University of Siegen
Department of Electrical Engineering and Computer Science
Computer Vision Group

Master's Thesis

Training of Lipschitz Continuous Deep Neural Networks

Hendrik Sommerhoff

Matriculation Number: 1020300

May 4, 2019

Advisors:
Prof. Dr. Michael Möller
M. Sc. Jonas Geiping

Abstract

Recent research works have proposed to use generic image denoising neural networks as plug-and-play regularizers for solving inverse image reconstruction problems. Even though this achieves state of the art results for many tasks, it is hard to prove the convergence of such algorithms. This thesis develops a provably convergent algorithmic scheme that uses a non-expansive denoising neural network as a regularizer that is trained by enforcing Lipschitz continuity with constant 1. To achieve this, different methods from recent related work for calculating an upper bound for the best Lipschitz constant of a neural network are applied to DnCNN, a common network architecture for denoising, and to a Fourier transform based architecture which makes these calculations easier and more efficient. Using these upper bounds, the network can be normalized during training time, resulting in a provably non-expansive neural network. Furthermore, a inherently non-expansive network based on wavelet denoising, which does not need any further normalization, is developed. Even though the denoising performance of the normalized networks is lower than their unnormalized counterparts, using them as plug-and-play regularizers for solving a gaussian deblurring reconstruction problem shows faster convergence speed and higher stability for some choices of hyperparameters.

Contents

List of Figures

List of Tables

1 Introduction

In recent years, Deep Learning has found its way into all areas of digital image processing and computer vision. Better training algorithms, easier access to larger amounts of training data and advances in GPU hardware have made it easier to train deeper and deeper neural networks, even on consumer-grade computers. Despite this rapid progress, many aspects, like the training process and several regularization techniques, are not yet well understood on a theoretical level and rely on heuristics instead.

This thesis explores an inherent mathematical property of neural networks (and other functions) called Lipschitz continuity. A network $\mathcal{N}$ is called Lipschitz continuous with constant L if

$$||\mathcal{N}(x_1) - \mathcal{N}(x_2)|| \leq L||x_1 - x_2||$$

for any input x_1 and x_2. In particular, we investigate the training of neural networks with the Lipschitz-constant $L = 1$, so called non-expansive networks. This topic was motivated by a recent work of Meinhardt et al. [1], which proposes to utilize a single neural network trained for image denoising as a plug-and-play regularizer to solve arbitrary linear inverse problems with variational methods. Linear inverse problems model phenomena, e.g. Gaussian blur, as the result of applying a linear operator A to a clean image u. Noisy measurements of a blurred image f make it impossible to extract the clean image u with a simple inversion of A again, even if A is known. For this reason, variational methods consider minimization problems of the form

$$\hat{u} = \underset{u}{\operatorname{argmin}} \, ||Au - f||_2^2 + R(u)$$

where R is a regularization term. A minimizer for this term can be found with an algorithm called proximal gradient descent. Meinhardt et al. [1] propose to replace a step during this algorithm with the application of DnCNN [2], a powerful denoising neural network. Even though the results are heuristically on par with state-of-the-art

methods, a convergence of this new algorithm is not yet proved because the underlying primal-dual algorithm complicates the convergence analysis. In this thesis an easier algorithm is developed for which the non-expansiveness of the neural network and the existence of a fixed point prove the convergence. It is shown that non-expansiveness is in general not a property of DnCNN, explore different normalization schemes and develop special architectures to enforce non-expansiveness in denoising networks. The use of such networks for image reconstruction problems is experimentally tested by the example of gaussian deblurring.

Apart from this particular use case, the Lipschitz continuity of deep neural network has been a prominent research topic in recent years. Yoshida et al. [3] propose a regularization term in the loss function which penalizes large singular values of linear layers. Even though this leads to a smaller Lipschitz constant, there is no proof as to which constant this will be. Gouk et al. [4] compute an upper bound for the Lipschitz constant for an arbitrary p-norm of a network and use it to normalize the weight matrices of each layer after a training step in order to provably enforce a user definable Lipschitz constant. Only experiments for classification networks were given by authors, and showed in general an increase in performance if used together with different regularization methods and the chosen Lipschitz constant was large enough. A slightly different method called singular value clipping was described by Sedghi et al. [5] in order to compute and restrict the Lipschitz constant of convolutional layers with respect to the euclidean norm. However, it turned out that this approach only approximates the real Lipschitz constant from below and needs larger amounts of computation for a closer result. Because of this, their result did not satisfy the demand of a method that can be used to normalize a network after each training step. A related field of research investigates the influence of the Lipschitz constant of a neural network on the security against adversarial examples [6] and the training of Generative Adversarial Networks [7]. Intuitively a small Lipschitz constant means that a small change in input only leads to a small change in the output of a network, which produces to better protection against adversarial attacks.

2 Mathematical Foundations

This chapter lists some important mathematical concepts in the fields of Linear Algebra and Analysis and introduces notation that will be used throughout this thesis [1]. Most functions in the following chapters will be defined as mappings between normed vector spaces. If the specific norm is not important it will be denoted with $||\cdot||$. Otherwise, the notation will be as follows:

Definition 1. *Let $x = (x_1, \cdots, x_n)^T \in \mathbb{K}^n$, for $\mathbb{K} \in \{\mathbb{R}, \mathbb{C}\}$ and $p \in [1, \infty]$. The p-norm (or ℓ_p-norm) is defined as:*

$$||x||_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}}, p < \infty \tag{2.1}$$

$$||x||_\infty = \max_i |x_i| \tag{2.2}$$

Let $T : \mathbb{K}^n \to \mathbb{K}^m$ be a linear Operator. The operator norm induced by $||\cdot||_p$ as a norm on $\mathbb{K}^n$ and $||\cdot||_q$ as a norm on $\mathbb{K}^m$ is defined as:

$$||T||_{p,q} = \sup_{||x||_p = 1} ||Tx||_q \tag{2.3}$$

For $p = q$ this will be shortened to $||T||_p$. In particular, $||T||_2$ is called the spectral norm. For a matrix $A \in \mathbb{K}^{n \times m}$ the Frobenius norm is defined as:

$$||A||_F = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{m} |A_{i,j}|^2} = \sqrt{\mathrm{tr}(A^H A)} \tag{2.4}$$

Lemma 1. *Let $A \in \mathbb{K}^{n \times m}$. $||A||_2 = \sigma_{max}$ where σ_{max} is the largest singular value of A*

[1] The notation in this thesis closely follows the lectures *Convex Optimization for Computer Vision* and *Deep Learning* by Michael Möller at the University of Siegen

(the square root of the largest eigenvalue of $A^T A$). Furthermore:

$$||A||_1 = \max_j \sum_{i=1}^{n} |A_{i,j}| \tag{2.5}$$

$$||A||_\infty = \max_i \sum_{j=1}^{m} |A_{i,j}| \tag{2.6}$$

Proof. See page 57 of [8]. □

The next definition of Lipschitz continuity is central to this thesis:

Definition 2. *Let X and Y be normed vector spaces and $f : X \to Y$ a function. f is called* Lipschitz continuous with constant L *if for any $x_1, x_2 \in X$ it holds that*

$$||f(x_1) - f(x_2)|| \leq L||x_1 - x_2|| \tag{2.7}$$

If f is Lipschitz continuous with constant L, it is also Lipschitz continuous for any $\hat{L} > L$. The smallest possible Lipschitz constant of f is denoted with $\overline{L}(f)$. The following lemma makes it easy to compute the Lipschitz constant of a function composition:

Lemma 2. *Let X_1, X_2 and X_3 be normed spaces, $f_1 : X_1 \to X_2$ and $f_2 : X_2 \to X_3$ Lipschitz continuous functions with constants L_1 and L_2 respectively. Then the $f_2 \circ f_1$ is Lipschitz continuous with constant $L_1 L_2$.*

Proof. Let $x_1, x_2 \in X_1$. Then

$$||f_2(f_1(x_1)) - f_2(f_1(x_2))|| \leq L_2||f_1(x_1) - f_1(x_2)|| \leq L_1 L_2||x_1 - x_2||$$

□

In general $\overline{L}(f \circ g) \leq \overline{L}(f)\overline{L}(g)$. For example consider $f, g : \mathbb{R} \to \mathbb{R}$ with $f(x) = \max(x, 0)$ and $g(x) = \min(x, 0)$. Both functions have Lipschitz constants $\overline{L}(f) = \overline{L}(g) = 1$, but their composition $f \circ g$ is the constant zero function and thus has Lipschitz constant 0. Remarkably, the smallest Lipschitz constant also depends on the domain of f since $\overline{L}(f|_{\mathbb{R}^-}) = 0$ in the above example. The sum of two Lipschitz continuous functions is also Lipschitz continuous [4]:

Lemma 3. *Let X_1, X_2 be normed spaces, $f_1, f_2 : X_1 \to X_2$ Lipschitz continuous functions with constants L_1 and L_2 respectively. Then $f_1 + f_2$ is Lipschitz continuous with constant $L_1 + L_2$.*

Proof. Let $x_1, x_2 \in X_1$. Then

$$\begin{aligned}
&||(f_1(x_1) + f_2(x_1)) - (f_1(x_2) + f_2(x_2))|| \\
=&||(f_1(x_1) - f_1(x_2)) + (f_2(x_1) - f_2(x_2))|| \\
\leq&||(f_1(x_1) - f_1(x_2))|| + ||(f_2(x_1) - f_2(x_2))|| \\
\leq&L_1||x_1 - x_2|| + L_2||x_1 - x_2|| \\
=&(L_1 + L_2)||x_1 - x_2||
\end{aligned}$$

□

Trivially, if f has Lipschitz constant L the function $g = \lambda f$ with $\lambda \in \mathbb{R}$ has Lipschitz constant $|\lambda|L$. In order to compute a Lipschitz constant one would have to check (2.7) for every possible x_1 and x_2. If the function is differentiable there is a slightly easier way.

Lemma 4. *Let $f : \mathbb{R}^n \to \mathbb{R}^m$ be a differentiable function and $||J_f(x)||_p < \infty$ for any x. Then f is Lipschitz continuous with respect to the p-norm with constant*

$$L = \max_{x \in \mathbb{R}^n} ||J_f(x)||_p \tag{2.8}$$

where J_f is the Jacobian matrix of f.

Proof. According to the mean value theorem for any $x_1, x_2 \in \mathbb{R}^n$ there is a point $\hat{x}$ on the line between x_1 and x_2 such that

$$f(x_2) - f(x_1) = J_f(\hat{x}) \cdot (x_2 - x_1)$$

Taking the norm on both sides leads to

$$\begin{aligned}
||f(x_2) - f(x_1)||_p =&||J_f(\hat{x}) \cdot (x_2 - x_1)||_p \\
\leq&||J_f(\hat{x})||_p||x_2 - x_1||_p
\end{aligned}$$

where the last inequality is because the matrix p-norm is consistent with the vector p-norm. Taking the maximum matrix norm over all $x \in \mathbb{R}^n$ yields the proof. □

The convergence properties of some algorithms depend on the Lipschitz constant of the gradient of a function:

Definition 3. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be continuously differentiable. f is called* L-smooth *if ∇f is Lipschitz continuous with constant L, i.e.*

$$||\nabla f(x_1) - \nabla f(x_2)|| \leq L||x_1 - x_2|| \tag{2.9}$$

for any $x_1, x_2 \in \mathbb{R}^n$.

In the following the concept of contractions and non-expansive functions will often be referenced. An important property of contractions is that, according to the Banach fixed-point theorem, they always have a fixed point.

Definition 4. *If a function $f : X \to X$ is Lipschitz continuous with constant $L < 1$ then f is called a* contraction. *If $L \leq 1$ then f is called* non-expansive. *This means that any contraction is also non-expansive.*

Theorem 1. *(Banach fixed-point theorem)*
Let M be a closed subset of a normed vector space X and let $f : M \to M$ be a contraction. Then there exists a unique fixed point $\hat{x} \in M$ with $f(\hat{x}) = \hat{x}$. The sequence $x_{n+1} = f(x_n)$ converges to $\hat{x}$ for any starting point $x_0 \in M$.

Proof. See [9] □

3 Image Reconstruction Problems

In image processing, often an observed image f is actually the result of a linear transformation of a clean image. For example, Gaussian blur can be modeled as the convolution $k * u$ of a known Gaussian kernel k and an image u which in turn can be written as Au where A is the matrix representation of this convolution. Unfortunately, it is not as simple as solving the linear equation $Au = f$ to deblur f, because the measured image is subject to noise. Small measurement errors ϵ lead to large changes of $\tilde{u} = A^{-1}(f + \epsilon)$. Problems of this kind are called ill-posed:

Definition 5. *Problems whose solution does not exist, is not unique or do not depend continuously on the initial conditions are referred to as* ill-posed*.*

Typical examples for ill-posed inverse problems in image processing are denoising, deblurring, super-resolution and inpainting. There are different approaches for finding a satisfying solution to an ill-posed problem, if a solution even exists at all. In the following section three commonly used methods will be explained. The first section deals with solving the problem with energy minimization approaches that circumvent the ill-posedness by introducing a regularization function. The second class of methods tries to learn a function that solves the problem directly for a given input image. Today this is commonly done with neural networks. The last approach is a hybrid of energy minimization and learning based methods.

3.1 Energy Minimization Methods

In order to find a satisfying solution to an ill-posed problem, prior information about the distribution of real images has to be incorporated in the form of regularization functions.

Variational methods consider an energy minimizing approach where

$$\hat{u} = \underset{u}{\operatorname{argmin}}\, E(u) = \underset{u}{\operatorname{argmin}}\, H_f(Au) + R(u) \tag{3.1}$$

The data fidelity term $H_f(Au)$ punishes large deviations of the measured image f and the transformed image Au. For example, the squared error $H_f(Au) = ||Au - f||_2^2$ could be used. To find a good regularization function R is a lot harder, because it should be large for images that intuitively do not *look like* real images and small otherwise, which is a statement that is hard to grasp mathematically. Often the TV-L1 regularizer [10] $R(u) = ||\nabla u||_1$, which punishes oscillations of u, is used because natural images tend to have relatively large areas with similar color and edges between objects are generally sharp. The following sections will explain algorithms for finding a minimizer for general functions like E and give criteria under which the algorithms converge.

Definition 6. *A subset C of a vector space V is called* convex *if for all $x, y \in C$ and all $\lambda \in [0,1]$ it holds that $\lambda x + (1-\lambda)y \in C$, i.e. the line that connects two points in C completely lies in C.*
A function $E : C \to \mathbb{R}$ is called convex if for all $x, y \in C$ and all $\lambda \in [0,1]$ it holds that $E(\lambda x + (1-\lambda)y) \leq \lambda E(x) + (1-\lambda)E(y)$

Convex functions have the nice property that every local optimum is also a global optimum. Furthermore, if E is convex and differentiable, the necessary condition for an optimum $\nabla E(x) = 0$ is also sufficient, meaning that there is a global optimum at x. Since it is almost impossible to compute the roots of ∇E directly in higher dimensions, iterative methods like gradient descent are most commonly used to find a minimizer. The idea behind gradient descent is to update the current iterate u^k in a direction in which the objective function decreases. Because the gradient of a function points into the direction of the steepest increase, a step into the opposite direction yields the fastest decrease of the function value. The algorithm is a fixed point iteration of the form

$$u^{k+1} = u^k - \tau \nabla E(u^k) \tag{3.2}$$

The parameter τ is called *step size* (or in context of neural networks also *learning rate*) and influences the convergence rate of the algorithm. Unfortunately the function $G(u) = u - \tau\nabla(E(u))$ is only a contraction in specific cases, so the Banach fixed-point

theorem can not be applied for a convergence analysis. However there is a slightly stronger version of non-expansiveness which is a sufficient condition for the convergence of a fixed point iteration if a fixed point exists [11, 12]:

Definition 7. *A operator G is called averaged if there exists a non-expansive operator H such that*

$$G = \alpha \operatorname{Id} + (1-\alpha)H \tag{3.3}$$

for a $\alpha \in (0,1)$

Compositions and convex combinations of averaged functions are averaged again. It is easy to see that averaged functions are non-expansive, because the Lipschitz constant of $\alpha \operatorname{Id} + (1-\alpha)H$ is $\alpha + (1-\alpha) = 1$. Later it will be important that the convex combination of an averaged operator and a contraction is averaged which will now be proved with the help of Lemma 3.

Lemma 5. *Let T_1 be averaged and T_2 be non-expansive. Let $\alpha \in (0,1)$. Then*

$$G = \alpha T_1 + (1-\alpha)T_2 \tag{3.4}$$

is averaged.

Proof. T_1 is averaged, so there exists a non-expansive operator $\tilde{T}$ and $\beta \in (0,1)$ such that

$$T_1 = \beta \operatorname{Id} + (1-\beta)\tilde{T}$$

With this, G can be written as

$$\begin{aligned} G =& \alpha T_1 + (1-\alpha)T_2 \\ =& \alpha(\beta \operatorname{Id} + (1-\beta)\tilde{T}) + (1-\alpha)T_2 \\ =& \alpha\beta \operatorname{Id} + \alpha(1-\beta)\tilde{T} + (1-\alpha)T_2 \\ =& \alpha\beta \operatorname{Id} + (1-\alpha\beta)\left(\frac{\alpha(1-\beta)}{1-\alpha\beta}\tilde{T} + \frac{1-\alpha}{1-\alpha\beta}T_2\right) \end{aligned}$$

Since $\gamma := \alpha\beta \in (0,1)$, all that remains is to show that the operator

$$H := \frac{\alpha(1-\beta)}{1-\alpha\beta}\tilde{T} + \frac{1-\alpha}{1-\alpha\beta}T_2$$

is non-expansive. Both $\tilde{T}$ and T_2 are non-expansive, so according to Lemma 3 a upper bound for the best Lipschitz constant of H can be given by the sum of the two factors in front of $\tilde{T}$ and T_2:

$$\begin{aligned} &\frac{\alpha(1-\beta)}{1-\alpha\beta} + \frac{1-\alpha}{1-\alpha\beta} \\ =&\frac{1-\alpha\beta}{1-\alpha\beta} \\ =&1 \end{aligned}$$

So H is non-expansive and G is of the form

$$G = \gamma\,\mathrm{Id} + (1-\gamma)H$$

□

With the notion of averaged operators, a sufficient condition for the convergence of gradient descent can be given:

Theorem 2. *If E is convex and L-smooth and $\tau \in (0, \frac{2}{L})$ then $G(u) = u - \tau\nabla E(u)$ is averaged, i.e. if E has a global minimum the gradient descent algorithm converges to that minimizer.*

Proof. See page 16 of [13] □

Coming back to the energy function $E(u) = H_f(Au) + R(u)$, one problem still remains. Even though the data fidelity term is usually differentiable and convex, e.g. as it is the case for $H_f(Au) = ||Au - f||_2^2$, this is not always true for common regularizers. For example, even though $R(u) = ||u||_1$ is convex, it is not differentiable at $u = 0$. A possible approach is to use proximal gradient methods which incorporate the proximity operator [14]

$$\mathrm{prox}_R(v) = \underset{u}{\mathrm{argmin}}\, R(u) + \frac{1}{2}||v-u||_2^2 \tag{3.5}$$

With this definition the so called forward-backward splitting algorithm can be used to solve minimization problems of the form $E(u) = F(u) + G(u)$ where F and G are convex and only F is L-smooth. Since the proximity operator is always averaged [15], for a step

size $\tau \in (0, \frac{2}{L})$ the algorithm converges to a minimizer of E for any starting point u_0:

$$u^{k+1} = \operatorname{prox}_{\tau G}(u^k - \tau \nabla F(u^k)) \tag{3.6}$$

This is a gradient descent step on F followed by a proximity step on G. For the specific energy function $E(u) = H_f(Au) + R(u)$ and an application of the chain rule this leads to

$$u^{k+1} = \operatorname{prox}_{\tau R}(u^k - \tau A^T \nabla H_f(Au^k)) \tag{3.7}$$

The experiments in this thesis, however, will be limited to the update rule

$$u^{k+1} = (1-\alpha)(u^k - \tau A^T \nabla H_f(Au^k)) + \alpha(u^k - \tau \nabla R_\epsilon(u)) \tag{3.8}$$

where R_ϵ is the Huber-loss [16] which smoothes the 1-norm around 0 in order to make it differentiable.

$$R_\epsilon(x_i) = \sum_i h_\epsilon(x_i) \tag{3.9}$$

with

$$h_\epsilon(z) = \begin{cases} \frac{1}{2}z^2 & \text{if } |z| \leq \epsilon, \\ \epsilon(|z| - \frac{1}{2}\epsilon), & \text{otherwise.} \end{cases} \tag{3.10}$$

Since equation 3.8 is the convex combination of gradient descent steps on H_f and R_ϵ, which are averaged operators, it converges to a minimizer of E.

3.2 Learning-based Approaches

Learning-based approaches aim to find parameters for a function $\mathcal{N}_\theta$ which directly maps an image u to a plausible solution $\mathcal{N}_\theta(u)$ of the inverse problem. The parameter vector θ is found by a training algorithm which requires often large amounts of training data, i.e. inputs x_i for which the desired output y_i is already known. Recent advances in the field of deep learning and the availability of large amounts of freely accessible training data sets make neural networks the most common choice for $\mathcal{N}_\theta$. McCann et al. [17] give an overview of typical network architectures and training algorithms, in particular for inverse imaging problems. Deep neural networks have been successfully applied to image deblurring [18], denoising [2, 19], inpainting [20–22], colorization [23] and many other tasks while achieving state-of-the-art results.

One advantage that neural networks have over variational methods is that they do not require as much knowledge of the underlying problem to work well. Often the same network architecture can be used for many different tasks. However, the training data has to fit the specific problem. This also means that each trained neural network in general only works for that precise problem. Moreover, the performance of a network is sensitive to the distribution of the data it was trained on. For example, a network that was trained to detect dogs in an image while using training data that did not contain any other kind of animal could incorrectly detect a cat as a dog during inference. The performance also depends on the choice of the training algorithm, the concrete architecture and various regularization techniques.

3.3 Hybrid Methods

Coming back to the energy minimization method for solving linear inverse problems in image processing from Chapter 3.1, in the following sections it will be explained how the hard to find regularization term of the energy function can be replaced by a neural network. Chang et al. [24] propose to learn the proximal operator as a projection onto the feasible set of images by training a neural network. Heide et al. [25] proved that any gaussian denoising algorithm can be expressed as a proximity operator. Since this is a central foundation for this thesis, a short proof for this statement will be given now, as presented as in [25]. Given an image $u \in \mathbb{R}^n$, the likelihood to observe an image f corrupted by gaussian noise with standard deviation σ is given by

$$p(f|u) = \frac{1}{\sqrt{(2\pi\sigma)^n}} \exp\left(\frac{-||f-u||_2^2}{2\sigma}\right) \tag{3.11}$$

In order to find u, given f, Bayes' rule is applied:

$$p(u|f) = \frac{p(f|u)p(u)}{p(f)} \tag{3.12}$$

The maximum-a-posteriori estimate of u is given by the maximum of this term, which is independent of the denominator

$$\begin{aligned} u_{MAP} &= \operatorname*{argmax}_u p(u)p(f|u) \\ &= \operatorname*{argmax}_u \log(p(u)p(f|u)) \\ &= \operatorname*{argmax}_u \log(p(u)) + \log(p(f|u)) \\ &= \operatorname*{argmin}_u -\log(p(u)) + \frac{1}{2\sigma}||f-u||_2^2 \\ &= \operatorname{prox}_{\sigma R}(f) \end{aligned}$$

This result motivates the replacement of the proximity operator in the update rule of forward-backward splitting (equation 3.7) by a generic gaussian denoising algorithm $\mathcal{G}$.

$$u^{k+1} = \mathcal{G}(u^k - \tau A^T \nabla H_f(Au^k)) \tag{3.13}$$

Heide et al. [25] and Venkatakrishnan et al. [26] have successfully used powerful denoising algorithms like Non-local means [27] and BM3D [28] to solve various linear inverse problems. Meinhardt et al. [1] considered using a denoising neural network like DnCNN [2] and achieved state of the art results. An overview of more algorithmic schemes and how to replace their regularizers with neural networks can be found in Table 1 of [29]. The advantage over pure learning based methods is that only a single network is required for all tasks. All that has to be changed is the operator A in the data term, which does not require a relearning of the network. Because the proximal operator is equivalent to an implicit gradient descent step, i.e.

$$\operatorname{prox}_{\tau R}(u^k) = u^k - \tau \nabla R(u^{k+1}) \tag{3.14}$$

the right summand of equation 3.8 can be replaced by a denoising algorithm as well, which leads to the update rule

$$u^{k+1} = (1-\alpha)(u^k - \tau A^T \nabla H_f(Au^k)) + \alpha \mathcal{G}(u^k) \tag{3.15}$$

The convergence of this method depends on $\mathcal{G}$ being at least non-expansive, because then, according to Lemma 5, the update step is an averaged operator, which means that the fixed point iteration converges if a fixed point exists. In general, neural networks

are not non-expansive, which motivates the explicit enforcement of Lipschitz constant 1 during the training. An overview of different convergence criteria that also include other algorithmic schemes can be found in [29]. Sreehari et al. [30] give restrictive criteria under which a neural network represents a proximal operator. Chan et al. [31] propose a provably convergent plug-and-play algorithm which converges for bounded denoisers, however not necessarily to a fixed point.

4 Training of Lipschitz Continuous Networks

This chapter starts with a more in-depth introduction to neural networks. The foundations layed down in that section will be used extensively throughout the rest of this thesis. After that, a short introduction to the state-of-the-art denoising network DnCNN is given and it is shown that a trained DnCNN is in general not non-expansive by giving examples for input images which violate the Lipschitz criterion. Motivated by that, related research on enforcing the Lipschitz continuity of neural networks is presented. It is shown, how to compute the Lipschitz constant of a neural network and how to use this constant in order to make the network non-expansive with respect to either the ℓ_1 or ℓ_2 norm. In the last section a special network architecture which is inherently non-expansive, and thus does not require any Lipschitz normalization during training, is described.

4.1 Recalling Neural Networks

At their core, neural networks are a framework for function approximation. Supposing a set of training features x_i and their ground truth labels y_i that are connected by an unknown function G with $G(x_i) = y_i$, the objective is to find a parameterized function $\mathcal{N}$ and a set of parameters θ such that $\mathcal{N}_\theta$ fits G well, i.e.

$$\mathcal{N}_\theta(x_i) \approx G(x_i) = y_i \tag{4.1}$$

It is important that the model is not only a good approximation on the training set but also generalizes to previously unseen examples.

One very simple neural network is linear regression which models the relationship between x_i and y_i as an affine linear function.

$$\mathcal{N}_\theta(x) = Wx + b \tag{4.2}$$

where $W \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ and θ consists of the vectorized entries of both W and b. Even though the expressiveness of linear models is limited to use cases in which the output depends linearly on the input, they can still be used as building blocks for deeper neural network architectures. Usually networks for more complicated problems comprise deeply nested functions. Note that simply nesting linear functions would result in a linear function again. For this reason most architectures alternate between linear and nonlinear functions.

$$\mathcal{N}_\theta(x) = (\phi_l \circ \sigma_l \circ \phi_{l-1} \circ \sigma_{l-1} \circ \cdots \circ \phi_1 \circ \sigma_1)(x) \tag{4.3}$$

Here the ϕ_i denote (affine) linear, the σ_i nonlinear functions (also called activation functions) and l is the number of layers. For example, the ϕ_i could be affine linear functions as above. Often the same activation function is used across the whole network, so, for $l = 2$, that would result in:

$$\mathcal{N}_\theta(x) = \sigma(W_2\sigma(W_1x + b_1) + b_2) \tag{4.4}$$

Once again (and for the rest of this thesis) θ denotes the collection of all network parameters, which in this case are the entries of W_i and b_i. A layer $Wx + b$ is called a fully-connected layer.

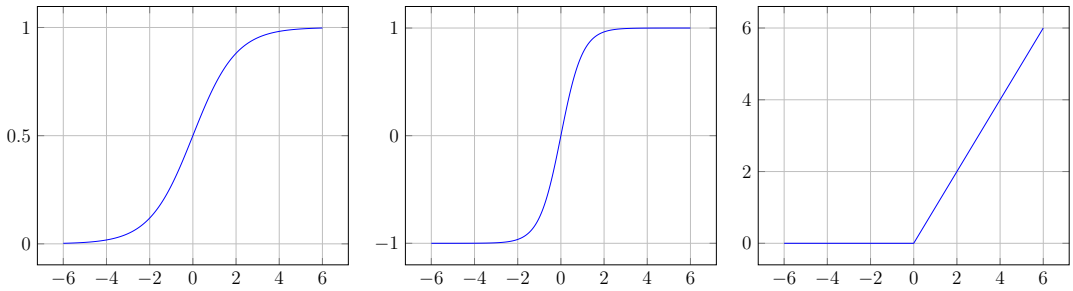
Some examples for pointwise activation functions are sigmoid, tanh and ReLU:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \tag{4.5}$$

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}, \tag{4.6}$$

$$\text{ReLU}(x) = \max(0, x) \tag{4.7}$$

Graphs for these functions can be seen in Figure 4.1. Because of its simple to compute and non vanishing gradient for large inputs, ReLU and variants are generally the first choice in modern networks [32].



(a) Sigmoid (b) tanh (c) ReLU

Figure 4.1: Different activation function

In order to find θ, a loss function which penalizes large deviation of the network's output and the ground truth labels y_i is defined. A popular loss function for regression is the mean squared error:

$$\mathcal{L}(x, y) = ||x - y||_2^2 \tag{4.8}$$

These losses are then summed over all training examples and define the error of the network given the current parameter θ.

$$E(\theta) = \sum_{i}^{n} \mathcal{L}(\mathcal{N}_\theta(x_i), y_i) \tag{4.9}$$

The objective is now to find $\hat{\theta} = \operatorname{argmin}_\theta E(\theta)$. This is usually done with one of the many variants of gradient descent, as defined in Section 3.1, with the following update rule:

$$\theta^{i+1} = \theta^i - \tau \nabla E(\theta^i) \tag{4.10}$$

Since E is usually a non-convex function, it is not guaranteed that this iteration converges to a global minimizer. Variants like gradient descent with momentum [33] or a very popular algorithm called Adam [34] are often used to avoid being stuck at a local minimum and to increase convergence speed. The gradient is computed by an algorithm for computing the chain rule called backpropagation.

If the number of training examples is very large it is impractical to compute the loss of all examples for every gradient descent step. The idea of stochastic gradient descent is to divide the training set into (random) non-overlapping subsets, called mini-batches, and to do a gradient descent step on any of these sets sequentially. Once a step on every mini-batch has been made, they are recomputed and the optimization is repeated. One iteration over all mini-batches is called an epoch. The gradient on these sets is an

approximation of the real gradient but becomes worse the closer the current iterate is to a minimizer. Despite this problem stochastic gradient descent is the main approach for training deep neural networks and has been shown to even reduce overfitting and thus improve generalization.

As stated above a neural network is a composition of linear and nonlinear functions, called layers. Besides fully connected layers there are many other types, two of which will be explained in more detail in the following sections. These two layers are the building blocks for the denoising network DnCNN [2] which will be used later on.

Convolutional Layers

Many tasks in computer vision like image classification and image denoising can be solved by neural networks. However, this means that the network has to take an image as its input. Supposing that images are represented as a one-dimensional vector $u \in \mathbb{R}^{c \cdot h \cdot w}$ of pixel values, where c is the number of channels and w and h are the width and height, the number of weights required for a fully connected layer which takes u as an input grows very fast. For example, for a relatively small RGB-image of size 80×80, a square fully connected layer without bias would have over 368 million parameters. This does not only slow down inference and training time, but also increases the memory requirements that are necessary for computing the gradients of the weights with the backpropagation algorithm dramatically.

Many problems in image processing are translation invariant, for example a network that should recognize an object in an image should still be able to recognize it even after it has been moved. This can be achieved by making the output of a layer at a specific pixel only dependent on the local neighborhood around that pixel. Mathematically this can be expressed as a convolution operation which, in a continuous setting, is defined as:

$$(k * f)(x) = \int_{\mathbb{R}^n} k(\tau) f(x - \tau) d\tau \tag{4.11}$$

The value of $(k * f)(x)$ can be interpreted as the weighted sum around $f(x)$ where the weights are given by the kernel k. For convolutional layers [35] this formula is discretized and it is assumed that k and f have finite support[1]. Thus the convolution of the kernel

[1] This thesis will only ever use zero padding

$(2s+1) \times (2t+1)$-dimensional filter kernel k and the image f is defined as

$$(k * f)[x, y] = \sum_{i=-s}^{s} \sum_{j=-t}^{t} k[i, j] f[x - i, y - j] \tag{4.12}$$

Pixels outside of the dimensions of the image are assumed to be zero. Note how this definition only applies to a single channel of an image. In convolutional layers the output of the convolution of each channel is summed up. This can be expressed as the convolution with a $n_c \times (2s+1) \times (2t+1)$-dimensional kernel where the filter does not "move" in the channel direction.

$$(k * f)[x, y] = \sum_{c=1}^{n_c} \sum_{i=-s}^{s} \sum_{j=-t}^{t} k[c, i, j] f[c, x - i, y - j] \tag{4.13}$$

This maps an image $f \in \mathbb{R}^{c \times h \times w}$ to an output $k * f \in \mathbb{R}^{h \times w}$. To generate multiple output channels usually multiple filters k_i are employed. Instead of using the above formula, it is more convenient to interpret the j-th output channel as the sum of n_c separate convolutions of each of the n_c input channels and the filters $k_{i,j}$, which in the following will denote the filter that maps the i-th input channel to the j-th output channel. If u_i represents the i-th input channel the complete formula for a convolutional layer can be written as

$$(\phi^{conv}(u))_j = \sum_{i=1}^{n_c} (k_{i,j} * u_i) \tag{4.14}$$

Like fully connected layers, convolutional layers usually have a learnable bias, which, however, will be ignored in this thesis because none of the used networks will use them and they do not influence the Lipschitz continuity. Since convolutional layers without bias are linear, they can be written as the matrix multiplication of a suitable matrix A and the (vectorized) input u.

Example 1. *Consider the 1-dimensional discrete convolution*

$$(k * f)(x) = \sum_{i=-s}^{s} k[s] f(x - s)$$

with

$$f = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix}$$

and

$$k = \begin{bmatrix} k_1 \\ k_2 \\ k_3 \end{bmatrix}$$

With zero padding the result of the convolution is

$$(k * f) = \begin{bmatrix} k_3 f_1 \\ k_2 f_1 + k_3 f_2 \\ k_1 f_1 + k_2 f_2 + k_3 f_3 \\ k_1 f_2 + k_2 f_3 + k_3 f_4 \\ k_1 f_3 + k_2 f_4 \\ k_1 f_3 \end{bmatrix}$$

Multiplication of the matrix A and the zero padded vector u yields the same result:

$$Au = \begin{bmatrix} k_2 & k_3 & 0 & 0 & 0 & k_1 \\ k_1 & k_2 & k_3 & 0 & 0 & 0 \\ 0 & k_1 & k_2 & k_3 & 0 & 0 \\ 0 & 0 & k_1 & k_2 & k_3 & 0 \\ 0 & 0 & 0 & k_1 & k_2 & k_3 \\ k_3 & 0 & 0 & 0 & k_1 & k_2 \end{bmatrix} \begin{bmatrix} 0 \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ 0 \end{bmatrix} = (k * f)$$

The matrix A is a circulant Toeplitz matrix whose diagonals have a constant value and is constructed from the weights of the filter kernel. The circulant matrix of a vector $x = [x_1, \cdots, x_m]$ is defined as

$$\text{circ}(x) := \begin{bmatrix} x_1 & x_2 & \cdots & x_m \\ x_m & x_1 & \cdots & x_{m-1} \\ \vdots & \vdots & \ddots & \vdots \\ x_2 & x_3 & \cdots & x_1 \end{bmatrix} \tag{4.15}$$

The matrix in the above example can be written as $\text{circ}(\hat{k})$ where $\hat{k}$ is the zero-padded convolution kernel. For 2-dimensional convolutions, as defined above, A is a double block circulant matrix [4, 5]

$$A = \begin{bmatrix} K_1 & K_2 & \cdots & K_s \\ K_s & K_1 & \cdots & K_{s-1} \\ \vdots & \vdots & \ddots & \vdots \\ K_2 & K_3 & \cdots & K_1 \end{bmatrix} \tag{4.16}$$

where K_i is the circulant matrix corresponding to the i-th row of k, padded with zeros to the same size as the output of $k * f$, as in the example above, i.e. $K_i = \text{circ}(\hat{k}_i)$.[2] Up until now the matrix representation only describes the transformation from a single input to a single output channel. If $A_{i,j}$ is the transformation from the j-th input to the i-th output channel, horizontally concatenating over j and vertically over i, i.e.

$$A = \begin{bmatrix} A_{1,1} & \cdots & A_{1,c_{k-1}} \\ \vdots & \ddots & \vdots \\ A_{c_k,1} & \cdots & A_{c_k,c_{k-1}} \end{bmatrix} \tag{4.17}$$

yields the matrix representation of the whole convolutional layer. Here c_{k-1} and c_k denote the number of input and output channels respectively. With this matrix representation a convolution could also be interpreted as a fully connected layer with shared weights [36].

Another way is to represent a convolutional layer as element wise multiplications in the frequency domain by using Fourier transformations [37]. The Fourier transform of a function $f : \mathbb{R}^n \to \mathbb{R}$ is defined as

$$\mathcal{F}(f)(\xi) = \int_{\mathbb{R}^n} f(t) e^{-2\pi i \langle \xi, x \rangle} dx \tag{4.18}$$

This is a linear operator and can be inverted by using the following formula

$$f(x) = \int_{\mathbb{R}^n} \mathcal{F}(f)(\xi) e^{2\pi i \langle x, \xi \rangle} d\xi \tag{4.19}$$

If a is a discrete signal of length n, i.e. $a = (a_1, \cdots, a_n)$, the Discrete Fourier Transform (DFT) maps this signal to a vector of the same dimension in the frequency domain and

[2] the index here describes the whole row and not a single element like in the example

can be represented as the matrix multiplication with the DFT matrix F (as defined in [5]):

$$\hat{a} = \mathcal{F}(a) = Fa \tag{4.20}$$

For images, which are discrete two dimensional signals, the transformation is accomplished by first transforming vertically and then transforming the result horizontally (or the other way around), which for square images $A \in \mathbb{R}^{n \times n}$ is equivalent to

$$\hat{A} = F^T A F \tag{4.21}$$

The convolution theorem states that

$$f * g = \mathcal{F}^{-1}(\mathcal{F}(f) \cdot \mathcal{F}(g)) \tag{4.22}$$

With this the output of a convolutional layer can be written as

$$(\phi^{conv}(u))_j = \sum_{i=1}^{n_c} (\mathcal{F}(k_{i,j}) \odot \mathcal{F}(u_i)) \tag{4.23}$$

where $\odot$ denotes the pointwise multiplication operator. Note that for this to work, the $k_{i,j}$ and u_i have to be zero-padded to the same size because otherwise the pointwise multiplication is not defined. In the following in this zero-padding is implicitly assumed.

Batch Normalization

The distribution of the output of a deep neural network can vary greatly between each layer. For each new training example the distribution of the input data of the layers changes slightly and each layer of the network has to adapt to this change. Furthermore, very large or very small layer outputs lead to very large or very small gradients during backpropagation, which affects the learning speed negatively. Ioffe et al. [38] proposed a technique which normalizes the output of each layer for each mini-batch by transforming it to to have zero mean and unit variance. Because this might be too restrictive, two learnable parameters that are able to undo this normalization are introduced. Following the notation of [4], the output of a layer is transformed like this:

$$\phi^{bn}(x) = \operatorname{diag}\left(\frac{\gamma}{\sqrt{\operatorname{Var}[x]}}\right)(x - \mathbb{E}[x]) + \beta \tag{4.24}$$

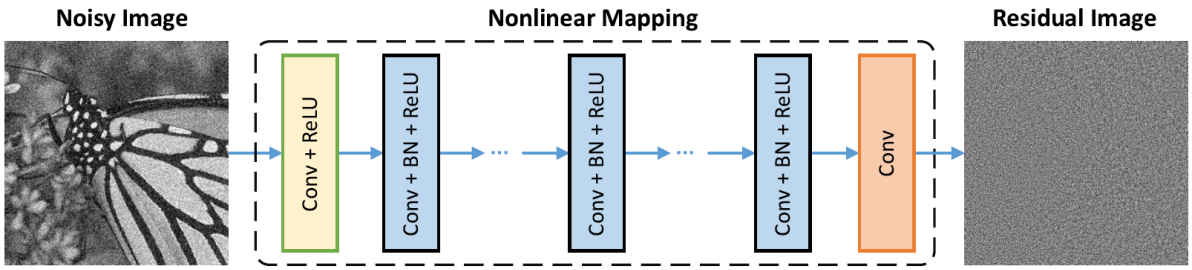


Figure 4.2: Architecture of DnCNN [2]

where γ and β are the learnable parameters of the same size as x and $\mathbb{E}[x]$ and $\mathrm{Var}[x]$ are the mean and variance of the current mini-batch. In modern deep networks batch normalization is very often used to decrease training time. Since there are no mini-batches during inference, a running mean and variance is calculated during training and used during inference.

4.2 Lipschitz Continuity of DnCNN

DnCNN (short for *Denoising Convolutional Neural Network*) [2] is a deep neural network which is able to reach state-of-the-art results for removing gaussian noise with known standard deviation from images. There is also a variant for blind denoising in which the noise level is not known beforehand.

The architecture of DnCNN can be seen in Figure 4.2. The first layer is a convolutional layer with 3×3 kernels that increase the number of channels to 64. This is followed by 15 convolutional layers, again with 3×3 kernels, which do not change the number of channels. Each of the middle layers use batch normalization between convolution and activation function. The last layer is one more convolutional layer which decreases the number of channels from 64 to the the same channel number as the input image had, which in this thesis will always be 1 (grayscale images).

Instead of directly learning to remove the noise from the input image, the output of the network is actually the raw noise itself. Removing the noise can then be achieved by subtracting it from the original image. This design decision is a form of residual learning [39] and makes it possible to train deeper networks in a shorter amount of time. However, this feature is detrimental when it comes to proving that the denoising by the

network is a contraction. The actual denoising function can be written as

$$\mathcal{G}(u) = \mathcal{N}_\theta(u) - u \tag{4.25}$$

where $\mathcal{N}_\theta(u)$ is the trained network. Since the Lipschitz constant of the (negative) identity function is 1, according to Lemma 3 an upper bound for the Lipschitz constant of $\mathcal{G}$ is *at least* 1, meaning that it cannot be proved that the denoiser is a contraction. Note that it is still possible that the best Lipschitz constant $\overline{L}(\mathcal{G})$ is smaller than 1, because for example the function $f(x) = x - x$ has best Lipschitz constant 0 and not 2, but it is not possible to be proven in the current theoretical framework used in this thesis.

To prove that it is *not* contractive or even expansive it is enough to give a counterexample, i.e. a pair of images u_1, u_2 for which $||\mathcal{G}(u_1) - \mathcal{G}(u_2)|| > ||u_1 - u_2||$. Surprisingly finding such images is not very hard. A variation of the fast gradient sign attack [40] was used. Given an image u_1 the idea behind this method is to go a small distance in the direction of the sign of the gradient of the network with respect to the input. Because the gradient points into the direction of the largest change, the hope is to have a large difference of the output of the network with only a slight change of the input. The formula for u_2 is as follows:

$$u_2 = u_1 + \epsilon \operatorname{sign}(\nabla \mathcal{G}(u_1)) \tag{4.26}$$

Using the official PyTorch version of DnCNN trained on a noise with standard deviation of 0.098 several expansive examples were able to be found with this method. In fact, almost any choice of u_1 which does not contain any noise is such a counterexample. Figure 4.3 shows such an example. It is also remarkable that DnCNN seems to add noise to the image if the image did not contain any noise in the first place.

4.3 Layer-wise Lipschitz Constant Computation

In order to constrain the weights of a network to make it contractive, a way to compute its current Lipschitz constant is required. Dividing the output of the network by this constant, for example, would make the network non-expansive. The calculated constant,

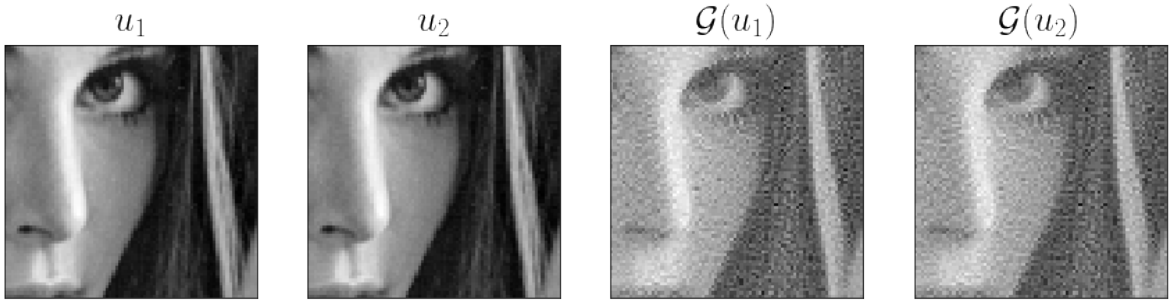


Figure 4.3: Proof that denoising with DnCNN is expansive w.r.t. the ℓ_2 norm.
$||\mathcal{G}(u_1) - \mathcal{G}(u_2)||_2 = 2.43 > 0.39 = ||u_1 - u_2||_2$

which is actually an upper bound of $\overline{L}$, has to be as small as possible in order to not restrict the network too much.

As proved in Lemma 2, the composition of Lipschitz continuous functions is Lipschitz continuous with a constant that is the product of the constants of its parts. Since neural networks are just function compositions this suggests to compute the Lipschitz constant for each layer independently to get an upper bound for the best Lipschitz constant. Gouk et al. [4] give ways to compute the Lipschitz constant of the most commonly used layers, which will be explained in the following. To underline that the Lipschitz continuity depends on the chosen norm, the Lipschitz constants with respect to p-norms will be denoted as L_p and the best Lipschitz constant as $\overline{L}_p$, if the norm matters in that context.

Fully Connected Layers

Fully connected layers are described by affine linear functions $\phi^{fc}(x) = Wx + b$. The Lipschitz constant with respect to a p-norm is a positive real number L_p that satisfies

$$\begin{aligned} &||(Wx_1 + b) - (Wx_2 + b)||_p \leq L_p||x_1 - x_2||_p \\ \Leftrightarrow &||Wx_1 - Wx_2||_p \leq L_p||x_1 - x_2||_p \\ \Leftrightarrow &||W(x_1 - x_2)||_p \leq L_p||x_1 - x_2||_p \end{aligned} \tag{4.27}$$

Setting $x = x_1 - x_2$ and assuming $x \neq 0$ yields

$$\frac{||Wx||_p}{||x||_p} \leq L_p \tag{4.28}$$

This means that every possible Lipschitz constant is smaller than the matrix norm induced by the p-norm, which is the supremum of the left expression. The smallest Lipschitz constant is the number L_p which fulfills the inequation with equality.

$$\overline{L}_p(\phi^{fc}) = \sup_{x \neq 0} \frac{||Wx||_p}{||x||_p} = ||W||_p \tag{4.29}$$

This shows that it is possible to compute the best Lipschitz constant and it does not depend on the bias of the layer. In case of $p \in 1, \infty$ the matrix norm can be computed easily with the equations in Lemma 1. In case of the 2-norm one has to compute the largest singular value of W. This can be done by calculating the singular value decomposition of W which consists of unitary matrices U and V and a diagonal matrix Σ such that

$$W = U\Sigma V^H \tag{4.30}$$

The largest entry of Σ is then the spectral norm of W. Another way to find the largest singular value is to find the largest eigenvalue of $A = W^T W$ which can be done using a power iteration of the form

$$x^{i+1} = \frac{Ax^i}{||Ax^i||_2} \tag{4.31}$$

Using this iteration, the sequence $(\sqrt{(||Ax^k||_2)})_k$ converges to the spectral norm of W. Note that this series approaches the real value of $||W||_2$ from below, which actually results in a value smaller than $\overline{L}_p$ that is not a Lipschitz constant. For a large amount of iterations this error is neglectable.

Convolutional Layers

As detailed in Chapter 4.1, convolutional layers can be interpreted as fully connected layers with shared weights and a weight matrix that is the concatenation of double block circular matrices of the form

$$A_{i,j} = \begin{bmatrix} K_1 & K_2 & \cdots & K_s \\ K_s & K_1 & \cdots & K_{s-1} \\ \vdots & \vdots & \ddots & \vdots \\ K_2 & K_3 & \cdots & K_1 \end{bmatrix} \tag{4.32}$$

where K_l is the Toeplitz matrix built with the entries from the l-th row of the convolution kernel $k_{i,j}$ that maps the i-th input channel to the j-th output channel. Due to the construction of $A_{i,j}$ each row and each column contains every entry of that convolution kernel exactly once, so the sum of each row or the sum of each column is the same as the sum of each kernel entry. This means that, according to Lemma 1, $||A_{i,j}||_1 = ||A_{i,j}||_\infty = ||k_{i,j}||_1$ where $||k_{i,j}||_1$ is the vector 1-norm applied to the 2-dimensional kernel $k_{i,j}$ interpreted as a 1-dimensional vector.

The full matrix that represents the convolutional layer is the block matrix

$$W = \begin{bmatrix} A_{1,1} & \cdots & A_{1,c_{k-1}} \\ \vdots & \ddots & \vdots \\ A_{c_k,1} & \cdots & A_{c_k,c_{k-1}} \end{bmatrix} \tag{4.33}$$

With the same explanation as in the last section the best Lipschitz constant of the layer is equal to the operator norm of that matrix

$$\overline{L}_p(\phi^{conv}) = ||W||_p \tag{4.34}$$

For $p = 1$ this is the maximum column sum of W. Analogously, for $p = \infty$ it is the maximum row sum.

$$\begin{aligned} ||W||_1 &= \max_j \sum_i ||A_{i,j}||_1 = \max_j \sum_i ||k_{i,j}||_1, \\ ||W||_\infty &= \max_i \sum_j ||A_{i,j}||_\infty = \max_i \sum_j ||k_{i,j}||_1 \end{aligned} \tag{4.35}$$

The 1-norm and ∞-norm can be computed very efficiently and directly depend on the values of the convolution kernels. The spectral norm is a lot harder to compute. Gouk et al. propose to use a power iteration analogous to equation 4.31 without constructing the matrix representation of the layer by using transposed convolutions [4, 41].

Sedghi et al. [5] propose an alternative approach that uses the Fourier transform of the convolutional layer to compute the singular values. This approach will be detailed in the following section. The notation from the paper will be adjusted to the notation used in this thesis. Let $k_{i,j} \in \mathbb{R}^{s \times s}$ be the convolution kernel that maps the i-th input channel to the j-th output channel. The discrete Fourier transform of this single 2-dimensional convolution kernel, $\mathcal{F}(k_{i,j})$, has a shape that depends on the size of the input image

that the convolution should be applied to. This is because the kernel has to be padded with zeros before the Fourier transform can be computed. The absolute value of each entry of the complex matrix $\mathcal{F}(k_{i,j})$ is a singular value of the corresponding convolution. However, this ignores the fact that in convolutional layers a single output channel is the *sum* of convolutions with all input channels, i.e.

$$\begin{aligned}(\phi^{conv}(u))_j &= \sum_{i=1}^{n_c}(k_{i,j} * u_i) \\ &= \sum_{i=1}^{n_c}(\mathcal{F}(k_{i,j}) \odot \mathcal{F}(u_i))\end{aligned}$$

The block $F_j := \mathcal{F}(k_{:,j})$ has the same shape as the input of the convolutional layer. There are as many blocks of this kind as there are output channels. Let $(F_j)_{s,t} \in \mathbb{R}^{1\times n_c}$ denote the the slice of F_j at the specific pixel position (s,t). Stacking all $(F_j)_{s,t}$ for $j = 1, ..., n_o$, where n_o is the number of output channels yields the matrix

$$M_{s,t} := \begin{bmatrix} — & (F_1)_{s,t} & — \\ & \vdots & \\ — & (F_{n_o})_{s,t} & — \end{bmatrix} \in \mathbb{R}^{n_o \times n_c} \tag{4.36}$$

The output slice of the layer at a *specific pixel position* (instead of at a specific channel as before) can then be written as

$$(\phi^{conv}(u))_{s,t} = M_{s,t}(\mathcal{F}(u)_{s,t}) \tag{4.37}$$

i.e. the matrix multiplication of $M_{s,t}$ with the corresponding slice of the Fourier transform of u at pixel position (s,t). Sedghi et al. [5] proved that the set of singular values of the convolutional layer is equal to the union of the singular values of the matrix $M_{s,t}$ at every possible pixel.

$$\sigma(\phi^{conv}) = \bigcup_{s,t} \sigma(M_{s,t}) \tag{4.38}$$

This means that any singular value of any $M_{s,t}$ greater than 1 would make the layer expansive.

Batch Normalization

Since batch normalization is a affine linear function

$$\phi^{bn}(x) = \text{diag}\left(\frac{\gamma}{\sqrt{\text{Var}[x]}}\right)(x - \mathbb{E}[x]) + \beta \tag{4.39}$$

The Lipschitz constant is equal to the matrix norm of the diagonal matrix

$$D := \text{diag}\left(\frac{\gamma}{\sqrt{\text{Var}[x]}}\right) \tag{4.40}$$

The row sum and column sum of diagonal matrices are always the same, so

$$||D||_1 = ||D||_\infty = \max|D_{i,i}| \tag{4.41}$$

The singular value decomposition of D is IDI^T where I is an appropriately sized identity matrix. This means that the spectral norm of a diagonal matrix is also $\max|D_{i,i}|$. Thus the best Lipschitz constant of a batch normalization layer is the same with respect to any p-norm with $p \in \{1, 2, \infty\}$

$$\overline{L}_p(\phi^{bn}) = \max_i \left|\frac{\gamma_i}{\sqrt{\text{Var}[x_i]}}\right| \tag{4.42}$$

Activation Functions

The largest slope of the ReLU is 1, thus making it non-expansive. Another interesting way to prove this is to see that ReLU is also the proximity operator of the convex characteristic function[3] of the non-negative real numbers

$$\chi_{[0,\infty)}(x) = \begin{cases} 0, & \text{if } x \in (0, \infty) \\ \infty, & \text{otherwise} \end{cases} \tag{4.43}$$

Because the function is ∞ if x is outside of the set, the prox operator reduces to

$$\text{prox}_{\chi_{[0,\infty)}}(v) = \underset{x \in [0,\infty)}{\text{argmin}} ||x - v||_2 \tag{4.44}$$

[3]This is different than the usual characteristic function which is 1 when x is in the set and 0 else

If $v < 0$, the closest point in $[0, \infty]$ is 0 and if $v \geq 0$ then v is in the set and the closest point is v itself. Thus, the proximal operator can be rewritten as

$$\text{prox}_{\chi_{[0,\infty)}}(v) = \max(0, v) = \text{ReLU}(v) \tag{4.45}$$

In fact most common activation functions are proximity operators of some function which is shown in [42]. Since they have Lipschitz constant 1 they can be ignored when calculating the Lipschitz constant of the whole network.

4.4 Lipschitz Normalization

In this section we will give methods on how to restrict the parameters of a layer in order to make it Lipschitz continuous with a given constant λ. The restriction of the parameters should happen during the training of the network by projecting the weights of the layers to the feasible set, i.e. weights with which the layer has the right Lipschitz constant. This can be achieved either after each or just some gradient descent steps, which then, with the projection operator π, can be modified to

$$\theta^{i+1} = \pi(\theta^i - \tau \nabla E(\theta^i)) \tag{4.46}$$

The current Lipschitz constant (before the normalization) with respect to a given p-norm will be conveniently denoted as $||W||_p$, where W is the linear part of the respective layer. This includes convolutional layers, which have been shown to just be a special case of fully connected layers.

Gouk et al. [4] propose to enforce a Lipschitz constant using the following projection

$$\pi_p(W, \lambda) = \frac{1}{\max(1, \frac{||W||_p}{\lambda})} W \tag{4.47}$$

Note that multiplying the weights of a filter of a convolutional layer with the fraction in this formula yields the same result because of the weight sharing property. This formula projects W onto the closest linear operator with the wanted Lipschitz constant, where "closest" means that the distance measured as the operator norm of the difference of

the projection and W is as small as possible.

$$\pi_p(W, \lambda) = \underset{||W'||_p \leq \lambda}{\operatorname{argmin}} ||W - W'||_p \tag{4.48}$$

Gouk et al. enforce Lipschitz continuity in their paper as a way to regularize a network and improve its performance on classification tasks. For this, they generally use relatively large $\lambda > 1$ and remark that smaller λ negatively influence network performance. For the purposes in this thesis $\lambda = 1$ is required and first tests have shown that the normalization scheme proposed by Gouk et al. is to restrictive. Normalizing the network completely destroyed the training progress up to that point. For this reason a different projection, that measures closeness as a change of the weights of the layers which is as small as possible, was used. This can be expressed as using the Frobenius norm instead of the operator norm in the projection formula

$$\pi_F(W, \lambda) = \underset{||W'||_p \leq \lambda}{\operatorname{argmin}} ||W - W'||_F \tag{4.49}$$

$||W - W'||_F$ should be interpreted as the Frobenius norm of the matrix representation of the layers with the respective weights. For Lipschitz continuity with respect to the 1-norm it is easier to find the projection with respect to the absolute sum matrix norm $||W||_S = \sum_{i,j} |W_{i,j}|$ which leads to the analogous formula

$$\pi_S(W, \lambda) = \underset{||W'||_p \leq \lambda}{\operatorname{argmin}} ||W - W'||_S \tag{4.50}$$

Finding a minimizer for these expressions requires to only change the subset of weights which make the Lipschitz constant too large while not changing the weights that have no influence. Finding this subset differs for each p-norm and will in the following be described for the cases $p = 1$ and $p = 2$.

For $p = 1$ and given a fully connected layer, the Lipschitz constant is the maximum absolute column sum of the weight matrix. Any column that sums to more than 1 has to be normalized by dividing its weights by that sum. If a column sums to less than 1 then it does not have a influence on the expansiveness of the layer. Let $W_{:,j}$ denote the i-th column of W. A formula for the above idea can then be given by the row-wise definition

$$\pi_S(W_{:,j}, \lambda) = \frac{1}{\max\left(1, \frac{\sum_i |W_{i,j}|}{\lambda}\right)} W_{:,j} \tag{4.51}$$

This is the ℓ_1 projection of the j-th row onto the the ℓ_1 ball with radius λ and applying this to each row indeed results in the closest matrix in terms of the absolute sum matrix norm, which is just the extension of the ℓ_1 norm on the whole matrix. In convolutional layers the formula can be modified similarly by only modifying some kernel weights

$$\pi_S(k_{:,j}, \lambda) = \frac{1}{\max\left(1, \frac{\sum_i ||k_{i,j}||_1}{\lambda}\right)} k_{:,j} \tag{4.52}$$

With these formulas the weights can be efficiently normalized with just a slight overhead after each gradient descent step.

In the case of $p = 2$, just like computing the Lipschitz constant, the normalization of a layer is more complicated. It can be achieved by clipping the singular values in the singular value decomposition $W = U\Sigma V^T$:

$$\pi_F(W, \lambda) = U\Sigma' V^T \tag{4.53}$$

with

$$\Sigma'_{i,i} = \max(\lambda, \Sigma_{ii}) \tag{4.54}$$

This is indeed the closest (w.r.t. the Frobenius norm) matrix with Lipschitz constant λ. To prove this (the proof is slightly modified from [43]), consider the identity

$$\begin{aligned} ||W||_F =& \sqrt{\mathrm{tr}(W^H W)} \\ =& \sqrt{\mathrm{tr}((U\Sigma V^H)^H (U\Sigma V^H))} \\ =& \sqrt{\mathrm{tr}(V\Sigma^H U^H U\Sigma V^H)} \\ =& \sqrt{\mathrm{tr}(V\Sigma^H \Sigma V^H)} \\ =& \sqrt{\mathrm{tr}(\Sigma V^H V\Sigma^H)} \\ =& \sqrt{\mathrm{tr}(\Sigma\Sigma^H)} \\ =& \sqrt{\sum_{i=1}^{n} \Sigma_{ii}^2} \end{aligned} \tag{4.55}$$

This also means that the Frobenius norm is invariant under unitary transformations. If $W = U\Sigma V^H$ is the singular value decomposition of W then $\Sigma = U^H W V$. The

minimization problem for the projection can now be written as

$$\begin{aligned}\pi_F(W,\lambda) &= \underset{||W'||_2\leq\lambda}{\operatorname{argmin}} ||W-W'||_F \\ &= \underset{||U^HW'V||_2\leq\lambda}{\operatorname{argmin}} ||U^HWV-U^HW'V||_F \\ &= \underset{||U^HW'V||_2\leq\lambda}{\operatorname{argmin}} ||\Sigma-U^HW'V||_F\end{aligned} \tag{4.56}$$

Finding $\pi_F(W,\lambda)$ is equivalent to minimizing

$$\hat{X} = \underset{||X||_2\leq\lambda}{\operatorname{argmin}} ||\Sigma - X||_F \tag{4.57}$$

The solution can then be computed with $\pi_F(W,\lambda) = U\hat{X}V^H$. With the binomial formula for norms induced by an inner product it follows that

$$\begin{aligned}||\Sigma-\hat{X}||_F^2 =& ||\Sigma||_F^2 - 2\,\mathrm{Re}\langle\Sigma,\hat{X}\rangle + ||\hat{X}||_F^2 \\ =& ||\Sigma||_F^2 - 2\,\mathrm{Re}(\mathrm{tr}\Big(\Sigma^H\hat{X}\Big)) + ||\hat{X}||_F^2 \\ \geq& ||\Sigma||_F^2 - 2\,\mathrm{Re}(\mathrm{tr}\Big(\Sigma^H\hat{\Sigma}\Big)) + ||\hat{\Sigma}||_F^2 \\ =& ||\Sigma-\hat{\Sigma}||_F^2\end{aligned} \tag{4.58}$$

Since $\hat{X}$ and $\hat{\Sigma}$ have the same singular values and thus the same Lipschitz constant, $\hat{\Sigma}$ would be a matrix closer to Σ on the feasible set, unless $\hat{X} = \hat{\Sigma}$. Because of this we can suppose that $\hat{X}$ is already of diagonal form. Otherwise it would not be a valid projection. Let σ and $\hat{\sigma}$ be vectors consisting of the diagonal entries of the diagonal matrices Σ and $\hat{\Sigma}$ respectively. The minimization problem can, again, be rewritten as

$$\hat{\sigma} = \underset{||\sigma'||_\infty\leq\lambda}{\operatorname{argmin}} ||\sigma-\sigma'||_2 \tag{4.59}$$

because the spectral norm is the largest singular value. This is the euclidean projection of σ onto the ∞-norm ball with radius λ which can be achieved with the formula

$$\hat{\sigma_i} = \max(\lambda, \sigma_i) \tag{4.60}$$

Thus formula 4.54 gives the right projection with respect to the Frobenius norm.

Convolutional layers can be normalized by either building the singular value decompo-

sition of the matrix representation or by doing a Fourier transformation of the filter kernels. In both cases the clipping of the singular values will generally not result in a convolution kernel of the same size as before. This thesis will use the variant with the Fourier transformation. As explained in the previous subchapter, the singular values of a convolutional layer can be computed by calculating the singular value decompositions of matrices that correspond to the pixels of the layer input.

$$(\phi^{conv}(u))_{s,t} = M_{s,t}(\mathcal{F}(u)_{s,t})$$

Just as with fully connected layers above, the singular values can be clipped in order to get a new matrix at that pixel. Since each matrix entry has a one-to-one correspondence to a entry in the Fourier transform of the filter kernel, a normalized kernel can be constructed by gathering the normalized matrix entries and transforming the result back into the spacial domain. As mentioned above, in general this will not result in a kernel of the same size. Inverse Fourier transformation and subsequent discarding of all values outside of the original kernel dimensions will alter the singular values again so that it is not clear if the network still has the right Lipschitz constant. Sedghi et al. [5] propose to repeatedly perform singular value clipping in the frequency domain and restricting the kernel size again. Since the set of Lipschitz continuous kernels with a specific constant and the set of kernels of a specific size are convex, this iteration converges to a kernel in the intersection of both sets. However, for the use case in this thesis this process turned out to be not efficient and it was not feasible to repeat it after each gradient descent step. For this reason a network architecture that exploits the convolution theorem and directly learns the convolutional weights in the frequency domain will be used. This way the kernel does not have to be inverse Fourier transformed and the layer is provably Lipschitz continuous after a single singular value clipping. The network architecture will be further explained in the following subchapter.

4.5 Inherently Non-Expansive Networks

It was already shown that some classes of functions like proximal operators are inherently non-expansive. In this section a network architecture that is the composition of non-expansive functions and is thus itself also non-expansive will be derived. This architecture will be based on the wavelet shrinkage denoising algorithm [44].

The wavelet transform, similar to the Fourier transform, is a linear operator that transforms a signal into a new representation with respect to a orthonormal basis. In a continuous setting the transform of a signal f is defined as

$$\mathcal{W}_\psi f(a,b) = \frac{1}{\sqrt{a}} \int_{-\infty}^{\infty} \overline{\psi\left(\frac{t-b}{a}\right)} f(t) dt \tag{4.61}$$

ψ is called *mother wavelet* and generates a orthonormal system of basis functions

$$\psi_{jk}(x) = 2^{\frac{j}{2}} \psi(2^j x - k) \tag{4.62}$$

These are translated and scaled versions of the mother wavelet. The mother wavelet can be any function ψ which satisfies

$$\int_{-\infty}^{\infty} \psi(t)^2 dt < \infty \tag{4.63}$$

In this thesis the Daubechies 4 tap wavelet [45] will be used, which is often used for discrete wavelet transforms in signal processing. The Daubechies wavelets are orthogonal which means that their associated wavelet transform is non-expansive. In case of a discrete 2-dimensional image the wavelet transform can be interpreted as follows. The image is high-pass filtered to yield three detail images for horizontal, vertical and diagonal details. Furthermore, the image is low-pass filtered and downscaled. The original image can be reconstructed from this downsampled image and the three detail images. This procedure can be applied recursively to the downscaled image and results in a pyramid-like structure. Figure 4.4 shows the wavelet transform with recursion depth 2 of the Lenna test image. The top right, bottom left and bottom right quadrants show the vertical, horizontal and diagonal detail images respectively, while the top left quadrant contains the low-passed image, which by itself has again been decomposed into three detail and one downscaled image. A lot of noise can be seen in the detail images in the lowest pyramid level. The idea of wavelet denoising is to apply a threshold function to the detail images in order to suppress the noise. One possible threshold function is given by simply setting the input to zero if the absolute value of the signal is smaller than a given value. This is called hard thresholding and can be seen in Figure 4.5.

$$H_\lambda(x) = \begin{cases} x, \text{ if } |x| > \lambda \\ 0, \text{ otherwise} \end{cases} \tag{4.64}$$

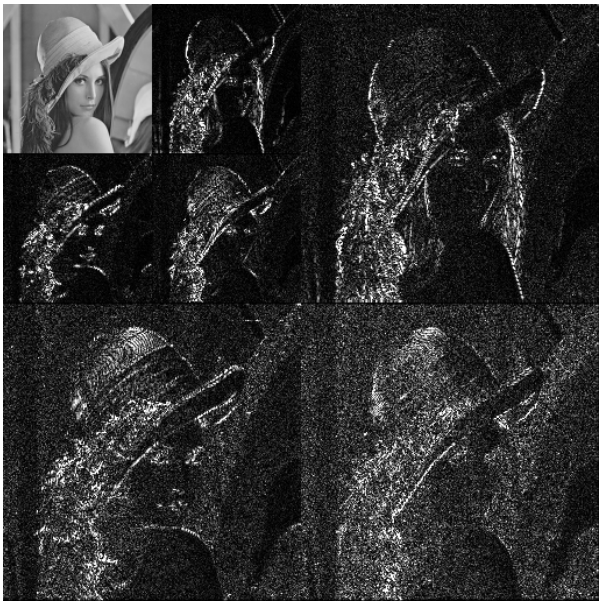


Figure 4.4: Example of of a wavelet pyramid

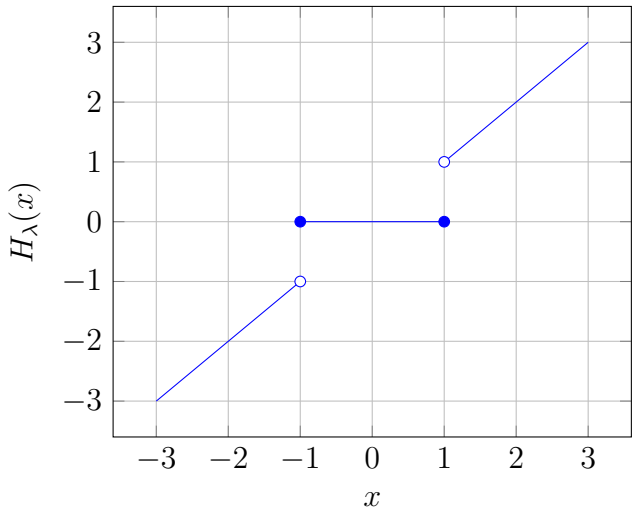


Figure 4.5: Hard thresholding with $\lambda = 1$

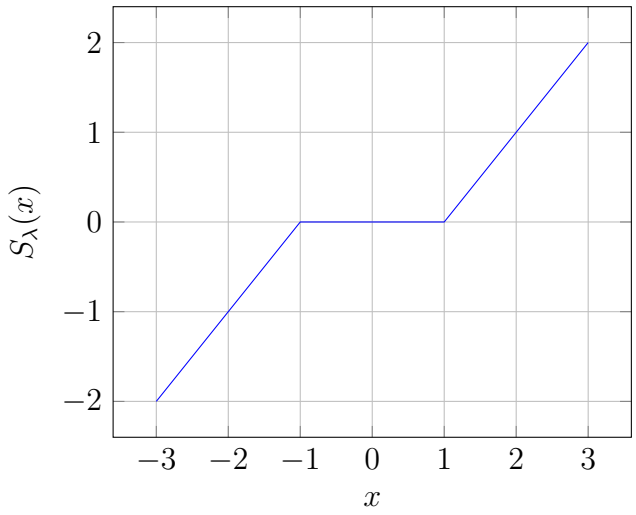


Figure 4.6: Soft thresholding with $\lambda = 1$

Hard thresholding is not non-expansive. Consider, for example, $x_1 = 2$ and $x_2 = 3$. Then

$$|H_2(x_1) - H_2(x_2)| = |0 - 3| = 3 > |x_1 - x_2| = |2 - 3| = 1 \tag{4.65}$$

In Figure 4.5 it can be seen that there are non-continuities at $\pm\lambda$ which means that hard thresholding is not Lipschitz continuous at all. For this reason instead *soft thresholding* [46, 47] will be used, which is also known as the shrinkage operator[4]. It is defined as

$$S_\lambda(x) = \begin{cases} x - \lambda, \text{ if } x > \lambda \\ x + \lambda, \text{ if } x < -\lambda \\ 0, \text{ otherwise} \end{cases} \tag{4.66}$$

If x is a vector this formula should be seen element-wise. The graph of the function in Figure 4.6 shows that there are no discontinuities and the largest slope of the function is 1. Another way to see that the shrinkage operator is always non-expansive is to see that S_λ is a proximal operator:

$$\text{prox}_{\lambda|x|} = S_\lambda(x) \tag{4.67}$$

Proofs for this can be found in [46] and [47]. Since the wavelet transform and its inverse are non-expansive themselves it is now clear that the whole denoising algorithm is non-expansive when using the shrinkage operator.

[4]sometimes also soft shrinkage

The choice of the parameter λ depends on the noise level and can be made using different heuristics [48]. In this thesis a learning based approach to learn a a different parameter λ for each level in the wavelet domain is used. This leads to a simple network architecture that is inherently non-expansive.

5 Numerical Experiments

First, all tried network architectures for image denoising in our experiments will be explained. After that, the trained networks will be used to solve inverse image problems as explained in Chapter 3.1. The results will be compared to the results of DnCNN and usual energy minimization methods.

5.1 Network Architectures

This thesis will conduct experiments on enforcing Lipschitz continuity on three different network architectures for image denoising. First, an attempt to normalize an architecture that is very similar to DnCNN was made. Because this architecture did not give satisfying results another architecture which learns convolutional weights directly in the frequency domain was tried. Lastly, a wavelet architecture which is non-expansive by definition and does not require any further Lipschitz normalization during training was tested. The following sections will explain especially the last two architectures in more detail. Each network will be trained on grayscale images corrupted with gaussian noise with standard deviation 0.098 [1]. The training and test data were taken from the PyTorch implementation of DnCNN[2]. For testing, "Set12", which was also used in the DnCNN paper, was used. All networks were trained using the Adam optimizer [34].

Because of the ease of implementation, the first experiments were done by trying to enforce Lipschitz continuity with respect to the 1-norm. As seen in chapter 4.3, the Lipschitz constant with respect to the 1-norm can be computed by simply taking the maximum row sum of the weight matrices. First tests were conducted with a simpler

[1] This assumes that pixel values have a range from 0 to 1. The convention in the DnCNN paper assumes values from 0 to 255 and thus a standard deviation of 25 in the their paper is equivalent to $0.098 \approx 25/255$ here, which explains the odd value used here

[2] `https://github.com/SaoYan/DnCNN-PyTorch`

version of DnCNN which only has 3 convolutional layers (1 layer that increases the number of channels to 64, 1 layer that keeps the number of channels the same and the output layer which decreased the number of channels back to 1). To keep it simple, and because it was not really necessary for such a shallow network, contrary to DnCNN, no batch normalization was used. Because of the statements in chapter 4.2, also no a skip connection was used and instead denoised image was learned directly.

Without Lipschitz normalization, the training resulted in a performance of an average PSNR of 27.6 dB on the test set[3]. However, with 1-norm Lipschitz normalization, as described as in Chapter 4.4, the network achieved much worse results. Using $\pi_1(W, 1)$ as proposed in [4] to normalize the layer weights after each gradient descent step, which means to divide all weights by the Lipschitz constant of the layer if it is greater than 1, resulted in a PSNR of approximately 10 dB on both training and test data. With the less aggressive projection $\pi_S(W, 1)$ a PSNR of 20.5 dB could be achieved, which is around the same PSNR value as the identity function would get for that noise level.

As discussed in Chapter 4.4 and in [5], Lipschitz-normalizing convolutional layers with respect to the euclidean norm by singular value clipping does not result in convolutional kernels of the same size as before the normalization. Setting the kernel values outside of the original size to zero changes the Lipschitz constant of the layer again. The algorithm proposed by [5] to circumvent this is to iterate the following two steps until convergence:

1. Clip the singular values of the convolutional layer in frequency domain
2. Transform the kernel back to the spatial domain. Since the kernel now has most likely the wrong size, discard all values outside of the original kernel's size

Multiple iterations of this are required in order to converge to a kernel that has the right size and also the right Lipschitz constant. Tests showed that the computation time for this procedure was too long to be feasible to do after each gradient descent step. Only doing the projection each n-th optimization step, for example each 100th step, completely destroyed the training process up to that step. For these reasons it was decided to instead use a architecture which directly learns coefficients in the frequency domain and computes convolutions as multiplications with the Fourier transformed input

[3]The full DnCNN achieves a PSNR of 30.436 for our noise level according to [2]

by using the convolution theorem. This can be interpreted as a convolution with a kernel of learnable size [37]. In the following this architecture will be referred to as *FFTNet*.

FFTNet consists of Fourier convolutional layers. Their weights are n_o many blocks of complex numbers of dimension $w \times h \times n_i$, where w and h are the input image dimensions, n_i is the amount of input channels and n_o is the amount of output channels of the layer. This means that each of those blocks has the same dimension as the Fourier transformed input volume which makes element-wise multiplication possible. Separate learnable weights for the real and imaginary part were used. Like in formula 4.23, the output of the element-wise multiplication of the input image and each block of weights is then summed up in channel direction. The result is a single channel of the output volume. Since there are n_o weight blocks, this results is an output volume of dimension $w \times h \times n_o$. Note that unlike DnCNN, which only consists of usual convolutional layers and can thus take any image size as input, the width and height of the input to a Fourier convolutional layer is fixed.

An open question is how to introduce activation functions to a layer like this. If there was an activation function that could be applied in the frequency domain, a forward and inverse Fourier transform per layer could be saved. There have been proposals to compute the ReLU as a convolution in the frequency domain [49]. However, because it is a lot more complicated to implement, and slight performance optimizations are not the focus of this thesis, it was instead decided to compute the ReLU in the spatial domain after an inverse Fourier transform. The result of the inverse Fourier transform is in general a complex number. Trabelsi et al. [50] tested different extensions of the ReLU to complex numbers and got the best results by just applying it separately to the real and imaginary part of the input [4]

$$\mathbb{C}\,\mathrm{ReLU}(a + ib) = \mathrm{ReLU}(a) + i\,\mathrm{ReLU}(b) \tag{5.1}$$

To make the computation of the singular values for the Lipschitz normalization as fast as possible, FFTNet only consists of two such Fourier convolutional layers. The first layer increases the number of channels to 40 and the second layer decreases them again to 1. The imaginary part is discarded for the output and the size of input images is fixed to 80×80. This means that the Lipschitz normalization requires the singular value decomposition of $80 \cdot 80$ many 40×1 matrices for the first and 1×40 matrices for

[4] The name $\mathbb{C}$ReLU is taken from [50]

the second layer (see equation 4.38). These are row and column vectors whose singular values can easily be calculated by taking their norm. If $v \neq 0$ is a column vector, its singular value decomposition is $U\Sigma V^T$ with [5]

$$\begin{aligned} U_{:,1} =& \frac{v}{||v||_2}, \\ \Sigma =& \begin{bmatrix} ||v||_2 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \text{ and} \\ V =& 1 \end{aligned} \tag{5.2}$$

In the case of column vectors like this, their singular value is just their euclidean norm and clipping that value is equivalent to normalizing the vector in case that the norm is greater than 1. For row vectors the reasoning is analogous. This can be efficiently done after each gradient descent step.

With Lipschitz normalization, the FFTNet achieved an average PSNR of 26.41 on the Set12 test set. Since the network can only take 80×80 images as input, the images in Set12 were split into overlapping patches. Even though this result is worse than DnCNN without normalization, it is a lot better than the results of normalizing the small DnCNN variant with respect to the 1-norm earlier in this chapter. Figure 5.1 shows the absolute values of the 40 weight maps in the first layer of the trained normalized network. As seen by the visible structure around the vertical and horizontal axes, the network learned more than a simple gaussian low-pass filter.

The last architecture is the inherently non-expansive wavelet based network described in chapter 4.5 and will be referred to as *WaveletNet*. It consists of a discrete wavelet transform of recursion depth 3 and subsequent application of the soft shrinkage operator on the detail images. The soft shrinkage operator has a different learnable parameter for each wavelet level, so the whole network only has 3 learnable parameters.

Table 5.1 shows the training results of all tested architectures in a single place. Since normalizing the small DnCNN architecture did not show competitive performance compared to FFTNet and WaveletNet, it will not be included in the following experiments.

[5] Note that only the first column of U matters here. The other columns would have to be orthogonal to v in order to be a valid singular value decomposition.

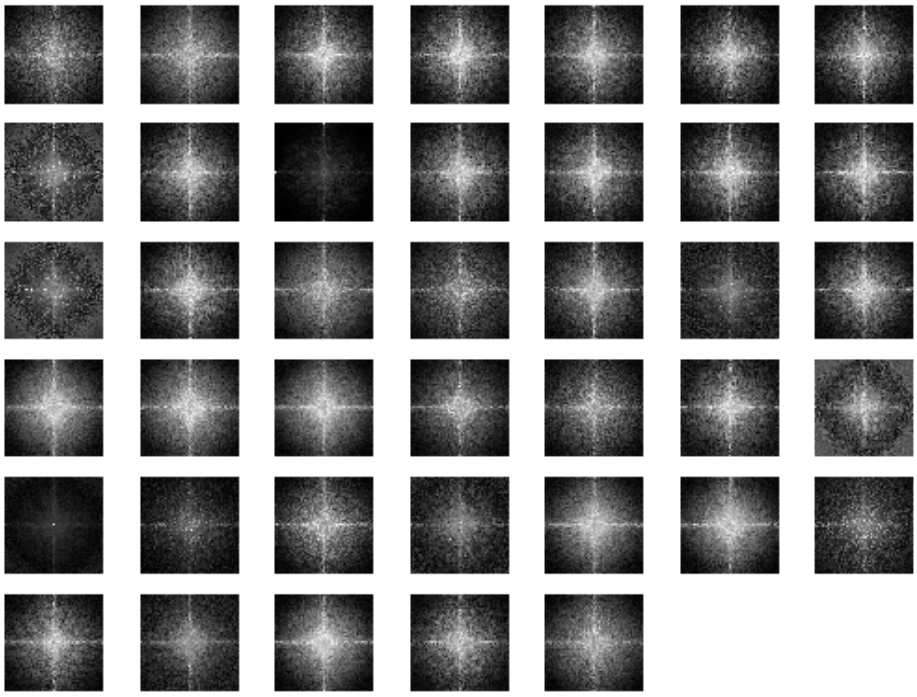


Figure 5.1: Absolute value of the weights of the first layer of normalized FFTNet

Network Architecture (Normalization Scheme)	PSNR[dB]
DnCNN (not normalized)	30.44
small DnCNN (not normalized)	27.63
small DnCNN (ℓ_1-normalized with π_1)	10.03
small DnCNN (ℓ_1-normalized with π_S)	20.51
FFTNet (not normalized)	27.31
FFTNet (ℓ_2-normalized)	26.41
WaveletNet	25.46

Table 5.1: Training results for different networks and different Lipschitz normalization schemes. WaveletNet is inherently non-expansive w.r.t. the euclidean norm

Conv Layer ℓ_1	Conv Layer ℓ_2	Batchnorm ℓ_1/ℓ_2
7.00	16.06	-
125.50	28.55	0.25
104.17	17.02	0.60
97.48	12.84	0.57
96.73	12.91	0.57
99.60	14.71	0.49
97.56	13.85	0.48
101.17	13.86	0.59
101.56	14.61	0.46
102.11	13.27	0.56
97.94	14.42	0.39
96.44	13.62	0.54
95.37	14.42	0.46
99.24	14.98	0.58
95.63	13.11	0.44
93.51	14.13	0.14
18.88	1.34	-

Table 5.2: Layer-wise Lipschitz constants of DnCNN with respect to 1-norm and 2-norm.

Table 5.2 shows the layer-wise Lipschitz constants of the full, unnormalized DnCNN architecture. The product of all values in the ℓ_1 or the ℓ_2 columns yields a Lipschitz constant for the whole network. Note that the Lipschitz constant of batch normalization layers is the same with respect to the 1-norm and 2-norm, so they share a column. This results in a ℓ_1 Lipschitz constant of approximately $7.6 \cdot 10^{26}$ and a ℓ_2 constant of $4.3 \cdot 10^{13}$. In comparison, the ℓ_2 Lipschitz constant of the unnormalized FFTNet is only 73.62 (first layer approx. 19.97 and second layer 3.69), which shows the problem of exploding Lipschitz constants for very deep networks because of its multiplicativity.

5.2 Image Reconstruction

The image reconstruction task used to conduct the experiments is gaussian deblurring. Images u were corrupted with gaussian blur of standard deviation 1. To this blurred image gaussian noise with standard deviation 0.02 is added, so that the final image is of the form

$$f = Au + n \tag{5.3}$$

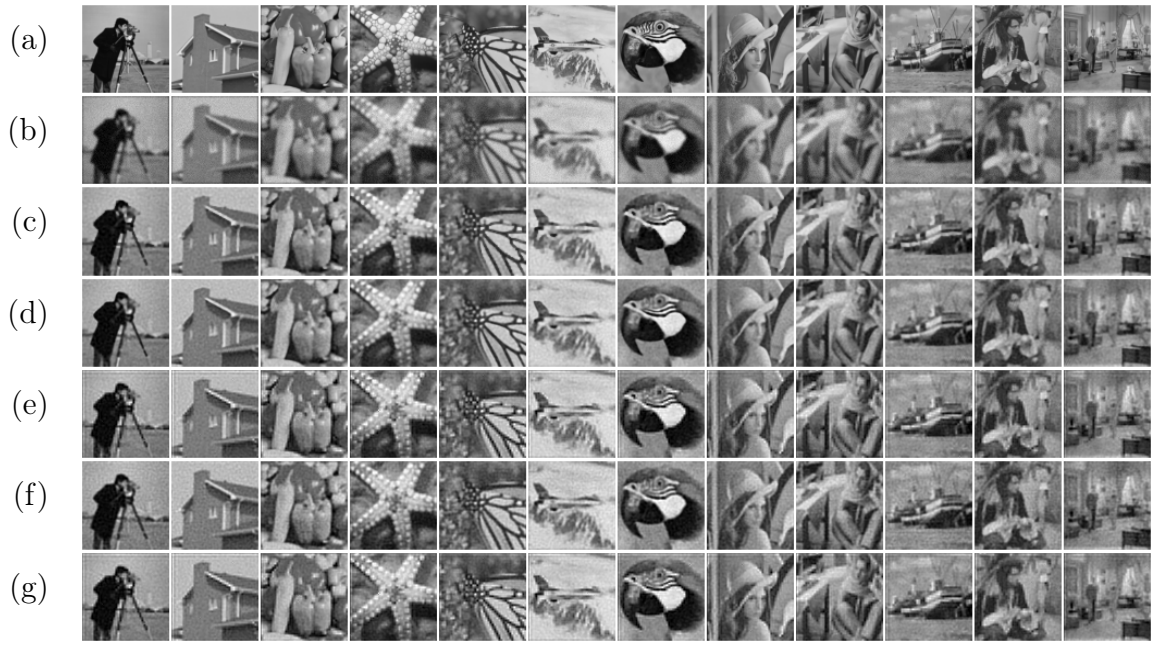


Figure 5.2: Best deblurred images for each algorithm out of 100 repetitions with random starting images u_0. (a) original image (b) blurred image (c) classical energy minimization (d) DnCNN (e) FFTNet (not normalized) (f) FFTNet (normalized) (g) WaveletNet

where A is the blur operator and n is the noise. The task is to recover u as closely as possible when only f and A is known.

Five different approaches are used to solve the problem. First, a classical energy minimization approach is employed by calculating

$$\hat{u} = \underset{u}{\operatorname{argmin}} \frac{1}{2}||Au - f||_2^2 + \alpha||Du||_{Huber} \tag{5.4}$$

with gradient descent. The other four approaches calculate a fixed point iteration of the form

$$u^{k+1} = (1 - \alpha)(u^k - \tau A^T(Au^k - f)) + \alpha\mathcal{G}(u^k) \tag{5.5}$$

where $\mathcal{G}$ is either DnCNN, normalized/unnormalized FFTNet or WaveletNet. The left side

$$u^k - \tau A^T(Au^k - f) \tag{5.6}$$

is the same as a gradient descent step on $||Au - f||_2^2$ (see Chapter 3.3).

The deblurring experiment was conducted on 12 different images. Since FFTNet can

Method		PSNR[dB]											
		Cameraman	House	Peppers	Starfish	Butterfly	Plane	Bird	Lena	Barbara	Boat	Man	Couple
Energy Min	min	28.63	29.96	28.87	27.53	26.46	28.12	26.23	29.21	28.98	29.02	29.18	28.81
	max	29.12	30.52	29.27	27.88	26.90	28.54	26.67	29.69	29.33	29.46	29.52	29.23
	mean	28.85	30.25	29.04	**27.70**	26.64	**28.34**	26.50	29.46	**29.88**	**29.25**	**29.34**	**29.02**
	std. dev.	0.088	0.100	0.093	0.068	0.078	0.075	0.079	0.086	0.080	0.078	0.078	0.083
DnCNN	min	26.20	22.16	25.48	24.96	21.76	28.12	26.79	29.63	28.91	28.94	19.28	24.37
	max	29.39	30.77	29.34	27.93	27.48	28.44	27.58	30.10	29.31	29.28	29.28	29.04
	mean	**29.15**	**30.31**	**29.05**	27.65	**26.98**	28.28	**27.19**	**29.88**	29.10	29.11	29.01	28.65
	std. dev.	0.415	1.020	0.594	0.455	0.994	0.070	0.140	0.097	0.081	0.074	1.038	0.855
FFTNet (not normalized)	min	28.47	29.43	28.16	27.00	26.12	27.71	26.18	28.70	28.43	28.71	28.78	28.64
	max	29.08	30.20	28.80	27.40	26.63	28.27	26.63	29.20	28.93	29.25	29.28	29.04
	mean	28.65	29.84	28.51	27.22	26.31	28.04	26.42	28.95	28.67	28.95	29.04	28.80
	std. dev.	0.101	0.143	0.106	0.084	0.097	0.130	0.103	0.105	0.108	0.111	0.098	0.092
FFTNet	min	27.94	29.03	28.13	27.33	26.04	27.64	25.92	28.54	28.36	28.61	28.52	28.38
	max	28.28	29.64	28.55	27.66	26.34	28.11	26.22	28.93	28.77	29.04	28.88	28.82
	mean	28.09	29.32	28.35	27.49	26.20	27.87	26.08	28.74	28.57	28.82	28.73	28.56
	std. dev.	0.074	0.100	0.086	0.070	0.059	0.077	0.061	0.082	0.102	0.083	0.088	0.091
WaveletNet	min	25.17	25.68	26.64	25.46	23.98	25.46	24.07	26.59	26.01	26.33	26.48	26.58
	max	25.63	26.05	27.10	25.83	24.29	25.84	24.32	27.00	26.45	26.75	27.01	27.04
	mean	25.52	25.92	26.93	25.68	24.15	25.65	24.19	26.83	26.24	26.57	26.79	26.90
	std. dev.	0.065	0.072	0.080	0.068	0.061	0.066	0.061	0.078	0.072	0.074	0.092	0.073

Table 5.3: Deblurring results on different images. Every experiment was repeated 100 times with different random starting points u_0 and different random gaussian noise.

only take 80×80 images as input due to its architecture, the test images were all scaled to that size. Before the experiments a grid search was used to find the best values for the hyperparameters α and τ. For the classical energy minimization approach $\alpha = 50$ and for DnCNN, FFTNet and WaveletNet $\alpha = 0.15$ was chosen. As expected, a value for α that is too small results in a image that is too noisy because the influence of the regularizer becomes too small. All methods use a learning rate of $\tau = 1.9$ and each experiment was repeated 100 times with different randomly initialized starting vectors u_0 for the fixed point iterations and random noise n. The best deblurred images out of all 100 repetitions can be seen in Figure 5.2 and statistical results for each experiment can be found in Table 5.3. The performance of WaveletNet is around 2 dB worse compared to the other methods in all experiments. The classical energy minimization approach and DnCNN have similar performance, where each one of the two is better than the other on some images. Both the normalized and unnormalized versions of FFTNet show similar results and are slightly worse than DnCNN in terms of the maximal PSNR. The normalized FFTNet has lower standard deviation than the unnormalized FFTNet on all test images. Interestingly, the variance of the results is much higher for DnCNN than for all other algorithms, with the difference of the highest and lowest PSNR spanning as far as 8 dB on the *House* test image, setting it even lower than the lowest result of WaveletNet. Comparatively, the variances of the other algorithms are close to zero. Since the starting vectors and the noise of the blurred image are the only factors that change between the repetitions of each experiment, this suggests that the DnCNN algorithm

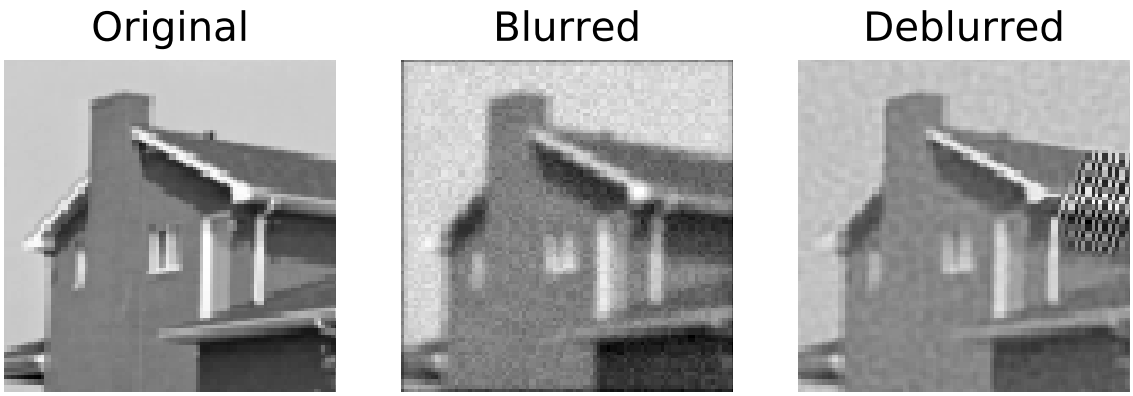


Figure 5.3: Failed deblurring by the DnCNN algorithm. The deblurred image has a PSNR of 24.25 dB

somehow fails for some configurations. Figure 5.3 shows such a case. Even though the fixed point iteration converges, the resulting image shows a clear artifact near the right image border.

To explore this effect, the experiment was repeated for different values of α between 0.2 and 0.8. The results in Figure 5.4 show that the standard deviation of the PSNR (vertical bars) decreases with greater values of α, however the mean performance also decreases, making DnCNN worse than all other methods if similar standard deviation is required. It is not clear if this behavior is caused by the expansiveness of DnCNN since there is no full-sized and normalized DnCNN for comparison. Greater α actually increases the influence of the network in the fixed point iteration (see equation 5.5), so that a possible correlation between expansiveness and higher standard deviation would more likely show a greater standard deviation for greater α and not the opposite.

Instead, it is also possible that the failures depend on the starting vector u_0. To test this, the experiments were repeated with a fixed initialization $u_0 = 0$, so that the only difference between repetitions is the random noise n. From the results in Table 5.4 it can be seen that now the standard deviation of the method with DnCNN is similar to that of the other methods and there are no more outliers. Interestingly, the standard deviations of the other algorithms did not improve much, which leads to the conclusion that the DnCNN method is more sensitive to its starting vector.

Even with this improvement, the deblurring performance of DnCNN from Meinhardt et al. [1], which was better than the energy minimization approach, could not be replicated

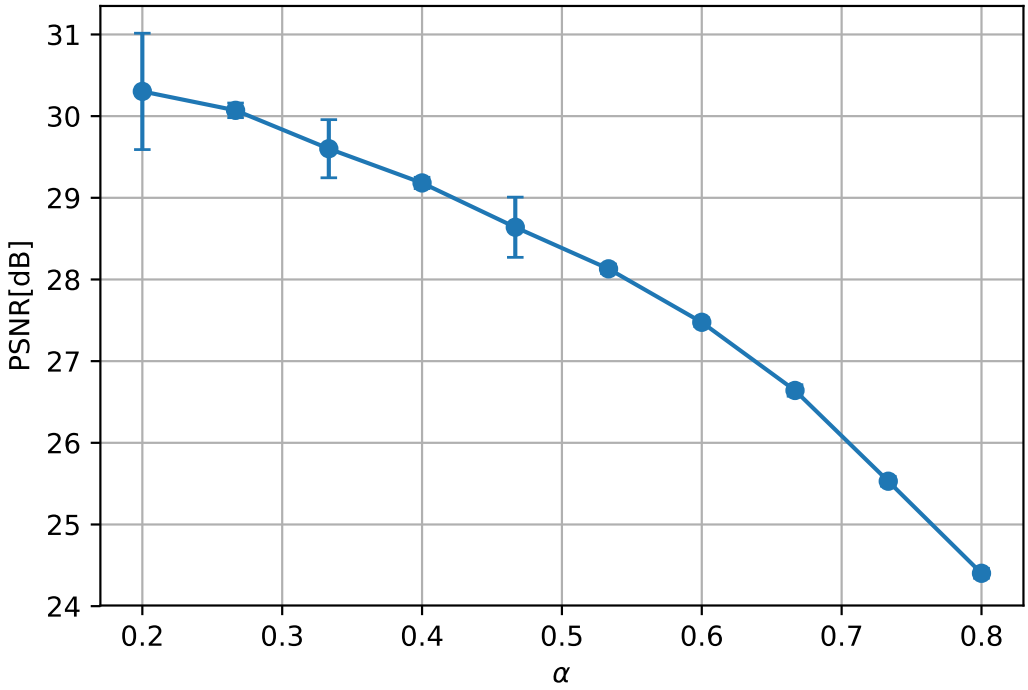


Figure 5.4: Different DnCNN deblurring results on the *House* image for varying α

Method		PSNR[dB]											
		Cameraman	House	Peppers	Starfish	Butterfly	Plane	Bird	Lena	Barbara	Boat	Man	Couple
Energy Min	min	28.70	29.98	28.87	27.58	26.49	28.21	26.30	29.24	28.99	29.02	29.15	28.82
	max	29.09	30.67	29.32	27.85	26.83	28.58	26.69	29.67	29.39	29.45	29.55	29.19
	mean	28.88	30.25	29.07	**27.73**	26.63	**28.35**	26.50	29.46	**29.19**	**29.24**	**29.36**	**29.03**
	std. dev.	0.080	0.111	0.092	0.061	0.068	0.081	0.074	0.092	0.084	0.080	0.085	0.080
DnCNN	min	28.83	30.14	28.95	27.59	26.89	28.04	26.65	29.57	28.86	28.93	28.92	28.64
	max	29.25	30.56	29.34	27.86	27.25	28.37	26.97	29.99	29.30	29.30	29.29	28.97
	mean	**29.01**	**30.32**	**29.14**	27.72	**27.05**	28.22	**26.83**	**29.74**	29.07	29.10	29.13	28.81
	std. dev.	0.080	0.095	0.085	0.052	0.068	0.068	0.070	0.084	0.086	0.072	0.079	0.066
FFTNet (not normalized)	min	28.55	29.69	28.35	27.21	26.19	27.99	26.31	28.78	28.53	28.87	28.94	28.72
	max	28.96	30.28	28.85	27.50	26.59	28.27	26.76	29.34	29.01	29.34	29.31	29.03
	mean	28.74	29.94	28.63	27.34	26.43	28.14	26.52	29.09	28.81	29.14	29.12	28.87
	std. dev.	0.084	0.108	0.095	0.067	0.077	0.062	0.083	0.104	0.095	0.086	0.079	0.076
FFTNet	min	27.90	29.06	28.19	27.27	26.03	27.73	25.88	28.59	28.44	28.61	28.54	28.45
	max	28.23	29.54	28.66	27.70	26.35	28.08	26.22	28.96	28.75	29.10	28.92	28.78
	mean	28.09	29.33	28.42	27.51	26.17	27.91	26.05	28.78	28.60	28.85	28.76	28.60
	std. dev.	0.065	0.091	0.092	0.075	0.062	0.073	0.063	0.083	0.077	0.086	0.072	0.070
WaveletNet	min	25.55	25.98	26.99	25.68	24.18	25.72	24.19	26.91	26.28	26.62	26.78	26.91
	max	25.78	26.46	27.30	25.96	24.41	25.98	24.47	27.24	26.58	26.95	27.16	27.18
	mean	25.69	26.15	27.13	25.86	24.29	25.86	24.35	27.04	26.44	26.78	26.97	27.06
	std. dev.	0.049	0.071	0.066	0.056	0.048	0.056	0.049	0.066	0.063	0.066	0.065	0.059

Table 5.4: Deblurring results on different images. Every experiment was repeated 100 times with $u_0 = 0$ and different random gaussian noise.

Method	Running Time	Number of iterations
Energy Min	384 ms ± 5.4 ms	71.3 ± 1.0
DnCNN	107 ms ± 3.83 ms	19.9 ± 2.3
FFTNet (not normalized)	113 ms ± 4.4 ms	27.4 ± 3.2
FFTNet (normalized)	91 ms ± 1.06 ms	22.5 ± 1.1
WaveletNet	221 ms ± 4.78 ms	22.2 ± 1.9

Table 5.5: Mean running time of deblurring methods and number of iterations until convergence ($u_0 = 0$). Values have the format (mean ± standard deviation). All algorithms ran on a Nvidia RTX 2080 GPU.

by the method in this thesis. A reason for this could be that they used a different underlying algorithm (the Pock-Chambolle primal-dual algorithm [51]) in which the regularizer was replaced by DnCNN, while this thesis uses a modified algorithm (5.5) because of its easier convergence analysis. They also trained their DnCNN on a noise level of $\sigma = 0.02$ as opposed to $\sigma = 25/255 \approx 0.098$ here. Furthermore, different hyperparameters like the learning rate or the convergence threshold could be at fault.

Directly comparing the unnormalized and normalized versions of FFTNet gives a more interesting insight in the convergence properties because they only differ in their Lipschitz constant. Table 5.5 compares the running time and number of iterations until convergence of all methods. The fixed point iteration is considered to have converged if the mean squared difference between two iterations is smaller than 10^{-8}. The method with the normalized FFTNet does not only converge faster but the number of iterations also has a much smaller variance than for the unnormalized FFTNet. This leads to an average decrease in running time of approximately 19%. The much larger increase in running time compared to the number of iterations for DnCNN and WaveletNet can be explained by their network architecture. A forward pass in general takes longer to compute than it is the case for FFTNet.

A comparison for different α values for the two FFTNet methods can be seen in Figure 5.4. Interestingly, for large α the performance of the unnormalized FFTNet method degraded drastically and even resulted in NaNs. At the same time the number of iterations exploded and had to be capped at 1000. In comparison, the normalized FFTNet method converged for all α and gave reasonable results, even though the PSNR became also lower for larger α.

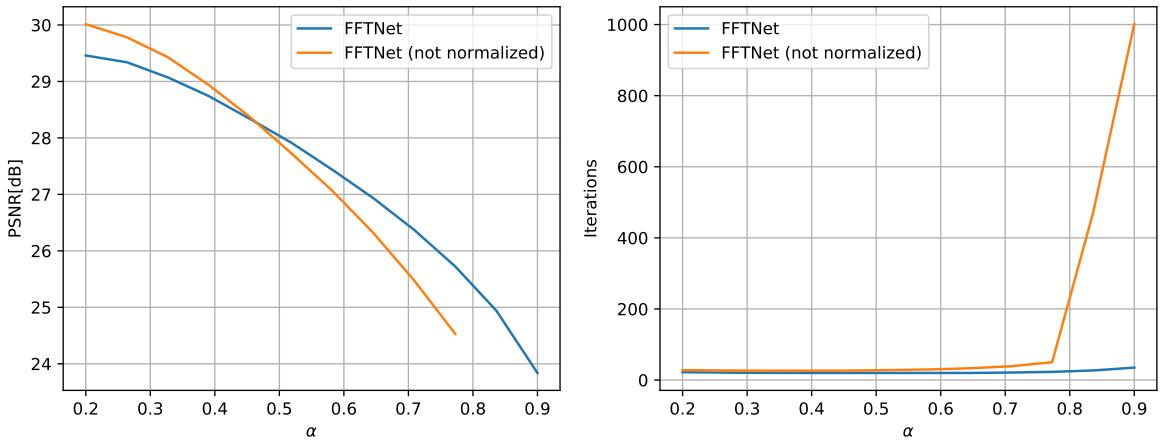


Figure 5.5: Comparison of normalized and unnormalized FFTNet for varying α on the *House* test image. For $\alpha > 0.75$, the calculation of the PSNR of the result with the unnormalized FFTNet resulted in a NaN because the MSE was too large. The maximum number of iterations was set to 1000.

6 Conclusions

This thesis succeeded in the goal to train a non-expansive denoising network. It gave a framework to compute an upper bound for the Lipschitz constant of a neural network consisting of common layer types and use that bound to effectively normalize it. However, with FFTNet, a special network architecture was required in order to achieve even close to competitive denoising performance. Even though this network had worse results than state-of-the-art denoisers, it has the desirable theoretical property that using it as a regularizer in energy minimization approach for image reconstruction problems produces a provably convergent fixed point iteration.

The deblurring experiments showed that, when using the same network architecture, this fixed point iteration converges faster if the network is non-expansive. Moreover, there were even hyperparameters for which the algorithm with the expansive network completely failed while the algorithm with the non-expansive network converged. In security critical applications, in which it is not acceptable to ever get invalid results, it might be feasible to sacrifice some image reconstruction quality for provable convergence.

There are many possible directions for future work. In this thesis no assumption on the domain of the neural network was made, i.e. it was assumed that the input was an arbitrary vector in $\mathbb{R}^n$. It is likely that the Lipschitz constant of the network restricted to the set of valid images with pixel values in the interval $[0, 1]$ is lower than the Lipschitz constant of the same network on the full domain. Normalizing with this Lipschitz constant is less restrictive on the network weights and should thus lead to better performance. Another assumption that was made is that *every* layer of the network has to be non-expansive in order for the whole network to be non-expansive, because the Lipschitz constant of a function composition is the product of the Lipschitz constant of the individual functions. However, instead of requiring all factors to be 1, a less restrictive condition is for the product of all factors to be 1, i.e. some layers can have a larger Lipschitz constant if at the same time other layers have a smaller one. Since it is

not clear which layer should get which constant, it is preferable to make the per-layer Lipschitz constant learnable.

The computed Lipschitz constants for networks in this thesis were only upper bounds for the best Lipschitz constants. In combination with activation functions, especially the ReLU which potentially maps large parts of its input to zero, it is possible that the best Lipschitz constant of a network is much smaller than those upper bounds. This could be especially useful for very deep networks, as could be seen by the exploding upper bounds for the Lipschitz constants of DnCNN. Future work could develop tighter bounds for these special cases.

The wavelet-based inherently non-expansive network in this thesis showed results that were not competitive with the other approaches. Since it only consisted of a single wavelet denoising layer, it could be explored how to make deeper architectures, that still remain inherently non-expansive, in order to increase the expressiveness of the network.

Lastly, a different approach by Gouk et al. [52] for enforcing the Lipschitz constant could be considered. In that work, the Lipschitz constant of a layer is approximated by the maximal gradient with respect to the input over the training set. This approximation, which Gouk et al. call *gain*, is in general smaller than the true best Lipschitz constant and thus using it for the normalization schemes in this thesis would not make the network non-expansive. However, since this approach incorporates the distribution of real images by using a training set to calculate the gain, it could be interesting to compare a network normalized by using gain and a network normalized by the method in this thesis in terms of their performance as regularizers in image reconstruction tasks and their respective convergence properties.

Bibliography

[1] Tim Meinhardt, Michael Möller, Caner Hazirbas, and Daniel Cremers. Learning proximal operators: Using denoising networks for regularizing inverse imaging problems. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[2] Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. Beyond a Gaussian denoiser: Residual learning of deep CNN for image denoising. *IEEE Transactions on Image Processing*, 26(7):3142–3155, 2017.

[3] Yuichi Yoshida and Takeru Miyato. Spectral Norm Regularization for Improving the Generalizability of Deep Learning. *arXiv e-prints*, page arXiv:1705.10941, May 2017.

[4] Henry Gouk, Eibe Frank, Bernhard Pfahringer, and Michael Cree. Regularisation of Neural Networks by Enforcing Lipschitz Continuity. *pre-print*, page arXiv:1804.04368, Apr 2018.

[5] Hanie Sedghi, Vineet Gupta, and Philip M. Long. The singular values of convolutional layers. In *International Conference on Learning Representations*, 2019.

[6] Yusuke Tsuzuku, Issei Sato, and Masashi Sugiyama. Lipschitz-Margin Training: Scalable Certification of Perturbation Invariance for Deep Neural Networks. In *Advances in Neural Information Processing Systems 31*, pages 6542–6551. Curran Associates, Inc., 2018.

[7] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations (ICLR)*, 2018.

[8] Gene H. Golub and Chares F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, third edition, 1996.

[9] John M. Lee. *Introduction to Smooth Manifolds*. Springer, 2003.

[10] Leonid I. Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. In *Proceedings of the Eleventh Annual International Conference of the Center for Nonlinear Studies on Experimental Mathematics : Computational Issues in Nonlinear Science: Computational Issues in Nonlinear Science*, pages 259–268, New York, NY, USA, 1992. Elsevier North-Holland, Inc.

[11] W. Robert Mann. Mean value methods in iteration. In *Proceedings of the American Mathematical Society*, 1953.

[12] M. A. Krasnosel'skii. Two remarks on the method of successive approximations. In *Uspekhi Mat. Nauk (Russian)*, volume 10, pages 123–127, 1955.

[13] Ernest Ryu and Stephen Boyd. A primer on monotone operator methods survey. *Applied and computational mathematics*, 15:3–43, 01 2016.

[14] Neal Parikh and Stephen Boyd. Proximal algorithms. *Foundations and Trends in Optimization*, 1(3), 2013.

[15] M. Burger, A. Sawatzky, and G. Steidl. *First Order Algorithms in Variational Image Processing*, pages 345–407. Springer International Publishing, Cham, 2016.

[16] Peter J. Huber. Robust estimation of a location parameter. *Ann. Math. Statist.*, 35(1):73–101, 03 1964.

[17] M. T. McCann, K. H. Jin, and M. Unser. Convolutional neural networks for inverse problems in imaging: A review. *IEEE Signal Processing Magazine*, 34(6):85–95, Nov 2017.

[18] Li Xu, Jimmy SJ Ren, Ce Liu, and Jiaya Jia. Deep convolutional neural network for image deconvolution. In *Advances in Neural Information Processing Systems 27*, pages 1790–1798. Curran Associates, Inc., 2014.

[19] K. Zhang, W. Zuo, and L. Zhang. Ffdnet: Toward a fast and flexible solution for cnn-based image denoising. *IEEE Transactions on Image Processing*, 27(9):4608–4622, Sep. 2018.

[20] Junyuan Xie, Linli Xu, and Enhong Chen. Image denoising and inpainting with deep neural networks. In *Advances in Neural Information Processing Systems 25*, pages 341–349. Curran Associates, Inc., 2012.

[21] Rolf Köhler, Christian Schuler, Bernhard Schölkopf, and Stefan Harmeling. Mask-specific inpainting with deep neural networks. pages 523–534, 09 2014.

[22] Raymond A. Yeh, Chen Chen, Teck Yian Lim, Alexander G. Schwing, Mark Hasegawa-Johnson, and Minh N. Do. Semantic Image Inpainting with Deep Generative Models. *arXiv e-prints*, page arXiv:1607.07539, Jul 2016.

[23] M. Richart, J. Visca, and J. Baliosian. Image colorization with neural networks. In *2017 Workshop of Computer Vision (WVC)*, pages 55–60, Oct 2017.

[24] J. H. Rick Chang, Chun-Liang Li, Barnabás Póczos, B. V. K. Vijaya Kumar, and Aswin C. Sankaranarayanan. One network to solve them all — solving linear inverse problems using deep projection models. *arXiv preprint arXiv:1703.09912*, 2017.

[25] Felix Heide, Markus Steinberger, Yun-Ta Tsai, Mushfiqur Rouf, Dawid Pająk, Dikpal Reddy, Orazio Gallo, Jing Liu, Wolfgang Heidrich, Karen Egiazarian, Jan Kautz, and Kari Pulli. Flexisp: A flexible camera image processing framework. *ACM Trans. Graph.*, 33(6):231:1–231:13, November 2014.

[26] S. V. Venkatakrishnan, C. A. Bouman, and B. Wohlberg. Plug-and-play priors for model based reconstruction. In *2013 IEEE Global Conference on Signal and Information Processing*, pages 945–948, Dec 2013.

[27] A. Buades, B. Coll, and J. . Morel. A non-local algorithm for image denoising. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 60–65 vol. 2, June 2005.

[28] K. Dabov, A. Foi, and K. Egiazarian. Video denoising by sparse 3d transform-domain collaborative filtering. In *2007 15th European Signal Processing Conference*, pages 145–149, Sep. 2007.

[29] Michael Moeller and Daniel Cremers. *Image Denoising—Old and New*, pages 63–91. Springer International Publishing, Cham, 2018.

[30] S. Sreehari, S. V. Venkatakrishnan, B. Wohlberg, G. T. Buzzard, L. F. Drummy, J. P. Simmons, and C. A. Bouman. Plug-and-play priors for bright field electron tomography and sparse interpolation. *IEEE Transactions on Computational Imaging*, 2(4):408–423, Dec 2016.

[31] S. H. Chan, X. Wang, and O. A. Elgendy. Plug-and-play admm for image restoration: Fixed-point convergence and applications. *IEEE Transactions on Computational Imaging*, 3(1):84–98, March 2017.

[32] Xavier Glorot, Antoine Bordes, and Y Bengio. Deep sparse rectifier neural networks. *Journal of Machine Learning Research*, 15, 01 2010.

[33] Ilya Sutskever, James Martens, George E. Dahl, and Geoffrey E. Hinton. On the importance of initialization and momentum in deep learning. In *International Conference on Machine Learning (ICML)*, 2013.

[34] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[35] Yann LeCun, Patrick Haffner, Leon Bottou, and Yoshua Bengio. *Object Recognition with Gradient-Based Learning*, pages 319–345. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

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

[37] Harry Pratt, Bryan M. Williams, Frans Coenen, and Yalin Zheng. Fcnn: Fourier convolutional neural networks. In *ECML/PKDD*, 2017.

[38] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, pages 448–456, 2015.

[39] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, June 2016.

[40] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples. *arXiv e-prints*, page arXiv:1412.6572, Dec 2014.

[41] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *arXiv e-prints*, page arXiv:1603.07285, Mar 2016.

[42] Patrick L. Combettes and Jean-Christophe Pesquet. Deep Neural Network Structures Solving Variational Inequalities. *arXiv e-prints*, page arXiv:1808.07526, Aug 2018.

[43] S. Lefkimmiatis, J. P. Ward, and M. Unser. Hessian schatten-norm regularization for linear inverse problems. *IEEE Transactions on Image Processing*, 22(5):1873–1888, May 2013.

[44] C. Taswell. The what, how, and why of wavelet shrinkage denoising. *Computing in Science Engineering*, 2(3):12–19, May 2000.

[45] Ingrid Daubechies. *Ten Lectures on Wavelets*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1992.

[46] Patrick Combettes and Valérie R. Wajs. Signal recovery by proximal forward-backward splitting. *SIAM Journal on Multiscale Modeling and Simulation*, 4, 01 2005.

[47] I. Daubechies, M. Defrise, and C. De Mol. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Communications on Pure and Applied Mathematics*, 57(11):1413–1457, 2004.

[48] G. P. Nason. *Choice of the Threshold Parameter in Wavelet Function Estimation*, pages 261–280. Springer New York, New York, NY, 1995.

[49] A. Vasilyev. Cnn optimizations for embedded systems and fft. *Stanford, Thesis*, 2015.

[50] Chiheb Trabelsi, Olexa Bilaniuk, Ying Zhang, Dmitriy Serdyuk, Sandeep Subramanian, Joao Felipe Santos, Soroush Mehri, Negar Rostamzadeh, Yoshua Bengio, and Christopher J Pal. Deep complex networks. In *International Conference on Learning Representations (ICLR)*, 2018.

[51] Antonin Chambolle and Thomas Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of Mathematical Imaging and Vision*, 40(1):120–145, May 2011.

[52] Henry Gouk, Bernhard Pfahringer, Eibe Frank, and Michael Cree. MaxGain: Regularisation of Neural Networks by Constraining Activation Magnitudes. *arXiv e-prints*, page arXiv:1804.05965, Apr 2018.