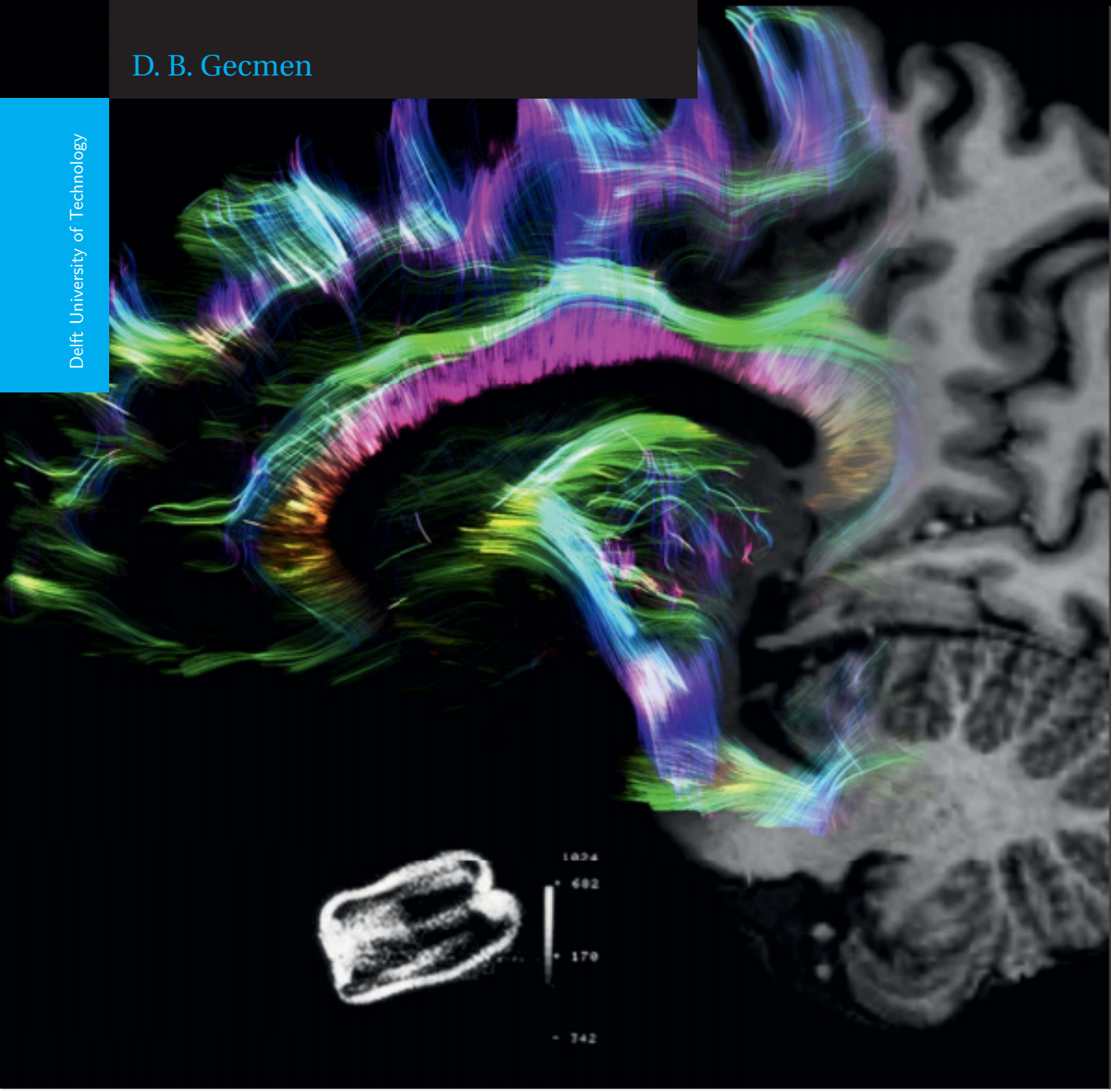
Deep Learning Techniques for Low-Field MRI

D. B. Gecmen

Delft University of Technology



TU Delft Delft University of Technology

Challenge the future

Deep Learning Techniques for Low-field Magnetic Resonance Imaging

by

Dilan Burçin Gecmen

In partial fulfillment of the requirements for the degree of
Master of Science in Applied Mathematics
at the Delft University of Technology,
as part of the master program
Applied Mathematics

Student number:	4221168	
Thesis committee:	Dr. ir. M.B. van Gijzen,	TU Delft, supervisor
	Ir. M. de Leeuw den Bouter,	TU Delft, supervisor
	Prof. dr. ir. H.X. Lin,	TU Delft
	Dr. ir. R.F. Remis,	TU Delft

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

PREFACE

To many I am indebted for their support in various ways. However, above all there are a few to whom I am most obliged. First of all, I would like to thank my supervisor Martin van Gijzen and project members Thomas O'Reilly and Danny de Gans for their incessant support. Without their generous guidance none of this would have been possible. Secondly, yet to no lesser extent, I want to thank my parents, sisters, friends, and my boyfriend. Their unconditional and loving trust in my abilities have often helped me beyond the scope of their and my own knowledge.

D. B. Gecmen
Delft, February 2020

ABSTRACT

Delft University of Technology (TU Delft), Leiden University Medical Center (LUMC), Pennsylvania State University (PSU) and Mbarara University of Science and Technology (MUST) have an ongoing collaboration to create an affordable, portable and simplified version of the magnetic resonance imaging (MRI) scan for the CURE children's hospital to diagnose children with hydrocephalus (water on the brain). As opposed to the conventional MRI scan, the low-field MRI prototype uses permanent magnets to create a magnetic field in the order of Milliteslas (mT). A downside of the low-field MRI application is the difficulty with spatial encoding due to small variations in the strength of magnetic field. This is a major problem for image reconstruction.

The purpose of this research was to implement a deep learning (DL) network to overcome two of the major bottlenecks in image reconstruction for low-field MRI. These are the lack of real measured data for DL purposes, and the signal model associated with the low-field MRI. For DL purposes we generated synthetic data and acquired measured data. Each dataset consists of samples and each sample consist of an image and the corresponding signal. Due to technical limitations the measured dataset is small, 53 samples. To partially circumvent the problem, the data set was augmented to a total of 1908 samples. In addition, we used Transfer learning, which is a powerful method that applies knowledge gained from one problem to a different but related problem.

We present three image reconstruction techniques, Model I, II, and III, based on convolutional and feed-forward neural networks, which take MR signal data as input and directly and quickly outputs an image. We demonstrated that DL generates high quality images using synthetic data. In addition, we showed that Model III needs less training to reconstructs good quality images compared to Models I and II, respectively. Finally, Models I and III were unsuccessfully applied to real measured data. However, this study shows that neural networks are able to find a mapping between signal and image, therefore this idea can be extended to work on real measured data.

CONTENTS

ABBREVIATIONS

AI	Artificial intelligence
ANN	Artificial neural network
BGD	Batch gradient descent
CGLS	Gradient method for least squares
CNN	Convolutional neural network
CPU	Central processing unit
CT	Computed tomography
DL	Deep learning
FFT	Fast fourier transform
FNN	Feedforward neural network
FID	Free induction decays
GAN	General adversarial network
GD	Gradient descent
INSY	Intelligent Systems Department
LReLU	Leaky rectified linear unit
ML	Machine learning
MLP	Multi-layer perceptron
MRI	Magnetic resonance imaging
NMR	Nuclear Magnetic Resonance
PSD	Phase-sensitive detection
QD	Quadrature detection
ReLU	Rectified linear unit
RF	Radio frequency
rSEMs	Rotating spatial encoding magnetic fields
SE	Spin echo
SEM	Spatial encoding magnetic field
SGD	Stochastic gradient descent
SSR	Super-resolution reconstruction
TE	Echo time

1

INTRODUCTION

From visible light, the visual system (part of the central nervous system) builds a representation of our surrounding environment. These representations, which consists out of images, play a key role in our daily lives. As a matter of fact, not only the visual system, but also various imaging devices are responsible for a huge part of the knowledge humankind has attained about themselves, the world, and even the universe. For the human body, Magnetic Resonance Imaging (MRI) has extended the range of the human vision regarding the human body. MRI produces detailed images of the inside of the body at high resolution, speed, and information content. This makes MRI an effective way to detect many different diseases. However, conventional MRI scanners found in hospitals are complex, use strong magnetic fields, require high power electricity to push through the coiled copper wires and as a result require very active cooling using liquid helium. This makes conventional MRI scanners complex, expensive and difficult to operate and maintain. As a result, this powerful technology is sparsely or not at all accessible to doctors and patients in many parts of the world.

Due to the lack of access to MRI scanners in developing countries, many diseases remain untreated. One of these diseases is hydrocephalus, often called 'water on the brain'. If left untreated hydrocephalus leads to permanent brain damage, and in many cases eventually death. This disease usually occurs in infants causing swelling of the skull and is often caused by a bacterial infection. In Uganda, there is an estimation of 1000 to 2000 cases of hydrocephalus on a yearly basis [1]. Many of these cases remain untreated as the country, with a population of over 40 million, has limited access to MRI scans. The CURE children's hospital in Uganda, specialized in the detection and treatment of infant hydrocephalus, uses computed tomography (CT) brain imaging. The use of CT scans is highly undesirable due to X-ray radiation, which is dangerous for developing children.

Delft University of Technology (TU Delft), Leiden University Medical Center (LUMC), Pennsylvania State University (PSU) and Mbarara University of Science and Technology (MUST) have an ongoing collaboration to create an affordable, portable and simplified version of the MRI scan for the CURE children's hospital to diagnose children with hydrocephalus. The low-field MRI prototype created by TU Delft and LUMC is given in Figure 1.1.

(a) Exterior of the low-field MRI prototype.

(b) Inside of the low-field MRI prototype.

Figure 1.1: The low-field MRI prototype designed by TU Delft and LUMC to image the head.

A conventional MRI scan uses a magnet, radio waves and mathematics to make highly detailed images. When a body is placed in the magnetic field of the scan, the magnetic hydrogen atoms align to the field. After this alignment, a radio wave pulses the atoms and thereby distorts their polarity. A built-in sensor then detects the time that the atoms need to return to their original alignment. Mathematical methods are used to process the measurements by a computer and create a black and white image. Hard bone and air do not give an MR signal, as a result, these areas appear black. Soft tissue, blood, spinal fluid, and bone marrow vary in intensity from black to white.

As opposed to the conventional MRI scan, the low-field MRI prototype uses permanent magnets to create a magnetic field in the order of Milliteslas (mT). A downside of the low-field MRI application is the difficulty with spatial encoding due to small variations in the strength of magnetic field. This is a major problem for image reconstruction.

1.1. SCOPE

In recent years, deep learning has played an increasingly important role in the field of medical imaging [2]. This raises the question of whether we can implement a deep learning-based approach for low-field MRI. The goal of this thesis is to extensively test various deep learning models for low-field image reconstruction using synthetic and measured data. This leads to our research question:

"Is it possible to implement a deep learning based approach to generate images using measured signals from the low-field MRI?"

1.2. REPORT OUTLINE

The report starts in Chapter 2 with an overview of the principles of the conventional MRI scan. Subsequently, the low-field MRI prototype and the signal model developed by TU Delft and LUMC is introduced in Chapter 3. Because of the non-homogeneous magnetic field of the prototype, standard model based reconstruction techniques do not yield good enough images to use in a clinical setting. Therefore, a deep learning-based approach will be considered in this research. In Chapter 4 the theory of deep learning will be discussed. The main focus is to present neural network architectures that can be used to develop an image reconstruction model. Among these architectures are the Feed Forward Neural Network and the Convolutional Neural Network, which are extended to our current problem.

For the evaluation of the deep learning models, synthetic data based on the signal model of the low-field MRI scan, as well as measured data from the actual scanner are used. The generation of simulated data and the acquisition of measured data are described in Chapter 5.

In Chapter 6 we formulate our three new deep learning models for image reconstruction. Besides the models, we elaborate upon the design of test cases used to run the simulations. The results of these test cases are presented, analyzed and evaluated for simulated and measured data in Chapters 7 and 8. Finally, Chapter 9 contains the conclusions and a discussion.

2

CONVENTIONAL: MAGNETIC RESONANCE IMAGING

An MRI scan takes detailed pictures of various human body tissues using electromagnetic waves coming from the hydrogen atoms in water and fat molecules. A hydrogen nucleus consists of a single proton, which can be visualized as a small bar magnet with a north-south pole spinning on its axis, see Figure 2.1.

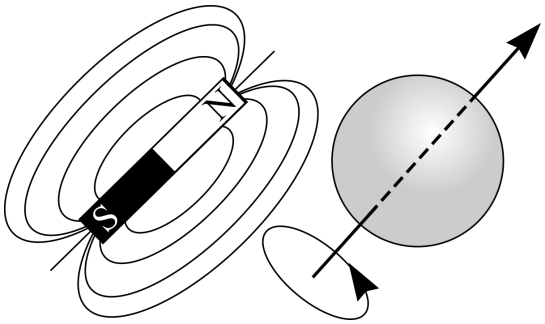


Figure 2.1: Illustration of a hydrogen proton as a small bar magnet spinning on its axis, which is randomly aligned [3].

When a radiologist in the hospital turns on the MRI scan, a strong constant magnetic field B_0 is produced, which forces the hydrogen protons to align with this field. B_0 remains in place for the duration of the measurement. For a short time, the MRI emits pulses from a weaker electromagnetic field B_1, applied perpendicularly to B_0. B_1 is used to disrupts the alignment of the protons with B_0 and make them 'jiggle'; this jiggling is also known as nuclear magnetic spin. After the end of the pulses, the protons are gradually aligned again with B_0. The duration of time that the re-alignment requires depends on the tissue being examined. The re-alignment of the protons results in the emission of low energy, radio frequency photons. The detector of an MRI monitors the emission of the photons, allowing a radiologist to identify different tissues.

MRI is a medical imaging technique that arises from the application of Nuclear Magnetic Resonance (NMR) to radiological imaging. For this reason, MRI could also be called Nuclear Magnetic Resonance Imaging (NMRI). Because MRI was invented during the Cold War, there was widespread concern about everything that contained the word nuclear. As a result, the medical world embraces MRI instead of NMRI.

In this Chapter the principles of MRI will be discussed. The theory and images in this Chapter are based on [3], [4] [5].

2.1. HISTORY OF MRI

As already mentioned, MRI is based on the phenomenon of Nuclear Magnetic Resonance (NMR). MRI has its underpinnings in the discovery of the spin of a proton in 1922 by physicists Otto Stern and Walter Gerlach [6]. This discovery prompted physicist Isidor Rabi to pursue the spin of the proton and its interaction with a magnetic field. In 1938 Rabi developed a technique to measure the movement of atomic nuclei, which he called NMR [7]. With the help of this new technique, Rabi described and measured NMR in molecular

beams and in 1944 he received the Nobel Prize in Physics for his efforts. Rabi's method was extended for use on liquids and solids in 1946, simultaneously and independently by physicists Felix Bloch [8] and Edward Purcell [9]. They measured a precessional signal of samples taken from solids and liquids.

Rabi's techniques were mainly used to analyze structures of chemical substances. In the 1960s, physician Raymond Damadian wondered whether the same techniques could be used on organisms. In 1971 he came to the conclusion that cancerous tissue contains more water than healthy tissue, which can be detected by a scanner that emits radio waves, and detects and measures emissions from the hydrogen atoms in the tissue [10]. At the same time, the first 2D and 3D MRI images of green peppers and clams where generated by chemist Paul Lauterbur [11]. After reading the work of Damadian, Lauterbur realized that his method could also have biomedical applications. According to Lauterbur, two-dimensional images can be stacked to create a three-dimensional image [12]. Meanwhile, the first biomedical application was carried out by physicist Peter Mansfield [13], who wanted to create a new method to complete scans in minutes instead of in hours. He was the first to successfully scan a body part using NMR, namely a student's finger within 15-23 minutes.

In 1977, Damadian and his team built a scanner that was large enough for the human body and after several experiments they succeeded in creating a two-dimensional image [14]. However, Damadian's methods were too slow to use in a clinical setting and they were rejected. Instead, the methods of Lauterbur and Mansfield were adopted and in 2003 they received the Nobel Prize in Medicine, which Damadian could not appreciate.

In 1980, physicist Paul Bottomley and his team built the first whole body MRI, which translated into a successful MRI product-line that is still in use.

2.2. HARDWARE COMPONENTS

An MRI scan consists of three main hardware components: a main magnet, a magnetic field gradient system, and an RF system. These components are the magnetic field sources used to manipulate the magnetic moments of the hydrogen atoms in tissue. In this Section, each hardware component is briefly discussed.

2.2.1. THE PRIMARY MAGNET

The purpose of magnets used for MRI is to produce a static, strong and homogeneous magnetic field that is referred to as the B_0 field over a clinically useful field of view (FOV). The optimum field strength for imaging is application dependent.

Most of the B_0 magnets in MRI scans are superconducting magnets that can generate a magnetic field of up to 9 Tesla (T), resulting in a high signal-to-noise ratio. A superconducting magnet is a solenoid electromagnet made with superconducting wire. The wire is immersed in liquid helium that causes the resistance of the wire to drop to zero to allow electric current to flow through the coil to create the magnetic field. This use of liquid helium makes a superconducting magnet very expensive.

Resistive magnets and permanent magnets are used for low-field MRI applications. Resistive magnets generate a B_0 field of < 0.5 T. The homogeneity limitation results in very heavy magnets and a constant power supply, which can be very expensive in maintenance. That is why LUMC and TU Delft have built a low-field MRI with permanent magnets. Permanent magnets are magnetized using an external magnetic field. When the external field is removed a residual permanent magnetization B_r, called the remanent flux density, is left in the magnet that is always present and at full strength. This means that a permanent magnet costs nothing to maintain. A major disadvantage is that these magnets are very heavy and that even heavier magnets are needed to generate higher fields.

2.2.2. THE GRADIENT SYSTEM

In conventional MRI systems, the magnetic field gradient consists of three sets of orthogonal gradient coils: the x-, y-, and z-gradients. Figure 2.2 shows the configuration of such coils. The primary function of the gradient coils is to produce deliberate variations in the B_0-field, which allows the spatial encoding of the MR signal in the x-, y- and z-direction. An important requirement for a gradient system is the maximum gradient strength. The unit of measurement of the gradient strength is in millitesla per meter (mT/m) and the higher the gradient strength, the better. Another important feature is the speed at which the maximum gradient strength can be obtained. The rise time is the time interval that a gradient system needs to reach full strength and the better the gradient system, the shorter the rise time.

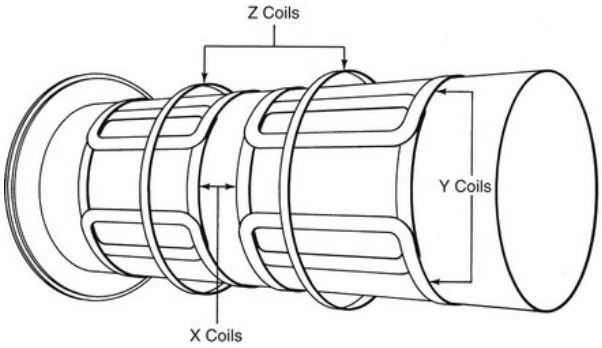


Figure 2.2: Schematic representation of the x-, y-, z-gradient coil used to establish the x-, y- and z- gradient [4].

2.2.3. THE RADIO FREQUENCY SYSTEM

The third main component of the MRI system is the radio frequency (RF) system. The RF system consists of coils that can serve as transmitters, receivers, or both. Coils that are used as transmitters are capable of generating a rotating magnetic field, called the B_1-field, for excitation of a spin system. The B_1 field is perpendicular to the B_0-field and is only enabled for a short time, called RF pulses. To convert the precessing magnetization into an electric signal, receiver coils are used. Transmitter and receiver coils are also called RF-coils because they resonate at radio frequency, for excitation of the spin system and signal detection. Necessary functions of the RF system are to provide a uniform B_1 field and high detection sensitivity. To this end, an MRI scan often contains RF coils with different designs and physics for specific body regions.

2.3. SIGNAL GENERATION AND DETECTION

The focus in this Section is on signals. In particular, what are signals and how are they generated and detected from objects.

2.3.1. MAGNETIZED NUCLEAR SPIN SYSTEMS

Atoms are the building blocks of every physical object. Atoms are composed of electrons, protons, and neutrons. In the central part of the atom, protons and neutrons are clustered together, this is called the nucleus of an atom. Electrons rotate around the nuclei. When atoms come together they form molecules, which in turn form physical objects. Nuclei with odd atomic numbers have a spin angular momentum of $\vec{J}$. The nucleus of the hydrogen atom consists of a single, positively charged proton, see Figure 2.3, therefore it possesses spin.

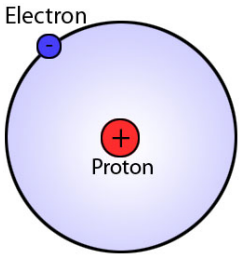


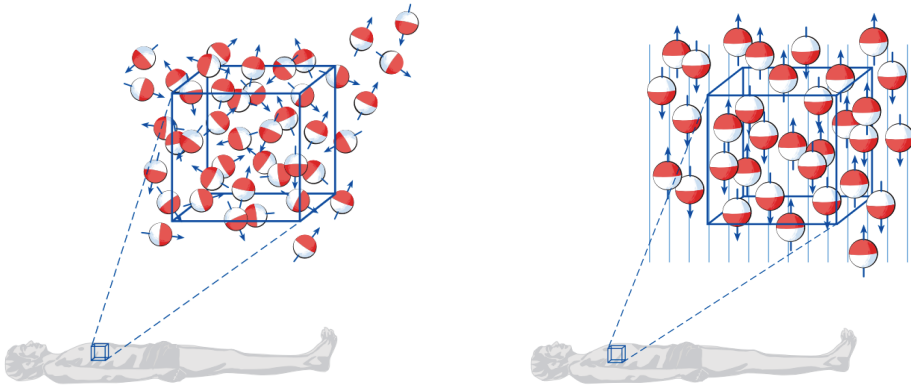
Figure 2.3: Illustration of a hydrogen atom [3].

Spin can be visualized as a physical rotation similar to the spinning of a top around its own axis. An important property of a nucleus with spin is known as nuclear magnetism created via setting it in an external magnetic field. This is physically represented by the vector $\vec{\mu}$, called the nuclear magnetic moment. The nuclear spin angular momentum $\vec{J}$ and the nuclear magnetic moment $\vec{\mu}$ are related to each other by

$$\vec{\mu} = \gamma \vec{J} \tag{2.1}$$

where γ is the gyromagnetic ratio. The value of this ratio is nucleus dependent and for hydrogen it is $\gamma = 42.6$ MHz$\cdot$ T^{-1}. Because $\vec{\mu}$ is a vector quantity, we need to know its magnitude and orientation. With or without a magnetic field, the magnitude of $\vec{\mu}$ is known. However, if there is no external magnetic field, the

direction of $\vec{\mu}$ is completely random, see Figure 2.4a. By applying a strong external magnetic field B_0, the spin vectors are forced to align, see Figure 2.4b. You can compare this with a compass needle, where in our case $\vec{\mu}$ tries to align with B_0.



(a) Nuclear magnetic moment vectors randomly oriented without a magnetic field.

(b) A part of the nuclear magnetic moment vectors aligns with B_0 and a part does not align.

Figure 2.4: Alignment of the nuclear magnetic moment vectors with and without B_0 [3].

B_0 is applied in the z-direction such that

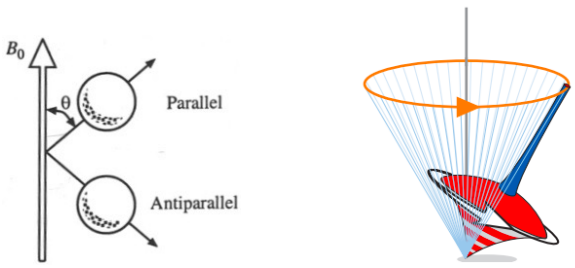
$$\vec{B_0} = B_0 \vec{k} \tag{2.2}$$

where $\vec{k}$ is the unit vector in the direction of the z-axis. For hydrogen atoms, the spin system is called spin-$\frac{1}{2}$. For atoms with a spin-$\frac{1}{2}$ system the direction of $\vec{\mu}$ points upwards (parallel) or downwards (antiparallel), see Figure 2.5a. When $\vec{\mu}$ is aligned with B_0, it experiences a torque τ given by

$$\tau = \vec{\mu} \times \vec{B}_0 \tag{2.3}$$

The movement described by the previous Equation is called nuclear precession, which can be compared to a wobbling top around its gravitational axis, see Figure 2.5b. The angular frequency of nuclear precession is called the Larmor frequency or the demodulation frequency and is given by

$$\omega_0 = \gamma B_0 \tag{2.4}$$



(a) Spin orientation [4].

(b) Spin precession [3].

Figure 2.5: For a spin-$\frac{1}{2}$ system, parallel and anti-parallel are the two preferred spin orientations in the presence of B_0.

2.3.2. NET MAGNETIZATION

As mentioned earlier, there are two spin states: parallel or antiparallel. According to quantum theory, spin states have an energy of

$$E = -\gamma m_I \hbar B_0 \tag{2.5}$$

where $\hbar$ is Planck's constant. For spins pointing up $m_I = \frac{1}{2}$,

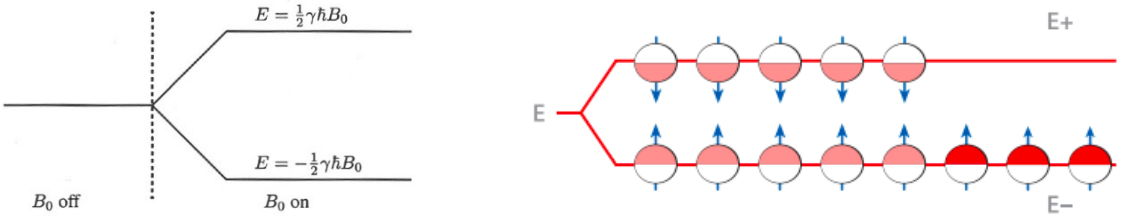
$$E\uparrow = -\frac{1}{2}\gamma\hbar B_0 \tag{2.6}$$

and for spins pointing down $m_I = -\frac{1}{2}$,

$$E\downarrow = \frac{1}{2}\gamma\hbar B_0 \tag{2.7}$$

The spin-up state is the lower energy state and the spin-down state is the higher energy state. The energy difference between the two states is called the Zeeman splitting, see Figure 2.6a, and is given by

$$\Delta E = E\downarrow - E\uparrow = \gamma\hbar B_0 \tag{2.8}$$



(a) The Zeeman splitting for a spin-$\frac{1}{2}$ system [4].

(b) More spins jump into the lower energy state $E-$ than into the higher state $E+$ [3].

Figure 2.6: Illustration of the Zeeman splitting.

A nuclear spin is more likely to take the lower energy state because of the higher stability. Therefore, more spins jump into the lower energy state, see Figure 2.6b. This produces a perceptible macroscopic magnetization that points in the same direction as B_0, referred to as net magnetization $\vec{M}$ given by

$$\vec{M} = \sum_{n=1}^{N_s} \vec{\mu}_n \tag{2.9}$$

where N_s is the number of spins and μ_n is the magnetic moment of the nth spin. The magnitude of $\vec{M}$ is directly proportional to B_0 and N_s. An illustration of net magnetization is given in Figure 2.7

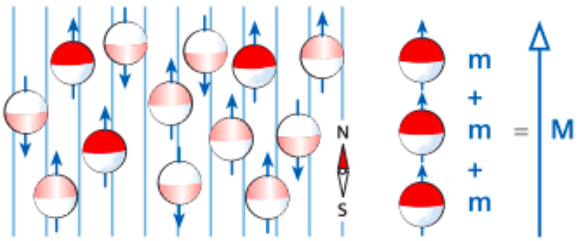


Figure 2.7: The excess spin magnets m form a perceptible macroscopic magnetization $\vec{M}$. [3]

For an object imaged, the net magnetization $\vec{M}$ induced by $\vec{B_0}$ for each point $\vec{r} = (x, y, z)$ is given by

$$\vec{M} = M_x\vec{i} + M_y\vec{j} + M_z\vec{k} \tag{2.10}$$

where $\vec{i}$, $\vec{j}$, $\vec{k}$ represent the unit vectors along the x, y, and z axes.

2.3.3. FREE PRECESSION, EXCITATION AND RELAXATION

During the excitation stage, the energy for an MRI system is supplied in the form of an external magnetic field $\vec{B}_1(\vec{r},t)$ perpendicular to $\vec{B_0}$ to induce time-varying changes in the magnetization:

$$\vec{M}(\vec{r},t) = M_x(\vec{r},t)\,\vec{i} + M_y(\vec{r},t)\,\vec{j} + M_z(\vec{r},t)\,\vec{k} \tag{2.11}$$

$\vec{B}_1$ alternates at the Larmor frequency for a short duration of time, also known as radio frequency (RF) pulses, and tips $\vec{M}$ away from equilibrium, i.e., z-axis. This causes $\vec{M}$ to have a component in the transverse plane, i.e., $(x,y)-$plane. The transverse magnetization is defined by:

$$M(\vec{r},t) = M_x(\vec{r},t) + iM_y(\vec{r},t) \tag{2.12}$$

If B_1 is removed and enough time is given, the spin eventually returns to its thermal equilibrium state. This process is illustrated in Figure 2.8.

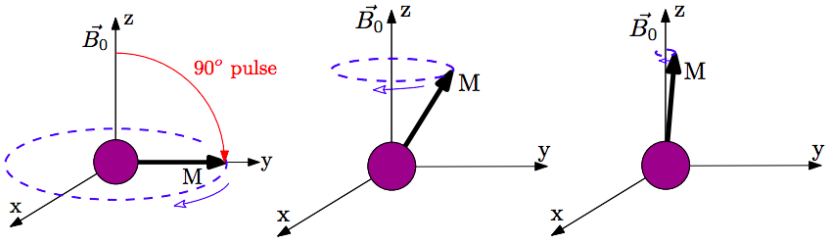


Figure 2.8: Left: Free precession of $\vec{M}$ about B_0 following a 90° excitation pulse that flips M in the transverse plane. Middle: During the relaxation period, M returns to its original alignment with B_0. This process of recovering the longitudinal magnetization M_z is called transverse relaxation. Right: The transverse magnetization M_{xy} is destructed and M_z is recovered.[3]

During the relaxation period, $\vec{M}$ spirals back to the z-axis, the magnitude is not preserved due to the relaxation process. This is different during the excitation period, where $\vec{M}$ spirals down from the z-axis with a fixed magnitude. An important equation to consider is the Larmor relation:

$$\omega = \gamma|\vec{B}| \tag{2.13}$$

This relationship states that any magnetization that is transverse (orthogonal) to an applied field will precess at a frequency ω that is proportional to the magnitude of the applied field. Let $t=0$ be the time the excitation pulse is complete and let $t>0$ be time during the recovery of the longitudinal magnetization. The magnitude of M_{xy} decreases exponentially with time T_2^* called the relaxation factor. The transverse magnetization is given by:

$$M_{xy}(\vec{r},t) = M_{xy}(\vec{r},0)e^{-\frac{t}{T_2^*(\vec{r})}}e^{-i\gamma\int_0^t B_{1,z}(\vec{r},s)ds} = M_{xy}(\vec{r},0)e^{-\frac{t}{T_2}}e^{-i\omega_0 t} \tag{2.14}$$

where $M_{xy}(\vec{r},0)$ is the magnetization along the transverse plane immediately after an RF pulse. This value is used to form an image in MRI and depends on the spin density and the RF pulse used to flip $\vec{M}$. The longitudinal magnetization M_z is given by

$$M_z(\vec{r},t) = M_z^0\left(1-e^{-\frac{t}{T_1}}\right) + M_z(\vec{r},0)e^{-\frac{t}{T_1}} \tag{2.15}$$

where M_z^0 is the longitudinal magnetization at thermal equilibrium, $M_z(\vec{r},0)$ is the longitudinal magnetization immediately after an RF pulse, and T_1 is the time required for the regrowth of $M_z(\vec{r},0)$ to M_z^0.

2.3.4. THE MR SIGNAL

Magnetic flux through the coil is given by

$$\Phi(t) = \int \vec{B}_r(\mathbf{r}) \cdot \vec{M}(\mathbf{r}, t) d\mathbf{r} \tag{2.16}$$

where $\vec{B}_r(\mathbf{r})$ is the laboratory frame magnetic field at location $\mathbf{r} = (x, y, z)$. By Faraday's law of induction, the induced voltage signal in the coil is

$$V(t) = -\frac{\partial \Phi(t)}{\partial t} = -\frac{\partial}{\partial t} \int \vec{B}_r(\mathbf{r}) \cdot \vec{M}(\mathbf{r}, t) d\mathbf{r} \tag{2.17}$$

where

$$\vec{B}_r = B_{r,x} \vec{i} + B_{r,y} \vec{j} + B_{r,z} \vec{k} \tag{2.18}$$

$V(t)$ is a high frequency signal, which can cause problems in later processing stages. Using signal demodulation, $V(t)$ is moved to a low-frequency band. This results in the MR signal $S(t)$ given by:

$$S(t) = \int \omega(\mathbf{r}) M_{xy}(\mathbf{r}, 0) e^{-t/T_2(\mathbf{r})} e^{-i\Delta\omega(\mathbf{r})t} d\mathbf{r} \tag{2.19}$$

A more thorough derivation of the signal S can be found in Appendix A.

3

NON-CONVENTIONAL: LOW FIELD MRI

In Uganda, many children with hydrocephalus do not have access to an MRI scan, which is a crucial clinical tool for imaging to identify which spaces within the brain are enlarged. The diagnosis of hydrocephalus does not require an expensive MRI scan for high resolution images. A low-field MRI scan can aid surgical treatment with high enough resolution. However, the development of a low-field prototype MRI scan, faces several challenges: permanent magnet design, spatial encoding without switching gradients and image reconstruction. This Chapter explains the low-field MRI prototype developed by TU Delft and LUMC.

3.1. MAGNET DESIGN

One of the hardest challenges in making the low-field MRI prototype is the design of the magnet that creates the homogeneous B_0 field in one direction. In Figure 3.1 a configuration of the axes in low-field MRI is given.

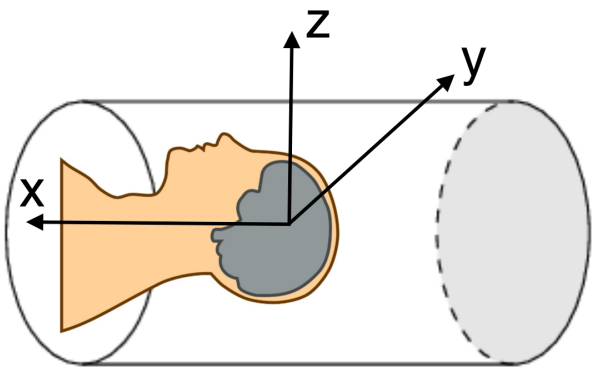
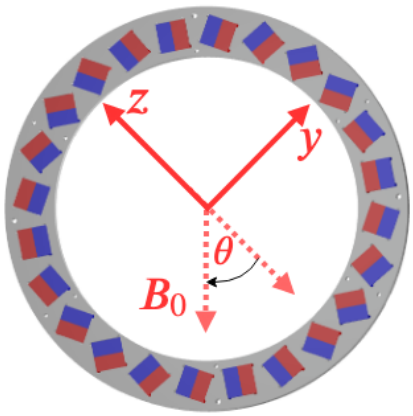


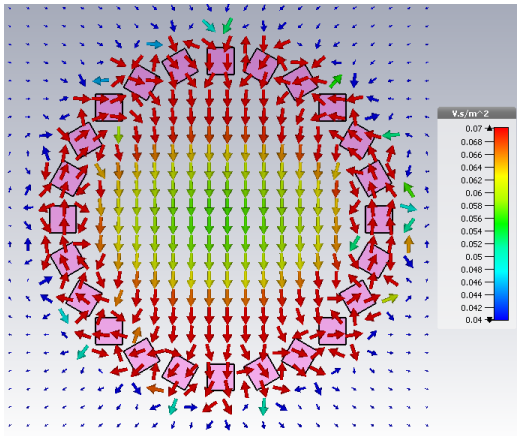
Figure 3.1: Configuration of the axes in low-field MRI.

Based on literature [15], the decision was made to use permanent magnets in a Halbach array to create B_0. In the 1980s, the Halbach array was invented by physicist Klaus Halbach to focus particle accelerator beams. For the low-field prototype a circular Halbach array is used. In theory, the circular Halbach array is an infinitely long cylinder made of magnetic material where the direction of the magnetization continuously varies. This configuration creates a strong magnetic field inside the cylinder while cancelling the field outside the cylinder.

The circular Halbach array used for the low-field prototype contains 24 1-inch cube NdFeB N52 permanent magnets which creates a fairly homogeneous B_0 that is oriented in the z-plane over an angle θ, see Figure 3.2b. The remanent flux density, B_r, of the permanent magnets is between 1450 – 1480 mT.



(a) Configuration of 24 1-inch cube permanent magnets. The B_0-field direction depends on angle θ.



(b) Simulation of the magnetic field of the Halbach array.

Figure 3.2: Circular Halbach array used in the low-field prototype.

To measure an object with a certain height x, four circular Halbach arrays are stacked together, see Figure 3.3.

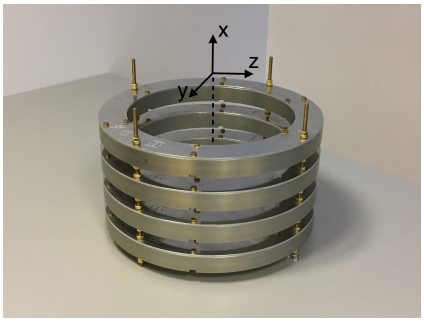
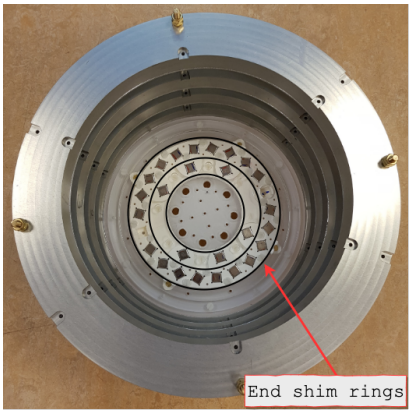
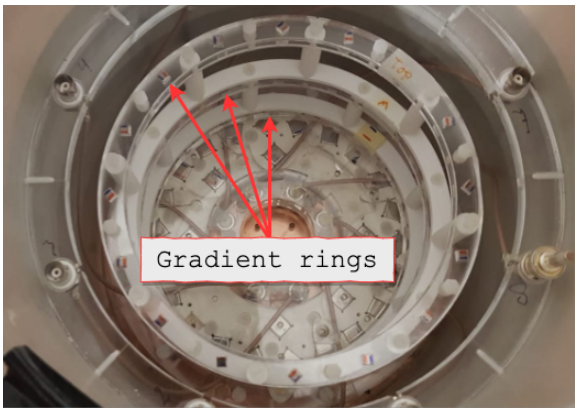


Figure 3.3: Four ring Halbach array.

The intensity of the magnetic field is the largest in the $z-$direction, this causes field inhomogeneities. To increase homogeneity in the $z-$direction two extra rings of smaller magnets are placed beneath the array, this process is called magnet shimming see Figure 3.4a. The outer ring consists of 20 magnets and the inner ring of 6 magnets. The magnets are 12 mm NdFeB N48 cubes with a B_r between $1370-1420$ mT.



(a) End shim rings.



(b) Gradient rings.

Figure 3.4: Extra magnet rings placed inside the Halbach array for shimming.

After the shimming process, B_0 is fairly homogeneous. The magnet configuration also creates a magnetic field in the $x-$direction with values in the order of 10^{-16} mT. Therefore, the magnetic field in the $x-$direction is neglected.

Due to B_0 being fairly homogeneous, we will not be able to resolve spatial difference. To obtain variations in the field, three rings of smaller magnets are placed inside the four ring Halbach array, see Figure 3.4b. Each ring contains 16, 5 mm NdFeB N42 cubes with a B_r between $1290-1320$ mT.

The field strength of the scan is measured for the Halbach array with the shimming rings and the gradient rings. For every 5 mm the magnetic flux density is measured in the $x-$, $y-$, and $z-$direction. A more detailed description of the low-field prototype and the magnets can be found in [16] [17].

The bore of the magnet lies along the x-axis. Measured from the bottom of the scan, the center of the magnet is assumed to be located at $x = 247$ mm which is 3 mm above the surface of the phantom placed in the RF coil. Starting at $x = 232$ mm, B_0 is measured for $x_i = 232 + i * 5$ mm with $i \in \{0,1,2,3,4,5,6\}$. We use the B_0-field for $x = 247$ mm, this position corresponds to the bottom of the phantom placed in the scan for measurements. These objects are filled sunflower oil until 3 mm from the top, this because of variations in B_0 in the $x-$direction.

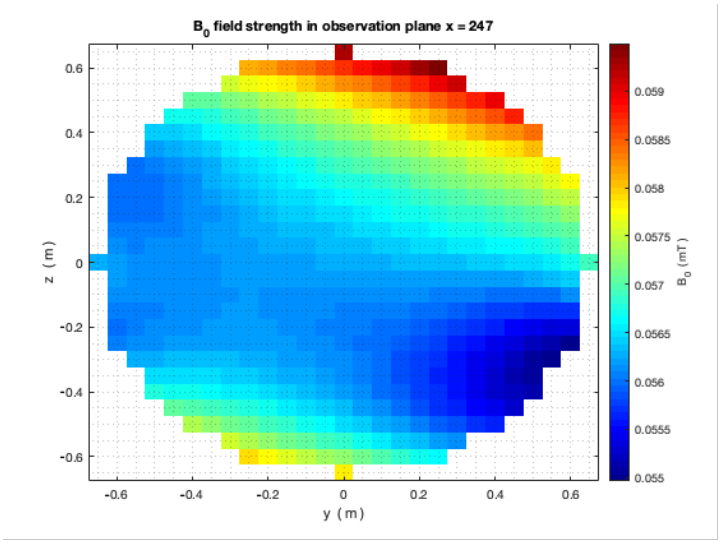


Figure 3.5: Field strength of the four ring Halbach array with the gradient and shimming rings. The field of view (FOV) in the $z-$direction is set to be 0.09 m

The measured signal strength at $x = 247$ mm is given in Figure 3.5. We will use this field to generate simulated signal from images. Note that the magnetic field is symmetrical, which means that different locations in the scan correspond to the same resonance frequency. To solve this, an outer gradient ring is added around the MRI scan to create a linear monotonic field, see Figure 3.6. This, to make sure that the resonance frequency varies linearly with the strength of the field. For the remainder of this research, the MRI scan without the outer gradient ring is used.

Figure 3.6: Outer gradient ring containing 36 1-inch cube NdFeB N52 magnets with B_r between $1450-1480$ mT.

3.2. THE SIGNAL MODEL

To derive the signal model for the low-field MRI, we start with the general signal expression $S(t)$ derived in the previous Chapter:

$$S(t) = \int \omega(\boldsymbol{r}) M_{xy}(\boldsymbol{r},0) e^{-t/T_2^*(\boldsymbol{r})} e^{-i\Delta\omega(\boldsymbol{r})t} d\boldsymbol{r} \tag{3.1}$$

The quality of magnetic resonance images is highly dependent upon the coil used to receive the radio frequency signal emitted from tissue. Therefore, the coil sensitivity $c(\boldsymbol{r})$ is added to (3.1) [18]:

$$S(t) = \int c(\boldsymbol{r})\omega(\mathbf{r}) M_{xy}(\mathbf{r},0) e^{-t/T_2^*(\mathbf{r})} e^{-i\Delta\omega(\boldsymbol{r})t} d\mathbf{r} \tag{3.2}$$

In general, the coil sensitivity decreases with distance from the coil, which can cause spatial variations in the signal strength. If uncorrected, these variations can be a challenge for image reconstruction methods. The relationship between the transverse magnetization $M_{xy}(\boldsymbol{r},0)$ and the spin density $\rho(\boldsymbol{r})$ is given by:

$$M_{xy}(\boldsymbol{r},0) = \frac{\gamma\hbar^2}{4k_B T}\rho(\boldsymbol{r})\omega(\boldsymbol{r}) \tag{3.3}$$

where $k_B = 1.381 \cdot 10^{-23}$ m^2 kg s^2 K^{-1} is the Boltzmann constant and T is the temperature. The spin density is the concentration of signal bearing spins and is therefore a direct measure of the tissue type. When considering an high field MRI, the B_0 field is homogeneous and therefore $M_{xy}(\boldsymbol{r},0)$ and $p(\boldsymbol{r})$ only differ with a constant value. For the low field problem we substitute (3.3) into (3.2) and absorb the constants into $c(\boldsymbol{r})$:

$$S(t) = \int c(\boldsymbol{r})\omega^2(\boldsymbol{r})\rho(\boldsymbol{r}) B^*_{r,xy}(\boldsymbol{r}) e^{-t/T_2^*(\boldsymbol{r})} e^{-i\Delta\omega(\boldsymbol{r})t} d\boldsymbol{r} \tag{3.4}$$

The decay rate T_2^* can be ignored due to spatial variations in B_0 in the low-field setting:

$$S(t) = \int c(\boldsymbol{r})\omega^2(\boldsymbol{r})\rho(\boldsymbol{r}) e^{-i\Delta\omega(\boldsymbol{r})t} d\boldsymbol{r} \tag{3.5}$$

Measurements from an MRI scan consists of noisy samples of the signal. These noisy samples are given by adding a measurement error to Equation (3.5) [5]:

$$b_i = S(t_i) + e_i \qquad i = 1,\ldots,L \tag{3.6}$$

Here, b_i denotes the i^{th} noisy sample of the signal, L is the number of time samples, and e_i is the i^{th} measurement error. Usually, the t_i values are equally spaced, and often the signal is strong at certain time values due to magnetization phases. The measurement errors are modelled by white Gaussian noise.
Traditionally, from raw measurements MR images are reconstructed by a simple inverse 2D or 3D fast Fourier transform (FFT). For low-field MRI, the inverse problem is ill-posed and FFT is inadequate. To perform image reconstruction, spatial variations in the B_0-field can be used. However, if the variations are too small, the signal contains little spatial information. On the other hand, if the variations are too large, T_2^* cannot be neglected and the FOV becomes too small.

Based on literature [5], model-based image reconstruction is an adequate alternative for image reconstruction. First, the spin density $\rho(\boldsymbol{r})$ is approximated by using a finite series expansion as follows:

$$\rho(\boldsymbol{r}) = \sum_{j=1}^{N} x_j \phi(\boldsymbol{r} - \boldsymbol{r}_j) \tag{3.7}$$

where x_j denotes the coefficients, $\phi(\cdot)$ is the object basis function, and r_j is the center of the jth translated basis function. Usually, rectangular basis function are used and therefore N denotes the number of pixels. Substituting the basis expansion (3.7) into the low-field signal model (3.6) and simplifying yields:

$$\begin{cases} S(t_i) = \sum_{j=1}^{N} a_{ij} x_j \\ a_{ij} = \int \phi(\boldsymbol{r} - \boldsymbol{r}_j) c(\boldsymbol{r})\omega^2(\boldsymbol{r}) e^{-i\Delta\omega(\boldsymbol{r})t_i} d\boldsymbol{r} \end{cases} \tag{3.8}$$

Usually, the basis functions are highly localized, so "center of pixel" approximation is used:

$$a_{ij} = c(\boldsymbol{r_j})\omega^2(\boldsymbol{r_j}) e^{-i\Delta\omega(\boldsymbol{r_j})t_i} \Delta x \Delta y \Delta z \tag{3.9}$$

where $\Delta x \Delta y \Delta z$ is the voxel size. In other words, $\Delta y \Delta z$ is the pixel size and Δx is the slice thickness. Combining Equations (3.6) and (3.8) results in the following system of equations:

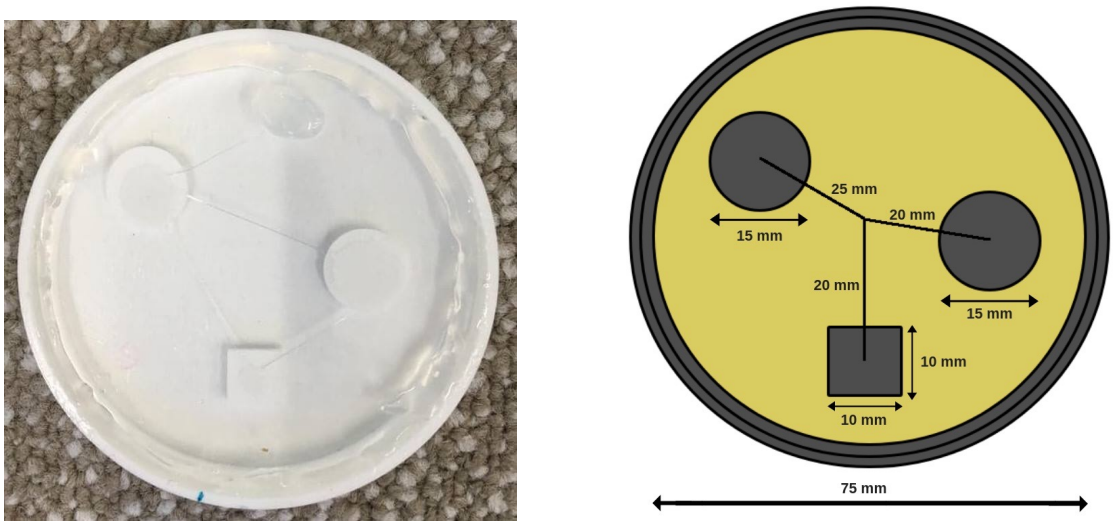
$$\boldsymbol{b} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{e} \tag{3.10}$$

Here, the elements of A are given by Equation (3.9). The RF system used in the prototype is a solenoid RF-coil, which serves as both the transmitter and the receiver. After sending out RF-pulses the solenoid coil creates a fairly homogeneous B_1-field. In the prototype there are no switching linear gradients to allow for spatial encoding. Therefore, conventional two-dimensional Fourier transform image reconstruction cannot be used as the method relies on a static, strong and homogeneous magnetic B_0-field and high strength linear spatial encoding magnetic fields (SEMs). The variations in the B_0 field can be used for spatial localization. However, the B_0-field is symmetric and so it is impossible to determine the contribution of each pixel to the signal from just a single measured signal.

Based on literature [19], two-dimensional imaging in low-field MRI is possible without gradient coils. The inhomogeneity of B_0 in the prototype can be used for spatial encoding, called the encoding technique of rotating spatial encoding magnetic fields (rSEMs). The magnet is rotated several times with an angular increment θ around a sample. At each increment, the field experienced by the sample changes due to the inhomogeneity of B_0. With each rotation new information is provided.

3.3. RESULTS

In order to ensure that the low-field MRI prototype and the imaging methods work correctly, imaging phantoms are used. The word phantom may evoke scary thoughts related to ghosts, shims, or illusions, but in the medical world phantoms are medical devices used to analyze, evaluate and tune the performance of imaging devices. To test the low-field MRI prototype of the LUMC, the phantom from Figure 3.7a is used.



(a) The phantom.

(b) Illustration of the phantom.

Figure 3.7: Phantom used by the LUMC.

In the phantom of Figure 3.7a different areas are filled with air and oil. This is more clearly shown in the illustration of the phantom given in Figure 3.7b, where the gray areas indicate air and the yellow area indicates oil. The signals generated from the phantom and super-resolution reconstruction (SRR) are used to produce an image with higher resolution. The results obtained[1] are given in Figure 3.8. The darker areas correspond to the two circles and the square filled with air of the formerly described image. Air does not give a signal, so these areas are dark in the final image. It is clear that standard model based reconstruction techniques do not yield in images good enough to use in a clinical setting.

[1] The results were obtained by research done by M. L. de Leeuw den Bouter.

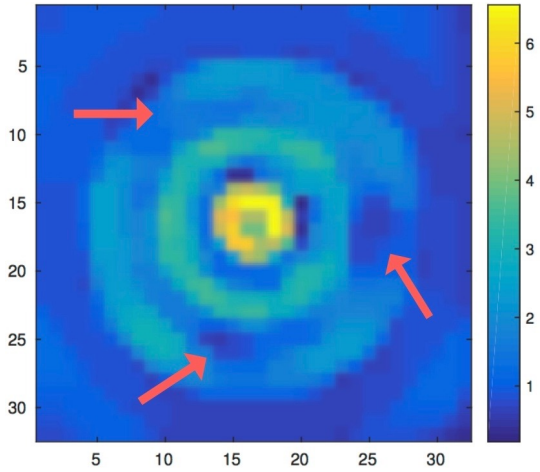


Figure 3.8: High resolution image from the LUMC phantom generated using SRR. The red arrows indicate the two circles and square.

4

DEEP LEARNING

Artificial Intelligence (AI) is the study and creation of machines that can perform tasks that would require intelligence if a human were to do the same job. AI is a blend of many sciences: mathematics, computer science, robotics, neurology, and many more. Each new idea in AI sparks another idea just like the limbs of a tree. The roots of this constantly changing field are planted by scientists who could imagine all the possibilities.

Over a hundred years ago, before the first computer was built, people already wondered whether one day machines might be intelligent. In 1844 mathematician and writer Countess Augusta Ada King wanted to create a mathematical model for how the brain gives rise to thoughts and nerves to feelings ("a calculus of the nervous system") [20]. Even though Ada did not achieve this she is often regarded as the first computer programmer in history. She was the first to recognize the application of machines beyond calculation [21].

Two strategies for prediction and problem solving are logic and intuition. Logical problem solving involves problems that are described by a list of formal and mathematical rules. Although this is a difficult mental undertaking for a human being, it is an easy task for a computer. The real challenge for AI lies in solving intuition related tasks, such as recognizing certain objects in an image, or sarcasm in spoken words. These tasks are difficult for computers to solve, but relatively straightforward for human beings [22].

The capability of computers tackling problems using real world knowledge and make 'subjective' decisions is called Machine Learning (ML), which is a sub-field of AI. The name ML was coined by computer scientist Arthur Samuel in 1959 [23]. Samuel defined ML as the field of study that gives computers the ability to learn without being explicitly programmed. A more recent and widely quoted definition is given by Tom Mitchell in 1997: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E" [23].

ML is a method for statistical learning where each sample in the input data set is described by human-defined features. As a result, the performance of ML algorithms highly depend on the representation of the data they are given. For example, in Figure 4.1 a representation of the data is displayed in cartesian and polar coordinates. If we want to separate the two classes of data by a linear line, the task becomes simple when the data is represented in polar coordinates. It is clear that the choice of representation has a huge effect on the performance of ML algorithms.

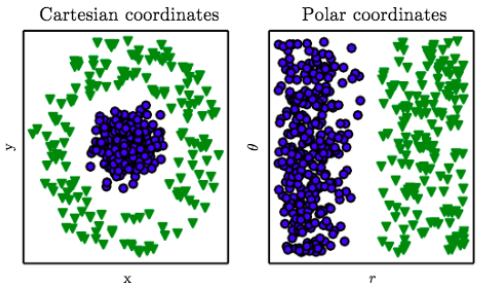


Figure 4.1: Data representation. [22]

For many AI tasks extracting the right features is difficult. For example, suppose we want to design a program that detects cats in pictures. A computer interprets an image in a different way than we humans do. For a computer a color image is a three-dimensional array of pixel values that define the red, green, and blue color components for each pixel, see Figure 4.2.

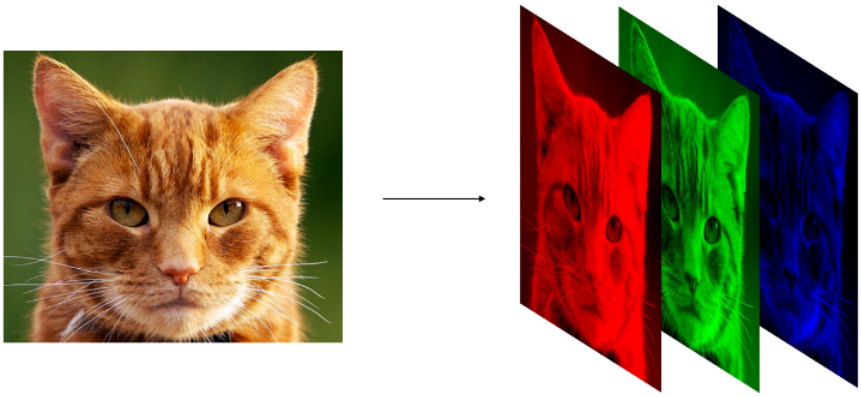


Figure 4.2: RGB image of a cat. [24]

We know that cats have whiskers, tails, fur textures, and so forth, so we might use these as features. However, each of the proposed features is difficult to describe in terms of individual pixel values. Therefore, the features should be described in configurations of groups of pixels rather than in individual pixels. Now let us create a simple algorithm that tells the computer what a cat exactly is using simple geometric shapes: if there is a circle with radius r_1, if there are two circles with radii $r_2 < r_1$, and if there are two triangles on top of the circle with radius r_1 measuring 45 degrees from the center of the circle, then the image is that of a cat, see Figure 4.3.

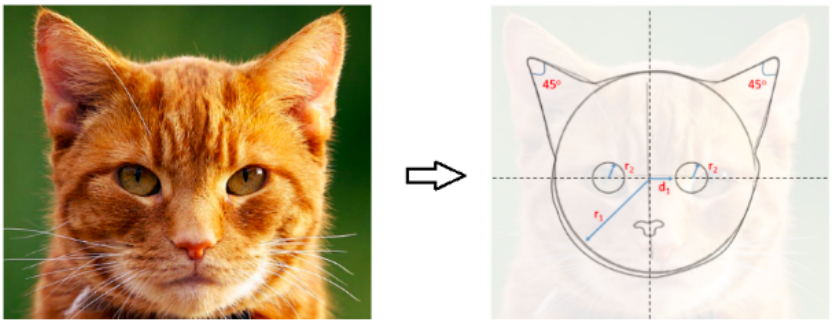


Figure 4.3: Classifying a cats face in an image using simple predescribed geometric shapes. [24]

Our classifier will clearly fail in a wide variety of situations. For instance, when applied to an image where only a small part of a cat is showing. One way to tackle this problem is to code up all possible orientations, sizes, shapes, and so forth for each cat in the world, which is an impossible task.

The problem of extracting abstract higher-level features from group of pixels without human intervention is solved by Deep Learning (DL) [25], a sub-field of ML, see Figure 4.4. DL solves the problem by allowing computers to learn and improve from experience and data. DL is an approach to AI that represents the world in terms of a nested hierarchy of concepts [22]. The computer learns difficult concepts based on simpler concepts using the so-called hidden layers. These hidden layers can extract increasingly abstract features, see Figure 4.5. Here, pixels are mapped to an object identity. The input layer is the visible layer, because it consists of variables that we can perceive, namely the pixels. The hidden layers extract increasingly abstract features. For example, edges are used to define corners and contours which are parts of higher-level objects. The output layer is created by a simple classifier, which identifies the objects in the input image: a car, a person or an animal.

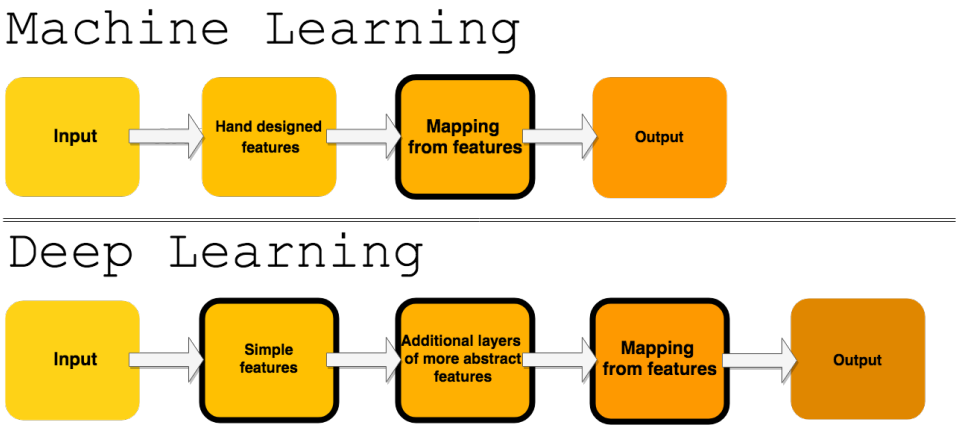


Figure 4.4: Flowcharts of how ML and DL work. The boxes with a black edge indicate components that are able to learn from data without human intervention.

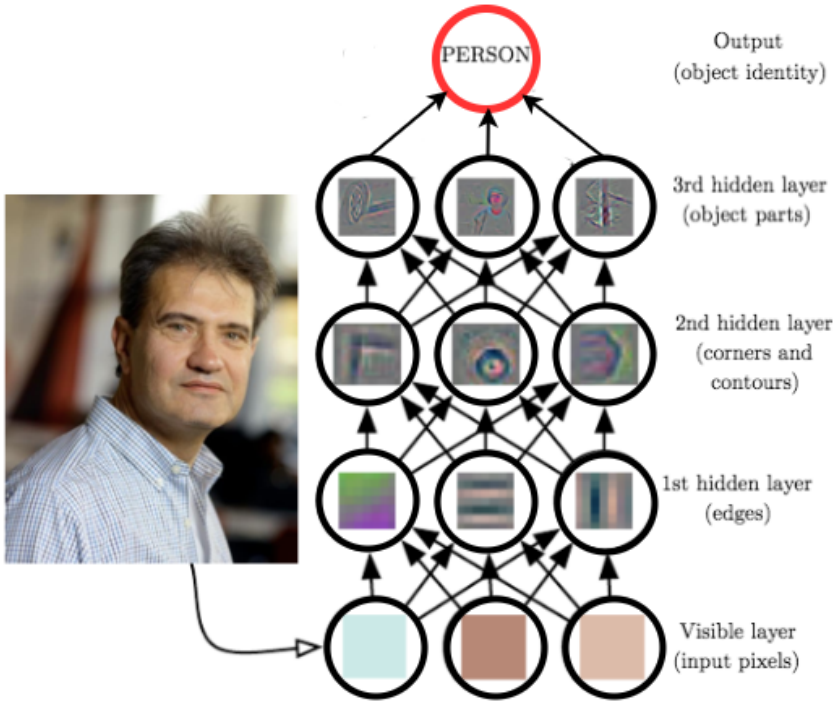


Figure 4.5: Example of a DL application. [22] [26]

DL is a fast-growing field that models high-level patterns in data as complex multilayer networks. In recent years a variety of neural network types have been studied, including fully-connected networks and convolutional neural networks. Companies such as Google and Microsoft use these type of networks to solve problems in areas such as image recognition and natural language processing. However, the models used for these tasks are complex and not easy to understand. Therefore, if we want to consider a DL approach for image reconstruction in low-field MRI, we must first cover the basic mechanics of DL. The core components of any DL problem are [22]:

1. The learning problem and the data set that we can learn from.
2. A model that transforms the data.
3. A performance measure to quantify how well our model is doing.
4. A learning algorithm that adjusts the parameters of the model to minimize the performance measure.
5. Validation and regularization techniques to measure and control model performance.

In the remainder of this Chapter, each component is discussed based on the first and simplest type of artificial neural network known as a multi-layer perceptron or a feedforward neural network. In addition, two extremely powerful neural network architectures are briefly discussed: the convolutional neural network and generative adversarial networks.

4.1. THE MULTI-LAYER PERCEPTRON

An artificial neural network (ANN) is an information processing system that is inspired by the way biological nervous systems, such as the brain, process information. An ANN contains a large number of highly interconnected processing artificial neurons, nodes or units. The aim of ANNs is to find regularities and patterns within the data by itself. One of the simplest ANN architectures is the perceptron [2], see Figure 4.6.

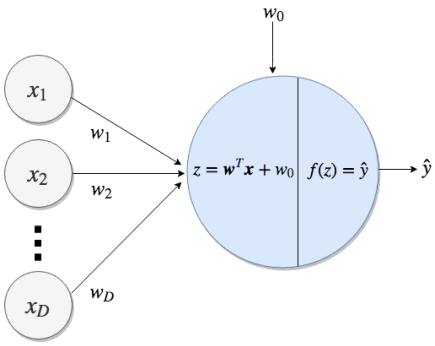


Figure 4.6: Visualisation of a perceptron.

The perceptron consists of an input layer having input units and an output layer having a single unit. The input and output are interconnected by modifiable weights and the output unit is connected to a bias. Formally, the perceptron is defined by:

$$\hat{y}(\boldsymbol{x}, \Theta) = f(\underbrace{\boldsymbol{w}^T\boldsymbol{x} + w_0}_{z}) \tag{4.1}$$

where $\boldsymbol{x} = \{x_i\}_{i=1}^{D}$ is the input vector, $\hat{y}$ is the output, $\Theta = \{\boldsymbol{w}, w_0\}$ is the parameter set containing the weights $\boldsymbol{w} = \{w_i\}_{i=1}^{D}$ and the bias w_0. The quantity z is known as the activation and is transformed using a differentiable, nonlinear activation function $f(\cdot)$. The perceptron can only be used for the classification of patterns that are linearly separable. This limitation has led to the development of the multi-layer perceptron (MLP), which serves for both non-linear function approximation and non-linear classification tasks [27]. A MLP also known as a feedforward artificial neural network (FNN) is an interconnection of perceptrons where data and calculations flow in one direction, from input to output. A $(L+1)-$layer FNN consists of an input layer, an output layer, and L hidden layers. The input layer is not counted as a hidden layer as no real processing takes place. The number of layers is called the depth and the number of units in a specific layer is called the width. The output in the i^{th} neuron within a layer l is given by:

$$y_i^{(l)} = f_i^{(l)}\left(z_i^{(l)}\right) \quad \text{with} \quad z_i^{(l)} = \sum_{j=1}^{m^{(l-1)}} w_{ij}^{(l)} y_j^{(l-1)} + w_{i0}^{(l)} \tag{4.2}$$

Here, $w_{ij}^{(l)}$ denotes the weighted connection between the i^{th} unit in layer l and the j^{th} unit in layer $(l-1)$, $w_{i0}^{(l)}$ denotes the bias associated with unit i within layer l, and $y^{(0)}$ is the input. The input layer has $D = m^{(0)}$ inputs and the output layers has $C = m^{(L+1)}$ targets. The bias parameters can be absorbed into the set of weight parameters by introducing a dummy unit $y_0^{(l)} = 1$ in each layer:

$$y_i^{(l)} = f_i^{(l)}\left(z_i^{(l)}\right) \quad \text{with} \quad z_i^{(l)} = \sum_{j=0}^{m^{(l-1)}} w_{ij}^{(l)} y_j^{(l-1)} \tag{4.3}$$

For an illustration of the MLP see Figure 4.7.

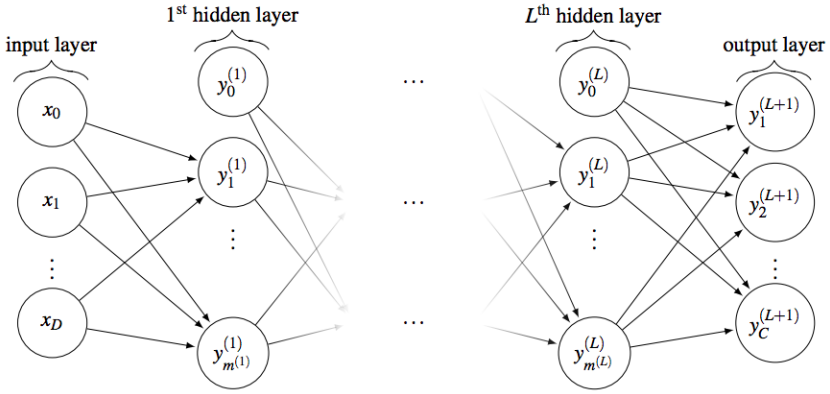


Figure 4.7: A $(L+1)$-layer perceptron. [**?**]

Activation functions are used to control the networks outputs. Across various different domains for instance self-driving cars, cancer detection systems, weather forecast, and many more it has been validated that a correct choice of activation function improves the results in neural network computing [28]. Generally, a distinguish is made between saturated and non-saturated activation functions. Common used saturated activation functions include the logistic sigmoid function:

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{4.4}$$

and the tangent hyperbolic function:

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

Graphs of both functions are given in Figure 4.8. For the training of multi-layer neural networks the tanh function is the preferred function as it gives better training performance compared to the sigmoid function. However, both activation functions suffer from vanishing gradients during backpropagation, see Section 4.1.2. This limitation led to further research into other activation functions and led to the birth of the rectified linear unit (ReLU) [22], see Figure 4.9, which is a non-saturated activation function:

$$f(x) = \max(0,x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases} \tag{4.6}$$

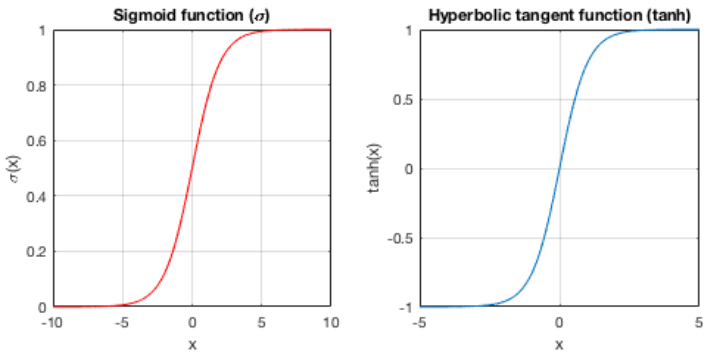


Figure 4.8: Left: The sigmoid activation function squeezes the real numbers to a range between [0,1]. Right: The tanh activation function squeezes the real numbers to a range between [-1,1].

ReLU has a strong biological and mathematical underpinning [29]. It is the most widely used activation function and has achieved state-of-the-art results in many deep learning applications. Compared to the sigmoid and tanh activations functions ReLU offers better performance and overall faster computation of neural networks on complex and high-dimensional data since it does not compute divisions and exponentials [30] [31] [32]. Another advantage of ReLU is that it introduces sparsity in the hidden layers of the neural network. Furthermore, there is no saturation in ReLU which means that we have efficient backpropagation without

vanishing or exploding gradients. These advantages combined make the ReLU a good choice for optimizing deep neural networks. However, ReLU suffers from a major drawback in that it easily overfits compared to the sigmoid function. This limitation has been reduced by using a regularization technique called dropout, see Section 4.1.4. Another significant limitation is that ReLU units in a network can be fragile during training. When training a network the networks parameters are optimized for a specific task using backpropagatian. This technique uses gradient descent in order to update the networks weights. For negative inputs ReLU units have zero output and as a consequence zero derivatives. If certain weights in the network lead to negative inputs this causes the so-called 'dead neurons' or 'dead ReLUs' as they do not contribute to the training of the network [22]. An approach to this problem is to set a small learning rate during the parameter update in gradient descent, see Section 4.1.1. Another approach to resolve the dead neuron issues is applying parametric ReLU (PReLU):

$$f(x) = \mathbb{1}_{(x<0)}(\alpha x) + \mathbb{1}_{(x\geq 0)}(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \alpha x, & \text{if } x < 0 \end{cases} \tag{4.7}$$

where α introduces a learnable small negative slope to the ReLU function. This prevents the gradients to become zero during the entire training process [33]. We have mentioned some of the most common used activation functions in neural networks. However, there are many more activation functions one may encounter in practice [28].

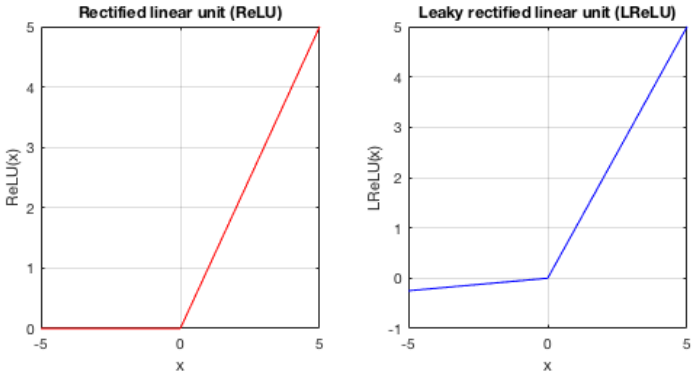


Figure 4.9: Left: The ReLU activation function. Right: The LReLU activation function.

4.1.1. PERFORMANCE AND OPTIMIZATION

After setting the hyperparameters that define the networks structure (e.g. depth and width) the next step is to optimize the networks weights. This is done by setting hyperparameters that determine how the network is trained (activation function, learning rate). The most common learning paradigms to train neural networks include supervised learning and unsupervised learning. The focus in this study is on supervised learning, since it is the problem with most structure. Supervised learning is the ML task of learning a mapping from inputs $\boldsymbol{x}$ to targets $\boldsymbol{y}$ given a labeled set of input-target pairs $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{N}$. Here $\mathcal{D}$ is called the training set, and N is the number of training samples.

Supervised training is accomplished by minimising some cost function which can be interpreted as an error measure E by iteratively adjusting the networks parameters to make the calculated output $\hat{\boldsymbol{y}}$ more similar to the target $\boldsymbol{y}$ [22] [34].

There are various error measures which can be considered depending on the particular application [35]. Common choices include the sum-of-squared error measure:

$$E(\boldsymbol{w}) = \sum_{n=1}^{N} E_n(\boldsymbol{w}) = \sum_{n=1}^{N} \sum_{k=1}^{C} (\hat{y}_k(\boldsymbol{x}_n, \boldsymbol{w}) - y_{n,k})^2 \tag{4.8}$$

and the cross-entropy error measure:

$$E(\boldsymbol{w}) = \sum_{n=1}^{N} E_n(\boldsymbol{w}) = -\sum_{n=1}^{N} \sum_{k=1}^{C} y_{n,k} \log(\hat{y}_k(\boldsymbol{x}_n, \boldsymbol{w}))) \tag{4.9}$$

Here, $y_{n,k}$ denotes the k^{th} entry of the target $\boldsymbol{y}_n$ associated with input $\boldsymbol{x}_n$ and $\boldsymbol{w}$ is the weight vector containing the weights and biases. The necessary criterion for minimisation is given by:

$$\frac{\partial E}{\partial \boldsymbol{w}} = \nabla E(\boldsymbol{w}) = 0 \tag{4.10}$$

where ∇E is the gradient. Due to the complexity of the error we cannot find an analytical solution to the equation $\nabla E(\boldsymbol{w}) = 0$. Therefore, we resort to iterative numerical procedures. Most techniques involve choosing some initial weight $\boldsymbol{w}[0]$ and update the weights accordingly after each iteration step τ:

$$\boldsymbol{w}[\tau + 1] = \boldsymbol{w}[\tau] + \Delta \boldsymbol{w}[\tau] \tag{4.11}$$

After each update, the gradient is re-evaluated for the new weight vector and the process is repeated. A simple method to minimise the error is called gradient descent (GD) [22], which is a first-order iterative optimization technique. The idea behind GD is to minimise the error measure by adjusting the weights, see Figure 4.10.

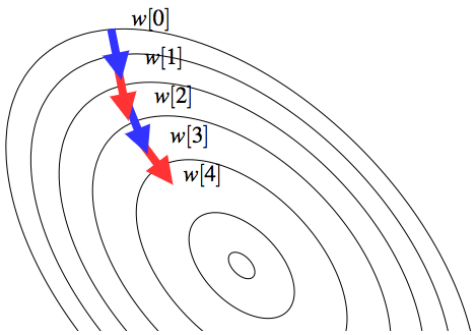


Figure 4.10: Gradient descent on a series of level curves.

This is done by taking a step in the direction of the steepest descent of the error measure, or in other words by adjusting the weights in the direction that shows the largest decrease in the error measure. The weight update in GD is given by:

$$\Delta \boldsymbol{w}[\tau] = -\eta \nabla E(\boldsymbol{w}[\tau]) \tag{4.12}$$

where $\eta > 0$ is the learning rate and indicates the relative size of the change in weights. The learning rate is a hyperparameter and much time is invested in finding and adjusting the value. A small learning rate can cost a huge, often unacceptable amount of time, given that in each update smaller changes are made to the weights. On the other hand, a large learning rate causes the model to converge very fast to a suboptimal solution and remain oscillating around the optimal solution without finding it. The optimal learning rate is one that gets fast learning while avoiding oscillation.

There are three variants of GD [36]: batch gradient descent, stochastic gradient descent, and mini-batch gradient descent. The difference between these variants is the amount of data used from the training set to compute the gradient for each parameter update, also known as the learning step. A trade-off is made between the accuracy of the parameter update and the time complexity of each update. Batch gradient descent (BGD) or vanilla gradient descent updates the parameters based on the overall error: $E(\boldsymbol{w}) = \sum_{n=1}^{N} E_n(\boldsymbol{w})$. BGD is guaranteed to converge to the global minimum for convex problems and to a local minimum for non-convex problems. However, training sets of many DL-related tasks contain an enormous amount of samples and scanning throughout the set before making an update can be a costly operation.

Unlike BGD, stochastic gradient descent (SGD) performs a parameter update for each training sample based on the error: $E_n(\boldsymbol{w})$. That is why SGD is much faster than BGD. A disadvantage of SGD is the high variance of different training samples and the frequent updates performed by SGD cause a heavily fluctuating cost function, see Figure 4.11. As a result, SGD keeps overshooting the minimum. By slowly decreasing the learning rate, SGD shows the same convergence behavior as BGD.
By combining BGD and SGD, we get the best of both worlds, namely mini-batch gradient descent. This is the algorithm of choice when training a neural network and often the term SGD is employed when mini-batches are used. Mini-batch gradient descent performs a weight update by processing a random subset (mini-batch) of M training examples. The weights are updated based on the cumulative error: $E_M(\boldsymbol{w}) = \sum_{n \in M} E_n(\boldsymbol{w})$. This

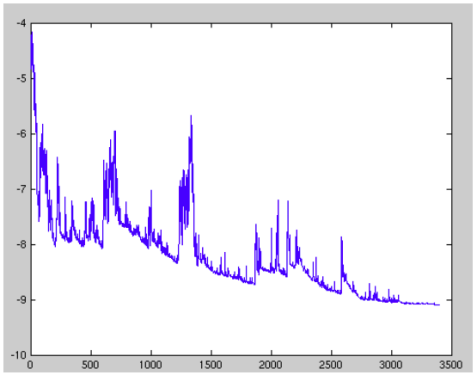


Figure 4.11: SGD fluctuation of the error measure. [36]

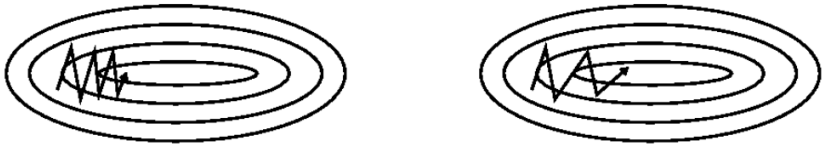
approach reduces the variance in the parameter updates, which can lead to a better and stable convergence. Common batch sizes range from 50 to 256, but this can vary for different applications. The number of times a mini-batch of data passes through a learning algorithm is known as the iteration step and the number of times a learning algorithm works through the entire training set is defined by the number of epochs.

Mini-batch gradient descent does not guarantee good convergence, because choosing a proper learning rate is difficult. In general, the optimal learning rate cannot be analytically calculated in advance [37]. Instead, a proper learning rate is estimated through trial and error. A good starting point is to use a default value of 0.1 or 0.01. Another approach is to grid search learning rates on a logarithmic scale from 0.1 to 10^{-5} or 10^{-6} [22].

Instead of using a fixed learning rate, a better approach is to decay the rate over time. This is known as the learning rate schedule or the learning rate decay. There are three common strategies for implementing the learning rate decay [22]:

- Step decay drops the learning rate by a factor every few epochs. Typically, the learning rate is reduced by half every five epochs, or by 0.1 every 20 epochs. The factor value depends on the type of problem and the model.
- Exponential decay is given by $\eta_\tau = \eta_0 e^{-k\tau}$ where η_0, k are hyperparameters and τ is the iteration step.
- $\frac{1}{\tau}$ decay is given by $\eta_\tau = \frac{\eta_0}{1+k\tau}$ where η_0, k are hyperparameters and τ is the iteration step.

In practice, step decay is preferred as no additional hyperparameter k needs to be set. A challenge when using learning rate schedules is that their hyperparameters must be predefined and therefore they cannot be adapted to the characteristics of the training set. Adaptive learning rate methods are an alternative to this problem. These methods monitor the performance of the model on the training set and adjust the learning rate accordingly. The most popular adaptive optimization algorithms include SGD with momentum (Figure 4.12), Adam and NAdam [22].



(a) SGD without momentum (b) SGD with momentum

Figure 4.12: To speed up the process of configuring the learning rate, Momentum can be added to the learning process. Momentum helps SGD to accelerate in the right direction and damps oscillations. [36]

4.1.2. BACKPROPAGATION

Supervised training algorithms generally involve an iterative process to minimize the error function, with adjustments to the networks weights being made in a sequence of iterations. At each iteration, a distinguish is made between two different stages. In the first stage, the derivatives of the error function with respect to the weights are evaluated. This is commonly done by error backpropagation or simply backprop [38], which enables ANNs to solve complex problems. Backprop is one of the simplest and most general learning algorithm for supervised training. The technique derives its name from the fact that partial derivatives of the error measure with respect to the weights of the network are determined by back propagating the error signals (computed by the output units) through the network, layer by layer. In general, backpropagation can be decomposed in the following four steps:

STEP 1: FORWARD PROPAGATION

Forward propagate the input value $\boldsymbol{y^0}$ through the network to find the activations of all the hidden units and output units by successive application of:

$$y_i^{(l)} = f_i^{(l)}(z_i^{(l)}) \text{ with } z_i^{(l)} = \sum_{j=0}^{m^{(l-1)}} w_{ij}^{(l)} y_j^{(l-1)} \tag{4.13}$$

STEP 2: ERRORS OUTPUT LAYER

The next step is to calculate the error of each unit in the output layer. This is done by subtracting the target value from the actual output for each output unit k:

$$\delta_k = \hat{y}_k - y_k \tag{4.14}$$

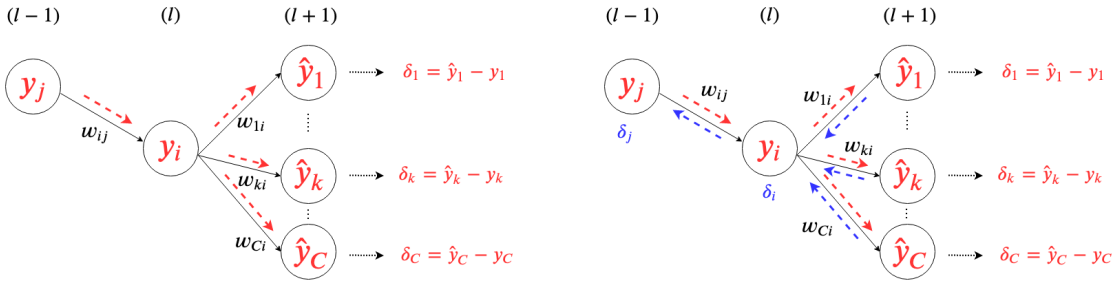
An illustration of step 1 and step 2 is given in Figure 4.13a.

STEP 3: BACKWARD PROPAGATION

The errors for the hidden units can be obtained by backpropagating the errors from units higher up in the network. This is done by using the backpropagation formula:

$$\delta_i^{(l)} = f'(z_i^{(l)}) \sum_{k=1}^{m^{(l+1)}} w_{ki}^{(l+1)} \delta_k^{(l+1)} \tag{4.15}$$

Backpropagation is illustrated in Figure 4.13b.



(a) Forward propagation

(b) Backward propagation

Figure 4.13: Left: Forward propagation indicated in red arrows starting from the hidden unit j in layer $(l-1)$ through unit i in layer (l) to the output units in layer $(l+1)$. After the forward propagation step the errors of each unit in the output layer are calculated. Right: Backward propagation indicated in blue arrows starting from the units in the output layer $(l+1)$ through unit i in layer (l) to unit j in layer $(l-1)$.

STEP 4: EVALUATE DERIVATIVES

The last step in the backpropagation algorithm is to evaluate the derivatives of the error measure with respect to the weights:

$$\frac{\partial E}{\partial w_{ij}^{(l)}} = \delta_i^{(l)} y_j^{(l-1)} \tag{4.16}$$

Details and a more thorough derivation of backprop can be found in [34].
In the second stage, the derivatives calculated in the fourth step of backprop are used to adjust the weights of the network. The simplest and most popular optimization procedure for weight adjustment is gradient descent. As already explained in the previous Section there are three variants of GD. Each variant in combination with backprop gives a different training algorithm. For instance, stochastic backprop is given by [39]:

Algorithm 1: Stochastic backpropagation

begin initialize network topology (depth, width), $\boldsymbol{w}$, criterion θ, η, $m \leftarrow 0$
  do $m \leftarrow m+1$
    $\mathbf{y}^0 \leftarrow$ random chosen input vector
    $w_{ij} = w_{ij} - \eta\delta_i y_j$
  until $\nabla E(\mathbf{w}) < \theta$
return $\boldsymbol{w}$
end

and mini-batch backprop is given by:

Algorithm 2: Mini-batch backpropagation

begin initialize network topology (depth, width), $\boldsymbol{w}$, criterion θ, η, $r \leftarrow 0$, mini-batch size n
  do $r \leftarrow r+1$
    $m \leftarrow 0$, $\Delta w_{ij} \leftarrow 0$,
    do $m \leftarrow m+1$
      $\mathbf{y}^0 \leftarrow$ random chosen input vector
      $\Delta w_{ij} = \Delta w_{ij} - \eta\delta_i y_j$
    until $m = n$
    $w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$
  until $\nabla E(\mathbf{w}) < \theta$
return $\boldsymbol{w}$
end

4.1.3. VALIDATION

Once training is complete, the question arises how well our learning algorithm performs on unseen data. Due to the problem of overfitting, see Section 4.1.4, the training set is not a good indicator of the predictive performance of the model on unseen data. Luckily, supervised learning algorithms involve data sets with labelled examples. If there is sufficient data, a general approach to measure the effectiveness of a learning algorithm is to shuffle the examples and split the data into three subsets: training, validation, and test [40]. The training set is usually the biggest and used to build a model. The validation and test sets are samples of the data held back from building a model. That is why these sets are also called the hold-out sets. For a large amount of data, both sets are roughly of the same size and much smaller than the training set.

During training the validation set is frequently used to evaluate the performance of the model through calculating the loss. Subsequently, the hyperparameters are fine-tuned based on the results on the validation set. After the training phase is over the test set is used for a final accuracy evaluation of the model. The test set is not used for learning or for any architectural or hyperparameter decisions.

The split ratio of the data depends on the total number of examples and the model that is being trained. In many applications the amount of data will be limited. In order to build a good model on would go for a larger training set. On the other hand, a small validation set will give a noisy estimate of predictive performance [34]. The common technique to this problem is to perform k-fold cross-validation, which is illustrated in Figure 4.14.

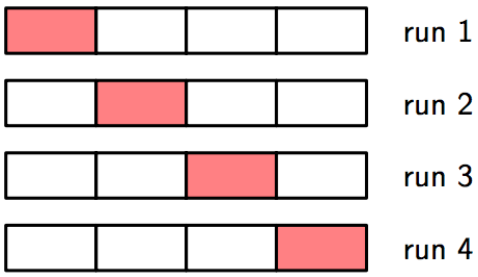


Figure 4.14: k-fold cross-validation illustrated for the case $k = 4$. After fixing the hyperparameters the data is randomly partitioned into k disjoint subsets (which are of equal size in a simple setting). $k-1$ groups are used to train the model and one is retained (red block) as a validation set. This process is then repeated until each subset is used as a validation set. After the model is evaluated for k runs, the performance results are averaged to give a final estimation of the models performance. [34]

4.1.4. REGULARIZATION

If the model is unable to capture the underlying pattern of the training data, we say that the model underfits or that the model has a high bias. The solution to underfitting is to increase the complexity of the model or to engineer features with a higher predictive power [41].

On the other hand, when the training error is low on the training set, but high on at least one of the hold-out sets, we say that the model overfits or that the model has a high variance. There are various solutions to prevent overfitting: a simpler model, dimensionality reduction of the data, more training data or regularization [41]. Regularization is the most commonly used method to prevent overfitting by forcing the learning algorithm to build a less complex model. This technique considerably reduces the variance, but also leads to a slightly higher bias, known as the bias-variance trade-off.

There are various regularization techniques that benefit neural networks: weight regularization, early stopping, dropout, batch normalization, and data augmentation [41].

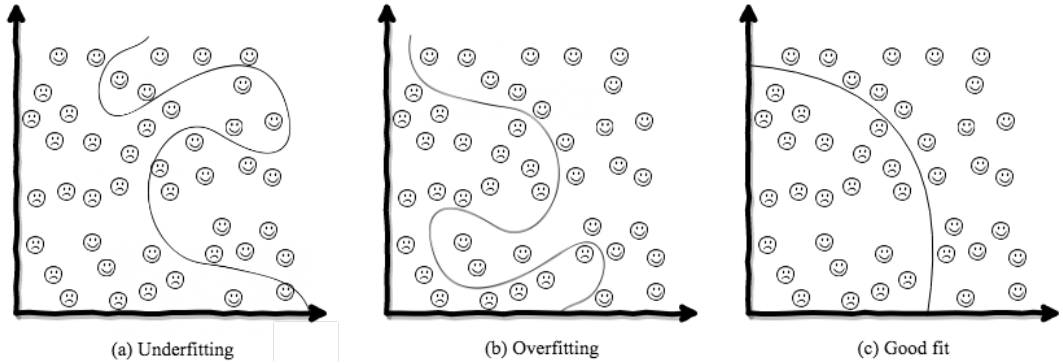


Figure 4.15: Graphs showing scenarios of underfitting, overfitting and just right separation.

WEIGHT REGULARIZATION

Weight regularization updates the learning algorithm to encourage the network to keep the weights small because large weights are a sign of a more complex model that overfits the training data [22]. To create a regularized model, a parameter norm penalty $\Omega(\boldsymbol{w})$ is added to the error measure which aims to control the complexity and form of the solution :

$$\hat{E}(\boldsymbol{w}) = E(\boldsymbol{w}) + \eta\Omega(\boldsymbol{w}) \tag{4.17}$$

Here, $\eta \in [0,\infty)$ is a hyperparameter that determines how much the weights are penalized. The two most common types of weight regularization are L_1-regularization and L_2-regularization. L_1-regularization, also known as lasso regression, defines the norm penalty as:

$$\Omega(\boldsymbol{w}) = ||\boldsymbol{w}||_1 \tag{4.18}$$

L_1 enforces sparsity of weights by making many weights equal to zero, so a feature selection is made by deciding which features are essential for making predictions and which are not [41]. This increases the explainability of the model. However, if the main goal is to improve the performance of the model on unseen data L_2-regularization, also known as ridge regression or Tikhonov regularization, usually produces better results. L_2-regularization defines the penalty norm as:

$$\Omega(\boldsymbol{w}) = ||\boldsymbol{w}||_2^2 \tag{4.19}$$

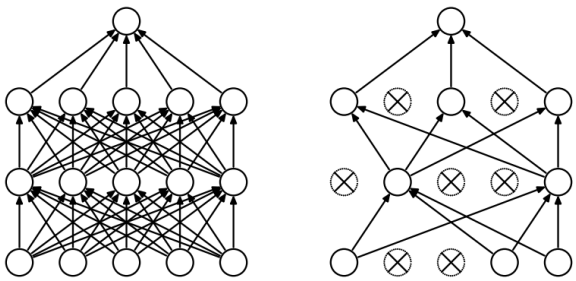
DROPOUT

Dropout is inspired by the role of sexual reproduction in evolution [42]. Sexual reproduction involves taking half of the genes of both parents, adding a hint of random mutation and combining both to produce offspring. On the other hand, asexual reproduction creates offspring that inherits a slightly mutated copy of the genes of only one parent. Intuitively one may assume that asexual reproduction is the better way to optimize individual fitness, because the set of genes that work well together is inherited directly by the offspring. Sexual reproduction will most likely to break co-adapted sets of genes, especially with larger sets. However, sexual reproduction is the way most advanced organisms developed. A possible explanation is that a criterion for natural selection is the mixing capacity of genes instead of individual fitness. The mix-ability of random sets of genes makes them more robust. A gene cannot rely on a large number of partner genes to always be present, it must learn something useful in collaboration with any set of other genes.

This idea of mix-ability is applied with dropout in neural nets. Dropout temporarily removes units (input and hidden) from the neural network, see Figure 4.16. The probability of retaining one unit is $p \in [0,1]$ independent of other units. For a wide range of networks and tasks, the optimal probability of retaining hidden units is close to $p = 0.5$ and for the input units, this value is closer to 1.

Dropout is implemented per layer in a neural network and samples a "thinned" version of the network consisting of all units that survived. A network with n units has a collection of 2^n possible thinned networks. For each training example a new thinned network is sampled and trained. After training dropout is not used to fit the network. However, the weights of the network will be larger because of dropout and just before finalizing the trained network the weights are scaled by the chosen dropout rate.

Like other regularization methods, dropout is more effective on problems with smaller datasets [22]. For very large datasets, dropout does not have much extra benefit and the computational cost of applying it can outweigh the benefit of regularization.



(a) Standard Neural Net (b) After applying dropout.

Figure 4.16: Dropout applied to a FNN with two hidden layers. The units that are crossed are dropped resulting in a thinned network. [43]

BATCH NORMALIZATION

Batch normalization is a technique that consists of standardizing the output of each layer before the units of the next layer receive them as input. In practice, batch normalization leads to faster and more stable training, as well as to some regularization.

EARLY STOPPING

The challenge of training a model is to train long enough to learn a mapping without overfitting. Early stopping is a technique used to prevent overtraining. When iteratively training a learning algorithm, the training error decreases steadily with the number of iterations. At a given moment, however, the model can start overfitting the training set and as a result the error on the validation set starts to rise. This overfitting can be prevented by stopping the training as soon as the error in the validation set reaches a minimum.

This may seem easy at first, but the ugly reality is that validation error curves always have more than one local minimum. This problem can be solved by using a stopping criterion [44].

DATA AUGMENTATION

Data augmentation is often used to regularize models that work with images. A synthetic image is made from the original image by applying different transformations that are likely to be found in unseen data.

4.2. CONVOLUTIONAL NEURAL NETWORK

When the training examples of an ANN are images, the input is very high-dimensional and a large number of parameters would be needed to characterize the network. Optimizing such large models is computationally intensive and because of the complexity of the model it is likely to overfit. The solution to these problems is a special type of FNN for processing data with a grid-like topology: the convolutional neural network (CNN) [45] [46]. CNNs are mainly used in the field of pattern recognition with images, which allows certain properties to be encoded in the architecture while drastically reducing the number of parameters required to set up the model. CNNs are composed by stacking three types of layers: convolutional layers, pooling layers, and fully-connected layers. These layers are grouped based on their functionality: the convolution and pooling layers together perform feature extraction and the fully connected layers maps the extracted features to an output, for an illustration see Figure 4.18.

In the remainder of this Section the building blocks of a CNN architecture are discussed. Because CNNs are mainly used to analyse images, we will explain the network using images as input.

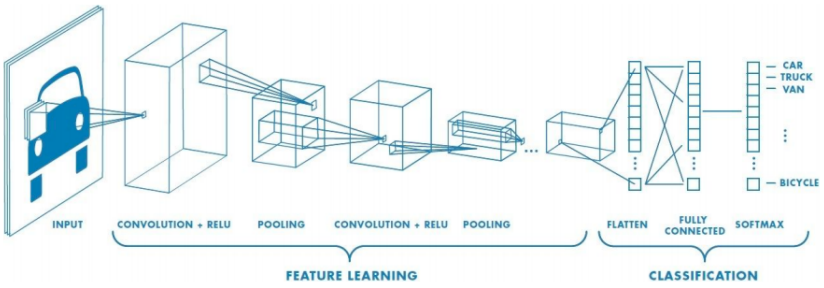


Figure 4.17: Illustration of a CNN architecture. [**?**]

4.3. CONVOLUTIONAL LAYER

The convolutional layer is the fundamental layer of a CNN that is used for feature extraction. The first layer in a CNN is always a convolutional layer that is used to extract the simple features from the input like lines, edges, and corners. The higher-level convolutional layers are used to extract higher-level features. This feature extraction is performed by a linear operator called convolution, where a kernel (filter), a set of weights, is systematically applied to the input (tensor) to create a feature map. Different kernels can be used to form multiple feature maps, each representing a different feature of the input. Two hyperparameters that define convolution are the size and the number of kernels. For simplicity, let's define a grayscale image I of size $(n_1 \times n_2)$ [22]:

$$I : \{1, \ldots, n_1\} \times \{1, \ldots, n_2\} \to \mathbb{R}, \quad (i, j) \mapsto I_{i,j} \tag{4.20}$$

The discrete convolution $(I * K)$ of image I with kernel $K \in \mathbb{R}^{(2h_1+1)\times(2h_2+1)}$ is given by:

$$(I * K)_{r,s} = \sum_{u=-h_1}^{h_1} \sum_{v=-h_2}^{h_2} K_{u,v} I_{r+u,s+v} \tag{4.21}$$

with

$$K = \begin{pmatrix} K_{-h_1,-h_2} & \cdots & K_{-h_1,h_2} \\ \vdots & K_{0,0} & \vdots \\ K_{h_1,-h_2} & \cdots & K_{h_1,h_2} \end{pmatrix} \tag{4.22}$$

Let l be a convolutional layer, the input of l consists of $m_1^{(l-1)}$ feature maps from the previous layer each of size $m_2^{(l-1)} \times m_3^{(l-1)}$. If $l = 1$ the input is a single image I that consists of one or more channels. The $m_1^{(l)}$ feature maps in layer l are computed as:

$$Y_i^{(l)} = B_i^{(l)} + \sum_{j=1}^{m_1^{(l-1)}} K_{i,j}^{(l)} * Y_j^{(l-1)} \quad \forall 1 \leq i \leq m_1^{(l)} \tag{4.23}$$

Here, $B_i^{(l)}$ is the bias matrix and $K_{i,j}^{(l)} \in \mathbb{R}^{(2h_1^{(l)}+1)\times(2h_2^{(l)}+1)}$ is the filter matrix, which connects the j^{th} feature map in layer $(l-1)$ with the i^{th} feature map in layer l.

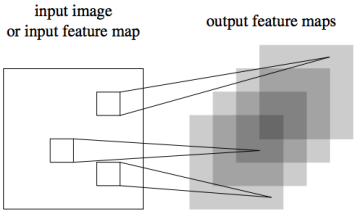


Figure 4.18: Feature map.

The complexity of the model can be considerably reduced by optimizing the output of a convolutional layer using the following hyperparameters: the size of the filter, the number of filters (depth of the next layer), the stride and setting zero-padding, see [22].

4.3.1. POOLING LAYER

The pooling layer is often placed after the convolutional layer. The purpose of pooling layers is to down sample the spatial dimensionality of the given input for the next convolutional layer. This reduces the number of parameters and the computational complexity of the network and therefore prevents overfitting.

4.3.2. FULLY CONNECTED LAYER

The fully connected layers perform classification based on the features extracted by the previous layers. The fully connected layers are traditional neural networks typically containing a soft max activation function in the last layer for classification.

4.4. DL FOR ILL-POSED INVERSE PROBLEMS

Motivated by the performance of DL models in many different domains, researches have begun to solve inverse problems in imaging. These new techniques are based on the use of various artificial neural networks. In this Section we briefly discuss the recent experimental work done in this area. In an inverse problem, a signal $f_{\text{true}} \in X$ is reconstructed from data $g \in Y$:

$$g = \mathrm{T}(f_{\text{true}}) + \delta g \tag{4.24}$$

Here, X and Y are topological vector spaces, $\mathrm{T} : X \to Y$, and $\delta g \in Y$ is the noise component. ML applied to the inverse problem amounts to reconstructing a non-linear mapping:

$$\mathrm{T}_\Theta : Y \to X \tag{4.25}$$

where, $\mathrm{T}_\Theta(g) \approx f_{\text{true}}$, and $\Theta \in Z$ is the parameter set. One way to construct a mapping is to use an FNN [47] [48]. Although this technique gives reliable results, the biggest disadvantage of using FNNs is the large memory requirement needed during training. CNNs have shown outstanding performance for solving inverse problems in imaging, see [49] [50] [51] [52].

5

DATA ACQUISITION AND PRE-PROCESSING

The goal of this research is to reconstruct an image from measured signal using supervised methods. Due to technical limitations the measured data set is small and this can be a problem since the size of a dataset is often responsible for poor performances in DL projects. This problem can be partially circumvented by augmenting the measured data set. Another technique is to use a larger synthetic data set, somewhat similar to the measured data set, train a neural network and then fine tune this pre-trained network with the smaller measured data set.

In this research, we will use four datasets. Each dataset consists of samples and each sample consist of an image and the corresponding signal. The first two datasets are synthetic in that the signals and images are calculated using simulations, while the last two datasets come from real measurements. Due to technical limitations measured dataset is small.

The images of the sets consists of variations of four geometric shapes: circles, ellipses, squares, and rectangles. In this Chapter the setup of the electronics and parameters of the MRI scan are given. Furthermore, the acquisition of the measured and simulated data is described.

5.1. MEASURED DATA

The low field MRI scan is a mix of different subsystems, each providing the necessary functionality to generate signals from phantoms. A schematic setup of the MRI scan is given in Figure 5.1.

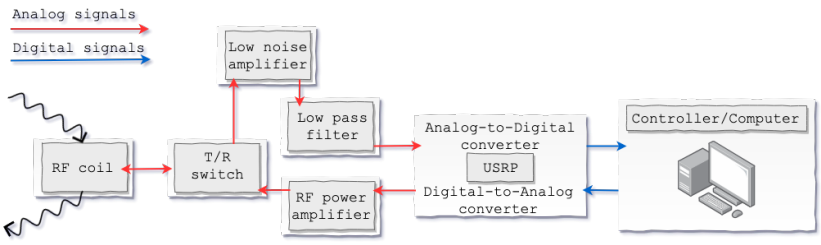


Figure 5.1: A simplified schematic representation of the low-field MRI scan setup. The wavy black arrows indicate signal leaving and entering the Rf coil.

Before starting any measurements, a phantom needs to be placed inside the RF coil in the scan, see Figure 5.2. From the PC, the user enables the USRP, a software defined radio, to generate and acquire RF signals. In our case, the USRP generates a single spin echo sequence: two successive RF pulses a $90° - 180°$ pair. The duration of a single RF pulse is set to 10 µs. Before the next spin echo sequence is applied to the same phantom, the system must wait before generating the next spin echo. The time between two successive pulse sequences is known as the repetition time (TR). The repetition time depends on the T1 value of the liquid placed in the phantom. We do not want to use a liquid that has a high T1 value, such as water (T1 = 4000 ms), because this leads to measurements taking too long. That is why we use sunflower oil for the measurements, which has a T1 value of 90 ms. The TR is set to 0.5 s, which gives the excited protons in the sunflower oil sufficient time to return to equilibrium.

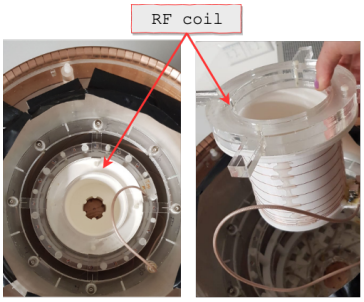


Figure 5.2: RF coil placed inside the magnet. The phantom used for measurements is placed inside the RF coil.

After the RF signal is generated, the RF power amplifier amplifies the signal to increase the amplitude so that the signal is loud enough for the desired reception to occur. The transmission and reception (T/R) switch controls the transmission of RF signals to the RF coil. After the second 180° pulse, the excited protons lose energy and realign with their equilibrium B_0. This realignment results in the emission of low energy RF photons that are converted into an electric signal by the receiver coil. The nuclei in the sample precess at a frequency f that is proportional to B_0:

$$f = \frac{\gamma}{2\pi} B_0 \tag{5.1}$$

For the measured field given in Figure 3.5, B_0-field is measured in the $z-$direction and as already said neglected in the $x-$direction. The bandwidth of the signal is defined to be the difference between the upper and lower frequency:

$$f_H - f_L = 150 \text{ kHz} \tag{5.2}$$

The receiver bandwidth (rBW) is measured as 215 kHz. This means that the signal falls within the range of frequencies involved in the reception.

After the signal is received by the RF coil, it is routed to the T/R switch and then passes through the low noise amplifier to increase the amplitude of weak signals. The signal is then directed to the low pass filter in the USRP, which blocks frequencies that are higher than the cutoff frequency. This is done to reduce noise and to prevent aliasing. The cutoff frequency is set to 100 kHz and the transition bandwidth is set to 200 kHz. This frequency is chosen on electrical grounds.

5.1.1. PHANTOM

For the measurements an efficient phantom has been designed that can contain multiple geometric shapes in 12 different positions. The shapes consist of squares, circles, ellipses, and rectangles. A 2D illustration of the phantom and shapes is given in Figure 5.3.

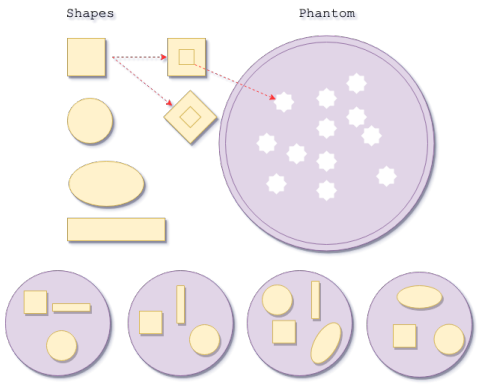


Figure 5.3: The shapes are depicted in yellow, the phantom in purple, and the holes in white. A square is placed at the back of each shape that fits into the holes.

The phantom has a radius of 40 mm. Each hole in the phantom is made of two congruent squares with

the same center at 45° angles. The dimensions of each square are $5 \times 5 \times 2.3$ mm. A block with the same dimensions is placed behind each shape so that each shape can be rotated by 45° in each hole.

The bore of the magnet lies along the x-axis. Measured from the bottom of the scan, it is assumed that the center of the magnet is $x = 247$ mm, which is 3 mm above the surface of the phantom in the RF coil. From $x = 232$ mm, B_0 is measured for $x_i = 232 + i * 5$ mm with $i \in \{0,1,2,3,4,5,6\}$. We use the B_0-field for $x = 247$ mm because this position corresponds to the bottom of the shapes placed in the phantom. Each shape is filled with sunflower oil until 3 mm from the top due to variations in B_0 in the direction of x.

We want to place the oil filled shapes in the phantom as easily as possible without spilling oil. Therefore the height of the shapes is 9 mm. As you can imagine, filling oil up to 3 mm is not an easy task with the naked eye. To make this easier for ourselves, the height is divided by a horizontal line, so that the upper height is 6 mm and the lower height 3 mm. The previous is illustrated in Figure 5.4.

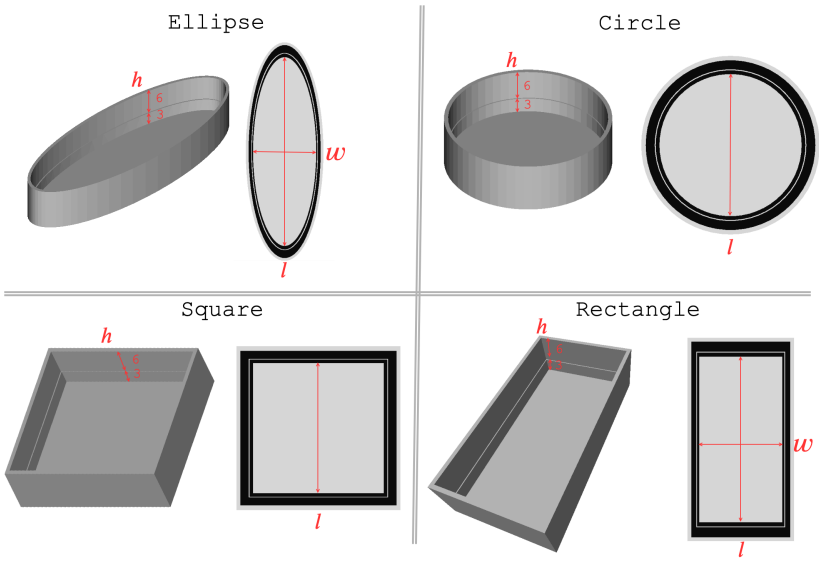


Figure 5.4: The height h, length l, and width w illustrated for each shape. For each geometric shape, variations are made with the same height but with different length and width, see Table 5.1. This is done to make the data set as diverse as possible to prevent the DL models from overfitting.

	Ellipse		Circle	Square	Rectangle	
Shape	Length	Width	Length	Length	Length	Width
1.	60	20	35	40	40	20
2.	50	10	30	30	50	20
3.	40	30	25	25	30	20
4.	40	20	20	20	40	30
5.	30	20	-	-	-	-
6.	20	10	-	-	-	-

Table 5.1: Length l and width w for each shape in millimeters.

SolidWorks, a computer-aided design software, is used to create 3D printable models. The phantom and shapes are printed using the 3D printer Formlabs Form 2, which uses clear resin (SLA). For the measurements in the scan the shapes are filed with sunflower oil. Using clear resin to print the shapes is very convenient as the material does not absorb sunflower oil.

The phantom is printed using the fused filament fabrication 3D printer Ultimaker 2+, which uses a continuous filament of polylactide (PLA). PLA is a compostable thermoplastic material made from renewable resources, such as sugarcane or corn starch. The 3D printed phantom holder and shapes are given in Figure 5.5.

Figure 5.5: 3D printed phantom holder and shapes. Each number is linked to the corresponding length and width in Table 4.1.

The process of filling the shapes with oil and placing the phantom in the scan is illustrated in Figure 5.6.

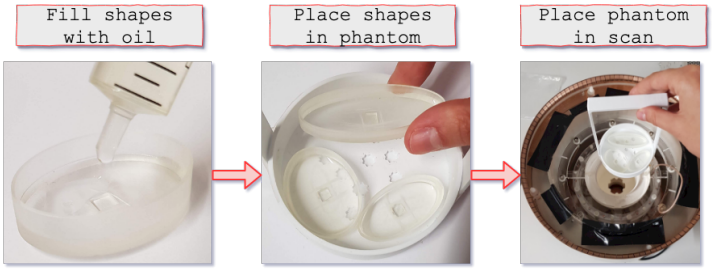


Figure 5.6: Process of placing the phantom with the oil filled shapes in the scan.

5.1.2. ACQUISITION METHOD

As already explained, the inhomogeneity of B_0 can be used for spatial encoding. To do this, the very heavy magnet needs to be physically rotated in discrete steps around the phantom. Instead of rotating the magnet, the RF coil is rotated in the magnet with an angular increment of θ. At each increment, the field experienced by the sample changes due to the inhomogeneity of B_0. An illustration of the previous is given in Figure 5.7.

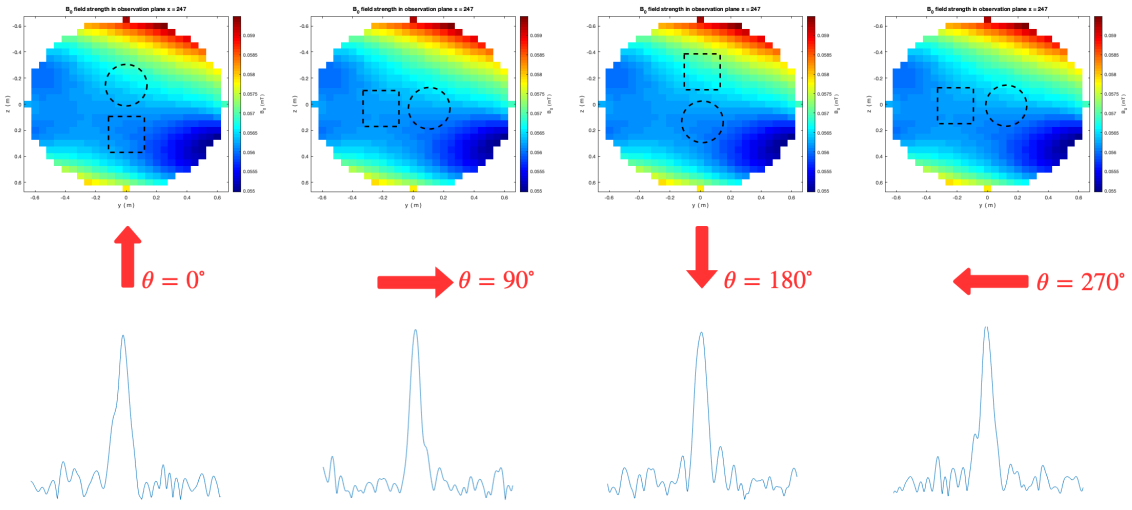


Figure 5.7: Phantom containing a circle and square rotated in the field. After each rotation of 90° the signal is measured and the time signal is plotted.

Starting from $\theta = 0°$ the RF coil is rotated by increments of ten degrees and after each rotation the signal is

measured ($0° - 350°$). The number of samples recorded after a spin echo sequence is set to 512. One spin echo sequence results in measured signal with a low SNR. Averaging can be applied to improve the SNR of the measurements. For each rotation, the measurements are repeated a hundred times. The hundred measurement are then summed up and result in the measured signal vector of size (1×512) for each angle θ_i. The measurements are done for 36 angles, so the signal matrix for one phantom is of size (36×512).

After rotating the coil 90° in the scan, the measured signal is getting worse. This is because the wire that is responsible for the transmission and reception of the pulse from and to the T/R switch is under tension. Therefore, after rotating $\theta_i = i \times 60°$ with $i \in \{1,2,3,4,5\}$ the RF coil is rotated 50° counterclockwise and the phantom inside the RF coil is rotated 60° clockwise. The rotation of the RF coil is illustrated in Figure 5.8.

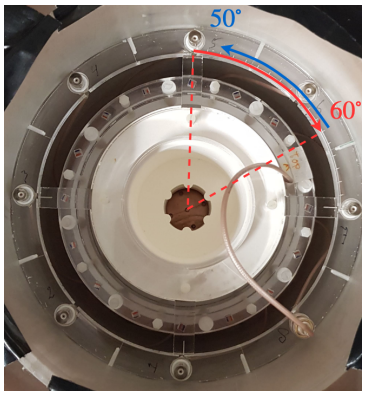


Figure 5.8: Illustration of the turning process of the RF coil after turning 60°.

A limiting factor in improving the quality of the measurements by averaging is the time needed to repeat the measurements. The acquisition time for one full rotation is 40 minutes in total. Due to time restrictions, the measured data set contains 53 labelled samples. Here, a labelled sample is an input-output pair of a signal and a corresponding image.

Of each variation of shapes in the phantom a digital image x is implemented. The image matrix x is represented by a square matrix of size (64×64) whose elements are pixel values corresponding to black or white. Black pixel values represent areas that do not give signal and white values are the areas filled with oil, which give signal. The process of creating one labelled sample is illustrated in Figure 5.9. Each sample $i \in \{1,\ldots,53\}$ consists of a signal and its corresponding image (label). The full set of samples can be found in Appendix B.

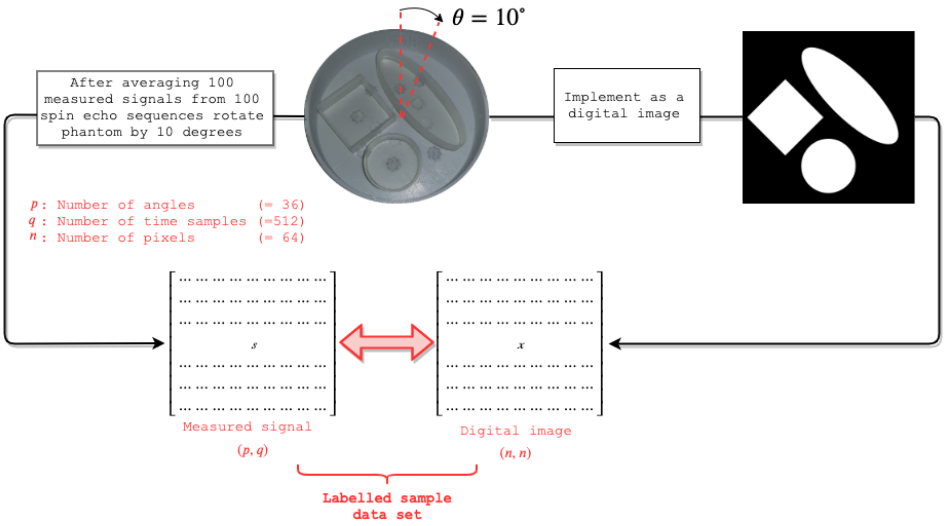


Figure 5.9: Process of creating one labelled sample for the data set. The labelled data set is used to apply deep learning models on so that the model can predict labels for unlabelled samples.

The signal data of each measurement is stored in a netCDF file, a standard file format for storing multidimensional data. A total of 53 netCDF files are created and each file consists of twelve one-dimensional variables and two multidimensional variables. The stored variables can be found in Table 5.2.

Format: netcdf4_classic

Global Attributes	
Echo time	0.01
Repitition time	0.5
RF	$2.61 \cdot 10^6$
FlipTime	1e-05
RFamplitude	1
ScanWidth	512
spinEcho	100
samplingRate	$1 \cdot 10^6$
phantom	'signal_*i*'
angle_increment	10
Dimensions	
no_angles	1
scan_width	512
Variables	
signal_real	
Size	36×512
Dimensions	no_angles, scan_width
Datatype	double
signal_imag	
Size	36×512
Dimensions	no_angles, scan_width
Datatype	double

Table 5.2: The input and output variables of a measurement stored in a netCDF file.

The neural network computational framework (TensorFlow) used for implementing our deep learning models operates on real-valued inputs and parameters. Therefore, the complex signal matrix must be split into its real and imaginary components and concatenated in one vector. The signal matrix is of size (72×512). For this research, neural networks are build that take one/two dimensional data as input/output. Therefore, the signal and image data are stored in two different ways, visualized in Figure 5.10.

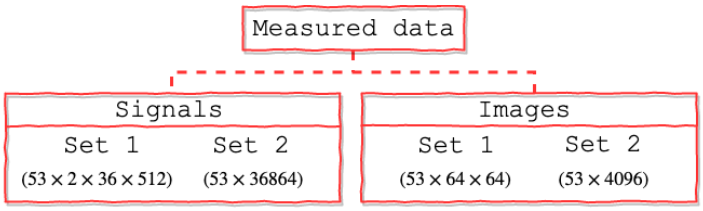


Figure 5.10: The measured signals and images from set 1 are flattened in set 2.

An important data pre-processing step is data scaling. This is done to improve the stability and performance of the DL models. Normalization reduces the variance, which in turn improves the convergence of a DL model. The signal values are within the range of 0 and 1, so only the image values are normalized. The images are gray scale and each pixel value is between 0 and 255. The pixel values are normalized by dividing each value by 255.

5.1.3. DATA ANALYSIS

Before applying any deep learning algorithms, the data needs to be analysed, because the model will be only as bad or as good as the collected data. A total of $i = 53$ signals are measured. The 53 measured signals can be arranged in a matrix $\boldsymbol{M}$ of size $(pq \times i)$, where the columns of size pq correspond to the 53 measured signals. We want to see if it is possible to reconstruct an image based on a linear combination of other measured signals, which means that there must be some sort of correlation between the measured signals. This leads to a system of equations of the form:

$$\boldsymbol{M}'_{pq\times i}\boldsymbol{y}_{i\times 1} = \boldsymbol{s}_{pq\times 1} \tag{5.3}$$

where $\boldsymbol{y}$ is the unknown, $\boldsymbol{s}$ is one of the measured signals, and $\boldsymbol{M}'$ is the matrix $\boldsymbol{M}$ where the column corresponding to signal $\boldsymbol{s}$ is replaced by zeros. The singular value decomposition (SVD) of $\boldsymbol{M}'$ is given by:

$$\boldsymbol{M}'_{pq\times i} = \boldsymbol{U}_{pq\times i}\Sigma_{i\times i}\boldsymbol{V}^T_{i\times i} \tag{5.4}$$

and $\boldsymbol{M}'$ has rank $r = 52$. The least squares solution to (5.3) is given explicitly by:

$$\boldsymbol{y} = \sum_{j=1}^{r} \frac{\boldsymbol{v}_j \boldsymbol{u}_j^T}{\sigma_j}\boldsymbol{s} \tag{5.5}$$

where, $\boldsymbol{u}_j$ and $\boldsymbol{v}_j$ are the columns of $\boldsymbol{U}$ and $\boldsymbol{V}$, respectively. Now $\boldsymbol{y}$ can be used to reconstruct the signal $\boldsymbol{s}$ and the corresponding image $\boldsymbol{x}$, where the reconstruction is denoted by $\boldsymbol{s}'$ and $\boldsymbol{x}'$:

$$\begin{cases} \boldsymbol{s}' &= \sum_{j=1}^{53} y_j \boldsymbol{s}_j \\ \boldsymbol{x}' &= \sum_{j=1}^{53} y_j \boldsymbol{x}_j \end{cases} \tag{5.6}$$

Let $\boldsymbol{s}$ and $\boldsymbol{x}$ correspond to the first measured sample, Signal_1 and Image_1, see Appendix B. The signal and the reconstruction are given in Figure 5.11. We can see that the amplitude of the signal for each angle is reconstructed reasonably well, in contrast to the noise, which is white Gaussian noise and assumed to be the background noise from the MRI scan. Therefore, it is different for each measurement and it is not possible to find a linear combination to reconstruct the noise.

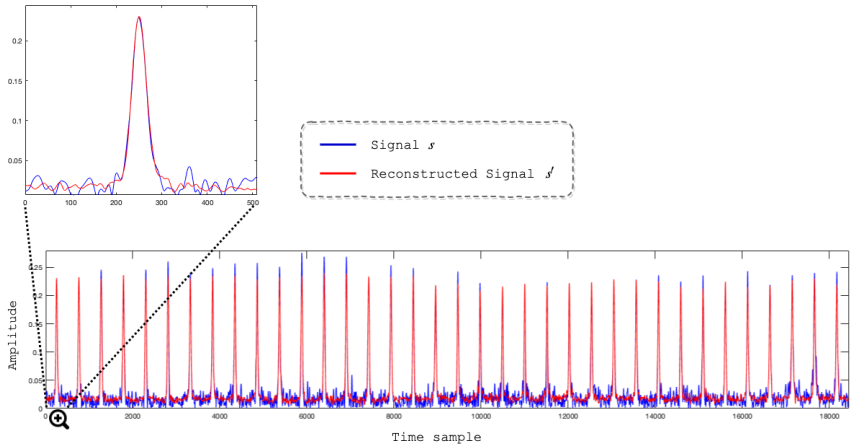


Figure 5.11: The measured time signals (blue) plotted against the reconstructed signals (red) for each angle. The measured signal corresponds to the first measured sample, Sample_1, given in Appendix B.

The reconstruction of the measured signal looks promising and therefore it should also be possible to reconstruct a fairly reasonable image. The image and its reconstruction are visualized in Figure 5.12. We can see that it is possible to reconstruct an image based on a linear combination of other measured signals, which means that there is a correlation between the measured signals. As expected shapes placed in the same position correspond to a similar signal.

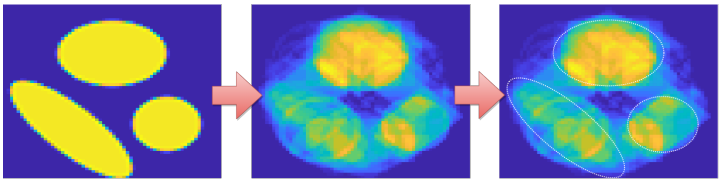


Figure 5.12: Left: The image corresponds to the first measured sample, Sample_1, given in Appendix B. Middle: Linear reconstruction of the image using (5.6) Right: Dotted lines around the edges of the shapes in the original image are placed in the reconstructed image for a better visualization of the reconstruction.

The samples in the measured data set are varied. For each measurement the shapes are placed inside the phantom in a different way. From Figure 5.13 we can see that the images are built up from other images containing shapes in the same location. This can be seen more clearly in Figure 5.14. Here, we can see that each image is a combination of the other two images. We can conclude that the signal contains information about the position of the shapes and therefore the measured data can be used for deep learning techniques.

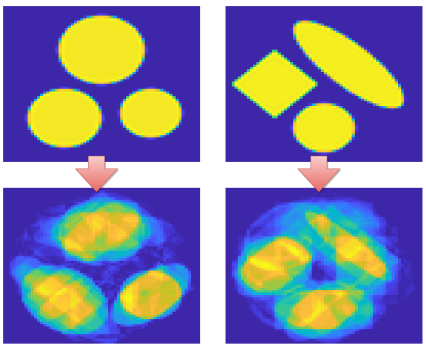


Figure 5.13: The images corresponds to the seventh measured sample, Sample_7, and 46th measured sample, Sample_46, given in Appendix B.

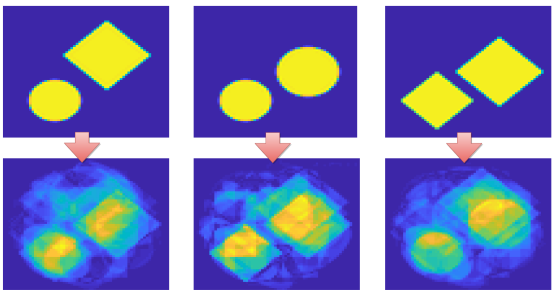


Figure 5.14: The images corresponds to the 26th measured sample, Sample_26, the 9th measured sample, Sample_9, and the 14th measured sample, Sample_14, given in Appendix B.

5.1.4. DATA AUGMENTATION

In addition to data-scaling, another important data pre-processing step is data augmentation. Our measured dataset is very small and limited datasets are a major bottleneck for deep learning experiments. As already mentioned, the acquisition time for one sample in the measured data set is 40 minutes. We need a minimum of 1000 samples to build a reasonable functioning neural network and it takes about a month of non-stop measurement to build a data set of that size. So adding more measured samples is simply not the solution. The problem can be partially circumvented by augmenting the data. I want to point out that we can never replace the data with something as good as the original, but we can try. Our approach is to augment the data by generating 36 realizations of each 2D image by rotating the images 10° and rotating the signal matrix accordingly. This results in a measured set of size 1908.

5.2. SIMULATED DATA

From Section 3.2 we know that the signal model for the low-field MRI is given by:

$$\boldsymbol{b} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{e} \tag{5.7}$$

where, $\boldsymbol{b}$ is the measured signal polluted by noise $\boldsymbol{e}$, $\boldsymbol{A}$ is the known matrix, and $\boldsymbol{x}$ is the unknown image. In the simulated case $\boldsymbol{A}$, $\boldsymbol{x}$ and $\boldsymbol{\epsilon}$ are known and $\boldsymbol{x}$ is the unknown.

We want to make the simulated data set as similar as possible to the measured data set. To calculate the matrix $\boldsymbol{A}$ the B_0-field strength discussed in Section 3.1 is used. In low-field MRI, the image is represented by a square matrix whose elements are proportional to the intensity of each pixel. Once this image or matrix is known, the signal can be calculated. First, an image of size (64×64) is generated. An image consists of random variations of ellipses, circles, squares, and rectangles. Each shape has a random position and is rotated randomly by $\frac{n\pi}{4}$ rad with $n \in \{1,\ldots,8\}$. The image is flattened to a (4096×1) vector.
For each RF pulse, the number of time steps is set to 512. Rotating the field with 10° after each RF pulse results in a simulated signal vector $\boldsymbol{b}$ of size (18432×1). An illustration of the previous is given by Figure 5.15.

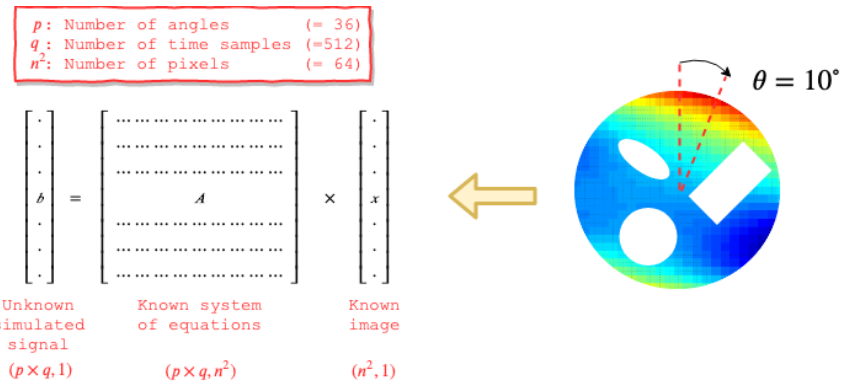


Figure 5.15: Simulated signal reconstruction.

As already mentioned in the previous Section, the neural networks build cannot take complex valued input and only take one/two dimensional data as input/output. Therefore, the simulated images and signals are also stored in two different ways, visualized in Figure 5.16.

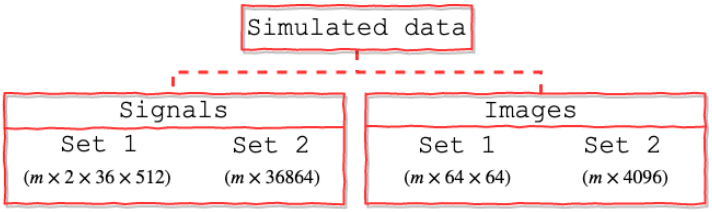


Figure 5.16: The simulated signals and images from set 1 are flattened in set 2.

To make the simulated data set more similar to the measured set, noise ϵ is added to the simulated signals. The noise is white Gaussian noise and assumed to be the background noise coming from the MRI scan. For each signal sample noise is added with a gaussian distribution $\mathcal{N}(0, \sigma_{\text{noise}})$ to obtain a signal similar to the measured signal-to-noise ratio (SNR). However, it is not quite clear what the SNR for the measured signals is. We assume that the SNR$\in [10, 80]$ and add white gaussian noise to each signal based on a random SNR value within its range.

The signal strength of the simulated signal is much greater than the measured signal. This is because the simulated setting assumes that all protons are excited. However, the measured setting is far from ideal, because it has much less excited protons. The simulated signal strength is approximately 10^2 greater than the measured signal strength. Therefore, the simulated signals are randomly scaled between a range of the minimum and maximum peak values (pv_{meas}) of the measured signal: $pv_{\text{meas}} \in [0.06, 0.3]$.

TISSUE TYPES

The human head can be separated into five general components: brain matter, bone marrow, cerebrospinal fluid, muscular skin and the skull. Besides black and white images, it is also interesting to research if it is possible to reconstruct an image containing various pixel values. We build a second simulated data set that consists of random variations of the Shepp-Logan phantom of size (64×64), depicted in Figure 5.17.

Figure 5.17: Shepp-Logan image consisting of five different pixel values.

6

ARCHITECTURES AND TEST CASES

In the previous Chapters, we introduced the theoretical knowledge needed to develop a model to reconstruct an image from MR signal. In this Chapter we present the details of three different model architectures. For each model we also present the hyperparameter space used as a starting point to find the best hyperparameter configuration.

For the low-field MRI, the inverse problem of reconstructing a signal $X \in \mathscr{X}$ from an image $Y \in \mathscr{Y}$ is given by the forward model:

$$X = \mathrm{T}(Y) + \epsilon \tag{6.1}$$

Here, $\mathscr{X}$ represents the input space (feature space), $\mathscr{Y}$ represents the output space (label space), m_{train} is the number of training samples, and ϵ is a single sampled $\mathscr{Y}-$valued noise component of the observed data. The unknown joint probability distribution on $\mathscr{X} \times \mathscr{Y}$ is defined as $P_{X,Y}$. The training set $\mathscr{D}_{m_{\text{train}}} = (X_i, Y_i)_{i=1}^{m_{\text{train}}}$ are pairs of i.i.d random variables to be distributed by $P_{X,Y}$. In supervised learning, DL applied to the inverse problem amounts to finding a non-linear mapping from a fixed class of prediction rules $\mathscr{T}$:

$$\mathrm{T}_{\boldsymbol{\Theta}}^{\dagger} : \mathscr{X} \to \mathscr{Y} \tag{6.2}$$

where, $\mathrm{T}_{\boldsymbol{\Theta}}^{\dagger} \in \mathscr{T}$, $\mathrm{T}_{\boldsymbol{\Theta}}^{\dagger}(X) \approx Y$ (pseudo-inverse property), and $\boldsymbol{\Theta} \in \mathscr{Z}$ is a multi dimensional parameter set. The function $\mathrm{T}_{\boldsymbol{\Theta}}^{\dagger}$ is learned by minimizing the squared error loss:

$$l\left(\mathrm{T}_{\boldsymbol{\Theta}}^{\dagger}(X), Y\right) = ||\mathrm{T}_{\boldsymbol{\Theta}}^{\dagger}(X) - Y||_{\mathscr{Y}}^2 = \frac{1}{p}\sum_{i=1}^{p}\left(Y_i - \mathrm{T}_{\boldsymbol{\Theta}}^{\dagger}(X_i)\right)^2 \tag{6.3}$$

Here, $\mathrm{T}_{\boldsymbol{\Theta}}^{\dagger}(X)$ is the model image, Y is the ground truth and p the number of image pixels. The weight update is based on the mean squared error, also known as the true risk:

$$R\left(\mathrm{T}_{\boldsymbol{\Theta}}^{\dagger}\right) = \mathbb{E}_{\mathrm{T}_{\boldsymbol{\Theta}}^{\dagger}}\left[l\left(\mathrm{T}_{\boldsymbol{\Theta}}^{\dagger}(X), Y\right)\right] \tag{6.4}$$

$R\left(\mathrm{T}_{\boldsymbol{\Theta}}^{\dagger}\right)$ is random as $\mathrm{T}_{\boldsymbol{\Theta}}^{\dagger}$ is a function of $\mathscr{D}_m$, which is random. However, the risk cannot be calculated because the distribution $P_{X,Y}$ is unknown. Instead, we calculate an approximation of the true, unknown statistical risk, called the empirical risk:

$$\hat{R}_{m_{\text{train}}}(\mathrm{T}_{\boldsymbol{\Theta}}^{\dagger}) = \frac{1}{m_{\text{train}}}\sum_{i=1}^{m_{\text{train}}} l\left(\mathrm{T}_{\boldsymbol{\Theta}}^{\dagger}(X_i), Y_i\right) \tag{6.5}$$

According to the strong law of large numbers, the empirical risk for each fixed prediction rule will converge to the true risk. To find the best performing predictor from a pool of candidates the empirical risk is minimized:

$$\mathrm{R}^* = \underset{\mathrm{T}_{\boldsymbol{\Theta}}^{\dagger} \in \mathscr{T}}{\operatorname{argmin}} \hat{R}\left(\mathrm{T}_{\boldsymbol{\Theta}}^{\dagger}\right) \tag{6.6}$$

6.1. MODEL ARCHITECTURES

In this Section three different model architectures are presented: Model I, Model II, and Model III.

6.1.1. MODEL I

The universal approximation theorem states that a feedforward neural network (FNN), see Section 4.1., with one hidden layer and a finite number of nodes can approximate continuous functions under some assumptions. To be precise, let M_0 denote the number of input nodes, $M = M_L$ denote the number of output nodes, and M_l the number of nodes in layer l. The theorem states [53]:

Theorem 1 *Let φ be a non constant, bounded, and monotonically increasing continuous function. Let I_{M_0} denote the M_0-dimensional unit hyper cube $[0,1]^{M_0}$. The space of continuous functions on I_{M_0} is denoted by $C(I_{M_0})$. Then, given any $\epsilon > 0$ and function $f \in C(M_0)$, there exists an integer M, real constants α_i, b_i and real vectors w_{ij}, with $i = 1,\dots,M_1$ and $j = 1,\dots,M_0$, such that we may define:*

$$F(x_1,\dots,x_{M_0}) = \sum_{i=1}^{M_1} \alpha_i \varphi \left(\sum_{j=1}^{M_0} w_{ij} x_j + b_i \right) \tag{6.7}$$

as an approximate realization of the function f; that is,

$$|F(x_1,\dots,x_{M_0}) - f(x_1,\dots,x_{M_0})| < \epsilon, \tag{6.8}$$

$\forall x \in I_{M_0}$.

Equation (6.7) represents the output of an FNN with a single hidden layer:

- The network consists of an input layer of M_0 nodes and a hidden layer of M_1 nodes. The input to the network is $x_1,\dots,x_{M_0}$.
- The weights $w_{i1},\dots,w_{iM_0}$ denote the weighted connections between the i^{th} node in the hidden layer and the nodes from the input layer, and the bias is denoted by b_i.
- The output layer has weights $\alpha_1,\dots,\alpha_{M_1}$. The output of the FNN is a linear combination of the outputs of the hidden nodes

The approximation theorem states that a *single hidden layer is sufficient for an FNN to compute an uniform ϵ approximation to a given training set* [53]. This is just an existence theorem, it does not say that an FNN with one hidden layer is optimal in the sense of ease of implementation, computation time, or generalization. However, the theorem states that a simple FNN with a single hidden layer is able to find a non-linear input-output mapping, even if the model needs a lot of data for training in order to find this mapping.

Based on the existence theorem, the first model architecture is a conventional feedforward neural network (FNN) with one hidden layer. The input and output of the FNN is a one dimensional vector. The simulated dataset described in Chapter 5 is used for the development of the FNN. A schematic illustration of the FNN architecture of Model I is visualized in Figure 6.1. The input FC_{in} of the network is produced by the data pre-processing steps detailed in Chapter 5 and is of size $(2pq \times 1)$. FC_{in} is fully connected to an $(m \times 1)$ dimensional hidden layer FC_h and activated by a non-linear activation function. FC_h is fully connected to an $(n^2 \times 1)$ dimensional output layer FC_{out}, which is reshaped to an $(n \times n)$ reconstructed image.

The number of hidden nodes in FC_h has not yet been specified. We need to be careful when specifying the number of hidden nodes, because of issues related to overfitting and underfitting, see Section 4.1.4. If we use too many hidden nodes, this can lead to overfitting. The network begins to learn increasingly subtle patterns in the training data and this does not generalize well to the testing data. Using too few hidden nodes may lead to underfitting. If it is possible to reconstruct an image from an FNN, we can certainly build a model based on different layers in addition to fully-connected layers to create an even more accurate model.

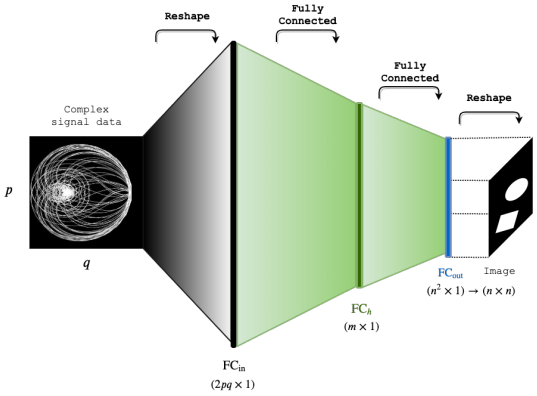


Figure 6.1: Schematic representation of the Model I architecture. The input to the network is an $(p \times q)$ signal reshaped to an $(2pq \times 1)$ dimensional vector and fed to FC_{in}. This process is depicted in black. Each of the $2pq$ nodes in FC_{in} is connected to m nodes in the hidden layer FC_h. This full connectivity between the layers is depicted in green. FC_h is fully connected to the output layer FC_{out}. The output, depicted in blue, is reshaped to an $(n \times n)$ image

6.1.2. MODEL II

The signal data contains spatial information because of the inhomogeneous B_0. Therefore, it is convenient to use convolutional layers, which works well with data with a spatial relationship. Convolutional layers are sparsely connected rather than fully connected and can go deeper to extract more abstract features without overfitting. Model II is a modification of Model I, where the dense layer is replaced by convolutional layers. The architecture of Model II loosely mimicks the convolutional neural network model VGG, which is short for Oxford's Visual Geomtery Group (VGG). See [54] for an explanation of this type of network. The VGG network is already mimicked by the DeepPET model [51], which reconstructs images for positron emission tomography (PET). VGG neural networks are unique, because the focus does not lie in tuning a large number of hyperparameters to achieve a desired accuracy. Instead, the focus lies on stacking the right number of sequential blocks of convolutions with filter (kernel) size (3×3), strides $s \in [1,2]$ and a factor 2 increase in the number of output feature maps. VGG networks are mainly used for image classification tasks and to this purpose the last three layers consists of dense layers.

Model II loosely mimicking the VGG architecture with modifications is visualized in Figure 6.2. The complex signal $(p \times q)$ is split into its real and imaginary components and treated as separate channels. The input to the neural network is of size $(2 \times p \times q)$, where 2 indicates the number of channels: real and imaginary.

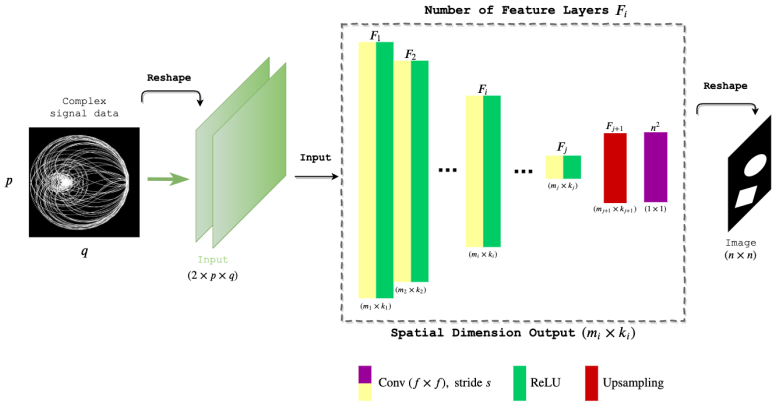


Figure 6.2: Schematic representation of the Model II architecture. The real and imaginary components of the complex signal $(p \times q)$ are separated into two channels $(2 \times p \times q)$. After convolving the reshaped input through convolutional layers, the output consists of n^2 feature maps of size (1×1), depicted in purple. The output is reshaped and represents the reconstructed $(n \times n)$ image.

The model contracts input data in a manner typical to CNNs. It consists of sequential blocks of convolutions with stride s and a factor 1 or 2 increase in the number of output features followed by an ReLU activation. The stride, which defines the step size of the kernel, is set to be 1 or 2. Throughout the network, spatial downsampling is achieved by convolutional layers employing a kernel stride of two. The final output consists of n^2 feature maps of size (1×1) where each feature is a nonlinear function of an extensive portion of the input signal. The contracted feature representation is reshaped into an image of size $(n \times n)$. We are going to implement variations of Model II, where the focus lies on the architecture: tuning the number of convolutional blocks, filter sizes, and strides.

6.1.3. MODEL III

The main cause for the big memory footprint in neural networks is the fully connected layers. However, they are fast, while convolutions use most of the computing power, even though they have a compact number of parameters. Finding the optimal parameter configuration for Model III can be a very tedious process as convolutions largely increase the computing power and the time needed for training is painfully long.

The question arises if we can built a more efficient model that uses both fully connected and convolutional layers to reduce the memory and computational requirements. Model III is a modification of Model I, between the dense layer FC_h and the output layer convolutional layers are placed.

A visualization of Model III is given in Figure 6.3. Model III is loosely mimicked from the AUTOMAP architecture with modifications. The AUTOMAP architecture reconstructs an MRI image using signals from a conventional MRI scan and therefore we can use elements of the model in the design of our own model. For a detailed explanation of AUTOMAP see [52].

In our model the input layer $\mathrm{FC}_{\mathrm{in}}$ is fully connected to an $(n^2 \times 1)$ dimensional hidden layer FC_h and activated by an activation function. The first hidden layer is then reshaped to an $(n \times n)$ matrix in preparation for convolutional processing. A total of k convolutional layers are added to the model. Each layer has f filters that convolve with stride 1 followed by an activation function. The final output is spatially down sampled (deconvolved) and represents the reconstructed $(n \times n)$ image.

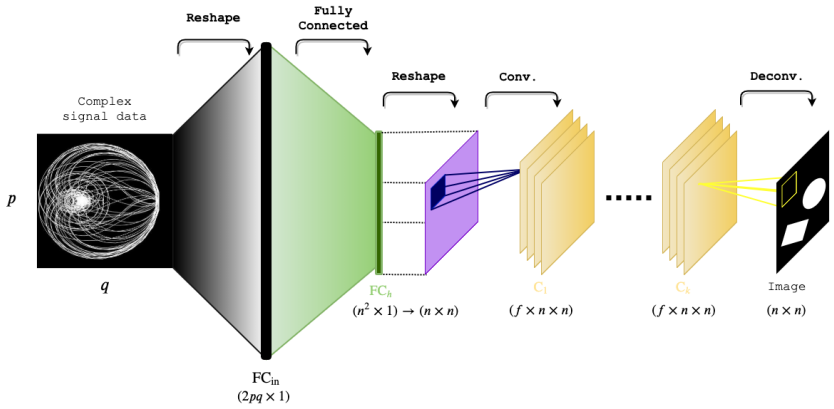


Figure 6.3: Schematic representation of the Model III architecture. The input to the network is a $(p \times q)$ signal reshaped to an $(2pq \times 1)$ dimensional vector and fed to $\mathrm{FC}_{\mathrm{in}}$. This process is depicted in gray. Each of the $2pq$ neurons in $\mathrm{FC}_{\mathrm{in}}$ is connected to n^2 neurons in the hidden layer FC_h and followed by an activation function. The full connectivity between $\mathrm{FC}_{\mathrm{in}}$ and FC_h is depicted in green. FC_h is reshaped to an $(n \times n)$ matrix in preparation for convolutional processing. The reshaped output is depicted in purple. Each convolutional layer is followed by an activation function. After convolving the reshaped output through k convolutional layers, the output consists of f feature maps and is spatially down sampled (deconvolved) to represent the reconstructed $(n \times n)$ image.

6.2. DESIGN OF TEST CASES

In this Section, test cases are built to find the best hyperparameter configuration for each of the proposed model architectures. A high-performance model with good generalization abilities requires the tuning of many interdependent hyperparameters. This is a tedious manual process of repetition (trial-and-error). To automate the workflow of conducting hyperparameter optimization a machine-assisted approach is used.

Before starting the learning process we need to ask ourselves which hyperparameters should be tuned and in which ranges. In this Section, we define these hyperparameters and set their range based on recommendations from literature. Finally, the initial hyperparameter configuration for each of the proposed model

architectures is given, which will be used as a starting point for training and developing the proposed models.

TRAIN AND VALIDATION SPLIT

The simulated dataset is randomly divided into three splits: train, validation and test. Throughout the experiments, the three sets are kept separate. A common train and validation split ratio is 80/20, also known as the Pareto principle. Out of $m = 20,000$ randomly generated signal and image pairs, $m_{\text{train}} = 16,000$ is used for training, $m_{\text{val}} = 4000$ is used for validation and a separate set is generated for testing:

$$\begin{cases} \mathscr{D}^{\text{sim}}_{m_{\text{train}}} = \left(X_i, Y_i\right)_{i=1}^{m_{\text{train}}} \\ \mathscr{D}^{\text{sim}}_{m_{\text{val}}} = \left(X_i, Y_i\right)_{i=1}^{m_{\text{val}}} \\ \mathscr{D}^{\text{sim}}_{m_{\text{test}}} = \left(X_i, Y_i\right)_{i=1}^{m_{\text{test}}} \end{cases}$$

The measured data set and its augmentation are small. From the 53 measured samples, $m_{\text{train}} = 40$ are used for training, $m_{\text{val}} = 10$ for validating, and $m_{\text{test}} = 3$ for testing. From the augmented data set, we consider three recommended train and validation split ratios from literature [22]: 60/40 ,70/30, 80/20. Before splitting the data, a small part is separated for testing purposes.

$$\begin{cases} \mathscr{D}^{\text{meas}}_{m_{\text{train}}} = \left(X_i, Y_i\right)_{i=1}^{m_{\text{train}}} \\ \mathscr{D}^{\text{meas}}_{m_{\text{val}}} = \left(X_i, Y_i\right)_{i=1}^{m_{\text{val}}} \\ \mathscr{D}^{\text{meas}}_{m_{\text{test}}} = \left(X_i, Y_i\right)_{i=1}^{m_{\text{test}}} \end{cases}$$

FULLY CONNECTED AND CONVOLUTIONAL LAYERS

For Model I, the number of hidden neurons in the dense layer is kept low enough in order to secure the ability of the network to generalize. If we have a large excess of nodes, the network can become a memory bank that can recall the training set to perfection, but does not perform well on samples that were not part of the training set. Ultimately, the selection of the number of hidden nodes comes down to trial and error. Besides the problems of overfitting and underfitting, we cannot use too many hidden nodes, as we might otherwise overload CPU memory. For example, a single fully connected layer between input and output would require a staggering $1.5 \cdot 10^8$ weights. Storing these weights in single precision (32-bit floating-point) would require 1.5 gigabyte of memory. For the hidden layer, we vary the number of hidden nodes to n^2.

Model II, inspired on the VGG and the DeepPET architecture, consists of only convolutional layers. First, the signal is passed through a stack of sequential convolutional blocks, where each block i convolves F_i filters with either a filter of size (3×3) or (5×5). After each block the number of output feature layers is increased with a factor 1 or 2. The input of each block is spatially downsampled by using convolutions with stride 2. Occasionally, convolutions with stride 1 are used in order to see if it makes any difference in regards to accuracy. The output of the sequential blocks is spatially upsampled to n^2 feature maps of size (2×2) which is then convolved by 4096 filters of size (2×2) and reshaped to the $(n \times n)$ reconstructed image.

The hidden layer in Model III is followed by convolutional layers. The convolution layers serve as a refinement of the image quality. Each layer convolves f filters with stride 1 and the output of the final convolutional layer is deconvolved with a filter with stride 1. In practice, it is common to use filters of size (3, 3) and (5, 5). It is rare to see filter sizes larger than (7,7) [22]. A (3×3) filter results in less weights, but more layers and therefore is able to learn more complex features. A larger filter size of (5×5) learns simpler nonlinear features using less layers but more weights. This is more computationally expensive, but uses less memory for back-propagation. As a starting point, based on the AUTOMAP architecture, each convolutional layer convolves 32 filters of size (5×5) and stride 1. Finally, the f output feature maps provided by the convolutional layer(s) are deconvolved with a (7×7) sized filter with stride 1.

The measured data set is small and this can create generalization issues. To avoid overfitting, the measured data requires a model that has a low complexity. Therefore, we can use Model I, a simple FNN, with a low number of hidden nodes.

ACTIVATION FUNCTION

A common used activation function in fully connected and convolutional layers is the ReLU activation due to the sparsity effects and the induced regularization. In Model I and II, the layers are followed by a ReLU activation.

In the AUTOMAP architecture the fully connected hidden layer is followed by a Tanh activation and the convolutional layers are followed by a ReLU activation. We conduct experiments using both the ReLU and Tanh activation after the hidden layer in Model III. The convolutional layers are followed by a ReLU activation. No activation function is used for the output layer. A regression problem is solved and therefore we are interested in directly predicting numerical values (pixel values) without any transformation.

WEIGHT INITIALIZATION

Before training starts, the weights must be initialized to small random numbers instead of using the same set of weights every time the network is trained. For instance, if the weights are set to be zero then the learning algorithm will fail to make changes to the weights of the network, which leads to the model being stuck.

Furthermore, it is important to properly initialize the weights $\mathcal{W}$ of the neural network, because this reduces the chance of gradient problems and speeds up the convergence to the least MSE. A too-small initialization leads to vanishing gradients and a too-large initialization leads to exploding gradients. Note that there is no single best way to initialize the weights of a neural network. Our understanding of how initial points affects generalization is especially primitive, offering little to no guidance for how to select the initial point [22]. The most used strategy for weight initialization depends on the activation used in the model. For ReLU activations [22], it is common to initialize the weights for each hidden layer (l) normal:

$$\mathcal{W}^{(l)} \sim \mathcal{N}(0, 0.05) \tag{6.9}$$

or uniform:

$$\mathcal{W}^{(l)} \sim \mathcal{U}(-0.05, 0.05). \tag{6.10}$$

For Model I, we experiment with normal and uniform distributed weights. Models II and III go deeper and deep networks face the difficulty that the variance of the layer outputs decreases as the data continuous upstream. This causes vanishing gradients and leads to a slow converging model. To mitigate slow learning for deep models, it is recommended to use Glorot or He weight initialization, see [32][55]. If the activation function is Tanh, one should use Glorot initialization, which performs random initialization from a distribution with a variance of $\frac{1}{\sqrt{N}}$. If the activation function is ReLU, He initialization is recommended, which performs random initialization from a distribution with a variance of $\frac{2}{N}$. In Model II the focus lies on finding the right architecture. Glorot uniform initialization is used as it is also used for initializing the weights in the VGG inspired DeepPET architecture.

For Model III, we experiment with Glorot and He initialization. From research, it is unclear if normal or uniform initialization works better and it all comes down to experimenting which one works better for our specific problem.

OPTIMIZATION ALGORITHM

Adaptive learning rate gradient methods are very popular optimization methods for training deep neural networks due to their fast convergence compared to classical stochastic gradient descent (SGD) [56]. However, it is argued that SGD has better generalization to the validation and test sets than adaptive methods [57]. The adaptive methods Adam and NAdam, see Section 4.1.1 and [58] [59], are well suited for problems that are large in terms of parameters and data. Besides that, no additional hyperparameters have to be tuned when using these optimizations. It is recommended to leave the parameters of Adam and NAdam at their default values as recommended by the original papers [58] [59]:

	Adam	NAdam	SGD
η	0.001	0.002	$[0.1, 10^{-3}, 10^{-5}]$
β_1	0.9	0.9	-
β_2	0.999	0.999	-

Table 6.1: Recommended values for Adam, Nadam, and SGD. Finding the right learning rate η for SGD typically involves grid search on a log scale from 0.1 to 10^{-5} [22]. The exponential decay rates β_1 and β_2 are left at their default values as recommended by the original papers [58] [59].

For Model I, we test three different optimization algorithms: SGD, Adam, and Nadam. For Models II and III, Adam is used.

BATCH SIZE AND EPOCHS

As already stated in Chapter 4, processing the entire synthetic data set leads to fluctuations and an unstable convergence. To control the stability and convergence speed, we process the synthetic data set in mini-batches. In this research, several common batch sizes are used: 128, 256, 512 and 1024. Based on the results, the size will be lowered or increased. For the measured data set, we will use a small batch size due to its size. The number of epochs is not that relevant. It is more important that the training and test error keep reducing and that the learning curves show no sign of underfitting or overfitting.

GRID SEARCH VS RANDOM SEARCH

After setting up a grid of hyperparameter values, we need to find the best configuration. We could manually search for good candidate values for hyperparameters or we could resort to machine-assisted help. Two most widely used machine assisted methods for hyperparameter tuning are: grid search and random search.

In grid search, every combination of hyperparameters is tried. By contrast, random search down samples grid search by randomly picking a predefined percentage of parameter combinations. In this research, we will mostly use grid search, because we build models from scratch and it is important to do this accurately. However, sometimes the ranges of the hyperparameters are very large and we end up training more than a 100 models, which can be very time consuming and unneccesary. If not all hyperparameters affect the performance of the model, then randomly selecting models can outperform grid search.

BIAS-VARIANCE TRADEOFF

Besides performing well on the training data, we also want the predictor to perform well on unseen data. To find the best model the MSE needs to be small on both the training and the validation data. The MSE for the predictor has the following bias-variance decomposition:

$$\text{MSE} = \mathbb{E}\left[\left(Y - \mathrm{T}^{\dagger}_{\boldsymbol{\Theta}}(X)\right)^2\right] \tag{6.11}$$

$$= \underbrace{\left(\mathbb{E}\left[\mathrm{T}^{\dagger}_{\boldsymbol{\Theta}}(X)\right] - Y\right)^2}_{\text{Bias}\left(\mathrm{T}^{\dagger}_{\boldsymbol{\Theta}}(X), Y\right)^2} + \underbrace{\mathbb{E}\left[\left(\mathrm{T}^{\dagger}_{\boldsymbol{\Theta}}(X_i) - \mathbb{E}\left[\mathrm{T}^{\dagger}_{\boldsymbol{\Theta}}(X)\right]\right)^2\right]}_{\text{Var}\left(\mathrm{T}^{\dagger}_{\boldsymbol{\Theta}}(X)\right)} \tag{6.12}$$

It is very important to keep in mind that between experiments, we should not make drastic changes based on the validation results. This will create bias towards the validation set and the model will not be well-generalized to the test set.

INITIAL PARAMETER CONFIGURATION

To give a better overview of the previously discussed, the initial hyperparameter configuration is given for each model architecture. These configurations will be used as a starting point for the experiments on synthetic data.

Model I	
Batch size	[128, 256]
Epochs	[10, 20]
Optimizer	[SGD, Adam, NAdam]
Metric	Mean Squared Error
Activation	ReLU
Weight initialization	[Normal, Uniform]
Learning rate	$\eta \in [0.1, 10^{-5}]$
Hidden neurons	$m_1 \in [100, 4500]$

Table 6.2: Initial parameter boundaries for Model I using synthetic data to develop Model I.

Model II	
Batch size	1024
Epochs	[50, 100, 150]
Optimizer	Adam
Metric	Mean Squared Error
Activation C	ReLU
Weight initialization	Glorot Uniform
Convolutions	[3×3, 2×2]
Features	[32, 64, 128, 256]
Stride	[1, 2]

Table 6.3: Initial parameter boundaries for Model II using synthetic data to develop Model II.

Model III	
Batch size	[512, 1024]
Epochs	20
Optimizer	Adam
Metric	Mean Squared Error
Activation FC_h	[Tanh, ReLU]
Activation C_1	ReLU
Weight initialization	[Glorot, He]
Hidden neurons FC_{h_1}	n^2 =4096
Convolutions C_1	5×5
Deconvolution	7×7
Features	32
Stride	1

Table 6.4: Initial parameter boundaries for Model III using synthetic data to develop Model III.

6.3. IMPLEMENTATION

The models are implemented and trained in Python. Keras [60], an open-source and user-friendly neural network library written in Python, is used for network definition and training.

The simulated data set has a size of 6.4 GB. If the CPU and RAM resources are limited, the size of the data set and network architectures can cause problems with crashing algorithms and memory errors. This problem can be solved by using Amazon Web Services (AWS). AWS offers machines with dozens of gigabytes for which computation time can be rented. From the Amazon cluster a C5 instance, ideal for computation intensive workloads [61], is used.

7

RESULTS: SYNTHETIC DATA

In this Chapter we present the results obtained after implementation and running simulations of the models using the initial parameter configurations and the simulated data set. We would like to know the effect of different architectures on the simulated data sets and which parameter configuration results in the best model. The results are divided in correspondence with the Models I, II and III. For each part we present results of various parameter configurations.

To compare models and test their relevance we predict an image of a control signal, a signal not used during training. The control images used are given in Figure 7.1. The first control image is a simple heart. The second control image consists of three randomly rotated geometric shapes: a heart, an ellipse, and a triangle. Heart and triangle shapes were not used during training. The third image is a modification of the famous Shepp-Logan phantom, a simplified model of the human head. These complicated control images are used to test the relevance of the models; is the network able to reconstruct complicated patterns not used during training. The Shepp-Logan image is the most difficult to reconstruct because we train the models on data that contain a maximum of 3 geometric shapes. If a model can reconstruct the Shepp-Logan image, we can conclude that the model has successfully reconstructed the mapping between signal and image. The SNR value for the control signals is a random value between [10, 80].

The best performing architecture is used as a starting point for experiments on synthetic data containing variations of the Shepp-Logan image, see Section 5.2. An extensive discussion of all results and associated conclusions is documented in Chapter 9.

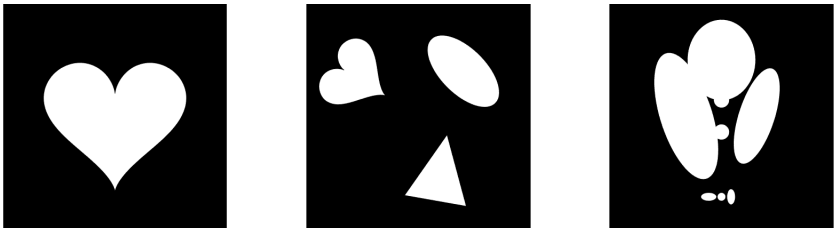


Figure 7.1: Three different control images used to check the performance of the Models I, II and III.

7.1. MODEL I

The initial parameter configuration for Model I is given in Table 6.2. Of the total number of permutations, 65% is picked using random search. The results are given in Table 7.1. For each optimizer, the results are split according to normal or uniform weight initialization and for each initialization, the validation loss is given in ascending value.

For each optimizer, the validation loss plotted against the training loss is given in Figure 7.2. It is clear that SGD is outperformed by Adam and Nadam. However, it is claimed that adaptive methods generally perform well in the initial phase of training, but SGD outperforms them in the long run. If we look at the generalization abilities of the model that has been trained with SGD, see Figure 7.3, it is clear that the model does not generalize well to unseen data. This may be due to the fact that the initial number of epochs is kept

low. Increasing the number of epochs could improve the performance of SGD. In addition to increasing the number of epochs, SGD requires more experiments (tuning of the learning rate) to converge and this can be a tedious process. To alleviate some of the pressure of choosing a learning rate and learning rate schedule and also because of the limited available computational power, SGD is dropped for the remainder of the experiment.

From Table 7.1 it is clear that the number of epochs has an influence on the convergence. By looking at the validation and training losses, we see that they are quite close to each other. There are no signs of overfitting or underfitting, so we can safely say that increasing the number of epochs will improve convergence.

	Val Loss	Train Loss	Batch	Epochs	Neurons	Weights	Learning rate
Adam							
1	0.021923	0.016371	128	20	4500	Normal	-
2	0.023897	0.018875	256	20	4500	Normal	-
3	0.024743	0.020641	128	20	100	Normal	-
4	0.028368	0.025728	256	10	4500	Normal	-
1	0.012053	0.009346	128	20	4500	Uniform	-
2	0.012837	0.010119	256	20	4500	Uniform	-
3	0.022961	0.020489	128	20	100	Uniform	-
4	0.026413	0.026413	256	20	100	Uniform	-
NAdam							
1	0.062805	0.061860	128	20	4500	Normal	-
2	0.070727	0.069776	128	20	100	Normal	-
3	0.080078	0.077190	256	10	4500	Normal	-
1	0.016070	0.01339	128	20	4500	Uniform	-
2	0.022283	0.017830	256	20	4500	Uniform	-
3	0.023804	0.021183	128	10	100	Uniform	-
4	0.032842	0.029542	256	20	100	Uniform	-
SGD							
1	0.160216	0.161904	128	20	4500	Normal	10^{-5}
2	0.164960	0.167248	256	20	100	Normal	10^{-5}
3	0.242931	0.246604	128	10	4500	Normal	10^{-5}
1	0.142097	0.143419	128	20	4500	Uniform	0.1
2	0.143182	0.145098	128	20	4500	Uniform	10^{-5}
3	0.162518	0.165643	128	10	4500	Uniform	0.1

Table 7.1: Results run 1 Model I. For each optimizer, the results are split according to normal or uniform weight initialization and for each initialization, the validation loss is given in ascending value.

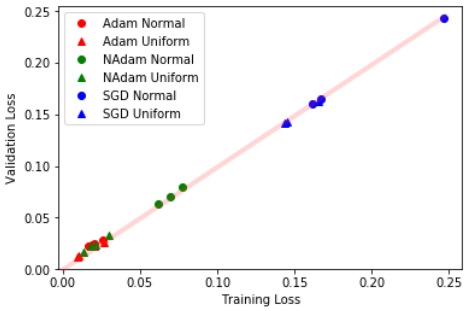


Figure 7.2: The training loss plotted against the validation loss with respect to the optimizers: SGD, Adam and NAdam. For each optimizer, the loss is plotted for both kernel initializers: Normal and uniform. The points fall in a straight line (red), which is a sign that the model is not overfitting or underfitting, the training loss is almost equal to the validation loss.

Figure 7.3: The performance of the model trained with SGD checked using the heart control image. The hyperparameters of the model used correspond to the row in yellow in Table 7.1.

The parameter space for the second run is given in 7.2. A grid search is performed that results in 16 permutations. The results are given in Table 7.2. The situation is fairly clear; generally Adam outperforms NAdam, see Figure 7.4.

In terms of initialization, uniform kernel initialization works best in combination with Adam and NAdam. Normal initialization and NAdam are dropped for the remainder of the experiment. Besides the initialization, we can see that a hidden layer that contains 4500 hidden neurons has a slightly lower loss than with 100 hidden neurons. Training a hidden layer that contains 100 hidden nodes is ten times faster. Therefore, it is interesting to see if increasing the number of epochs decreases the validation loss even further. Note that even if the loss decreases, this does not mean that the generalization abilities of the model increases. It might be the case that a low number of hidden neurons is not enough to find the right mapping between input and output.

Model I	
Batch size	[256, 512]
Epochs	30
Optimizer	[Adam, NAdam]
Metric	Mean Squared Error
Activation	ReLU
Weight initialization	[Normal, Uniform]
Hidden neurons m_1	[100, 4500]

Table 7.2: Second run parameter configuration Model I.

	Val Loss	Train Loss	Batch	Nodes	Weights
Adam					
1	0.022541	0.016684	256	4500	Normal
2	0.025939	0.020961	512	4500	Normal
3	0.026214	0.022865	256	100	Normal
4	0.035425	0.031925	512	100	Normal
1	0.013020	0.009759	256	4500	Uniform
2	0.018388	0.014479	512	4500	Uniform
3	0.028093	0.025093	256	100	Uniform
4	0.033066	0.031656	512	100	Uniform
NAdam					
1	0.026869	0.022879	256	4500	Normal
2	0.029113	0.027175	512	4500	Normal
3	0.034254	0.029234	512	100	Normal
4	0.034902	0.030988	256	100	Normal
1	0.016751	0.013799	256	4500	Uniform
2	0.024483	0.019382	512	4500	Uniform
3	0.028931	0.025341	256	100	Uniform
4	0.033031	0.027649	512	100	Uniform

Table 7.2: Results run 2 Model I. For each optimizer, the results are split according to normal or uniform weight initialization and for each initialization, the validation loss is given in ascending value.

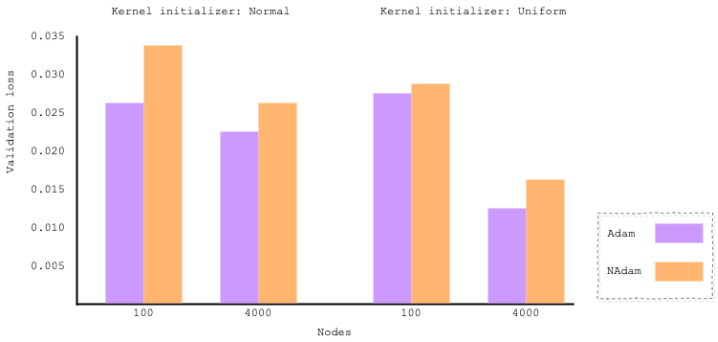


Figure 7.4: A comparison plot between the hyperparameters: Nodes, kernel initializer, and optimizer.

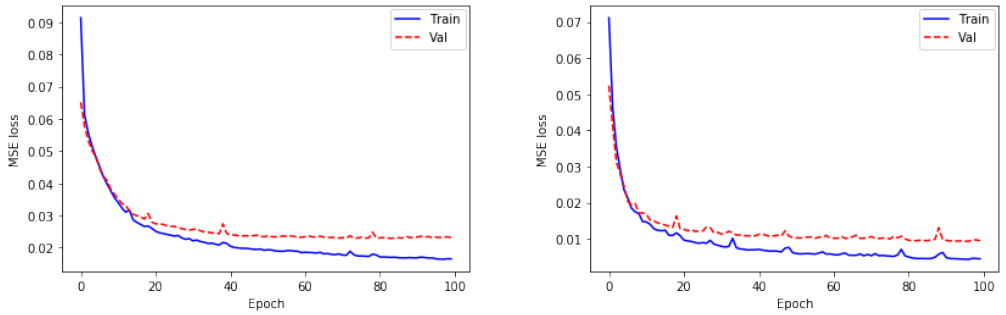
For the third run of experiments the initial parameter configurations are given in Table 7.3. Based on the previous experiment, we are interested to see what the impact is of the width of the hidden layer on the generalization abilities of the model. Two models are trained, each containing a different number of hidden nodes. The number of epochs is increased to 100 and the batch size is fixed at 256. The results of the third round of experiments are given in Table 7.4. It is evident that a wider network decreases the loss even further. Looking at the loss curves, see Figure 7.5, both the training and validation loss keep decreasing, with the validation loss eventually plateauing. Hence, increasing the capacity of the model does not lead to overfitting the training data set, but increases the generalization abilities of the model to unseen data, see Figure 7.6.

Model I	
Batch size	256
Epochs	100
Optimizer	Adam
Metric	Mean Squared Error
Activation	ReLU
Weight initialization	Uniform
Hidden nodes m_1	[100, 1000]

Table 7.3: Third run parameter configuration Model I.

Val Loss	Train Loss	Nodes
Adam		
0.023185	0.016521	100
0.009557	0.004504	1000

Table 7.4: Results run 3 Model I. The losses of two models each containing a different width.



(a) 100 hidden nodes

(b) 1000 hidden nodes

Figure 7.5: The convergence behaviour of the MSE computed between the ground truth simulation images and the reconstructed images. The training error (blue) and validation error (red) are depicted for each epoch. The error for both losses decrease, which mean that the network leans to represent the data features.

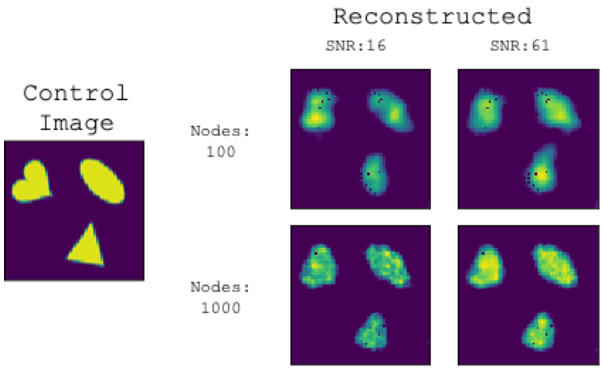


Figure 7.6: Top: Image reconstruction with the model containing 100 hidden nodes in the hidden layer. Bottom: Image reconstruction with the model containing 1000 hidden nodes in the hidden layer.

FINALIZED MODEL

As we have seen, a wider network increases the generalization abilities of the network to new data. For the final model, the number of hidden nodes is set to 4000. The results of the control images and the losses are shown in Figure 7.7. We can see that the model is able to reconstruct images reasonably well; especially the heart. However, with smaller and complicated patterns, the model captures the rough position but fails to predict small details. For instance, the corners of the triangle and the contours of the small shaped heart in the second control image.

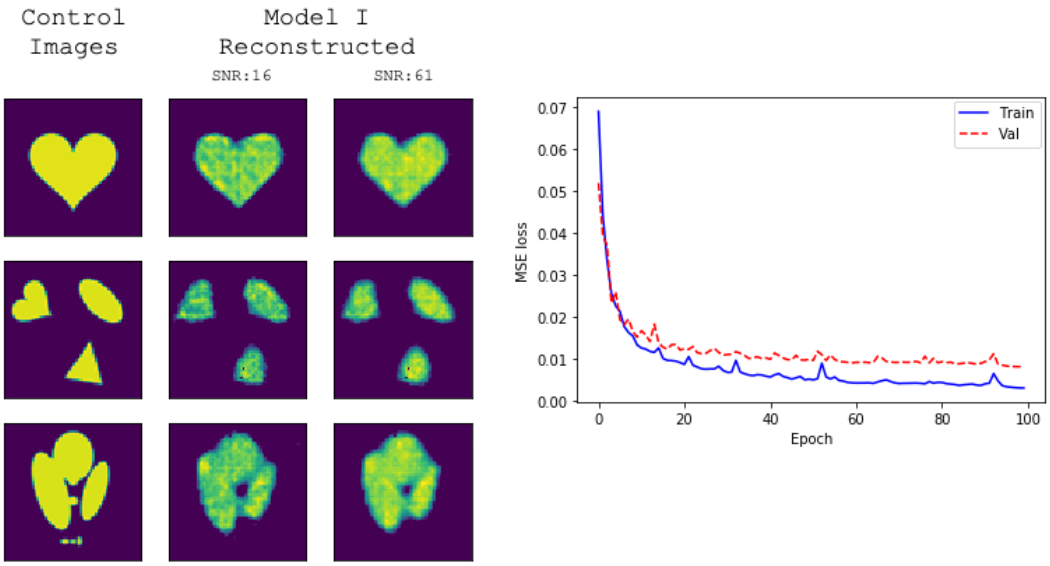


Figure 7.7: Left: Image reconstruction Model I. Right: The convergence behaviour of the MSE computed between the ground truth simulation images and the reconstructed images. The training error (blue) and validation error (red) are depicted for each epoch. The error for both losses decrease, which mean that the network leans to represent the data features.

7.2. MODEL II

As discussed in Chapter 6, for Model II, we implement and explore different architectures loosely mimicking the convolutional neural network model VGG. Our goal is to find the optimal convolutional architecture. We implement several models varying the number of convolutional layers, the filter size, the number of filters, and the stride. For each model, Adam is used for learning and the weights are uniform Glorot initialized, as discussed in Chapter 6. The number of epochs is varied and the batch size is set to 1024. If we increase the number of epochs and decrease the batch size, we should get better results, but when the number of filters is large the computation time increases and training a network can take almost an entire day.

The validation loss for the most relevant convolutional architectures, C1 through C3, are given in Table 7.4. The models are evaluated by comparing the reconstructed image quality on the control images. The models are also compared between each other based on the computational time for training.

Other convolutional architectures that are not included in this Chapter can be found in Appendix C. These models went deeper, but failed to perform and are therefore left out of this Chapter.

Name	Val Loss	Train Loss	Epochs
Adam			
C1	0.028198	0.028504	50
	0.023082	0.022977	150
C2	0.018602	0.017483	100
C3	0.018232	0.017490	50

Table 7.4: Performance of the most relevant convolutional architectures with different epochs.

MODEL: C1

Model C1 consist of sequential convolutional blocks, each containing 32 filters of size (3×3) with stride 2, see Figure 7.8. The output of this sequence is then upsampled and another convolutional layer with 4096 filters of size (2×2) convolves the output to 4096 features of size (1×1), which is then reshaped to the reconstructed image of size (64×64). The results of the control images and the loss curves for training and validation are given in Figure 7.9. We can see that the network is able to generate the rough positions of the shapes. However, looking at the learning curve we can see that the validation and training losses are still decreasing, which means that the model is not done learning and increasing the number of epochs would lead to a model that is better able to generalize to unseen data. The number of epochs is increased to 150 and the results on the control images and the loss curves are given in Figure 7.10. Comparing the control images in Figures 7.9 and 7.10, we can see that increasing the number of epochs results in a slightly better reconstructed heart and the position of the shapes. However, the model still has difficulties with reconstructing the edges and contours of the smaller shapes. Increasing the number of epochs from 50 to 150 has increased the computational time considerably, it took 5 hours to train the network with 150 epochs. If we look at the loss curve we can see that the losses are not converging and are still slightly decreasing. However, it seems that the model is close to converging, and based on that we can conclude that adding more epochs and increasing the computational time would not give us more information than we already have.

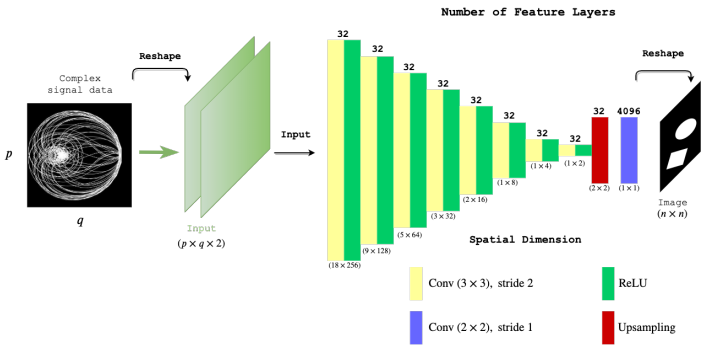


Figure 7.8: Schematic representation of the C1 architecture.

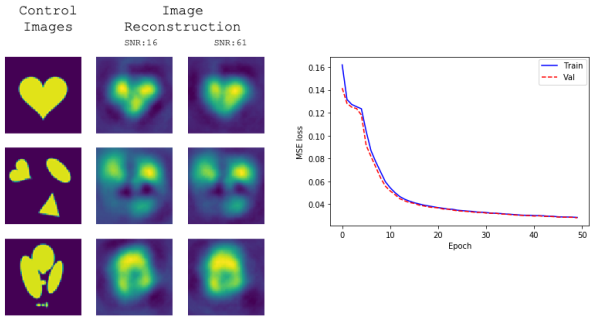


Figure 7.9: Left: Image reconstruction. Right: The convergence behaviour of the MSE computed between the ground truth simulation images and the reconstructed images. The training error (blue) and validation error (red) are depicted for each epoch. The error for both losses decrease, which mean that the network leans to represent the data features.

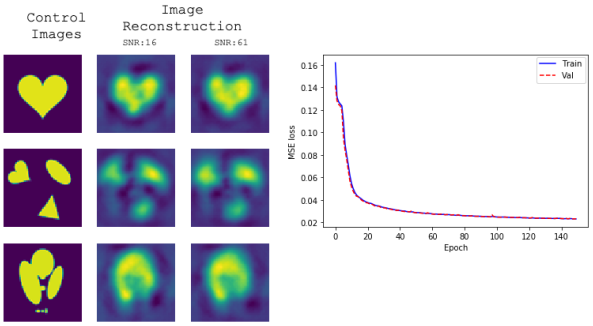


Figure 7.10: Left: Image reconstruction. Right: The convergence behaviour of the MSE computed between the ground truth simulation images and the reconstructed images. The training error (blue) and validation error (red) are depicted for each epoch. The error for both losses decrease, which mean that the network leans to represent the data features.

MODEL: C2

In the previous model, 32 filters of size (3×3) are used in each convolutional block. We built a model C2, see Figure 7.11, where the network consists of three convolutional blocks and each block consists of 3 convolutional layers convolving a different number of filters of size (3×3) and stride 2. The number of epochs is set to 100 and the results of the control images and the loss curves for training and validation are given in Figure 7.12. If we compare the results with the results from model C1, see Figure 7.10, we can see that C2 needs less epochs to learn even more from the data.

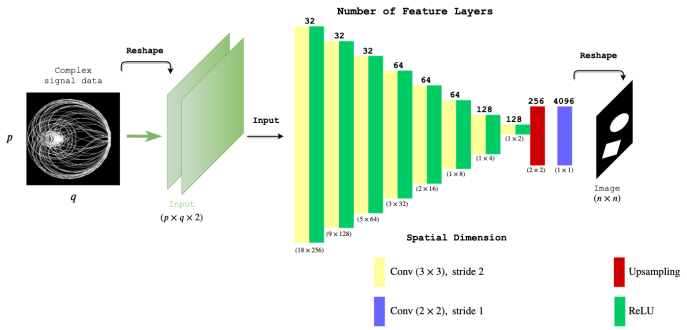


Figure 7.11: Schematic representation of the C2 architecture.

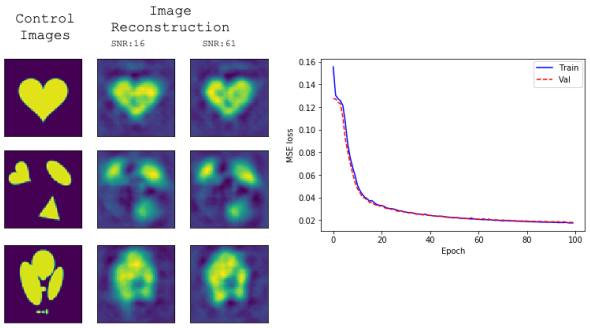


Figure 7.12: Left: Image reconstruction. Right: The convergence behaviour of the MSE computed between the ground truth simulation images and the reconstructed images is given in Figure. The training error (blue) and validation error (red) are depicted for each epoch. The error for both losses decrease, which mean that the network leans to represent the data features.

MODEL: C3

For a visualization of the C3 architecture, see Figure 7.13. Model C3 consist of sequential convolutional blocks, of which every two consecutive blocks contain a factor 2 increase in the number of filters. The filters are of size (3×3) with stride 2. The output of this sequence is then upsampled and another convolutional layer with 4096 filters of size (2×2) convolves the output to 4096 features of size (1×1), which is then reshaped to the reconstructed image of size (64×64). The number of epochs is set to 50 and the results are given in Figure 7.14. Compared to the results from C1, see Figure 7.10, we can see that the C3 is better able to reconstruct the heart and the Shepp-Logan with 50 epochs. Both models have difficulties reconstructing the second control image. As the losses of C3 are still decreasing, we could increase the number of epochs to increase the learning abilities of the network. However, training the model C3 with 50 epochs was roughly four times longer than training model C1 with 150 epochs, but the model yields better results.

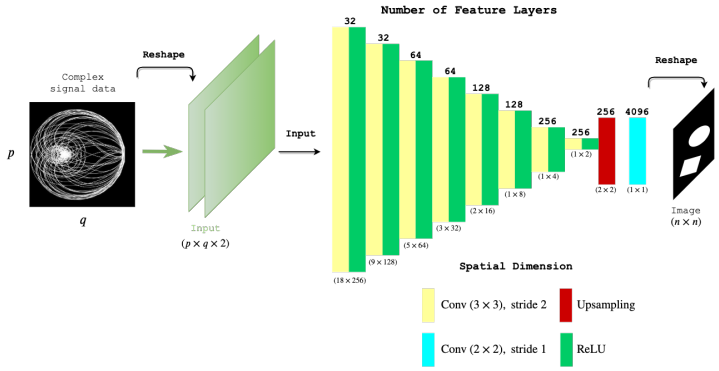


Figure 7.13: Schematic representation of the C3 architecture.

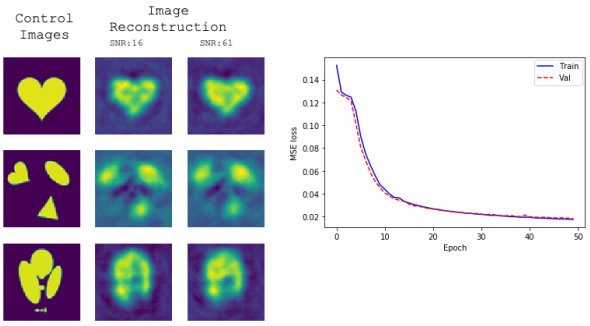


Figure 7.14: Left: Image reconstruction. Right: The convergence behaviour of the MSE computed between the ground truth simulation images and the reconstructed images. The training error (blue) and validation error (red) are depicted for each epoch. The error for both losses decrease, which mean that the network leans to represent the data features.

7.3. MODEL III

The initial parameter configuration for Model III is given in Table 6.4. A grid search is performed and we are left with 16 permutations. The results are given in Table 7.5. For each initializer, the results are split according to normal or uniform weight initialization and for each initialization, the validation loss is given in ascending value. The losses of each weight initializer are given in Figure 7.15. It is clear that Glorot weight initialization outperforms He weight initialization for both uniform and normal initialization. For the remainder of the experiment He initialization is dropped. We can also see that Tanh activation on the hidden layer FC_h performs better than a ReLU activation. As mentioned in Section 6.2., the AUTOMAP architecture used the Tanh function on the fully connected hidden layer. This indicates that literature can be very useful when setting parameters.

	Val Loss	Train Loss	Batch	Activation FC_h	Weights
			Adam		
1	0.003796	0.002304	512	Tanh	Glorot uniform
2	0.005600	0.005023	1024	Tanh	Glorot uniform
3	0.006659	0.005701	1024	ReLU	Glorot uniform
4	0.009880	0.007049	512	ReLU	Glorot uniform
1	0.005733	0.003813	512	Tanh	Glorot normal
2	0.006273	0.005491	1024	Tanh	Glorot normal
3	0.008438	0.006449	1024	ReLU	Glorot normal
4	0.021993	0.024004	512	Tanh	Glorot normal
1	0.005269	0.003846	512	Tanh	He uniform
2	0.014180	0.013883	1024	Tanh	He uniform
3	0.016181	0.013444	1024	ReLU	He uniform
4	0.021270	0.015900	512	ReLU	He uniform
1	0.010690	0.009990	512	Tanh	He normal
2	0.014817	0.011277	512	ReLU	He normal
3	0.015331	0.015024	1024	Tanh	He normal
4	0.045965	0.042600	1024	ReLU	He normal

Table 7.5: Results run 1 Model III. The optimizer and the number of epochs are fixed: Adam and 20. The results are split according to Glorot/He normal and uniform weight initialization and for each initialization, the validation loss is given in ascending value.

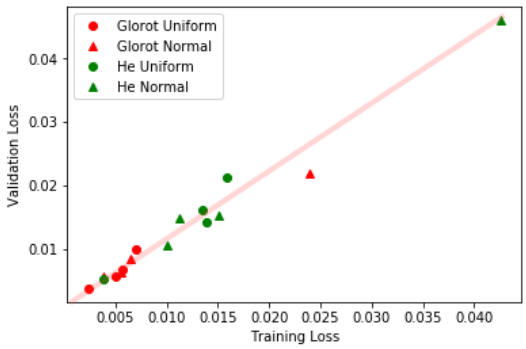


Figure 7.15: The training loss plotted against the validation loss of the weight initializers: Glorot and He. For each initialization the loss is plotted for both normal and uniform initialization. The points fall in a straight line (red), which means that the model is not overfitting or underfitting.

For the next round of experiments we look at three different batch sizes in order to compare if uniform or normal Glorot initialization performs better. The results are given in Table 7.6. For each batch size, we can see that Glorot normal performs slightly better.

For the remainder of the experiments, we focus on finding the optimal number of convolutional layers and filter size. We experiment with filters of size (3×3) and (5×5). Based on the previous results, a batch size of 128 is used and Glorot normal weight initialization. The parameter configurations are given in 7.7 and

	Val Loss	Train Loss	Batch size	Weights
Adam				
1	0.002938	0.001180	128	Glorot Normal
2	0.002986	0.001317	128	Glorot Uniform
1	0.003255	0.001476	256	Glorot Normal
2	0.017861	0.018078	256	Glorot Uniform
1	0.004171	0.003055	512	Glorot Normal
2	0.004404	0.003152	512	Glorot Uniform

Table 7.6: Results run 2 Model III. Activation function used on the hidden layer is the tanh.

the results are given in 7.8. A filter size of (5×5) performs slightly better than a filter of size (3×3) for each deconvolution. Larger filter sizes have a higher number of weights but need lesser layers as opposed to small filter sizes. On the downside, a larger filter size is computationally expensive and training takes more time. As the validation loss does not differ too much, we shall compare the performance of the models using one of the control images, see Figure 7.16. We can see that both models are able to find the rough position of the shapes. However, the smaller kernel size is able to capture the edges of the heart, so it is able to detect smaller and more complex features.

Model III	
Batch size	128
Epochs	30
Optimizer	Adam
Metric	Mean Squared Error
Activation FC_h	Tanh
Activation C_1	ReLU
Activation C_2	ReLU
Weight initialization	Glorot Normal
Hidden neurons FC_{h_1}	n^2 =4096
Convolutions C_1	$[5 \times 5, 3 \times 3]$
Deconvolution	$[3 \times 3, 5 \times 5, 7 \times 7]$
Features	32
Stride	1

Table 7.7: Third run parameter configuration Model III.

	Val Loss	Train Loss	Kernel size C_1, C_2	Kernel size Deconv.
Adam				
1	0.002686	0.000993	(3×3)	(3×3)
2	0.002684	0.000993	(3×3)	(5×5)
3	0.002725	0.001035	(3×3)	(7×7)
1	0.002682	0.001075	(5×5)	(3×3)
2	0.002648	0.001163	(5×5)	(5×5)
3	0.002703	0.001095	(5×5)	(7×7)

Table 7.8: Results run 3 Model III. Each model contains a different kernel size and the number of filters is fixed to 32.

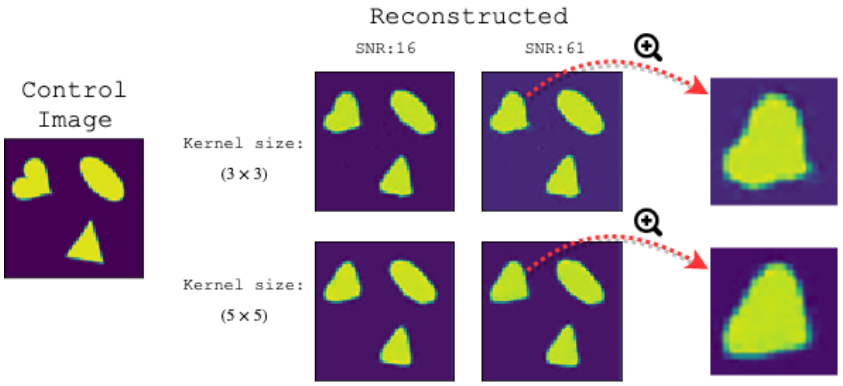


Figure 7.16: Image reconstruction using two different models that contain two hidden convolutional layers: C_1 and C_2. Both models spatially downsample the output of the convolutional layers, C_1 and C_2, using a filter of size (5×5). Top: C_1 and C_2 convolve 32 filters of size (3×3). Bottom: C_1 and C_2 convolve 32 filters of size (5×5).

Adding more layers can lead to abstracting more complex features. However, adding too much layers can lead to overfitting. For the final round of experiments, we run four models each containing a different number of convolutional layers. The parameter configuration is given in 7.9 and the results are given in 7.10. We can see that after adding more than two layers the validation loss starts to decrease.

We know that using more hidden layers will add more complexity and the model will be able to extract more complex features. If we look at Figure 4.5 in Chapter 4, we can see that in each layer a certain feature is extracted. From Model I, we can see that a fully connected hidden layer outputs fairly good images. However, the corners and contours are not that defined. Therefore, maybe one extra convolutional layer could do the trick instead of adding more layers. Which is correct, if we look at Figure 7.17, we see that adding more layers does not increase the generalization abilities of the network. More layers only increases the complexity, which is clearly not necessary to find the right mapping between input and output.

Model III	
Batch size	128
Epochs	50
Optimizer	Adam
Metric	Mean Squared Error
Activation FC_h	Tanh
Activation C	ReLU
Weight initialization	Glorot Normal
Hidden neurons FC_{h_1}	n^2 =4096
Convolutions C	3×3
Deconvolution	5×5
Features	32
Stride	1

Table 7.9: Fourth run parameter configuration Model III.

Val Loss	Train Loss	# Conv. layers
Adam		
0.003526	0.000789	1
0.002473	0.000635	2
0.002541	0.000641	3
0.002356	0.000596	4

Table 7.10: Results run 4 Model III. Four different architectures, each containing a different number of convolutional layers.

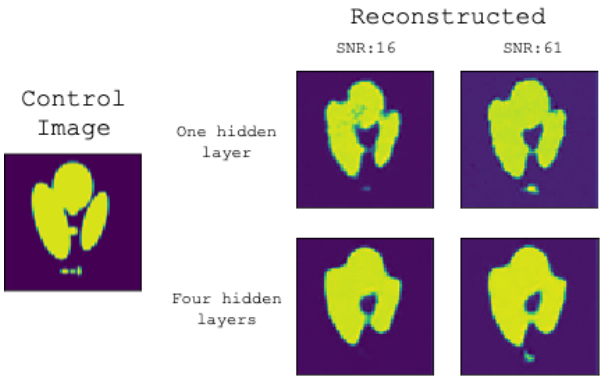


Figure 7.17: Image reconstruction using two models with a different number of convolutional layers. Top: Model with one convolutional layer. Bottom: Model with four convolutional layers.

FINAL MODEL

As we have seen, a deeper network does not increase the generalization abilities of the network to new data. The final model is trained using one convolutional layer. The loss curves and the results of the control images are shown in Figure 7.18. We can conclude that Model III is able to reconstruct good quality images.

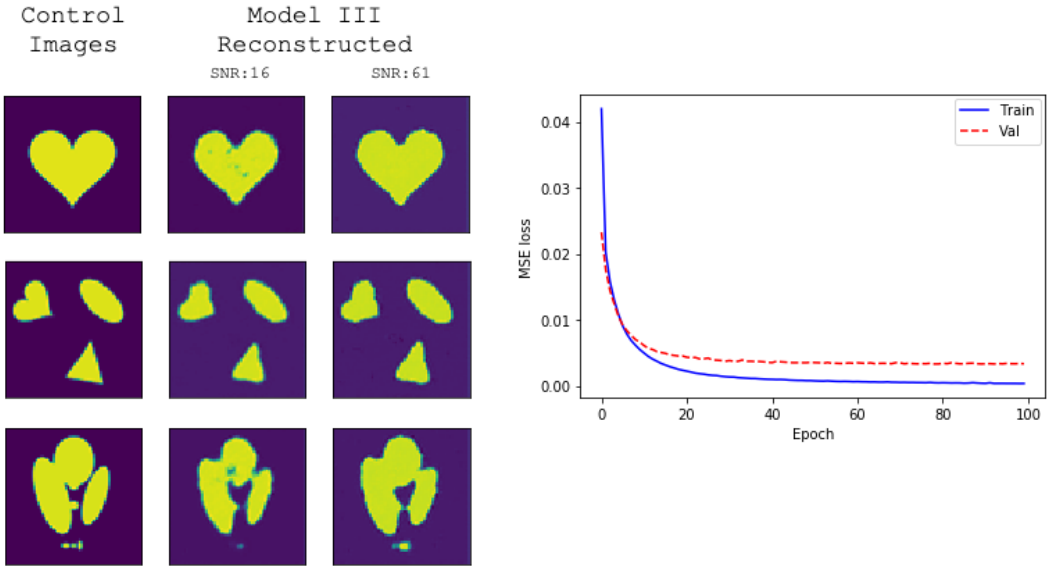


Figure 7.18: Left: Image reconstruction. Right: The convergence behaviour of the MSE computed between the ground truth simulation images and the reconstructed images. The training error (blue) and validation error (red) are depicted for each epoch. The error for both losses decrease, which mean that the network leans to represent the data features.

TISSUE TYPES

Finaly, we tested Model III on the data set containing random variations of the Shepp-Logan phantom, see Section 5.2. The same parameter configuration is used as in Table 7.9 only the number of epochs was changed to 100. The results are given Figure 7.19. We can see that the architecture of Model III with the final hyperparameters is able to reconstruct a relatively good Shepp-Logan phantom.

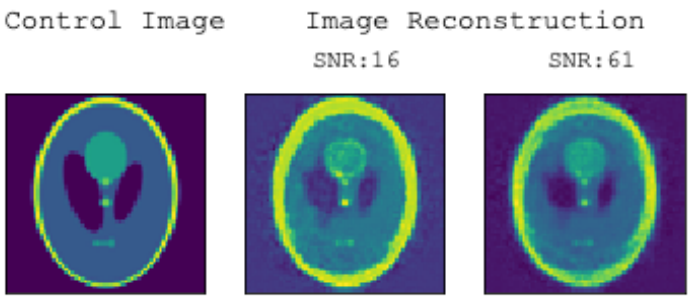


Figure 7.19: Image reconstruction using the standard Shepp-Logan phantom as test sample.

8

RESULTS: MEASURED DATA

In this Chapter we present the results obtained after implementation and running simulations using measured data. The number of measured samples is limited and in practice DL models trained with limited data tend to overfit and produce inaccurate results. This problem can be partially circumvented by keeping the architecture trained with measured data simple and use the augmented measured data set described in Section 4.1. Another technique is to use a pre-trained model with synthetic data and fine tune the pre-trained model with the measured data set. This approach is known as transfer learning, which is also a type of data extension, see [22]. For the pre-trained network, Model III is used, because it showed the most promising results on synthetic data.

A complex model with many parameters is more prone to overfitting. A network trained with a small dataset exhibits sporadic fluctuations, due to the randomness from the network initialisation and training. Therefore, it is recommended to use a simple model to mitigate the small dataset problem, see [62]. On the measured data set we used Model I, see section 6.1, because Model I is the least complex. For training, based on the results on synthetic data, Adam is used as the optimizer and ReLU as the activation in the hidden layer.

The results are subdivided in correspondence with the measured data set. First, we present results of training using the 53 measured data samples, followed by training on the augmented data set. Thereafter, we present the results using transfer learning. An extensive discussion of all outcomes and corresponding conclusions is documented in Chapter 9.

8.1. 53 SAMPLES

The initial parameter configuration for the measured data is given in Table 8.1. The results on the test signals and the average training and validation loss using 5-fold cross-validation, see Section 4.1.3, are given in Table 8.2. We can see that increasing the number of hidden nodes increases the difference between the validation and training loss. In Figure 8.1, image reconstruction using the test signals and the loss curves are given for the model containing 100 nodes. The validation loss fluctuates, which is caused by the small batch size. Besides the fluctuations, from the loss curves we can observe that the model is overfitting. This can also be seen in the image reconstruction with the test signals. The reconstructed images are combinations of the images in the training set. This means that the model is excessively adjusted to the training data. It sees patters that are not relevant and performs poorly when predicting.

Measured Model	
Train/validation/test split	40/10/3
Batch size	1
Epochs	100
Optimizer	Adam
Metric	Mean Squared Error
Activation FC_h	ReLU
Hidden neurons m_1	[10, 100, 500, 1000]

Table 8.1: Initial parameter configuration: 53 measured samples.

# Nodes	Val Loss	Train Loss	$\Delta_{\text{Val}-\text{Train}}$
Adam			
10	0.171030	0.043441	0.13
100	0.156191	0.012327	0.14
500	0.159050	0.010686	0.15
1000	0.168578	0.001091	0.17

Table 8.2: Difference between the mean validation and mean train losses for each model with a different number of hidden nodes trained with 5-fold cross-validation.

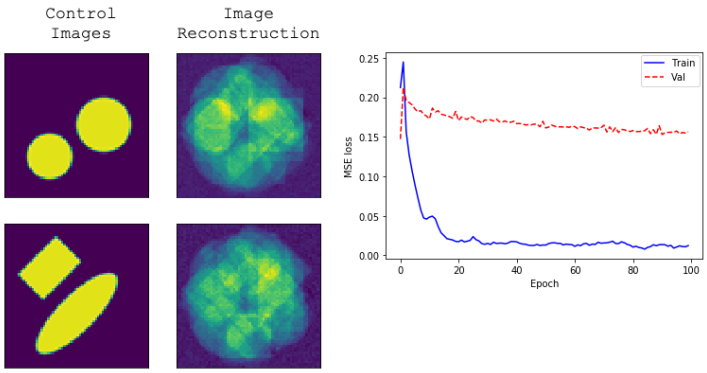


Figure 8.1: Image reconstruction using the model with a hidden layer of 100 hidden nodes.

8.2. DATA AUGMENTATION

The initial parameter configuration for the augmented data set is given in Table 8.3. The train-validation split ratio of the data relies on the problem being solved. As we have limited data, we experiment with three different train and validation splits to test which one works best four our problem, as explained in Section 6.2. Before the data is split, eight random samples are omitted, which are used as test samples. In addition to fine-tuning the split ratio, the number of hidden nodes is also tuned. The results are given in Table 8.4.

We can see that increasing the number of nodes decreases the gap between the training and validation loss. The losses of each model using a 70/30 split ratio is given in Figure 8.2. We can see that after each epoch the training loss keeps decreasing and the validation loss slightly increases. The models are not generalizing well enough on the validation set, so the models are overfitting. The models recognize specific samples in the training set and therefore can no longer generalize to unseen data.

Measured Model	
Train/validation	[60/40, 70/30, 80/20]
Batch size	32
Epochs	30
Optimizer	Adam
Metric	Mean Squared Error
Activation FC_h	ReLU
Hidden neurons m_1	[100, 1000]

Table 8.3: Initial parameter configuration: augmented measured data.

# nodes FC_h	Val Loss	Train Loss	$\Delta_{\text{Val-Train}}$
60/40			
100	0.184792	0.024768	0.16
1000	0.159211	0.008027	0.15
70/30			
100	0.187405	0.025653	0.16
1000	0.155818	0.008685	0.15
80/20			
100	0.188177	0.026704	0.16
1000	0.160979	0.009462	0.15

Table 8.4: Results training models with augmented data using different train and validation splits.

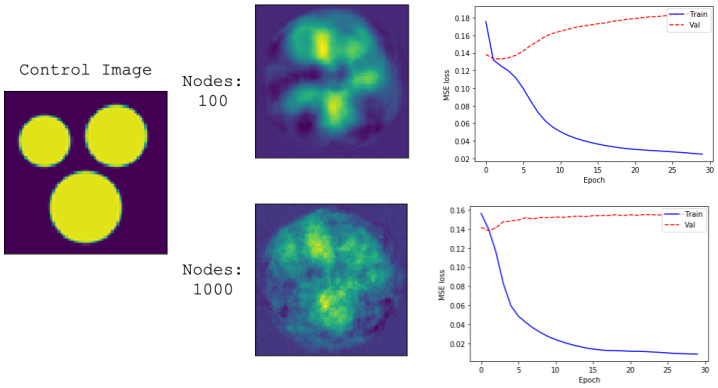


Figure 8.2: Image reconstruction using the models trained with a 70/30 split ratio. The loss curves for each model are given next to the reconstructed image.

8.3. TRANSFER LEARNING

Transfer learning is a powerful method that applies knowledge gained from one problem to another different but related problem. For this purpose, we used the finalized version of Model III from Chapter 7. The results on test data is given in Figure 8.3. We can see that the performance of the pre-trained model fine-tuned with measured data performs very poorly. The reconstructed images are very noisy, which means that the network has failed to see patterns in the measured data. The model is pre-trained on synthetic data and has therefore learned assumptions and simplifications used in the simulations, which clearly do not hold for measured data.

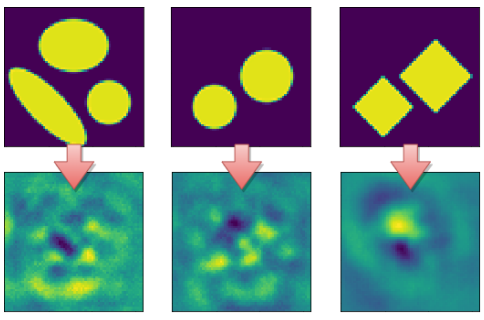


Figure 8.3: The images corresponds to the 1^{th} measured sample, the 9^{th} measured sample, and the 14^{th} measured sample given in Appendix B.

9

Conclusions and Recommendations

After our research to low-field MRI and Deep Learning, we can answer our research question:

"Is it possible to implement a deep learning based approach to generate images using measured signals from the low-field MRI?"

The answer to this question is believed to be yes; our research provides a promising framework but lacks measured data and computing power to achieve perfect results. Throughout our research, we have seen promising results on synthetic data indicating it might be possible to reconstruct images using measured signals.

We developed three different DL models: Model I, II and III. Each model consists of different architectures and is tuned using different hyperparameters. The process of DL model development can be seen as a black box. Hence, we developed several test cases attempting to point out the effect of tuning the hyperparameters and to explain the connection between input and output. However, some parts of the tuning process remain a black box, because the literature on DL is extensive, but far from complete.

Analysis of simulations run with the models on synthetic data shows that Models I and III provided most promising results with regard to image reconstruction and computational time. Model I is capable of reconstructing images with quite good quality, see Figure 7.7. The model is able to predict the rough position of the shapes. However, it failed to predict the edges and contours of the smaller shapes. In Model II, a convolutional layer placed beneath the hidden layer seemed to provide promising results with the same amount of epochs, see Figure 7.18. The small filter size in the convolutional layer proved to be able to detect the pattern related to the edges and the contours of the smaller shapes. Using a larger filter size decreases the validation loss slightly, but failed to identify patterns related to the edges and contours of the smaller shapes.

In addition to the prediction abilities, uniform weight initialization proved to be the best for models that contain one hidden layer and Glorot normal weight initialization for models containing a combination of a hidden layer and a convolutional layer.

Model II did not prove to be a good approach, because of the number of epochs needed to reconstruct good images. Increasing the number of epochs results in an exponential increase in the computation time and the improvement of the prediction is only marginal. Model C1 consists of convolutional blocks and increasing the number of epochs yielded in a slightly better prediction. However, we used blocks containing only 32 features, which explains the need for a lot of epochs. In model C3 the total number of filters was increased and this indeed yielded better reconstructed images for less epochs and the same amount of time as C1 trained with 150 epochs, see Figures 7.10 and 7.14. Furthermore, many epochs were required in order to improve images generated by models C2 and C3, which increased computation time considerably. Finding the right combination of convolutional blocks required a lot of trial-and-error as well as a lot of computational resources.

The experiments on the measured data failed, due to the limited size of the measured data set. To partially circumvent this problem one possible solution proposed was to augment the measured data set. The data was augmented by generating 36 realizations of each 2D image by rotating the images 10° and rotating the signal matrix accordingly. This resulted in a measured set of size 1908. For experiments on the small measured data set and its augmented variant, we decided to use Model I. We did not use a complex model, as it is more

prone to overfitting. Model I is a simple FNN with just one hidden layer. As expected, from the experiments on the measured data set, the models overfitted the training set. Therefore, we tried to augment the data. Each measured sample in the augmented set corresponds to a certain angular rotation within the magnetic field. As stated before, a signal measured from one angle contains little information and it is clear that the DL models were not able to extract this information.

Transfer learning produced noisy images, see Figure 8.3, because of a mismatch between the measured data and the simulated data. As the simulated and measured data are related to the same problem, we applied transfer learning to the measured data. Model III showed the most promising results on synthetic data, therefore, we used the pre-trained Model III and fine tuned the model with measured data. This resulted in very noisy images indicating that there is no sufficient correlation between the measured and synthetic signals. The pre-trained model has learned simplifications and assumptions used in the simulations. The signal model, see equation (3.10), assumes that the field-strength of the magnet can be neglected in the $x-$direction. Incorporating the magnetic field in the $x-$direction into the signal model would result in a model that is a better representation of the low-field MRI. In addition, the lack of measured data could also be the reason that transfer learning failed. There are some similarities between the synthetic and measured data, but the lack of actual data might be the reason the model failed to learn these similarities.

Our research, has shown that there is a correlation between the measured signals. Hence, we should not dismiss DL entirely. If there is enough available measured data, DL could be able to reconstruct an image. The synthetic images contain much more complicated patterns compared to the measured images. This complexity requires larger amounts of data in order to learn the patterns within the data. To this end, Model III seems to be the best fit, as the hidden layer quickly extracts general patterns and the convolutional layer is able to find more complicated patters.

9.1. FURTHER RESEARCH

The low-field MRI problem is not simple. There is still a lot of research that remains to be done on this topic. Therefore, we propose some ideas for further research.

- **Computational time**
  A big problem in this thesis was the computational time to find an optimal solution for the models using synthetic data. Especially, with models using only convolutional layers. Using a large amount of filters increases the number of weights, which is computationally expensive. For the training of the models a CPU was used instead of a GPU, due to lack of availability. However, using a GPU accelerates the training considerably. This could mean that Model II might be able to outperform the other two models. We have shown in Model III that the convolutional layer was able to detect more complex patterns, such as the edges and contours of the smaller shapes. A fully convolutional network could detect the smaller dots in the Shepp-Logan phantom.

- **Data size**
  The DL models showed promising results on the synthetic data, with Model III showing the best results. Unfortunately, the size of the measured data limited the performance of DL models. We recommend increasing the data size by doing more measurements, however it should be kept in mind that this requires a lot of time, which can limit the feasibility of this solution. If we look at the magnet field, in Figure 3.5, we can see that some regions provide the same information. Therefore, one could experiment by lowering the number of angles. This could speed up the measurement process. Note that a single measured signal from one angle for each phantom variation is likely to be not enough, as our data augmentation technique based on that idea failed. We would recommend studying more data augmentation methods to augment the measured samples.

- **Signal model**
  As it is quite hard to increase the measured data set and transfer learning failed, we still recommended to use synthetic data and transfer learning. We have seen that the signal model is not representative for the low-field MRI. Therefore, additional research needs to be done in order to reconstruct a better signal model. For instance, taking into consideration the magnetic field in the $x-$direction might improve results.

BIBLIOGRAPHY

[1] J. W. Quinn and G. Barnard, *CURE Hydrocephalus: Setting a Course for Sustainability,* http://www.globalsurgery.info/wp-content/uploads/2014/09/CURE-Hydrocephalus-Uganda.pdf (2015).

[2] F. Rosenblatt, *The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain,* Psychological Review **65**, 386-408 (1985) .

[3] V. M. Runge, S. Schonberg, and X.Li, *Magnets, Spins, and Resonances: An introduction to the basics of Magnetic Resonance* (Siemens Medical Solutions, 2013).

[4] Z. Liang and P. C. Lauterbur, *Principles of Magnetic Resonance Imaging: A Signal Processing Perspective* (SPIE Optical Engineering Press, 2000).

[5] J. A. Fessler, *Model-based image reconstruction for MRI,* IEEE Signal Processing Magazine **27**, 81–89 (2018) .

[6] H. Schmidt-Böcking, L. Schmidt, H. J. Lüdde, W. Trageser, and T. Sauer, *The Stern-Gerlach Experiment Revisited,* The European Physical Journal **41**, 327-364 (2016) .

[7] I. I. Rabi, J. R. Zacharias, S. Millman, and P. Kusch, *A New Method of Measuring Nuclear Magnetic Moment,* Psychological Review **53**(4), 318-327 (1938) .

[8] F. Bloch, W. W. Hansen, and M. Packard, *The Nuclear Induction Experiment,* Psychological Review **70**, 474-485 (1946) .

[9] E. M. Purcell, H. C. Torrey, and R. V. Pound, *Resonance Absorption by Nuclear Magnetic Moments in a Solid,* Psychological Review **69**, 37-38 (1946) .

[10] R. V. Damadian, *Tumor Detection by Nuclear Magnetic Resonance,* Science **171**, 1151-1153 (1971) .

[11] E. D. Becker, *Obituary: Paul Christian Lauterbur,* Physics Today 60, 77-78 (2007) .

[12] P. C. Lauterbur, *Image Formation by Induced Local Interaction; Examples Employing Nuclear Magnetic Resonance,* Nature 242, 190-191 (1973) .

[13] R. Turner, *Peter Mansfield (1933–2017),* Nature, 543, 180-186 (2017) .

[14] R. Damadian and L. M. M. Goldsmith, *NMR in cancer: XVI. FONAR image of the live human body,* Physiol Chem Phys 9(1), 97-100 (1977) .

[15] C. Z. Cooley, *Portable low-cost magnetic resonance imaging,* Ph.D. thesis (MIT 2014).

[16] A. Meijer, *Optimizing the gradient ring of a low-field MRI scanner,* Bachelor's thesis, (TU Delft 2019).

[17] R. Burgwal, *Measuring NMR and developing 2D imaging in a low-cost, portable MRI prototype,* Master's thesis, (Leiden University 2018).

[18] M. L. de Leeuw den Bouter, M. B. van Gijzen, and R. Remis, *Conjugate gradient variants for Lp-regularized image reconstruction in low-field mri,* SN Applied Sciences 1, 1736 (2019) .

[19] C. Z. Cooley, J. P. Stockmann, B. D. Armstrong, M. Sarracanie, M. H. Lev, M. S. Rosen, and L. L. Wald, *Two-dimensional imaging in a lightweight portable MRI scanner without gradient coils,* Magnetic resonance in medicine: Official journal of the Society of Magnetic Resonance in Medicine **73**, 872-883 (2014) .

[20] B. Woolley, *The Bride of Science: Romance, Reason, and Byron's Daughter* (McGraw-Hill, 2000).

[21] J. Fessler and J. Francis, *Lovelace & Babbage and the creation of the 1843 'notes',* IEEE Annals of the History of Computing **25**, 16–26 (2003) .

[22] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (MIT Press, 2016).

[23] T. Mitchell, *Machine Learning* (McGraw Hill, 1997).

[24] J. Vijayan, *Classifying Devanagari vowels with TFLearn,* https://jayanand90.github.io/Classifying-Devanagari-vowels-with-TFLearn/ (2017).

[25] Y. LeCun, Y. Bengio, and G. Hinton, *Deep Learning,* Nature **521**, 436-44 (2015) .

[26] D. Hoving, *MRI for Africa,* Retrieved from https://www.delta.tudelft.nl/article/mri-africa, (2017) .

[27] A. Baese and V. Schmid, *Pattern Recognition and Signal Analysis in Medical Imaging* (Academic Press, 2014).

[28] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, *Activation Functions: Comparison of trends in Practice and Research for Deep Learning,* arXiv:1811.03378 (2018) .

[29] R. Hahnloser, R. Sarpeshkar, M. Mahowald, R. J. Douglas, and H. Seung, *Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit,* Nature **405**, 947–951 (2000) .

[30] M. Zeiler, M. Ranzato, R. Monga, M. Mao, K. Yang, Q. Le, P. Nguyen, A. Senior, V. Vanhoucke, J. Dean, and G. Hinton, *On rectified linear units for speech processing,* In *ICASSP*, 2013 .

[31] G. E. Dahl, T. N. Sainath, and G. E. Hinton, *Improving deep neural networks for LVCSR using rectified linear units and dropout,* In *ICASSP*, 2013 .

[32] X. Glorot, A. Bordes, and Y. Bengio, *Deep Sparse Rectifier Neural Networks,* Journal of Machine Learning Research **15**, 315-323 (2011) .

[33] A. H. A. Maas and A. Ng, *Rectifier Nonlinearities Improve Neural Network Acoustic Models,* International Conference on Machine Learning - Proc. icml, 2013 .

[34] C. Bishop, *Pattern Recognition and Machine Learning* (Springer, 2006).

[35] C. Bishop, *Neural Networks for Pattern Recognition* (Clarendon Press Oxford, 1995).

[36] S. Ruder, *An overview of gradient descent optimization algorithms,* arXiv preprint arXiv:1609.04747(2016) .

[37] R. Reed and R. Marks, *Neural smithing: Supervised Learning in Feedforward Artificial Neural Networks* (MIT Press, 1999).

[38] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning representations by back-propagating errors,* Nature **3**, 533-536 (1986) .

[39] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, 2nd ed. (Wiley, 2001).

[40] R. D. Brian and N. L. Hjort, *Pattern Recognition and Neural Networks* (Cambridge University Press, 1995).

[41] A. Burkov, *The Hundred-Page Machine Learning Book* (Andriy Burkov, 2019).

[42] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, *Improving neural networks by preventing co-adaptation of feature detectors,* arXiv preprint arXiv:1207.0580 (2012) .

[43] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, *Dropout: A Simple Way to Prevent Neural Networks from Overfitting,* Journal of Machine Learning Research **15**, 1929-1958 (2014) .

[44] G. Montavon, G. Orr, and K. Müller, *Neural Networks: Tricks of the Trade*, 2nd ed. (Springer Publishing Company, Incorporated, 2012).

[45] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition,* Proceedings of the IEEE **86**, 2278-2324 (1998) .

[46] Y. LeCun, K. Kavukcuoglu, and C. Farabet, *Convolutional networks and applications in vision,* In IEEE International Symposium on Circuits and Systems (ISCAS), 253-256 (2010) .

[47] M. Argyrou, D. Maintas, C. Tsoumpas, and E. S. Stiliaris, *Tomographic Image Reconstruction based on Artificial Neural Network (ANN) techniques,* In IEEE Nuclear Science Symposium and Medical Imaging Conference Record (NSS/MIC) (2012) .

[48] P. Paschalis, N. D. Giokaris, A. Karabarbounis, G. K. Loudos, D. Maintas, C. N. Papanicolas, V. Spanoudaki, C. Tsoumpas, and E. Stiliaris, *Tomographic image reconstruction using Artificial Neural Networks,* Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **527**, 211-215 (2004) .

[49] M. T. McCann, K. H. Jin, and M. Unser, *Convolutional Neural Networks for Inverse Problems in Imaging: A Review,* IEEE Signal Processing Magazine 34, 85-95 (2017) .

[50] J. Adler and O. Ozan, *Solving ill-posed inverse problems using iterative deep neural networks,* Inverse Problems 33(12), 124007 (2017) .

[51] I. Häggström, C. Schmidtlein, G. Campanella, and T. Fuchs, *DeepPET: A deep encoder-decoder network for directly solving the pet image reconstruction inverse problem,* Medical Image Analysis **54**, 253 (2019).

[52] B. Zhu, J. Z. Liu, B. Rosen, and M. Rosen, *Image reconstruction by domain-transform manifold learning,* Nature **555**, 487-492 (2018) .

[53] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. (Prentice Hall PTR, Upper Saddle River, NJ, USA, 1998).

[54] K. Simonyan and A. Zisserman, *Very Deep Convolutional Networks for Large-Scale Image Recognition,* arXiv 1409.1556 (2014).

[55] K. He, X. Zhang, S. Ren, and J. Sun, *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification,* arXiv:1502.01852 (2015) .

[56] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht, *The Marginal Value of Adaptive Gradient Methods in Machine Learning,* .

[57] N. S. Keskar and R. Socher, *Improving Generalization Performance by Switching from Adam to SGD,* arXiv preprint at arXiv:1712.07628 (2017) .

[58] D. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization,* In International Conference on Learning Representations *ICLR* (2015) (2014).

[59] T. Dozat, *Incorporating nesterov momentum into adam,* Proc. Workshop Track *ICLR*, 1-4 (2016) .

[60] F. Chollet *et al.*, *Keras,* https://keras.io (2015).

[61] Amazon, *Amazon EC2 Instances,* https://aws.amazon.com/ec2/instance-types/c5/ (2019).

[62] T. Shaikhina and N. A. Khovanova, *Handling limited datasets with neural networks in medical applications: A small-data approach,* Artificial intelligence in medicine **75**, 51-63 (2017) .

Appendices

A

SIGNAL DETECTION

Magnetic flux through the coil is given by

$$\Phi(t) = \int \vec{B}_r(\mathbf{r}) \cdot \vec{M}(\mathbf{r}, t) d\mathbf{r} \tag{A.1}$$

where $\vec{B}_r(\mathbf{r})$ is the laboratory frame magnetic field at location $\mathbf{r} = (x, y, z)$. By Faraday's law of induction, the induced voltage signal in the coil is

$$V(t) = -\frac{\partial \Phi(t)}{\partial t} = -\frac{\partial}{\partial t} \int \vec{B}_r(\mathbf{r}) \cdot \vec{M}(\mathbf{r}, t) d\mathbf{r} \tag{A.2}$$

where

$$\vec{B}_r = B_{r,x}\vec{i} + B_{r,y}\vec{j} + B_{r,z}\vec{k} \tag{A.3}$$

Equation (A.2) is the simplest formula for signal detection and can be rewritten in scalar form as

$$V(t) = -\int \left[B_{r,x}(\mathbf{r}) \frac{\partial M_x(\mathbf{r}, t)}{\partial t} + B_{r,y}(\mathbf{r}) \frac{\partial M_y(\mathbf{r}, t)}{\partial t} \right] dr \tag{A.4}$$

As M_z varies slowly compared to the free precession of M_x and M_y the term $B_{r,z}(\mathbf{r})M_z(\mathbf{r}, t)$ is ignored in Equation (A.4). To further develop the expression of $V(t)$, $B_{r,x}$ and $B_{r,y}$ are rewritten as

$$\begin{cases} B_{r,x} = |B_{r,xy}(\mathbf{r})| \cos(\phi_r(\mathbf{r})) \\ B_{r,y} = |B_{r,xy}(\mathbf{r})| \sin(\phi_r(\mathbf{r})) \end{cases} \tag{A.5}$$

where $\phi_r(\mathbf{r}) \in [0, 2\pi]$ is the reception phase angle. From the free precession Equation (2.16) we obtain

$$\begin{cases} M_x(\mathbf{r}, t) = |M_{xy}(\mathbf{r}, 0)| e^{-t/T_2(\mathbf{r})} \cos[-\omega(\mathbf{r})t + \phi_e(\mathbf{r})] \\ M_y(\mathbf{r}, t) = |M_{xy}(\mathbf{r}, 0)| e^{-t/T_2(\mathbf{r})} \sin[-\omega(\mathbf{r})t + \phi_e(\mathbf{r})] \end{cases} \tag{A.6}$$

Here, $\phi_e(\mathbf{r}) \in [0, 2\pi]$ denotes the initial phase shift introduced by the RF excitation, $\omega(\boldsymbol{r})$ is the precessional frequency, and $M_{xy}(\mathbf{r}, 0)$ is the equilibrium magnetization. The derivative of (A.6) with respect to t is

$$\begin{cases} \frac{\partial M_x(\mathbf{r}), t}{\partial t} = \omega(\mathbf{r}) |M_{xy}(\mathbf{r}, 0)| e^{-t/T_2(\mathbf{r})} \sin[-\omega(\mathbf{r})t + \phi_e(\mathbf{r})] - \frac{1}{T_2(\mathbf{r})} |M_{xy}(\mathbf{r}, 0)| e^{-t/T_2(\mathbf{r})} \cos[-\omega(\mathbf{r})t + \phi_e(\mathbf{r})] \\ \frac{\partial M_y(\mathbf{r}), t}{\partial t} = -\omega(\mathbf{r}) |M_{xy}(\mathbf{r}, 0)| e^{-t/T_2(\mathbf{r})} \cos[-\omega(\mathbf{r})t + \phi_e(\mathbf{r})] - \frac{1}{T_2(\mathbf{r})} |M_{xy}(\mathbf{r}, 0)| e^{-t/T_2(\mathbf{r})} \sin[-\omega(\mathbf{r})t + \phi_e(\mathbf{r})] \end{cases} \tag{A.7}$$

Because free precession is at a much faster rate than relaxation $\omega(\mathbf{r}) \gg \frac{1}{T_2(\mathbf{r})}$, the second terms are omitted in Equation (A.7), resulting in

$$\begin{cases} \frac{\partial M_x(\mathbf{r}), t}{\partial t} = \omega(\mathbf{r}) |M_{xy}(\mathbf{r}, 0)| e^{-t/T_2(\mathbf{r})} \sin[-\omega(\mathbf{r})t + \phi_e(\mathbf{r})] \\ \frac{\partial M_y(\mathbf{r}), t}{\partial t} = -\omega(\mathbf{r}) |M_{xy}(\mathbf{r}, 0)| e^{-t/T_2(\mathbf{r})} \cos[-\omega(\mathbf{r})t + \phi_e(\mathbf{r})] \end{cases} \tag{A.8}$$

Substituting Equations (A.5) and (A.8) into Equation (A.4) and simplifying gives

$$V(t) = \int \omega(\mathbf{r}) |B_{r,xy}(\mathbf{r})| |M_{xy}(\mathbf{r},0)| e^{-t/T_2(\mathbf{r})} \cos\left[-\omega(\mathbf{r})t + \phi_e(\mathbf{r}) - \phi_r(\mathbf{r}) + \frac{\pi}{2}\right] d\mathbf{r} \tag{A.9}$$

where $B_{r,xy}(\mathbf{r})$ is the detection sensitivity of the receiver coil, $\omega(\mathbf{r})$ is the free precession frequency, $\phi_r(\mathbf{r})$ is the reception phase angle and $\phi_e(\mathbf{r})$ is the initial phase shift introduced by RF excitation. Because $M_{xy}(\mathbf{r},0)$ precesses at the Larmor frequency, $V(t)$ is a high-frequency signal, which can cause problems in later processing stages. Using the phase-sensitive detection method (PSD), or signal demodulation method, $V(t)$ is moved to a low-frequency band. The PSD method, see Figure A.1a, consists of two steps: multiplying $V(t)$ by a sinusoidal reference signal $2\cos\omega_0 t$ and removing the high-frequency component by applying low-pass-filtering. The output of PSD is a low-frequency signal given by

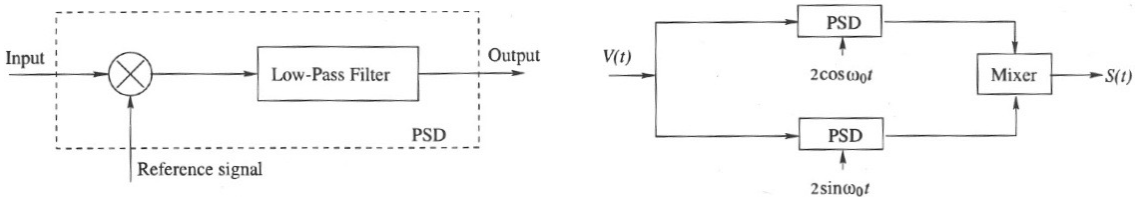
$$V_{\text{psd}}(t) = S_R(t) = \int \omega(\mathbf{r}) |B_{r,xy}(\mathbf{r})| |M_{xy}(\mathbf{r},0)| e^{-t/T_2(\mathbf{r})} \cos\left[-\Delta\omega(\mathbf{r})t + \phi_e(\mathbf{r}) - \phi_r(\mathbf{r}) + \frac{\pi}{2}\right] d\mathbf{r} \tag{A.10}$$

where $\omega(\mathbf{r}) = \omega_0 + \Delta\omega(\mathbf{r})$, with $\Delta\omega(\mathbf{r})$ the spatially dependent resonance frequency in the rotating frame. It cannot be determined if the precession is clockwise ($\Delta\omega > 0$) or counterclockwise ($\Delta\omega < 0$). Therefore $V(t)$ is multiplied with reference signal $2\sin\omega_0 t$ and passed through the low-pass filter again:

$$V_{\text{psd}}(t) = S_I(t) = \int \omega(\mathbf{r}) |B_{r,xy}(\mathbf{r})| |M_{xy}(\mathbf{r},0)| e^{-t/T_2(\mathbf{r})} \sin\left[-\Delta\omega(\mathbf{r})t + \phi_e(\mathbf{r}) - \phi_r(\mathbf{r}) + \frac{\pi}{2}\right] d\mathbf{r} \tag{A.11}$$

Detecting rotating magnetization using two orthogonal detectors is called quadrature detection (QD), see Figure A.1b. The output of QD is often given in complex form:

$$S(t) = S_R(t) + iS_I(t) = \int \omega(\mathbf{r}) |B_{r,xy}(\mathbf{r})| |M_{xy}(\mathbf{r},0)| e^{-t/T_2(\mathbf{r})} e^{-i[\Delta\omega(\mathbf{r})t - \phi_e(\mathbf{r}) + \phi_r(\mathbf{r}) + \pi/2]} d\mathbf{r} \tag{A.12}$$



(a) PSD (b) Quadrature detection

Figure A.1: Illustration of PSD [4].

Using the complex conjugate of the reception field $B^*_{r,xy}$ of $B_{r,xy}$ and

$$\begin{cases} |B_{r,xy}(\mathbf{r})| e^{-i\phi_r(\mathbf{r})} = B^*_{r,xy}(\mathbf{r}) \\ |M_{xy}(\mathbf{r},0)| e^{i\phi_e(\mathbf{r})} = M_{xy}(\mathbf{r},0) \end{cases} \tag{A.13}$$

Equation (A.14) can be rewritten as

$$S(t) = \int \omega(\mathbf{r}) B^*_{r,xy}(\mathbf{r}) M_{xy}(\mathbf{r},0) e^{-t/T_2(\mathbf{r})} e^{-i\Delta\omega(\mathbf{r})t} d\mathbf{r} \tag{A.14}$$

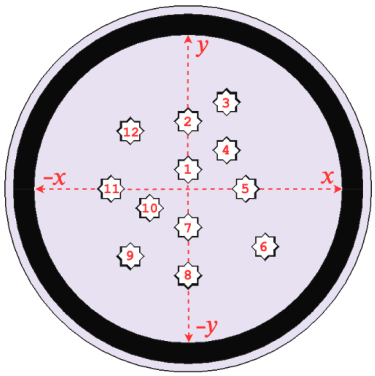
where the scaling constant $e^{-i\pi/2}$ is omitted. Furthermore, when considering an high field MRI $B_{r,xy}$ is homogeneous and the signal expression can be further simplified to

$$S(t) = \int \omega(\mathbf{r}) M_{xy}(\mathbf{r},0) e^{-t/T_2(\mathbf{r})} e^{-i\Delta\omega(\mathbf{r})t} d\mathbf{r} \tag{A.15}$$

B

MEASURED DATA SET

The position of the holes in the phantom are visualized in Figure B.1. The radius of the phantom and the position of the holes are given in millimeters and scaled down to a 0-1 range.



Hole number	Position (x, y)
1	(0,0.05)
2	(0,0.175)
3	(0.1,0.225)
4	(0.1,0.1)
5	(0.15,0)
6	(0.2,-0.15)
7	(0,-0.1)
8	(0,-0.225)
9	(-0.15,-0.175)
10	(-0.1,-0.05)
11	(-0.2,0)
12	(-0.15,0.15)

Figure B.1

The width and height of each shape placed in the phantom is given in Table B.7

	Ellipse		Circle	Square	Rectangle	
Shape	Length	Width	Length	Length	Length	Width
1.	60	20	35	40	40	30
2.	50	10	30	30	50	20
3.	40	30	25	25	40	20
4.	40	20	20	20	30	20
5.	30	10	-	-	-	-
6.	20	10	-	-	-	-

Table B.1: Length l and width w for each shape given in millimeters.

Each shape can be rotated in the holes in the phantom. The possible rotations for each shape in an arbitrary hole is given in Figure B.2.

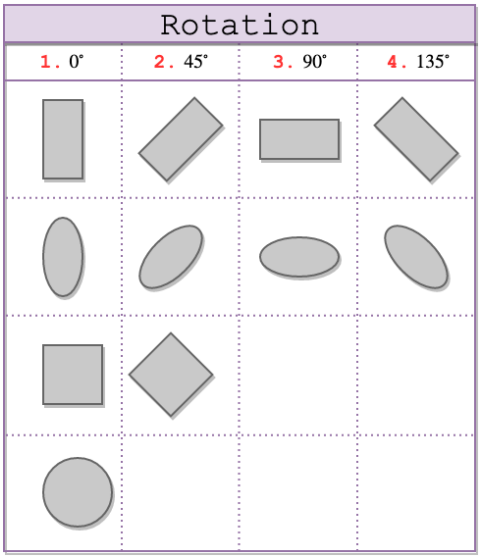


Figure B.2

The description of each sample is given in the Tables below.

Samples Measured Data set					
Sample_i	Signal_i	Image_i			
Sample_1	Signal_1		Image_1		
		Shape	*Hole*	*Rotation*	
		Ellipse_1	9	4	
		Ellipse_3	2	3	
		Circle_3	6	1	
Sample_2	Signal_2		Image_2		
		Shape	*Hole*	*Rotation*	
		Ellipse_1	9	4	
		Ellipse_3	4	2	
Sample_3	Signal_3		Image_3		
		Shape	*Hole*	*Rotation*	
		Ellipse_1	2	3	
		Ellipse_3	9	4	
		Ellipse_4	6	2	
Sample_4	Signal_4		Image_3		
		Shape	*Hole*	*Rotation*	
		Ellipse_1	5	1	
		Ellipse_3	9	4	
		Ellipse_5	12	4	
Sample_5	Signal_5		Image_3		
		Shape	*Hole*	*Rotation*	
		Ellipse_2	4	4	
		Ellipse_3	8	3	
		Ellipse_4	11	2	

Table B.2

Samples Measured Data set					
Sample_i	Signal_i		Image_i		
Sample_6	Signal_6		Image_6		
		Shape	*Hole*	*Rotation*	
		Circle_1	12	1	
		Circle_2	8	1	
Sample_7	Signal_7		Image_7		
		Shape	*Hole*	*Rotation*	
		Circle_1	2	1	
		Circle_2	9	1	
		Circle_3	6	1	
Sample_8	Signal_8		Image_8		
		Shape	*Hole*	*Rotation*	
		Circle_1	1	1	
Sample_9	Signal_9		Image_9		
		Shape	*Hole*	*Rotation*	
		Circle_2	5	1	
		Circle_3	9	1	
Sample_10	Signal_10		Image_10		
		Shape	*Hole*	*Rotation*	
		Square_1	1	2	
Sample_11	Signal_11		Image_11		
		Shape	*Hole*	*Rotation*	
		Square_2	2	1	
		Square_3	8	1	
Sample_12	Signal_12		Image_12		
		Shape	*Hole*	*Rotation*	
		Square_2	5	2	
		Square_3	12	2	
		Square_4	9	1	
Sample_13	Signal_13		Image_13		
		Shape	*Hole*	*Rotation*	
		Square_1	10	2	
Sample_14	Signal_14		Image_14		
		Shape	*Hole*	*Rotation*	
		Square_2	5	2	
		Square_3	9	2	
Sample_15	Signal_15		Image_15		
		Shape	*Hole*	*Rotation*	
		Rectangle_1	5	1	
		Rectangle_4	11	4	
Sample_16	Signal_16		Image_16		
		Shape	*Hole*	*Rotation*	
		Rectangle_2	1	2	

Table B.3

Samples Measured Data set					
Sample_i	Signal_i	Image_i			
Sample_17	Signal_17	Image_17			
		Shape	*Hole*	*Rotation*	
		Rectangle_1	2	3	
		Rectangle_3	8	3	
Sample_18	Signal_18	Image_18			
		Shape	*Hole*	*Rotation*	
		Rectangle_2	4	4	
		Rectangle_4	11	2	
Sample_19	Signal_19	Image_19			
		Shape	*Hole*	*Rotation*	
		Rectangle_3	11	1	
		Rectangle_4	5	4	
Sample_20	Signal_20	Image_20			
		Shape	*Hole*	*Rotation*	
		Circle_1	5	1	
		Ellipse_3	11	1	
Sample_21	Signal_21	Image_21			
		Shape	*Hole*	*Rotation*	
		Circle_2	12	1	
		Circle_3	6	1	
		Ellipse_2	9	4	
Sample_22	Signal_22	Image_22			
		Shape	*Hole*	*Rotation*	
		Circle_2	11	1	
		Ellipse_2	4	4	
		Ellipse_4	8	2	
Sample_23	Signal_23	Image_23			
		Shape	*Hole*	*Rotation*	
		Circle_1	11	1	
		Ellipse_1	5	1	
Sample_24	Signal_24	Image_24			
		Shape	*Hole*	*Rotation*	
		Circle_3	6	1	
		Ellipse_3	9	4	
		Ellipse_4	2	2	
Sample_25	Signal_25	Image_25			
		Shape	*Hole*	*Rotation*	
		Circle_1	2	1	
		Ellipse_4	6	2	
		Ellipse_5	9	2	
Sample_26	Signal_26	Image_26			
		Shape	*Hole*	*Rotation*	
		Circle_3	9	1	
		Square_2	4	2	
Sample_27	Signal_27	Image_27			
		Shape	*Hole*	*Rotation*	
		Circle_2	9	1	
		Circle_3	5	1	
		Square_3	12	1	

Table B.4

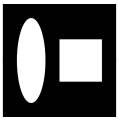
Samples Measured Data set					
Sample_i	Signal_i	Image_i			
Sample_28	Signal_28	Image_28			
		Shape	*Hole*	*Rotation*	
		Circle_2	11	1	
		Square_2	5	1	
Sample_29	Signal_29	Image_29			
		Shape	*Hole*	*Rotation*	
		Circle_3	6	1	
		Square_2	2	2	
		Square_3	9	2	
Sample_30	Signal_30	Image_30			
		Shape	*Hole*	*Rotation*	
		Circle_1	2	1	
		Square_3	8	1	
Sample_31	Signal_31	Image_31			
		Shape	*Hole*	*Rotation*	
		Ellipse_2	1	1	
		Ellipse_4	6	2	
		Rectangle_3	11	1	
Sample_32	Signal_32	Image_32			
		Shape	*Hole*	*Rotation*	
		Ellipse_1	7	2	
		Rectangle_4	12	2	
Sample_33	Signal_33	Image_33			
		Shape	*Hole*	*Rotation*	
		Ellipse_4	3	4	
		Rectangle_2	7	3	
Sample_34	Signal_34	Image_34			
		Shape	*Hole*	*Rotation*	
		Ellipse_3	2	4	
		Ellipse_5	6	1	
		Rectangle_4	8	1	
Sample_35	Signal_35	Image_35			
		Shape	*Hole*	*Rotation*	
		Ellipse_4	9	4	
		Rectangle_1	4	4	
Sample_36	Signal_36	Image_36			
		Shape	*Hole*	*Rotation*	
		Ellipse_4	3	4	
		Square_1	10	2	
Sample_37	Signal_37	Image_37			
		Shape	*Hole*	*Rotation*	
		Ellipse_3	12	2	
		Square_3	5	2	
		Square_4	8	2	
Sample_38	Signal_38	Image_38			
		Shape	*Hole*	*Rotation*	
		Ellipse_1	11	1	
		Square_2	5	1	

Table B.5

Samples Measured Data set					
Sample_i	Signal_i	Image_i			
Sample_39	Signal_39		Image_39		
		Shape	*Hole*	*Rotation*	
		Ellipse_2	2	3	
		Ellipse_4	6	2	
		Square_3	9	2	
Sample_40	Signal_40		Image_40		
		Shape	*Hole*	*Rotation*	
		Ellipse_3	8	3	
		Ellipse_5	11	3	
		Square_3	4	1	
Sample_41	Signal_41		Image_41		
		Shape	*Hole*	*Rotation*	
		Rectangle_1	4	2	
		Square_3	9	2	
Sample_42	Signal_42		Image_42		
		Shape	*Hole*	*Rotation*	
		Rectangle_3	11	1	
		Square_2	5	2	
Sample_43	Signal_43		Image_43		
		Shape	*Hole*	*Rotation*	
		Rectangle_2	1	4	
		Square_3	9	2	
Sample_44	Signal_44		Image_44		
		Shape	*Hole*	*Rotation*	
		Rectangle_4	12	2	
		Square_3	5	2	
		Square_4	9	1	
Sample_45	Signal_45		Image_45		
		Shape	*Hole*	*Rotation*	
		Rectangle_3	5	1	
		Square_2	11	1	
Sample_46	Signal_46		Image_46		
		Shape	*Hole*	*Rotation*	
		Circle_3	8	1	
		Ellipse_1	4	4	
		Square_3	11	2	
Sample_47	Signal_47		Image_47		
		Shape	*Hole*	*Rotation*	
		Circle_2	9	1	
		Ellipse_4	12	2	
		Rectangle_2	5	1	
Sample_48	Signal_48		Image_48		
		Shape	*Hole*	*Rotation*	
		Ellipse_2	9	4	
		Rectangle_4	6	2	
		Square_2	2	1	
Sample_49	Signal_49		Image_49		
		Shape	*Hole*	*Rotation*	
		Ellipse_3	12	2	
		Rectangle_3	8	3	
		Square_4	5	1	

Table B.6

Samples Measured Data set					
Sample_i	Signal_i	Image_i			
Sample_50	Signal_50		Image_50		
		Shape	*Hole*	*Rotation*	
		Circle_3	8	1	
		Ellipse_5	11	1	
		Rectangle_1	4	4	
Sample_51	Signal_51		Image_51		
		Shape	*Hole*	*Rotation*	
		Circle_3	11	1	
		Ellipse_4	8	3	
		Rectangle_3	4	4	
Sample_52	Signal_52		Image_52		
		Shape	*Hole*	*Rotation*	
		Circle_3	3	1	
		Rectangle_4	6	2	
		Square_2	11	1	
Sample_53	Signal_53		Image_53		
		Shape	*Hole*	*Rotation*	
		Circle_3	3	1	
		Ellipse_5	5	3	
		Rectangle_4	11	2	
		Square_3	8	1	

Table B.7

C

ADDITIONAL VERSIONS OF MODEL II

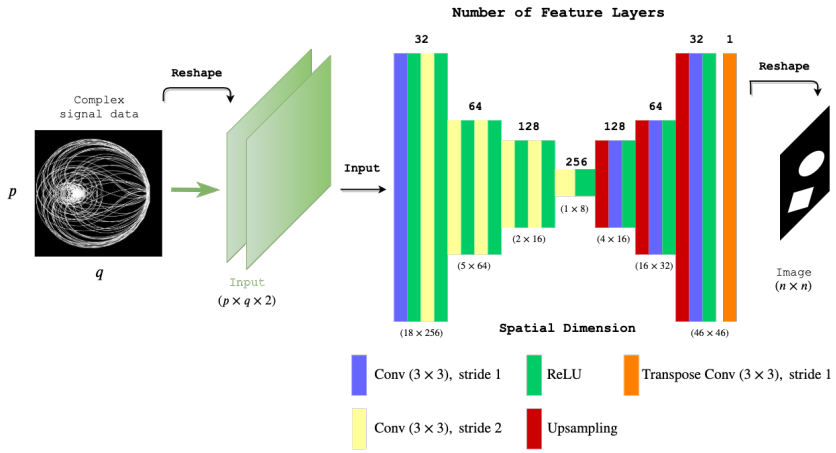


Figure C.1: Schematic representation of the architecture.

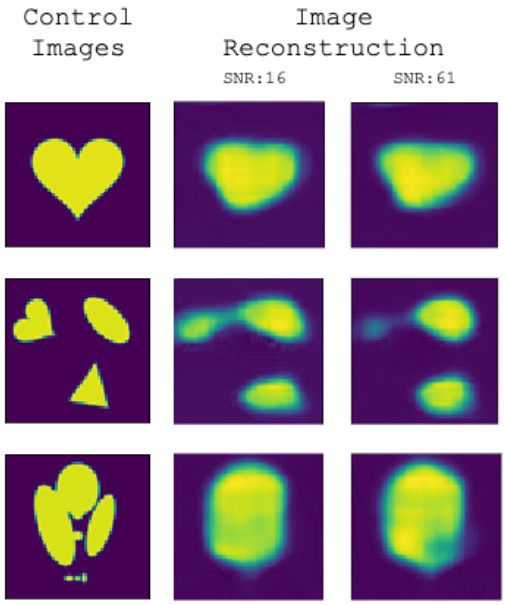


Figure C.2: Image reconstruction.

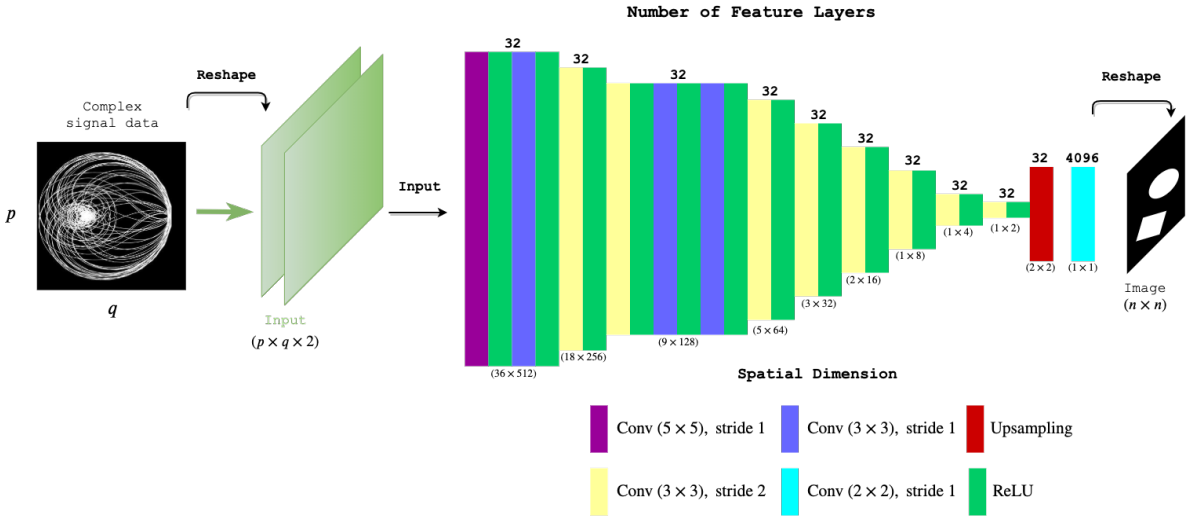


Figure C.3: Schematic representation of the architecture.

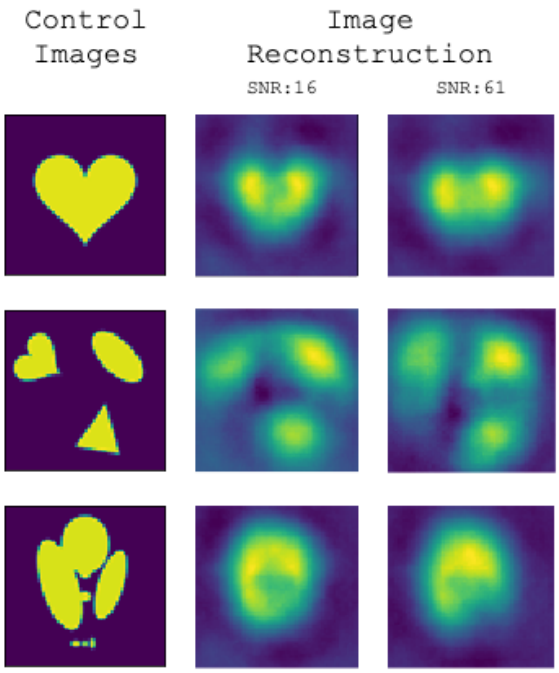


Figure C.4: Image reconstruction.