Project work

Fabian Hielscher

Image reconstruction performance comparison between compressed sensing and automated transform by manifold approximation

July 25, 2019

supervised by:
Prof. Dr.-Ing. Tobias Knopp
M.Sc. Mirco Grosser

Hamburg University of Technology
Institute for Biomedical Imaging
Schwarzenbergstraße 95
21073 Hamburg

University Medical Center Hamburg-Eppendorf
Section for Biomedical Imaging
Martinistraße 52
20246 Hamburg

Ich versichere an Eides statt, die vorliegende Arbeit selbstständig und nur unter Benutzung der angegebenen Quellen und Hilfsmittel angefertigt zu haben.

Hamburg, den ??.??.2010

Contents

Abstract

MRI scan times can be decreased if only a portion of k-space data is sampled. However, undersampling data leads to image degradation as not all relevant data is available. Compressed sensing and AUTOMAP (Automated transform by manifold approximation) are two different techniques that use undersampled k-space data to reconstruct images with little image degradation. These techniques are compared and evaluated in this thesis. AUTOMAP is an image reconstruction framework which is implemented by a neural network. The AUTOMAP reconstruction performance is optimized by input data preprocessing and hyperparameter optimization. It turns out that the AUTOMAP reconstruction performance improves by additional utilization of the momentum method and dropout layers with high keep probabilities. Image reconstruction comparison show that state of the art compressed sensing outperforms AUTOMAP that is trained with natural images.

1 Introduction

MRI is an imaging modality that uses non-ionizing radiation to create diagnostic images. MRI images show superior soft tissue contrast as compared to CT scans making it a well-suited examination of the brain, joints, spine, and other soft tissue body parts. The number of MRI scan examinations rose constantly in Germany from 7.4 million in 2008 to 11.1 million in 2015 [5]. This increasing demand requires efficient scan times. One of the biggest problems in MRI is the long scan time, which can take up over 60 minutes to complete. Most radiologists use breath-hold techniques for most abdominal T2 MRI sequences. Breath-hold physiology changes blood flow parameters and can alter expected TOF loss and associated dark T2 signal, which can be reduced by shorter scan times [6]. Shorter scan times are less stressful for patients and enable more economical MRI operation due to higher patient throughput [7].

The number of all different combinations of pixels that form an image is huge, e.g. the number of different 6×6 8-bit grayscale images is already bigger than the number of atoms in the universe. But from all these different combinations of pixels only a little fraction contains natural images we recognize. Many signals (like images) are sparse when represented in some domain [8]. There are many forms of lossy compression that utilize this fact, e.g. JPEG and JPEG2000 respectively use discrete cosine transform and wavelet transform to represent images in sparse domains. These transforms are well suited for image compression, because they transform the images into vectors of sparse coefficients, which are stored for decoding and reconstructing images. This leads to the question whether artifact-free images can be recovered from incomplete data as we omit most of the acquired data by compression. That means, imaging experiments could be accelerated by collecting a reduced amount of data. As every pixel contributes to one image, image data acquisition must take place in a sparse domain.

Deep learning is a broad class of machine learning techniques that shows particular promise in extracting high level abstractions from the raw data of very large, heterogeneous, high-dimensional data sets [1]. There are many applications for deep learning techniques with biomedical data e.g. enhancement of cancer diagnosis and classification [2], annotation of gene expression patterns [3] or real-time 3D image registration for functional MRI [4].

Compressed sensing (CS) and automated transform by manifold approximation (AUTOMAP) are two different techniques that use undersampled k-space data to reconstruct MRI images. The goal of this thesis was to train the AUTOMAP neural network on general natural images (in contrast to domain-specific images). The performance of the neural network was optimized

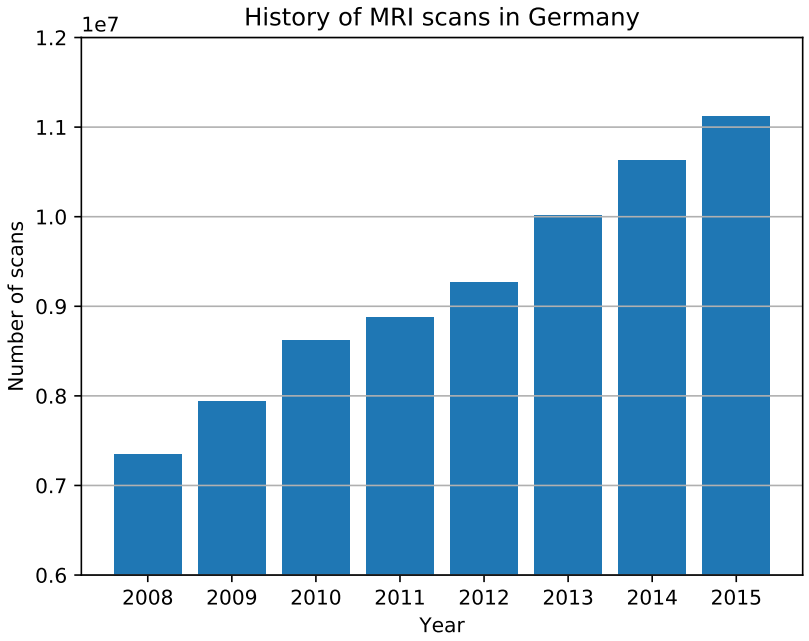


Figure 1.1.: Number of magnetic resonance imaging (MRI) scan examinations in Germany from 2008 to 2015 [5]

using different training strategies and it was compared to a state-of the art compressed sensing reconstruction.

2 Theory

In this chapter the basic techniques of MRI data acquisition and image reconstruction are described. The following sections in this chapter give insights about CS, AUTOMAP and the way image reconstructions are evaluated in this thesis.

2.1. MRI

2.1.1. Basics

Protons have a positive charge and possess a spin, hence they have a magnetic field. In the absence of an external magnetic field, the individual magnetic moments are randomly oriented and cancel each other out. If an external magnetic field is applied to protons, they align parallel (low energy state) or anti-parallel (high energy state) with the magnetic field lines and precess around them. The frequency of this precession is proportional to the external magnetic field and is described by the Larmor equation:

$$\omega_0 = \gamma B_0 \tag{2.1}$$

where ω_0 denotes the Larmor frequency and γ denotes the gyro-magnetic ratio, which is different for different materials (e.g. γ_{water}=42.5 MHz/T). As most of the protons align parallel to the magnetic field, the net magnetization vector, which is the sum of all the magnetic moments, also aligns the magnetic field.

Resonance occurs when a RF-pulse of the same frequency as the precessing protons is applied to the protons. The energy transfer causes many protons to flip and align anti-parallel on a higher energy level which decreases the longitudinal net magnetization due to cancellation of magnetic moments. A 90°-RF-pulse flips the net magnetization by 90° from longitudinal axis (z-axis) to the transversal plane (xy-plane). If the RF-pulse is switched off, the longitudinal and transversal net magnetization start relaxing back to the equilibrium state.

Longitudinal relaxation describes the realignment of the magnetization vector with the external magnetic field. T_1 relaxation is described by

$$M_z(t) = M_0(1 - \mathrm{e}^{-\frac{t}{T_1}}) \tag{2.2}$$

where T_1 denotes a decay constant (longitudinal relaxation time).

Transverse relaxation describes the dephasing of spins due to spin-spin interactions. The decaying transversal net magnetization induces an electric current which is the signal received in MR by a receiver coil and is called free induction decay. T_2 relaxation is described by:

$$M_{xy}(t) = M_0 \mathrm{e}^{-\frac{t}{T_2}} \tag{2.3}$$

where T_2 denotes a decay constant (transversal relaxation time).

In practice, inhomogeneities in the main magnetic field lead to much faster transverse magnetization decay than predicted by natural atomic mechanisms. The time constant for this observed decay rate is denoted T_2* whereas T_2 can be considered the time constant for the natural decay rate.

2.1.2. Pulse sequence

A pulse sequence describes a succession of RF pulses over time. The choice of a pulse sequence and corresponding parameters determine signal intensities of different types of tissues. If the 90°-pulse is repeated after a sufficiently short amount of time (TR) tissues with different T_1 can be distinguished due to the different transversal net magnetization after the second pulse. This difference occurs, because of the difference in longitudinal magnetization of tissues when the second 90°-pulse is applied. The resulting image is called a T_1-weighted image, because the tissue contrast is mainly due to their difference in T_1. Using different pulse sequences allows the generation of different tissue contrasts.

Right after a 90°-pulse is switched off the protons start to dephase which leads to transversal magnetization decrease. Major causes for this process are static local field disturbances and interactions between neighboring spins. To maximize signal intensity a 180°-pulse can be applied exactly between switching off the 90°-pulse and the signal measurement. The 180°-pulse flips the dephasing protons around 180° which initiates rephasing. This way at the time of measurement the protons are in phase again which leads to higher transversal magnetization, hence higher signal intensity. This sequence is called spin echo. TE is the time to echo and can be chosen by the operator.

2.1.3. Spatial encoding

With MRI the signal is localized in the 3D space by manipulating the magnetic properties of the nuclei in a predictable way. Each MR scanner has three sets of spatial encoding electrical coils to produce magnetic fields in the x, y, and z directions. These gradients distort the static main magnetic field B_0 linearly in a predictable pattern, which causes the resonance frequency to vary as a function of position.

There are three steps involved in determining where a 3D location a signal is arising from:

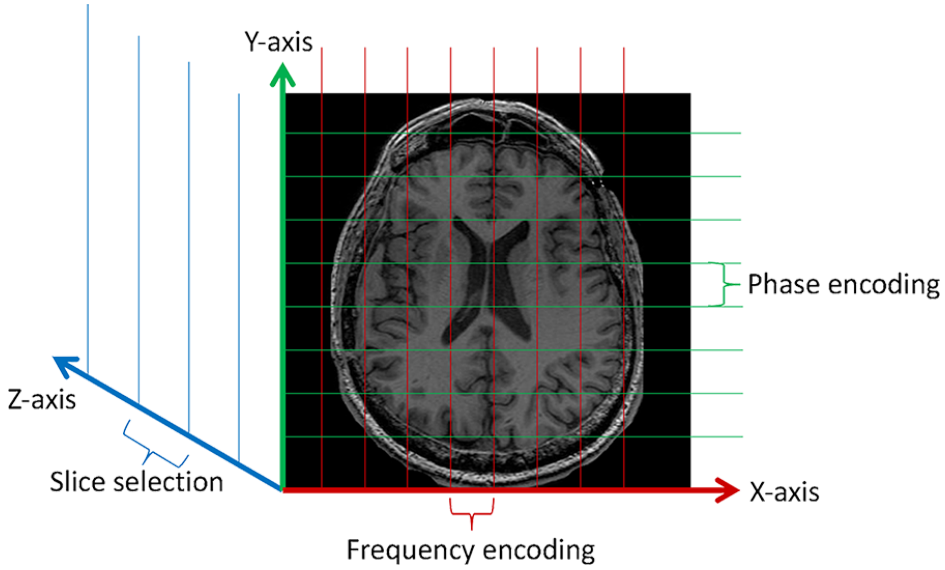


Figure 2.1.: Gradients for slice selection, frequency encoding and phase encoding are utilized to locate signals in 3D space. [9]

1. Slice selection along z-axis (G_{ss} gradient)
2. Segment of slice along x-axis selected by frequency encoding (G_{fe} gradient)
3. Part of segment along y-axis selected by phase encoding (G_{pe} gradient)

Slice selection

If the slice selection gradient G_{ss} is applied along the z-axis to detect a slice, the effective magnetic field along z is

$$B(z) = B_0 + G_{ss}z \tag{2.4}$$

The resonance frequency scales linearly with position z and is determined by (2.1)

$$\omega(z) = \gamma B(z) = \gamma(B_0 + G_{ss}z) \tag{2.5}$$

A single slice is stimulated by applying an RF-pulse tuned specifically to the frequencies in that slice. Protons not in the slice will not get excited since their Larmor frequency will not match the frequency of the pulse, hence they won't efficiently receive energy from the pulse. Note that different slices are selected with corresponding center frequencies of G_{ss} during the RF-pulse.

As each slice has a finite width Δz and contains a small range of frequencies $\Delta\omega$, the relationship between slice thickness and gradient is inversely proportional as shown in

$$\Delta\omega = \gamma G_{ss}\Delta z \tag{2.6}$$

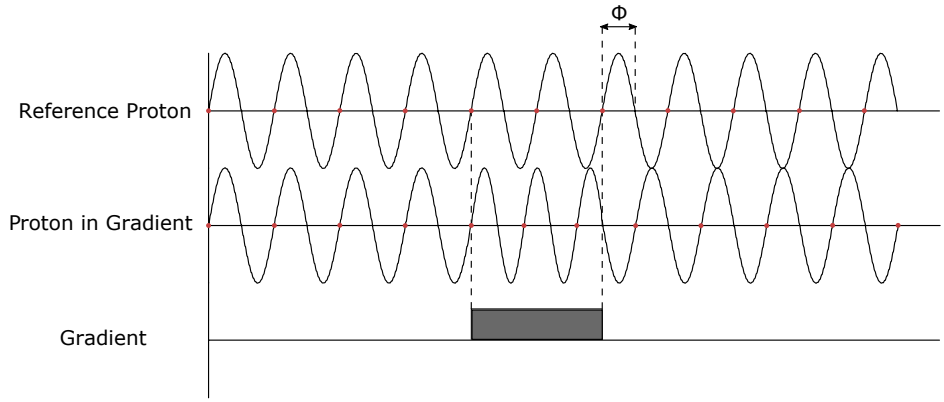


Figure 2.2.: MR pulse timing diagram with simplified rectangular gradient representation. The phase of the proton in the gradient is permanently shifted (Φ) after the gradient is applied, whereas the reference proton's phase does not shift.

Hence, the stronger the slice selection gradient the smaller the slice thickness.

To compensate for phase dispersion of transverse magnetization, caused by G_{ss}, a slice-rephasing lobe of opposite polarity compared to G_{ss} is used. This will force the hydrogen to rephase or precess in phase.

Frequency encoding

Frequency encoding is used to define location along one axis within a slice (xy-plane).

If a frequency-encoding gradient G_{fe} is applied along the x-axis, the effective magnetic field along x is

$$B(x) = B_0 + G_{fe}x \tag{2.7}$$

The resonance frequency scales linearly with position x and is determined by (2.1)

$$\omega(x) = \gamma B(x) = \gamma(B_0 + G_{fe}x) \tag{2.8}$$

For each frequency encoding step, a single aggregate MR signal is recorded reflecting contributions from each of the voxels within a slice. Hence if at least 256 frequency-encoding steps are performed, 256 MR signals can be recorded from the slice.

The frequency-encoding gradient strength represents a trade-off between sampling time and signal-to-noise ratio. G_{fe} is applied during readout to change the Larmor frequencies of the nuclei along the x-axis. This way columns within a slice can be distinguished by frequency. Spatial detection in the last dimension will require a different method, phase-encoding.

Phase encoding

Frequency-encoding gradients and phase-encoding gradients work in the same way, but are used for different purposes. All gradients temporarily change the resonant frequencies of protons while the gradient is being applied. The protons go back to their initial precession frequencies, when the gradient is turned off. As demonstrated in figure 2.2, these protons have gained memory of their historical encounter with the gradient by a permanent phase shift Φ. The phase shift is proportional to the strength of the gradient (G_p) and time that gradient is applied (t). [10]

$$\Phi = \gamma G_p t \tag{2.9}$$

If two phase shifted sine waves with the same frequency interfere, the resulting sine wave retains its frequency but changes in amplitude, depending on the phase shift. It is not possible to differentiate the location of these signals by a single observation. In fact, the number of phase encoding steps equals the number of voxels along the phase encoding axis. For each phase encoding step, a single aggregate MR signal is recorded reflecting contributions from each of the voxels within a slice. Hence if at least 256 phase-encoding steps are performed, 256 MR signals can be recorded from the slice. [11]

While frequency encoding only takes a few milliseconds of signal reading, phase encoding involves repeating the imaging sequence. In a spin echo sequence, a single phase encoding step is performed during each repetition time (TR). As TR values can be of up to 3 seconds, phase encoding is therefore much slower than frequency encoding. All the signals (transversal net magnetization) from the same slice are directly recorded in k-space.

2.1.4. k-space

The 2D Fourier transform is defined by the following equation:

$$F(k_x, k_y) = \iint f(x, y) \mathrm{e}^{-2\pi i(k_x x + k_y y)} dx dy \tag{2.10}$$

The MR signal is recorded in quadrature, so each k-space point contains real and imaginary component. The most popular method for filling k-space over the last 30 years has been the line-by-line Cartesian method.

k-space is an array of complex numbers. The cells of k-space are ordered on a rectangular grid, which has the same dimensions as the final image. The axes on this grid k_x and k_y represent spatial frequencies in x- and y-direction. The complex number in each cell contains information about every pixel in the final image for the corresponding spatial frequency. Hence, each pixel in the final image maps to every point in k-space.

To transform fully sampled k-space data to image space, the 2D discrete inverse Fourier transform is applied to data. The following formula defines this transform X of an m-by-n matrix Y:

$$X_{p,q} = \frac{1}{mn} \sum_{j=1}^{m} \sum_{k=1}^{n} \omega_m^{(j-1)(p-1)} \omega_n^{(k-1)(q-1)} Y_{j,k} \tag{2.11}$$

where i is the imaginary unit, p runs from 1 to m, q runs from 1 to n and ω_m and ω_n are complex roots of unity

$$\omega_m = \mathrm{e}^{2\pi i/m} \tag{2.12}$$

$$\omega_n = \mathrm{e}^{2\pi i/n} \tag{2.13}$$

2.2. Compressed Sensing

Compressed sensing is a signal processing technique for recovering a signal from an incomplete measurement by solving a non-linear inverse problem.

MRI acquisitions can be modified to take advantage of the CS paradigm, where each sample represents a Fourier coefficient in k-space. Instead of measuring the whole k-space, a small subset is sufficient to make accurate reconstructions. The three following requirements need to be fulfilled for successful CS applications:

Transform Sparsity

The desired image must be sparse in a known transform domain. Depending on the image data, different transform domains are applicable to achieve sparsity. If we take a look at angiograms, pixel space is already sparse, because these images contain primarily contrast-enhanced blood vessels with a few high and lots of low pixel intensity values [12]. This does not hold for other MR images, where the ℓ_1-norm of wavelet coefficients or total variation are utilized.

Incoherence of Undersampling Artifacts

Undersampling describes the process of measuring only a portion of k-space. For a successful image reconstruction, the resulting aliasing artifacts should be noise-like in the domain in which the image is sparse.

If the undersampling is done equispaced (e.g. sampling every second line in k-space) the reconstruction generates a superposition of shifted signal copies resulting in unwanted coherent aliasing. If samples are picked randomly, the zero-filled Fourier reconstruction shows incoherent artifacts that resemble additive noise. [12]

Variable-density sampling schemes for Cartesian, radial, and spiral imaging have been proposed in the past, because the aliasing appears incoherent [12].

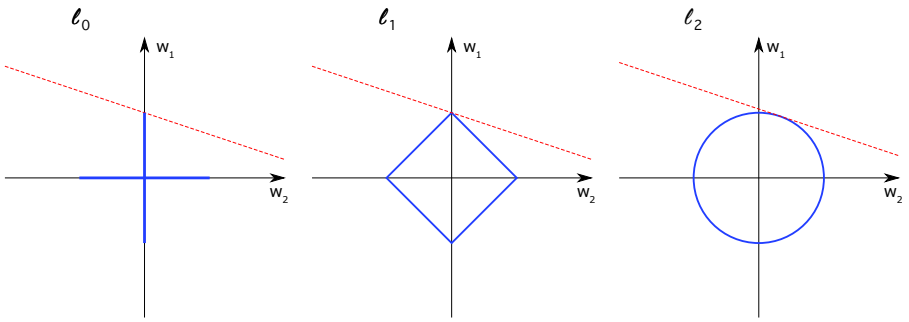


Figure 2.3.: Comparison between ℓ_0, ℓ_1 and ℓ_2 norm for a minimization problem in the form of min $\|\boldsymbol{w}\|$ such that $\boldsymbol{A}\boldsymbol{w} = \boldsymbol{y}$. Blue points indicate constant values of $\|\boldsymbol{w}\|$ for the corresponding norm. Red points are solutions of $\boldsymbol{A}\boldsymbol{w} = \boldsymbol{y}$. Note that the red line $\boldsymbol{y}$ intersects the ℓ_2 circle at a non-sparse point, while ℓ_0 and ℓ_1 both enforce w_2 to be 0.

While the periphery of k-space contains high spatial frequency information which are important for edges, details and sharp transitions, the center of k-space carries important basic contrast information. Sampling should not be completely random in order to avoid missing important information. Note that most information about the image is contained in the center of k-space. Therefore, it is common to use a fully sampled calibration are in the center of k-space.

Nonlinear Reconstruction

The last condition that needs to be fulfilled for CS is a non-linear method for image reconstruction, which enforces both sparsity of the image representation and consistency of the reconstruction with the acquired samples. [12]

A sensible sparsity constraint is the ℓ_0 norm $\|\boldsymbol{w}\|_0$, which is defined as the number of non-zero elements of a vector $\boldsymbol{w}$. It has been demonstrated that solving ℓ_0 regularized problems is NP-hard [13]. As illustrated in the two dimensional example of figure 2.3 the ℓ_1 norm can be used to approximate the optimal ℓ_0 norm via convex relaxation.

Let $\boldsymbol{x}$ denote an image of interest, which is represented by a complex vector. The sparsifying transform and undersampled Fourier transform are denoted by $\boldsymbol{\Psi}$ and $\boldsymbol{\mathcal{F}_u}$ respectively. To promote sparsity of $\boldsymbol{x}$ in a sparse domain, $\|\boldsymbol{\Psi x}\|_1$ needs to be minimized. This term makes sure that solutions of x are compressible in $\boldsymbol{\Psi}$. To enforce data consistency, $\|\boldsymbol{\mathcal{F}_u x} - \boldsymbol{y}\|_1^2$ must be minimized as well.

This term ensures that undersampled Fourier transform of $\boldsymbol{x}$ match the acquired k-space data. This type of variable selection including ℓ_1 regularization is used in regression analysis and known as LASSO:

$$\underset{x}{\operatorname{argmin}} \frac{1}{2} \|\boldsymbol{\mathcal{F}_u x} - \boldsymbol{y}\|_2^2 + \lambda_w \|\boldsymbol{\Psi x}\|_1 \tag{2.14}$$

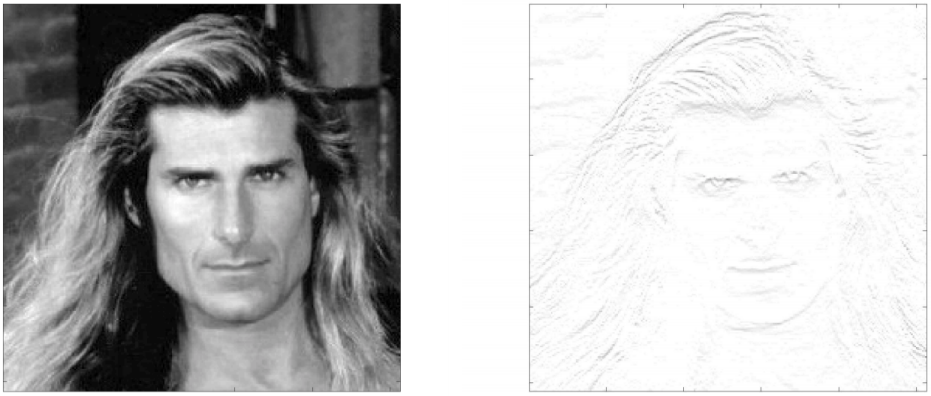


Figure 2.4.: The original Fabio image (left) and the absolute values after application of a discrete gradient operator(right) [14].

where λ_w denotes a weighting factor for the trade-off between data consistency and sparsity of $\boldsymbol{x}$ in $\boldsymbol{\Psi}$.

Many natural images admit very sparse gradients, as shown in figure 2.4 [14]. Total variation denoising is most often used in digital image processing. It removes unwanted detail whilst preserving important details such as edges. The total variation seminorm

$$\|\nabla \boldsymbol{x}\|_1 =: \|\boldsymbol{x}\|_{TV} \tag{2.15}$$

can be added to (2.14) resulting in

$$\underset{x}{\operatorname{argmin}} \frac{1}{2} \|\boldsymbol{\mathcal{F}_u x} - \boldsymbol{y}\|_2^2 + \lambda_w \|\boldsymbol{\Psi x}\|_1 + \lambda_v \|\boldsymbol{x}\|_{TV} \tag{2.16}$$

where λ_v denotes one additional weighting factor. If this factor is tuned high, solutions of x tend to be smoother, but may lose detail due to forced reduction of total variation.

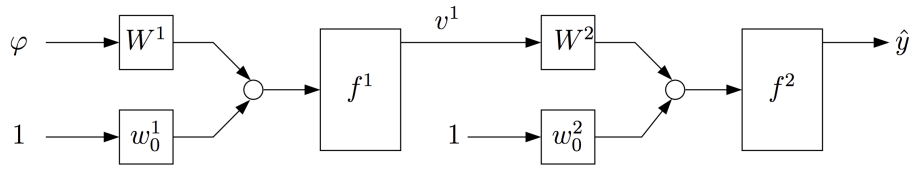


Figure 2.5.: Multilayer perceptron network [15].

2.3. Artificial Neural Networks

Artificial Neural Networks (ANNs) are computing systems that map data inputs $\boldsymbol{\varphi}$ to data outputs $\hat{\boldsymbol{y}}$. ANNs are commonly used for classification and regression problems. Many different network architectures exist. TensorFlow 1.8.0 is utilized for neural network implementation.

2.3.1. Basics

Feedforward propagation

In a trained Neural Network data inputs are propageted forward layer by layer to generate ouput. This is called feedforward, because the network's graph is a directed acyclic graph. Other types like Recurrent Networks with cycles also exist, but are not discussed in this thesis.

In 2.5 we see an example of a multilayer perceptron feedforward network with one hidden layer and one output layer, where the upper index denotes to the corresponding layer. The output of the hidden layer is determined by

$$\boldsymbol{v}^1(\boldsymbol{\varphi}) = f^1(\boldsymbol{W}^1\boldsymbol{\varphi} + \boldsymbol{w}_0^1) \tag{2.17}$$

f^1 is the activation function, $\boldsymbol{W}^1$ is a weight matrix and $\boldsymbol{w}_0^1$ is a bias vector. The number of rows of $\boldsymbol{W}$ and $\boldsymbol{w_0}$ match the chosen number of neurons for each layer. The higher the number of neurons, the higher the number of weights and biases and with more network parameters the network gets more flexibility to fit the data. Using too many neurons is computationally more expensive and can lead to overfitting [16].

The output generated by the network in dependence of the data input is [15]

$$\hat{\boldsymbol{y}}(\boldsymbol{\varphi}) = f^2(\boldsymbol{W}^2 f^1(\boldsymbol{W}^1\boldsymbol{\varphi} + \boldsymbol{w}_0^1) + \boldsymbol{w}_0^2) \tag{2.18}$$

Activation functions

Activation functions are scalar-to-scalar functions used for limiting the amplitude of the output of a neuron. They are sometimes called threshold function or transfer function. [17]

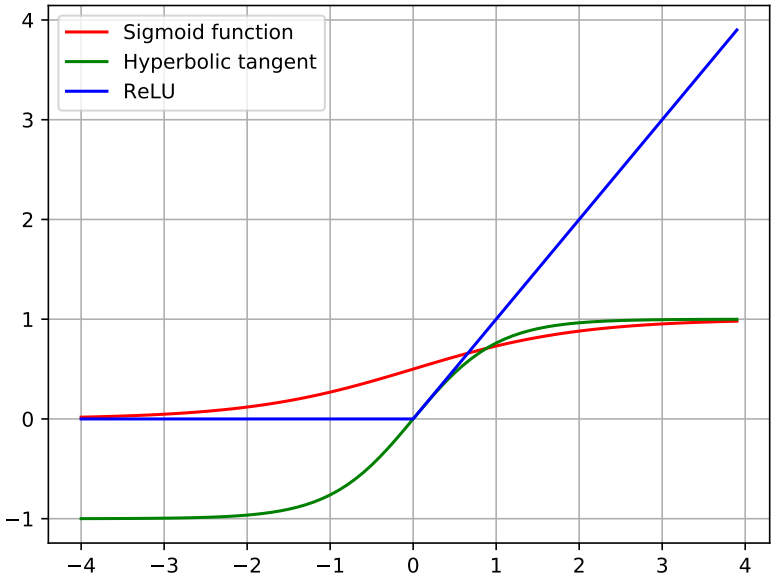


Figure 2.6.: Three different activation functions

There are many different types of activation functions. Sigmoid, hyperbolic tangent and ReLU are the most common and are shown in figure 2.6.

If we substitute the two activation functions from equation (2.18) with linear functions, the output of the network can be represented as a linear combination of inputs. Hence, without nonlinear activation functions, any feed forward neural network with an arbitrary number of hidden layers and neurons can be represented without a hidden layer, because all outputs are linear combinations of inputs.

Nonlinear activation functions are the key element of a neural network to model nonlinear behaviour. It is shown, that a two layer network with an arbitrary squashing activation function acts as a universal approximator to any desired degree of accuracy, provided sufficiently many neurons [18].

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{2.19}$$

Sigmoid and hyperbolic tangent are squashing functions, which map inputs ranging from $-\infty$ to $+\infty$ to outputs of continuous values of either 0 to 1 or -1 to 1 respectively. The choice between these activation functions depends on the range of the data input of the corresponding layer.

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.20}$$

Neural networks using sigmoidal activation functions can suffer from the vanishing gradient problem. Vanishing gradients occur when lower layers have gradients of nearly 0 because higher layer units are nearly saturated at the asymptotes of the activation function. This leads to slow optimization performance and the trained network usually converges to a poor local minimum. Rectifiers like ReLU offer an alternative to sigmoidal activation functions which do not suffer from the problem mentioned above. [19]

$$ReLU(x) = max(0, x) \tag{2.21}$$

2.3.2. Training

There are two modes ANNs are used. In recall-mode the network is already trained and ready for feedforward propagation of unseen data for output generation. In training-mode a data-set with given inputs and outputs is used to train the network. Gradient descent based algorithms are used to iteratively improve the solution.

Objective

Training is achieved by minimizing a cost function V with respect to the network parameters $\boldsymbol{\theta}$, which are the weights and biases over all n data samples of the data-set $\boldsymbol{Z}$. The most common used metric is the mean squared error. The error is the difference of the prediction $\hat{\boldsymbol{y}}$ with respect to the known output $\boldsymbol{y}$.

$$V(\boldsymbol{Z}, \boldsymbol{\theta}) = \frac{1}{2n} \sum_{k=1}^{n} (y_k - \hat{y}_k(\boldsymbol{Z}, \boldsymbol{\theta}))^2 \tag{2.22}$$

$$\hat{\boldsymbol{\theta}} = \min_{\theta} V(\boldsymbol{Z}, \boldsymbol{\theta}) \tag{2.23}$$

At each training step l a mini-batch of training samples is used to adapt the network parameter $\boldsymbol{\theta}$ until the training is stopped. There are many different training methods which differ in the way $\boldsymbol{\theta}$ is updated at each step.

Steepest descent

In the neighbourhood of a given value $\boldsymbol{\theta_0}$ of the parameter vector, the cost function can be developed into a Taylor series. One can distinguish between training methods that only use V and ∇V for updating the esitmate and Newton methods that use V, ∇V and $\nabla^2 V$. Newton methods are not discussed in this thesis (to improve readability, the argument $\boldsymbol{Z}$ of V has been dropped).[15]

$$V(\boldsymbol{\theta}) = V(\boldsymbol{\theta}^0) + \nabla V^T(\boldsymbol{\theta})\big|_{\boldsymbol{\theta}^0}(\boldsymbol{\theta} - \boldsymbol{\theta}^0) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}^0)^T \nabla^2 V(\boldsymbol{\theta})\big|_{\boldsymbol{\theta}^0}(\boldsymbol{\theta} - \boldsymbol{\theta}^0) + \dots \tag{2.24}$$

Note that the gradient of the cost function can be rewritten as

$$\nabla V(\boldsymbol{\theta}) = \frac{1}{n}\boldsymbol{J}^T(\boldsymbol{\theta})\boldsymbol{E}(\boldsymbol{\theta}) \tag{2.25}$$

where $\boldsymbol{E}$ denotes the vector of prediction errors. The Jacobian contains the partial derivatives of the prediction errors with respect to $\boldsymbol{\theta}$.

Steepest descent is a method to train a neural network. The update equation is shown in equation (2.26). The learning rate and the search direction are denoted α and f respectively. The Steepest descent is obtained in the direction of the negative gradient of the first order approximation of the cost function of equation (2.24).

$$\boldsymbol{\theta}_{l+1} = \boldsymbol{\theta}_l + \alpha f_l = \boldsymbol{\theta}_l - \alpha \nabla V(\boldsymbol{\theta}_l) = \boldsymbol{\theta}_l - \alpha \frac{1}{n}\boldsymbol{J}^T(\boldsymbol{\theta})\boldsymbol{E}(\boldsymbol{\theta}) \tag{2.26}$$

Note that the learning rate, which resembles the step size, does not contain the index l, thus it stays fixed during the whole training process. There are learning rate schedule techniques, that try to adjust the learning rate during training by e.g. annealing, which reduces the learning rate according to a predefined schedule. These schedules have to be defined in advance and are unable to adapt to a data-sets characteristics.

RMSProp

Root Mean Square Propagation (RMSProp) is an unpublished training algorithm by Geoffrey Hinton [20]. Even though adaptive gradients methods like RMSProp lack theoretical justifications in the non-convex optimization setting [21], it outperformed other adaptive methods like Adagrad, Adadelta as well as SGD with momentum in a large number of tests [22].

RMSProp divides the gradient by the square root of an exponentially decaying moving average of entrywise squared gradients [20]. During training at each step l the decaying moving average of previous gradients $\boldsymbol{\eta}$ is accumulated by the following equation separately for each network parameter $\boldsymbol{\theta}_i$ (to improve readability, the argument $\boldsymbol{\theta}$ of V has been dropped)

$$\eta_{i,l} = \lambda \eta_{i,l-1} + (1-\lambda)\left(\frac{\partial V}{\partial \theta_{i,l}}\right)^2 \tag{2.27}$$

where λ resembles the decay constant. Hinton suggests $\lambda = 0.9$ which is also the default value for the TensorFlow implementation [20][23]. At the first step $\eta_{i,0}$ is initialized to 0.

The update equation for each of the network parameter is

$$\theta_{i,l+1} = \theta_{i,l} - \frac{\alpha}{\sqrt{\eta_{i,l} + \epsilon}}\frac{\partial V}{\partial \theta_{i,l}} \tag{2.28}$$

where ϵ denotes a small constant value to avoid division by zero. The decaying moving average ensures that learning continues to make progress even after many steps of network parameter updates have been done. Entrywise division by the RMS improves learning [20]. Each parameter has it's own scaled learning rate to reduce oscillation during training. The exponential average in the denominator helps weighting the more recent gradient updates more than the less recent ones.

Momentum Method

The momentum method, which is often referred to classical momentum (compared to Nesterov Accelerated Gradient), is a technique for accelerating gradient descent that accumulates a velocity vector in directions of persistent reduction in the objective across iterations. [24]

The velocity $\boldsymbol{\nu}$ is calculated at each step by

$$\boldsymbol{\nu}_l = \mu \boldsymbol{\nu}_{l-1} - \alpha \nabla V(\boldsymbol{\theta}_l) \tag{2.29}$$

where $\mu \in [0, 1]$ is the momentum coefficient, which can be interpreted as the decay of velocity.

The update function for the network parameter is given by

$$\boldsymbol{\theta}_{l+1} = \boldsymbol{\theta}_l + \boldsymbol{\nu}_l \tag{2.30}$$

The change of network parameter equals the velocity. Note, that if μ is set to 0, equation (2.30) becomes the steepest descent update (2.26). This method can be adapted to other learning methods like RMSProp by changing the second term of equation (2.29) to the corresponding network parameter update. The velocity term increases for dimensions whose partial derivatives point in the same directions and reduces updates for dimensions whose partial derivatives change directions. The result of this procedure is faster convergence by building up speed with consistent gradients and damped oscillation in directions of high curvature by combining gradients with opposite signs [20].

Dropout

Dropout is a regularization technique to reduce overfitting and provides a way of approximately combining exponentially many different neural network architectures efficiently. The key idea is to randomly drop units along with their incoming and outgoing connections during training. Training with dropout can be seen as training a collection of 2^n thinned networks with extensive weight sharing, where each thinned network gets trained very rarely, if at all. [25]

The only hyperparameter to be set for dropout is the keep probability $p_{keep} \in [0, 1]$, which is the probability that a unit is kept during training. This parameter controls the intensity of

(a) Standard Neural Network **(b)** After applying dropout

Figure 2.7.: Left: Neural network with one hidden layer **Right:** Example of a thinned neural network produced by applying dropout with $p_{keep} = 0.5$ to the network on the left. Crossed out units and their connections are dropped.

dropout. $p_{keep} = 1$ implies no dropout and vice versa. Smaller p_{keep} requires a big number of weights, which slows down the training and leads to underfitting. Large p_{keep} may not produce enough dropout to prevent overfitting. [25]

In recall-mode, it is not feasible to average the predictions from many thinned models. This can simply be approximated by one network, where all units are kept and each weight is multiplied by p_{keep} [25]. Note, that in the equivalent TensorFlow implementation of dropout, the output of each kept unit is multiplied by the inverse of p_{keep} during training [26]. This way, multiplying each weight by p_{keep} becomes redundant in recall-mode.

It is worth mentioning that, if a standard neural network is optimal with n units on any given task, a good neural network using dropout should have at least $\frac{n}{p_{keep}}$ units, to prevent a reduction in capacity of the network during learning [25].

Early Stopping

So far, many techniques for training of neural networks were discussed. The last topic to mention is when to stop training. A sufficiently big neural network could be trained for a given task until the training-set fits the data perfectly, but the performance of unseen test-data would be poor (overfitting). Stopping the training too soon on the other hand leads to underfitting, so that the model does not fit the training data as well as it could. The goal is to stop training at the right time, where the network generalizes best (performance of unseen data). The optimal amount of training steps needed to achieve this depends on many factors and varies from task to task.

Early stopping if a form of regularization, which is used to automatically stop training with some form of stopping criterion to prevent overfitting. There are many different variations of early stopping with different stopping criterions. The simplest form of early stopping is dividing the data into a training and a validation set. After each training step (only training-set

is used for training) the prediction error for the validation-set is evaluated. If the validation error starts increasing, training stops. [27]

A problem with this method is the oscillation in the validation error, which still might improve significantly after a number of training steps even though it increased once. To overcome this problem *patience* $\in \mathbb{N}$ is introduced, which describes the maximum number of consecutive training steps, where no improvement of the validation error is observed [28]. If that number is reached, training stops. The network used for recall-mode is not the most recent one, but the one with the lowest validation error (*patience* steps back from the most recent network).

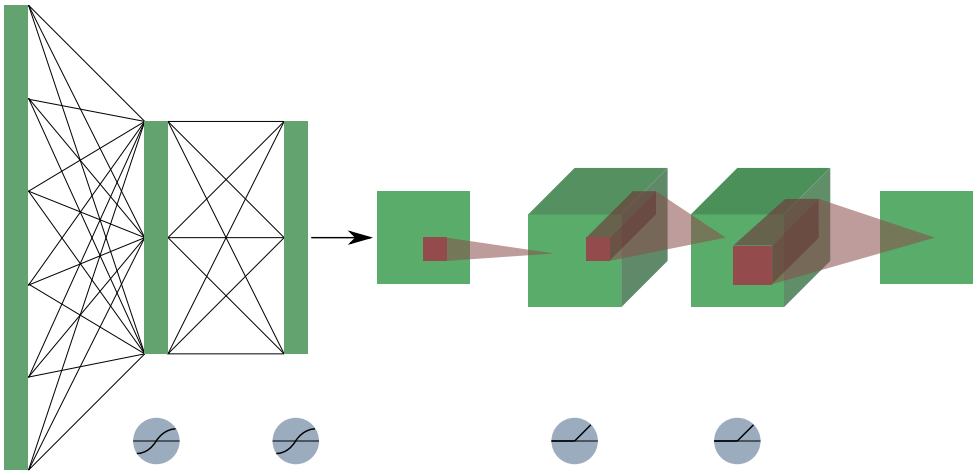
Algorithm 2.1 Pseudocode for early stopping

Let p be the patience
Let θ_0 be the initial network parameters
$\hat{v} \leftarrow \infty$
$\theta \leftarrow \theta_0$
$\hat{\theta} \leftarrow \theta$
$i \leftarrow 0$
while $i < p$ **do**
  Update θ by running the training algorithm
  $v \leftarrow$ ValidationError(θ)
  if $v < \hat{v}$ **then**
    $\hat{v} \leftarrow v$
    $\hat{\theta} \leftarrow \theta$
    $i \leftarrow 0$
  else
    $i \leftarrow i + 1$
  end if
end while
Best network parameters are $\hat{\theta}$
Best performance on validation-set is $\hat{v}$

2.3.3. AUTOMAP

Automated transform by manifold approximation (AUTOMAP) is a unified framework for image reconstruction. AUTOMAP maps data from the sensor to the image domain in a data-driven supervised learning task, which can be implemented with a deep neural network feed-forward architecture. The network consists of two fully connected layers (FC 1, FC 2), which approximate the between-manifold projection from the sensor domain to the image domain and three convolutional layers (Conv 1, Conv 2, Deconv). [29]

The fully connected layers approximate the between-manifold projection from the sensor domain to the image domain. The convolutional layers extract high-level features from the



Layer	Input	FC 1	FC 2		Conv 1	Conv 2	Deconv
Output dimension	$2n^2$ x 1	n^2 x 1	n^2 x 1	n x n	n x n x 64	n x n x 64	n x n
Kernel size					5 x 5	5 x 5	7 x 7
Activation function		tanh	tanh		ReLU	ReLU	

Figure 2.8.: Graphical representation of AUTOMAP neural network layers

data and force the image to be represented sparsely in the convolutional-feature space [29]. These features are important for learning abstract image characteristics of the dataset.

Network architecture

For a target image resolution of $n \times n$ pixels, the input layer of the network consists of $2n^2$ neurons. Each of these neurons contain real-valued data, gathered by separation of real and imaginary parts from complex sensor domain data of dimension n^2. The fully connected layers FC 1 and FC 2 both consist of n^2 neurons and use hyperbolic tangent as activation functions. [29]

Conv 1 and Conv 2 convolve the output of the previous layers with 64 filters. Both layers use kernels of size 5×5 with stride 1 followed by ReLU activation function. Deconv uses one filter with a kernel size of 7×7 with stride 1. Zero padding is used for all convolutional layers to preserve dimensions between input and output. [29]

A graphical representation of the neural network architecture is shown in figure 2.8.

Training details

One percent multiplicative noise is added to the data input during training to improve generalization. This forces the network to learn robust representations from corrupted inputs. The RMPSProp algorithm with a learning rate of $2 \cdot 10^{-5}$, momentum of 0 and a decay rate of 0.9 is utilized. To promote sparse convolutional representations, an additional ℓ_1-norm penalty

with $\lambda = 10^{-4}$ is applied to Conv 2. Note that these parameters are adopted as proposed. [29] The batch size for each training batch is chosen to be 32.

2.4. Evaluation

To compare performance between different neural networks, objective performance measures need to be introduced. The mean squared error (MSE), the peak signal to noise ratio (PSNR) and the structural similarity (SSIM) are utilized. The mean squared error is used for measuring the prediction errors during the training process and is defined in the following equation.

$$MSE(y, \hat{y}) = \frac{1}{n} \sum_{k=1}^{n} (y_k - \hat{y}_k)^2 \tag{2.31}$$

Note that the value range for the errors is 2 throughout this thesis (network output ranges from -1 to 1). This applies only to the MSE.

The PNSR approaches infinity as the MSE approaches zero, as shown in the next equation

$$PSNR(y, \hat{y}) = 10 \log_{10} \left(\frac{2^B - 1}{MSE} \right) \tag{2.32}$$

where B denotes the bit depth of the images, which is 8 in this thesis.

Complementary to the approach of MSE and PSNR which are based on error sensitivity, the SSIM aims to assess perceptual image quality. Natural image signals are highly structured and carry important information about the structure of the objects in the visual scene. Especially spatially proximate pixels exhibit strong dependencies, which are not observed by MSE or PSNR as they are independent of the underlying signal structure and consider only pointwise signal differences. [30]

The SSIM is defined as the product of three independent components: luminance, contrast and structure

$$SSIM(x, y) = l^{\alpha}(x, y) \cdot c^{\beta}(x, y) \cdot s^{\gamma}(x, y) \tag{2.33}$$

where the exponents α, β and γ denote weighting factors for the corresponding components:

$$l(x, y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \tag{2.34}$$

$$c(x, y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \tag{2.35}$$

$$s(x, y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3} \tag{2.36}$$

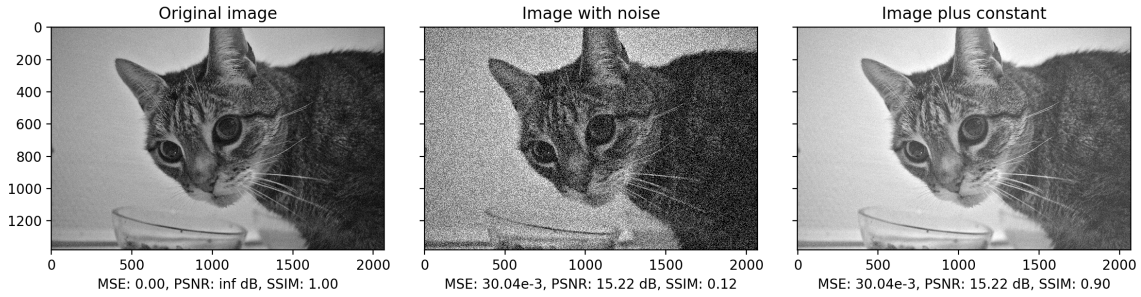


Figure 2.9.: Example images with equal MSE and different SSIM. **Left:** Original image **Middle:** Image with noise **Right:** Image with added constant, which is chosen to match MSE of the image with noise. In this case the SSIM represents the perceived quality much better than the MSE.

These weighting factors are set to $\alpha = \beta = \gamma = 1$ and $C_3 = \frac{C_2}{2}$ resulting in a simplified representation of the SSIM:

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{2.37}$$

with

$$C_1 = (K_1 L)^2 \tag{2.38}$$

$$C_2 = (K_2 L)^2 \tag{2.39}$$

where $K_1 =$ and $K_2 = 0.03$ are small constants to avoid division by a weak denominator in equation (2.37). These constants are proposed by Wang, and are shown to be insensitive to variation regarding the SSIM [30]. μ and σ^2 donate the arithmetic mean and variance. The covariance between x and y is represented by σ_{xy}. L represents the bit depth of the images.

Note, that x and y are positions of local $n \times n$ windows which move pixel by pixel over the images to calculate the local statistics (mean, variance and covariance), resulting in one SSIM value for each position. The mean of these values denotes the desired perceived image quality and will be termed SSIM for convenience.

The chosen size of n will alter the resulting SSIM. Many SSIM implementations use a normalized Gaussian kernel of width $\sigma = 1.5$ which is also utilized in this thesis. The skimage library is used to calculate the SSIM which truncates the weighted window at size 11 [31].

SSIM values vary between -1 and 1, where 1 indicates perfect similarity and can only be reached with identical images. A value of 0 indicates no structural similarity. If the local image structures are inverted the covariance becomes negative resulting in a negative SSIM value.

3
Experimental Setup

The whole process, starting from raw image data and ending in the reconstruction of images, is described in this chapter. A simplified representation of the signal processing chain is shown in figure 3.1.

Four experiments are conducted. An overview is shown in table 3.1. In the first experiment different input data preprocessing approaches are performed and evaluated without undersampling. The neural network is trained with default AUTOMAP hyperparamters (described in section 2.3.3) as proposed in the Nature paper [29]. The preprocessing technique with the best performance is utilized for the following experiments. The performance of different sampling masks is evaluated in the second experiment with the same default AUTOMAP hyperparameters. AUTOMAP's performance is evaluated with different hyperparameters in the third experiment. The preprocessed input data is undersampled with the best performing mask of the previous experiment. In the last experiment the reconstruction performance between CS and AUTOMAP is compared.

The first three experiments are evaluated with the MSE between groundtruth and reconstruction. The last experiment utilizes the the SSIM.

Experiment	**Preprocessing**	**Undersampling**	**Hyperparameter**
E_1 - Input data preprocessing	Evaluation	None	Default
E_2 - Undersampling masks	Result from E_1	Evaluation	Default
E_3 - Hyperparameter tuning	Result from E_1	Result from E_2	Evaluation
E_4 - CS comparison	Result from E_1	Result from E_2	Result from E_3

Table 3.1.: Overview of performed experiments.

Data preparation

Data preparation plays a significant role in terms of image reconstruction performance. Processing steps from raw image data to ground truth and different approaches to prepare the input data for the neural network are shown in this section.

The raw image domain data-set $\boldsymbol{Z}_{im,raw}$ containing generic images is acquired from ImageNet [32]. Images with width or length lower than 64 pixels are omitted, resulting in 7137 images

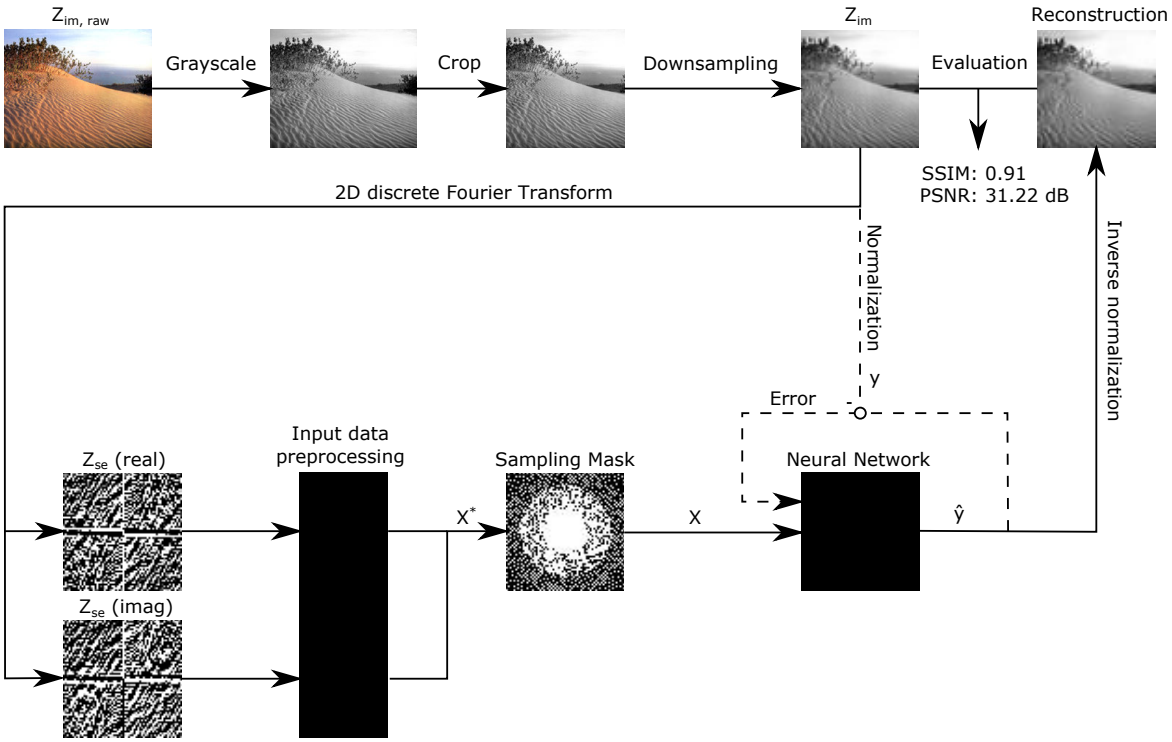


Figure 3.1.: Signal processing chain from raw image data to final image reconstruction. The dashed line describes data flow that only appears during neural network training.

from 17 different categories. To augment the data-set, each image is rotated by steps of 90 degrees. The augmented data-set of 28548 images is divided into a training-set (70%), validation-set (15%) and test-set (15%).

Each image of the data-set is converted to an 8-bit grayscale intensity image by Y-channel luminance extraction of the RGB colours. The ground truth image resolution for this thesis is 64×64 pixels as a result of hardware limitations. Cropping each image around the center to a 64×64 pixel image leads to some unrecognizable images. The higher the resolution of the original image, the smaller the observable image section of the cropped image. To compensate this problem, each image is cropped around the center to an $n \times n$ resolution, where $n = min(width, length)$, followed by downscaling to 64×64 pixels by using a high-quality anti aliasing downsampling filter (Python Imaging Library). The preprocessed dataset (image domain), acquired by the steps mentioned above, is denoted $\boldsymbol{Z}_{im}$.

Normalization with $min = -1$ and $max = 1$ is applied to $\boldsymbol{Z}_{im}$ by use of equation (3.3), resulting in the ground truth data-set $\boldsymbol{y}$. This data-set is used to estimate the prediction error during the training process.

The fast 2-dimensional discrete Fourier Transform is applied to all images of $\boldsymbol{Z}_{im}$, resulting in a data-set $\boldsymbol{Z}_{se}$ of complex values. This data-set can be used to fit a model, but leads to bad performance due to vast (and varying) value ranges between the center and periphery of k-space data. Thus, normalization needs to be applied to the data.

3.1. Input data preprocessing

There are many different techniques to prepare the normalized fully sampled data-set $\boldsymbol{X}^*$ for the given sensor-domain data-set $\boldsymbol{Z}_{se}$. In this section four binary options for the input data preprocessing are presented from section 3.1.1 to section 3.1.4. All combinations are applied to $\boldsymbol{Z}_{se}$ and lead to 16 differently normalized data-sets that are evaluated without undersampling and default AUTMOMAP hyperparameters.

3.1.1. Log-transformation

Before normalizing the data, an optional logarithmic transformation is applied to the data. In other contexts it was shown that logarithmic normalization techniques reduce prediction errors in neural networks [33].

The transformation scales each complex variable of the data-set by a scaling factor $s \in [0, 1]$, which depends on it's magnitude. To squash the values closer to each other, values of high magnitude result in low scaling factors and vice versa. Note, that the transformed data-sets phase information are preserved. The transformation is defined

$$f_{log}(x) = x(|x| + 1)^{-1/e} \tag{3.1}$$

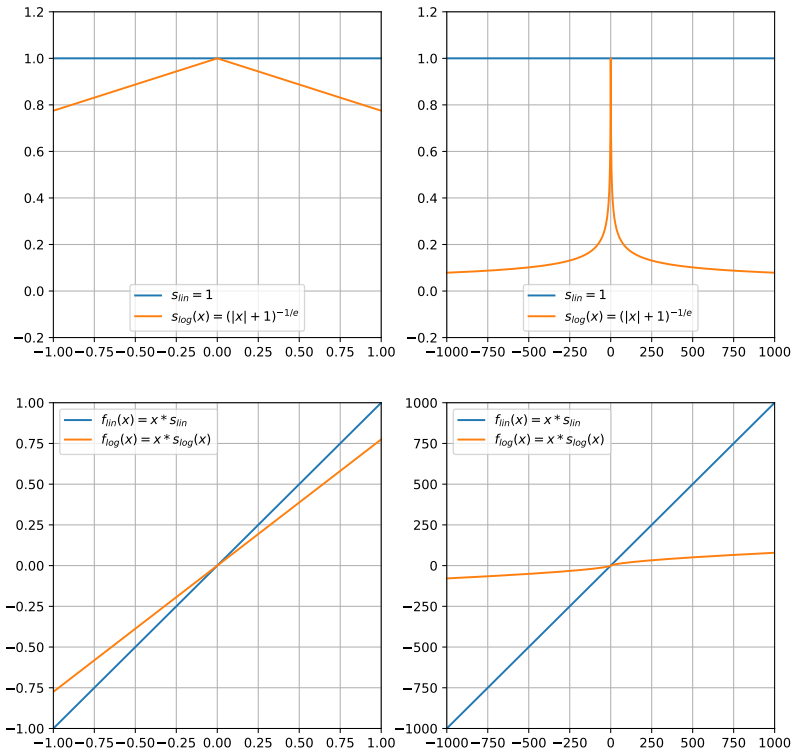


Figure 3.2.: Top row: Scaling factors s_{lin} and s_{log} in two different number ranges. **Bottom row:** Functions f_{lin} and f_{log} in two different number ranges. The bigger the absolute value of x, the smaller the scaling factor.

and shown in figure 3.2.

3.1.2. Normalization technique

There are three common ways to normalize data. One approach, which is often denoted as standardization, is to subtract the mean of the data-set from each sample and divide by the standard deviation of the data-set. This way, the resulting data-sets mean and standard deviation are 0 and 1 respectively.

$$Z_i^* = \frac{Z_i - \mu_Z}{\sigma_Z} \tag{3.2}$$

The other two common approaches are scaling the data to a maximum of 1 and a minimum of either 0 or -1. The first layer of the network uses the hyperbolic tangent activation function, ranging from -1 to 1. To make the data fit this scale, the minimum of -1 is used.

$$Z_i^* = \frac{2(Z_i - min(Z))}{max(Z) - min(Z)} - 1 \tag{3.3}$$

The data that normalization is applied to depends on the choice of normalization basis.

3.1.3. Normalization basis

For a data-set in image domain, there is no significant difference between the distributions of intensity values for different pixels. This is not true for the spatial frequencies in k-space. One approach is normalizing over the whole data-set, which results in different value ranges depending on the spatial frequency. For this method only two values (mean/std or min/max) must be acquired to transform unseen data. This matrix of normalization constants is denoted $\boldsymbol{C}_{img} \in \mathbb{R}^2$.

To compensate for the problem of varying ranges of input unit values, the normalization can be applied to every input unit separately. This can be interpreted as pixel-wise normalization, because every input unit corresponds to one specific pixel (spatial frequency) from the k-space data. To apply this form of normalization to unseen data, the matrix $\boldsymbol{C}_{pix} \in \mathbb{R}^{2x64x64}$ must be acquired.

3.1.4. Phase preservation

The data-set $\boldsymbol{Z}_{se}$ contains complex numbers. The normalization could be applied to the magnitude of the data-set, thus, scaling the real and imaginary part equally, without changing the phase. The value ranges for the real and imaginary parts can differ a lot, which can result in early saturation for the hidden units during training (vanishing gradient problem). To

resolve this issue, the real and imaginary parts can be scaled independently, resulting in a data-set of similar value ranges. Note, that the phase information is not preserved but also not lost. The phase is transformed by a non-linear transformation in dependence of the ratio of scaling-factors (real and imaginary).

3.2. Undersampling masks

Four different sampling masks are evaluated. These 64×64 pixels masks determine which complex k-space data points $\boldsymbol{X}^*$ from each sample will be used as input X for the neural network.

Random sampling with three different calibration areas ($0 \times 0, 4 \times 4$ and 8×8) in the center are evaluated. The calibration areas ensure that the essential low frequencies near the k-space center are acquired for image reconstruction.

The last approach is variable density Poisson-disk sampling which employs acquisition of data with preferential sampling near the k-space center.

In addition to the chosen reduction factor of 2.0, factors of 1.5 and 4.0 are also evaluated. These factors are the ratio of the number of pixels of the reconstruction images and the number of complex k-space data points used for reconstruction.

To compare performance between these masks, the MSE between groundtruth and reconstruction is evaluated. The input data for the neural network is normalized with the best performing set of normalization parameters from the previous experiment.

3.3. AUTOMAP hyperparameter tuning

The momentum method was not used in the original AUTOMAP paper [29]. We evaluate the performance of AUTOMAP with a momentum coefficient ranging from 0 to 0.8 in steps of 0.2. This variable coefficient leads to 5 different networks.

Noise on the training-set was used to simulate measurement noise and improve generalization. Using dropout acts as a regularization technique improving generalization as well. We evaluate the network performance for all possibilities regarding the use of noise and dropout. This leads to 4 different networks. The keep probability for dropout is set to a large value $p_{keep} = 0.99$. This value was heuristically found to work well and prevents significant training slow down.

Adjustment of the learning rate of RMSProp can improve performance. We evaluate three different learning rates: $2 \cdot 10^{-6}, 2 \cdot 10^{-5}, 2 \cdot 10^{-4}$.

All the options mentioned above lead to a total of 60 different combinations. Note that the input of the neural network is normalized with the best parameter found in the input data

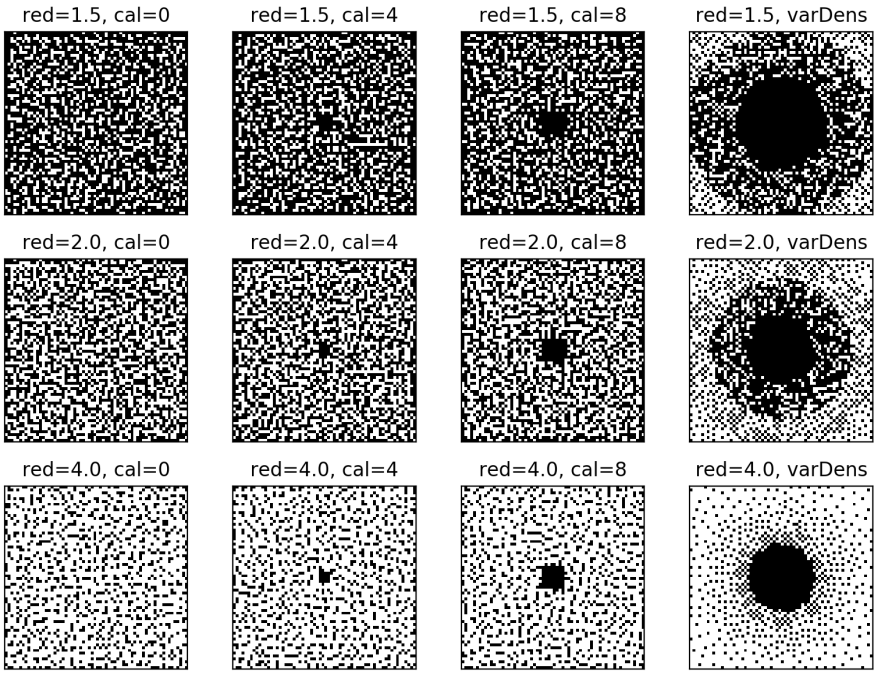


Figure 3.3.: Sampling masks where black pixels include and white pixels exclude k-space data samples.

preprocessing experiment. The data is undersampled with the best mask with a reduction factor of two found in the undersampling masks experiment.

3.4. Performance comparison

The last experiment is performed to compare the reconstruction performance between CS and AUTOMAP. The PSNR, MSE and SSIM are calculated to quantify the error between ground truth and reconstruction. One brain image and nine images from the test-set are used for reconstruction with CS. AUTOMAP's performance is measured with the whole test-set (4282 images).

CS reconstuctions are conducted by utilization of equation 2.16 with $\lambda_w = 0$ and $\lambda_v = 2.7 \cdot 10^{-3}$.

4 Results

The results of the experiments of the previous chapter are presented in this chapter.

4.1. Input data preprocessing

Table 4.1.: AUTOMAP MSEs with differently normalized data-sets. Default hyperparameter with learning rate of $2 \cdot 10^{-5}$, multiplicative noise on k-space data input, no dropout and no momentum are utilized. K-space input data is fully sampled.

Transform	Norm basis	Norm/Standard	Phase preservation	MSE
lin	$\boldsymbol{C}_{img}$	mean/std	abs	$6.261 \cdot 10^{-3}$
lin	$\boldsymbol{C}_{img}$	mean/std	real/imag	$6.365 \cdot 10^{-3}$
lin	$\boldsymbol{C}_{img}$	min/max	abs	$221.793 \cdot 10^{-3}$
lin	$\boldsymbol{C}_{img}$	min/max	real/imag	$110.581 \cdot 10^{-3}$
lin	$\boldsymbol{C}_{pix}$	mean/std	abs	$30.369 \cdot 10^{-3}$
lin	$\boldsymbol{C}_{pix}$	mean/std	real/imag	$20.397 \cdot 10^{-3}$
lin	$\boldsymbol{C}_{pix}$	min/max	abs	$3.938 \cdot 10^{-3}$
lin	$\boldsymbol{C}_{pix}$	min/max	real/imag	$3.941 \cdot 10^{-3}$
log	$\boldsymbol{C}_{img}$	mean/std	abs	$35.530 \cdot 10^{-3}$
log	$\boldsymbol{C}_{img}$	mean/std	real/imag	$39.550 \cdot 10^{-3}$
log	$\boldsymbol{C}_{img}$	min/max	abs	$144.208 \cdot 10^{-3}$
log	$\boldsymbol{C}_{img}$	min/max	real/imag	$58.437 \cdot 10^{-3}$
log	$\boldsymbol{C}_{pix}$	mean/std	abs	$55.099 \cdot 10^{-3}$
log	$\boldsymbol{C}_{pix}$	mean/std	real/imag	$40.629 \cdot 10^{-3}$
log	$\boldsymbol{C}_{pix}$	min/max	abs	$20.511 \cdot 10^{-3}$
log	$\boldsymbol{C}_{pix}$	min/max	real/imag	$19.492 \cdot 10^{-3}$

All results of differently normalized data-sets are shown in table 4.1. The logarithmic transform leads to worse performance regarding all other parameters with the exception of $\boldsymbol{C}_{img}$ with normalization (min/max) which results in MSEs of $144.208 \cdot 10^{-3}$ and $58.437 \cdot 10^{-3}$. Hence, only linear transformation is considered in the following paragraph.

Using $\boldsymbol{C}_{img}$ leads to small MSEs of $6.261{\cdot}10^{-3}$ and $6.365{\cdot}10^{-3}$ in combination with standardization (mean/std), but not with normalization (min/max). $\boldsymbol{C}_{pix}$ on the other hand leads to the best performances of the experiment of $3.941{\cdot}10^{-3}$ and $3.938{\cdot}10^{-3}$ in combination with normalization while the MSEs rose to over $20{\cdot}10^{-3}$ by standardization. The option of phase preservation impacts the four best performing cases only slightly (less than 2%). These cases show little advantage of scaling the real and imaginary data simultaneously.

4.2. Undersampling masks

Table 4.2.: AUTOMAP MSEs with different undersampling masks. Default hyperparameter with learning rate of $2 \cdot 10^{-5}$, multiplicative noise on k-space data input, no dropout and no momentum are utilized. The amount of k-space undersampling is determined by the reduction factor.

Reduction	Mask	MSE
1.5	cal0	$16.373{\cdot}10^{-3}$
1.5	cal4	$12.544{\cdot}10^{-3}$
1.5	cal8	$10.910{\cdot}10^{-3}$
1.5	varDens	$4.478{\cdot}10^{-3}$
2.0	cal0	$84.220{\cdot}10^{-3}$
2.0	cal4	$23.769{\cdot}10^{-3}$
2.0	cal8	$18.683{\cdot}10^{-3}$
2.0	varDens	$5.478{\cdot}10^{-3}$
4.0	cal0	$130.352{\cdot}10^{-3}$
4.0	cal4	$44.833{\cdot}10^{-3}$
4.0	cal8	$31.820{\cdot}10^{-3}$
4.0	varDens	$11.731{\cdot}10^{-3}$

Note that the input of the network is preprocessed by the best parameter found in the previous experiment. For each of the three reduction factors the variable density mask performs best. Calibration areas of 8×8 pixel perform better than 4×4 which perform better than no calibration area for all reduction factors. This shows the importance of sampling the center frequencies in k-space.

4.3. AUTOMAP hyperparameter tuning

The results of the previous two experiments determine the input of the neural network. The input data is preprocessed by transformations with best performing parameters. For the choice of a reduction factor of two the variable density undersampling mask is utilized for this experiment.

Table 4.3.: Top 10 AUTOMAP MSEs with different hyperparameters.

Momentum [0.0, 0.2, 0.4, 0.6, 0.8]	**Noise** [On, Off]	**Dropout(p=0.99)** [On, Off]	**Learning rate** $[2 \cdot 10^{-6}, 2 \cdot 10^{-5}, 2 \cdot 10^{-4}]$	**MSE**
0.8	Off	On	$2 \cdot 10^{-5}$	$3.422 \cdot 10^{-3}$
0.6	Off	On	$2 \cdot 10^{-5}$	$3.446 \cdot 10^{-3}$
0.4	Off	On	$2 \cdot 10^{-5}$	$3.498 \cdot 10^{-3}$
0.8	On	On	$2 \cdot 10^{-5}$	$3.510 \cdot 10^{-3}$
0.6	On	On	$2 \cdot 10^{-5}$	$3.514 \cdot 10^{-3}$
0.2	On	On	$2 \cdot 10^{-5}$	$3.523 \cdot 10^{-3}$
0.4	On	On	$2 \cdot 10^{-5}$	$3.533 \cdot 10^{-3}$
0.8	On	On	$2 \cdot 10^{-6}$	$3.594 \cdot 10^{-3}$
0.2	Off	On	$2 \cdot 10^{-5}$	$3.723 \cdot 10^{-3}$
0.6	Off	Off	$2 \cdot 10^{-5}$	$3.740 \cdot 10^{-3}$

The top 10 performing neural networks with corresponding tuning parameters and MSE are given in table 4.3. This table shows the lowest MSE of $3.42 \cdot 10^{-3}$ with momentum=0.8, no multiplicative noise on k-space input data, dropout with keep probability of 0.99 and learning rate of 2e-5. The violin plot in Figure 4.1 shows the distribution of all 60 samples regarding MSEs and corresponding hyper parameters. The 9 best performing neural networks all use dropout and momentum of at least 0.2. With each step of rising momentum a lower MSE was achieved.

4.4. Performance comparison

At this point all the parameters including input data preprocessing, undersampling mask selection and AUTOMAP hyper parameter tuning are selected by measuring the performance in terms of minimizing the MSE between the ground truth and the reconstructed image.

The performance of AUTOMAP reconstruction on unseen undersampled k-space data (reduction factor of 2) reached a MSE of $3.42 \cdot 10^{-3}$ with a standard deviation of $2.11 \cdot 10^{-3}$. The SSIM results in a value of 0.91 with a standard deviation of 0.037 and the PSNR reached 30.68 dB with a standard deviation of 3.14 dB.

To compare the performance between CS and AUTOMAP, 10 images (1 brain image and 9 from the test-set) are reconstructed with undersampled k-space data (reduction factor of 2) by both techniques. The CS reconstructions reached a SSIM of 0.95 with a MSE of $2.36 \cdot 10^{-3}$ and a PSNR of 32.29 dB. The results are shown in Table 4.4. Five of ten reconstructions, their absolute errors and their local SSIMs are presented in Figure 4.2.

The SSIM of the AUTOMAP brain image reconstruction is with 0.77 significantly lower than SSIMs from the test-set, in fact only 5 of 4282 images from the test-set show lower

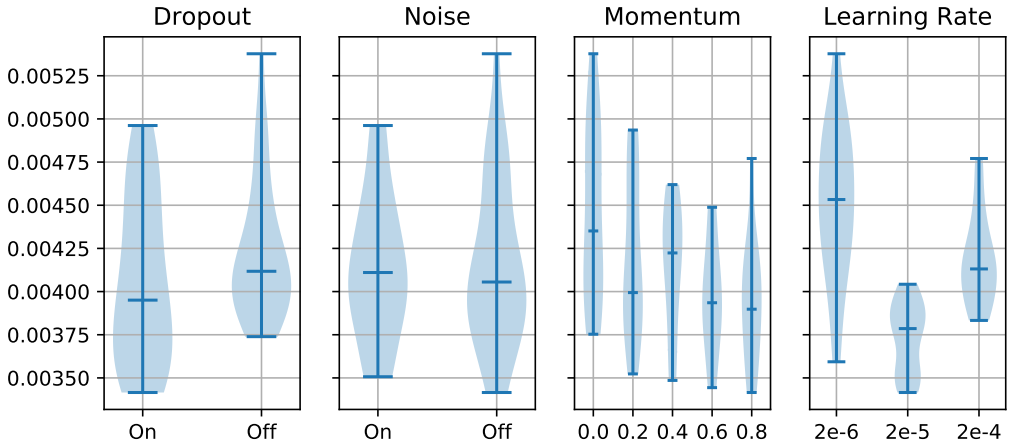


Figure 4.1.: AUTOMAP MSEs with different hyper parameters. These violin plots show minimum, median and maximum by horizontal line marks and the whole sample distribution by the shape for each hyper parameter (shape width correlates with number of samples for the corresponding MSE).

SSIMs. The largest absolute errors are found in the brain periphery where pixel intensities strongly fluctuate between high and low values. On the other hand, the local SSIMs show low structural similarities at areas of constant pixel intensity values (black background, smooth surface near the brain center). The CS reconstruction of the brain image is with a SSIM of 0.93 not significantly different to the other SSIMs of the test-set.

Table 4.4.: Reconstruction metric results for CS and AUTOMAP

	MSE	**PSNR**	**SSIM**
CS	$2.36 \cdot 10^{-3}$	32.29 dB	0.95
AUTOMAP	$3.42 \cdot 10^{-3}$	30.68 dB	0.91

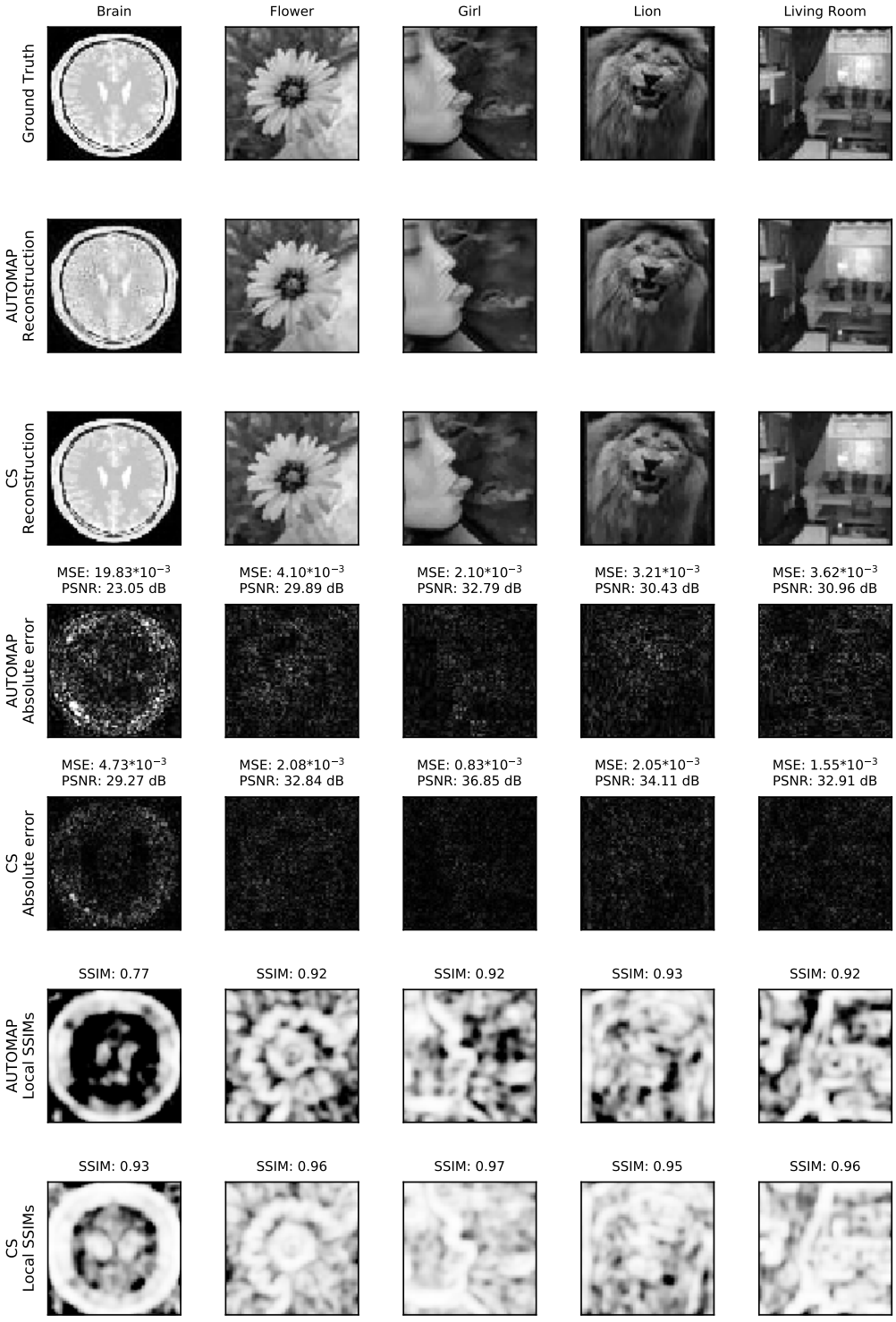


Figure 4.2.: Reconstructions of the brain image and four images from the test-set. The absolute errors range from 0 (black) to 0.25 (white) and the local SSIMs range from 1 (white) to 0.75 (black) to improve visualization.

5 Discussion

Performance

The AUTOMAP reconstruction performance on the brain image was significantly worse than other image reconstruction performances from the test data-set. The reason for this is the fact, that the neural network was trained on data which deviates more from the brain test image than the unseen data from the test-set. In other words, AUTOMAP learns patterns of structural characteristics with the training-set that do not apply to the brain image. This leads to the question what characteristics separates the brain image from the test-data. There may be many reasons which need further research to answer completely. Obvious differences are the lower structural information content (solid black background) and the fact that the brain image was not taken by a camera (computer generated) which leads to different image characteristics.

Even though the input transformation and the neural network hyperparameters were optimized, CS outperformed AUTOMAP.

On the two GeForce GTX 1080 Ti GPUs that were used for training, AUTOMAP reconstructs 1 image in 1.31 s and 1000 images in 1.89 s. The CPU needs with 0.49 s for 1 reconstruction less time, but it requires 6.02 s to reconstruct 1000 images. Hence, for a large number (more than 100) of reconstructions the GPU should be utilized for reconstruction. A comparison between the required time for reconstruction is shown in Figure 5.1. These reconstruction times are much faster compared to CS. On the other hand, any parameter changes of the neural network require retraining which can take up to 8 hours.

A MSE comparison between our results and the original AUTOMAP paper results are presented in Figure 5.2. In the original paper the reconstruction image resolution (128×128), undersampling reduction factor (2.5) and training-data (brain images) differ from this thesis. Note that the NRMSE from the original paper is converted to the MSE with an error value range of 2 to ensure comparability.

The training process of the final neural network is presented in Figure 5.3 and shows the MSEs of the training-, validation- and test-set over all epochs. The gray spikes indicate patience which reaches 20 at epoch 470, hence, the parameter state of epoch 450 is saved for reconstruction.

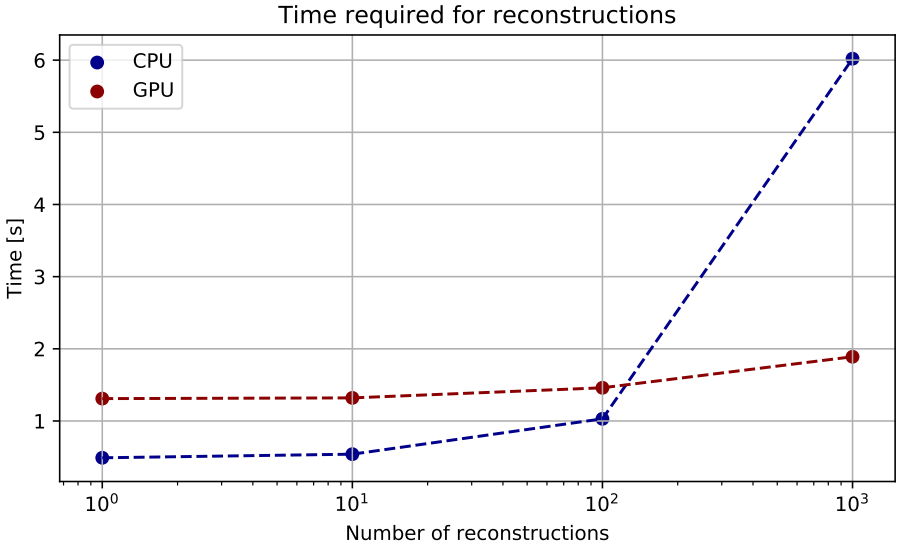


Figure 5.1.: Comparison of required time to reconstruct images between CPU and GPU.

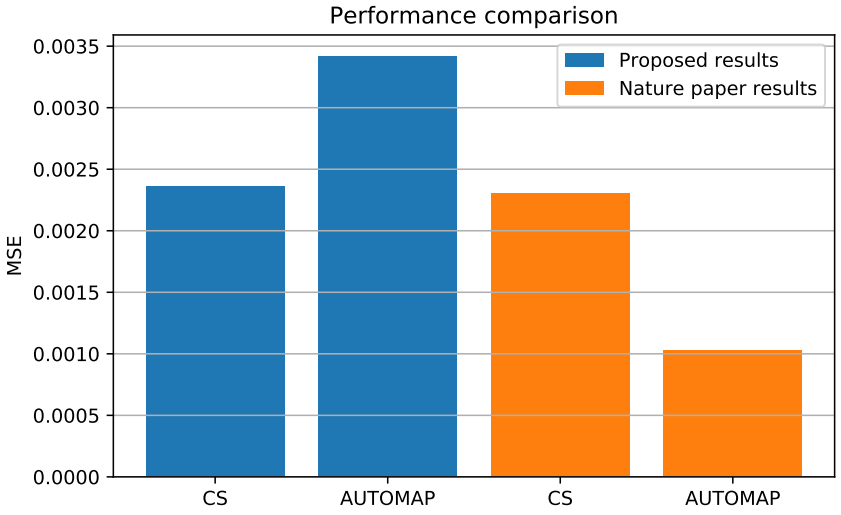


Figure 5.2.: MSE comparison between our and Nature paper reconstruction results.

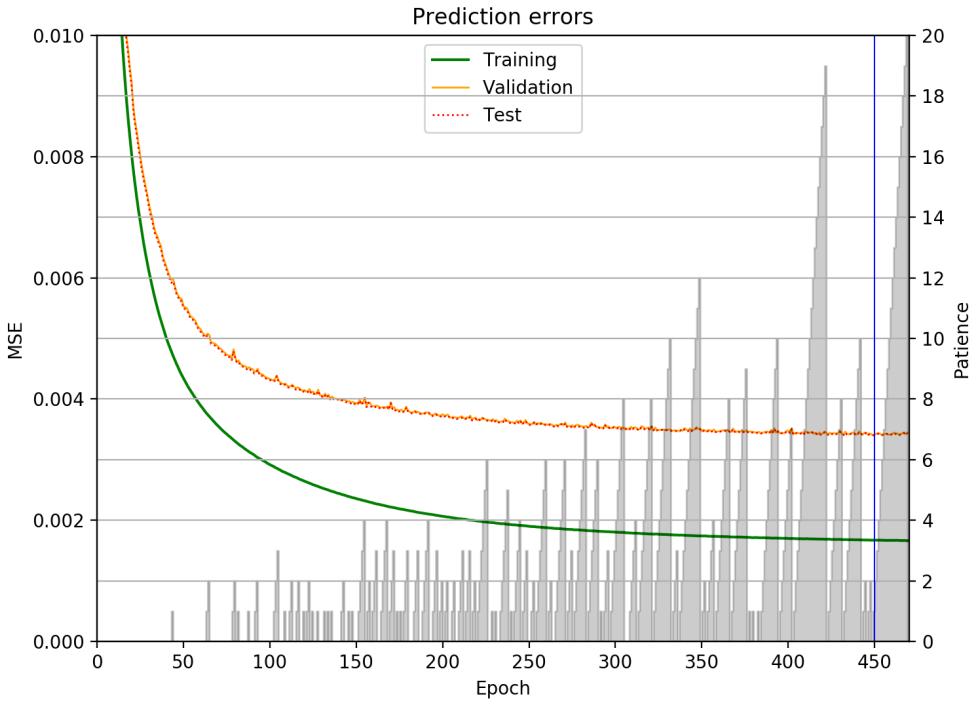


Figure 5.3.: Prediction errors during training for training- validation- and test-set for the final neural network.

One can observe that the MSE of the training-set converges near $1.7 \cdot 10^{-3}$. In the original paper a lower MSE of $1.02 \cdot 10^{-3}$ was achieved on the test-set. There are two possible reasons why no further improvement takes place on the training-set.

- Network architecture limits the performance due to too few trainable parameters to model the relationship between input and output
- Given input information content is not sufficient to reconstruct the ground truth

The first point seems highly unlikely as the original paper uses the same architecture to reconstruct images with four times as many pixels. The second point depends on the undersampling mask and the data-set. Poisson-disk undersampling with an even higher reduction factor was used aswell in the original paper, hence the difference in the data-sets appears to be the most probable reason for the varying results.

The performance of the network is highly dependent on both the data it is trained with and the data it is tested with. The neural network from the original paper was trained using only T1-weighted brain images from the Human Connectome Project (HCP). In contrast to training with natural images, training with this data-set results in greater sparsity of the activation values in the fully connected layers. Note that no regularization term is used to force sparsity. These sparse hidden layer activations indicate successful extraction of robust features. [29]

In practice, AUTOMAP reconstruction performance improvement is expected if training images are from the corresponding domain. Hence, for e.g. abdominal MR images the network should be trained with images from the same domain. CS uses a generic regularization and can be applied to data of different image domains without further training. Without changing the regularization parameters, the performance of CS didn't show differences in the domain of data.

Memory demand

The width and height of the reconstructed images was chosen to be $n = 64$ due to hardware and time limitations. Most of the computing time is spend during backpropagation. Gradients are calculated and network parameters are updated with each training step. The more network parameters are used the higher time requirements for training. Additionally, a large amount of internal memory is needed to train the neural network as each instance of the training batch is loaded into memory while all trainable parameters stay present as well. This leads to the question how memory demand scales with higher (more practical) image resolution.

The number of parameters for a fully connected layer connected to another fully connected layer is the sum of weights and biases. The number of weights is the product of the number of neurons of both layers and the number of biases is the number of neurons of the second layer.

The number of parameters for a convolutional layer connected to another convolutional layer is the sum of weights and biases. The number of weights is the product of number of kernels, the number of channels of the input image and the size (width $\times$ height) of the kernel. The number of biases is the number of kernels.

Table 5.1.: Number of weights and biases for AUTOMAP

Layer	**FC 1**	**FC 2**	**Conv 1**	**Conv 2**	**Deconv**	**All**
n weights	$2n^4$	n^4	$64 \cdot 5^2 \cdot$	$64^2 \cdot 5^2$	$1 \cdot 7^2 \cdot 64$	$3n^4 + 107136$
n biases	n^2	n^2	64	64	1	$2n^2 + 129$
Total	$n^2(2n^2 + 1)$	$n^2(n^2 + 1)$	1664	102464	3137	$3n^4 + 2n^2 + 107265$

The number of weights and biases for each layer of AUTOMAP is presented in Table 5.1. Hence, the total number of trainable parameters for AUTOMAP is $3 \cdot 64^4 + 2 \cdot 64^2 + 107265 \approx 50.4 \cdot 10^6$. If we compare this number to the ResNet50 which has $25.61 \cdot 10^6$ parameters, AUTOMAP exceeds this number at an image resolution of only 54×54 pixels.

If we want to store the neural network weights and biases for images with a resolution of 512×512 pixels and assume that each trainable variable is stored as a 32 bit floating point with 4 bytes of memory allocation, the minimum storage needed to store these information is 824.64 GB. Note that this number represents a lower bound as in practice many more framework dependent data needs to be stored.

6
Conclusion

Efficient image reconstruction with undersampled k-space data plays a significant role for both economy and healthcare. AUTOMAP was trained to reconstruct natural images with undersampled k-space data. The input data transformation, choice of undersampling mask and the neural network hyperparameter were optimized. K-space data normalization leads to best results if each spatial frequency from k-space is scaled separately throughout the dataset. Variable density Poisson-disk sampling yields the performance for all tested reduction factors in comparison to random sampling with calibration areas. 60 differently configured neural networks showed best results with additional utilization of the momentum method and dropout layers with high keep probabilities of $p_{keep} = 0.99$. Training the network demands time and computational power, especially with higher more practical resolutions. AUTOMAP is capable to reconstruct images faster than CS, but the reconstruction performance on unseen test-data shows a lower SSIM (0.91) than CS (0.95).

The performance can be improved by training with domain-specific images that enable AUTOMAP to extract more robust features. Furthermore, we expect AUTOMAP to outperform CS if domain-specific images are used during training.

List of Figures

List of Tables

Bibliography

[1] Polina Mamoshina, Armando Vieira, Evgeny Putin, and Alex Zhavoronkov. Applications of deep learning in biomedicine. *Molecular pharmaceutics*, 13(5):1445–1454, 2016.

[2] Rasool Fakoor, Faisal Ladhak, Azade Nazi, and Manfred Huber. Using deep learning to enhance cancer diagnosis and classification. In *Proceedings of the international conference on machine learning*, volume 28. ACM New York, USA, 2013.

[3] Tao Zeng, Rongjian Li, Ravi Mukkamala, Jieping Ye, and Shuiwang Ji. Deep convolutional neural networks for annotating gene expression patterns in the mouse brain. *BMC bioinformatics*, 16(1):147, 2015.

[4] Robert W Cox and Andrzej Jesmanowicz. Real-time 3d image registration for functional mri. *Magnetic Resonance in Medicine: An Official Journal of the International Society for Magnetic Resonance in Medicine*, 42(6):1014–1018, 1999.

[5] Eurostat. Number of magnetic resonance imaging (mri) scan examinations in germany from 2008 to 2015 (in 1,000). Eurostat, August 2018.

[6] Christopher J Lisanti and David B Douglas. Effects of breath-hold and cardiac cycle on the mri appearance of the aorta and inferior vena cava in t2 abdominal imaging. *American Journal of Roentgenology*, 192(5):1348–1358, 2009.

[7] Emmanuel Candes, Nathaniel Braun, and Michael Wakin. Sparse signal and image recovery from compressive samples. In *2007 4th IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, pages 976–979. IEEE, 2007.

[8] Emmanuel J Candès et al. Compressive sampling. In *Proceedings of the international congress of mathematicians*, volume 3, pages 1433–1452. Madrid, Spain, 2006.

[9] Sarah Abdulla. Mri spatial localisation, July 2019. Available at `https://www.radiologycafe.com/images/physics/mri-spatial-localisation@2x.png`.

[10] Stephen H Cutcliffe, Christine M Roysdon, and Judith A Adams. Current bibliography in the history of technology (1986). *Technology and Culture*, 29(2):338, 1988.

[11] JP Felmlee, RL Morin, JR Salutz, and GB Lund. Magnetic resonance imaging phase encoding: a pictorial essay. *radiographics*, 9(4):717–722, 1989.

[12] Michael Lustig, David Donoho, and John M Pauly. Sparse mri: The application of compressed sensing for rapid mr imaging. *Magnetic Resonance in Medicine: An Official Journal of the International Society for Magnetic Resonance in Medicine*, 58(6):1182–1195, 2007.

[13] Balas Kausik Natarajan. Sparse approximate solutions to linear systems. *SIAM journal on computing*, 24(2):227–234, 1995.

[14] Felix Krahmer, Christian Kruschel, and Michael Sandbichler. Total variation minimization in compressed sensing. In *Compressed Sensing and its Applications*, pages 333–358. Springer, 2017.

[15] Herbert Werner. Linear and nonlinear system identification, May 2016. Available at `https://www.tuhh.de/t3resources/ics/PDFs/LN_Systemident/lnsi.pdf`.

[16] Lars Kai Hansen and Peter Salamon. Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence*, 12(10):993–1001, 1990.

[17] Bekir Karlik and A Vehbi Olgac. Performance analysis of various activation functions in generalized mlp architectures of neural networks. *International Journal of Artificial Intelligence and Expert Systems*, 1(4):111–122, 2011.

[18] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[19] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3, 2013.

[20] Kevin Swersky Geoffrey Hinton, Nitish Srivastava. Neural networks for machine learning - lecture 6a - overview of mini-batch gradient descent, 2012. Available at `http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`, Version 1.4.

[21] Amitabh Basu, Soham De, Anirbit Mukherjee, and Enayat Ullah. Convergence guarantees for rmsprop and adam in non-convex optimization and their comparison to nesterov acceleration on autoencoders. *arXiv preprint arXiv:1807.06766*, 2018.

[22] Mahesh Chandra Mukkamala, Matthias Hein, and Joachim Weickert. *Variants of RMSProp and Adagrad with Logarithmic Regret Bounds*. PhD thesis, Universität des Saarlandes Saarbrücken, 2017.

[23] Google inc. Rmsprop implementation. Online. Available at `https://www.tensorflow.org/api_docs/python/tf/train/RMSPropOptimizer`.

[24] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[25] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[26] Google inc. Dropout implementation. Online. Available at `https://www.tensorflow.org/api_docs/python/tf/train/RMSPropOptimizer`.

[27] Gérard Dreyfus. *Neural networks: methodology and applications*. Springer Science & Business Media, 2005.

[28] Yoshua Bengio, Ian J Goodfellow, and Aaron Courville. Deep learning. *Nature*, 521(7553):436–444, 2015.

[29] B. Zhu, J. Z. Liu, S. F. Cauley, B. R. Rosen, and M. S. Rosen. Image reconstruction by domain-transform manifold learning. 555:487–492, March 2018.

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

[31] Sklearn. Ssim implementation. Online, May 2019. Available at `https://github.com/scikit-image/scikit-image/blob/master/skimage/measure/_structural_similarity.py`.

[32] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. Ieee, 2009.

[33] Tian-hu Zhang and Xue-yi You. Comparing the linear and logarithm normalized artificial neural networks in inverse design of aircraft cabin environment. In *Building Simulation*, volume 9, pages 729–734. Springer, 2016.

A

Reconstruction examples

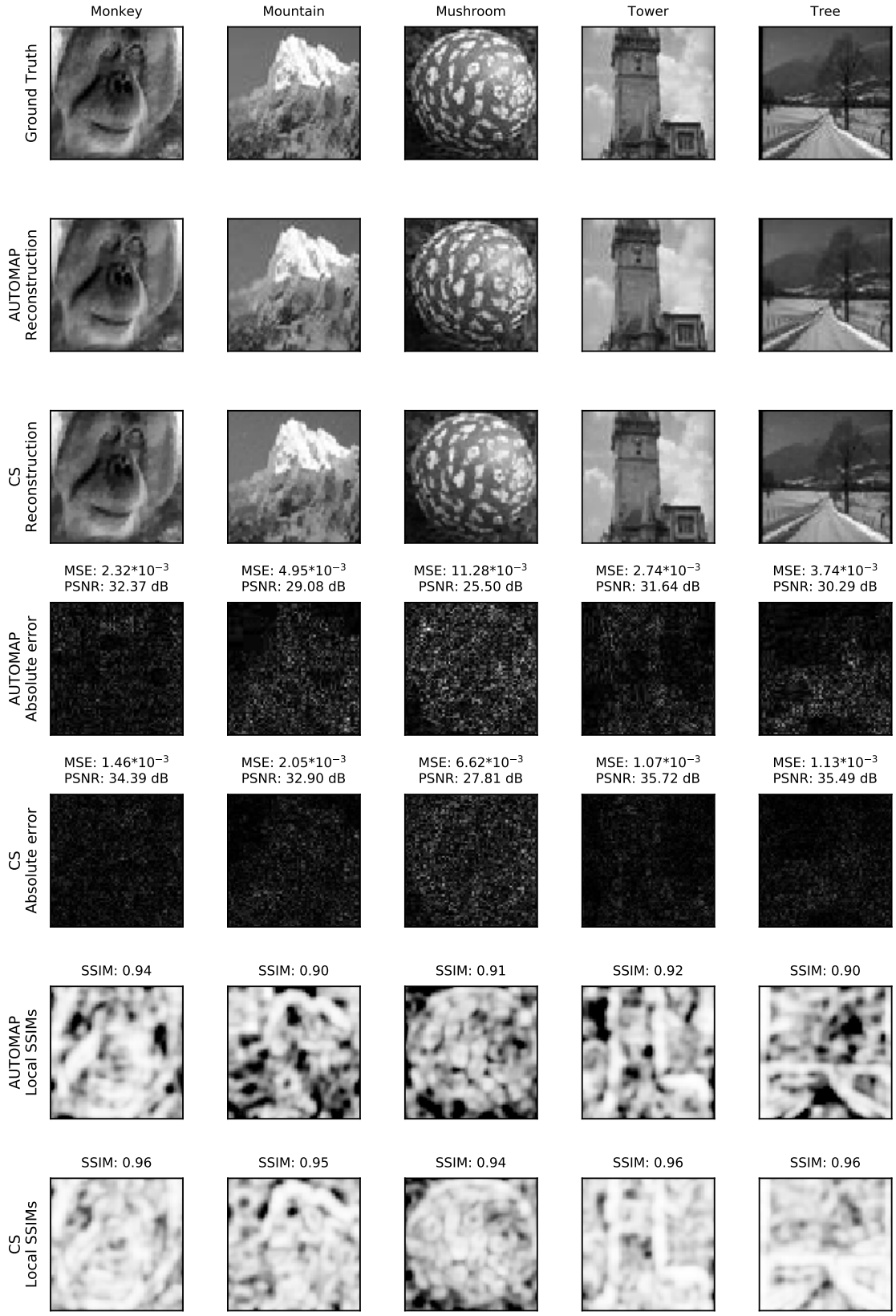


Figure A.1.: Reconstructions of five images from the test-set. The absolute errors range from 0 (black) to 0.25 (white) and the local SSIMs range from 1 (white) to 0.75 (black) to improve visualization.