaicing problem is a local problem it appears as the network benefits from a larger connectivity at some level. The convolutional neural network was able to demosaic raw images but only removed noise from images when trained on images containing noise. It would be especially interesting to continue to improve the networks ability to demosaic raw images with and without noise that has not been color or gamma corrected to be able to incorporate neural networks in the image processing pipeline. As the convolutional neural network methods are much more computationally heavy than a linear interpolation method a combination of demosaicing and e.g. denoising should be included in the neural network method to make it worth replacing a classical method.

The convolutional neural network performed on average a peak signal to noise ratio of 34 dB. The consensus of changing different parameters of the network was that small networks with less complex operations, e.g. convolutions, kernel channels etc., did not differ in the results much compared to deeper networks with more complex operations more than that the training time drastically increased.

References

[1] J. E. Adams Jr and J. F. Hamilton Jr, "Adaptive color plane interpolation in single sensor color electronic camera," July 29 1997. US Patent 5,652,621.

[2] B. E. Bayer, "Color imaging array," July 20 1976. US Patent 3,971,065.

[3] X. Li, B. Gunturk, and L. Zhang, "Image demosaicing: A systematic survey," in *Electronic Imaging 2008*, pp. 68221J–68221J, International Society for Optics and Photonics, 2008.

[4] H. S. Malvar, L.-w. He, and R. Cutler, "High-quality linear interpolation for demosaicing of bayer-patterned color images," in *Acoustics, Speech, and Signal Processing, 2004. Proceedings.(ICASSP'04). IEEE International Conference on*, vol. 3, pp. iii–485, IEEE, 2004.

[5] M. Gharbi, G. Chaurasia, S. Paris, and F. Durand, "Deep joint demosaicking and denoising," *ACM Trans. Graph.*, vol. 35, pp. 191:1–191:12, Nov. 2016.

[6] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, *et al.*, "Photo-realistic single image super-resolution using a generative adversarial network," *arXiv preprint arXiv:1609.04802*, 2016.

[7] J. R. Janesick, *Photon Transfer DN* $\rightarrow \lambda$. SPIE Press Book, SPIE, 2007.

[8] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[9] I. Basheer and M. Hajmeer, "Artificial neural networks: fundamentals, computing, design, and application," *Journal of microbiological methods*, vol. 43, no. 1, pp. 3–31, 2000.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[12] O. Kapah and H. Z. Hel-Or, "Demosaicing using artificial neural networks," in *PROC SPIE INT SOC OPT ENG*, vol. 3962, pp. 112–120, 2000.

[13] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting.," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[14] H. Zhao, O. Gallo, I. Frosio, and J. Kautz, "Loss functions for image restoration with neural networks," *IEEE Transactions on Computational Imaging*, 2016.

[15] L. Zhang, L. Zhang, X. Mou, and D. Zhang, "A comprehensive evaluation of full reference image quality assessment algorithms," in *Image Processing (ICIP), 2012 19th IEEE International Conference on*, pp. 1477–1480, IEEE, 2012.

[16] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from

error visibility to structural similarity," *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.

[17] R. Mantiuk, K. Myszkowski, and H.-P. Seidel, "Visible difference predicator for high dynamic range images," in *Systems, Man and Cybernetics, 2004 IEEE International Conference on*, vol. 3, pp. 2763–2769, IEEE, 2004.

[18] R. Mantiuk, K. J. Kim, A. G. Rempel, and W. Heidrich, "Hdr-vdp-2: a calibrated visual metric for visibility and quality predictions in all luminance conditions," in *ACM Transactions on Graphics (TOG)*, vol. 30, p. 40, ACM, 2011.

[19] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[20] B. T. Polyak, "Some methods of speeding up the convergence of iteration methods," *USSR Computational Mathematics and Mathematical Physics*, vol. 4, no. 5, pp. 1–17, 1964.

[21] I. Sutskever, J. Martens, G. E. Dahl, and G. E. Hinton, "On the importance of initialization and momentum in deep learning.," *ICML (3)*, vol. 28, pp. 1139–1147, 2013.

[22] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[23] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[24] W. Luo, Y. Li, R. Urtasun, and R. Zemel, "Understanding the effective receptive field in deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, pp. 4898–4906, 2016.

[25] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3431–3440, 2015.

[26] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9, 2015.

[27] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.

[28] Y.-Q. Wang, "A multilayer neural network for image demosaicking," in *Image Processing (ICIP), 2014 IEEE International Conference on*, pp. 1852–1856, IEEE, 2014.

[29] H. Zhao, O. Gallo, I. Frosio, and J. Kautz, "Loss functions for image restoration with neural networks," *IEEE Transactions on Computational Imaging*, vol. 3, pp. 47–57, March 2017.

[30] V. Bychkovsky, S. Paris, E. Chan, and F. Durand, "Learning photographic global tonal adjustment with a database of input / output image pairs," in *The Twenty-Fourth IEEE Conference on Computer Vision and Pattern Recognition*, 2011.

[31] Z. Sun, M. Ozay, and T. Okatani, *Design of Kernels in Convolutional Neural Networks for Image Classification*, pp. 51–66. Cham: Springer International Publishing, 2016.

[32] J. Ba and R. Caruana, "Do deep nets really need to be deep?," in *Advances in Neural Information Processing Systems 27* (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds.), pp. 2654–2662, Curran Associates, Inc., 2014.

A Images

(a) Patch output, SSIM: 0.929 *(b) Pixel output, SSIM: 0.871*

Figure 29: Demosaiced images with different output structures

(a) Convolutional neural network, SSIM: 0.929 *(b) Fully connected network, SSIM: 0.763*

Figure 30: Demosaiced images with different network structures

(a) Sampling step length 1, SSIM: 0.929 *(b) Sampling step length 2, SSIM: 0.990*

Figure 31: Demosaiced images using different sampling step lengths.

(a) 16×16 *patches, SSIM: 0.990*

(b) 32×32 *patches, SSIM: 0.988*

Figure 32: Demosaiced images using different patch sizes.

(a) 2 convolutional layers, SSIM: 0.988 *(b) 4 convolutional layers, SSIM: 0.988* *(c) 8 convolutional layers, SSIM: 0.990*

Figure 33: Demosaiced images using different numbers of layers

(a) 8 kernel filters, SSIM: 0.987 *(b) 16 kernel filters, SSIM: 0.988* *(c) 32 kernel filters, SSIM: 0.989*

Figure 34: Demosaiced images using different kernel filter amounts

(a) 3×3 *kernel, SSIM: 0.988* *(b)* 5×5 *kernel, SSIM: 0.990*

Figure 35: Demosaiced images from a CNN using different kernel sizes

(a) Convolutional neural network, SSIM: 0.988 *(b) Hamilton Adams, SSIM: 0.979*

Figure 36: Demosaiced images using different networks structures.

(a) L_2 loss, SSIM: 0.988 *(b) SSIM loss, SSIM: 0.950* *(c) PSNR loss, SSIM: 0.990*

Figure 37: Demosaiced images using different loss functions for the optimization

(a) Mined data set, SSIM: 0.988 *(b) Raw data set, SSIM: 0.685* *(c) Noisy raw data, set SSIM: 0.763*

Figure 38: Demosaiced Kodak image using networks trained with different image datasets

(a) Mined data set, SSIM: 0.987 *(b) Raw data set, SSIM: 0.693* *(c) Noisy raw data set, SSIM: 0.778*

Figure 39: Demosaiced raw Adobe fiveK image using networks trained with different image datasets

(a) Mined data set, SSIM: 0.848 *(b) Raw data set, SSIM: 0.595* *(c) Noisy raw data set, SSIM: 0.757*

Figure 40: Demosaiced noisy raw Adobe fiveK image using networks trained with different image datasets

(a) Kodak lighthouse, SSIM: 0.990 *(b) Adobe fiveK raw, SSIM: 0.987* *(c) Adobe fiveK noisy raw, SSIM: 0.834*

Figure 41: Demosaiced images using Hamilton Adams method.

(a) Baseline network, SSIM: 0.988 *(b) Network with residual layer, SSIM: 0.989*

Figure 42: Demosaiced images using a CNN with and without a residual layer

(a) CNN, SSIM: 0.988

(b) CNN using residual layer, SSIM: 0.989

(c) Baseline FCNN, SSIM: 0.990

Figure 43: Demosaiced images using a CNN, a CNN with a residual layer and an FCNN

(a) 3×3 *kernel, SSIM: 0.990*

(b) 5×5 *kernel, SSIM: 0.987*

Figure 44: Demosaiced images using an FCNN with different kernel sizes

(a) 8 channels, SSIM: 0.989 *(b) 16 channels, SSIM: 0.990* *(c) 32 channels, SSIM: 0.986*

Figure 45: Demosaiced images using an FCNN with different number of kernel channels

(a) 2 layers, SSIM: 0.984 *(b) 4 layers, SSIM: 0.990* *(c) 8 layers, SSIM: 0.987*

Figure 46: Demosaiced images using an FCNN with different number of convolutional layers

(a) Residual layer, SSIM: 0.987 *(b) Deconvolutional layer, SSIM: 0.990*

Figure 47: Demosaiced images using different upsampling methods

(a) 3×3 *convolution, SSIM: 0.988* *(b) Fully connected layer, SSIM: 0.990* *(c)* 1×1 *convolution, SSIM: 0.990*

Figure 48: Demosaiced images using different output layers