Camera RAW Image Demosaicing Using Modern Neural Networks

Master of Science Thesis
at the Faculty of Mathematics, Computer Science and Statistics
of *Ludwig-Maximilians-Universität München* in cooperation with
ARRI - Arnold & Richter Cine Technik

Author:
Thomas Eingartner

Supervisor:
Dr. Dirk André Deckert (LMU)

Co-Supervisors:
Harald Brendel (ARRI)
Dr.-Ing. Markus Mayer (ARRI)
Dr. Tamara Seybold (ARRI)

June 27, 2022

Contents

List of Symbols . . . v

Introduction . . . 1

1 Regression with CNNs . . . 3
1.1 Supervised learning setting . . . 3
1.1.1 The regression problem . . . 3
1.1.2 Generalisation bounds . . . 3
1.2 Convolutional neural networks . . . 5
1.2.1 Tensor products . . . 6
1.2.2 The convolutional layer . . . 6
1.2.3 Multiple layers . . . 11
1.3 Optimisation of CNNs . . . 12
1.3.1 Gradient decent algorithm for convex functions . . . 12
1.3.2 Adjoint operators . . . 14
1.3.3 Backpropagation for generic neural networks . . . 15
1.3.4 Backpropagation for CNNs . . . 18

2 Demosaicing with CNNs . . . 25
2.1 Demosaicing . . . 25
2.1.1 General formulation . . . 25
2.1.2 The Bayer pattern . . . 26
2.2 Challenges . . . 28
2.2.1 Nyquist–Shannon sampling theorem . . . 28
2.2.2 Aliasing artefacts . . . 32
2.2.3 False colour . . . 34
2.3 Classic debayering strategies . . . 34
2.3.1 Bilinear interpolation . . . 35
2.3.2 Gradient based interpolation . . . 35
2.4 CNN architectures . . . 37
2.4.1 U-Net . . . 37
2.4.2 3-stage CNN . . . 40
2.5 Training . . . 45
2.5.1 Data . . . 45
2.5.2 Hyperparameters . . . 47

3 Generative adversarial networks . . . 49
3.1 Probabilistic formulation . . . 49
3.1.1 The general setting . . . 49
3.1.2 The optimal discriminator . . . 51
3.2 Algorithmical aspects . . . 54
3.2.1 Parametrisation . . . 54
3.2.2 Minimisation of the virtual training criterion . . . 55
3.3 Application for image processing . . . 56
3.3.1 Supervised adversarial training . . . 56
3.3.2 Demosaicing with GANs . . . 57

Epilogue . . . 59

Bibliography . . . 61

List of Symbols

$\mathbb{N}$	natural numbers without 0
$\mathbb{Z}$	integers
$\mathbb{R}$	real numbers
$\mathbb{C}$	complex numbers
$\mathbb{E}[X]$	expected value of X
$\mathbb{E}_{\omega\sim D}[X(\omega)]$	expected value of X with respect to the distribution D
$\mathbb{P}_X$	pushforward measure of the measure $\mathbb{P}$ and the random variable X
$X \sim Y$	X is distributed as Y
$\mu \ll \nu$	absolute continuity of μ with respect to ν
λ	Lebesgue measure
λ^d	d-dimensional Lebesgue measure
$\|A\|$	cardinality of the set A
$A^\complement$	complement of the set A
$[n]$	the set {1, 2, ..., n} for $n \in \mathbb{N}$
$C^0(X)$	space of continuous functions from X to $\mathbb{R}$
$C^0(X, Y)$	space of continuous functions from X to Y
$C^n(X)$	space of n-times differentiable functions from X to $\mathbb{R}$
$C^n(X, Y)$	space of n-times differentiable functions from X to Y
$C^n_c(X)$	space of continuous or n-times differentiable, compactly supported functions from X to $\mathbb{R}$, $n \in \mathbb{N} \cup \{0\}$
$\mathbb{1}_A$	indicator function/characteristic function of the set A
f'	derivative of the univariate function f
$\partial_x f$	partial derivative of f with respect to the variable x
$\partial_x^n f$	n-th partial derivative of f with respect to the variable x
∇f	gradient of f
Df	total derivative of f
$\langle x, y\rangle$	scalar product of x and y
$\|x\|$	euclidean norm of x
$\bar{\alpha}$	complex conjugate of α
Hom(X, Y)	homomorphisms from X to Y
$\mathcal{L}^0(X)$	space of measurable functions from X to $\mathbb{R}$
$L^0(X)$	quotient space of $\mathcal{L}^0(X)$ where functions equaling almost everywhere are identified
$\mathcal{L}^p(X)$	space of p-integrable functions from X to $\mathbb{R}$

$L^p(X)$	quotient space of $\mathcal{L}^p(X)$ where functions equaling almost everywhere are identified
$\mathcal{L}^\infty(X)$	space of functions from X to $\mathbb{R}$ that are bounded almost everywhere
$L^\infty(X)$	quotient space of $\mathcal{L}^\infty(X)$ where functions equaling almost everywhere are identified
$\|f\|_{L^p(X)}$	L^p-norm of f, where $p \in \mathbb{N} \cup \{0, \infty\}$
supp(f)	support of f
argmin(f)	arguments of the minima of f
argmax(f)	arguments of the maxima of f

Introduction

Digital cinema cameras require several processing steps to convert raw sensor data into a usable image. Some critical steps include colour balance, denoising, colour correction and a nonlinear resampling function to reduce bit depth. These algorithms must run in real-time over millions of pixels on a battery powered device, and therefore must be as energy and memory efficient as possible without sacrificing optimal image quality. In the development of new cameras, these algorithms often need to be hand-tuned as a group in order to achieve the goals of quality and efficiency, and results in the image processing chain being scaled in its computing and storage requirements. The optimisation of the individual algorithmic complexities can be very laborious with classical algorithms, ideally the entire processing chain could be optimised in one step. These requirements of utmost quality, full-chain optimisation and efficient algorithms leads naturally to the advantages offered by neural networks. In recent years, the hardware for evaluating neural networks in mobile devices has improved significantly in terms of energy consumption and costs. In addition, neural networks are now capable of efficiently performing complex image calculations. Convolutional neural networks (CNNs) in particular have established themselves for this purpose. To prove the initial concepts and to reduce complexity, this work focuses on the primary element driving image quality, the demosaicing algorithm. We show that CNNs are often superior to classical algorithms in terms of image quality. This lays the foundation for implementing an entire image processing pipeline in a single neural network in a camera in the future.

In order to implement a demosaicing algorithm as a neural network and to find an optimal architecture for this problem, we first investigate the mathematical structure of CNNs and how they can be specifically optimised using a gradient decent algorithm. Secondly, we are discussing the requirements for a demosaicing algorithm by proving the Nyquist-Shannon sampling theorem and investigating classical demosaicing algorithms. For the demosaicing problem, in addition to the U-Net, a 3-stage CNN architecture-, reflecting the algorithmic structure of gradient-based demosaicing algorithms-, is tested. First, this architecture reconstructs the green channel of the image (which has the most luminance information) and then calculates the red and blue channels. Synthetic training data was shown to be best suited for training, and that the quality of this data is crucial for learning success.

One of the hyperparameters to be defined in optimisation problems is the choice of a loss function. Especially in image processing tasks, standard loss functions, such as the mean squared error, do not necessarily reflect the perceptual distance between the source image and its reconstruction. While there are approaches to other image quality metrics in the field of colour science and signal processing theory, this problem leads to the topic of Generative Adversarial Networks (GANs) from the machine learning perspective. GANs are based on the concept that no loss function is being selected manually. Instead, another neural network, which is trained simultaneously, takes over the task of a loss function. We will review the arguments in [13] and provide more detailed proofs that under ideal conditions this interplay of the generator network and the discriminator network converges to an optimal minimum.

1 Regression with CNNs

1.1 Supervised learning setting

1.1.1 The regression problem

Machine learning refers to the task of making predictions using experience based on data. This can mean either searching for a hypothesis that assigns specific, discrete labels to data (classification), or a hypothesis that assigns real or complex values that are as plausible as possible (regression). In this thesis we will focus on the latter. We follow the general machine learning setting [20], except that we consider the problem of regressing multivalued functions instead of scalar functions. Let $\mathcal{X}$ denote a set of *features* and $\mathcal{Y}$ a measurable subset of $\mathbb{R}^d$, $d \in \mathbb{N}$, whose elements we call *labels*. Let $f : \mathcal{X} \longrightarrow \mathcal{Y}$ be a target labeling function. Suppose that f is unknown but we are given a finite set of samples $S = ((x_1, y_1), \dots, (x_m, y_m)) \in (\mathcal{X} \times \mathcal{Y})^m$ such that $y_i = f(x_i)$ for all $i \in [m]$, where throughout this thesis we write $[m] = \{1, 2, \dots, m\}$. The points $x_1, \dots, x_m$ are independently drawn at random from a probability distribution D over $\mathcal{X}$. This is referred to as the *deterministic scenario*. In the more general *stochastic scenario*, both the labels and the targets follow a distribution over $\mathcal{X} \times \mathcal{Y}$.

Given a set of *hypotheses* $H \subseteq \{h \mid h : \mathcal{X} \longrightarrow \mathcal{Y}\}$ the goal of the regression problem is to find a $h \in H$, such that h is a good approximation to f, by using S. For that, a notion of distance between f and candidates from H is needed. A *loss function* $L : \mathcal{Y} \times \mathcal{Y} \longrightarrow \mathbb{R}_+$ is a measurable function that measures the magnitude of the error for a single data point. For example, for $(x_1, y_1) \in \mathcal{X} \times \mathcal{Y}$ the loss of an hypothesis $h \in H$ is $L(h(x_1), y_1)$. If there is an $M > 0$ such that $L(h(x), f(x)) \leq M$ for all $h \in H, x \in \mathcal{X}$, then we say that L is a *bounded loss function* and the regression problem is referred to as a *bounded regression problem*.

1.1.2 Generalisation bounds

In order to compare different candidate hypotheses, we introduce an error that measures how far the values of the tested hypothesis deviate from the actual values. The problem is that in practice this error can only be calculated on the training data, but not over the entire distribution on which the training data is based on. For this reason, only an approximation of the error can be calculated. For this approximation, however, we need an estimate of how far it is from the actual error. This can be calculated without the knowledge of the actual distribution, at least probabilistically. These estimates are called *generalisation bounds* and can be derived with the help of *concentration inequalities*, such as the Hoeffding inequality (see Lemma 1.1).

The *generalisation error* with respect to f for a hypothesis $h \in H$ is defined as

$$R(h) = \mathbb{E}_{x \sim D}[L(h(x), f(x))].$$

In practice, it is usually not possible to calculate the generalisation error as this could mean summing over an infinite set. For that, the *empirical error* is introduced and it is defined as

$$\hat{R}_S(h) = \frac{1}{m} \sum_{i=1}^{m} L(h(x_i), y_i), \quad m \in \mathbb{N}.$$

The fact that we only have $\hat{R}_S$, but not R is one of the main challenges. An algorithm

$$\begin{aligned} \mathcal{A} : (\mathcal{X} \times \mathcal{Y})^m &\longrightarrow H \\ S &\longmapsto h_S \end{aligned} \tag{1.1}$$

to find an optimal hypothesis only gets feedback on the quality of a hypothesis based on $\hat{R}_S$.

Since the values of R and $\hat{R}_S$ may differ, it may happen that $\mathcal{A}$ returns a hypothesis $h \in H$ such that there exists another $h' \in H$ such that $\hat{R}_S(h) < \hat{R}_S(h')$ but $R(h) > R(h')$. This means that although h' would be the better choice, h achieves a lower value in our training criterion. This phenomenon is called *overfitting*. Since $\hat{R}_S$ depends solely on data from S, overfitting means that h probably generalises poorly to previously unseen data from $(\mathcal{X} \times \mathcal{Y}) \setminus \bigcup_{i=1}^{m} (x_i, y_i)$. For this reason it is of interest that the deviation of the values of R and $\hat{R}_S$ is as small as possible. The easiest way to achieve this is to choose m large. The accuracy of $\hat{R}_S$ can be estimated by computing $\hat{R}_{\tilde{S}}$ on an extra data set $\tilde{S}$, the so called *validation set* which is not shown to $\mathcal{A}$, and comparing $\hat{R}_{\tilde{S}}$ with $\hat{R}_S$.

Note that generalisation bounds can be given for both, finite, and infinite hypothesis spaces. Because of the finiteness of computational memory the hypothesis space is always finite in practice. For this reason we restrict to providing a generalisation bound only for the finite case. For that we recall Hoeffding's inequality.

Lemma 1.1 (Hoeffding's inequality) Let $X_1, \dots, X_m$ be independent random variables with X_i taking values in an interval $[a_i, b_i] \subsetneq \mathbb{R}$ almost surely for all $i \in [m]$. Let $\epsilon > 0$, then the following inequalities hold for $S_m = \sum_{i=1}^{m} X_i$:

$$\mathbb{P}[S_m - \mathbb{E}[S_m] \geq \epsilon] \leq e^{-\frac{2\epsilon^2}{\sum_{i=1}^{m}(b_i - a_i)^2}} \tag{1.2}$$

$$\mathbb{P}[S_m - \mathbb{E}[S_m] \leq -\epsilon] \leq e^{-\frac{2\epsilon^2}{\sum_{i=1}^{m}(b_i - a_i)^2}} \tag{1.3}$$

A proof can be found in [20], Theorem D.1. Now we can formulate the learning guarantee for finite hypothesis sets.

Theorem 1.2 (Generalisation bound for finite hypothesis sets) Let L be a loss function bounded by $M > 0$ and assume $|H| < \infty$. Then, for $\delta > 0$, the following inequality holds for all $h \in H$ with probability at least $1 - \delta$:

$$R(h) \leq \hat{R}_S(h) + M \sqrt{\frac{\log |H| + \log \frac{1}{\delta}}{2m}}.$$

Proof. Since L only takes values in $[0, M]$ we get for $h \in H$ by Hoeffding's inequality (inequation (1.2)) that

$$\mathbb{P}\left[R(h) - \hat{R}_S(h) > \epsilon\right] \leq e^{-\frac{2m^2\epsilon^2}{mM^2}} = e^{-\frac{2m\epsilon^2}{M^2}}.$$

Then, by the finiteness of H we get

$$\begin{aligned}\mathbb{P}\left[\exists h \in H \mid R(h) - \hat{R}_s(h) > \epsilon\right] &= \mathbb{P}\left[\bigcup_{h\in H}\{R(h) - \hat{R}_S(h) > \epsilon\}\right] \\ &\leq \sum_{h\in H}\mathbb{P}\left[R(h) - \hat{R}_S(h) > \epsilon\right] \leq |H|e^{-\frac{2m\epsilon^2}{M^2}}.\end{aligned} \tag{1.4}$$

Setting the right-hand side to be equal to δ yields

$$-\frac{2m\epsilon^2}{M^2} = \log\frac{\delta}{|H|} \qquad \Rightarrow \qquad \epsilon = \sqrt{-\frac{M^2\log\frac{\delta}{|H|}}{2m}} = M\sqrt{\frac{\log|H| + \log\frac{1}{\delta}}{2m}}.$$

Thus, considering the complementary event as in 1.4 yields

$$\mathbb{P}\left[\forall h \in H : R(h) - \hat{R}_S(h) \leq M\sqrt{\frac{\log|H| + \log\frac{1}{\delta}}{2m}}\right] \geq 1 - \delta$$ ■

Generalisation bounds give us a notion of how safe our training will lead to success. Nevertheless, they are not sufficient in our case, since we need to have concrete influence on algorithm $\mathcal{A}$ in order to obtain an acceptable hypothesis. In fact, we do not have direct access to the distribution D, but only have empirical training data and a priori knowledge from the experience of classical algorithms that we can use. The importance of the latter is underlined by the no-free-lunch theorem, that states that the learning algorithm will, without incorporating a priori knowledge, not be better than guessing. The choice of the hypothesis space H is very crucial here. It can be parametrised very generally via neural networks. Somewhat specialised, and particularly well suited for image processing, are *convolutional neural networks (CNNs)*. In the next sections we are going to investigate the mathematical structure and training techniques of CNNs in order to be able to choose a good hypothesis space of CNNs.

1.2 Convolutional neural networks

In Section 1.1 we formulated the goal of selecting a hypothesis h from H in such a way that the error $R(h)$ is minimised. This is achieved through optimisation. For that, it is necessary to parametrise H in a way that $R(h)$ can by optimised with respect to the parameters. For applications in which images are to be analysed or processed, it is usually sufficient to restrict H to translation equivariant functions. These can be efficiently represented with so-called *convolutional neural networks (CNNs)*.

In this chapter we want to lay the mathematical foundation to be able to describe precisely 2D deep convolutional networks architectures. For that, we have to fix notation for some preliminary objects such as tensor products and elementwise functions. We will introduce the convolutional operator, which is at the heart of CNNs. In order to be able to use this on image data in a practical way, it is necessary to apply some other supporting operators, such as the padding operator and the cropping operator. In addition, we will define element-wise functions which are needed such that non-linear functions can also be represented.

The composition of the layers just mentioned form the so-called convolutional layer. In order to be able to represent even more complex functions, the composition of several successive convolutional layers can be built. This is called a *deep CNN*.

1.2.1 Tensor products

Often the data to be processed with a CNN consists of two-dimensional arrays with several channels. For example, when processing image data, these can be red, green and blue channels. In addition, the number of channels of the layers is often increased in deep CNNs in order to be able to extract abstract features from the data. For this purpose, it is useful to represent the data as coefficients along an axis that indexes the channels. This can be done using the tensor product.

Definition 1.3 (Tensor product of vector spaces) Let K be a field and V, W be finite dimensional K-vector spaces. Then, the *tensor product* of V and W is a pair $(V \otimes W, \otimes)$ of the *tensor product space* $V \otimes W$ together with a bilinear map $\otimes : V \times W \longrightarrow V \otimes W$, where $V \otimes W$ and $\otimes$ are defined as follows:

- Let $\mathcal{V} = \{v_i \in V \mid i \in I\}$ and $\mathcal{W} = \{w_j \in W \mid j \in J\}$ be bases of V and W. Then we define $V \otimes W$ as the K-vector space with basis $\mathcal{V} \times \mathcal{W} = \{(v_i, w_j) \mid i \in I, j \in J\}$. Elements of $V \otimes W$ are of the form
$$\sum_{(i,j)\in I\times J} c_{ij}(v_i \otimes w_j), \quad c_{ij} \in K.$$
- For $v = \sum_{i\in I} a_i v_i \in V, w = \sum_{j\in J} b_j w_j \in W$ we define the bilinear map $\otimes$ by
$$v \otimes w = \sum_{(i,j)\in I\times J} a_i b_j (v_i \otimes w_j).$$

One can now verify that $V \otimes W$ is indeed a vector space under addition and scalar multiplication of the coefficients and that $\otimes$ is bilinear.

Definition 1.4 (Inner product for direct product- and tensor product spaces) Let K be a field and $V_1, \dots, V_r$ be finite dimensional K-vector spaces. For vectors $v_i, w_i \in V_i, i = 1, \dots, r$ we define the inner products of tuples and tensor products as

$$\langle (v_1, \dots, v_r), (w_1, \dots, w_r) \rangle = \sum_{i=1}^{r} \langle v_i, w_i \rangle,$$
$$\langle v_1 \otimes \dots \otimes v_r, w_1 \otimes \dots \otimes w_r \rangle = \prod_{i=1}^{r} \langle v_i, w_i \rangle.$$

1.2.2 The convolutional layer

We will elaborate 2D convolutional neural networks (2D-CNNs or, throughout this thesis, CNNs) in a mathematical framework. We use a similar notation as presented in [8], Section 4.2 but several definitions have been adapted to be inline with machine learning frameworks like PyTorch, which has been used for the empirical part of this thesis (see Remark 1.12). We begin by describing the actions of a generic layer of a CNN. As input of such a layer regard an element

$$x = \sum_{j=1}^{m_1} x_j \otimes e_j \in \mathbb{R}^{n_1 \times l_1} \otimes \mathbb{R}^{m_1}, \quad m_1, n_1, l_1 \in \mathbb{N},$$

where $\{e_j\}_{j=1}^{m_1}$ denotes an orthonormal basis of $\mathbb{R}^{m_1}$. We say that x is a m_1-*channeled* tensor, where each channel consists of a matrix of size $n_1 \times l_1$. We refer to $x_j \in \mathbb{R}^{n_1\times l_1}$ as a *feature map*. A CNN-Layer can be represented as a parameter-dependent map

$$\mathbb{R}^{n_1\times l_1} \otimes \mathbb{R}^{m_1} \longrightarrow \mathbb{R}^{\hat{n}_1\times \hat{l}_1} \otimes \mathbb{R}^{m_2},$$

where the parameters are of the form

$$W = \sum_{j=1}^{m_2} W_j \otimes \bar{e}_j \in \mathbb{R}^{p\times q\times m_1} \otimes \mathbb{R}^{m_2}, \quad p, q, m_2 \in \mathbb{N};\ p \leq n_1;\ q \leq l_1$$

and $V \in \mathbb{R}^{m_2}$, where $\{\bar{e}_j\}_{j=1}^{m_2}$ denotes an orthonormal basis of $\mathbb{R}^{m_2}$. We say that each $W_j \in \mathbb{R}^{p\times q\times m_1}$ is a *filter* of size $p \times q$ used in the convolution and we call V the *bias*. Then, we can write the function represented by the convolution layer as

$$f : (\mathbb{R}^{n_1\times l_1} \otimes \mathbb{R}^{m_1}) \times \mathbb{R}^{m_2} \times (\mathbb{R}^{p\times q\times m_1} \otimes \mathbb{R}^{m_2}) \longrightarrow \mathbb{R}^{\hat{n}_1\times \hat{l}_1} \otimes \mathbb{R}^{m_2},$$

where $\hat{n}_1, \hat{l}_1 \in \mathbb{N}$ depend on the particular form of f. We will define f as a composition of a cropping operator, a convolution/cross-correlation operator and an element-wise applied non-linearity function. For vector spaces V_1, V_2, V_3 we write Hom(V_1, V_2) to denote the set of linear maps (vector space homomorphisms) from V_1 to V_2 and Hom$(V_1, V_2; V_3)$ to denote set of bilinear maps from $V_1 \times V_2$ to V_3 (maps that are vector space homomorphisms in both components). In the context of CNNs we denote arbitrary but fixed orthogonal bases as $\{E_{j,k}\}_{j,k=1}^{n_1,l_1}$ for $\mathbb{R}^{n_1\times l_1}$, $\{\tilde{E}_{j,k}\}_{j,k=1}^{p,q}$ for $\mathbb{R}^{p\times q}$, $\{\bar{E}_{j,k}\}_{j,k=1}^{n_2,l_2}$ for $\mathbb{R}^{n_2\times l_2}$ and $\{\hat{E}_{j,k}\}_{j,k=1}^{\hat{n}_1,\hat{l}_1}$ for $\mathbb{R}^{\hat{n}_1\times\hat{l}_1}$, where the last space is going to be defined later.

The filters are usually much smaller than the input data to be processed. Convolving the filter means *moving* the filter step by step over the input data and computing the inner product of the filter with the input data it covers. To formulate this mathematically we crop the input data such that its size equals the filter before applying the inner product.

Definition 1.5 (Cropping operator) The *cropping operator* $\mathrm{K}_{k,l} \in$ Hom$(\mathbb{R}^{n_1\times l_1} \otimes \mathbb{R}^{m_1}, \mathbb{R}^{p\times q\times m_1})$ for an input x is defined as

$$\mathrm{K}_{k,l}\left(\sum_{j=1}^{m_1} x_j \otimes e_j\right) = (\kappa_{k,l}(x_j))_{j=1}^{m_1},$$

where we define $\kappa_{k,l} \in$ Hom$(\mathbb{R}^{n_1\times l_1}, \mathbb{R}^{p\times q})$ as

$$\kappa_{k,l}(x_j) := \sum_{s=1}^{p}\sum_{t=1}^{q} \langle x_j, E_{k+s-1,l+t-1}\rangle \tilde{E}_{s,t},$$

for any $k \in [n_1 - p + 1], l \in [l_1 - q + 1]$.

Note, that if $\{E_{j,k}\}_{j,k=1}^{n_1,l_1}$ and $\{\tilde{E}_{j,k}\}_{j,k=1}^{p,q}$ are the standard basis for their respective spaces, i.e.

$$(E_{j,k})_{\mathrm{j},\mathfrak{k}} = \begin{cases} 1, & (j,k) = (\mathrm{j},\mathfrak{k}) \\ 0, & \text{else} \end{cases} \qquad \text{for all } \mathrm{j} \in [n_1], \mathfrak{k} \in [l_1]$$

$$\text{and} \quad (\tilde{E}_{j,k})_{\mathrm{j},\mathfrak{k}} = \begin{cases} 1, & (j,k) = (\mathrm{j},\mathfrak{k}) \\ 0, & \text{else} \end{cases} \qquad \text{for all } \mathrm{j} \in [p], \mathfrak{k} \in [q],$$

then $\kappa_{k,l}(x_j)$ is the $p \times q$ submatrix of x_j containing all elements from (k,l) to $(k+p-1, l+q-1)$.

Definition 1.6 (Convolution operator) The *convolution operator*

$$C \in \mathrm{Hom}(\mathbb{R}^{p\times q\times m_1} \otimes \mathbb{R}^{m_2}, \mathbb{R}^{n_1\times l_1} \otimes \mathbb{R}^{m_1}; \mathbb{R}^{\hat{n}_1\times \hat{l}_1} \otimes \mathbb{R}^{m_2})$$

is defined as

$$C(W,x) := \sum_{j=1}^{m_2} c_j(W,x) \otimes \bar{e}_j,$$

where for $j \in [m_2]$ we define $c_j \in \mathrm{Hom}(\mathbb{R}^{p\times q\times m_1} \otimes \mathbb{R}^{m_2}, \mathbb{R}^{n_1\times l_1} \otimes \mathbb{R}^{m_1}; \mathbb{R}^{\hat{n}_1\times \hat{l}_1})$ as

$$c_j(W,x) := \sum_{k=1}^{\hat{n}_1} \sum_{l=1}^{\hat{l}_1} \langle W_j, \mathrm{K}_{\gamma(k,l,\delta)}(x)\rangle \hat{E}_{k,l}, \tag{1.5}$$

where $W = \sum_{j=1}^{m_2} W_j \otimes \bar{e}_j \in \mathbb{R}^{p\times q\times m_1} \otimes \mathbb{R}^{m_2}$ and

$$\gamma(k,l,\delta) := (1+(k-1)\delta, 1+(l-1)\delta).$$

We call δ the *stride* of the convolution. The numbers $\hat{n}_1, \hat{l}_1 \in \mathbb{N}$ are chosen maximal such that equation (1.5) is well-defined (see Remark 1.7 for explicit formulas for $\hat{n}_1$ and $\hat{l}_1$).

Note that although this operator is referred to as convolution in the machine learning community it is not a discrete convolution in the mathematical sense but a *cross-correlation*.

We can think of the input $x \in \mathbb{R}^{n_1\times l_1} \otimes \mathbb{R}^{m_1}$ as a stack of matrices, where each matrix can be thought of an image-like array, which consists of multiple channels (or features) of that image. In this sense, the input can be regarded as an object in 3D-space. Each filter $W_j \in \mathbb{R}^{p\times q\times m_1}$ convolves with the whole stack of images. This means that the convolution actually operates along all three axes. However, the operation is referred to as 2D-convolution since the input- and output data is thought of features of 2D-images. In this sense, the term 2D-convolution refers to the number of degrees of freedom in which the filter can move.

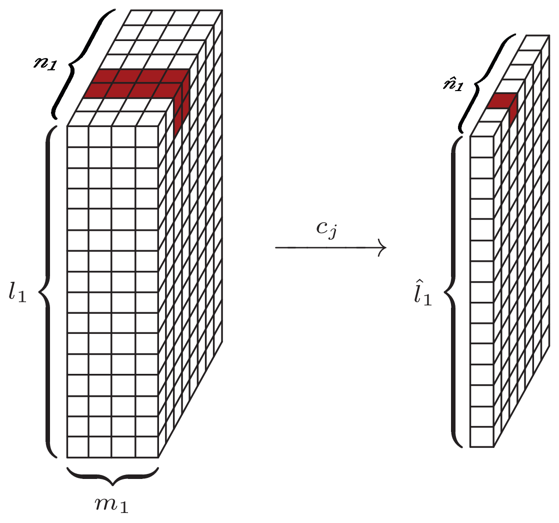


Figure 1.1: Output component of the convolutional operator for $n_1, l_1 = (8,16)$, $m_1 = 4$, $(p,q) = (2,2)$ and $\delta = 1$

Remark 1.7 (Explicit Formulas for $\hat{n}_1$ and $\hat{l}_1$) In order to compute the output dimensions $\hat{n}_1, \hat{l}_1 \in \mathbb{N}$ of the convolutional operator for an input $x \in \mathbb{R}^{n_1 \times l_1} \otimes \mathbb{R}^{m_1}$, filters $W \in \mathbb{R}^{p \times q \times m_1} \otimes \mathbb{R}^{m_2}$ and stride $\delta \in \mathbb{N}$ we observe in view of the definition of the cropping operator that $\hat{n}_1, \hat{l}_1 \in \mathbb{N}$ are chosen maximal such that the condition

$$\gamma(\hat{n}_1, \hat{l}_1, \delta) \in [n_1 - p + 1] \times [l_1 - q + 1]$$

is fulfilled, where γ is defined as in Definition 1.6. Regarding only the first component, this is equivalent to

$$\hat{n}_1 = \max\{k \in \mathbb{N} \mid 1 + (k-1)\delta \leq n_1 - p + 1\} = \max\left\{k \in \mathbb{N} \,\middle|\, k \leq \frac{n_1 - p}{\delta} + 1\right\}.$$

The analogous computation applies for $\hat{l}_1$ and we can conclude

$$\hat{n}_1 = \left\lfloor \frac{n_1 - p}{\delta} \right\rfloor + 1 \quad \text{and} \quad \hat{l}_1 = \left\lfloor \frac{l_1 - q}{\delta} \right\rfloor + 1. \tag{1.6}$$

In many applications of CNNs, such as image to image translation tasks with $\delta = 1$, it is desired that the dimension of the output of the convolutional operator coincides with the input dimensions. However, for example in the frequent case that $\delta = 1$ and $p, q > 1$ we get from Remark 1.7 that $\hat{n}_1 = n_1 - p + 1 < n_1$ and $\hat{l}_1 = l_1 - q + 1 < l_1$. Often a *padding operator* is prefixed in order to maintain the input dimensions. It works in the way that the input $x \in \mathbb{R}^{n_1 \times l_1} \otimes \mathbb{R}^{m_1}$ is expanded to an element $x' \in \mathbb{R}^{n_1' \times l_1'} \otimes \mathbb{R}^{m_1}$ by adding extra rows and columns, of for example zeros, to x. There are also other padding methods such as reflecting or repeating the outer rows and columns of x but we restrict to zero padding in this thesis. Regarding the more general case $\delta \geq 1$ it can be practical if the output size is $\left(\lfloor \frac{n_1}{\delta} \rfloor, \lfloor \frac{l_1}{\delta} \rfloor\right)$. Let the amount of rows and columns be denoted by $\alpha, \beta \in \mathbb{N}_0$. Then, $n_1' = n_1 + \alpha$, $l_1' = l_1 + \beta$ and if we choose $\alpha := p - \delta$ and $\beta := q - \delta$ then, with equation (1.6), we see that for the new size $(\hat{n}_1, \hat{l}_1)$ of $C(x')$ we indeed get

$$\hat{n}_1 = \left\lfloor \frac{n_1' - p}{\delta} \right\rfloor + 1 = \left\lfloor \frac{n_1 + \alpha - p}{\delta} \right\rfloor + 1 = \left\lfloor \frac{n_1 - \delta}{\delta} \right\rfloor + 1 = \left\lfloor \frac{n_1}{\delta} \right\rfloor.$$

Analogously we get $\hat{l}_1 = \left\lfloor \frac{l_1}{\delta} \right\rfloor$. In the following, let $\{E'_{j,k}\}_{j,k=1}^{n_1', l_1'}$ denote an orthonormal basis of $\mathbb{R}^{n_1' \times l_1'}$.

Definition 1.8 (Padding Operator) Let $\alpha, \beta \in \mathbb{N}_0$. The *padding operator*

$$\mathfrak{P}^{\alpha,\beta} : \mathbb{R}^{n_1 \times l_1} \otimes \mathbb{R}^{m_1} \longrightarrow \mathbb{R}^{n_1' \times l_1'} \otimes \mathbb{R}^{m_1},$$

where $n_1' = n_1 + \alpha$, $l_1' = l_1 + \beta$, is defined as

$$\mathfrak{P}^{\alpha,\beta}(x) := \sum_{j=1}^{m_1} \mathfrak{p}_j^{\alpha,\beta}(x) \otimes e_j,$$

where $\mathfrak{p}_j^{\alpha,\beta}(x) \in \mathbb{R}^{n_1' \times l_1'}$ is

$$\mathfrak{p}_j^{\alpha,\beta}(x) := \sum_{k=1}^{n_1} \sum_{l=1}^{l_1} \langle x_j, E_{k,l} \rangle E'_{k+\lfloor \frac{\alpha}{2} \rfloor, l+\lfloor \frac{\beta}{2} \rfloor}.$$

where $x = \sum_{j=1}^{m_1} x_j \otimes \bar{e}_2$ and $x_j \in \mathbb{R}^{n_1 \times l_1}$. If for a stride $\delta \in \mathbb{N}$ and filter size $(p, q) \in \mathbb{N}^2$ we have $(\alpha, \beta) = (p - \delta, q - \delta)$ we write $\mathfrak{P}^{\text{same}} := \mathfrak{P}_{\alpha,\beta}$.

Note that if we write the composition of the padding operator and the convolutional operator, then we assume the input size of the convolutional operator to be the output size of the padding operator, i.e.

$$\mathbb{R}^{n_1\times l_1}\otimes\mathbb{R}^{m_1}\xrightarrow{\mathfrak{P}}\mathbb{R}^{n_1'\times l_1'}\otimes\mathbb{R}^{m_1}\xrightarrow{C}\mathbb{R}^{\hat{n}_1\times\hat{l}_1}\otimes\mathbb{R}^{m_2}.$$

The representability of functions by a CNN is significantly increased if a channel-wise offset is allowed. This is achieved by the bias operator.

Definition 1.9 (Bias Operator) The *bias operator* $B : \mathbb{R}^{m_2}\times(\mathbb{R}^{\hat{n}_1\times\hat{l}_1}\otimes\mathbb{R}^{m_2})\longrightarrow\mathbb{R}^{\hat{n}_1\times\hat{l}_1}\otimes\mathbb{R}^{m_2}$ is defined as

$$B(V,x):=\sum_{j=1}^{m_2}b_j(V,x)\otimes\bar{e}_j,$$

where $b_j(V,x)\in\mathbb{R}^{\hat{n}_1\times\hat{l}_1}$ is

$$b_j(V,x):=\sum_{k=1}^{\hat{n}_1}\sum_{l=1}^{\hat{l}_1}\left(\langle x_j,\hat{E}_{kl}\rangle+V_j\right)\hat{E}_{kl},$$

where $x=\sum_{j=1}^{m_2}x_j\otimes\bar{e}_j$ and $x_j\in\mathbb{R}^{\hat{n}_1\times\hat{l}_1}$.

With the operators just mentioned, linear functions with offset can already be described. However, the full potential of neural networks is only revealed by the introduction of an activation function. This is non-linear and is applied to the input element by element. Especially for networks with several layers, very complex mappings can thus be represented. In this still somewhat general section, we introduce element-wise functions. Later, these appear as activation functions.

Definition 1.10 (Elementwise function) Let K be a field, E be a K-inner product space and $\{e_k\}_{k=1}^n$ be an orthogonal basis of E. We define an *elementwise function* as a map $\Psi : E\longrightarrow E$ of the form

$$\Psi(v)=\sum_{k=1}^{n}\psi(\langle v,e_k\rangle)e_k,$$

where $\psi : K\longrightarrow K$.

We are now able to explicitly define the composition of the operators just introduced and thus the function described by a convolutional layer.

Definition 1.11 (Convolutional layerwise function) For an input x, filters W, biases $V$and the operators defined as above we define the *layerwise function* f as

$$\begin{aligned}f:\ (\mathbb{R}^{n_1\times l_1}\otimes\mathbb{R}^{m_1})\times\mathbb{R}^{m_2}\times(\mathbb{R}^{p\times q\times m_1}\otimes\mathbb{R}^{m_2})&\longrightarrow\mathbb{R}^{\hat{n}_1\times\hat{l}_1}\otimes\mathbb{R}^{m_2}\\(x,V,W)&\longmapsto\Psi(B(V,C(W,\mathfrak{P}^{\mathsf{same}}(x)))),\end{aligned}$$

where $\Psi:\mathbb{R}^{\hat{n}_1\times\hat{l}_1}\otimes\mathbb{R}^{m_2}\longrightarrow\mathbb{R}^{\hat{n}_1\times\hat{l}_1}\otimes\mathbb{R}^{m_2}$ is an elementwise function with associated map $\psi:\mathbb{R}\longrightarrow\mathbb{R}$ and write $f\in\mathsf{Conv}(m_1,m_2,(p,q),\delta)$

Remark 1.12 (Differences to [8]) It should be mentioned that our definition of the cropping operator and the convolution operator differs from the definition in [8]. Our definition complies more with the actual implementation of machine learning frameworks like PyTorch [1] and TensorFlow [2], where in [8] the cropping operator reduces the dimension of $x \in \mathbb{R}^{n_1 \times l_1} \otimes \mathbb{R}^{m_1}$ by taking the sum over the channel components, which gives an element $\tilde{x} \in \mathbb{R}^{n_1 \times l_1}$. Then $\tilde{x}$ is convolved with filters of the shape $\tilde{W}_j \in \mathbb{R}^{p \times q}$. Also, the definition of the convolutional layer in [8] does not include a bias- or padding operator.

1.2.3 Multiple layers

We are going to convey the definitions of operators and the convolutional layerwise function of Section 1.2.2 to a setting with multiple consecutive convolutional layers. For that we have to specify the spaces of the corresponding inputs, outputs and parameters. Let $L \in \mathbb{N}$ be the amount of layers, $m \in \mathbb{R}^{L+1}$ the amount of channels per layer, $n, l \in \mathbb{N}^{L+1}$ the sizes of the inputs and outputs per layer, $p, q \in \mathbb{N}^L$ the filter sizes, $\psi \in \{\psi \mid \psi : \mathbb{R} \longrightarrow \mathbb{R}\}^L$ the activation functions and $\delta \in \mathbb{N}^L$ the strides of the layers. For each $i \in [L]$ we redefine the padding, cropping, convolution and bias operators and the elementwise function with new input and output shapes as

$$\begin{aligned}
\mathfrak{P}_i^{\alpha_i,\beta_i} :\ \mathbb{R}^{n_i \times l_i} \otimes \mathbb{R}^{m_i} &\longrightarrow \mathbb{R}^{n_i' \times l_i'} \otimes \mathbb{R}^{m_i} && \text{for } \alpha_i, \beta_i \in \mathbb{N}_0 \text{ and} \\
x &\longmapsto \mathfrak{P}_i^{\alpha,\beta}(x) && n_i' = n_i + \alpha, l_i' = l_i + \beta; \\
\mathrm{K}_i^{k,l} :\ \mathbb{R}^{n_i' \times l_i'} \otimes \mathbb{R}^{m_i} &\longrightarrow \mathbb{R}^{p_i \times q_i \times m_i} && \text{for } k \in [n_i' - p_i + 1], \\
x &\longmapsto \mathrm{K}_i^{k,l}(x) && l \in [l_i' - q_i + 1]; \\
C_i :\ (\mathbb{R}^{p_i \times q_i \times m_i} \otimes \mathbb{R}^{m_{i+1}}) \times (\mathbb{R}^{n_i' \times l_i'} \otimes \mathbb{R}^{m_i}) &\longrightarrow \mathbb{R}^{n_{i+1} \times l_{i+1}} \otimes \mathbb{R}^{m_{i+1}} \\
(W, x) &\longmapsto C_i(W, x); \\
B_i :\ \mathbb{R}^{m_{i+1}} \times (\mathbb{R}^{n_{i+1} \times l_{i+1}} \otimes \mathbb{R}^{m_{i+1}}) &\longrightarrow \mathbb{R}^{n_{i+1} \times l_{i+1}} \otimes \mathbb{R}^{m_{i+1}} \\
(V, x) &\longmapsto B_i(V, x); \\
\Psi_i :\ \mathbb{R}^{n_{i+1} \times l_{i+1}} \otimes \mathbb{R}^{m_{i+1}} &\longrightarrow \mathbb{R}^{n_{i+1} \times l_{i+1}} \otimes \mathbb{R}^{m_{i+1}};
\end{aligned}$$

where Ψ_i is the elementwise function for ψ_i on $\mathbb{R}^{n_{i+1} \times l_{i+1}} \otimes \mathbb{R}^{m_{i+1}}$ for $i \in [L]$. The exact mapping rules are as in the corresponding definitions. For $i \in [L]$ we write $\mathfrak{P}_i^{\text{same}} = \mathfrak{P}^{\alpha_i,\beta_i}$ if $(\alpha_i, \beta_i) = (p_i - \delta_i, q_i - \delta_i)$.

With the new definitions of the operators for each layer $i \in [L]$ we are able to formulate the function of the concatenation of multiple convolutional layers. We call such a concatenation *convolutional neural network (CNN)*.

Definition 1.13 (CNN-function) Let L, m, n, l, ψ, δ and $\mathfrak{P}_i^{\text{same}}, \mathrm{K}_i^{k,l}, C_i, B_i, \Psi_i$ for each $i \in [L]$ be defined as above. Then we define the *CNN-function* as

$$\begin{aligned}
f :\ (\mathbb{R}^{n_1 \times l_1} \otimes \mathbb{R}^{m_1}) \times \mathbb{R}^{m_2} \times (\mathbb{R}^{p_1 \times q_1 \times m_1} \otimes \mathbb{R}^{m_2}) &\longrightarrow \mathbb{R}^{n_{L+1} \times l_{L+1}} \otimes \mathbb{R}^{m_{L+1}} \\
(x, V, W) &\longmapsto (f_L(\bullet, V_L, W_L) \circ \ldots \circ f_1(\bullet, V_1, W_1))(x),
\end{aligned}$$

where for all $i \in [L]$

$$\begin{aligned}
f_i :\ (\mathbb{R}^{n_i \times l_i} \otimes \mathbb{R}^{m_i}) \times \mathbb{R}^{m_{i+1}} \times (\mathbb{R}^{p_i \times q_i \times m_i} \otimes \mathbb{R}^{m_{i+1}}) &\longrightarrow \mathbb{R}^{n_{i+1} \times l_{i+1}} \otimes \mathbb{R}^{m_{i+1}} \\
(x, V, W) &\longmapsto \Psi_i(B_i(V, C_i(W, \mathfrak{P}_i^{\text{same}}(x)))).
\end{aligned}$$

[1]See `https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html`.
[2]See `https://www.tensorflow.org/api_docs/python/tf/nn/conv2d`.

In the notation of f_i we suppress the dependence of the stride δ_i and write

$$f \in \mathsf{CNN}_L(m_1, \dots, m_L; p_1, \dots, p_L; q_1, \dots, q_L; \psi_1, \dots, \psi_L, \delta_1, \dots, \delta_L) = \mathsf{CNN}_L(m, p, q, \psi, \delta).$$

If $m_1 = \dots = m_L$ we can also write $\mathsf{CNN}_L(m_1; p, q, \psi)$ and likewise for p, q, ψ and δ.

Note that by Remark 1.7, n and l are completely determined by the input dimensions (n_1, l_1) and the stride δ. By the definition of $\mathfrak{P}_i^{\text{same}}$ we have that if $\delta = 1$ then $n_1 = \dots = n_{L+1}$ and $l_1 = \dots = l_{L+1}$.

Remark 1.14 (Depthwise 2D Convolutions) The convolutional layerwise function from Definition 1.11 applies a weight $W_j \in \mathbb{R}^{p \times q \times m_1}$, $j \in [m_2]$ by cross-correlation to an input element $x \in \mathbb{R}^{n_1 \times l_1} \otimes \mathbb{R}^{m_1}$ for each output dimension $1, \dots, m_2$. This means that although x actually spans two directions for each channel, a calculation is performed along three directions via the channel axis (if $m_1 > 1$). This makes it possible to recognise interrelationships across channels. In some cases, it is sufficient to assume the channels are independent. The input channels can then be processed separately via two-directional weights. Equivalently we can regard convolutional layerwise functions $f_j \in \mathsf{Conv}(1, 1, (p, q), \delta)$, $j \in [m_1]$ and define the *depthwise convolutional layerwise function* as

$$\begin{aligned} \overline{\overline{f}} \colon (\mathbb{R}^{n_1 \times l_1} \otimes \mathbb{R}^{m_1}) \times \mathbb{R}^{m_1} \times (\mathbb{R}^{p \times q} \otimes \mathbb{R}^{m_1}) &\longrightarrow \mathbb{R}^{\hat{n}_1 \times \hat{l}_1} \otimes \mathbb{R}^{m_1} \\ (x, V, W) &\longrightarrow \sum_{j=1}^{m_1} f_j(x_j \otimes e_j, V_j, W_j \otimes \bar{e}_j) \otimes \bar{e}_j \end{aligned}$$

We write $\overline{\overline{f}} \in \overline{\overline{\mathsf{Conv}}}(m_1, (p, q), \delta)$.

We are now able to describe CNNs precisely by a mathematical function. So far, we have assumed the parameters to be arbitrary elements. To actually train CNNs we still need to understand how the resulting function depends on the parameters and we need a method to optimise the parameters of the CNN. It has proven useful in the field of machine learning to use gradient-based optimisation methods, which is why we need to calculate the derivatives of the CNN operators. This will be done in the next section.

1.3 Optimisation of CNNs

We approach the optimisation of CNNs step by step. First, we consider the gradient decent algorithm for convex functions. Then we discuss how it can be applied in general to the structure of generic neural networks. In a final concretisation, we will derive the gradient decent algorithm explicitly for CNNs.

1.3.1 Gradient decent algorithm for convex functions

The optimisation of neural networks is mostly done by a gradient-based optimisation algorithm. In this section we consider the ideal case of convex functions with Lipschitz continuous derivatives. For that we write

$$\mathfrak{F}_L^{1,1}(\mathbb{R}^d) := \{f \in C^1(\mathbb{R}^d) \mid \nabla f \text{ is Lipschitz continuous with Lipschitz constant } L > 0\}.$$

Recall that a minimum of a convex function is a global minimum, which is not necessarily unique but becomes unique if the function is assumed to be strictly convex. The gradient decent algorithm will be presented in a similar way as in [22], Theorem 2.1.14.

Lemma 1.15 Let $f \in \mathcal{F}_L^{1,1}(\mathbb{R}^d)$ be convex. Then, for $x, y \in \mathbb{R}^d$, the following equations hold:

$$f(x) + \nabla f(x) \cdot (y - x) \leq f(y) \tag{1.7}$$

$$0 \leq f(y) - f(x) - \nabla f(x) \cdot (y - x) \leq \frac{L}{2}\|y - x\|^2 \tag{1.8}$$

$$\|\nabla f(x) - \nabla f(y)\|^2 \leq L(\nabla f(x) - \nabla f(y)) \cdot (x - y) \tag{1.9}$$

Proof. Let $t \in [0, 1]$, then by the definition of convexity

$$f(tx + (1-t)y) \leq tf(x) + (1-t)f(y)$$

Note that

$$-\frac{t}{1-t} = \frac{1-1}{1-t} - \frac{t}{1-t} = \frac{1-t}{1-t} - \frac{1}{1-t} = 1 - \frac{1}{1-t}. \tag{1.10}$$

Then we get

$$\begin{aligned} f(y) &\geq \frac{1}{1-t}(f(tx + (1-t)y) - tf(x)) \\ &= \frac{1}{1-t}f(tx + (1-t)y) - \frac{t}{1-t}f(x) \\ &\overset{(1.10)}{=} f(x) + \frac{1}{1-t}(f(tx + (1-t)y) - f(x)) \\ &= f(x) + \frac{1}{1-t}(f(x + (1-t)(y-x)) - f(x)) \xrightarrow{t \to 1} f(x) + \nabla f(x) \cdot (y - x), \end{aligned}$$

which implies equation (1.7). We get the first inequality of (1.8) from (1.7). For the second inequality regard

$$\begin{aligned} f(y) - f(x) - \nabla f(x)(y - x) &= \int_0^1 \nabla f(x + s(y-x)) \cdot (y - x)\, ds - \nabla f(x) \cdot (y - x) \\ &= \int_0^1 (\nabla f(x + s(y-x)) - \nabla f(x)) \cdot (y - x)\, ds \\ &\leq \int_0^1 L(x + s(y-x) - x) \cdot (y - x)\, ds \\ &= L\|y - x\|^2 \int_0^1 s\, ds = \frac{L}{2}\|y - x\|^2. \end{aligned}$$

Equation (1.9) follows directly from the Lipschitz continuity of ∇f, namely

$$\|\nabla f(y) - \nabla f(x)\|^2 = (\nabla f(y) - \nabla f(x)) \cdot (\nabla f(y) - \nabla f(x)) \leq (\nabla f(y) - \nabla f(x))L(y - x).$$ ■

Theorem 1.16 (Gradient decent) Let $f \in \mathcal{F}_L^{1,1}$ be convex and $0 < h < \frac{2}{L}$. Let $x^\star$ be such that $f(x^\star)$ is minimal. Let $x_0 \in \mathbb{R}^d$ and define recursively $x_{k+1} = x_k - h\nabla f(x_k)$ for $k \in \mathbb{N}$. Then

$$f(x_k) - f(x^\star) \leq \frac{2(f(x_0) - f(x^\star))\|x_0 - x^\star\|^2}{2\|x_0 - x^\star\|^2 + h(2 - Lh)(f(x_0) - f(x^\star))^2 k}.$$

In particular, $f(x_k) \xrightarrow{k \to \infty} f(x^\star)$.

Proof. For $k \geq 1$ define $r_k := \|x_k - x^\star\|$. Then

$$\begin{aligned} r_{k+1}^2 &= \|x_{k+1} - x^\star\|^2 = \|x_k - x^\star - h\nabla f(x_k)\|^2 \\ &= r_k^2 - 2h\nabla f(x_k)\cdot(x_k - x^\star) + h^2\|\nabla f(x_k)\|^2 \overset{(1.9)}{\leq} r_k^2 - h\left(\frac{2}{L} - h\right)\|\nabla f(x_k)\|^2, \end{aligned} \tag{1.11}$$

where in the last inequality we used the fact that $\nabla f(x^\star) = 0$. From equation (1.8) we get

$$\begin{aligned} f(x_{k+1}) &\leq f(x_k) + \nabla f(x_k)\cdot(x_{k+1} - x_k) + \frac{L}{2}\|x_{k+1} - x_k\|^2 \\ &= f(x_k) - h\left(1 - \frac{L}{2}h\right)\|\nabla f(x_k)\|^2 \\ &= f(x_k) - \omega\|\nabla f(x_k)\|^2, \end{aligned} \tag{1.12}$$

where $\omega = h(1 - \frac{L}{2}h)$. Define $\delta_k := f(x_k) - f(x^\star)$. Note that calculation 1.11 implies $r_k \leq r_0$. Then with the Cauchy-Schwarz inequality we get

$$\delta_k \overset{(1.7)}{\leq} \nabla f(x_k)\cdot(x_k - x^\star) \leq r_k\|\nabla f(x_k)\| \leq r_0\|\nabla f(x_k)\| \tag{1.13}$$

Thus,

$$\delta_{k+1} = f(x_{k+1}) - f(x^\star) \overset{(1.12)}{\leq} f(x_k) - f(x^\star) - \omega\|\nabla f(x_k)\|^2 = \delta_k - \omega\|\nabla f(x_k)\|^2 \overset{(1.13)}{\leq} \delta_k - \frac{\omega}{r_0^2}\delta_k^2. \tag{1.14}$$

This implies

$$\frac{1}{\delta_{k+1}} = \frac{1}{\delta_{k+1}}\frac{\delta_{k+1} + \delta_k - \delta_{k+1}}{\delta_k} \overset{(1.14)}{\geq} \frac{1}{\delta_{k+1}}\frac{\delta_{k+1} + \frac{\omega}{r_0^2}\delta_k^2}{\delta_k} = \frac{1}{\delta_k} + \frac{\omega}{r_0^2}\frac{\delta_k}{\delta_{k+1}} \geq \frac{1}{\delta_k} + \frac{\omega}{r_0^2}.$$

Summing this inequality for $k = 1, \ldots, k+1$ we get

$$\frac{1}{\delta_{k+1}} \geq \frac{1}{\delta_0} + \frac{\omega}{r_0^2}(k+1).$$

Finally,

$$f(x_k) - f(x^\star) \leq \frac{1}{\frac{1}{\delta_0} + \frac{\omega}{r_0^2}k} = \frac{\delta_0 r_0^2}{r_0^2 + \omega\delta_0^2 k} = \frac{2(f(x_0) - f(x^\star))\|x_0 - x^\star\|^2}{2\|x_0 - x^\star\|^2 + h(2 - Lh)(f(x_0) - f(x^\star))^2 k}. \qquad ■$$

1.3.2 Adjoint operators

In order to convey the gradient decent algorithm from Section 1.3.1 to neural networks we have to prepare the notion of adjoint (differential) operators.

Definition 1.17 (Adjoint operator) Let H_1, H_2 be inner product spaces. For a linear operator $T : H_1 \longrightarrow H_2$ we call the operator $T^\star : H_2 \longrightarrow H_1$ the *Hermitian adjoint* or *adjoint* of T if it fulfils the property

$$\langle Tx, y\rangle_{H_2} = \langle x, T^\star y\rangle_{H_1}$$

for all $x \in H_1, y \in H_2$.

Theorem 1.18 (Well-definedness and uniqueness of the adjoint operator) Let H_1, H_2 be inner product spaces. For every linear operator $T : H_1 \longrightarrow H_2$ the Hermitian adjoint exists and is unique.

Proof. For every $y \in H_2$ regard the continuous linear functional $x \longmapsto \langle Tx, y\rangle_{H_2}$. By the Riesz-Fréchet representation theorem (see [7], Theorem 5.5) there is a unique $z \in H_1$ such that $\langle Tx, y\rangle = \langle x, z\rangle$ for all $x \in H_1$. Now define the operator $T^\star : H_2 \longrightarrow H_1, y \longmapsto z$ that assigns y to z as just described. Then, $\langle Tx, y\rangle = \langle x, T^\star y\rangle$ for all $(x, y) \in H_1 \times H_2$ and $T^\star$ is unique since z is unique. ■

We will use the adjoint of the differential operator. Let $f : \mathbb{R}^s \longrightarrow \mathbb{R}^t$, $s, t \in \mathbb{N}$ be totally differentiable and $Df(x) \in \text{Hom}(\mathbb{R}^s, \mathbb{R}^t)$ be the total differential of f at $x \in \mathbb{R}^s$. Then we write $D^\star f(x)$ instead of $(Df(x))^\star$.

Proposition 1.19 (Reversing property) For inner product spaces H_1, H_2, H_3 and linear operators $T_1 : H_1 \longrightarrow H_2, T_2 : H_2 \longrightarrow H_3$ it holds that $(T_2 \circ T_1)^\star = T_1^\star \circ T_2^\star$.

Proof. For $x \in H_1, y \in H_3$ we compute

$$\langle (T_2 \circ T_1)x, y\rangle_{H_3} = \langle T_1 x, T_2^\star y\rangle_{H_2} = \langle x, (T_1^\star \circ T_2^\star)y\rangle_{H_1}.$$ ■

Proposition 1.20 (Adjoint real matrices) Let $T : \mathbb{C}^m \longrightarrow \mathbb{C}^n$, $n, m \in \mathbb{N}$ be a linear bounded operator with associated matrix $A = (\alpha_{ij})_{ij} \in \mathbb{C}^{n\times m}$. Then, the adjoint matrix $A^\star := (\overline{\alpha}_{ji})_{ji} \in \mathbb{C}^{m\times n}$ is the representing matrix of $T^\star$.

Proof. Let $\{e_i\}_{i=1}^m$ and $\{\bar{e}_i\}_{i=1}^n$ be the standard bases for $\mathbb{C}^m$ and $\mathbb{C}^n$ respectively. Then, for all $j \in [m]$, $i \in [n]$ we get

$$\alpha_{ij} = \langle Ae_j, \bar{e}_i\rangle_{\mathbb{C}^n} = \langle Te_j, \bar{e}_i\rangle_{\mathbb{C}^n} = \langle e_j, T^\star \bar{e}_i\rangle_{\mathbb{C}^m} = \langle e_j, B\bar{e}_i\rangle_{\mathbb{C}^m}.$$

for a matrix $B = (\beta_{ij})_{ij} \in \mathbb{C}^{m\times n}$. But

$$\langle e_j, B\bar{e}_i\rangle_{\mathbb{C}^m} = \overline{\langle B\bar{e}_i, e_j\rangle_{\mathbb{C}^m}} = \bar{\beta}_{ji},$$

which implies $A^\star = B$. ■

1.3.3 Backpropagation for generic neural networks

Now that we have studied the optimisation of convex functions using the gradient decent algorithm we can apply the procedure to neural networks. It is important to keep in mind that neural networks almost never reflect a convex function in terms of its parameters. On the contrary, the topology of minima can be very complicated and in practice one almost never finds the absolute minimum. Nevertheless, gradient-based methods of optimisation are used to find at least local minima. To increase the chance that the local minimum is also globally a low value, one often starts the training with a large learning rate in the hope to skip local minima and reduces the learning rate later in training.

In this section, we introduce a general description of generic neural networks for which we will carry out a formulation of the gradient decent algorithm. The notation and the structure of this section is based on [8] but was made a little more precise in some places.

For all $i \in [L]$, let $E_i = \mathbb{R}^{s_i}, H_i = \mathbb{R}^{t_i}$ and $E_{L+1} = \mathbb{R}^{s_{L+1}}$, $s_i, s_{L+1}, t_i \in \mathbb{N}$ be inner product spaces. A *deep neural network* with $L \in \mathbb{N}$ layers is represented as a composition of L functions

$$\begin{aligned} f : E_1 \times (H_1 \times \dots \times H_L) &\longrightarrow E_{L+1} \\ (x, \theta) &\longmapsto (f_L(\bullet, \theta_L) \circ \dots \circ f_1(\bullet, \theta_1))(x), \end{aligned}$$

where $f_i \in C^1(E_i \times H_i, E_{i+1})$. Additionally we define the *head map* at layer $i \in [L]$ as

$$\begin{aligned} \alpha_i : E_1 \times (H_1 \times \dots \times H_i) &\longrightarrow E_{i+1} \\ (x, \theta) &\longmapsto (f_i(\bullet, \theta_i) \circ \dots \circ f_1(\bullet, \theta_1))(x) \end{aligned}$$

and the *tail map* at layer $i \in [L]$ as

$$\begin{aligned} \omega_i : E_i \times (H_i \times \dots \times H_L) &\longrightarrow E_{L+1} \\ (x, \theta) &\longmapsto (f_L(\bullet, \theta_L) \circ \dots \circ f_i(\bullet, \theta_i))(x). \end{aligned}$$

We write $x_{i+1} := \alpha_i(x)$ and refer to the x_{i+1} as *state variables*, and the variables θ_i as *parameters* for $i \in [L]$. For convenience we set $\omega_{L+1} = \text{id}_{E_{L+1}}$.

For $f_i \in C^1(E_i \times H_i, E_{i+1})$ we write $D_{x_i} f_i(\tilde{x}_i, \tilde{\theta}_i) \in \text{Hom}(E_i, E_{i+1})$ and $D_{\theta_i} f_i(\tilde{x}_i, \tilde{\theta}_i) \in \text{Hom}(H_i, E_{i+1})$ for the total derivatives with respect to the state variable at $\tilde{x}_i \in E_i$ or the parameters at $\tilde{\theta}_i \in H_i$ respectively and we use the same notation for all differentiable functions that depend on state variables and parameters.

It follows directly from the definitions that for $i \in [L]$ it holds that

$$\begin{aligned} F(\bullet, \theta) &= \omega_{i+1}(\bullet, (\theta_{i+1}, \dots, \theta_L)) \circ \alpha_i(\bullet, (\theta_1, \dots, \theta_i)) && \text{for } \theta \in H_1 \times \dots \times H_L, \\ \omega_i(\bullet, (\theta_i, \dots, \theta_L)) &= \omega_{i+1}(\bullet, (\theta_{i+1}, \dots, \theta_L)) \circ f_i(\bullet, \theta_i), && \text{for } \theta \in H_i \times \dots \times H_L, \\ \text{and} \quad \alpha_i(\bullet, (\theta_1, \dots, \theta_i)) &= f_i(\bullet, \theta_i) \circ \alpha_{i-1}(\bullet, (\theta_1, \dots, \theta_{i-1})) && \text{for } \theta \in H_1 \times \dots \times H_L. \end{aligned}$$

The equations imply that for $i \in [L]$ the output f can be decomposed into

$$f(\bullet, \theta) = \omega_{i+1}(\bullet, (\theta_{i+1}, \dots, \theta_L)) \circ f_i(\bullet, \theta_i) \circ \alpha_{i-1}(\bullet, (\theta_1, \dots, \theta_{i-1})) \quad \text{for } \theta \in H_1 \times \dots \times H_L. \tag{1.15}$$

In order to formulate a gradient decent algorithm similar to Theorem 1.16 for generic neural networks we need the derivative of f with respect to θ.

Lemma 1.21 Let $x \in E_1, \theta \in H_1 \times \dots \times H_L$ and $i \in [L]$. Then,

$$D^\star_{\theta_i} f(x, \theta) = D^\star_\theta f_i(x_i, \theta_i) \cdot D^\star_x \omega_{i+1}(x_{i+1}, (\theta_{i+1}, \dots, \theta_L)), \tag{1.16}$$

where $x_{i+1} = \alpha_i(x, (\theta_1, \dots, \theta_i))$.

Proof. We compute the derivative of equation (1.15) with respect to θ_i:

$$\begin{aligned} D_{\theta_i} f(x, \theta) &= D_{\theta_i} \omega_{i+1}(f_i(\alpha_{i-1}(x, (\theta_1, \dots, \theta_{i-1})), \theta_i), (\theta_{i+1}, \dots, \theta_L)) \\ \text{(chain rule)} \quad &= D_x \omega_{i+1}(x_{i+1}, (\theta_{i+1}, \dots, \theta_L)) \cdot D_\theta f_i(x_i, \theta_i) \end{aligned}$$

Then, by taking the adjoint and applying Remark 1.19 we obtain equation (1.16). ■

As described in Section 1.1 we want to optimise a loss function with respect to the parameters θ over a set of n network input samples $S = \{(x_{(1)}, y_{(1)}), \dots, (x_{(n)}, y_{(n)})\}$, where $x_{(j)} \in E_1$ is the j-th input sample with associated response or target $y_{(j)} \in E_{L+1}$. The most common loss function is the squared loss, given by

$$\begin{aligned} J :\ E_1 \times E_{L+1} \times (H_1 \times \dots \times H_L) &\longrightarrow \mathbb{R}_+ \\ (x, y, \theta) &\longmapsto \frac{1}{2}\|y - f(x,\theta)\|^2. \end{aligned}$$

In the following proposition we examine how J depends on the parameters θ_i of the i-th layer, $i \in [L]$.

Proposition 1.22 Let $x \in E_1, y \in E_{L+1}, \hat{y} := f(x,\theta), \theta \in H_1 \times \dots \times H_L$ and $i \in [L]$. Then, for $\zeta \in H_i$ it holds that

$$D_{\theta_i} J(x, y; \theta) \cdot \zeta = \langle D^\star_{\theta_i} f_i(x_i, \theta_i) \cdot D^\star_x \omega_{i+1}(x_{i+1}, (\theta_{i+1}, \dots, \theta_L)) \cdot (\hat{y} - y), \zeta \rangle, \tag{1.17}$$

where $x_i = \alpha_{i-1}(x)$.

Proof. Let $\zeta \in H_i$ be arbitrary. Then,

$$\begin{aligned} D_{\theta_i} J(x, y, \theta) \cdot \zeta &= \frac{1}{2} D_{\theta_i} \langle f(x,\theta) - y, f(x,\theta) - y\rangle \cdot \zeta \\ \text{(product rule)} \quad &= \langle f(x,\theta) - y, \underbrace{D_{\theta_i} f(x,\theta)}_{\in \mathrm{Hom}(H_i, E_{L+1})} \cdot \zeta \rangle = \langle D^\star_{\theta_i} f(x,\theta) \cdot (f(x,\theta) - y), \zeta \rangle \end{aligned}$$

Since $f(x,\theta) = \hat{y}$ and $D^\star_{\theta_i} f(x,\theta) = D^\star_\theta f_i(x_i,\theta_i) \cdot D^\star_x \omega_{i+1}(x_{i+1}, (\theta_{i+1}, \dots, \theta_L))$ we get equation (1.17) by Lemma 1.21. ■

According to the Riesz representation theorem this representation of $D_{\theta_i} J(x, y, \theta)$ as inner product is unique. Therefore we allow ourselves the abuse of notation of writing

$$D_{\theta_i} J(x, y; \theta) = D^\star_\theta f_i(x_i, \theta_i) \cdot D^\star_x \omega_{i+1}(x_{i+1}, (\theta_{i+1}, \dots, \theta_L)) \cdot (\hat{y} - y). \tag{1.18}$$

Theorem 1.23 (Backpropagation) For all $x_i \in E_i, \theta \in H_i \times \dots \times H_L$ it holds that

$$D^\star_x \omega_i(x_i, (\theta_i, \dots, \theta_L)) = D^\star_x f_i(x_i, \theta_i) \cdot D^\star_x \omega_{i+1}(x_{i+1}, (\theta_{i+1}, \dots, \theta_L)), \tag{1.19}$$

where $x_{i+1} = f_i(x_i, \theta_i)$ for all $i \in [L]$.

Proof. By the definition of ω_i we have

$$\begin{aligned} D_x \omega_i(x_i, (\theta_i, \dots, \theta_L)) &= D_x \left(\omega_{i+1}(f_i(x_i, \theta_i), (\theta_{i+1}, \dots, \theta_L))\right) \\ \text{(chain rule)} \quad &= D_x \omega_{i+1}(x_{i+1}) \cdot D_x f_i(x_i, \theta_i) \end{aligned}$$

Now taking the adjoint and applying the reversing property (1.19) implies equation (1.19) for all $i \in [L]$ since $\omega_{L+1} = \mathrm{id}_{E_{L+1}}$. ■

Now we are ready to present a gradient decent algorithm for generic neural networks.

Algorithm 1.24 (Gradient decent for genereric neural networks) For an input $(x, y) \in E_1 \times E_{L+1}$, parameters $\theta \in H_1 \times \cdots \times H_L$ and a learning rate $\eta > 0$ we define the following update rule.

```
1:  $x_1 \leftarrow x$
2:  for $i = 1, \dots, L$ do
3:      $x_{i+1} \leftarrow f_i(x_i, \theta_i)$
4:  end for
5:  for $i = L, \dots, 1$ do
6:      $\tilde{\theta}_i \leftarrow \theta_i$
7:      if $i = L$ then
8:          $e_L \leftarrow x_{L+1} - y$
9:      else
10:         $e_i \leftarrow D_x^\star f_{i+1}(x_{i+1}, \tilde{\theta}_{i+1}) \cdot e_{i+1}$
11:     end if
12:     $D_{\theta_i} J(x, y; \theta) \leftarrow D_\theta^\star f_i(x_i, \tilde{\theta}_i) \cdot e_i$
13:     $\theta_i \leftarrow \theta_i - \eta D_{\theta_i} J(x, y; \theta)$
14: end for
15: return $\theta$
```

In the lines 2-4 a forward pass is executed in order to compute and store the state at each layer. This values are then used in the backpropagation step. In line 12 $D_{\theta_i} J(x, y; \theta)$ is computed according to proposition 1.22, where the error vector e_i is obtained in line 8 or line 10. If $i = L$ we set $e_L \leftarrow x_{L+1} - y$ since $\omega_{L+1} = \text{id}$ and from proposition 1.22, namely

$$D_{\theta_L} J(x, y; \theta) = D_{\theta_L}^\star f_L(x_L, \theta_L) \underbrace{D_x^\star \omega_{L+1}(x_{L+1})}_{=\text{id}} \cdot e_L = D_{\theta_L}^\star f_L(x_L, \theta_L) \cdot e_L. \tag{1.20}$$

Otherwise, if $i \neq L$, e_i is obtained by Theorem 1.23 (backpropagation). Note that Algorithm 1.24 can be easily extended to a batch of input points $((x_{(j)}, y_{(j)}))_{j \in A}$, where $A \subseteq [n]$, by averaging the contribution to the gradient from each point $(x_{(j)}, y_{(j)})$ over the batch.

1.3.4 Backpropagation for CNNs

With the knowledge we have just gained, we can now apply the gradient decent algorithm to CNNs. The principle is identical to the one in the previous section on generic neural networks. However, in order to apply backpropagation, we have to explicitly calculate the derivatives of the individual operators of the CNN layers. The derivative of the entire CNN-Layer can then be derived with the chain rule.

Lemma 1.25 (Derivative of the convolution operator) In the setting of Definition 1.6 for the convolution operator C the derivatives

$$D_x C(\tilde{W}, \tilde{x}) \in \text{Hom}(\mathbb{R}^{n_1 \times l_1} \otimes \mathbb{R}^{m_1}, \mathbb{R}^{\hat{n}_1 \times \hat{l}_1} \otimes \mathbb{R}^{m_2})$$
$$\text{and } D_W C(\tilde{W}, \tilde{x}) \in \text{Hom}(\mathbb{R}^{p \times q \times m_1} \otimes \mathbb{R}^{m_2}, \mathbb{R}^{\hat{n}_1 \times \hat{l}_1} \otimes \mathbb{R}^{m_2})$$

with respect to the input and the filters respectively at $(\tilde{W}, \tilde{x}) \in (\mathbb{R}^{p \times q \times m_1} \otimes \mathbb{R}^{m_2}) \times (\mathbb{R}^{n_1 \times l_1} \otimes \mathbb{R}^{m_1})$ exist and are given by

$$D_x C(\tilde{W}, \tilde{x}) = C(\tilde{W}, \bullet) \quad \text{and} \quad D_W C(\tilde{W}, \tilde{x}) = C(\bullet, \tilde{x}).$$

Proof. Let $\nu := \sum_{j=1}^{m_1} \nu_j \otimes e_j \in \mathbb{R}^{n_1 \times l_1} \otimes \mathbb{R}^{m_1}$. Then, for the cropping operator we have

$$\begin{aligned}
\partial_t \, \mathrm{K}_{k,l}\left(\sum_{i=1}^{m_1} \tilde{x}_j + t\nu_j \otimes e_j\right)\bigg|_{t=0} &= \partial_t \left(\sum_{s=1}^{p}\sum_{t=1}^{q} \langle \tilde{x}_j + t\nu_j, E_{k+s-1,l+t-1}\rangle \tilde{E}_{s,t}\right)_{j=1}^{m}\Bigg|_{t=0} \\
&= \left(\sum_{s=1}^{p}\sum_{t=1}^{q} \langle \partial_t \tilde{x}_j + t\nu_j\big|_{t=0}, E_{k+s-1,l+t-1}\rangle \tilde{E}_{s,t}\right)_{j=1}^{m} \\
&= \left(\sum_{s=1}^{p}\sum_{t=1}^{q} \langle \nu_j, E_{k+s-1,l+t-1}\rangle \tilde{E}_{s,t}\right)_{j=1}^{m} \\
&= \mathrm{K}_{k,l}\left(\sum_{i=1}^{m_1} \nu_j \otimes e_j\right).
\end{aligned} \tag{1.21}$$

Putting this into the convolution operator we get

$$\begin{aligned}
D_x C(\tilde{W}, \tilde{x}) \cdot \nu = \partial_t \sum_{j=1}^{m_2}\sum_{k=1}^{\hat{n}_1}\sum_{l=1}^{\hat{l}_1} \langle \tilde{W}_j, \mathrm{K}_{\gamma(k,l,\delta)}(\tilde{x}_j + t\nu_j)\rangle \hat{E}_{k,l} \otimes \bar{e}_j\Bigg|_{t=0} \\
= \sum_{j=1}^{m_2}\sum_{k=1}^{\hat{n}_1}\sum_{l=1}^{\hat{l}_1} \langle \tilde{W}_j, \partial_t \mathrm{K}_{\gamma(k,l,\delta)}(\tilde{x}_j + t\nu_j)\big|_{t=0}\rangle \hat{E}_{k,l} \otimes \bar{e}_j \\
\overset{(1.21)}{=} \sum_{j=1}^{m_2}\sum_{k=1}^{\hat{n}_1}\sum_{l=1}^{\hat{l}_1} \langle \tilde{W}_j, \mathrm{K}_{\gamma(k,l,\delta)}(\nu_j)\rangle \hat{E}_{k,l} \otimes \bar{e}_j = C(\tilde{W}, \nu).
\end{aligned}$$

Analogously we get for $\mu := \sum_{j=1}^{m_2} \mu_j \otimes \bar{e}_j \in \mathbb{R}^{p \times q \times m_1} \otimes \mathbb{R}^{m_2}$

$$\begin{aligned}
D_W C(\tilde{W}, \tilde{x}) \cdot \mu = \partial_t \sum_{j=1}^{m_2}\sum_{k=1}^{\hat{n}_1}\sum_{l=1}^{\hat{l}_1} \langle \tilde{W}_j + t\mu_j, \mathrm{K}_{\gamma(k,l,\delta)}(\tilde{x}_j)\rangle \hat{E}_{k,l} \otimes \bar{e}_j\Bigg|_{t=0} \\
= \sum_{j=1}^{m_2}\sum_{k=1}^{\hat{n}_1}\sum_{l=1}^{\hat{l}_1} \langle \partial_t \tilde{W}_j + t\mu_j\big|_{t=0}, \mathrm{K}_{\gamma(k,l,\delta)}(\tilde{x}_j)\rangle \hat{E}_{k,l} \otimes \bar{e}_j \\
= \sum_{j=1}^{m_2}\sum_{k=1}^{\hat{n}_1}\sum_{l=1}^{\hat{l}_1} \langle \mu_j, \mathrm{K}_{\gamma(k,l,\delta)}(\tilde{x}_j)\rangle \hat{E}_{k,l} \otimes \bar{e}_j = C(\mu, \tilde{x}).
\end{aligned}$$

■

Lemma 1.26 (Derivative of the padding operator) In the setting of Definition 1.8, the total derivative of the padding operator $\mathfrak{P}_{\alpha,\beta}$ exists and is given by $D_x \mathfrak{P}_{\alpha,\beta}(\tilde{x}) \cdot \nu = \mathfrak{P}_{\alpha,\beta}(\nu)$ for all inputs and test points $\tilde{x}, \nu \in \mathbb{R}^{n_1 \times l_1} \otimes \mathbb{R}^{m_1}$.

Proof. For $\nu = \sum_{j=1}^{m_1} \nu_j \otimes e_j \in \mathbb{R}^{n_1 \times l_1} \otimes \mathbb{R}^{m_1}$ we have

$$\begin{aligned}
\partial_t \mathfrak{P}_{\alpha,\beta}(\tilde{x} + t\nu)\big|_{t=0} &= \partial_t \sum_{j=1}^{m_1}\sum_{k=1}^{n_1}\sum_{l=1}^{l_1} \langle \tilde{x}_j + t\nu_j, E_{k,l}\rangle E'_{k+\lfloor\frac{\alpha}{2}\rfloor, l+\lfloor\frac{\beta}{2}\rfloor} \otimes e_j\Bigg|_{t=0} \\
&= \sum_{j=1}^{m_1}\sum_{k=1}^{n_1}\sum_{l=1}^{l_1} \langle \partial_t \tilde{x}_j + t\nu_j\big|_{t=0}, E_{k,l}\rangle E'_{k+\lfloor\frac{\alpha}{2}\rfloor, l+\lfloor\frac{\beta}{2}\rfloor} \otimes e_j
\end{aligned}$$

$$= \sum_{j=1}^{m_1} \sum_{k=1}^{n_1} \sum_{l=1}^{l_1} \langle \nu_j, E_{k,l} \rangle E'_{k+\lfloor \frac{\alpha}{2} \rfloor, l+\lfloor \frac{\beta}{2} \rfloor} \otimes e_j = \mathfrak{P}_{\alpha,\beta}(\nu) \qquad \blacksquare$$

Lemma 1.27 (Derivative of the bias operator) In the setting of Definition 1.9, the total derivatives of the bias operator B

$$D_x B(\tilde{V}, \tilde{x}) \in \text{Hom}(\mathbb{R}^{\hat{n}_1 \times \hat{l}_1} \otimes \mathbb{R}^{m_2}, \mathbb{R}^{\hat{n}_1 \times \hat{l}_1} \otimes \mathbb{R}^{m_2})$$
$$\text{and } D_V B(\tilde{V}, \tilde{x}) \in \text{Hom}(\mathbb{R}^{m_2}, \mathbb{R}^{\hat{n}_1 \times \hat{l}_1} \otimes \mathbb{R}^{m_2})$$

with respect to the input and the biases respectively at the point $(\tilde{V}, \tilde{x}) \in (\mathbb{R}^{m_2}) \times \mathbb{R}^{\hat{n}_1 \times \hat{l}_1} \otimes \mathbb{R}^{m_2}$ are given by

$$D_x B(\tilde{V}, \tilde{x}) = \text{id}_{\mathbb{R}^{\hat{n}_1 \times \hat{l}_1} \otimes \mathbb{R}^{m_2}} \quad \text{and} \quad D_V B(\tilde{V}, \tilde{x}) \cdot \mu = \sum_{j=1}^{m_2} \sum_{k=1}^{\hat{n}_1} \sum_{l=1}^{\hat{l}_1} \mu_j \hat{E}_{kl} \otimes \bar{e}_j$$

for all test points $\mu \in \mathbb{R}^{m_2}$.

Proof. Let $\nu = \sum_{j=1}^{m_2} \nu_j \otimes \bar{e}_j \in \mathbb{R}^{\hat{n}_1 \times \hat{l}_1} \otimes \mathbb{R}^{m_2}$. Then,

$$\begin{aligned}
\partial_t B(\tilde{V}, \tilde{x} + t\nu)\Big|_{t=0} &= \partial_t \sum_{j=1}^{m_2} \sum_{k=1}^{\hat{n}_1} \sum_{l=1}^{\hat{l}_1} (\langle \tilde{x}_j + t\nu_j, \hat{E}_{kl} \rangle + \tilde{V}_j) \hat{E}_{kl} \otimes e_j \Bigg|_{t=0} \\
&= \sum_{j=1}^{m_2} \sum_{k=1}^{\hat{n}_1} \sum_{l=1}^{\hat{l}_1} \partial_t (\langle \tilde{x}_j + t\nu_j, \hat{E}_{kl} \rangle + \tilde{V}_j)\Big|_{t=0} \hat{E}_{kl} \otimes e_j \\
&= \sum_{j=1}^{m_2} \sum_{k=1}^{\hat{n}_1} \sum_{l=1}^{\hat{l}_1} \langle \nu_j, \hat{E}_{kl} \rangle \hat{E}_{kl} \otimes e_j \\
&= \sum_{j=1}^{m_2} \nu_j \otimes e_j = \nu
\end{aligned}$$

which proofs $D_x B(\tilde{V}, \tilde{x}) = \text{id}_{\mathbb{R}^{\hat{n}_1 \times \hat{l}_1} \otimes \mathbb{R}^{m_2}}$. For $D_V B(\tilde{V}, \tilde{x})$ let $\mu \in \mathbb{R}^{m_2}$, then

$$\begin{aligned}
\partial_t B(\tilde{V} + t\mu, \tilde{x})\Big|_{t=0} &= \partial_t \sum_{j=1}^{m_2} \sum_{k=1}^{\hat{n}_1} \sum_{l=1}^{\hat{l}_1} (\langle \tilde{x}_j, \hat{E}_{kl} \rangle + \tilde{V}_j + t\mu_j) \hat{E}_{kl} \otimes e_j \Bigg|_{t=0} \\
&= \sum_{j=1}^{m_2} \sum_{k=1}^{\hat{n}_1} \sum_{l=1}^{\hat{l}_1} \partial_t (\langle \tilde{x}_j, \hat{E}_{kl} \rangle + \tilde{V}_j + t\mu_j) \hat{E}_{kl}\Big|_{t=0} \otimes e_j \\
&= \sum_{j=1}^{m_2} \sum_{k=1}^{\hat{n}_1} \sum_{l=1}^{\hat{l}_1} \mu_j \hat{E}_{kl} \otimes e_j \qquad \blacksquare
\end{aligned}$$

Definition 1.28 (Hadamard product) Let E be a vector space and $\{e_k\}_{k=1}^n$ an orthogonal basis of of E. Then we define the *Hadamard product* as a symmetric bilinear operator $\odot \in \text{Hom}(E, E; E)$ over $\{e_k\}_{k=1}^n$ as

$$e_k \odot e_{k'} := \Delta_{k,k'} e_k, \quad \text{for all } k, k' \in [n],$$

where $\Delta_{i,j} = \begin{cases} 1, & i = j \\ 0, & \text{else} \end{cases}$ is the Kronecker delta.

Example 1.29 (Hadamard product on $\mathbb{R}^n$) Let $E = \mathbb{R}^n$ and $\{e_k\}_{k=1}^n$ be the standard basis of E. Then, for $v, v' \in E$:

$$v \odot v' = \left(\sum_{k=1}^n v_k e_k\right) \odot \left(\sum_{k'=1}^n v'_{k'} e_{k'}\right) = \sum_{k,k'=1}^n v_k v'_{k'} e_k \odot e_{k'} = \sum_{k,k'=1}^n v_k v'_{k'} \Delta_{k,k'} e_k = \sum_{k=1}^n v_k v'_k e_k.$$

For this reason the Hadamard product is also referred to as elementwise multiplication.

Lemma 1.30 (Derivative of elementwise functions) Let K be a field, E be a finite-dimensional K-inner product space, $\{e_k\}_{k=1}^n$ be an orthogonal basis of E and $\Psi : E \longrightarrow E$ be an elementwise function for a function $\psi : K \longrightarrow K$. Then, for $v, z \in E$,

$$D\Psi(z) \cdot v = \Psi'(z) \odot v,$$

where Ψ' is the corresponding elementwise function for the derivative ψ' of ψ.

Proof. This can be compute directly.

$$\begin{aligned} D\Psi(z) \cdot v &= \partial_t \Psi(z + tv)\big|_{t=0} \\ &= \partial_t \sum_{k=1}^n \psi(\langle z + tv, e_k\rangle) e_k \Bigg|_{t=0} \\ \text{(chain rule)} \quad &= \sum_{k=1}^n \psi'(\langle z, e_k\rangle)\, \partial_t \langle z + tv, e_k\rangle\big|_{t=0}\, e_k \\ &= \sum_{k=1}^n \psi'(\langle z, e_k\rangle) \langle v, e_k\rangle e_k = \Psi'(z) \odot v \end{aligned}$$

■

Theorem 1.31 (Derivative of convolutional layerwise function) In the setting of Definition 1.11 for the convolutional layerwise function $f \in \mathrm{Conv}(m_1, m_2, (p, q), \delta)$ the total derivatives

$$\begin{aligned} D_x f(\tilde{x}, \tilde{V}, \tilde{W}) &\in \mathrm{Hom}(\mathbb{R}^{n_1 \times l_1} \otimes \mathbb{R}^{m_1}, \mathbb{R}^{\hat{n}_1 \times \hat{l}_1} \otimes \mathbb{R}^{m_2}), \\ D_V f(\tilde{x}, \tilde{V}, \tilde{W}) &\in \mathrm{Hom}(\mathbb{R}^{m_2}, \mathbb{R}^{\hat{n}_1 \times \hat{l}_1} \otimes \mathbb{R}^{m_2}), \\ \text{and } D_W f(\tilde{x}, \tilde{V}, \tilde{W}) &\in \mathrm{Hom}(\mathbb{R}^{p \times q \times m_1} \otimes \mathbb{R}^{m_2}, \mathbb{R}^{\hat{n}_1 \times \hat{l}_1} \otimes \mathbb{R}^{m_2}) \end{aligned}$$

with respect to the input, biases and weights respectively at the point $(\tilde{x}, \tilde{V}, \tilde{W}) \in (\mathbb{R}^{n_1 \times l_1} \otimes \mathbb{R}^{m_1}) \times \mathbb{R}^{m_2} \times (\mathbb{R}^{p \times q \times m_1} \otimes \mathbb{R}^{m_2})$ are given by

$$\begin{aligned} D_x f(\tilde{x}, \tilde{V}, \tilde{W}) \cdot x &= \Psi'(B(\tilde{V}, C(\tilde{W}, \mathfrak{P}^{\mathsf{same}}(\tilde{x})))) \odot C(\tilde{W}, \mathfrak{P}^{\mathsf{same}}(x))), \\ D_V f(\tilde{x}, \tilde{V}, \tilde{W}) \cdot V &= \Psi'(B(\tilde{V}, C(\tilde{W}, \mathfrak{P}^{\mathsf{same}}(\tilde{x})))) \odot \sum_{j=1}^{m_2} \sum_{k=1}^{\hat{n}_1} \sum_{l=1}^{\hat{l}_1} V_j \hat{E}_{kl} \otimes \bar{e}_j, \\ \text{and } D_W f(\tilde{x}, \tilde{V}, \tilde{W}) \cdot W &= \Psi'(B(\tilde{V}, C(\tilde{W}, \mathfrak{P}^{\mathsf{same}}(\tilde{x})))) \odot C(W, \mathfrak{P}^{\mathsf{same}}(\tilde{x})) \end{aligned}$$

for all test points $x \in \mathbb{R}^{n_1 \times l_1} \otimes \mathbb{R}^{m_1}$, $V \in \mathbb{R}^{m_2}$ and $W \in \mathbb{R}^{p \times q \times m_1} \otimes \mathbb{R}^{m_2}$.

Proof. With the chain rule we get

$$\begin{aligned} D_x f(\tilde{x}, \tilde{V}, \tilde{W}) = D_x \Psi(B(\tilde{V}, C(\tilde{W}, \mathfrak{P}^{\mathsf{same}}(\tilde{x})))) \circ D_x B(\tilde{V}, C(\tilde{W}, \mathfrak{P}^{\mathsf{same}}(\tilde{x}))) \\ \circ D_x C(\tilde{W}, \mathfrak{P}^{\mathsf{same}}(\tilde{x})) \circ D_x \mathfrak{P}^{\mathsf{same}}(\tilde{x}) \end{aligned}$$

$$\begin{aligned}
\text{(Lemma 1.30)} \quad &= (\Psi'(B(\tilde{V}, C(\tilde{W}, \mathfrak{P}^{\text{same}}(\tilde{x})))) \odot \bullet) \circ D_x B(\tilde{V}, C(\tilde{W}, \mathfrak{P}^{\text{same}}(\tilde{x}))) \\
&\quad \circ D_x C(\tilde{W}, \mathfrak{P}^{\text{same}}(\tilde{x})) \circ D_x \mathfrak{P}^{\text{same}}(\tilde{x}) \\
\text{(Lemma 1.27)} \quad &= (\Psi'(B(\tilde{V}, C(\tilde{W}, \mathfrak{P}^{\text{same}}(\tilde{x})))) \odot \bullet) \circ D_x C(\tilde{W}, \mathfrak{P}^{\text{same}}(\tilde{x})) \circ D_x \mathfrak{P}^{\text{same}}(\tilde{x}) \\
\text{(Lemma 1.25)} \quad &= (\Psi'(B(\tilde{V}, C(\tilde{W}, \mathfrak{P}^{\text{same}}(\tilde{x})))) \odot \bullet) \circ C(\tilde{W}, \bullet) \circ D_x \mathfrak{P}^{\text{same}}(\tilde{x}) \\
\text{(Lemma 1.26)} \quad &= (\Psi'(B(\tilde{V}, C(\tilde{W}, \mathfrak{P}^{\text{same}}(\tilde{x})))) \odot \bullet) \circ C(\tilde{W}, \bullet) \circ \mathfrak{P}^{\text{same}}(\bullet).
\end{aligned}$$

This implies that for $x \in \mathbb{R}^{n_1 \times l_1} \otimes \mathbb{R}^{m_1}$ we have

$$D_x f(\tilde{x}, \tilde{V}, \tilde{W}) \cdot x = \Psi'(B(\tilde{V}, C(\tilde{W}, \mathfrak{P}^{\text{same}}(\tilde{x})))) \odot C(\tilde{W}, \mathfrak{P}^{\text{same}}(x))).$$

Analogously,

$$\begin{aligned}
D_W f(\tilde{x}, \tilde{V}, \tilde{W}) &= D_x \Psi(B(\tilde{V}, C(\tilde{W}, \mathfrak{P}^{\text{same}}(\tilde{x})))) \circ D_x B(\tilde{V}, C(\tilde{W}, \mathfrak{P}^{\text{same}}(\tilde{x}))) \\
&\quad \circ D_W C(\tilde{W}, \mathfrak{P}^{\text{same}}(\tilde{x})) \\
\text{(Lemma 1.30)} \quad &= (\Psi'(B(\tilde{V}, C(\tilde{W}, \mathfrak{P}^{\text{same}}(\tilde{x})))) \odot \bullet) \circ D_x B(\tilde{V}, C(\tilde{W}, \mathfrak{P}^{\text{same}}(\tilde{x}))) \\
&\quad \circ D_W C(\tilde{W}, \mathfrak{P}^{\text{same}}(\tilde{x})) \\
\text{(Lemma 1.27)} \quad &= (\Psi'(B(\tilde{V}, C(\tilde{W}, \mathfrak{P}^{\text{same}}(\tilde{x})))) \odot \bullet) \circ D_W C(\tilde{W}, \mathfrak{P}^{\text{same}}(\tilde{x})) \\
\text{(Lemma 1.25)} \quad &= (\Psi'(B(\tilde{V}, C(\tilde{W}, \mathfrak{P}^{\text{same}}(\tilde{x})))) \odot \bullet) \circ C(\bullet, \mathfrak{P}^{\text{same}}(\tilde{x}))
\end{aligned}$$

Thus, for $W \in \mathbb{R}^{p \times q \times m_1} \otimes \mathbb{R}^{m_2}$ we have

$$D_W f(\tilde{x}, \tilde{V}, \tilde{W}) \cdot W = \Psi'(B(\tilde{V}, C(\tilde{W}, \mathfrak{P}^{\text{same}}(\tilde{x})))) \odot C(W, \mathfrak{P}^{\text{same}}(\tilde{x}))$$

Finally,

$$\begin{aligned}
D_V f(\tilde{x}, \tilde{V}, \tilde{W}) &= D_x \Psi(B(\tilde{V}, C(\tilde{W}, \mathfrak{P}^{\text{same}}(\tilde{x})))) \circ D_V B(\tilde{V}, C(\tilde{W}, \mathfrak{P}^{\text{same}}(\tilde{x}))) \\
\text{(Lemma 1.30)} \quad &= (\Psi'(B(\tilde{V}, C(\tilde{W}, \mathfrak{P}^{\text{same}}(\tilde{x})))) \odot \bullet) \circ D_V B(\tilde{V}, C(\tilde{W}, \mathfrak{P}^{\text{same}}(\tilde{x})))
\end{aligned}$$

Hence, with Lemma 1.27 we get for a bias $V \in \mathbb{R}^{m_2}$,

$$D_V f(\tilde{x}, \tilde{V}, \tilde{W}) \cdot V = \Psi'(B(\tilde{V}, C(\tilde{W}, \mathfrak{P}^{\text{same}}(\tilde{x})))) \odot \sum_{j=1}^{m_2} \sum_{k=1}^{\hat{n}_1} \sum_{l=1}^{\hat{l}_1} V_j \hat{E}_{kl} \otimes \bar{e}_j. \qquad \blacksquare$$

For $L \in \mathbb{N}$ and $n, l \in \mathbb{N}^{L+1}$ we denote by $\{\hat{E}^i_{j,k}\}_{j,k=1}^{n_i, l_i}$ a orthonormal basis of $\mathbb{R}^{n_i \times l_i}$ for all $i \in [L+1]$.

Corollary 1.32 (Derivative of the CNN-function) For $i \in [L]$ in the setting of Definition 1.13 for the CNN-function $f \in \text{CNN}_L(m, p, q, \psi, \delta)$ the total derivatives of the i-th convolutional layerwise function f_i at the point $(\tilde{x}, \tilde{V}, \tilde{W}) \in (\mathbb{R}^{n_i \times l_i} \otimes \mathbb{R}^{m_i}) \times \mathbb{R}^{m_{i+1}} \times (\mathbb{R}^{p_i \times q_i \times m_i} \otimes \mathbb{R}^{m_{i+1}})$ are given by

$$D_x f_i(\tilde{x}, \tilde{V}, \tilde{W}) \cdot x = \Psi_i'(B_i(\tilde{V}, C_i(\tilde{W}, \mathfrak{P}_i^{\text{same}}(\tilde{x})))) \odot C_i(\tilde{W}, \mathfrak{P}_i^{\text{same}}(x))), \tag{1.22}$$

$$D_V f_i(\tilde{x}, \tilde{V}, \tilde{W}) \cdot V = \Psi_i'(B_i(\tilde{V}, C_i(\tilde{W}, \mathfrak{P}_i^{\text{same}}(\tilde{x})))) \odot \sum_{j=1}^{m_{i+1}} \sum_{k=1}^{n_{i+1}} \sum_{l=1}^{l_{i+1}} V_j \hat{E}_{kl}^{i+1} \otimes \bar{e}_j, \tag{1.23}$$

$$\text{and } D_W f_i(\tilde{x}, \tilde{V}, \tilde{W}) \cdot W = \Psi_i'(B_i(\tilde{V}, C_i(\tilde{W}, \mathfrak{P}_i^{\text{same}}(\tilde{x})))) \odot C_i(W, \mathfrak{P}_i^{\text{same}}(\tilde{x})) \tag{1.24}$$

for all test points $x \in \mathbb{R}^{n_i \times l_i} \otimes \mathbb{R}^{m_i}$, $V \in \mathbb{R}^{m_{i+1}}$ and $W \in \mathbb{R}^{p_i \times q_i \times m_i} \otimes \mathbb{R}^{m_{i+1}}$.

Now, being able to explicitly compute the derivatives of layerwise functions with respect to the input and parameters, we are almost ready to formulate the gradient decent algorithm for CNNs. Let $\tilde{x} \in \mathbb{R}^{n_i \times l_i} \otimes \mathbb{R}^{m_i}$, $\tilde{V} \in \mathbb{R}^{m_{i+1}}$ and $\tilde{W} \in \mathbb{R}^{p_i \times q_i \times m_i} \otimes \mathbb{R}^{m_{i+1}}$. We know from proposition 1.22 that in order to compute the derivatives of the squared loss for the i-th layer we need the adjoints $D_x^\star f_i(\tilde{x}, \tilde{V}, \tilde{W})$, $D_V^\star f_i(\tilde{x}, \tilde{V}, \tilde{W})$ and $D_W^\star f_i(\tilde{x}, \tilde{V}, \tilde{W})$. Let's consider $D_x f_i(\tilde{x}, \tilde{V}, \tilde{W}) \in \text{Hom}(\mathbb{R}^{n_i \times l_i} \otimes \mathbb{R}^{m_i}, \mathbb{R}^{n_{i+1} \times l_{i+1}} \otimes \mathbb{R}^{m_{i+1}})$ as an example. Since $\mathbb{R}^{n_i \times l_i} \otimes \mathbb{R}^{m_i}$ can be identified with $\mathbb{R}^{n_i \cdot l_i \cdot m_i}$, it is an linear bounded operator between finite dimensional vector spaces and can as such be represented by a matrix in the canonical basis which we denote by $[D_x f_i(\tilde{x}, \tilde{V}, \tilde{W})] \in \mathbb{R}^{n_{i+1} \cdot l_{i+1} \cdot m_{i+1} \times n_i \cdot l_i \cdot m_i}$. From proposition 1.20 it follows that the adjoint matrix $[D_x f_i(\tilde{x}, \tilde{V}, \tilde{W})]^\star$ (which equals the transpose in this real case) can be identified with the representing matrix for $D_x^\star f_i(\tilde{x}, \tilde{V}, \tilde{W})$. Note that for the sake of simple notation we allow ourselves the abuse of notation $[D_x f_i(\tilde{x}, \tilde{V}, \tilde{W})] \cdot x$ for an element $x \in \mathbb{R}^{n_i \times l_i} \otimes \mathbb{R}^{m_i}$ instead of writing $\phi_{i+1}^{-1}([D_x f_i(\tilde{x}, \tilde{V}, \tilde{W})] \cdot \phi_i(x))$, where $\phi_i : \mathbb{R}^{n_i \times l_i} \otimes \mathbb{R}^{m_i} \longrightarrow \mathbb{R}^{n_i \cdot l_i \cdot m_i}$ is the canonical isomorphism.

Algorithm 1.33 (Gradient decent for CNNs) In the setting of corollary 1.32 the gradient decent algorithm for CNNs for a data point $(x, y) \in (\mathbb{R}^{n_1 \times l_1} \otimes \mathbb{R}^{m_1}) \times (\mathbb{R}^{n_{L+1} \times l_{L+1}} \otimes \mathbb{R}^{m_{L+1}})$ is defined as follows:

```
1:  $x_1 \leftarrow x$
2:  for $i = 1, \dots, L$ do
3:      $x_{i+1} \leftarrow \Psi_i(B(V_i, C(W_i, \mathfrak{P}^{\text{same}}(x_i))))$
4:  end for
5:  for $i = L, \dots, 1$ do
6:      $\tilde{V}_i \leftarrow V_i$
7:      $\tilde{W}_i \leftarrow W_i$
8:      if $i = L$ then
9:          $e_L \leftarrow x_{L+1} - y$
10:     else
11:         $e_i \leftarrow [D_x f_{i+1}(x_{i+1}, \tilde{V}_{i+1}, \tilde{W}_{i+1})]^\star \cdot e_{i+1}$
12:     end if
13:     $[D_{V_i} J(x, y, V, W)] \leftarrow [D_V f_i(x_i, V_i, W_i)]^\star \cdot e_i$
14:     $[D_{W_i} J(x, y, V, W)] \leftarrow [D_W f_i(x_i, V_i, W_i)]^\star \cdot e_i$
15:     $V_i \leftarrow V_i - \eta[D_{V_i} J(x, y, V, W)]$
16:     $W_i \leftarrow W_i - \eta[D_{W_i} J(x, y, V, W)]$
17: end for
18: return $V_i, W_i$
```

In the lines 2-4 a forward pass is executed in which the state variables are stored for each layer. In lines 6 and 7 the parameters of the current layer are saved for later use. They would be overwritten before being needed in the next iteration. For the last layer the error vector is initialised as shown in equation (1.20). In the lines 13 and 14 the representing matrices for the derivatives of the squared loss with respect to the weights and biases of the i-th layer are computed according to Theorem 1.23 (backpropagation). Note that in order to compute the matrices $[D_V f_i(x_i, V_i, W_i)]^\star$ and $[D_W f_i(x_i, V_i, W_i)]^\star$ explicitly the equations (1.23) and (1.24) can be evaluated on the corresponding standard bases before taking adjoints. In the lines 15 and 16 the biases and weights are updated according to the definition of Algorithm 1.16 (gradient decent).

Remark 1.34 (Residual connections) A problem that can arise when optimising the parameters using backpropagation is that of the *vanishing gradient*. The activation function (here, the layerwise function Ψ_i) is often chosen sigmoidally (i.e. as a bounded, differentiable, real function), so its gradient may converge to zero. As can be seen in the backpropagation equation (1.19), the gradients of the back layers are multiplied to calculate the gradients in the front layers. For small gradients in each layer, this may lead to the dependence of the loss function on the parameters of the front layers decreasing exponentially from layer to layer. This can result in a standstill in the gradient decent updates. One solution to circumvent this problem is to add *residual* connections. For example, for $f \in \text{CNN}_L(m, p, q, \psi, \delta)$ as in Definition 1.13 adding a residual connection in the i-th layer means that the CNN-function becomes

$$f_L(\bullet, V_L, W_L) \circ \ldots \circ f_{i+1}(\bullet, V_L, W_L) \circ (f_i(\bullet, V_L, W_L) + \text{id}) \circ f_{i-1}(\bullet, V_L, W_L) \circ \ldots \circ f_1(\bullet, V_1, W_1).$$

This allows the network to pass information past the layers and prevent exponential dependency.

2 Demosaicing with CNNs

We will now explicitly apply the theoretical knowledge of statistical learning using CNNs gained in Chapter 1 to the task of demosaicing camera RAW images. For this purpose, we first formulate the demosaicing problem mathematically. To gain a better understanding of the requirements for demosaicing, we will then consider the problem against the theoretical background of sampling theory and find that the Nyquist-Shannon sampling theorem gives a notion of which frequencies can be reconstructed by demosaicing. The actual implementation and empirical investigations will then be carried out for the case of a Bayer pattern, as these are also used in ARRI cameras. We consider classical debayer algorithms and will find that we can choose CNN architectures that reflect the algorithmic structure of these classical algorithms. In the last section of this chapter we will implement a CNN and train it with the methods described in Chapter 1. We will see that the 3-stage architecture, which is inspired by gradient-based debayering, is superior to classical debayering in our tests.

2.1 Demosaicing

As already described in Section 1.1, it is essential in statistical learning to incorporate a priori knowledge into the actual learning process. For this purpose, we will now examine the actual task of demosaicing camera RAW images. We will use the knowledge gained from this for the choice of the actual CNN architecture and other hyperparameters.

2.1.1 General formulation

The sensors of ARRI's digital cinema cameras consist of an orthogonal array of pixels, each of them emitting electrons when electromagnetic radiation, such as light, hits them. Within the dynamic range of the pixels the amount of photocurrent that is returned from the sensor is proportional to the amount of photons hitting the sensor. It also depends of the wavelength of the photons and the spectral sensitivity of the pixels. In this way, a black and white image can be constructed from the sensor data by mapping pixels with low photocurrent to dark, and pixels with high photocurrent to bright grey tones.

In order to obtain colour images, a colour filter array (CFA) is placed in front of the pixel array. The CFA splits the pixels in subgroups with different spectral sensitivities. CFAs consist of a repetitive mosaic pattern of different primary colours.

Definition 2.1 (Mosaic pattern) Let $\alpha, \beta \in \mathbb{N}$. A *mosaic pattern* of size $\alpha \cdot \beta$ with $\mathbb{N} \ni \mu \leq \alpha \cdot \beta$ colour primaries is a matrix $M \in \{1, 2, \dots, \mu\}^{\alpha \times \beta}$.

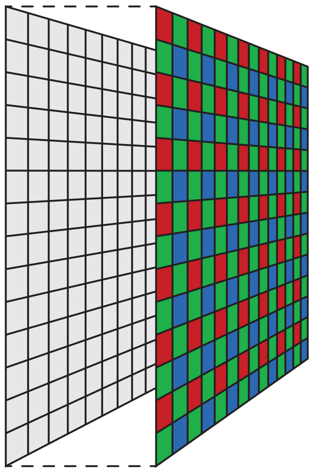


Figure 2.1: A Bayer pattern that is placed on an image sensor.

Example 2.2 (ALEV 4 Bayer pattern) The ARRI ALEV 4 Sensor has the mosaic pattern

$$M = \begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix},$$

where 1 corresponds to red, 2 to green and 3 to blue.

For a pixel with a specific primary colour the information of the corresponding other colours of the CFA is missing. This information can be obtained by interpolating the corresponding other channels. This interpolation map is not unique. For this reason we regard it as an inverse of the unique projection map that maps a coloured image into the CFA-domain.

Definition 2.3 (Mosaicing map) Let $M \in \{1, 2, \dots, \mu\}^{\alpha\times\beta}$ be a mosaic pattern. Let $m, n \in \mathbb{N}$, $m \geq \alpha, n \geq \beta$. Regard the map

$$m_M : \mathbb{R}^{m\times n\times\mu} \longrightarrow \mathbb{R}^{m\times n}, \quad I \longmapsto J$$

that projects an image of size (m, n) with μ colour primaries onto the mosaic-sensor domain, where $J_{i,j} = I_{i,j,M_{s,t}}$ for all $i \in \{1, \dots, m\}$, $j \in \{1, \dots, n\}$ and $s \in \{1, 2, \dots, \alpha\}$, $t \in \{1, 2, \dots, \beta\}$ are such that

$$s \equiv i \mod \alpha, \quad t \equiv j \mod \beta.$$

We call m_M the *mosaicing map* for M.

Definition 2.4 (Demosaicing map) Let $m_M : \mathbb{R}^{m\times n\times\mu} \longrightarrow \mathbb{R}^{m\times n}$ be a mosaicing map. We call $d_M : \mathbb{R}^{m\times n} \longrightarrow \mathbb{R}^{m\times n\times\mu}$ a *demosaicing map* if $m_M \circ d_M = \mathrm{id}_{\mathbb{R}^{m\times n}}$.

2.1.2 The Bayer pattern

Different mosaic patterns have been tried in the past. However, since it was patented in 1975 ([5]), the most established one is the so-called *Bayer pattern*.

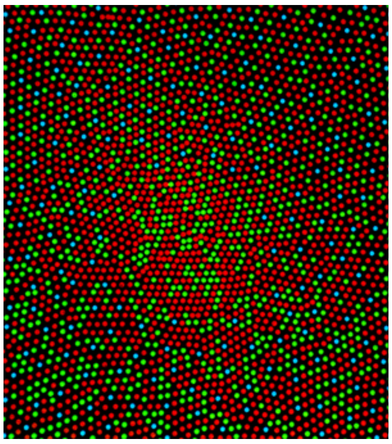


Figure 2.2: Representation of the retinal photoreceptor mosaic to illustrate the relative proportions of L, M, and S cones. The graphic is taken from [12].

Definition 2.5 (Bayer pattern, debayering) A mosaic pattern of the form

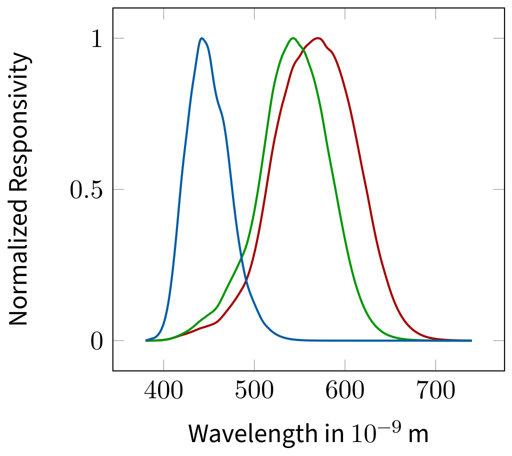
$$B \in \left\{ \begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix}, \begin{pmatrix} 3 & 2 \\ 2 & 1 \end{pmatrix}, \begin{pmatrix} 2 & 1 \\ 3 & 2 \end{pmatrix}, \begin{pmatrix} 2 & 3 \\ 1 & 2 \end{pmatrix} \right\},$$

is called *Bayer pattern* and the corresponding demosaicing map d_B is called *debayering map*.

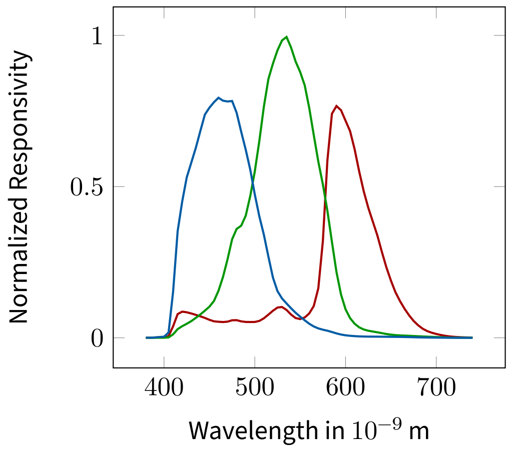
Since the goal of this work is to find a *good* demosaicing map for the ALEV 4 Bayer Pattern we will refer only to debayering maps from now. Nevertheless, the techniques and statements in this thesis can also be applied to other mosaic patterns.

One of the reasons for the success of the Bayer pattern is that it is based on the colour vision of the human visual system. Young (1802 in [28]) and Helmholtz (1852 in [14]) proposed the theory of trichromacy, stating that the human colour perception can be described by three colour stimuli. Empirically, it could be shown that most colours can be represented by a combination of short, medium, and long-wave stimuli. Although many effects (such as after-images or spatial effects in colour perception) could not be explained, the theory is still valid as a good approximation to reality and colour spaces such as CIE1931, which are based on this approach, are still used today.

Mullen (1985 in [21]) could show that the human visual system is more sensitive to differences in luminance than to differences in chrominance. The reason for this lies in the retinal distribution of cone types (L, M and S cones for long-wavelength, middle-wavelength and short-wavelength) and the spectral sensitivities of the human visual system. There are approximately twice as many L cones as M cones and approximately six times as many M cones as S cones (see [12]). Figure 2.2 illustrates this relationship of the density of cones. To understand why the Bayer pattern consists of twice as many green dyes as green or red ones we also need to look at the spectral sensitivities of the cones. Figure 2.3 shows a comparison of the spectral sensitivities of humans and the ALEV 3 sensor built into the ARRI ALEXA camera. The curves of most other digital cameras based on a Bayer pattern are similar. As we can see in Figure 2.3a there is a large overlap between the sensitivities of the L, and M cones and their sensitivities are pretty much in the middle of the wavelengths visible to humans. For this reason luminance perception of the human retina involves primarily the M and L cones, which are particularly sensitive to green light, when seeing in daylight. In comparison, when looking at Figure 2.3b, it is noticeable that the channels are more separated. Since the red channel has its peak in the higher wavelengths, the luminance



(a) Human spectral responsivities of the L, M, and S cones. It corresponds to the excitation of photopigment. The data was obtained from [26].



(b) Spectral responses for the red, green, and blue pixels of the ARRI ALEV 3 sensor. It corresponds to the amount of photocurrent as the result of exposure to light power (amperes per watt).

Figure 2.3: Exemplary comparison of the spectral responsivities of humans and cameras. The curves are normalised such that the largest value of all three curves is 1.

information is usually attributed to the green channel in digital cameras.

2.2 Challenges

For a given mosaic pattern it is mathematically trivial to find a demosaicing map in the sense of Definition 2.4. However, to be used in digital cameras, various requirements have to be fulfilled. For a RGB-image $I \in \mathbb{R}^{m\times n\times 3}$ and the corresponding Bayer image $J := m_B(I)$ the debayering problem can be seen as finding a debayering map d_B such that the perceptive difference between I and $d_B(J)$ is minimised. For this purpose, as many spatial frequencies as possible that are contained in I must be reconstructed in J. On the other hand, the creation of artefacts (unwanted image details that are not contained in I) must be minimised.

2.2.1 Nyquist–Shannon sampling theorem

Images coming from nature are generally not band-limited. This means that they can contain infinite high spatial frequencies. However, a digital image sensor only consists of a finite number of pixels and can therefore only provide a limited spatial resolution. A notion for this correlation between pixel pitch and the ability to capture spatial frequencies is given by the Nyquist–Shannon sampling theorem. We will prove the Nyquist-Shannon sampling theorem for the one-dimensional case, although we will use it later in the discussion of sampling two-dimensional images. It can then be considered dimension-wise.

We fix the notation, and recall some basic properties of the Fourier transformation and the Fourier series.

- For an integrable function $f \in L^1(\mathbb{R})$ the *fourier transform* $\hat{f}$ is defined as

$$\hat{f}(k) = \int_{\mathbb{R}} f(x)e^{-2\pi ikx}\, d\lambda(x),$$

and the *inverse fourier transform* is defined as

$$\check{f}(k) = \int_{\mathbb{R}} f(x) e^{2\pi i k x}\, d\lambda(x).$$

for all $k \in \mathbb{R}$.

- For $f \in L^1(\mathbb{R})$ it holds that $\hat{f} \in L^\infty(\mathbb{R})$ (see [18], Chapter 5).
- Regarded as a linear operator $\mathcal{F} : f \longmapsto \hat{f}$, *Plancherel's theorem* states that there is an unique extension of $\mathcal{F} : L^1(\mathbb{R}) \cap L^2(\mathbb{R}) \longrightarrow L^\infty(\mathbb{R})$ to an unitary operator on $L^2(\mathbb{R})$. In particular, $\|f\|_{L^2} = \|\mathcal{F} f\|_{L^2}$ (see [2], Chapter X, Theorem 3.23).
- If $f \in L^2(\mathbb{R})$ is periodic with period $T > 0$ (i.e. $f(x+T) = f(x)$ for almost every $x \in \mathbb{R}$), then the *Riesz–Fischer theorem* states that the *Fourier series*

$$f_N := \sum_{n=-N}^{N} c_n e^{-\frac{\pi i n \bullet}{T}}, \quad \text{where } c_n := \frac{1}{2T}\int_{-T}^{T} f(x) e^{\frac{\pi i n x}{T}}\, d\lambda(x), \tag{2.1}$$

converges to f in $L^2(\mathbb{R})$, i.e. $\|f_N - f\|_{L^2(\mathbb{R})} \xrightarrow{N\to\infty} 0$ (see [6], Theorem 13.12).

- If $f \in C^1([-T,T])$ is integrable, then the Fourier series f_N converges uniformly to f on $[-T,T]$. This is a weaker version of what is known as the *Dirichlet-Jordan test* (see [29], Chapter II, Theorem 8.1).

Lemma 2.6 (Sampling formula) Let $f \in L^1(\mathbb{R})$ be such that $\hat{f} \in C_c^1(\mathbb{R})$ is compactly supported and $\operatorname{supp}(\hat{f}) \subseteq [-b,b]$ for a $b > 0$. Then for all $x \in \mathbb{R}$ it holds that

$$f(x) = \sum_{n\in\mathbb{Z}} f\left(\frac{n}{2b}\right) \operatorname{sinc}(n - 2bx). \tag{2.2}$$

Proof. Since in equation (2.2) we want to evaluate f at specific points we start by ensuring that f actually is evaluable. We do this by showing that f is continuous:

$$\begin{aligned}\lim_{\epsilon\to 0} |f(x) - f(x+\epsilon)| &= \lim_{\epsilon\to 0} \left| \int_{\mathbb{R}} \hat{f}(k) e^{2\pi i k x}\, d\lambda(k) - \int_{\mathbb{R}} \hat{f}(k) e^{2\pi i k (x+\epsilon)}\, d\lambda(k) \right| \\ &\leq \lim_{\epsilon\to 0} \int_{\mathbb{R}} \left| \hat{f}(k) e^{2\pi i k x} \left(1 - e^{2\pi i k \epsilon}\right) \right| \, d\lambda(k).\end{aligned}$$

We define $g_\epsilon : k \longmapsto \hat{f}(k) e^{2\pi i k x}\left(1 - e^{2\pi i k \epsilon}\right)$ to check the conditions for the dominated convergence theorem:

1. It holds for all $k \in \mathbb{R}$ that $g_\epsilon(k) \xrightarrow{\epsilon\to 0} 0$.
2. Let $\bar{g} := |\hat{f}|$, then $|g_\epsilon(k)| \leq \bar{g}(k)$ and $\bar{g} \in L^1(\mathbb{R})$ because $\hat{f}$ is compactly supported and continuous.

With the dominated convergence theorem we get

$$\lim_{\epsilon\to 0} |f(x) - f(x+\epsilon)| \leq \int_{\mathbb{R}} \lim_{\epsilon\to 0} |g_\epsilon(k)|\, d\lambda(k) = 0.$$

Hence, $f \in C(\mathbb{R})$. We now turn to the proof of the main statement. For $k \in \mathbb{R}$, $N \in \mathbb{N}$, define

$$\hat{f}_N(k) := \sum_{n=-N}^{N} c_n e^{-\frac{2\pi i k n}{2b}}, \tag{2.3}$$

where

$$c_n := \frac{1}{2b} \int_{-b}^{b} \hat{f}(k) e^{\frac{2\pi i k n}{2b}} \, d\lambda(k) = \frac{1}{2b} f\left(\frac{n}{2b}\right). \tag{2.4}$$

Then, as $\hat{f} \in C^1([-b,b])$, we have that $\hat{f}_N \xrightarrow{N\to\infty} \hat{f}$ pointwise on $[-b,b]$ by the Dirichlet-Jordan test. Since supp$(\hat{f}) \subseteq [-b,b]$, this implies

$$\hat{f} \xleftarrow{N\to\infty} \hat{f}_N \cdot \mathbb{1}_{[-b,b]} \overset{(2.3)}{=} \sum_{n=-N}^{N} c_n e^{-\frac{2\pi i n \bullet}{2b}} \cdot \mathbb{1}_{[-b,b]} \overset{(2.4)}{=} \sum_{n=-N}^{N} \frac{1}{2b} f\left(\frac{n}{2b}\right) e^{-\frac{2\pi i n \bullet}{2b}} \cdot \mathbb{1}_{[-b,b]}. \tag{2.5}$$

Taking the inverse Fourier transform of $\hat{f}$ we get for all $x \in \mathbb{R}$,

$$f(x) = \int_{\mathbb{R}} \hat{f}(k) e^{2\pi i k x} \, d\lambda(k) \overset{(2.5)}{=} \int_{-b}^{b} \lim_{N\to\infty} \hat{f}_N(k) e^{2\pi i k x} \, d\lambda(k).$$

As $\hat{f}_N$ and $\hat{f}$ are continuous on $[-b,b]$, the limit and the integral can be interchanged since

$$\begin{aligned} \left| \int_{-b}^{b} \hat{f}_N(k) e^{2\pi i k x} \, d\lambda(k) - \int_{-b}^{b} \hat{f}(k) e^{2\pi i k x} \, d\lambda(k) \right| &= \left| \int_{-b}^{b} (\hat{f}_N(k) - \hat{f}(k)) e^{2\pi i k x} \, d\lambda(k) \right| \\ (\left|e^{2\pi i k x}\right| = 1) \quad &\leq \int_{-b}^{b} \left|\hat{f}_N(k) - \hat{f}(k)\right| \, d\lambda(k) \\ &\leq 2b \cdot \|\hat{f}_N - \hat{f}\|_{L^\infty([-b,b])} \xrightarrow{N\to\infty} 0. \end{aligned}$$

Finally, this implies

$$\begin{aligned} f(x) &= \lim_{N\to\infty} \int_{-b}^{b} \hat{f}_N(k) e^{2\pi i k x} \, d\lambda(k) \\ &= \lim_{N\to\infty} \int_{-b}^{b} \sum_{n=-N}^{N} \frac{1}{2b} f\left(\frac{n}{2b}\right) e^{-\frac{2\pi i n k}{2b}} e^{2\pi i k x} \, d\lambda(k) \\ &= \lim_{N\to\infty} \sum_{n=-N}^{N} \int_{-b}^{b} \frac{1}{2b} f\left(\frac{n}{2b}\right) e^{-\frac{2\pi i n k}{2b}} e^{2\pi i k x} \, d\lambda(k) \\ &= \sum_{n\in\mathbb{Z}} \frac{1}{2b} f\left(\frac{n}{2b}\right) \int_{-b}^{b} e^{2\pi i k \left(-\frac{n}{2b} + x\right)} \, d\lambda(k) \\ &= \sum_{n\in\mathbb{Z}} \frac{1}{2b} f\left(\frac{n}{2b}\right) \frac{\left[e^{2\pi i k \left(-\frac{n}{2b} + x\right)} \right]_{-b}^{b}}{2\pi i \left(-\frac{n}{2b} + x\right)} \\ &= \sum_{n\in\mathbb{Z}} f\left(\frac{n}{2b}\right) \frac{e^{\pi i n - 2\pi i b x} - e^{-\pi i n + 2\pi i b x}}{2i(\pi n - 2\pi b x)} \\ &= \sum_{n\in\mathbb{Z}} f\left(\frac{n}{2b}\right) \frac{\sin(\pi(n - 2bx))}{\pi(n - 2bx)} = \sum_{n\in\mathbb{Z}} f\left(\frac{n}{2b}\right) \operatorname{sinc}(n - 2bx). \end{aligned}$$

■

Corollary 2.7 (Nyquist–Shannon sampling theorem) Let f be as in Theorem 2.6. Then, for $(x_n)_{n\in\mathbb{Z}}$, where $x_n = \frac{n}{2b}$ for $n \in \mathbb{Z}$ the function f is completely determined by $(x_n, f(x_n))_{n\in\mathbb{Z}}$ thanks to equation (2.2). The frequency b is called *Nyquist frequency*.

Note that the converse of the Nyquist–Shannon sampling theorem does not hold in general, i.e. there are functions that can be completely reconstructed from samples whose abscissa frequency is lower than $2b$.

The question arises if the sampling formula also holds true for functions not fulfilling the condition of bandlimitedness in the sense that the formula converges to the function to be approximated for large sampling frequencies. In fact, this turns out to be true as we show with the following theorem.

Theorem 2.8 (Arbitrary approximation with the sampling formula) Let $f \in L^2(\mathbb{R})$ be such that $\hat{f} \in C^1(\mathbb{R})$. Let

$$\tilde{f}_b : \mathbb{R} \longrightarrow \mathbb{R}, \quad x \longmapsto \sum_{n\in\mathbb{Z}} f\left(\frac{n}{2b}\right) \operatorname{sinc}(n-2bx),$$

for a $b > 0$. Then, $\|f - \tilde{f}_b\|_{L^2} = \|\mathbb{1}_{[-b,b]^c}\hat{f}\|_{L^2} \xrightarrow{b\to\infty} 0$.

Proof. Define $\hat{f}_b := \hat{f} \cdot \mathbb{1}_{[-b,b]}$. Then, as in the proof of Lemma 2.6, we can expand $\hat{f}_b$ as Fourier series

$$\hat{f}_b(k) = \mathbb{1}_{[-b,b]}(k) \cdot \sum_{n\in\mathbb{Z}} c_n e^{-\frac{2\pi ikn}{2b}}, \qquad c_n = \frac{1}{2b}\int_{-b}^{b} \hat{f}_b(k) e^{\frac{2\pi ikn}{2b}}\, d\lambda(k),$$

where the equality for the limit holds in $L^2[-b,b]$ by the Riesz-Fischer theorem. And, also from the proof of Lemma 2.6, we now that $(\hat{f}_b)^\vee = \tilde{f}_b$. Now, with Plancherel's theorem we get

$$\|f - \tilde{f}_b\|_{L^2} = \|\hat{f} - \hat{f}_b\|_{L^2} = \|\hat{f} - \mathbb{1}_{[-b,b]}\hat{f}\|_{L^2} = \|\mathbb{1}_{[-b,b]^c}\hat{f}\|_{L^2}.$$

Since $\mathbb{1}_{[-b,b]^c}\hat{f} \xrightarrow{b\to\infty} 0$ pointwise and $\hat{f}$ is an integrable majorant as $\|\hat{f}\|_{L^2} = \|f\|_{L^2} < \infty$, we get by the dominated convergence theorem that $\|\mathbb{1}_{[-b,b]^c}\hat{f}\|_{L^2(\mathbb{R})} \xrightarrow{b\to\infty} 0$. ■

Example 2.9 We regard the rectangle function $f := \mathbb{1}_{[-1,1]}$. Since f is not differentiable at the points $\{-1,1\}$ it is intuitively clear that this leads to infinite frequency components. So it seems that the prerequisites for Lemma 2.6 are not fulfilled, as the support of $\hat{f}$ is not contained in an interval. Nevertheless, since $f \in L^2(\mathbb{R})$ and $\hat{f} = 2\operatorname{sinc}(2\bullet) \in C^1(\mathbb{R})$ (see below) we know by Theorem 2.8 that the sampling formula is applicable and that when the sampling frequency is increased to infinity, the approximation error vanishes. Interpolating f at the points $\left(\frac{n}{2b}\right)_{n\in\mathbb{Z}}$ for a $b > 0$ using the sampling formula (2.2) yields the approximation

$$\tilde{f}_b(x) = \sum_{n\in\mathbb{Z}} \mathbb{1}_{[-1,1]}\left(\frac{n}{2b}\right) \operatorname{sinc}(n-2bx) = \sum_{n=\lceil -2b\rceil}^{\lfloor 2b\rfloor} \operatorname{sinc}(n-2bx).$$

In order to examine the convergence of the approximation we compute the Fourier transform of f restricted to the interval $[-b,b]$. For that, recall that $\widehat{\mathbb{1}_{[-\frac{1}{2},\frac{1}{2}]}} = \operatorname{sinc}$ and it is a basic property of the Fourier transform, that for $g \in L^1(\mathbb{R})$, it holds for almost every $k \in \mathbb{R}$ that

$$\widehat{g(a\bullet)}(k) = \frac{1}{|a|}\hat{g}\left(\frac{k}{a}\right), \qquad a \in \mathbb{R}. \tag{2.6}$$

The Fourier transform $\hat{f}$ is now obtained by

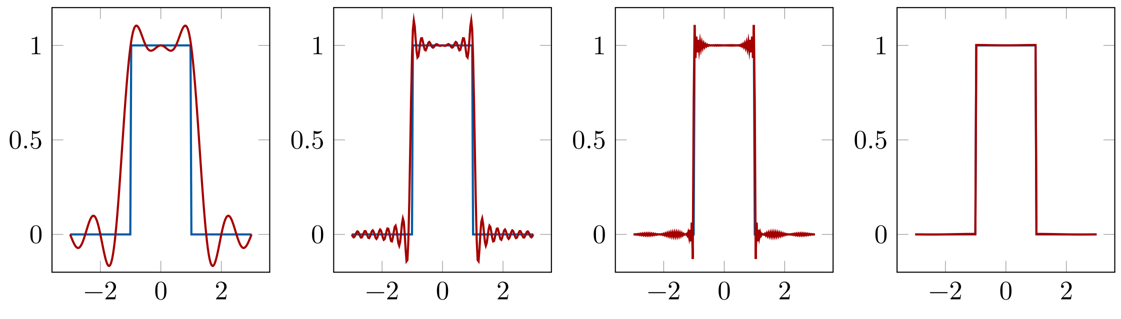
$$\hat{f}(k) = \widehat{\mathbb{1}_{[-1,1]}}(k) = \widehat{\mathbb{1}_{[-\frac{1}{2},\frac{1}{2}]}\left(\frac{\bullet}{2}\right)}(k) \overset{(2.6)}{=} 2 \cdot \widehat{\mathbb{1}_{[-\frac{1}{2},\frac{1}{2}]}}(\bullet)(2k) = 2\,\mathrm{sinc}(2k).$$

and we define its bandlimited version by $\hat{f}_b(k) := \hat{f}(k)\mathbb{1}_{[-b,b]} = 2\,\mathrm{sinc}(2k) \cdot \mathbb{1}_{[-b,b]}$. We get the approximation of f represented by the sampling formula by back transformation, namely

$$\tilde{f}_b(x) := \int_{\mathbb{R}} \hat{f}_b(k)e^{2\pi ikx}\,d\lambda(k) = \int_{-b}^{b} \hat{f}(k)e^{2\pi ikx}\,d\lambda(k) = 2\int_{-b}^{b} \mathrm{sinc}(2k)e^{2\pi ikx}\,d\lambda(k).$$

Theorem 2.8 gives as an estimation of the approximation error under the L^2-norm.

$$\begin{aligned}
\|f - \tilde{f}_b\|_{L^2}^2 &= \|\hat{f} \cdot \mathbb{1}_{[-b,b]^{\complement}}\|_{L^2}^2 = \int_{[-b,b]^{\complement}} |\hat{f}(k)|^2\,d\lambda(k) \\
&= 4\int_{[-b,b]^{\complement}} \mathrm{sinc}^2(2k)\,d\lambda(k) = 4\int_{[-b,b]^{\complement}} \frac{\sin^2(2\pi k)}{(2\pi k)^2}\,d\lambda(k) \\
&\leq \frac{1}{\pi^2}\int_{[-b,b]^{\complement}} \frac{1}{k^2}\,d\lambda(k) = \frac{1}{\pi^2}\left(\left[\frac{-1}{k}\right]_{-\infty}^{-b} + \left[\frac{-1}{k}\right]_{b}^{\infty}\right) = \frac{2}{\pi^2 b} \xrightarrow{b\to\infty} 0.
\end{aligned}$$



(a) b=1 (b) b=4 (c) b=16 (d) b=100

Figure 2.4: Approximations of f (blue) by $\tilde{f}_b$ (red) for different values of b. As can be seen sampling f, which is not bandlimited, with low frequency yields an interpolating function alternating around the actual function. By increasing the sampling frequency the interpolation error gets smaller.

2.2.2 Aliasing artefacts

In the previous section, we used the Nyquist-Shannon sampling theorem to obtain a sufficient condition to capture frequencies from a bandlimited image with a sensor that has the appropriate pixel pitch. If this condition is not met, frequencies above the Nyquist frequency may be mistakenly perceived as lower frequencies. These then appear as unwanted artefacts in the recorded image and cannot be detected or corrected afterwards.

In theory, a debayer algorithm can only reliably reconstruct frequencies below the Nyquist frequency, but since the pixels are divided into colour channels, it is already a challenge not to produce unnecessary aliasing effects for frequencies below the Nyquist frequency. Although the converse of the Nyquist-Shannon sampling theorem does not hold, we must assume that there are always frequencies that cannot be detected. Especially since images from nature are generally not band-limited. To prevent aliasing effects from frequencies above the Nyquist frequency, an optical low pass filter (OLPF) is placed in front

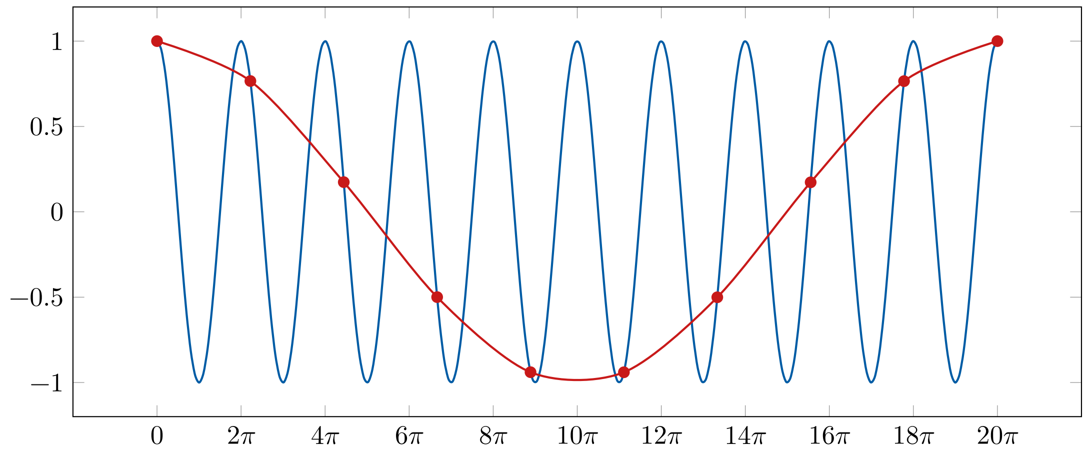
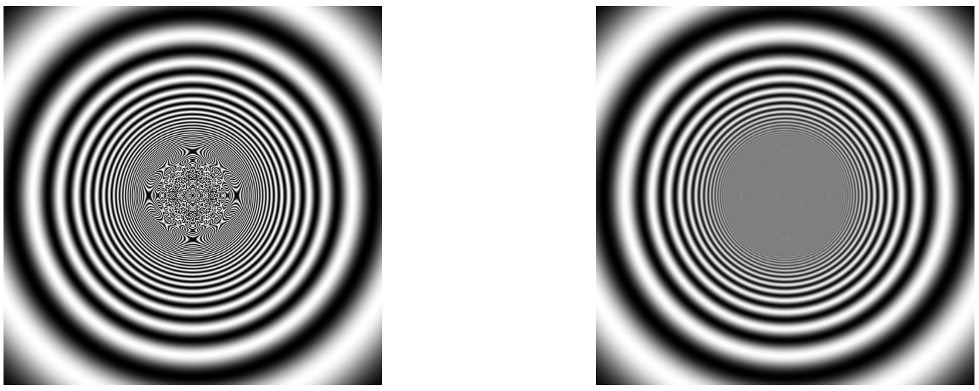


Figure 2.5: Illustration of the aliasing effect. The blue curve is a cosine signal with frequency $\frac{1}{2\pi}$. The red curve is created by sampling the cosine with frequency $\frac{9}{20\pi}$. As can be seen, the red curve suggests a lower frequency as is actually contained in the original signal.

of the sensor to remove these frequencies.



(a) No filtering has been applied. Deficient frequencies can be observed in the high-frequency ranges.

(b) An OLPF simulation has been applied. The alias artefacts are reduced. See [24] for more information about OLPF simulation.

Figure 2.6: Example for aliasing artefacts. The function $\left[-\frac{\pi}{10}, \frac{\pi}{10}\right]^2 \setminus (0,0) \longrightarrow \mathbb{R},\ x \longmapsto \sin\left(\frac{1}{|x|}\right)$ has been sampled at 512 equidistant points to a greyscale image.

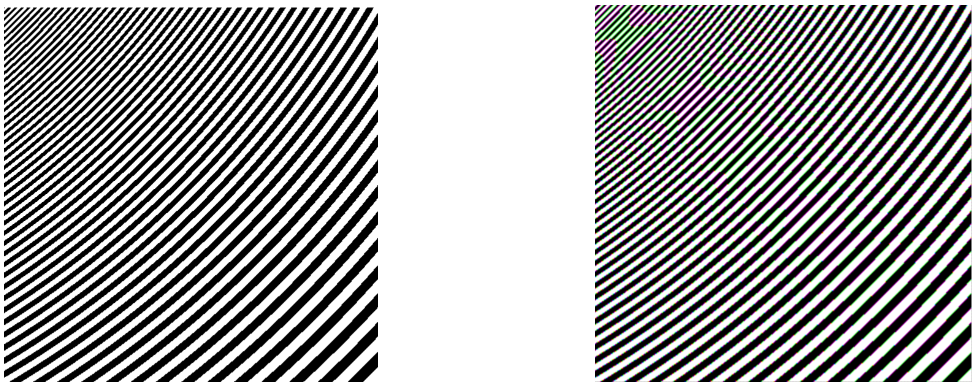
Remark 2.10 (Error estimation for the aplication of an OLPF) The application of an OLPF leads to the question of how much information is lost by cutting off high frequencies. For this purpose, the L^2 norm is suitable, especially since it allows the application of Plancherel's theorem. In addition, we would like to know the error of the n-th derivative, since this characterises the error for different types of interpolation models. For example, in linear interpolation, the error is characterised by the second derivative. All this can be expressed by the difference of the original with the band-limited function under the Sobolev norm H^n. We assume that the OLPF is such that $\hat{f}$ is restricted to the interval $[-b, b]$.

$$\|\mathcal{F}^{-1}(\hat{f}) - \mathcal{F}^{-1}(\hat{f} \cdot \mathbb{1}_{[-b,b]})\|_{H^n} = \sum_{m=1}^{n} \|\partial^m(\mathcal{F}^{-1}(\hat{f})) - \mathcal{F}^{-1}(\hat{f} \cdot \mathbb{1}_{[-b,b]})\|_{L^2}$$

$$
\begin{aligned}
&= \sum_{m=1}^{n} \|\partial^m \mathcal{F}^{-1}(\hat{f} - \hat{f} \cdot \mathbb{1}_{[-b,b]})\|_{L^2} \\
&= \sum_{m=1}^{n} \|\partial^m \mathcal{F}^{-1}(\hat{f} \cdot \mathbb{1}_{[-b,b]^\complement})\|_{L^2} \\
&= \sum_{m=1}^{n} \|\mathcal{F}^{-1}((-2\pi i k)^m \cdot \hat{f} \cdot \mathbb{1}_{[-b,b]^\complement})\|_{L^2} \\
\text{(Plancherel's theorem)} \quad &= \sum_{m=1}^{n} \|(-2\pi i k)^m \cdot \hat{f} \cdot \mathbb{1}_{[-b,b]^\complement}\|_{L^2} \xrightarrow{b\to\infty} 0
\end{aligned}
$$

2.2.3 False colour

In order to obtain the missing spectral components debayer algorithms interpolate values from the surrounding pixels. Since the colours of an image are determined by which pixels of the Bayer pattern respond to a stimulus, there is a direct relationship between position and colour. Errors in the interpolation of the debayer map can lead to colour shifts. This effect is generally more severe if the edges are achromatic.



(a) Ground truth image I (b) Debayered image $d_B(m_B(I))$

Figure 2.7: Example for false colour artefacts. The function $\left[\frac{5\pi}{512}, \frac{\pi}{60}\right]^2 \longrightarrow \mathbb{R},\ x \longmapsto \text{sgn}\left(\sin\left(\frac{1}{|x|}\right)\right)$ has been sampled (without filtering) at 512 equidistant points in each direction to a black and white image that can be written as an element of $\mathbb{R}^{512\times512}$. It has then been copied three times along a new axis such that we get a black and white image $I \in \mathbb{R}^{512\times512\times3}$. We refer to I as the ground truth image. We then used bilinear interpolation (see Section 2.3.1) as debayering map d_B for computing $d_B(m_B(I))$. As can be seen, there are faulty green and magenta colours on some edges.

2.3 Classic debayering strategies

For the implementation of a debayering map as neural network it is necessary to incorporate a priori knowledge into the hyperparameters. For this reason, it makes sense to follow-up with classic implementations of debayering maps. This allows us to, at least in part, convey established algorithmical structures to the neural networks.

Many debayer algorithms have been proposed and many of those used in industry today are subject to operational secrecy, as well as the *ARRI debayer algorithm 7* (ADA7). For this reason, no explicit description of ADA7 is given in this thesis, but Section 2.3.2 shows the principle approach used in ADA7, so as in the most commonly used debayer algorithms.

2.3.1 Bilinear interpolation

For a Bayer pattern B the goal of a debayering map $d_B$is to reconstruct a given image $I \in \mathbb{R}^{m\times n\times 3}$ from its *bayered* version $m_B(I)$ such that the visual difference between I and $d_B(m_B(I))$ is minimised. One of the simplest approaches to do this is bilinear interpolation. The algorithm relies on interpolating the channels independently by linearly averaging the available pixels with the same colour of the channel to be computed. Figure 2.8 shows a bayer pattern with the corresponding indexing of the pixels, which can be helpful for reading algorithms 2.11 and 2.12.

Algorithm 2.11 (Bilinear debayering without border treament) Assume we are given an image $I \in \mathbb{R}^{m\times n\times 3}$, $m, n \in 2\mathbb{N}$ and the corresponding mosaic image $J := m_B(I)$.

```
1:  Initialise $\tilde{I} \in \mathbb{R}^{m\times n\times 3}$
2:  for $i = 4, 6, \dots, m-2$ do
3:      for $j = 4, 6, \dots, n-2$ do
4:          $\tilde{I}_{i,j,2} \leftarrow \frac{J_{i,j-1}+J_{i,j+1}+J_{i-1,j}+J_{i+1,j}}{4}$
5:          $\tilde{I}_{i-1,j-1,2} \leftarrow \frac{J_{i-1,j-2}+J_{i-1,j}+J_{i-2,j-1}+J_{i,j-1}}{4}$
6:          $\tilde{I}_{i,j,1} \leftarrow \frac{J_{i-1,j-1}+J_{i-1,j+1}+J_{i+1,j-1}+J_{i+1,j+1}}{4}$
7:          $\tilde{I}_{i,j-1,1} \leftarrow \frac{J_{i-1,j-1}+J_{i+1,j-1}}{2}$
8:          $\tilde{I}_{i-1,j,1} \leftarrow \frac{J_{i-1,j-1}+J_{i-1,j+1}}{2}$
9:          $\tilde{I}_{i-1,j-1,2} \leftarrow \frac{J_{i-2,j-2}+J_{i-2,j}+J_{i,j-2}+J_{i,j}}{4}$
10:         $\tilde{I}_{i,j-1,3} \leftarrow \frac{J_{i,j-2}+J_{i,j}}{2}$
11:         $\tilde{I}_{i-1,j,1} \leftarrow \frac{J_{i-2,j}+J_{i,j}}{2}$
12:     end for
13: end for
14: return $\tilde{I}$
```

2.3.2 Gradient based interpolation

Many of the current debayer algorithms are based on gradient-based interpolation. In a first step, the green channel is reconstructed, whereby care is taken to interpolate along the edges as far as possible. The green information is then used to calculate the red and blue channels.

🟥 $J_{i-3,j-3}$	🟩 $J_{i-3,j-2}$	🟥 $J_{i-3,j-1}$	🟩 $J_{i-3,j}$	🟥 $J_{i-3,j+1}$	🟩 $J_{i-3,j+2}$
🟩 $J_{i-2,j-3}$	🟦 $J_{i-2,j-2}$	🟩 $J_{i-2,j-1}$	🟦 $J_{i-2,j}$	🟩 $J_{i-2,j+1}$	🟦 $J_{i-2,j+2}$
🟥 $J_{i-1,j-3}$	🟩 $J_{i-1,j-2}$	🟥 $J_{i-1,j-1}$	🟩 $J_{i-1,j}$	🟥 $J_{i-1,j+1}$	🟩 $J_{i-1,j+2}$
🟩 $J_{i,j-3}$	🟦 $J_{i,j-2}$	🟩 $J_{i,j-1}$	🟦 $J_{i,j}$	🟩 $J_{i,j+1}$	🟦 $J_{i,j+2}$
🟥 $J_{i+1,j-3}$	🟩 $J_{i+1,j-2}$	🟥 $J_{i+1,j-1}$	🟩 $J_{i+1,j}$	🟥 $J_{i+1,j+1}$	🟩 $J_{i+1,j+2}$
🟩 $J_{i+2,j-3}$	🟦 $J_{i+2,j-2}$	🟩 $J_{i+2,j-1}$	🟦 $J_{i+2,j}$	🟩 $J_{i+2,j+1}$	🟦 $J_{i+2,j+2}$

Figure 2.8: Visualisation of the submatrix of J that is used to compute $(\tilde{I}_{s,t,u})_{s\in\{i-1,i\},t\in\{j-1,j\},u\in[3]}$ for $i \in \{4, 6, \dots, m-2\}, j \in \{4, 6, \dots, n-2\}$ in the setting of algorithms 2.11 and 2.12.

Algorithm 2.12 (Gradient based debayering without border treatment) Assume we are given an image $I \in \mathbb{R}^{m\times n\times 3}$, $m, n \in 2\mathbb{N}$ and the corresponding mosaic image $J := m_B(I)$. Let $\epsilon \geq 0$ be an edge detection parameter.

```
1:  Initialise $\tilde{I} \in \mathbb{R}^{m\times n\times 3}$
2:  for $i = 4, 6, \dots, m-2$ do
3:      for $j = 4, 6, \dots, n-2$ do
```

```
4:          $\alpha \leftarrow \left|\frac{J_{i,j-2}+J_{i,j+2}}{2} - J_{i,j}\right|$
5:          $\beta \leftarrow \left|\frac{J_{i-2,j}+J_{i+2,j}}{2} - J_{i,j}\right|$
6:          if $\alpha < \beta + \epsilon$ then
7:              $\tilde{I}_{i,j,2} \leftarrow \frac{J_{i,j-1}+J_{i,j+1}}{2}$
8:          else if $\beta < \alpha + \epsilon$ then
9:              $\tilde{I}_{i,j,2} \leftarrow \frac{J_{i-1,j}+J_{i+1,j}}{2}$
10:         else
11:             $\tilde{I}_{i,j,2} \leftarrow \frac{J_{i-1,j}+J_{i+1,j}+J_{i,j-1}+J_{i,j+1}}{4}$
12:         end if
13:         $\alpha \leftarrow \left|\frac{J_{i-1,j-3}+J_{i-1,j+1}}{2} - J_{i-1,j-1}\right|$
14:         $\beta \leftarrow \left|\frac{J_{i-3,j-1}+J_{i+1,j-1}}{2} - J_{i-1,j-1}\right|$
15:         if $\alpha < \beta + \epsilon$ then
16:             $\tilde{I}_{i-1,j-1,2} \leftarrow \frac{J_{i-1,j-2}+J_{i-1,j}}{2}$
17:         else if $\beta < \alpha + \epsilon$ then
18:             $\tilde{I}_{i-1,j-1,2} \leftarrow \frac{J_{i-2,j-1}+J_{i,j-1}}{2}$
19:         else
20:             $\tilde{I}_{i-1,j-1,2} \leftarrow \frac{J_{i-2,j-1}+J_{i,j-1}+J_{i-1,j-2}+J_{i-1,j}}{4}$
21:         end if
22:         $\tilde{I}_{i-1,j,1} \leftarrow \frac{(J_{i-1,j-1}-\tilde{I}_{i-1,j-1,2})+(J_{i-1,j+1}-\tilde{I}_{i-1,j+1,2})}{2} + J_{i-1,j}$
23:         $\tilde{I}_{i,j-1,1} \leftarrow \frac{(J_{i-1,j-1}-\tilde{I}_{i-1,j-1,2})+(J_{i+1,j-1}-\tilde{I}_{i+1,j-1,2})}{2} + J_{i,j-1}$
24:         $\tilde{I}_{i,j,1} \leftarrow \frac{(J_{i-1,j-1}-\tilde{I}_{i-1,j-1,2})+(J_{i+1,j-1}-\tilde{I}_{i+1,j-1,2})+(J_{i-1,j+1}-\tilde{I}_{i-1,j+1,2})+(J_{i+1,j+1}-\tilde{I}_{i+1,j+1,2})}{4} + J_{i,j}$
25:         $\tilde{I}_{i,j-1,3} \leftarrow \frac{(J_{i,j-2}-\tilde{I}_{i,j-2,2})+(J_{i,j}-\tilde{I}_{i,j,2})}{2} + J_{i,j-1}$
26:         $\tilde{I}_{i-1,j,1} \leftarrow \frac{(J_{i-2,j}-\tilde{I}_{i-2,j,2})+(J_{i,j}-\tilde{I}_{i,j,2})}{2} + J_{i-1,j}$
27:         $\tilde{I}_{i-1,j-1,1} \leftarrow \frac{(J_{i-2,j-2}-\tilde{I}_{i-2,j-2,2})+(J_{i,j-2}-\tilde{I}_{i,j-2,2})+(J_{i-2,j}-\tilde{I}_{i-2,j,2})+(J_{i,j}-\tilde{I}_{i,j,2})}{4} + J_{i-1,j-1}$
28:     end for
29: end for
30: return $\tilde{I}$
```

In the lines 4 - 20 the missing green values at the red and blue bayer positions are computed. Because of the representation of the luminance channel of the green pixels on the bayer pattern and their higher sampling rate, these contain the most information about the course of edges. For this reason, this channel is reconstructed first and is used later to calculate the missing blue and red channels.

In the lines 4 - 11 the green value $\tilde{I}_{i,j,2}$ at the blue Bayer position $J_{i,j}$ is computed. For that, $\tilde{I}_{i,j,2}$ is obtained by averaging the surrounding green pixels either horizontally (line 7), vertically (line 9) or over all four surrounding pixels (line 11). The decision how to interpolate depends on whether an edge is detected and in which direction it runs. This is determined by the parameters α and β which are computed in lines 4 and 5. These characterise horizontally and vertically how much the signal changes in the vicinity of the pixel to be interpolated. Then these changes are compared and depending on this the interpolation method is chosen in such a way that interpolation does not proceed across an edge. The edge detection parameter ϵ determines the tolerance for the comparison of α and β. In the lines 13 - 20 the same procedure is repeated to compute the the missing green value $\tilde{I}_{i-1,j-1,2}$ at the red Bayer position $J_{i-1,j-1}$.

Once the luminance is determined, the chrominance values are obtained from the differences between the colour and luminance signals. Depending on the position of the adjacent pixels, this is done either horizontally, vertically or by averaging all four adjacent pixels of the same colour. This is done in lines 22-24 for the red pixels and in lines 25-27 for the blue pixels.

Note that the calculation of the pixels at the edge must be carried out separately, as no gradients or averages can be calculated there in the way just described.

2.4 CNN architectures

The choice of the neural network architecture is crucial for trainability, image quality and evaluation performance. The architecture allows to influence the hypothesis space in such a way that a priori knowledge can be incorporated. For this thesis two different types of architectures were examined. The first architecture is based on a simple U-Net structure and is more general. The second was inspired by gradient based debayering algorithms and relies on reconstructing the green channel before using it to reconstruct the red and blue channels. The latter architecture turned out to be far superior in terms of image quality and training speed.

2.4.1 U-Net

A promising architecture to be applied to the debayering problem is the U-Net. It was first proposed in [23] for biomedical image segmentation, but has since been used for many other applications in image processing. The architecture consists of a contracting path to capture context and a symmetric expanding path that enables translation of local information.

Originally, CNNs were used for classification tasks. Here, a label is assigned to an image. This is usually done by scaling the image down from layer to layer, for example by a strided convolution layer, a pooling layer or by reshaping. In this way the rear layers of the CNNs can have a large perceptive area even though the may have comparatively small filter sizes. In the case of image-to-image translation tasks, however, there may also be image information that requires precise localisation at the same time. For this purpose, a CNN with downscaling layers is supplemented symmetrically with the corresponding upscaling layers so that the size of the output image corresponds to the input. Each of these layer pairs is also provided with skip connections so that local information can bypass the downscaling and upscaling.

For the experiments in this thesis, the U-Net has an input layer with filter size 6 and stride 2, which scales down the input image by factor 2. The input layer has one input channel (as the sensor image in the form of the Bayer pattern is one dimensional) and 256 output channels. This allows the network to extract different features, such as frequency components of the image. The input layer is followed by a depthwise convolutional layer (see Remark 1.14) with filter size 3, stride 1 and 256 input and output channels. This layer has the function of increasing the complexity of the mapping represented by the CNN. Also, it increases the perceptive area of the network as by the stride of the input layer the joint filter size of these first two layers is 10 by 10 pixels. Note that the 2D depthwise convolution can also be replaced by a 2D convolution with three dimensional filters. However, the experiments in the scope of this thesis showed that the more computationally efficient 2D depthwise convolutions achieve the same image quality. The depthwise convolutional layer is followed by ten so-called dense convolution layers, i.e. convolutional layers with filters that have a kernel of size 1 by 1. These are a very resource-efficient way to handle additional non-linearity and detail. Note, that although they operate pixel-wise they may have a spatial effect as they operate along the channel dimension which is spatially correlated since we are calculating on a lower resolution space. They can also be regarded as a pixel-wise operating multilayer perceptron. A transposed convolutional layer is used as the output of the network (see [10] for the definition and more information about transposed convolutional layers). Although this is not a mathematical inverse of the convolutional layer, the transformations of the input layer in terms of shapes can be undone. This is, the lower resolution image which is consumed by the deep layers is scaled up again. The output of the input 2D convolutional layer is added as a residual connection to the output of the multilayer perceptron layer in order to avoid the vanishing gradient problem and to conserve low-level features (see Remark 1.34).

Compared to ADA7, the U-Net architecture could not achieve the same image quality. Especially in high-frequency areas, significantly more aliasing effects can be observed. Figure 2.10 shows a section of an image of a Siemens star that was debayered with the U-Net. In the high-frequency areas, the aliasing effects are visible in the form of wave patterns.

Although testing models on images of Siemens stars does by far not reflect the variety of problematic areas, it is an necessary condition for a debayer map to be able to reconstruct a Siemens star with few aliasing effects. For this reason the approach of using a U-Net was not pursued further.

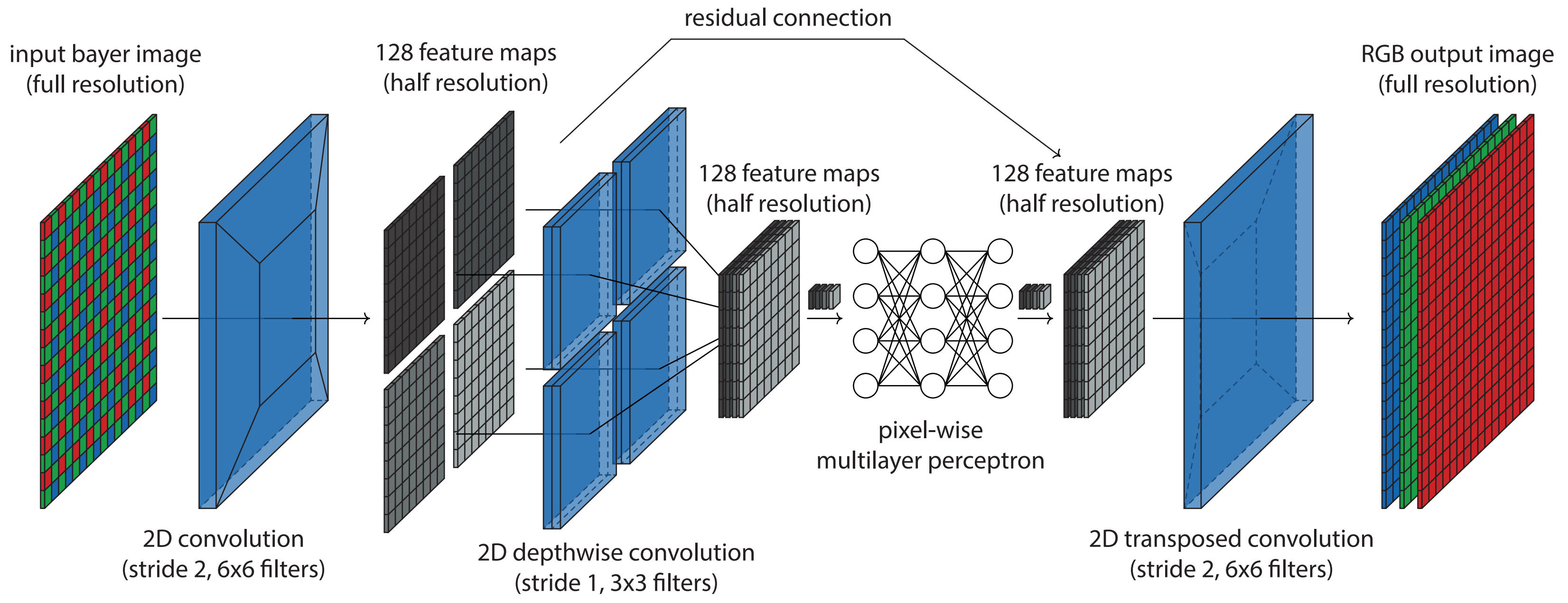
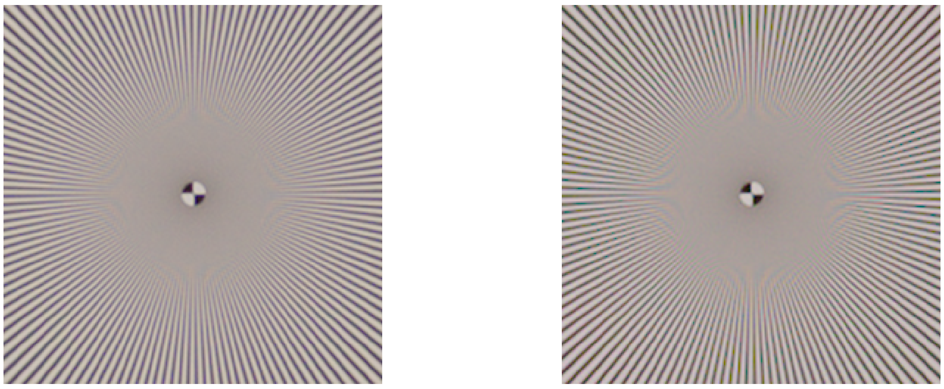


Figure 2.9: Illustration of the U-Net used for the debayering task. The feature maps between the input and output layer are of half resolution referred to the amount of pixels along the image boundaries, i.e. each channel has the quarter of pixels compared to the input.



(a) ADA7 (b) U-Net

Figure 2.10: Evaluation of ADA7 and the U-Net on the Siemens star test chart.

2.4.2 3-stage CNN

As we know from the no-free-lunch theorems, it is necessary to integrate as much a priori knowledge as possible into the models used. The choice of CCNs alone, as well as the hyperparameters, such as the filter size, are already a use of prior knowledge. Nevertheless, if we take a look at the quite successful gradient-based debayer algorithms, there is still prior information that we have not yet used. For example, the fact that the luminance information is mainly in the green pixels, which also have the highest density in the CFA. In Algorithm 2.12 this fact is used by reconstructing the green channel first, and then using it to recover the red, and green channels. [9] proposes an architecture for general image restoration tasks based on the same procedure. This so-called *3-stage architecture* splits the input image in form of the Bayer pattern into a red, a blue, and two green subimages. These are treated as channels and computed with a network consisting of a special concatenation of multiple CNNs. This network returns 12 channels, 4 of which correspond to one colour channel each. This 12 channels with half the amount of pixels as compared to the input image can then be reshaped to three full resolution image channels. Except for the reshaping at the beginning and at the end, the networks can be fully described using the operators that we defined in Section 1.2.2.

The reshaping at the input and the output is referred to as *pixel unshuffle* and *pixel shuffle* respectively and has been proposed by [25]. It is defined generally as a bijective map transforming an element $x \in \mathbb{R}^{n \times l \times m \cdot r^2}$ to an element $\mathfrak{S}(x) = y \in \mathbb{R}^{r \cdot n \times r \cdot l \times m}$, $n, l, m, r \in \mathbb{N}$, where

$$y_{i,j,c} := x_{\lfloor \frac{i}{r} \rfloor, \lfloor \frac{j}{r} \rfloor, m \cdot r \cdot \mathrm{mod}(j,r) + m \cdot \mathrm{mod}(i,r) + c} \qquad \forall i \in [n], j \in [l], c \in [m]. \tag{2.7}$$

In our concrete case, suppose the network input is a Bayer pattern image $x^{\text{input}} \in \mathbb{R}^{2n \times 2l}$, $n, l \in \mathbb{N}$. We write $\mathfrak{S}$ for the pixel shuffle operator. Then, the unshuffled input is an element

$$\mathfrak{S}^{-1}(x^{\text{input}}) =: x^{\text{unshuffled}} = \sum_{k=1}^{4} x_k^{\text{unshuffled}} \otimes e_k \in \mathbb{R}^{n \times l} \otimes \mathbb{R}^4,$$

where $\{e_k\}_{k=1}^4$ is a orthonormal basis of $\mathbb{R}^4$ and

$$(x_1^{\text{unshuffled}})_{i,j} = x_{2i,2j}^{\text{input}}, \qquad (x_2^{\text{unshuffled}})_{i,j} = x_{2i+1,2j}^{\text{input}},$$
$$(x_3^{\text{unshuffled}})_{i,j} = x_{2i,2j+1}^{\text{input}}, \qquad (x_4^{\text{unshuffled}})_{i,j} = x_{2i+1,2j+1}^{\text{input}},$$

for all $i \in [n], j \in [l]$. Now, $(x_1^{\text{unshuffled}})_{i,j}$ contains the red, $(x_2^{\text{unshuffled}})_{i,j}$ and $(x_3^{\text{unshuffled}})_{i,j}$ contain the green, and $(x_4^{\text{unshuffled}})_{i,j}$ contains the blue pixels of the mosaic image.

The counterpart of the pixel unshuffling at the input of the network is the pixel shuffling at the output. It takes the output $x^{\text{shuffled}} \in \mathbb{R}^{n\times l} \otimes \mathbb{R}^{12}$ of the 3rd CNN stage (see Algorithm 2.13) and reshapes it to an RGB image $x^{\text{output}} := \mathfrak{S}(x^{\text{shuffled}}) \in \mathbb{R}^{2n\times 2l} \otimes \mathbb{R}^3$, where

$$\begin{aligned}
(x_1^{\text{output}})_{2i,2j} &= (x_1^{\text{shuffled}})_{i,j} & (x_2^{\text{output}})_{2i,2j} &= (x_1^{\text{shuffled}})_{i,j} & (x_3^{\text{output}})_{2i,2j} &= (x_1^{\text{shuffled}})_{i,j}\\
(x_1^{\text{output}})_{2i+1,2j} &= (x_2^{\text{shuffled}})_{i,j} & (x_2^{\text{output}})_{2i+1,2j} &= (x_2^{\text{shuffled}})_{i,j} & (x_3^{\text{output}})_{2i+1,2j} &= (x_2^{\text{shuffled}})_{i,j}\\
(x_1^{\text{output}})_{2i,2j+1} &= (x_3^{\text{shuffled}})_{i,j} & (x_2^{\text{output}})_{2i,2j+1} &= (x_3^{\text{shuffled}})_{i,j} & (x_3^{\text{output}})_{2i,2j+1} &= (x_3^{\text{shuffled}})_{i,j}\\
(x_1^{\text{output}})_{2i+1,2j+1} &= (x_4^{\text{shuffled}})_{i,j} & (x_2^{\text{output}})_{2i+1,2j+1} &= (x_4^{\text{shuffled}})_{i,j} & (x_3^{\text{output}})_{2i+1,2j+1} &= (x_4^{\text{shuffled}})_{i,j},
\end{aligned}$$

for all $i \in [n], j \in [l]$. Note that in comparison to the definition of pixel shuffle in equation (2.7) the channels have be permuted in order to simplify the implementation. Now we are ready to fully describe the 3-stage architecture. We do this by explicitly describe the forwards pass of the model as an algorithm.

Algorithm 2.13 (3-stage CNN forward pass) The 3-stage CNN architecture is defined by three stages. For each of them, we define the CNN functions

$$\begin{aligned}
f^{\text{stage 1}} &\in \mathsf{CNN}_{10}((4,63,63,63,63,63,63,63,63,12),3;3;\psi^{\text{relu}};1)\\
f_R^{\text{stage 2}} &\in \mathsf{CNN}_{10}((8,63,63,63,63,63,63,63,63,4),3;3;\psi^{\text{relu}};1)\\
f_B^{\text{stage 2}} &\in \mathsf{CNN}_{10}((8,63,63,63,63,63,63,63,63,4),3;3;\psi^{\text{relu}};1)\\
f^{\text{stage 3}} &\in \mathsf{CNN}_{10}((12,63,63,63,63,63,63,63,63,12),3;3;\psi^{\text{relu}};1),
\end{aligned}$$

where CNN is defined as in Section 1.2.3 and ψ^{relu} is defined as in Section 2.5.2. For an input element $x^{\text{input}} \in \mathbb{R}^{2n\times 2l}$, $n, l \in \mathbb{N}$ the forward pass is defined as follows

```
1: $\mathbb{R}^{n\times l} \otimes \mathbb{R}^4 \ni x^{\text{unshuffled}} \leftarrow \mathfrak{S}^{-1}(x^{\text{input}})$
2: $x^{\text{stage 1 residual}} \leftarrow \sum_{k=1}^{4} x_1^{\text{unshuffled}} \otimes e_k^{\text{stage 1}} + \sum_{k=5}^{6} x_2^{\text{unshuffled}} \otimes e_k^{\text{stage 1}} + \sum_{k=7}^{8} x_3^{\text{unshuffled}} \otimes e_k^{\text{stage 1}} + \sum_{k=9}^{12} x_4^{\text{unshuffled}} \otimes e_k^{\text{stage 1}}$ for a orthonormal basis $\{e_k^{\text{stage 1}}\}_{k=1}^{12}$ of $\mathbb{R}^{12}$.
3: $x^{\text{stage 1}} \leftarrow f^{\text{stage 1}}(x^{\text{unshuffled}}) + x^{\text{stage 1 residual}}$
4: $x^{\text{stage 1 R}} \leftarrow \sum_{k=1}^{4} x_k^{\text{stage 1}} \otimes e_k^{\text{stage 1}}$
5: $x^{\text{stage 1 RG}} \leftarrow \sum_{k=1}^{8} x_k^{\text{stage 1}} \otimes e_k^{\text{stage 1}}$
6: $x^{\text{stage 1 B}} \leftarrow \sum_{k=9}^{12} x_k^{\text{stage 1}} \otimes e_k^{\text{stage 1}}$
7: $x^{\text{stage 1 GB}} \leftarrow \sum_{k=5}^{12} x_k^{\text{stage 1}} \otimes e_k^{\text{stage 1}}$
8: $x^{\text{stage 1 G}} \leftarrow \sum_{k=5}^{8} x_k^{\text{stage 1}} \otimes e_k^{\text{stage 1}}$
9: $x^{\text{stage 2 R}} \leftarrow f_R^{\text{stage 2}}(x^{\text{stage 1 RG}}) + x^{\text{stage 1 R}}$
10: $x^{\text{stage 2 B}} \leftarrow f_B^{\text{stage 2}}(x^{\text{stage 1 RB}}) + x^{\text{stage 1 B}}$
11: $x^{\text{stage 2}} \leftarrow \sum_{k=1}^{4} x_k^{\text{stage 2 R}} \otimes e_k^{\text{stage 2}} + \sum_{k=5}^{8} x_k^{\text{stage 1 G}} \otimes e_k^{\text{stage 2}} + \sum_{k=9}^{12} x_k^{\text{stage 2 B}} \otimes e_k^{\text{stage 2}}$
12: $x^{\text{stage 3}} \leftarrow f^{\text{stage 3}}(x^{\text{stage 2}}) + x^{\text{stage 2}}$
13: $x^{\text{output}} \leftarrow \mathfrak{S}(x^{\text{stage 3}})$
```

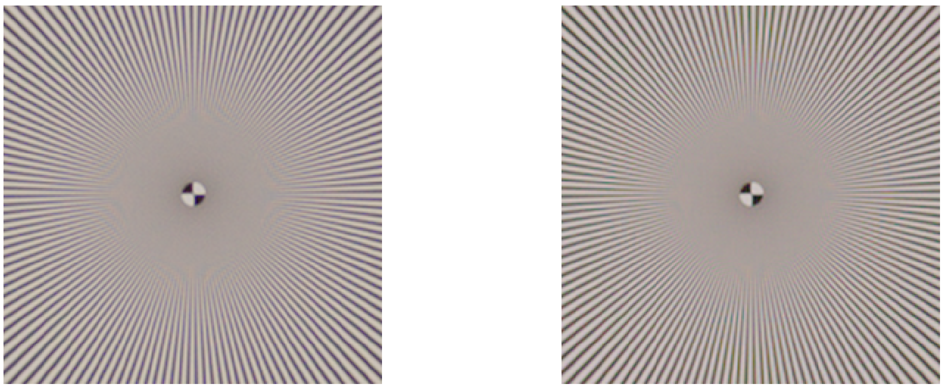
If, when adding residual connections, the summands are not in the same basis, the corresponding transformation must be applied first. This has been omitted from the notation for reasons of simplicity.

In line 1 of Algorithm 2.13 the input is unshuffled to the four subimages of the Bayer pattern as described above. Stage 1 increases the amount of channels from 4 to 12. Since we want to use a residual connection, the bypassed residual element has to be scaled accordingly. For that, the residual element of stage one is created in line 2 by concatenating copies of the channels of $x^{\text{unshuffled}}$ along the new channel axis.

Note, that we have to make four copies of the red, and blue channels, while we only need two copies of each green channel in order to get the 12-channeled residual. In line 2 the CNN-function of stage one is applied and the residual element is added. In the lines 4 to 8 various combinations of channels are extracted from the output of stage one for the next stage. The element $x^{\text{stage 1 G}}$ is the enhanced green channel and will be consumed by stage two for reconstructing the red and the blue channel. For that, both, the red and the green channel $x^{\text{stage 1 RG}}$ are fed into the stage two function for the red channel $f_R^{\text{stage 2}}$ in line 9. Since we want to receive the red channel, only $x^{\text{stage 1 R}}$ is added as a residual connection. The blue channel is computed with $f_B^{\text{stage 2}}$ in the same way. After that, the red and blue outputs of stage two and the green output of stage one are concatenated to the output $x^{\text{stage 2}}$ of stage two in line 11. As a final step of enhancement, the whole feature $x^{\text{stage 2}}$ is propagated through the CNN-function of stage three and the corresponding residual is added in line 12. Finally, pixel shuffle is applied in line 12 in order to obtain the output full resolution RGB image.

The number of channels in the deep layer and the size of the filters were varied and examined for differences in image quality. Larger values than those used here did not contribute to a visible increase in quality. However, it is quite possible that similarly good results can be achieved with smaller networks of similar structure. Note that although a filter size of 3x3 may seem small, each filter actually covers a larger area on the full resolution image due to the pixel shuffle. In fact, a 3x3 filter corresponds to an area of 6x6 pixels.

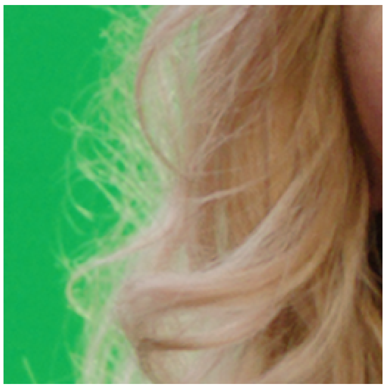
The 3-stage architecture is able to reconstruct more spatial frequencies of a Siemens star than ADA7 (see Figure 2.11).



(a) ADA7 (b) 3-stage CNN

Figure 2.11: Evaluation of ADA7 and the 3-stage CNN on the Siemens star test chart. As can be seen, the 3-stage architecture shows a clearer image and less aliasing effects in the high frequency areas.

The 3-stage CNN also appears to be superior to ADA7 in terms of image quality on other test data. A particularly important requirement for a debayering algorithm is good performance on green screen data. Although in most cases the 3-stage CNN can reconstruct more detail and achieve better contrast (see Figure 2.12), in some rare cases there are also more aliasing effects than in ADA7 (see Figure 2.13). On the most images showing nature, the 3-stage architecture seems to be superior in terms of image quality (see Figure 2.14)


(a) ADA7

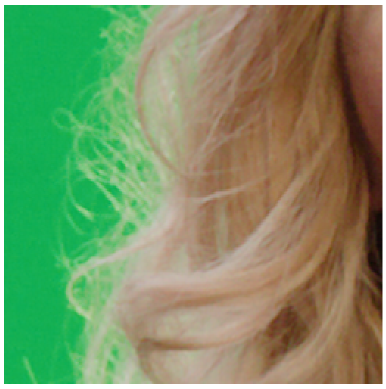
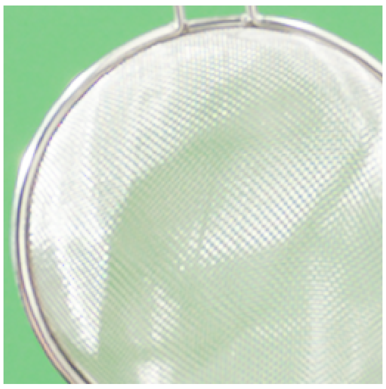

(b) 3-stage CNN

Figure 2.12: Evaluation of ADA7 and the 3-stage CNN on an image showing blonde hair in front of a green screen. As can be seen, the 3-stage architecture is able to reconstruct more details and shows more contrast as compared to ADA7.


(a) ADA7

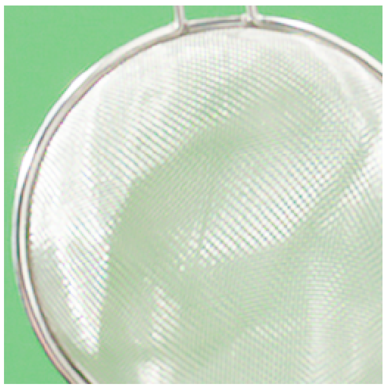
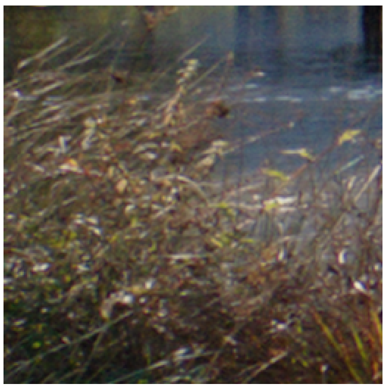

(b) 3-stage CNN

Figure 2.13: Evaluation of ADA7 and the 3-stage CNN on an image showing a sieve in front of a green screen. As can be seen, the 3-stage architecture shows some faulty colours in the high frequency areas and some staircase effects on the edges.


(a) ADA7

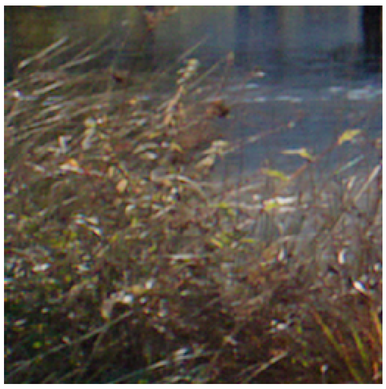

(b) 3-stage CNN

Figure 2.14: Evaluation of ADA7 and the 3-stage CNN on an image showing thicket in front of a lake. As can be seen, the 3-stage architecture is able to reconstruct more detail than ADA7 but also shows more coloured artefacts.

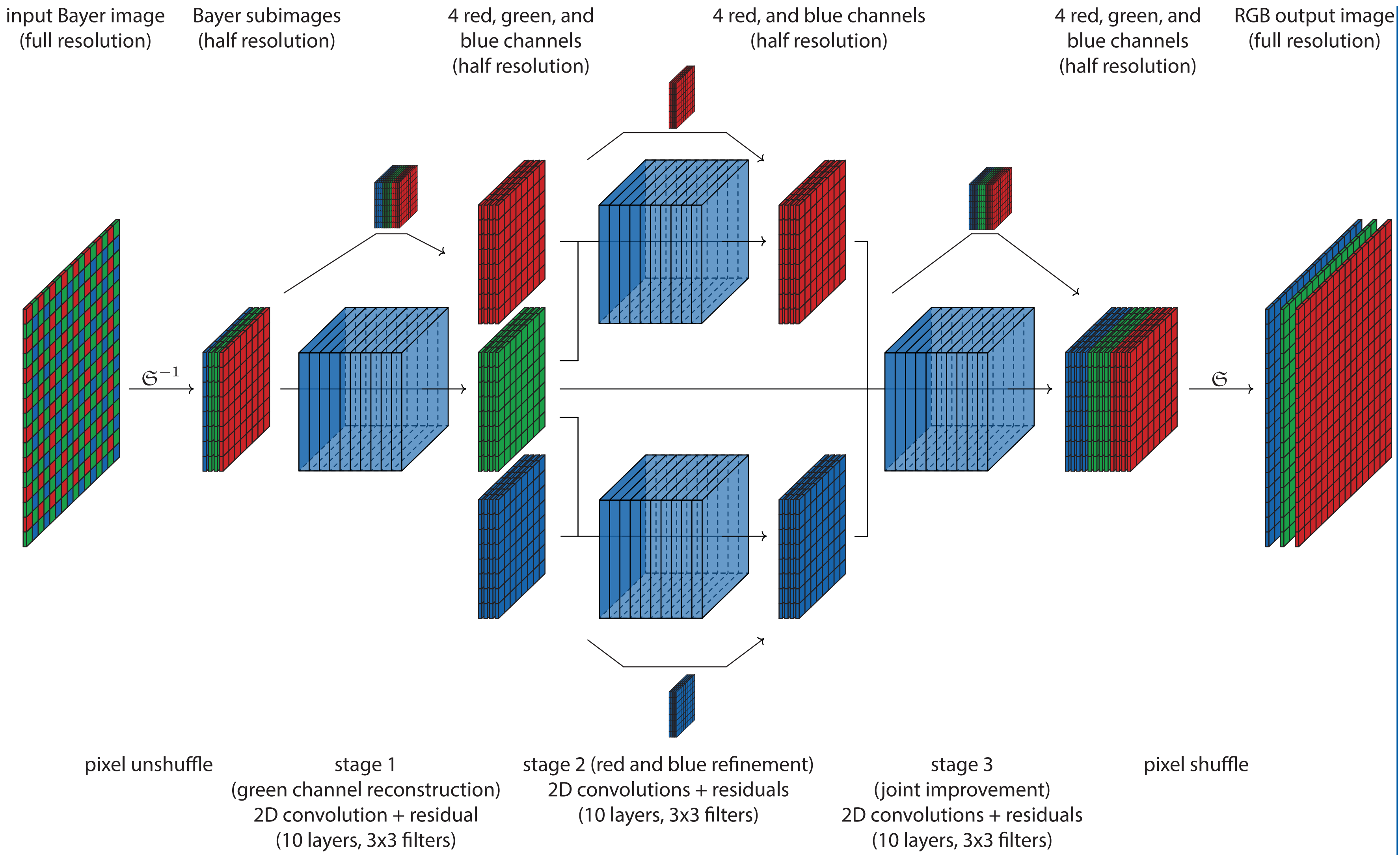


Figure 2.15: Illustration of the 3-stage CNN used for the debayering task. The feature maps between the pixel unshuffle, and pixel shuffle operators are of half resolution referred to the amount of pixels along the image boundaries, i.e. each channel has the quarter of pixels compared to the input.

2.5 Training

2.5.1 Data

One of the most central challenges is obtaining training data. This must cover all features of the problem to be learned. In the case of demosaicing, for example, this means that the entire set of shapes in the sense of the course of edges is represented in as many frequencies, signal values and colours as possible. On the other hand, images that come from nature are not evenly distributed in space of all shapes and colours and one can question whether the training data should reflect this. If the distribution of the data is weighted with respect to a spurious distribution, the risk is high that the neural network to be trained exhibits a bias, since the neural network reduces the expected value of the loss function if it preferentially reflects this distribution.

The question arises whether images can be used that were actually taken with a camera. Provided that these reflect a large variability of natural scenes, they would fulfil the requirements for representativeness of the natural distribution. The difficulty here is to obtain the ground truth data, i.e. RGB images $(I_i)_{i\in[s]}, I \in \mathbb{R}^{m\times n\times 3}, m, n \in 2\mathbb{N}, s \in \mathbb{N}$ since the RAW data is presumably only available in the form of images as Bayer patterns, i.e. of the form $(J_i)_{i\in[s]}, J \in \mathbb{R}^{m\times n}$. Using an existing debayer algorithm to obtain I from J would bias to the weak spots of the debayer algorithm, and hence, the neural network would never surpass the algorithm in terms of image quality. In order to create ground truth full RGB reference data, [3] took the approach of placing a filter wheel in front of a special ARRI Alexa research camera in which the CFA has been removed. The filter wheel is placed in front of the lens and can be equipped with red, green, and blue filters similar as also contained in a CFA. In this way, the non-existent spectral resolution of the camera can be compensated for by temporally staggered images. The main drawback is that it is not possible to record real sequences with arbitrary image content. In fact, the scene recorded while the wheel is rotating must be motionless so that the images associated with each colour filter do not show relative spatial variations that could cause unwanted colour artefacts.

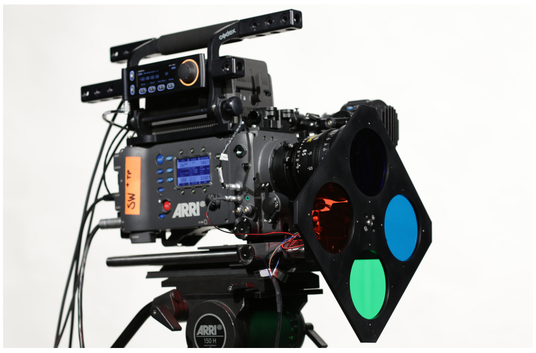
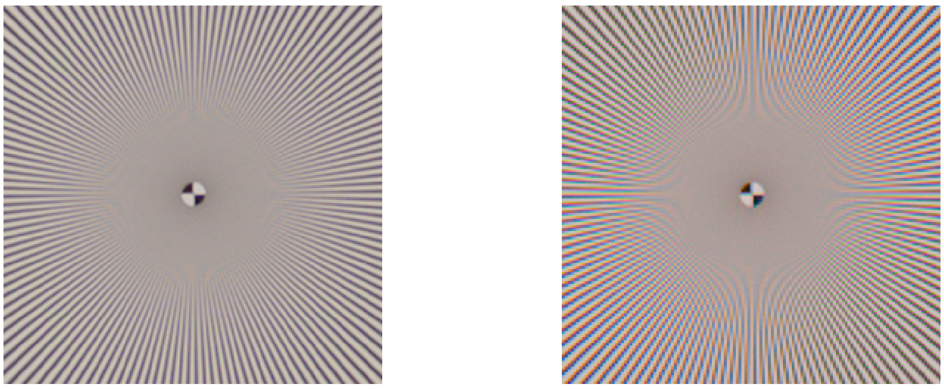


Figure 2.16: ARRI Alexa provided with a colour wheel. The picture was taken from [3].

Another possibility to obtain training data from camera images is by downscaling. Assume we are given a bayer image $J \in \mathbb{R}^{m\times n}$. With the pixel shuffle operator (see Section 2.4.2) we obtain an image $\mathfrak{S}(J) =: \tilde{J} \in \mathbb{R}^{\frac{m}{2}\times\frac{m}{2}\times 4}$ from which we can define a red channel $R := (\tilde{J}_{i,j,1})_{i,j}$, a green channel $G := \frac{(\tilde{J}_{i,j,2})_{i,j}+(\tilde{J}_{i,j,3})_{i,j}}{2}$ and a blue channel $B := (\tilde{J}_{i,j,4})_{i,j}$, i.e. we have a full RGB image $(R, G, B) \in \mathbb{R}^{\frac{m}{2}\times\frac{m}{2}\times 3}$. Since the image is halved in resolution also the Nyquist frequency is halved. This may lead to aliasing effects as the OLPF's modulation transfer function (MTF) is configured for double the resolution. To prevent this we can apply a OLPF simulation to (R, G, B). To do this, however, oversampling is usually necessary, which means that the resolution is reduced again, for example by half or a quarter. This approach was investigated in the context of this work, but did not lead to satisfactory results. High frequencies seem not to have been learned, or learned incorrectly (see Figure 2.17). It is possible that the MTF in the OLPF simulation needs to be configured specifically for the training application. This approach could be followed in future work.



(a) ADA7 (b) CNN trained with real camera data

Figure 2.17: Evaluation of ADA7 and the 3-stage CNN (see Section 2.4.2) trained with camera data, that has been downscaled with the pixel shuffle operator and processed with the OLPF simulation.

The most successful approach to generating training data in this thesis was synthetic data based on the basic shapes of Siemens stars and chirps. A *Siemens star* in grey scale with $s \in \mathbb{N}$ *sparks*, centered at $x^\star \in [0,1]^2$ is a function

$$\begin{aligned} S_{x^\star,s} :\ [-1,1]^2 &\longrightarrow \mathbb{R} \\ x &\longmapsto \cos\left(s \cdot \arctan\left(\frac{x_2^\star - x_2}{x_1^\star - x_1}\right)\right) \end{aligned}$$

and a *chirp* at the same center with *frequency scaler* $t \in \mathbb{R}$ is a function

$$\begin{aligned} C_{x^\star,t} :\ [-1,1]^2 \setminus \{x^\star\} &\longrightarrow \mathbb{R} \\ x &\longmapsto \sin\left(\frac{t}{\|x - x^\star\|}\right). \end{aligned}$$

In order to get sharp edges, binary images (that can then be offsetted and scaled) may be desired. This can be attained by taking the absolute value of the function outputs. Coloured versions of siemens stars and chirps can be generated, for example by defining a RGB-scaler triple $(r,g,b) \in \mathbb{R}^3$ and then regarding the function

$$\begin{aligned} \tilde{S}_{x^\star,s} :\ [-1,1]^2 &\longrightarrow \mathbb{R}^3 \\ x &\longmapsto (rS_{x^\star,s}, gS_{x^\star,s}, bS_{x^\star,s}). \end{aligned}$$

The same applies for chirps. For the training data, Siemens stars and chirps were varied in colour, randomly scaled in size and overlaid several times. It was ensured that the Siemens stars in some images had enough sparks to represent high-frequency images, close to the Nyquist frequency. An OLPF simulation, specifically designed to reflect the ARRI ALEV 4 sensor was applied. The training images were sampled in a resolution of 60 by 60 pixels and encoded in the ARRI qlut domain.

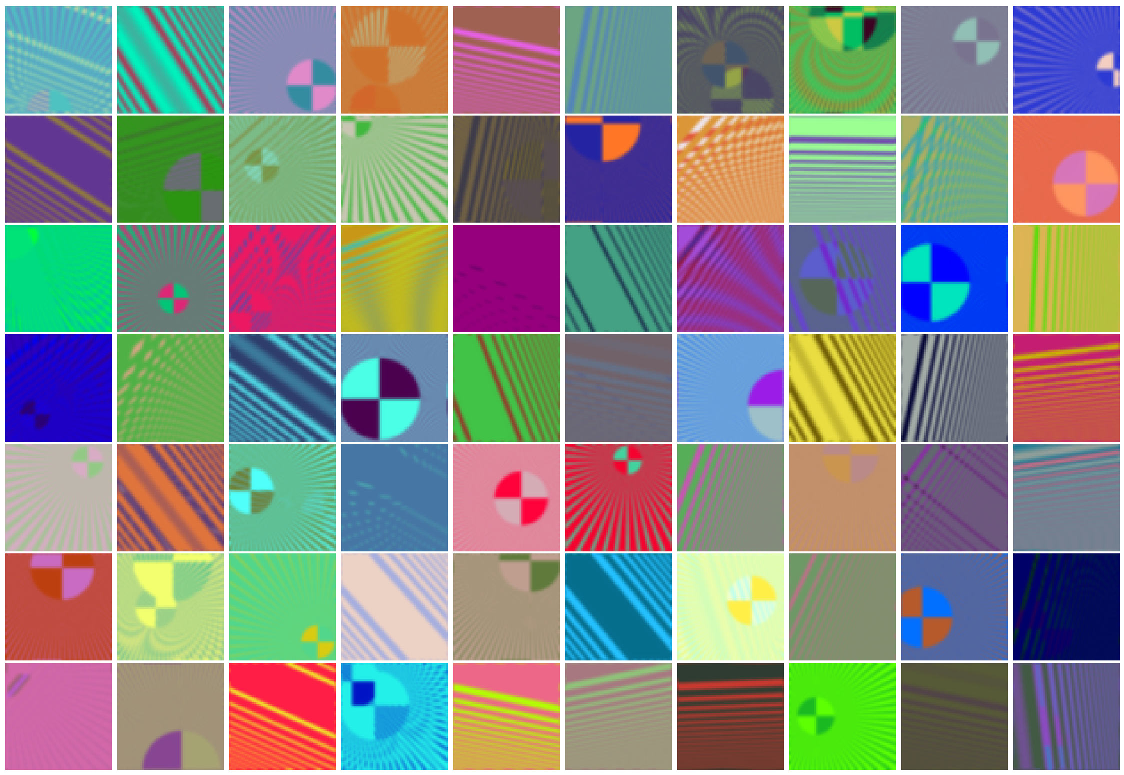


Figure 2.18: 70 (of the 20000) examples of the synthetic training data used for training. Since the training data is encoded in qlut it was linearised to be presented as 8-bit images in this thesis.

2.5.2 Hyperparameters

In machine learning, *hyperparameters* are parameters used to control the learning process. In contrast to parameters as discussed in Chapter 1, hyperparameters are not obtained by the training process but have to be fixed in advance. The most important hyperparameters for the demosaicing application are the *batch size*, the *activation function*, the *loss function*, the *learning rate* and the choice of an optimiser algorithm.

- The **batch size** describes how many training data samples are propagated forwardly through the neural network at a gradient decent update. The resulting gradients are averaged before being used for backpropagation. A higher batch size reduces the risk of the training getting stuck in a local minimum but requires at the same time more computer memory. Note that the size of training images, regarded as hyperparameter is closely related to the batch size. For example, doubling the resolution (amount of pixels) of the training images by at the same time halving the batch size results in the same amount of pixels used for one training iteration with the only difference being that the variation of image content is reduced. A batch size of 32 gave good results, although no great effect was found when varying the batch size between 8 and 64.

- In Definition 1.10 we defined the layerwise function which is applied element-wise at the end of a convolutional layer. In concrete application, this is usually referred to as the **activation function**. The *rectified linear unit (ReLu)*-function

$$\psi^{\text{relu}} : \mathbb{R} \longrightarrow \mathbb{R}, \quad x \longmapsto x \cdot \mathbb{1}_{\mathbb{R}_+}(x).$$

has established itself as the standard activation function in the field of deep learning. This gave good results and was used in the proposed 3-stages architecture (Section 2.4.2). However, in some experiments with deep neural networks without residual connections the vanishing gradient problem occurred and the training did not converge. In this case using the *scaled exponential linear unit (SeLu)*-function

$$\psi^{\text{selu}}_{\alpha,\lambda} : \mathbb{R} \longrightarrow \mathbb{R}, \quad x \longmapsto \lambda \begin{cases} x, & x > 0 \\ \alpha e^x - \alpha, & x \leq 0 \end{cases}, \quad \alpha, \lambda \in \mathbb{R},$$

was effective in counteracting the problem. The values $\alpha = 1.6733$ and $\lambda = 1.0507$ have been used, as proposed by [16].

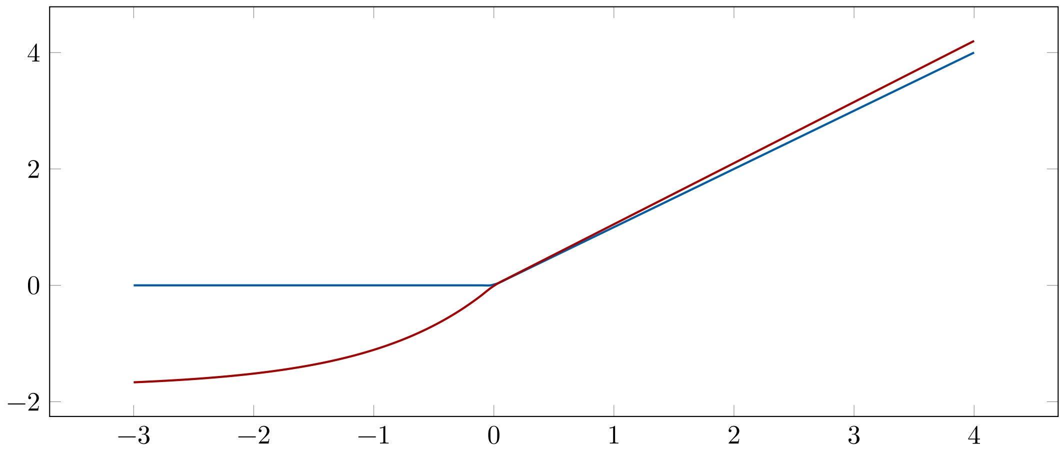


Figure 2.19: The ReLu (blue), and the SeLu (red) functions.

- The **learning rate**, also called *step size*, refers to the scaling factor for the gradient in a gradient decent algorithm (it was denoted as h in Theorem 1.16). Since the optimisation of a CNNs is not convex the training has been started with a high learning rate (10^{-3}) and then be reduced by multiplicative steps of 10^{-1} if the loss function did not decrease after an epoch (a complete pass through of all training data). The training was stopped if there was no improvement at a learning rate of 10^{-9}.

3 Generative adversarial networks

In the previous chapter we used a mean squared error to compare the error of the output of the neural network to be trained with the ground-truth sample point and update the parameters accordingly. However, this metric has limited representativeness for the actual human perception of the distance between two images. First, the loss function is calculated in the qlut domain, in which the distance between colours is not consistent with human perception. On the other hand, spatially occurring artefacts can be perceived by humans as a strong deviation, although they are not accordingly reflected in the loss function which is only computed pixel-wise.

The question arises whether the loss function can be generalised in such a way that even more abstract features can be captured without having to be explicitly defined. One approach is to replace the loss function with a neural network. This learns to pay attention to the errors of the neural network that is actually being trained and to improve it in this way. This type of training is called *adversarial training*, because the two neuronal networks can also be seen as opponents that try to outsmart each other.

This approach is used in the so called *generative adversarial networks (GANs)*, which are even more general. In fact, they don't even require a set of training pairs consisting of features and labels but only rely on a distribution of data which they try to imitate. This kind of training is referred to as *unsupervised learning*. By reviewing [13] and providing more detailed proofs, we are going to study GANs in their general form and show that, under ideal mathematical preconditions, any distribution can be captured by training a GAN. We also consider how the gradient decent algorithm from Chapter 1 can be used for the actual implementation. Finally, we discuss how GANs can be applied to the demosaicing problem.

Attempts to apply GANs to the demosaicing problem have not yielded satisfactory results. In this sense, this chapter should be seen as a motivation for future investigation. Related work, such as [15], has produced impressive results with GANs for image-to-image translation tasks. This suggests that they can also be used for demosaicing.

3.1 Probabilistic formulation

3.1.1 The general setting

In order to be able to properly calculate with the occurring probability distributions, we want to be able to write the measures as integrals over densities. For this we have to assume that the distributions are absolutely continuous with respect to the Lebesgue measure later in this section. In addition, we will presume that the densities are continuous and compactly supported in order to ensure the finiteness of integrals.

Let $d \in \mathbb{N}$ and $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space. We consider an unknown random variable $X : \Omega \longrightarrow \mathbb{R}^d$ and assume that we are only given a finite set $\mathcal{X} \subsetneq \mathbb{R}^d$ that consists of samples of the image of X. Our goal is to find a random variable $G : \Omega \longrightarrow \mathbb{R}^d$ such that the distribution $\mathbb{P}_G$ is a *good* approximation of $\mathbb{P}_X$, where G is implemented as a neural network. This leads to the question of how to measure the distance between $\mathbb{P}_X$ and $\mathbb{P}_G$. The original GAN paper [13] proposes to use another neural network, which represents a function $D : \mathbb{R}^d \longrightarrow (0,1)$ that tries to distinguish samples drawn from $\mathcal{X}$ from samples generated by G. On the other hand, G is trained to deceive D in such a way that D is no longer able to tell apart the fake samples from the real ones. This counterplay can be described by a single functional through which the training objective will be fully described.

Definition 3.1 (Gan loss functional) Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space and let $X : \Omega \longrightarrow \mathbb{R}^d$ be the unknown data random variable. We define the *GAN loss functional* by

$$\begin{aligned} V : L^0(\Omega, \mathbb{R}^d) \times L^0(\mathbb{R}^d, (0,1)) &\longrightarrow \bar{\mathbb{R}} \\ (G, D) &\longmapsto \mathbb{E}[\log(D \circ X)] + \mathbb{E}[\log(1 - D \circ G)]. \end{aligned}$$

D tries to distinguish between samples from X and G and uses $(0,1)$ as a rating scale. Samples from X should be assigned to values close to 1 and samples from G should be assigned to values close to 0. Hence, for a fixed generator G, training D means solving the optimisation problem $\max_D V(G, D)$.

We assume from now that $X : \Omega \longrightarrow \mathbb{R}^d$ is such that the pushforward measure $\mathbb{P}_X$ is absolutely continuous with respect to the Lebesgue measure λ^d. Then, by the theorem of Radon-Nikodým ([11], Theorem A.4.8), there is a probability density function $\nu \in L^1(\mathbb{R}^d)$, $\nu \geq 0$, $\int_{\mathbb{R}^d} \nu(x)\, d\lambda^d(x) = 1$, unique up to sets of measure zero in λ^d, such that for every Borel set $B \in \mathcal{B}(\mathbb{R}^d)$ we can write

$$\mathbb{P}(X \in B) = \int_B \nu(x)\, d\lambda^d(x).$$

As [27] points out, the assumption that $\mathbb{P}_X$ is absolutely continuous may be erroneous. For example, consider the case where $\mathcal{X}$ restricts to a lower dimensional sub-manifold of $\mathbb{R}^d$, making $\mathbb{P}_X$ a singular probability distribution. Nevertheless, this does not seem to be a problem in most applications. In order to be able to write the GAN loss-functional as an integral over $\mathbb{R}^d$ we also have to restrict the set of possible generators. For that we introduce the following notation.

Definition 3.2 (Continuous random variable) For a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ we write

$$\mathcal{L}_a^0(\Omega, \mathbb{R}^d) := \{f \in \mathcal{L}^0(\Omega, \mathbb{R}^d) \mid \mathbb{P} \circ f^{-1} \ll \lambda^d\}$$

for the set of real random variables that admit an absolute continuous pushforward measure. Let $L_a^0(\Omega, \mathbb{R}^d)$ be the corresponding normed quotient space, where all functions equaling almost everywhere are identified.

Proposition 3.3 Let $\nu \in L^1(\mathbb{R}^d)$ be the unknown density of $\mathbb{P}_X$. Then for all $G \in L_a^0(\Omega)$ there is a density function $\mu \in L^1(\mathbb{R}^d)$ such that

$$V(G, D) = \int_{\mathbb{R}^d} \nu(x) \log(D(x)) + \mu(x) \log(1 - D(x))\, d\lambda^d(x).$$

Proof. This is a direct consequence of the theorem of Radon Nikodým. Let μ be the Radon-Nikodým derivative $\frac{\mathbb{P}_G}{\lambda^d}$. Then

$$\begin{aligned} V(G,D) &= \int_\Omega \log(D(X(\omega))\, d\mathbb{P}(\omega) + \int_\Omega \log(1-D(G(\omega))\, d\mathbb{P}(\omega) \\ &= \int_\Omega \log(D(x))\, d\mathbb{P}_X(x) + \int_\Omega \log(1-D(x))\, d\mathbb{P}_G(x) \\ \begin{pmatrix}\text{Radon Nikodým,} \\ \mathbb{P}_X, \mathbb{P}_G \ll \lambda^d\end{pmatrix} \quad &= \int_{\mathbb{R}^d} \nu(x)\log(D(x)) + \mu(x)\log(1-D(x))\, d\lambda^d(x). \end{aligned}$$ ■

In the following statements we presuppose that $\mu, \nu \in C_c^0(\mathbb{R}^d)$. We take this theoretical loss of generality into account in order to ensure that the loss functional is finite. In practice, this condition is automatically fulfilled since computational memory is finite and neural networks represent continuous functions as composition of continuous functions.

3.1.2 The optimal discriminator

Using the representation of V from proposition 3.3 we can compute the optimal discriminator $D^\star$ for a given generator in terms of the unknown data distribution. This is therefore a purely theoretical result, but it can be used to further investigate the optimisation problem of GANs in terms of the existence of a solution.

Lemma 3.4 Let $K \subseteq \mathbb{R}^d$ be compact and $f \in C^0(K \times \mathbb{R})$. Assume that $y \longmapsto f(x,y)$ is strictly convex for almost every $x \in K$ and that $\partial_y f$ exists. If $u^\star \in C^0(K)$ is such that $(\partial_y f)(x, u^\star(x)) = 0$ for almost every $x \in K$, then $u^\star$ is a minimiser of the functional

$$I : C^0(K) \longrightarrow \mathbb{R}, \quad u \longmapsto \int_K f(x, u(x))\, d\lambda^d(x)$$

and as such unique up to Lebesgue zero sets.

Proof. Let $u \in C^0(K)$. First, we verify that $I(u) < \infty$. Since K is compact and $x \longrightarrow f(x,u(x))$ is continuous as composition of continuous functions we get

$$I(u) = \int_K f(x,u(x))\, d\lambda^d(x) \leq \|x \longmapsto f(x,u(x))\|_{L^\infty(K)} \cdot \lambda^d(K) < \infty.$$

From the convexity of $y \longmapsto f(x,y)$ we have that

$$f(x,u^\star(x)) + (\partial_y f)(x,u^\star(x)) \cdot (u(x) - u^\star(x)) < f(x,u(x)).$$

for almost every $x \in K$. This was proved in Lemma 1.15, equation (1.7) for the convex case. For the strictly convex case, it can be directly seen from the proof of Lemma 1.15 that the strict inequality holds. Since we have $\partial_y f(x,u^\star(x)) = 0$ for almost every $x \in K$ integrating implies $I(u^\star) < I(u)$ so $u^\star$ is the unique minimiser of I. ■

Proposition 3.5 (Optimal discriminator) Let $G \in L_a^0(\Omega, \mathbb{R}^d)$ be fixed, μ the density of $\mathbb{P}_G$ and ν the unknown density of $\mathbb{P}_X$. We assume $\mu, \nu \in C_c^0(\mathbb{R}^d)$ and let $K := \operatorname{supp}(\nu) \cup \operatorname{supp}(\mu)$. Then, the *optimal discriminator* $D^\star \in \operatorname{argmax}_D V(G,D)$ is unique up to Lebesgue zero sets and given by

$$D^\star : K \longrightarrow (0,1), \quad x \longmapsto \frac{\nu(x)}{\nu(x)+\mu(x)}. \tag{3.1}$$

Proof. Define

$$f : K \times (0,1) \longrightarrow \mathbb{R}, \quad (x,y) \longmapsto \nu(x) \log(y) + \mu(x) \log(1-y).$$

We have that $y \longmapsto f(x,y) \in C^2((0,1))$ and the derivatives are

$$\partial_y f(x,y) = \frac{\nu(x)}{y} - \frac{\mu(x)}{1-y}, \qquad \partial_y^2 f(x,y) = -\frac{\nu(x)}{y^2} - \frac{\mu(x)}{(1-y)^2}.$$

For $x \in K$ we have that $\partial_y^2 f(x,y) < 0$, and hence, that $y \longmapsto -f(x,y)$ is strictly convex for every $x \in K$. Define $D^\star$ as in equation (3.1), then $D^\star \in C^0(K)$ and $D^\star(K) \subseteq (0,1)$. Now, for $x \in \mathbb{R}^d$ we get

$$\begin{aligned}
\partial_y f(x, D^\star(x)) &= \frac{\nu(x)}{\frac{\nu(x)}{\nu(x)+\mu(x)}} - \frac{\mu(x)}{1 - \frac{\nu(x)}{\nu(x)+\mu(x)}} \\
&= \frac{\nu^2(x) + \nu(x)\mu(x)}{\nu(x)} - \frac{\nu(x)\mu(x) + \mu^2(x)}{\mu(x)} \\
&= \frac{\nu^2(x)\mu(x) + \nu(x)\mu^2(x) - \nu^2(x)\mu(x) - \nu(x)\mu^2(x)}{\nu(x)\mu(x)} = 0.
\end{aligned}$$

Lemma 3.4 implies that $D^\star$ is a minimiser of $u \longmapsto \int_{\mathbb{R}^d} f(x, u(x))\, d\lambda^d(x)$, and hence, a maximiser of $V(G, \bullet)$ and that it is unique up to Lebesgue zero sets. ■

Since we have assumed for X that P_X is absolutely continuous, we can consequently restrict the hypothesis space of G with the the same assumption. This means that for finding an optimal G it suffices to optimise over the set of densities. For that we define the virtual training criterion.

Remark 3.6 Note that if we assume that the optimal discriminator $D^\star$ is known we can rewrite the min-max criterion as

$$\begin{aligned}
\max_D V(G,D) = V(G, D^\star) \\
&\overset{(3.3)}{=} \int_{\mathbb{R}^d} \nu(x) \log(D^\star(x)) + \mu(x) \log(1 - D^\star(x))\, d\lambda^d(x) \\
&\overset{(3.5)}{=} \int_{\mathbb{R}^d} \nu(x) \log\left(\frac{\nu(x)}{\nu(x)+\mu(x)}\right) + \mu(x) \log\left(1 - \frac{\nu(x)}{\nu(x)+\mu(x)}\right) d\lambda^d(x) \\
&= \int_{\mathbb{R}^d} \nu(x) \log\left(\frac{\nu(x)}{\nu(x)+\mu(x)}\right) + \mu(x) \log\left(\frac{\mu(x)}{\nu(x)+\mu(x)}\right) d\lambda^d(x).
\end{aligned}$$

The training of the discriminator is the main difficulty in practice. One problem, for example, is that because of the interdependence of the generator and the discriminator, the training is generally rather unstable, since the topology of the minima of the discriminator's optimisation is constantly changing due to the training of the generator. The discriminator would therefore have to capture the entire complexity of the generator between each update of the generator's parameters, which is not possible in practice because the training time would become extremely long. However, in order to substantiate the reasonableness of the GAN concept, it is crucial to know whether, under the condition that the optimal discriminator is achieved, the distribution of the generator converges to the distribution of the data. To do this, we now assume the optimal discriminator and use it to define the *virtual training criterion*, which restricts the training problem to only one optimisation. We will then see that the minimum of the virtual training criterion is reached exactly when the generator reflects the data distribution.

Definition 3.7 (Virtual training criterion) Let $\nu \in C_c^0(\mathbb{R}^d)$ be the unknown data distribution. Then we define the *virtual training criterion* as

$$\begin{aligned} C:\ C_c^0(\mathbb{R}^d) &\longrightarrow \mathbb{R} \\ \mu &\longmapsto \int_{\mathbb{R}^d} \nu(x)\log\left(\frac{\nu(x)}{\nu(x)+\mu(x)}\right) + \mu(x)\log\left(\frac{\mu(x)}{\nu(x)+\mu(x)}\right)\, d\lambda^d(x). \end{aligned}$$

We now consider the virtual training criterion by expressing it in terms of the so-called *Kullback-Leibler divergence* and the *Jensen-Shannon divergence* derived from it. These serve the purpose of measuring the divergence of two probability distributions. The Kullback-Leibler divergence was first proposed in [17].

Definition 3.8 (Kullback–Leibler divergence) Let $\mu, \nu \in L^1(\mathbb{R}^d)$ be probability densities. The *Kullback–Leibler divergence* is defined as

$$\begin{aligned} \mathrm{KL}:\ L^1(\mathbb{R}^d)\times L^1(\mathbb{R}^d) &\longrightarrow \overline{\mathbb{R}} \\ (\mu,\nu) &\longmapsto \mathrm{KL}\,(\mu\,||\,\nu) := \int_{\mathbb{R}^d} \mu(x)\log\left(\frac{\mu(x)}{\nu(x)}\right)\, d\lambda^d(x). \end{aligned}$$

Lemma 3.9 For probability densities $\mu, \nu \in L^1(\mathbb{R}^d)$ it holds that $\mathrm{KL}\,(\mu\,||\,\nu) \geq 0$ and equality holds if and only if $\mu = \nu$.

Proof. If $\mu = \nu$ almost everywhere we have

$$\mathrm{KL}\,(\mu\,||\,\mu) = \int_{\mathbb{R}^d} \mu(x)\log\left(\frac{\mu(x)}{\mu(x)}\right)\, d\lambda^d(x) = \int_{\mathbb{R}^d} \mu(x)\log(1)\, d\lambda^d(x) = 0.$$

Since $-\log$ is a convex function we get with Jensen's inequality

$$-\mathrm{KL}\,(\mu\,||\,\nu) = \int_{\mathbb{R}^d} \mu(x)\log\left(\frac{\nu(x)}{\mu(x)}\right)\, d\lambda^d(x) \leq \log\left(\int_{\mathbb{R}^d} \mu(x)\frac{\nu(x)}{\mu(x)}\, dx\right) = \log(1) = 0 \tag{3.2}$$

Let's assume $\mathrm{KL}\,(\mu\,||\,\nu) = 0$. This implies equality in equation (3.2). But since $-\log$ is a strictly convex function this is by Jenson's inequality only possible if $x \longmapsto \log\left(\frac{\nu(x)}{\mu(x)}\right)$ is constant almost everywhere, namely $x \longmapsto \log\left(\frac{\nu(x)}{\mu(x)}\right) = 0$ almost everywhere, and hence $\nu = \mu$ in $L^1(\mathbb{R}^d)$. ■

Definition 3.10 (Jensen–Shannon divergence) Let μ, ν be as in Definition 3.8 and $\xi := \frac{1}{2}(\mu+\nu)$. The *Jensen–Shannon divergence* is defined as

$$\begin{aligned} \mathrm{JS}:\ L^1(\mathbb{R}^d)\times L^1(\mathbb{R}^d) &\longrightarrow \bar{\mathbb{R}} \\ (\mu,\nu) &\longmapsto \mathrm{JS}\,(\mu\,||\,\nu) := \frac{1}{2}\left(\mathrm{KL}\,(\mu\,||\,\xi) + \mathrm{KL}\,(\nu\,||\,\xi)\right). \end{aligned}$$

Lemma 3.11 For probability densities $\mu, \nu \in L^1(\mathbb{R}^d)$ it holds that $\mathrm{JS}\,(\mu\,||\,\nu) \geq 0$ and equality holds if and only if $\mu = \nu$.

Proof. The first implication can be computed directly. Assume $\mu = \nu$. Then we get with Lemma 3.9,

$$\text{JS}\left(\mu\,||\,\nu\right) = \frac{1}{2}\left(\text{KL}\left(\mu\middle\|\frac{1}{2}(\mu+\mu)\right) + \text{KL}\left(\mu\middle\|\frac{1}{2}(\mu+\mu)\right)\right) = \frac{1}{2}\left(\text{KL}\left(\mu\,||\,\mu\right) + \text{KL}\left(\mu\,||\,\mu\right)\right) = 0.$$

On the other hand

$$\text{JS}\left(\mu\,||\,\nu\right) = \frac{1}{2}\Bigg(\underbrace{\text{KL}\left(\mu\middle\|\frac{1}{2}(\mu+\nu)\right)}_{\text{(Lemma 3.9)}\ \geq 0} + \underbrace{\text{KL}\left(\nu\middle\|\frac{1}{2}(\mu+\nu)\right)}_{\text{(Lemma 3.9)}\ \geq 0}\Bigg) \geq 0. \tag{3.3}$$

If we presume $\text{JS}\left(\mu\,||\,\nu\right) = 0$ then equation (3.3) together with Lemma 3.9 imply $\mu = \frac{1}{2}(\mu+\nu) = \nu$. ■

Proposition 3.12 (Unique minimum of virtual training criterion) Let $\nu \in C_c^0(\mathbb{R}^d)$ be the unknown probability density. Then $C(\nu) = \min_{\{\mu\in C_c^0(\mathbb{R}^d)|\mu \text{ probability density}\}} C(\mu)$. In particular, this minimum is unique in $C_c^0(\mathbb{R})$ up to Lebesgue zero sets and $C(\nu) = -\log 4$.

Proof. Let $\mu \in C_c^0(\mathbb{R}^d)$ be a probability density. Then

$$\begin{aligned}
C(\mu) = \int_{\mathbb{R}^d} &\nu(x)\log\left(\frac{\nu(x)}{\nu(x)+\mu(x)}\right) + \mu(x)\log\left(\frac{\mu(x)}{\nu(x)+\mu(x)}\right)\\
&+ \left(\log(2) - \log(2)(\nu(x)+\mu(x))\right) d\lambda^d(x)\\
&= -\log(2)\int_{\mathbb{R}^d} \nu(x)+\mu(x)\,d\lambda^d(x)\\
&\quad + \int_{\mathbb{R}^d} \nu(x)\left(\log(2) + \log\left(\frac{\nu(x)}{\nu(x)+\mu(x)}\right)\right) d\lambda^d(x)\\
&\quad + \int_{\mathbb{R}^d} \mu(x)\left(\log(2) + \log\left(\frac{\mu(x)}{\nu(x)+\mu(x)}\right)\right) d\lambda^d(x)\\
(\mu,\nu\ \text{normalised})\quad &= -2\log(2)\\
&\quad + \int_{\mathbb{R}^d} \nu(x)\log\left(\frac{2\nu(x)}{\nu(x)+\mu(x)}\right) d\lambda^d(x)\\
&\quad + \int_{\mathbb{R}^d} \mu(x)\log\left(\frac{2\mu(x)}{\nu(x)+\mu(x)}\right) d\lambda^d(x)\\
&= -\log(4) + \text{KL}\left(\nu\middle\|\frac{1}{2}(\nu+\mu)\right) + \text{KL}\left(\mu\middle\|\frac{1}{2}(\nu+\mu)\right)\\
&= -\log(4) + 2\,\text{JS}\left(\mu\,||\,\nu\right).
\end{aligned}$$

From Lemma 3.11 we know that $\text{JS}\left(\mu\,||\,\nu\right) \geq 0$ and equality holds if and only if $\mu = \nu$ up to Lebesgue zero sets, which concludes the proof. ■

3.2 Algorithmical aspects

3.2.1 Parametrisation

In Section 3.1 we discussed GAN networks in the non-parametric setting, i.e. the neural networks representing G and D were regarded as elements of infinite dimensional function spaces. From now we regard the parametrised versions of G and D

$$\begin{aligned} \mathbb{R}^m &\longrightarrow (\mathbb{R}^s \longrightarrow K) & \mathbb{R}^n &\longrightarrow (K \longrightarrow (0,1)) \\ \theta &\longmapsto G_\theta & \xi &\longmapsto D_\xi . \end{aligned} \tag{3.4}$$

Since we want G to return hitherto unseen output, i.e. data that is not contained in the training set, G can't be fully determined by θ but also has to contain a prior distribution that draws random input samples for G_θ. For that, let's consider another random variable $Z : \Omega \longrightarrow \mathbb{R}^s$, $s \in \mathbb{N}$ and assume $G = G_\theta \circ Z$. Usually Z is chosen to be uniformly distributed. Now, the goal is to find parameters θ such that G_θ turns the distribution of Z into the distribution of X, i.e. $G_\theta \circ Z \sim X$. In practice, Z often is a noise generator that draws random numbers from a random seed, i.e. $Z : \mathbb{N} \longrightarrow \mathbb{R}^s$.

3.2.2 Minimisation of the virtual training criterion

We showed in proposition 3.12 that the virtual training criterion C has a unique minimum if the discriminator is optimal. Fulfilling this assumption in an optimisation algorithm for θ would mean optimising ξ til convergence between each update of θ, which could take an infinite amount of iterations. For that reason, we introduce a hyperparameter $K \in \mathbb{N}$ that determines how many training cycles for ξ are performed before updating θ.

Algorithm 3.13 (GAN training algorithm) Let $\theta \in \mathbb{R}^m$ and $\xi \in \mathbb{R}^n$ be initial parameters for G_θ and D_ξ with $m, n \in \mathbb{N}$ and let $K \in \mathbb{N}$ be the amount of training iterations of ξ per training iteration of θ. Let I be the batch size, and J the number of training iterations.

```
1: for $j = 1, \dots, J$ do
2:     for $k = 1, \dots, K$ do
3:         Draw random samples $\{z_i = Z(\omega_i) \mid \omega_i \in \Omega\}_{i=1}^I$ from the noise prior random variable.
4:         Draw samples from the data set $\{x_i \in \mathcal{X}\}_{i=1}^I$
5:         Update $\xi$ by ascending the gradient
               $D_\xi \frac{1}{I} \sum_{i=1}^{I} \left(\log(D_\xi(x_i)) + \log(1 - D_\xi(G_\theta(z_i)))\right).$
6:     end for
7:     Draw random samples $\{z_i = Z(\omega_i) \mid \omega_i \in \Omega\}_{i=1}^I$ from the noise prior random variable.
8:     Update $\theta$ by descending the gradient
               $D_\theta \frac{1}{I} \sum_{i=1}^{I} \log\left(1 - D(G_\theta(z_i))\right).$
9: end for
```

The gradient decent in lines 5 and 8 can be performed as described in Algorithm 1.24 the only difference being that the gradient used for the updates is not the squared loss but based on the GAN loss functional. The backpropagation rule can be recalculated and adapted accordingly.

It should be noted that this is not a convex optimisation problem, since the composition of the parametrisation functions (3.4) with the functions represented by the the actual networks G and D may not be convex. However, it turns out that the virtual training criterion is convex. This means that under the theoretical assumption that the discriminator between each update of θ is optimal and that the optimisation of the generator's parameters is ideal, the GAN actually converges to the data distribution as the virtual training criterion reaches its minimum.

Proposition 3.14 (Convexity of the virtual training criterion) The virtual training criterion

$$\begin{aligned} C:\, & C_c^0(\mathbb{R}^d) \longrightarrow \mathbb{R} \\ & \mu \longmapsto \int_{\mathbb{R}^d} \nu(x) \log\left(\frac{\nu(x)}{\nu(x)+\mu(x)}\right) + \mu(x)\log\left(\frac{\mu(x)}{\nu(x)+\mu(x)}\right) d\lambda^d(x). \end{aligned}$$

is strictly convex, where $\nu \in C_c^0(\mathbb{R}^d)$ is the unknown data distribution.

Proof. First we note that $C_c^0(\mathbb{R}^d)$ is a convex subset of the vector space $C^0(\mathbb{R}^d)$ as convex combinations of compactly supported functions are compactly supported. Now define

$$f:\, \mathbb{R}^2_{\geq 0} \longrightarrow \mathbb{R}, \quad (a,x) \longmapsto a\log\left(\frac{a}{a+x}\right) + x\log\left(\frac{x}{a+x}\right).$$

Since $f \in C^2(\mathbb{R}^2_{\geq 0})$ we can compute the derivatives

$$\begin{aligned} \partial_x f:\, \mathbb{R}_{\geq 0} \longrightarrow \mathbb{R}, \qquad & x \longmapsto \underbrace{a\frac{a+x}{a}\frac{-a}{(a+x)^2} + x\frac{a+x}{x}\frac{a+x-x}{(a+x)^2}}_{=0} + \log\left(\frac{x}{a+x}\right) \\ \partial_x^2 f:\, \mathbb{R}_{\geq 0} \longrightarrow \mathbb{R}, \qquad & x \longmapsto \frac{a+x}{x}\frac{a+x-x}{(a+x)^2} = \frac{a}{x(a+x)} \geq 0. \end{aligned}$$

This implies that $f(a,\bullet)$ is convex for every $a \geq 0$ and strictly convex if and only if $a \neq 0$ ([1], Chapter 9, Corollary 2.13). Note that for $(a,x) \in \mathbb{R}^2_{\geq 0}$, $f(a,x) \neq 0$ implies $a \neq 0$. Thus, f is strictly convex whenever it is non-zero. Now let $\mu_1, \mu_2 \in C_c^0(\mathbb{R}^d)$ be probability densities and $t \in [0,1]$. Then, by the definition of convexity we have

$$\begin{aligned} C(t\mu_1 + (1-t)\mu_2) &= \int_{\mathbb{R}^d} f(\nu(x), t\mu_1(x) + (1-t)\mu_2(x))\, d\lambda^d(x) \\ &= \int_{\{f(\nu(x),t\mu_1(x)+(1-t)\mu_2(x))\neq 0\}} f(\nu(x), t\mu_1(x) + (1-t)\mu_2(x))\, d\lambda^d(x) \\ (f \neq 0 \text{ strictly convex}) &< \int_{\{f(\nu(x),t\mu_1(x)+(1-t)\mu_2(x))\neq 0\}} tf(\nu(x),\mu_1(x)) + (1-t)f(\nu(x),\mu_2(x))\, d\lambda^d(x) \\ &\leq \int_{\mathbb{R}^d} tf(\nu(x),\mu_1(x)) + (1-t)f(\nu(x),\mu_2(x))\, d\lambda^d(x) \\ &= t\int_{\mathbb{R}^d} f(\nu(x),\mu_1(x))\, d\lambda^d(x) + (1-t)\int_{\mathbb{R}^d} f(\nu(x),\mu_2(x))\, d\lambda^d(x) \\ &= tC(\mu_1) + (1-t)C(\mu_2). \end{aligned}$$

■

3.3 Application for image processing

3.3.1 Supervised adversarial training

In the previous consideration of GANs, we have examined the general case where the generator G mimics the distribution of the data. For the case of image processing, however, it is desired that a certain processing is applied to input images so that an expected output image is produced. This means that we want to train GANs with data consisting of input-output pairs, as we described in Chapter 1.

We will see that this is feasible and that it corresponds quite closely to a supervised learning process, with the difference that no explicit loss function needs to be defined for the training, but this is provided automatically by the discriminator. Furthermore, since we are often dealing with non-deterministic hypotheses in image processing, such as ill-posed problems like the debayer problem, GANs allow the generator to output the data with some variance within an expected range.

The method proposed here is essentially based on the fact that the discriminator also sees the input image and thus not only single images, but the graph of a function or, in the stochastic scenario, the samples of an input-output relationship. This can also be seen as restricting the stochastic output of the generator to certain features. For this reason, this type is often called *conditional GAN (cGAN)*. It was proposed in [19] and applied in [15] for image-to-image tasks.

For a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ and $s, t \in \mathbb{N}$ the random variable of the data distribution X can then be rewritten as

$$X : \Omega \longrightarrow \mathbb{R}^s \times \mathbb{R}^t, \quad \omega \longmapsto (X_1(\omega), X_2(\omega)).$$

and the GAN loss functional for conditional GANs is redefined as

$$V : L^0((\Omega \times \mathbb{R}^s), \mathbb{R}^t) \times L^0((\mathbb{R}^s \times \mathbb{R}^t), (0,1)) \longrightarrow \bar{\mathbb{R}}$$
$$(G, D) \longmapsto \mathbb{E}[\log(D \circ X)] + \mathbb{E}[\log(1 - D(X_1, G(\bullet, X_1)))].$$

The findings on GANs obtained above can be applied to this setting by assuming that $\mathbb{P}_X$ and $\mathbb{P}_{(X_1, G(\bullet, X_1))}$ are absolute continuous with respect to the Lebesgue measure. This provides us with corresponding probability densities ν, μ and as in proposition 3.3 we can rewrite the GAN loss functional as

$$V(G, D) = \int_{\mathbb{R}^d} \nu(x) \log(D(x)) + \mu(x) \log(1 - D(x))\, d\lambda^d(x).$$

If we assume that $\mu, \nu \in C_c^0(\mathbb{R}^s \times \mathbb{R}^t)$ and set $K := \operatorname{supp}(\nu) \cup \operatorname{supp}(\mu)$ then, with the same computation as in proposition 3.5, we get that the optimal discriminator is given by

$$D^\star : K \longrightarrow (0,1), \quad x \longmapsto \frac{\nu(x)}{\nu(x) + \mu(x)}.$$

The definition of the virtual training criterion and the proof that it is convex is identical as above.

In the case that a deterministic output of the generator is desired, the generator can also be considered as a function $G(\omega, \bullet) : \mathbb{R}^s \longrightarrow \mathbb{R}^t$ for a fixed $\omega \in \Omega$.

3.3.2 Demosaicing with GANs

In this thesis, a GAN network was trained with the data described in Section 2.5.1. The GAN loss functional, discriminator and generator were used in the supervised setting as described in Section 3.3.1. Since there is no objective error criterion, the training success cannot be monitored as with a loss function during training. For this reason, the output of the discriminator was not chosen as a scalar, but as an image-like array, which was displayed during the training. In this way it was possible to determine whether the discriminator could identify critical areas. For the generator and the discriminator, the U-Net architecture as described in 2.4.1 was chosen.

Although the discriminator seemed to be able to identify faulty areas correctly to some extent, the training was extremely unstable. There was often a oscillation back and forth between the generator and the discriminator, so that the error pattern of the discriminator seemed to be inverted after a few iterations. [4] suggests using the *Wasserstein* metric to improve the stability of the training but this did not improve the result either.

Since in other publications such as [15] GANs have been successfully used for image-to-image translation tasks, it is likely that good results can be achieved by a suitable choice of hyperparameters, such as a different architecture for the discriminator. However, these experiments could not be carried out due to the limited time scope of this thesis and are therefore open for future work.

Epilogue

To pursue the initial research question of whether neural networks are applicable to the processing of camera RAW images, we investigated CNNs in the context of statistical learning and found that they are in principle well suited for problems such as demosaicing. In order to incorporate a priori knowledge into the hyperparameters and CNN architectures, we analysed the requirements for debayer algorithms and came, by considering the Nyquist-Shannon sampling theorem, to the realisation that the main challenge is to reconstruct high-frequency regions and avoid unwanted artefacts at the same time. Also, focusing on the Bayer pattern, which has an increased resolution in the green channel, gave the decisive hint that a CNN structure that reconstructs the green channel first and uses it then to compute the red and blue channels (as proposed in [9]) might be more effective. The results confirmed that CNNs can achieve at least equivalent results compared to classical algorithms. Finally, it was investigated whether the loss function can be dispensed using GANs, by replacing it with another neural network. We showed mathematically that this concept is sound. However, an implementation of GANs as demosaicing algorithms was not successful, as the training process was very unstable.

When implementing the CNN for debayering, it was observed that the distribution and diversity of the data has very central influence on the quality of the reconstructed image. We have discussed in Section 2.5.1 that only synthetic training data can be used. We created this data from a combination of Siemens stars and chirps. It is up to further investigation if the trained model performs better on real camera data if the training data is distributed in such a way that it more closely resembles signals from natural images. This research approach could be promising for future studies.

Another aspect that should be explored in more depth is how results can be assessed quantitatively. For this purpose, error metrics that reflect the human perception could be used. Only if it is possible to benchmark machine learning models it is possible to have genuine scientific debates.

Finally, it must not be forgotten that for applying image processing algorithms to motion picture cameras it must be assured that the algorithms are temporally stable. Otherwise, disturbing temporal artefacts may occur as the model might treat similar images quite differently. In future work, this could be achieved by methods to increase the regularity of the function represented by the neural network, or by training and evaluating the model in temporal space. This would mean that multiple temporally consecutive images have to be fed into the neural network at once.

Bibliography

[1] H. Amann and J. Escher. *Analysis I*. Grundstudium Mathematik. Birkhäuser Basel, 2006. ISBN: 9783764377557.

[2] H. Amann and J. Escher. *Analysis III*. Grundstudium Mathematik. Birkhäuser Basel, 2008. ISBN: 9783764388836.

[3] Stefano Andriani et al. “Beyond the Kodak image set: A new reference set of color image sequences.” In: *2013 IEEE International Conference on Image Processing, ICIP 2013 - Proceedings* (Sept. 2013), pp. 2289–2293. DOI: 10.1109/ICIP.2013.6738472.

[4] Martin Arjovsky, Soumith Chintala, and Léon Bottou. *Wasserstein GAN*. 2017. DOI: 10.48550/ARXIV.1701.07875. URL: https://arxiv.org/abs/1701.07875.

[5] Bryce E. Bayer. *Color imaging array*. U.S. Patent No. 3,971,065. 1975.

[6] Richard Beals. *Analysis: An Introduction*. Cambridge University Press, 2004. DOI: 10.1017/CBO9780511755163.

[7] H. (Haim) Brézis. *Functional Analysis, Sobolev Spaces and Partial Differential Equations*. eng. Universitext. New York ; Springer, 2011. ISBN: 9780387709147.

[8] Anthony L. Caterini and Dong Eui Chang. *Deep Neural Networks in a Mathematical Framework*. 1st. Springer Publishing Company, Incorporated, 2018. ISBN: 3319753037.

[9] Kai Cui et al. “Color Image Restoration Exploiting Inter-Channel Correlation With a 3-Stage CNN.” In: *IEEE Journal of Selected Topics in Signal Processing* 15.2 (2021), pp. 174–189. DOI: 10.1109/JSTSP.2020.3043148.

[10] Vincent Dumoulin and Francesco Visin. *A Guide to Convolution Arithmetic for Deep Learning*. 2016. DOI: 10.48550/ARXIV.1603.07285. URL: https://arxiv.org/abs/1603.07285.

[11] Rick Durrett. *Probability: Theory and Examples*. 5th ed. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 2019. DOI: 10.1017/9781108591034.

[12] Mark D Fairchild. *Color Appearance Models*. John Wiley & Sons, 2013.

[13] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].

[14] H von Helmholtz. “LXXXI. On the theory of compound colours.” In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 4.28 (1852), pp. 519–534.

[15] Phillip Isola et al. *Image-to-Image Translation with Conditional Adversarial Networks*. 2016. DOI: 10.48550/ARXIV.1611.07004. URL: https://arxiv.org/abs/1611.07004.

[16] Günter Klambauer et al. “Self-Normalizing Neural Networks.” In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS’17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 972–981. ISBN: 9781510860964.

[17] S. Kullback and R. A. Leibler. "On Information and Sufficiency." In: *The Annals of Mathematical Statistics* 22.1 (1951), pp. 79–86. DOI: 10.1214/aoms/1177729694.

[18] E.H. Lieb et al. *Analysis*. Crm Proceedings & Lecture Notes. American Mathematical Society, 2001. ISBN: 9780821827833.

[19] Mehdi Mirza and Simon Osindero. *Conditional Generative Adversarial Nets*. 2014. DOI: 10.48550/ARXIV.1411.1784. URL: https://arxiv.org/abs/1411.1784.

[20] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2012. ISBN: 026201825X.

[21] Kathy T Mullen. "The contrast sensitivity of human colour vision to red-green and blue-yellow chromatic gratings." In: *The Journal of physiology* 359.1 (1985), pp. 381–400.

[22] Yurii E. Nesterov. *Introductory Lectures on Convex Optimization - A Basic Course*. Vol. 87. Applied Optimization. Springer, 2004. ISBN: 978-1-4613-4691-3. DOI: 10.1007/978-1-4419-8853-9. URL: https://doi.org/10.1007/978-1-4419-8853-9.

[23] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. DOI: 10.48550/ARXIV.1505.04597. URL: https://arxiv.org/abs/1505.04597.

[24] Michael Schöberl et al. "Dimensioning of optical birefringent anti-alias filters for digital cameras." In: *2010 IEEE International Conference on Image Processing*. 2010, pp. 4305–4308. DOI: 10.1109/ICIP.2010.5651784.

[25] Wenzhe Shi et al. *Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network*. 2016. DOI: 10.48550/ARXIV.1609.05158. URL: https://arxiv.org/abs/1609.05158.

[26] Andrew Stockman and Lindsay T Sharpe. "The spectral sensitivities of the middle-and long-wavelength-sensitive cones derived from measurements in observers of known genotype." In: *Vision research* 40.13 (2000), pp. 1711–1737.

[27] Yang Wang. *A Mathematical Introduction to Generative Adversarial Nets (GAN)*. 2020. arXiv: 2009.00169 [cs.LG].

[28] Thomas Young. "On the nature of light and colours." In: *A Course of Lectures on Natural Philosophy and the Mechanical Arts (J. Johnson, 1807)* 1 (1802), pp. 464–465.

[29] A. Zygmund and Robert Fefferman. *Trigonometric Series*. 3rd ed. Cambridge Mathematical Library. Cambridge University Press, 2003. DOI: 10.1017/CBO9781316036587.

Declaration

I declare that this thesis is my own unaided work. All sources used for its development are acknowledged as references.

Munich, June 27, 2022

. . . .
Thomas Eingartner