A DEEP LEARNING METHOD FOR QUANTITATIVE SUSCEPTIBILITY MAPPING SINGULARITY CORRECTION

by

Adeola Oluwasanjo Aderemi

B.S., Electrical and Electronics Engineering, University of Lagos, 2018

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Electrical and Electronics Engineering
Boğaziçi University

2025

ACKNOWLEDGEMENTS

I am super grateful to my family for their incredible support over the course of this program.

I also want to thank Prof Alpay Özcan for his support and his willingness to sit down with me and offer his invaluable guidance throughout the course of this thesis. I am grateful.

I appreciate Memunat for not leaving me to my reticence and being nice enough to have a virtual writing session together. And to Susanna who sends me pictures of Charlie and cheers me up.

I am of course also super grateful to everyone in the BUSSML lab, including Yildiz, Irem, Hamit, Merve and Öykü, for being great lab members. And a very big thank you to Türkiye Scholarships for sponsoring my program. Special thanks also to my thesis examiners, Prof. Burak Acar and Dr. Engin Baysoy for their time and insightful questions.

This work was supported by the Scientific and Technological Research Council of Turkey (TUBITAK) under the Grant Number 122E508 titled "Geriye Dönük Klinik MRG Tarayıcı Verilerinin Niceliksel Duyarlılık Haritalama Yöntemleriyle İşlenmesi ve Yöntemin Küçük Hayvan MRG Tarayıcısında Gerçekleştirilmesi/Processing Retrospective Clinical MRI Scanner Data with Quantitative Susceptibility Mapping and its Implementation on a Small Animal MRI Scanner".

ABSTRACT

A DEEP LEARNING METHOD FOR QUANTITATIVE SUSCEPTIBILITY MAPPING SINGULARITY CORRECTION

Brain susceptibility maps have been used clinically to study the progression of neurodegenerative diseases such as Alzheimer's disease, Parkinson's disease and Multiple Sclerosis. One way of obtaining these brain susceptibility maps is through Quantitative Susceptibility Mapping (QSM). QSM is an inverse problem that aims to calculate susceptibility from MRI phase images. This is possible because the MRI phase image is, up to a constant, a convolution of a dipole kernel filter with an underlying susceptibility distribution. The problem is an inverse problem because the 3D dipole filter has zeros in the Fourier domain, which means direct inversion is impossible. Recently, deep learning-based solutions have been proposed to solve this problem. In this work, using synthetically generated data, we first examine the UNet deep learning architecture which has been the default architecture in the QSM community. We note that using a different variant can speed up training convergence with up to a 21% reduction in validation mean absolute error. Furthermore, we propose a QSM reconstruction method that solves the QSM problem by dividing the problem into two parts in the Fourier domain. On the one hand, we solve for susceptibility values by direct division in the Fourier domain in regions where the dipole kernel is stable. On the other hand, for regions where the kernel is zero, we train a deep learning model to learn the correct susceptibility map. Our final map is obtained by merging these two maps. We show that our reconstruction method works by testing on a generated test set. Our approach was also tested using data from a QSM challenge. On this data, we achieved the top position when compared with the other models tested during that challenge.

ÖZET

KANTİTATİF MANYETİK DUYARLILIK HARİTALAMASI TEKİLLİK DÜZELTMESİ İÇİN BİR DERİN ÖĞRENME YÖNTEMİ

Beyin duyarlılık haritaları, Alzheimer hastalığı, Parkinson hastalığı ve Multipl Skleroz gibi nörodejeneratif hastalıkların ilerleyişini incelemek için klinik olarak kullanılmaktadır. Bu haritaların elde edilme yollarından biri, Kantitatif Duyarlılık Haritalama (Quantitative Susceptibility Mapping, QSM) yöntemidir. QSM, MRI faz görüntülerinden duyarlılığı hesaplamayı amaçlayan bir ters problemdir. Bu mümkündür çünkü MRI faz görüntüsü, bir sabit katsayıya kadar, dipol çekirdek filtresi ile altta yatan duyarlılık dağılımının konvolüsyonudur. Bu, ters bir problemdir çünkü 3B dipol filtresi, Fourier uzayında sıfır değerler aldığından doğrudan tersine çevirme yapılamaz. Son zamanlarda, bu problemi çözmek için derin öğrenme tabanlı çözümler önerilmiştir. Bu çalışmada, sentetik olarak üretilmiş veriler kullanılarak QSM topluluğunda yaygın olarak kullanılan UNet derin öğrenme mimarisi incelenmiştir. Farklı bir mimari varyant kullanmanın, doğrulama ortalama mutlak hatasında %21'e varan azalma ile eğitim sürecinde daha hızlı yakınsamaya yol açtığını gözlemliyoruz. Ayrıca, QSM problemini Fourier uzayında iki alt probleme ayırarak çözen bir QSM yeniden yapılandırma yöntemi öneriyoruz. Dipol çekirdeğinin kararlı olduğu bölgelerde doğrudan bölme yöntemiyle duyarlılık değerlerini hesaplıyoruz. Çekirdeğin sıfır olduğu bölgelerde ise, doğru duyarlılık haritasını öğrenmesi için bir derin öğrenme modeli eğitiyoruz. Sonuç duyarlılık haritamız, bu iki haritanın birleştirilmesiyle elde edilmektedir. Yöntemimizin başarısını, oluşturulan bir test veri seti üzerinde test ederek gösteriyoruz. Ayrıca yaklaşımımız, bir QSM yarışmasındaki verilerle de test edilmiştir ve bu yarışmada test edilen diğer modellerle karşılaştırıldığında en iyi performansı göstermiştir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

B_0	Main Magnetic field
d	Dipole kernel in image space
D	Dipole kernel in Fourier space
FT	Fourier Transform
$g(x)$	Activation function
k	Frequency index in k-space
L	Loss function
$loss$	Loss functions for deep learning methods reviewed
M	Mask, defined according to context
N	Number of samples
$S(t)$	MR signal in time
t	Time
TE	Echo time
$thresh$	Threshold value in k-space
TV	Total Variation
V	Volume
W	Weighting function

α	Blending parameter *or* scale
$\beta_t(i)$	Backward variable
χ	Susceptibility
ΔB	Magnetic perturbation
ϵ	Learning rate
γ	Gyromagnetic ratio
∇	Derivative
ω_0	Lamor Frequency
Ω	Regularisation function

ϕ	Phase
$\sigma(x)$	Sigmoid activation function
Θ	Parameter set

LIST OF ACRONYMS/ABBREVIATIONS

3D	Three Dimensional
ASPEN	Approximated Susceptibility through Parcellated Encoder-decoder Networks
BN	Batch Normalization
CNN	Convolutional Neural Network
COSMOS	Calculation of Susceptibility through Multiple Orientation Sampling
DIP	Deep Image Prior
DL	Deep Learning
DNN	Deep Neural Network
FFT	Fast Fourier Transform
FINE	Fidelity Imposed Network Edit
FT	Fourier Transform
GPU	Graphics Processing Unit
GRE	Gradient Echo
iLSQR	Iterative Sparse Linear Equation and Least-squares Algorithm
ISBI	International Symposium on Biomedical Imaging
LR	Learning Rate
MAE	Mean Absolute Error
MEDI	Morphology Enabled Dipole Inversion
ML	Machine Learning
MoDIP	Model-based Deep Image Prior
MR	Magnetic Resonance
MRI	Magnetic Resonance Imaging
NRMSE	Normalized Root Mean Square Error
OCR	Optical Character Recognition
OS	Operating System
QSM	Quantitative Susceptibility Mapping

ReLU	Rectified Linear Unit
RF	Radio Frequency
ROI	Region of Interest
SGD	Stochastic Gradient Descent
SNR	Signal to Noise Ratio
STAR-QSM	Streaking Artifact Reduction for QSM
TKD	Truncated K-Space Division

1. INTRODUCTION

In MR imaging, differences in tissue susceptibility can create varying magnetic field strengths across tissues which can in turn introduce a location-dependent (or tissue) phase difference in the acquired MR signal. Quantitative Susceptibility Mapping (QSM) is a way of quantifying the different susceptibility values responsible for these phase differences.

Clinically, susceptibility differences caused by changing tissue properties, for example iron deposition, can be useful for assessing disease status. In particular, obtaining QSM maps of these susceptibility differences has been found to be useful for the diagnosis of diseases including but not limited to Parkinson's Disease, Multiple Sclerosis (MS) and Neuro-Behcet's disease. Parkinson's Disease, for example, correlates with an increase in iron deposit in the substantia nigra in the brain. Due to its paramagnetic nature, increasing iron concentration will alter the susceptibility values of brain tissue. Being able to detect this increase in susceptibility is useful for Parkinson's Disease treatment. It has been shown that QSM is not only able to do this but it also shows higher sensitivity than alternate methods [1]. Additionally, QSM can differentiate between calcification and hemorrhage, which may be useful in tumor subclassification [2]. To further illustrate the importance of QSM, a QSM reconstruction challenge was started with the purpose of testing various QSM algorithms' ability to faithfully recover susceptibility maps from phase images, showing that the ability to obtain QSM maps is an important problem [3].

A spinning charged object such as the nuclei of an atom has a magnetic field around it, and these tiny fields in a given body sum up to zero magnetization in the absence of an external magnetic field because they are randomly-oriented [4]. In the presence of an external magnetic field B_0, the field vectors will precess around that magnetic field B_0.

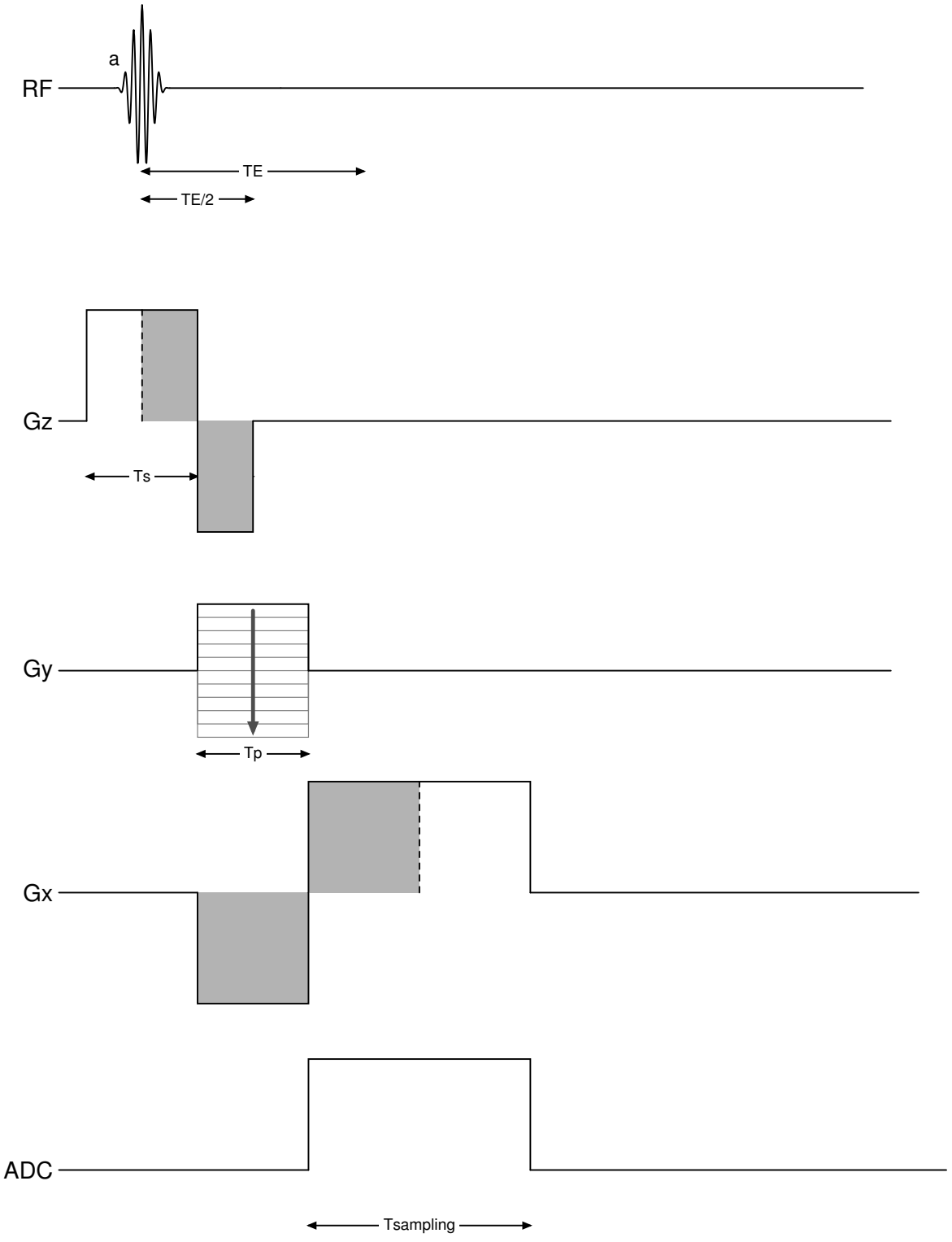


Figure 1.1. A GRE Pulse Sequence.

This means that the vertical component aligns in the direction of B_0, while the horizontal components rotate randomly around B_0 at speed $\omega_0 = \gamma B_0$. If a smaller RF pulse, B_1 with speed ω_0 is applied to the system for a short period of time, resonance is said to occur and the magnetic field vector which was initially aligned in the direction of B_0 rotates around this new field. The amount of rotation depends on how long the RF pulse was turned on, as angular rotation, α, $\alpha = \omega t$. Typically, the field B_0 is chosen to be along the z-direction while B_1 can be chosen to be along the x-direction. B_1 is usually turned on for a period of time such that the initial nuclear magnetic vector along z is rotated 90 degrees into the x-y plane. This magnetic vector in the x-y plane and in the presence of an external magnetic field in the z-direction will rotate around the z-direction. After the field B_1 has been turned off, a gradient magnetic field G is also turned on in the z-direction. It is a gradient in the sense that it produces a magnetic field that varies with distance such that the actual field strength is $G.r$. $G = [G_x, G_y, G_z]$ represents the gradient components in the xyz directions, in general. These gradient fields are used to map the signal into frequency domain, as will be subsequently shown. According to Faraday's law, the rotating magnetic field around the z direction will produce an EMF, this signal is what is collected as the MR signal. A typical Gradient Recalled Echo (GRE) sequence used for the signal generation is shown in Figure 1.1. In the figure, G_z is used for selecting the slice of interest, then both G_x and G_y are used for encoding in the x and y directions respectively.

In the absence of significant susceptibility differences in our region of interest, the MRI signal obtained can be described as [5]

$$S(t) = \int_V \rho(r) e^{-i2\pi\bar{\gamma}(B_0+Gr)t}\, dr, \tag{1.1}$$

which, in the ω_0 rotating frame of reference, becomes

$$S(t) = \int_V \rho(r) e^{-i2\pi\bar{\gamma}Grt}\, dr \tag{1.2}$$

where G is a gradient magnetic field and Gr is the magnetic field. $\bar{\gamma}$ is the gyromagnetic

ratio (γ) normalized by 2π, that is

$$\bar{\gamma} = \frac{\gamma}{2\pi}, \tag{1.3}$$

and V is the entire region over which we are integrating. At any given time, we can set $k = \bar{\gamma}Gt$ then the equation becomes

$$S(k) = \int_V \rho(r) e^{-i2\pi kr}\, dr \tag{1.4}$$

which shows that the signal obtained in the MR machine is the Fourier transform of the actual MR image signal, $\rho(r)$, where $k = \bar{\gamma}Gt$ is called the frequency variable. The variable k maps from time to frequency as already mentioned, and for a 1D imaging, we can move around in the k-space of frequency space by sampling at different times.

When susceptibility differences are present, the signal obtained is expressed as

$$S(k) = \int_V \rho(r) e^{-i\gamma \Delta B(r) TE} e^{-i2\pi kr}\, dr. \tag{1.5}$$

This equation shows that a phase is introduced into our signal in this case. The phase can be written as $\phi(r) = -\gamma \Delta B(r) TE$, where the angular speed is $\Delta\omega(r) = \gamma \Delta B(r)$. TE is the echo time and it is the time at the which the signal is sampled and the induced magnetic field is $\Delta B(r)$.

This induced magnetic field (or phase, after scaling by the gyromagnetic ratio and TE), due to a tissue's susceptibility obtained at any point, is a convolution in space between the magnetic dipole kernel and the tissue susceptibility [6], that is

$$\Delta B(r) = d(r) * \chi(r) \tag{1.6}$$

and this is referred to as the QSM forward equation in this work. The dipole kernel is

defined as

$$d(r) = \frac{3cos^2\theta - 1}{4\pi r^3}. \tag{1.7}$$

As convolution translates to multiplication in Fourier domain, this equation can be expressed as

$$FT(\Delta B(r)) = \Delta B(k) = (\frac{1}{3} - \frac{k_z^2}{k_x^2 + k_y^2 + k_z^2})FT(\chi) \tag{1.8}$$

where FT is the Fourier transform, and

$$FT(d) = D = \frac{1}{3} - \frac{k_z^2}{k_x^2 + k_y^2 + k_z^2} \tag{1.9}$$

at the given location is the dipole kernel in Fourier domain, k_x, k_y, k_z are the frequency indices and χ is the susceptibility. This relationship serves as the basis for obtaining susceptibility maps(QSM in this case) from MRI phase images.

Because of periodicity, the phase image obtained from a given MR image is discontinuous and hence arbitrary jumps must be corrected before Equation 1.8 can be inverted to obtain the susceptibility distribution. Phase unwrapping is necessary to remove these phase jumps and obtain a smooth phase distribution and several phase unwrapping methods have been developed to tackle this, such as the Laplacian phase unwrapping method [7]. Magnetic fields vary with distance, as in $B \propto 1/r^3$, therefore a field originating at any point would have effects at some distance away from itself. This means that other regions, including tissue or air outside our region of interest (ROI), would contribute to the field in which we are interested. These unwanted fields are termed background fields and they also need to be removed so that there will be only the local field. One way to achieve this is by high-pass filtering of the MR image. This is motivated by the assumption that far-away background fields are slowly varying in our ROI hence using a high-pass filter can effectively remove them. The two processes described are some of the preprocessing steps used in QSM processing. But these are

not necessary when working with simulated data in which the QSM convolution can be performed without background sources. Dipole inversion is the process of obtaining the actual QSM map from the corrected phase image. This could technically be obtained by a simple division in the Fourier domain (also known as k-space in MRI literature), as in $FT\chi = D^{-1}FT\phi$, but this proves impossible due to the presence of regions in the Fourier space where the dipole kernel, $D = 0$ or $k_z^2 = \frac{k^2}{3}$ and $D^{-1} \to \infty$.

Several approaches have been proposed to solve the QSM ill-conditioned inverse problem, and one classic example is the truncated k-space division (TKD) [8]. In TKD values of the dipole kernel below a certain threshold are set to zero and division is done in the Fourier domain, which is generally called the k-space in MR imaging with the k serving as our frequency variable,to obtain the susceptibility map. At positions corresponding to these zeros, the phase is multiplied by zero instead of dividing. Another method is Morphology Enhanced Dipole Inversion (MEDI) which casts the problem as finding the projection of the phase image into the dipole kernel space with the constraint that the gradient of the susceptibility map in regions in the magnitude image that are not edges are also minimized [9]. In Calculation of Susceptibility through Multiple Orientation Sampling (COSMOS) [10], the susceptibility image is obtained from phase images obtained at various orientations. Although this method is considered the gold standard in QSM literature, it is both time-consuming and can be clinically impractical. These methods are further discussed in the following chapter.

Recently, deep learning has been proposed to solve this QSM dipole inversion [11]. Deep Learning approaches to dipole inversion try to obtain the QSM map from a single orientation phase image by using a deep neural network to minimize a loss function, the L_1 loss for example, between a label such as a COSMOS map (or known phantom susceptibility map) and the network's output with the aim of generating an output image similar to the label given just the input phase image [11]. This can potentially prove very useful to solve the QSM problem. In the case of using COSMOS labels, deep learning can enable us to obtain COSMOS map quality susceptibility maps, without the need to make multiple MR phase measurements.

In this thesis, we propose a novel deep learning based method for the QSM reconstruction problem. Furthermore, we show that making a simple modification on the UNet deep learning architecture can lead to faster convergence when training the UNet for QSM reconstruction.

2. QSM METHODS

As already mentioned in the introduction, several methods have already been proposed for the QSM inversion problem. Some of those methods will be described in this chapter. The methods will be categorised as deep learning and non-deep learning methods.

2.1. Non Deep Learning Methods

Some non deep learning methods have already been mentioned in the introductory chapter. These methods will be expanded on here and other methods will be added as well.

2.1.1. Truncated K-Space Division(TKD)

TKD was originally introduced in [12]. It is one of the simplest methods for obtaining a QSM map from a phase image. Mathematically, the TKD solution is defined as

$$\chi(k) = \frac{sgn(D(k))\Delta B(k)}{max(|D(k)|, thresh)} \tag{2.1}$$

where sgn represents the signum operation which returns the sign of a particular voxel of dipole kernel in k-space, ΔB, defined in Equation 1.6, is the magnetic field perturbation obtained from the phase image, $thresh$ is a small threshold value chosen to be between 0 and $\frac{2}{3}$ and max is the maximum operator. There are several implementations of this algorithm online and some do not use the signum operation, preferring instead to replace the areas smaller than the threshold with zero instead of replacing by the threshold. In [12], the threshold value between 0.2 and 0.5 were reported to effectively reduce streaking artifacts.

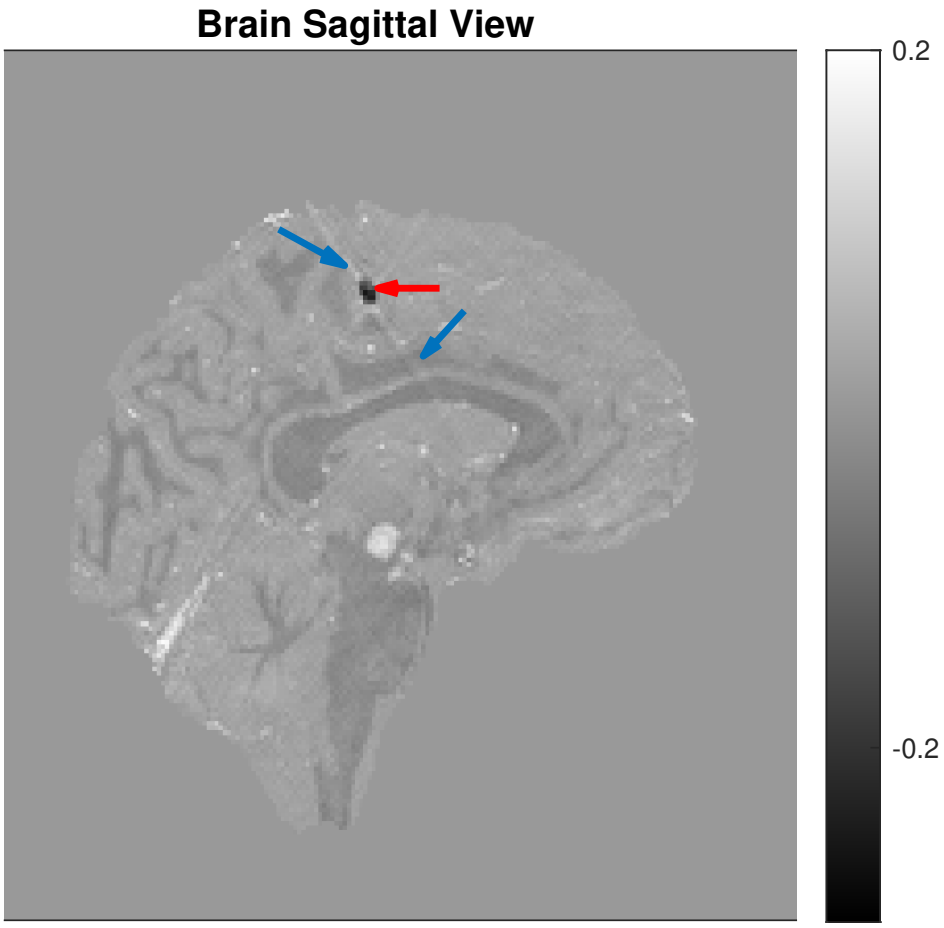


Figure 2.1. Streaking artifacts in a phantom brain image. Red arrow points to a region with calcification and the blue arrows point at TKD streaks. Map generated from data provided during QSM challenge 2.0 [13].

Smaller threshold values are generally avoided as they can lead to severe streaking artifacts in the presence of noise [14] while susceptibility becomes underestimated with increasing threshold values. Streaking artifacts from susceptibility maps reconstructed with TKD is illustrated in Figure 2.1. Calcification is present in this brain phantom and it manifests as the dark spot being pointed at by the red arrow. The streaking artifacts are the diagonal lines pointing away from the calcified region. They are indicated by the blue arrows for emphasis.

2.1.2. Morphology Enabled Dipole Inversion (MEDI)

MEDI belongs to a class of algorithms that use some form of gradient descent to solve the dipole version. In general, this group of methods obtain the susceptibility distribution by minimizing a loss function as follows

$$||W(\Delta B - FT^{-1}(D \cdot FT(\chi)))||_2 + \lambda R(\chi) \tag{2.2}$$

where the forward convolution has just been evaluated in the Fourier domain. λ is a tunable regularisation coefficient, W is a weight matrix that accounts for spatially varying noise level [14] and R is the regularisation function. The main difference among algorithms of this type is the nature of regularisation applied. The regularisation is usually some form of total variation. This is generally used as it helps to get rid of the streaking artifacts described in the previous section. The streaking artifacts are a manifestation, in image domain, of the cone region of zero present in the dipole kernel in the Fourier domain. In the case of MEDI, the regularisation function is

$$R(\chi) = ||M\nabla\chi||_1 \tag{2.3}$$

where the ∇ operator indicates a 3D gradient. The mask, M, is essentially an indicator function. It is obtained by taking the gradient of the magnitude image, and setting the mask to 0 where the gradient of the magnitude image is non-negligible and 1 otherwise. Mathematically,

$$M(x,y,z) = \begin{cases} 0 & \text{if } |Gm(x,y,z)| \geq \mu \\ 1 & \text{if } |Gm(x,y,z)| < \mu \end{cases} \tag{2.4}$$

where $Gm(x,y,z)$ represents the gradient of the magnitude image in the $[x,y,z]$ directions and μ is a threshold based on image noise level. One may think of Equation 2.3 as a form of "total" variation except that the gradients are now masked. Minimizing the loss function in Equation 2.2, with the regularisation function set to that in

Equation 2.3, then sets the gradients of the reconstructed susceptibility map to zero in regions in which the magnitude image has smaller gradients (i.e where $|Gm(x, y, z)| < \mu$ and therefore $M = 1$), as the L_1 norm in Equation 2.3 would promote sparsity. The downside to this formulation is that edges in the magnitude image may not necessarily match the true anatomical boundary which means the correct boundaries may not be obtained in the reconstructed susceptibility map [14].

2.1.3. Streaking Artifact Reduction for QSM (STAR-QSM)

This method was proposed in [15] to alleviate streaking artifacts in susceptibility maps with large dynamic ranges. It is similar to the previously discussed MEDI in the sense that the problem formulation is essentially the same as in Equation 2.2. The difference is in the definition of the regularisation term. Here, the regularisation term is the L_1 norm of the gradient magnitude and is defined as

$$R(\chi) = ||W \cdot G \cdot \chi||_1 \tag{2.5}$$

where the L_1 norm is taken over the volume of the gradient magnitude which is defined as

$$W \cdot G \cdot \chi = \sqrt{(W_{Gx}G_x\chi)^2 + (W_{Gy}G_y\chi)^2 + (W_{Gz}G_z\chi)^2} \tag{2.6}$$

where G_x, G_y, G_z are the gradient operators and W_{Gx}, W_{Gy}, W_{Gz} are weighting factors. The regularised loss function can be optimized in one step to obtain the susceptibility map or it can be done in two steps with different values of regularisation parameters, λ. In this second case, a large value of λ means more regularisation which means the obtained map is smoother. This large value is used to obtain the map for the case where there is a large value of susceptibility. A smaller normal value of λ is then used to obtain the ssusceptibility map with weaker susceptibility values. The two maps are later superimposed on each other. This process is fully discussed in [15].

2.1.4. Compressed Sensing-based QSM

This method belongs to a class of methods where the QSM solution is divided into solving at the stable versus non-stable region. In this method, the QSM is obtained in the stable region via direct division, similar to the TKD mehod. Estimation in the non-stable region is done via compressed sensing. The problem here is formulated as

$$\chi' = min_{\chi}||\chi'_k \cdot M - M \cdot FT(\chi)||_2 + \alpha||\Phi\chi||_1 + \beta \cdot TV(\chi) \tag{2.7}$$

where the mask indicates stable regions. Stable regions are, as usual, defined as a region where the dipole kernel is greater than some threshold value. TV indicates total variation and Φ is a wavelet transform and α and β represent weighting coefficients. As in total variation, the $||\Phi\chi||_1$ term promotes sparsity in $\Phi\chi$. According to the authors, the method produced a streaking free map at a threshold value of 0.0625 and in general performs better than direct thresholding (TKD). This method inspired the work in this thesis and the threshold suggested here is used throughout this work.

2.1.5. iLSQR

The Iterative Sparse Linear Equation and Least-squares Algorithm(iLSQR) method also focuses on obtaining the susceptibility map in non-stable regions and relying on direct inversion for the stable regions. For this method, the authors [16] noted that although the dipole kernel is zero on the conical surface, its derivative is not. The new equation derived from this derivative and the original one are then solved iteratively to obtain the susceptibility map. It is noted in [15] that this method is unable to supress the streaking artifacts in cases where there is a large dynamic range of susceptibility distribution.

2.1.6. Calculation of Susceptibility Through Multiple Orientation Sampling (COSMOS)

All the mehtods already discussed in this chapter use single phase measurements/images to obtain a corresponding susceptibility map. In COSMOS, phase images are obtained at various orientations to avoid the zeros in the dipole kernel [10]. The problem is then again cast in the form $Ax = b$, where $A = D, x = \chi, b = \phi$ as already defined, and solved for our susceptibility, χ. Doing so leads to the COSMOS images which are thought of as a gold standard for QSM maps because they lead to good susceptibility images [14]. In addition to this, COSMOS maps also have the advantage that they lead to improved QSM maps because they also deal with the anisotropic nature of the brain by their very nature.

Orientation dependence in the susceptibility of a material is called anisotropy. In the 2016 QSM challenge [3], it was found that the anisotropy of magnetic susceptibility is a factor that affects the correctness of QSM maps. It has been observed experimentally that certain parts of the brain (e.g white matter regions) exhibit magnetic anisotropy [17] [18]. What this means is that by reorienting a brain with respect to the main magnetic field, different susceptibility level can be observed in the QSM map. And as susceptibility is responsible for the observed phase image, this also implies that the acquired phase image would change with changing orientation.

COSMOS maps have been shown to give better results specifically in anisotropic regions of the brain than methods that use single orientation to solve the inverse problem. It was shown using the brain of a female chimpanzee [18] that a COSMOS map obtained at orientations 0, 60 and 120 degrees provides better result in white matter regions than a QSM map obtained using a single-oritentation method such as the iterated orthogonal and right angle decomposition (iLSQR) method. This is intuitive as COSMOS maps are obtained by using multiple different orientations. Although COSMOS maps offer many advantages, obtaining them is time-consuming and impractical for routine lab work. Apart from the time it would take to obtain images of the object

at different orientations, it may additionally be impractical to orient a human in the MRI machine at the optimal angles required for obtaining COSMOS images.

2.2. Deep Learning Based Methods for QSM

Deep learning has recently found use in a lot of signal processing applications, including in quantitative susceptibility mapping. As deep learning is a rapidly growing field, several deep learning solutions have been proposed to the QSM problem. Some of those solutions are described here and several others can still be found in literature.

2.2.1. QSM Using Deep Neural Network:QSMnet and QSMnet+

In [19], the authors proposed to use a 3D UNet architecture to learn the mapping from an input phase map to a susceptibility map. They do this by minimizing a loss function between the model output and a ground truth obtained using the COSMOS algorithm. Their algorithm offers the advantage that it provides a way to obtain a COSMOS map using only a single input phase map which does not require the patient to rotate their head in impossible angles. As training data, they obtained in vivo MRI data from twelve patients. For each patient, phase images in five orientations were obtained. Those images were used to obtain ground truth COSMOS maps which then served as the ground truth for training and testing. They further used rotation as a form of data augmentation to increase the size of their training data. In addition to this, they cropped the obtained phase and susceptibility into $64 \times 64 \times 64$ patches, further increasing the amount of data available for training. Training was done on these patches but testing was done on full images. Their loss function was defined as

$$Loss = w_1 loss_{model} + w_2 loss_{L1} + w_3 loss_{Gradient} \tag{2.8}$$

where w_1, w_2, w_3 are loss weights they chose empirically, $loss_{model}$ is defined as

$$loss_{model} = ||d * \chi - d * y||_1 \tag{2.9}$$

which is the l_1 of the difference between dipole convolution of the output and the dipole convolution of the label. χ and y are the model output and COSMOS label respectively. The loss is suppposed to enforce consistency in the dipole model between the label and the output. The second loss in Equation 2.8, $loss_{L1}$, is just the l_1 of the difference between the output and the label. The third loss is meant to preserve edge information between reconstructed maps and labels, it is defiend as

$$loss_{gradient} = ||\nabla\chi - \nabla y|| + ||\nabla d * \chi - \nabla d * y|| \tag{2.10}$$

where ∇ indicates 3D gradients.

One of the limitations of this approach was that it did not generalize well to susceptibility values that are outside the range of the training data. The authors in [20] aim to address this limitation by introducing a data augmentation technique which allows them to create simulation data for a wider range of susceptibility values. This approach was called $QSMnet^+$. In addition to this limitation, several other limitations were listed in [19] such as the potential inability of the network to generalize to inputs at resolutions different to the training data resolution. One other limitation is the cropping of the trainig data to increase training data size. As convolution is nonlocal, a crop of the phase image will not capture the entire effect of the susceptibilty at the same location even though that is used as its label. This implies the model can not really learn the deconvolution that maps a phase image to the true susceptibility map, but this issue has not been explored.

2.2.2. DeepQSM

In [21], the authors proposed to solve the QSM inverse problem by training a deep neural network on simulated digital phantoms. They did this by simulating the susceptibility distribution from a uniform distribution between -0.2 and 0.2. The QSM forward equation was then used to simulate the corresponding phase. As in the method described in Section 2.2.1, they also used a 3D UNet as their architecture of choice.

Their simulated data was made up of simple geometric shapes such as spheres and rectangular prisms. Each shape has a specific susceptiblity value. They were able to show that a deep neural network may be used to approximate the QSM dipole deconvolution and that training data with anatomical priors may not be necessary to learn this deconvolution. As their loss function, they used a simple l_2 difference between the network output and their simulated ground truth. The loss is defined as

$$||\chi - y||_2. \tag{2.11}$$

For their data they generated $160 \times 160 \times 160$ susceptibility volumes which they then convolve with the dipole kernel to obtain their phase map. From these, $64 \times 64 \times 64$ susceptibility - phase image pairs where then randomly cropped and used for training, as in [19].

2.2.3. ASPEN and k-QSM

Approximated Susceptibility through Parcellated Encoder-decoder Networks (ASPEN), like DeepQSM described in Section 2.2.2, trains a 3D UNet on simulated phantom data. COSMOS maps are limited by the quality of preprocessing and are subject to numerical/optimization errors, simulated susceptibility maps do not have this problem. Unlike [21], the simulated data is not simple geometric shapes, they are instead made to match the frequency distribution of real life brain susceptibility images [22]. As usual, phase maps are obtained from simulated susceptibility maps via the QSM forward equation. The loss function in this case is the L_1 between the model output and the ground truth susceptibility map. During inference, the output from the trained model is then substituted, in k-space, into TKD region defined by a threshold of 0.2 Although the method is concerned with k-space substitution, it learns in image space. The loss used here can be thought of as a surrogate loss function [23, pp. 274–274], as it does not directly optimize for the susceptibility in the k-space region that is used in the final susbstitution. This has the downside that the trained model has to learn the entire inversion in other to get only a region in k-space correctly.

For k-QSM, the authors trained a deep neural network to obtain susceptibility maps in k-space [24]. The input in this case was not a phase map but a TKD map. The TKD map is initially obtained from the phase map, the FFT of this TKD map is then concatenated with a dipole kernel. This concatenated TKD and dipole kernel serve as the input to the neural network. The FFT of the TKD is complex, so to feed this into a neural network, the real and imaginary parts are first obtained then concatenated. This would invevitably limit the amount of data that can be fed at a time to the network, compared to feeding a map in image space directly, as the matrix size obtained from the concatenation is double the matrix size in image space. The loss function in this case is defined as

$$loss = \omega_1 \frac{1}{N} \sum_{i=1}^{N} \left|\left| \hat{\chi}_i - \chi_{label_i} \right|\right|_2^2 + \omega_2 \frac{1}{N} \sum_{i=1}^{N} \left| \hat{\chi}_i|_{cond} - \chi_{TKD_i}|_{cond} \right| \tag{2.12}$$

where ω_1 and ω_2 are weights on the losses, $\hat{\chi}_i$ is the model output, χ_{label_i} is a COSMOS label and χ_{TKD_i} is a map generated via TKD. The second loss basically says that the output from the model should be the same as the input in regions where a condition, *cond*, is satisfied. The condition here is that the dipole kernel, D, is bigger than some threshold value. The first part of the loss says the output should be close to the COSMOS label, essentially filling the parts of the output not covered by the second loss.

2.2.4. FINE and MoDIP

Both Fidelity Imposed Network Edit (FINE) [25] and Model-based Deep Image Prior (MoDIP) [26] are examples of deep learning based QSM methods that are also iterative. In FINE, the authors first pretrained a UNet model for the QSM inversion. For the initial pretraining, they used the same UNet architecture as in [19] and the same loss function as defined in that paper. The initial model was trained with COSMOS labels and on patches as in the other methods previously described. The aim of this initial training was to implicitly obtain QSM priors for regularisation. After this initial training, during testing, the pretrained model is finetuned on the fidelity loss to ensure

that the produced result is consistent with the dipole equation. The fidelity loss is defined as

$$||W(FT^{-1}D \cdot FT(\Phi(f;\Theta)) - f)||_2^2 \tag{2.13}$$

where W is a weight term and it is the inverse of the square root of the noise covariance matrix [25], $\Phi()$ is the deep learning model, Θ is its weights to be optimized and f is the input phase. According to the authors, finetuing the network in this manner can provide more accurate susceptibility maps than using explicit regularisation.

MoDIP is based on the deep image prior (DIP) paper [27]. Unlike FINE, both DIP and MoDIP do not attempt to learn image prior in a supervised manner. Instead, the prior is implicitly encoded into the structure of the deep neural network [26, 27]. The model is iterative and does not require pretraining which means no need to gather hard-to-obtain data. Although no pretraining is required, the fact that it is iterative means it takes longer to produce results than non-iterative DL methods. At inference time, the model uses a small randomly initialised UNet to produce an initial susceptibility map. This map is then finetuned to be consistent with the forward dipole equation via conventional optimization, hence producing the final QSM image.

3. PROPOSED DEEP LEARNING SOLUTIONS

In order to test the proposed deep learning methods, phantom data were generated as in [21]. Experiments with those phantom data are described in this chapter.

3.1. Data Generation

Using the qsm_forward python library [28], 100000 susceptibility phantoms were first generated. Each phantom was generated with a different seed using a Gaussian with mean 0 and standard deviation of 1. The phantoms are filled with spheres and rectangular prisms. The number of spheres and rectangular prisms is randomly chosen to be between 80 and 120 for each phantom, and these shapes cover roughly 75 percent of the entire area of the phantom. The matrix size of each phantom is chosen to be $64 \times 64 \times 64$, just as used in previous Deep Learning QSM experiments. Given this susceptibility distribution phantom, the field perturbation was calculated as $\Delta B(r) = FT^{-1}(DFT(\chi))$ where the symbols are as previously defined. The susceptibility and field perturbation pairs are then used to train and validate our deep neural network. The methods described here are implemented in Python 3.10 and PyTorch 2.4.

3.2. Deep Neural Network: UNet

As in [21] and [19], a UNet architecture was chosen for these experiments. The original architecture in [21] consists of a contracting path using 3×3×3 3D convolutions and ReLU layer followed by a $2 \times 2 \times 2$ max pooling layer. The pooling layer takes a 2×2×2 input volume and returns the maximum of this volume, it halves the dimensions of its input as a result.

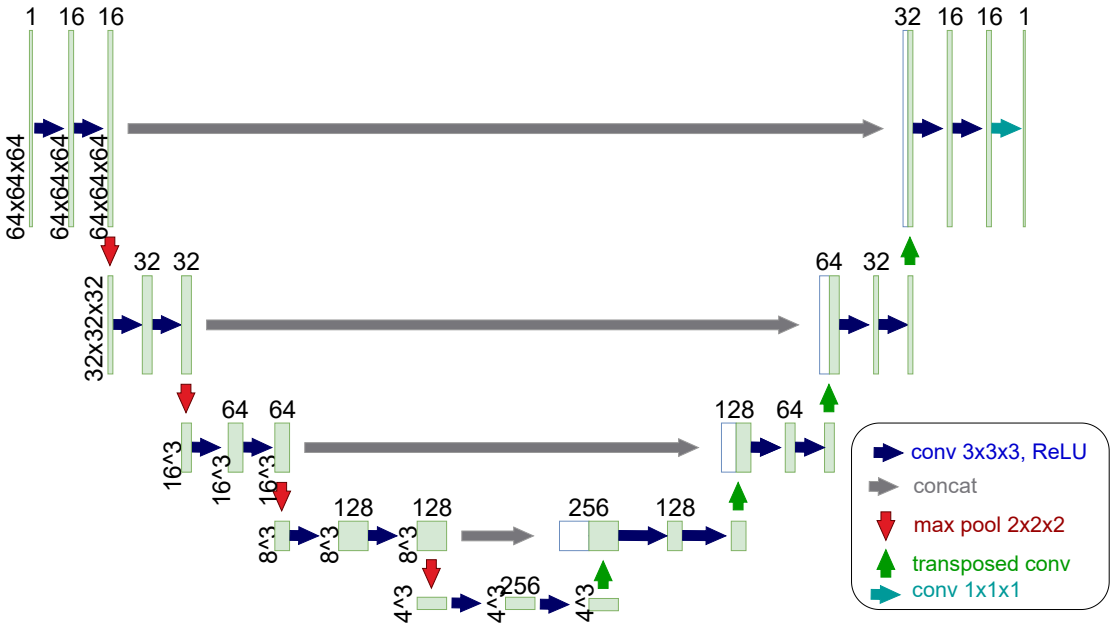


Figure 3.1. UNet architecture using transposed convolution for upsampling.

The other path of the UNet architecture is an expanding path which uses a transposed convolution layer with a stride of $2 \times 2 \times 2$ to double the size of its own input. Each transposed convolution layer is also followed by a convolution and ReLU layer like in the contracting path. For all the architecture variants used in these experiments, the loss function used is an L_1 loss between the output of the neural network and the ground-truth susceptibility distribution. The architecture is shown in Figure 3.1. The simulated phase serves as the input to the model. Adam optimizer was used with a learning rate of 4e-4, and in order to speed up training, we additionally used PyTorch's mixed precision training, which enables calculation to be done with 16bits, thus increasing the amount of data that can be fit into the GPU RAM. Mixed precision training is further described in Appendix A.2. While the UNet architecture as described is able to reconstruct the susceptibility distribution from the phase image, it is often not perfect, and results sometimes have undesirable checkerboard patterns. In order to tackle these, several modifications to this architecture are proposed, and these modifications are described in the following sections.

3.2.1. Transposed Convolution vs Nearest-Neighbor Upsampling

Transposed convolution is the go-to method for upsampling in the UNet in QSM literature, usually with a stride of two. With this configuration, it can be used to upsample a feature map to twice its input size. An illustration of a strided transposed convolution is shown in [29, pp. 25–27]. Transposed convolution initially upsamples the input map by filling in between samples in the input map with zeros, much like traditional upsampling and this upsampling is then followed by a convolutional filter to allow learning [29]. This is similar to traditional upsampling in which filling is done by inserting zeros between samples in the input followed by filtering with a known filter depending on the upsampling method. In transposed convolution, this filter has been replaced with a convolution layer, which means our network has to do extra work learning the correct upsampling method in addition to learning representations for the task in which we are interested. We hypothesize that optimizing the model can be made easier by replacing this transposed convolution with a traditional upsampling method such as nearest-neighbor upsampling then followed by a convolution layer for feature learning. To this end, the transposed convolution layer was replaced with a nearest-neighbor upsampling function followed by a convolutional layer.

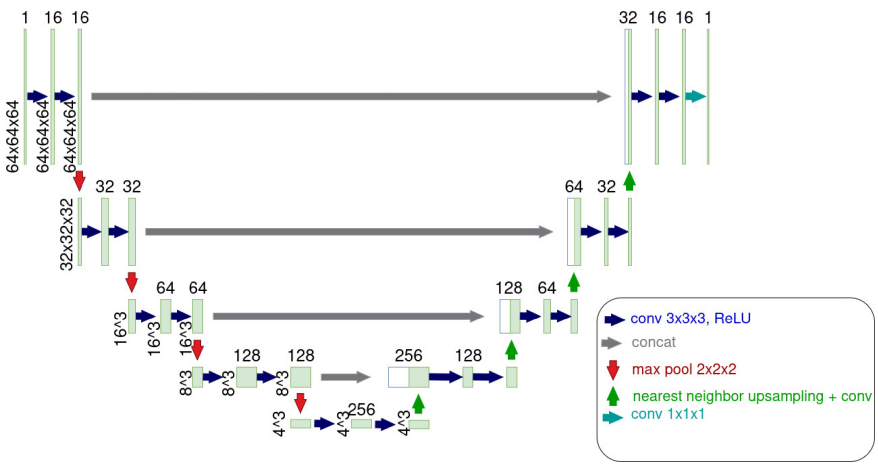
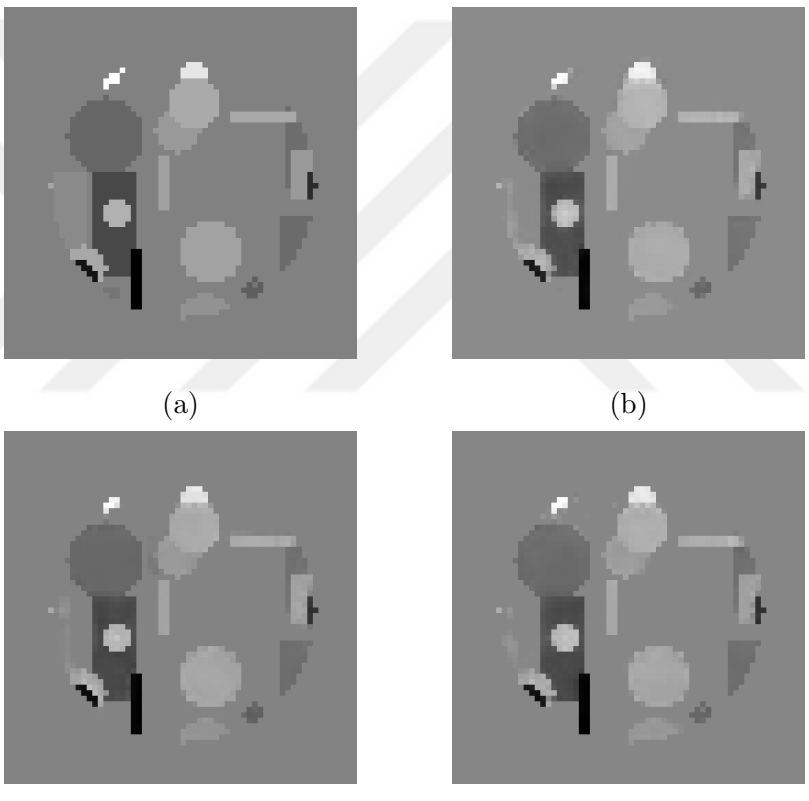


Figure 3.2. UNet architecture using nearest neighbor and transposed convolution for upsampling.

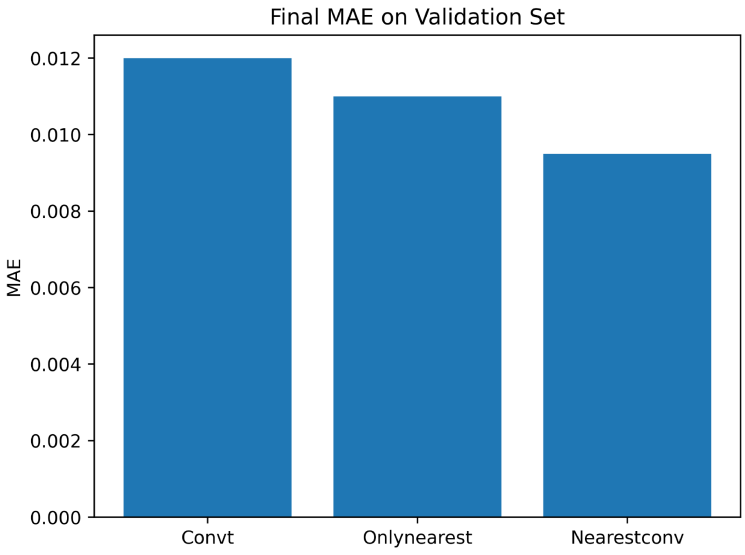
The modified architecture is shown in Figure 3.2. In another experiment, the transposed convolution layer was simply replaced with a nearest-neighbor upsampling layer; this should work because the UNet architecture uses an additional double convolution layer after all types of upsampling anyway which means that we still essentially have an upsampling followed by convolution layer. Experiments were performed by training our model on 40000 training examples and validating on another 5000 examples. Results are reported on the validation set.



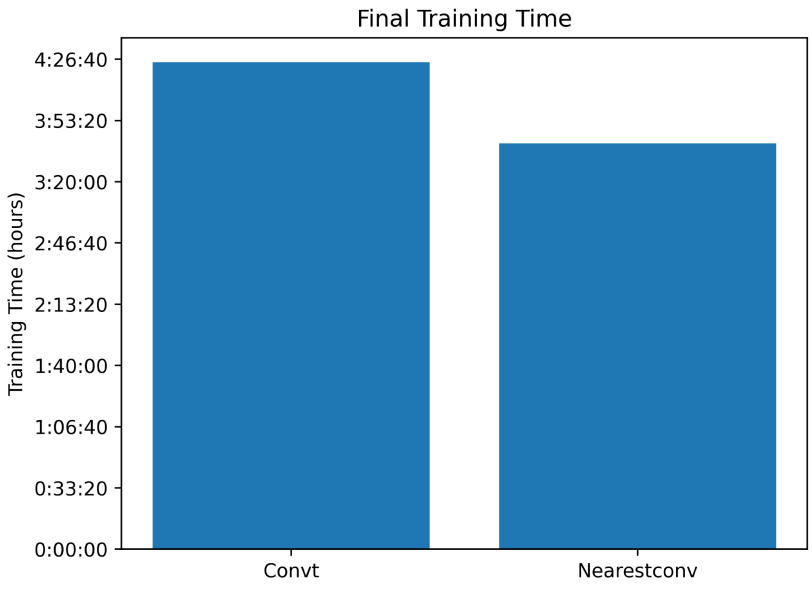
(a) (b)

(c) (d)

Figure 3.3. Susceptibility Image Results. (a) Shows the ground truth susceptibility map, (b) shows the result obtained when transposed covnvolution is used for upsampling, (c) shows the result in the case where the nearest upsampling followed by convolution is used while (d) shows the case where only nearest upsampling is used.



(a)



(b)

Figure 3.4. Final MAE and training times. (a) Shows the final validation MAE. Nearest upsampling and convolution shows almost 21% smaller MAE compared with transposed convolution. (b) Shows the total training times for nearest upsampling followed by convolution and transposed convolution.

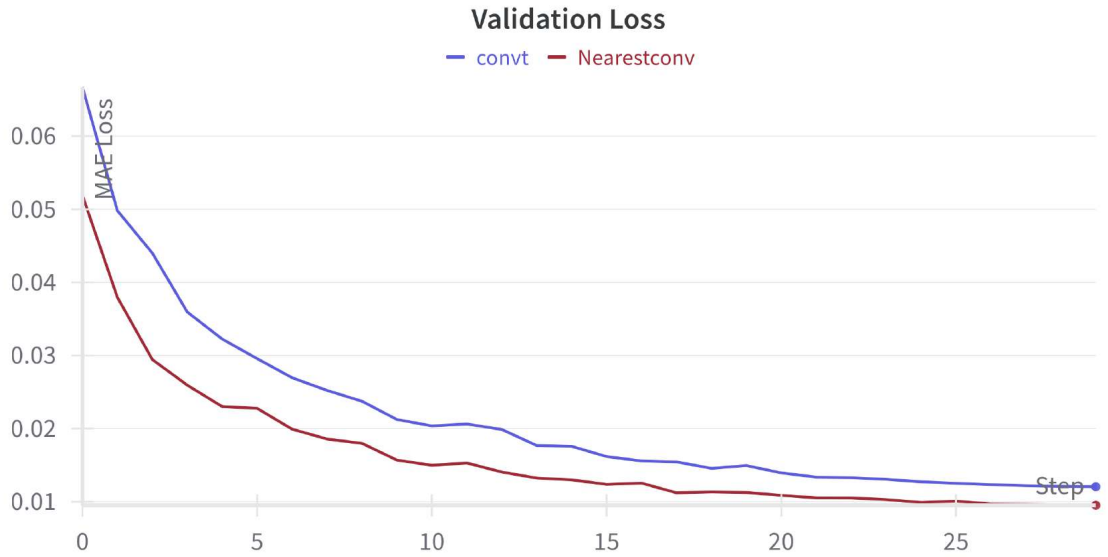


Figure 3.5. Validation curves for transposed convolution (purple) and nearest upsampling followed by a convolution (red).

In our experiments, we observed a nearly 21% decrease in the mean absolute error in the validation set when the transposed coonvolution is replaced by an upsampling layer followed by a convolution, showing the effectiveness of this replacement. Additionally, using only the nearest-neighbor upsampling, without following it with a convolution, also shows a 8% decrease in error compared to the transposed convolution method, although they have very similar-appearing images in the end. In terms of runtime on a machine with an Intel Xeon Silver CPU, 128GB RAM and an NVIDIA RTX A4500 with 20GB vRAM running Ubuntu 22.04 OS, the transposed convolution method ran in 4 hours and 25 minutes while upsampling with convolution ran in 3 hours and 41 minutes, showing the speed improvement achieved by this simple swap. The obtained susceptibility maps are shown in Figure 3.3, where largely similar maps can be seen. Figure 3.3a shows the ground truth susceptibility map, 3.3b shows the result obtained when transposed covnvolution is used for upsampling, 3.3c shows the result in the case where the nearest upsampling followed by convolution is used while 3.3d shows the case where only nearest upsampling is used. MAE and runtime results are shown

in Figure 3.4. A graph comparing the learning curves for the three methods is also shown in Figure 3.5. The best performing curve is the one which uses nearest-neighbor upsampling followed a convolution layer. It should be noted that this type of swap has already been done in image generation, where it is used to reduce checkerboard artifacts caused by transposed convolutions [30], and that study motivated our use of the simple nearest-neighbor upsampling method. We have been able to show, through our experiments, that preferring a nearest neighbor upsampler followed by convolution can speed up training convergence in QSM applications. Although upsampling was indicated in figures in [31], it was not indicated which type of upsampling method was used or why, additionally in [26] a convolution layer followed by nearest-neighbor upsampling was used for upsampling, but no justification was given for this decision.

3.3. Proposed QSM Reconstruction Method

It is impossible to obtain susceptibility distributions by a simple inversion in the Fourier domain due to the zeros in the values of the dipole kernel, D. While some deep learning methods have already been proposed which perform the dipole inversion wholly by converting an input phase image into a susceptibility distribution, we propose here to use a deep neural network to obtain the correct susceptibility distribution only in and around the region where D is zero.

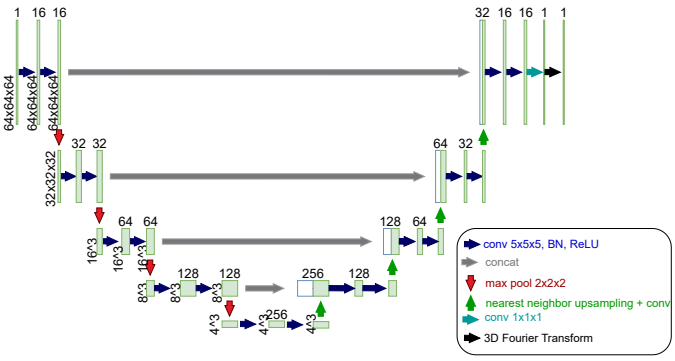


Figure 3.6. UNet architecture with a Fourier transform module at the end.

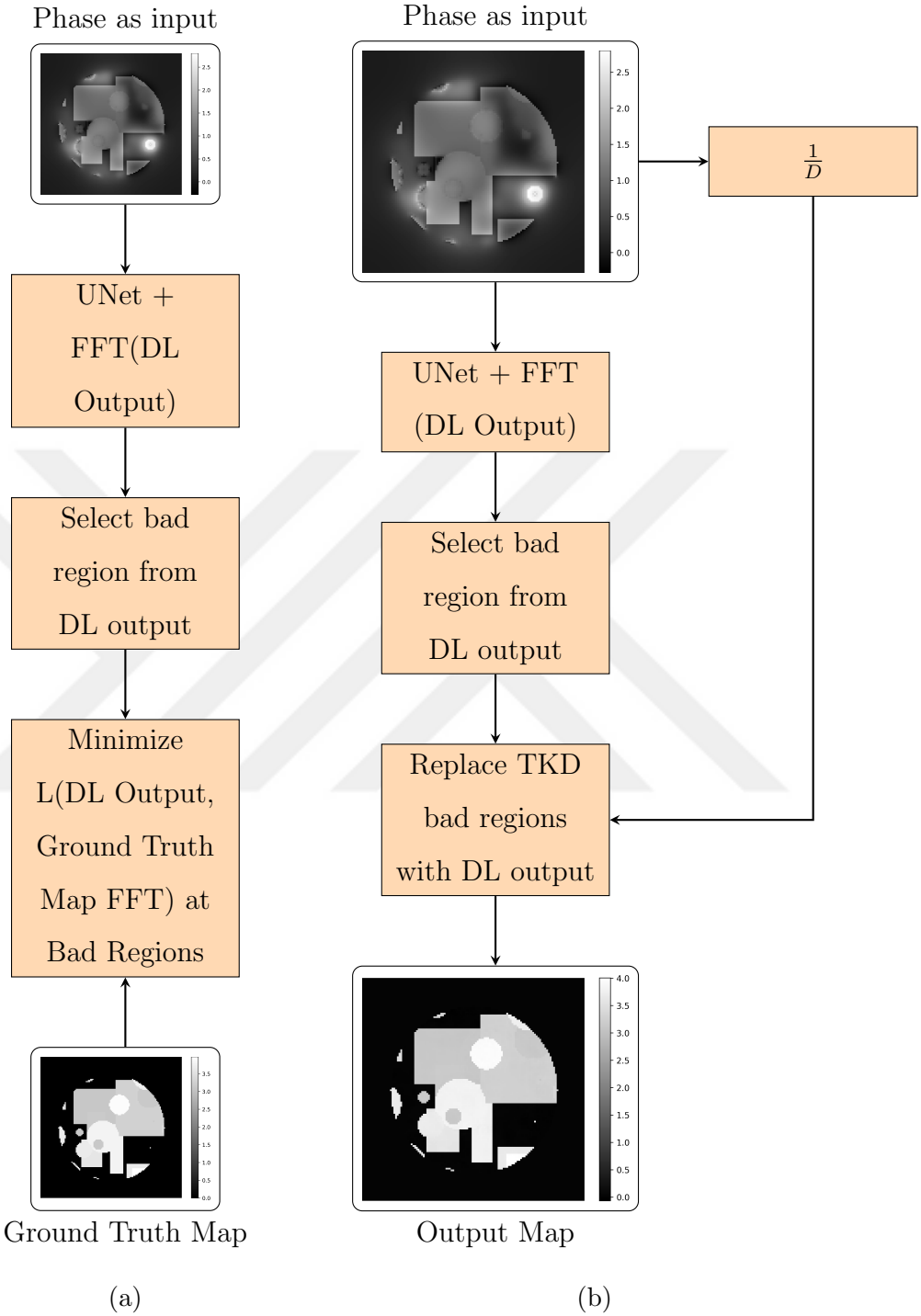


Figure 3.7. Algorithm flowcharts for (a) training and (b) testing.

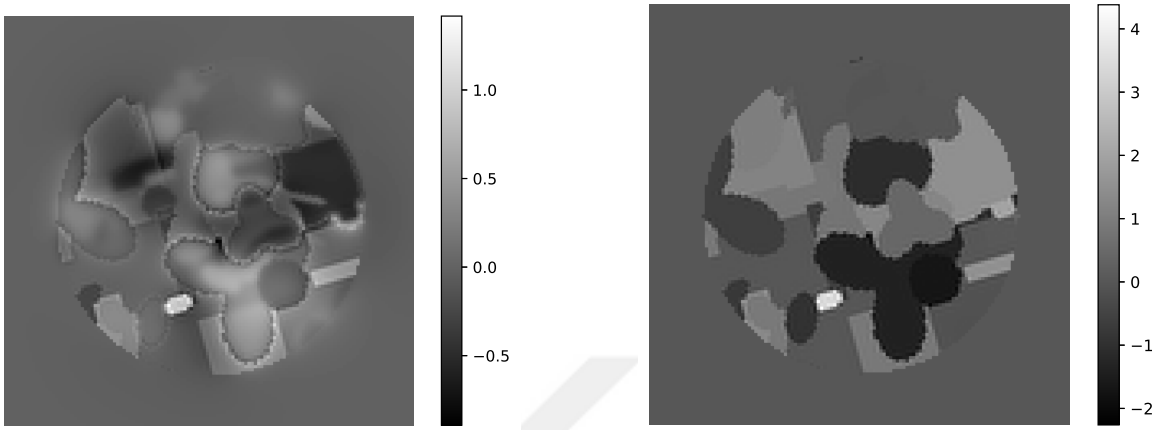
Unlike those methods which either work exclusively in the space domain or exclusively in the Fourier domain, our approach aims to bridge the two domains. We do this by defining a deep learning architecture which takes as input an unwrapped

and backgorund-removed phase image and returns as output the missing susceptibility in the Fourier domain. The architecture is a UNet architecture with the transposed convolution layers replaced with a nearest neighbor upsampling followed by a convolutional layer as already described in Section 3.2.1, we modify it by adding a Fourier transform layer at the end of this architecture, which enables it to return output in the Fourier domain. In addition to these, and different from the previous architecture, our kernel size is chosen to be 5 instead of 3 for all 3D convolutions and we used batch normalization in training as in [19]. In the regions where the dipole kernel, D, is stable, we obtain the Fourier space susceptibility distribution by simply dividing the phase image by the corresponding value of D element-wise in the Fourier domain. The final susceptibility distribution is then the combination of our deep learning model output and the result of this division. In terms of the models described in Chapter 2, this means we use direct division in k-space to satisfy the data fidelity term in Equation 2.2 and then we obtain the prior(that is the regularisation term) from the deep learning model. Unlike in [22] where the authors learn the whole susceptibility and then extract the correct susceptibility from the singularity regions, we directly use the deep learning model to learn the prior. We show that learning the entire map to obtain the correction is not necessary in our results. Our algorithm is shown pictorially in training and testing mode in Figure 3.7. Our proposed architecture is shown in Figure 3.6.

3.3.1. Training Data

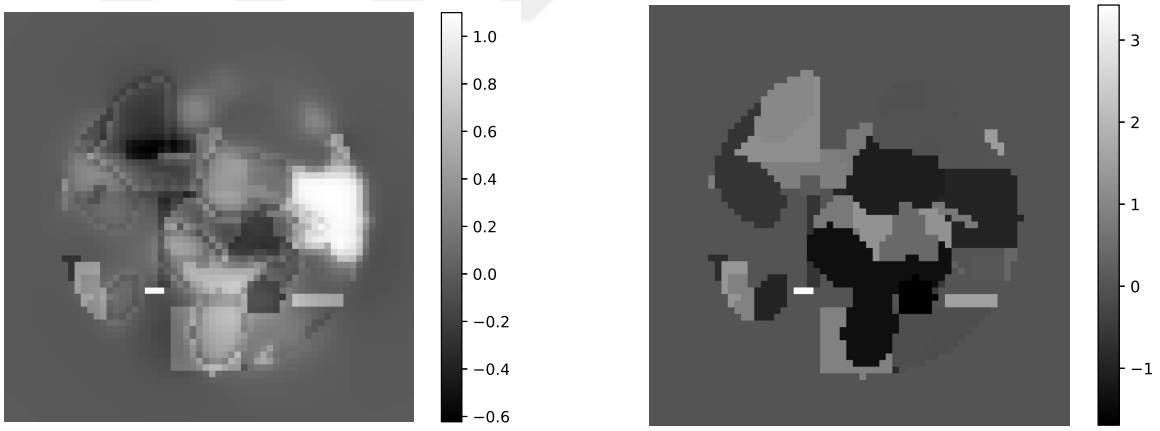
The training data was generated as in Section 3.1. In addition to the simple rectangular prism and spherical shapes used in that section, we additionally introduced ellipses and rose petal-shaped objects into the volumes. These shapes were introduced in order to mimic the shape of some of the structures found in real brain images, the shapes were introduced to compare the results on simulated brain image from the QSM challenge in [13]. Additionally, data augmentation by rotation was done for the $128 \times 128 \times 128$-sized data generated in this way. The rotation was between -30° and 30° about the z-axis. Performing rotations on the $64 \times 64 \times 64$-sized data gave rise to artefacts due to the interpolation performed, therefore no rotation was done for the

$64\times64\times$ 64-sized data.



(a) (b)

Figure 3.8. (a) Phase image and (b) susceptibility image for complicated shapes and $128\times128\times$ 128 volume.



(a) (b)

Figure 3.9. (a) Phase image and (b) susceptibility image for complicated shapes and $64\times64\times$ 64 volume.

Example images are shown in Figure 3.8. Here, training is done on 20000 examples and validated on 5000 examples. The data volumes are generated by padding to twice the size in each direction as described in [13]. We observed that the 64 volumes have low resolution for the generated susceptibility distributions, which is seen in how

jagged and nonsmooth those images appear (see Figure 3.9), and have nonzero phase values outside of the 64×64×64 volume for the generated phase image. To reduce this effect, we additionally generated data of size $128 \times 128 \times 128$. The star shapes can be seen to be much smoother in this case as seen in Figure 3.8. We compare results from training on these two volume sizes on a brain phantom dataset provided by [13].

3.3.2. Deep Learning Training

Experiments were performed concurrently using machines with Intel Xeon Silver CPU, 128GB RAM and an NVIDIA RTX A4500 GPU, 20GB vRAM and another machine with Intel Xeon Gold CPU, 128GB RAM and two 16GB NVIDIA RTX A4000 GPUs available at the Systems Science and Mathematics Lab (BUEE-SSML) in the department of Electrical Engineering at Boğaziçi University. All systems were running Ubuntu 22.04 OS. The loss function is defined in the Fourier domain as

$$L = ||\chi(M) - \hat{\chi}(M)||_1 \tag{3.1}$$

where χ is the ground truth susceptibility in Fourier domain, $\hat{\chi}$ is the output of our deep learning architecture and M is a mask obtained from D which is used to select the region in which we want our predictions. It is defined as follows

$$M = \begin{cases} 0 & \text{if } |D| > thresh \\ 1 & \text{if } |D| \leq thresh \end{cases} \tag{3.2}$$

$thresh$ is chosen to be 0.0625, following [32], for all experiments.

Adam optimizer was used to train all our models with a learning rate of 1e-4. The learning rate is varied over the training steps using the 1CycleLR strategy [33], described in Appendix A.3.2.2, which has been found to be useful for deep learning model optimization. The learning rate was increased linearly for the first 50 training steps and reduced for the rest in all experiments. For training on the 64 volumes a

batchsize of 16 was used, while a batch size of 4 was used for training on 128 volumes. The smaller batch size was chosen for the 128 volume because this was the optimal size that could be fit into the availaible GPU RAM. To make the effective training batch size for the 128 volumes same as that of the 64 volumes (that is 16), gradient accumulation (described in Appendix A.3.1.3) with accumulation steps equal to 4 was additionally used for training. Although we tried to train with 16bits mixed precision training in this case as well, this proved impossible. Training with mixed precision failed as performing Fourier transforms with 16 bits led to the code returning NaN values. To prevent model overfitting, weight decay was used on the model weights. The weight decay value was set to 1e-4 as well. It was initailly observed that training with weight decays was unstable, it was found that the reason for this was the application of the weight decay to the batch normalization layers. To deal with this, the weight decay value for batch normalization layers was set to zero. Both weight decay and batch normalization are described in the appendix. This means that weight decay was applied to all the trainable layers except for the batch normalization layers. For each epoch, weight averaging was also used. This was achieved by taking the exponential moving average of the model for each step over an epoch. The number of epochs was set to 30 for all training run.

3.4. Results

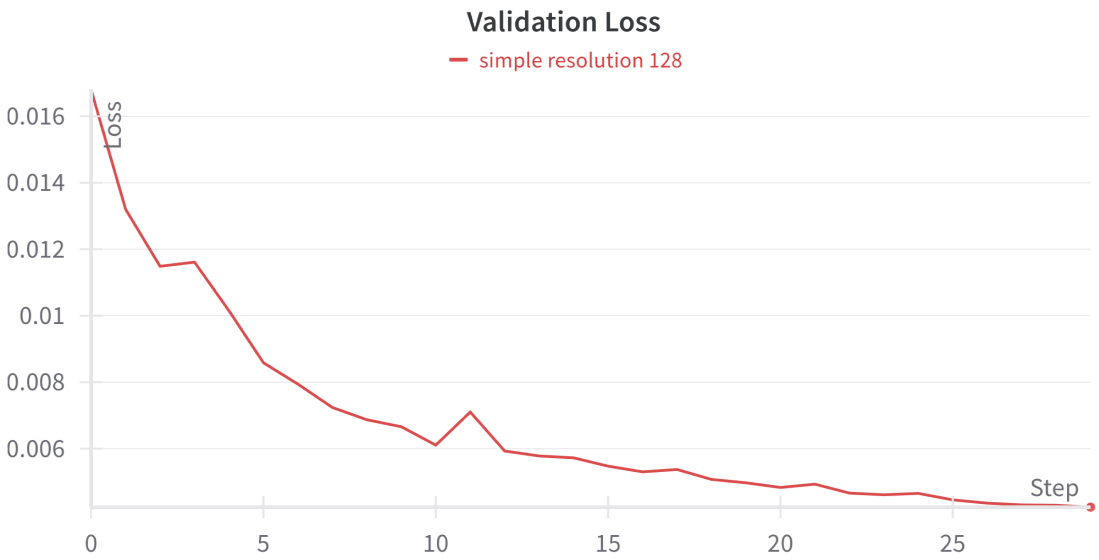
Training results for different datasets used and on the phantom brain dataset are described in this section. We show resulting QSM maps and training curves for different training regimes. The training curves were based on the loss function defined in Equation 3.1. The metric used is the normalized root mean square error (NRMSE) in the ROI, which is the same as that used in the QSM challenge in [13] and is defined as

$$NRMSE = 100\frac{\sqrt{\sum_{ROI}(\chi_{recon} - \chi_{true})^2}}{\sqrt{\sum_{ROI}\chi_{true}^2}}. \tag{3.3}$$

3.4.1. Results on Simulated Phantom

In this section, results are presented on simulated phantom data similar to the training data.

(a)



(b)

Figure 3.10. (a) Training curve and (b) validation curve for model from Section 3.4.1.

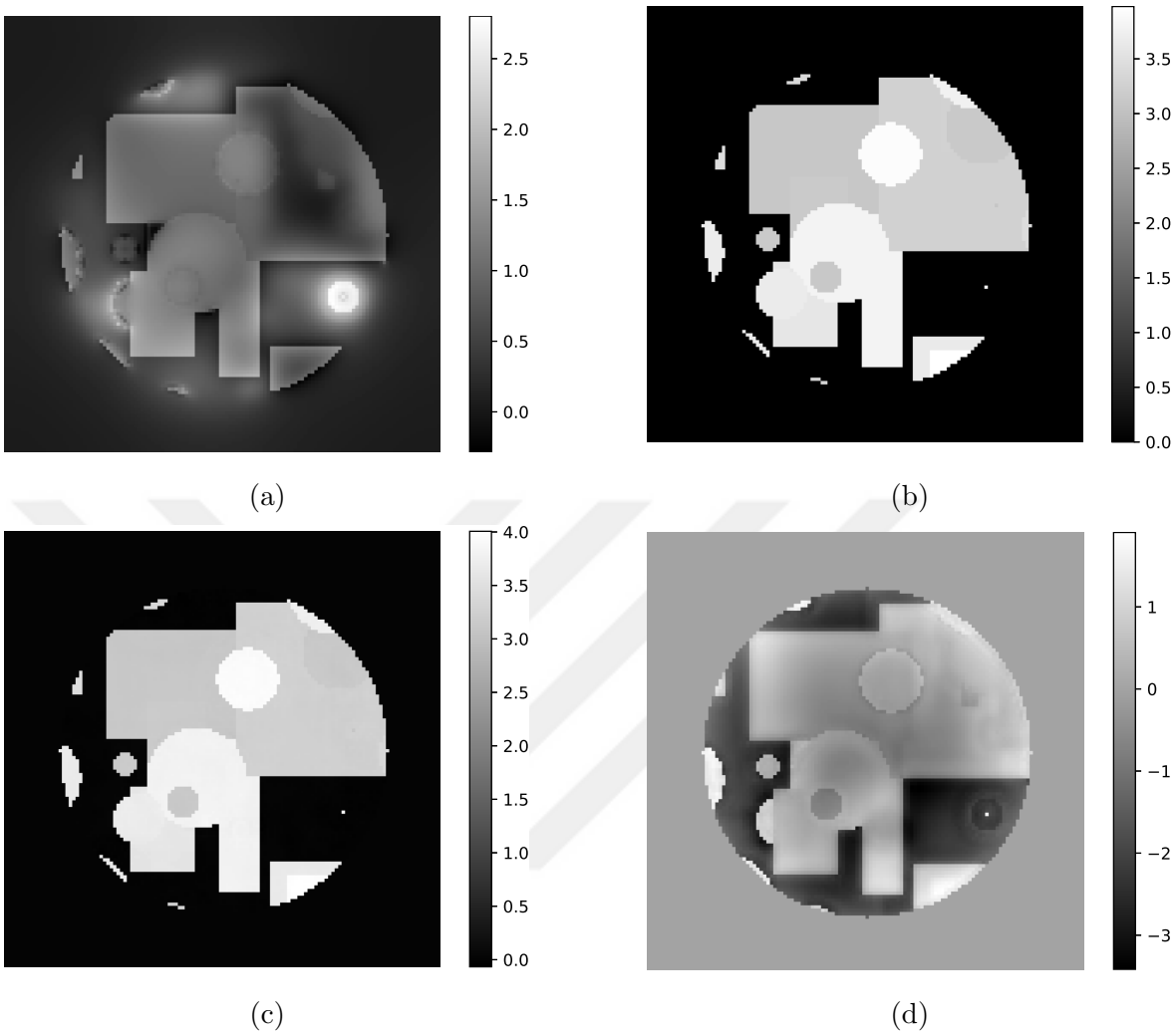


Figure 3.11. Results on simulated phantom for model from Section 3.4.1. (a) Shows the input phase image; (b) shows the ground truth susceptibility distribution; (c) shows the predicted output based on our approach with NRMSE = 0.56% while (d) shows the output reconstructed from only UNet output; NRMSE = 98.18%.

Training was done on data with only simple rectangular and spherical shapes. The learning curve for this training run is shown in Figure 3.10, example phase image input and the resulting model prediction are shown in Figure 3.11 and the ground truth is also shown for comparison. Test results are presented on simulated susceptibility that is uniformly distributed bewteen 3 and 4. This is to ensure that we are not testing on data from the same distribution as the training or validation data. Additionally, a susceptibility map reconstructed from just the output of the deep learning model is

shown in Figure 3.11. We can see that its NRMSE value is larger than the value of our final reconstructed map, showing that the model's output does not have to have a good NRMSE value for us to obtain a good correction in the regions containing the singularities (i.e for us to obtain a good prior in those regions).

3.4.2. Results on Brain Phantom

The brain phantom was obtained from the QSM reconstruction challenge 2.0 [13]. There were two susceptibility maps, one having a calicification and the other without calcification. The maps were $0.65 \times 0.65 \times 0.65$ mm resolution. Using the maps at this resolution, phase images were obtained using the QSM dipole forward equation for the two maps. After this, the susceptibility maps and their corresponding phase maps were downsampled via k-space cropping from 0.65mm to 1.06mm. In the challenge, they were downsampled to 1mm resolution. 1.06mm resolution was chosen here as it gives maps with matrix sizes divisible by 16, which is required by the neural network. After downsampling, noise at 100 peak SNR was added to the phase map without calcification and noise at 1000 peak SNR was added to the phase image with calcification, just as was done in the challenge. The given brain mask was equally downsampled to be the same size as the susceptibility and phase maps.

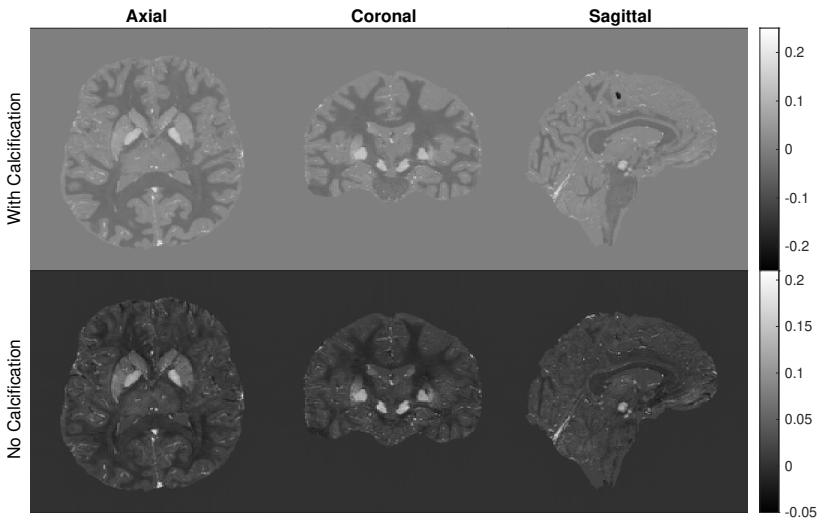


Figure 3.12. Brain phantom susceptibility with and without calcification.

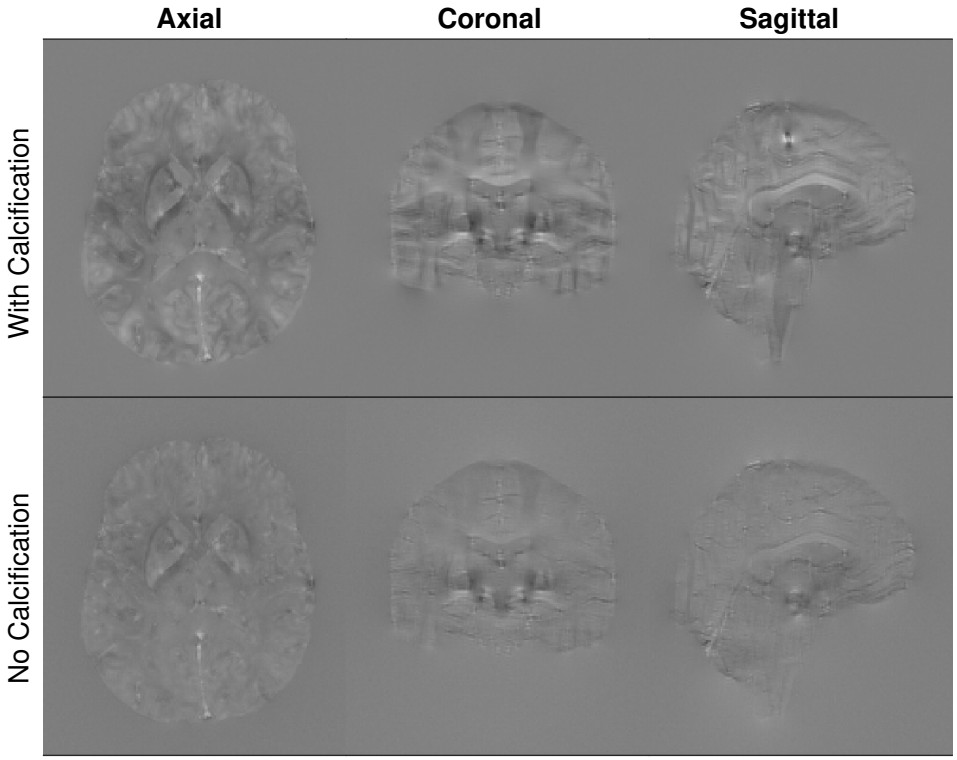


Figure 3.13. Corresponding phase maps for the phantoms in Figure 3.12.

The processed maps are shown in Figure 3.12 and Figure 3.13. With the newly generated $64\times64\times64$ and $128\times128\times128$ maps (described in Section 3.3.1), we train deep learning models from scratch and compare the results on this brain phantom. The learning curves for these models are shown in Figures 3.14 and 3.15.

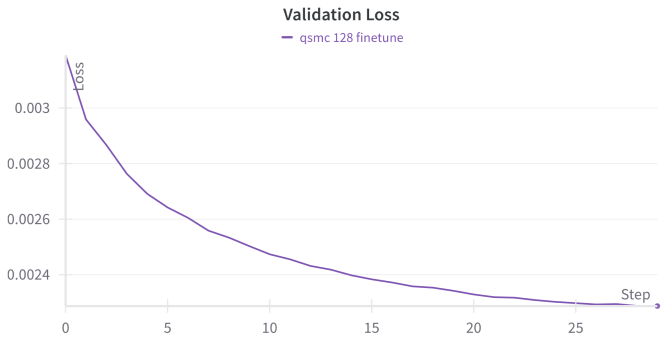
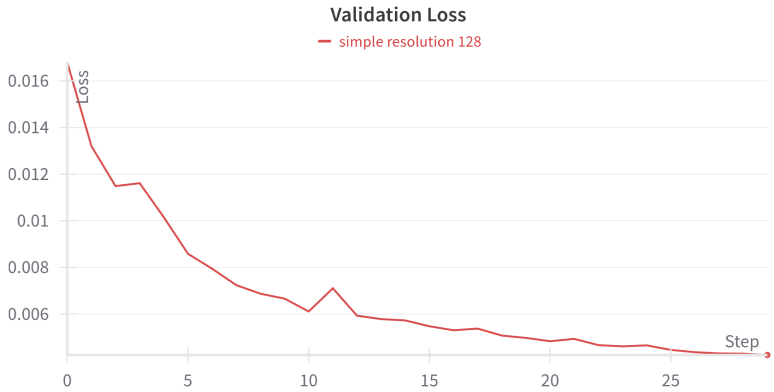
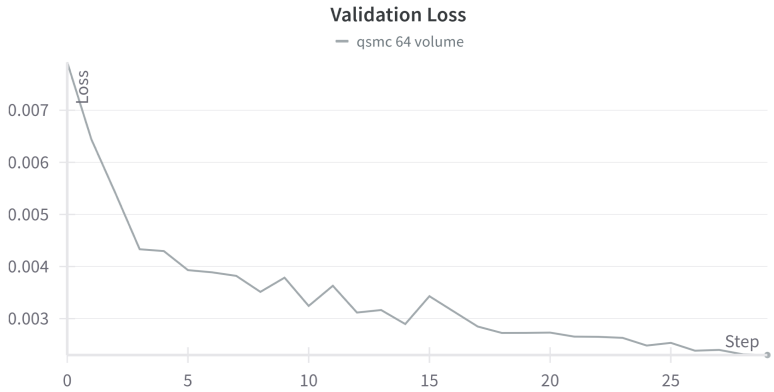


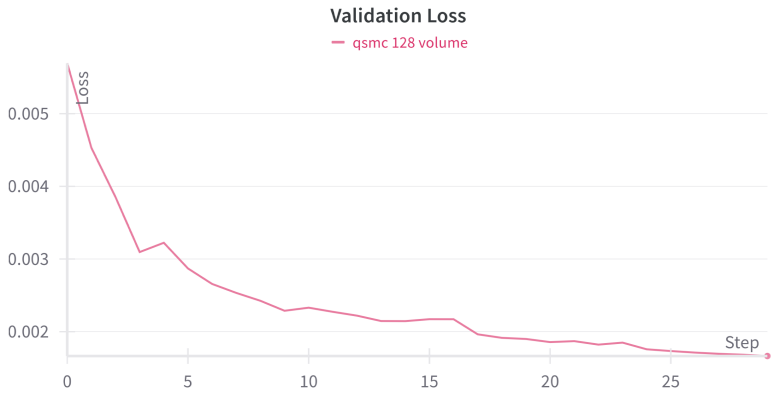
Figure 3.14. Curve for finetuned model.



(a)



(b)



(c)

Figure 3.15. Validation curves for all models compared in this section with (a) showing curve for model on simple shapes, (b) showing for model on complicated shapes and size $64 \times 64 \times 64$ and (c) for model on complicated shapes and size $128 \times 128 \times 128$.

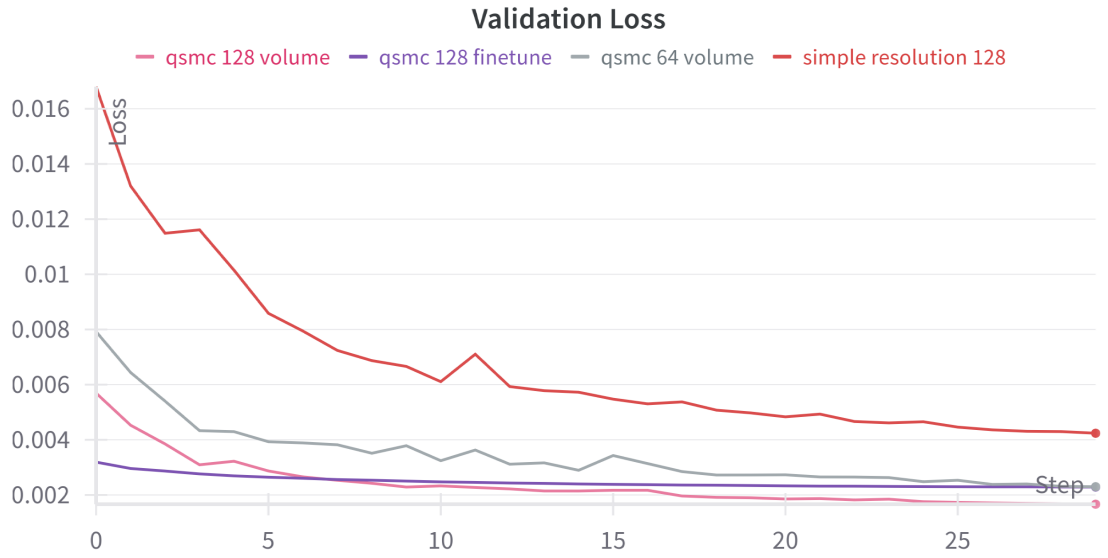


Figure 3.16. Validation curves from Figure 3.15 shown together for comparison.

The model trained on the bigger-sized data has a better learning curve (see Figure 3.16) but has a similar performance on the brain phantom (in terms of NRMSE values). This suggests that training at one (even big) resolution may not be very advantageous in this case in terms of generalization, so one may stick to smaller size data and still obtain similar results as may be obtained by using larger-dimensioned maps. Both of these models perform better on the brain phantom than the model trained on the simple geometric shapes, suggesting that adding more complicated shapes to the model may be useful. We also finetuned the model obtained from Section 3.4.1 using the data with complicated shapes to compare performance.The learning curve for this is also shown in Figure 3.14. Although this model trained faster, in the sense that it reached a small loss faster than other models, it did not seem to improve much on those other models in the long run, as can be seen in Figure 3.16 which shows all four learning curves compared here together. For this reason, it was not included for further analysis.

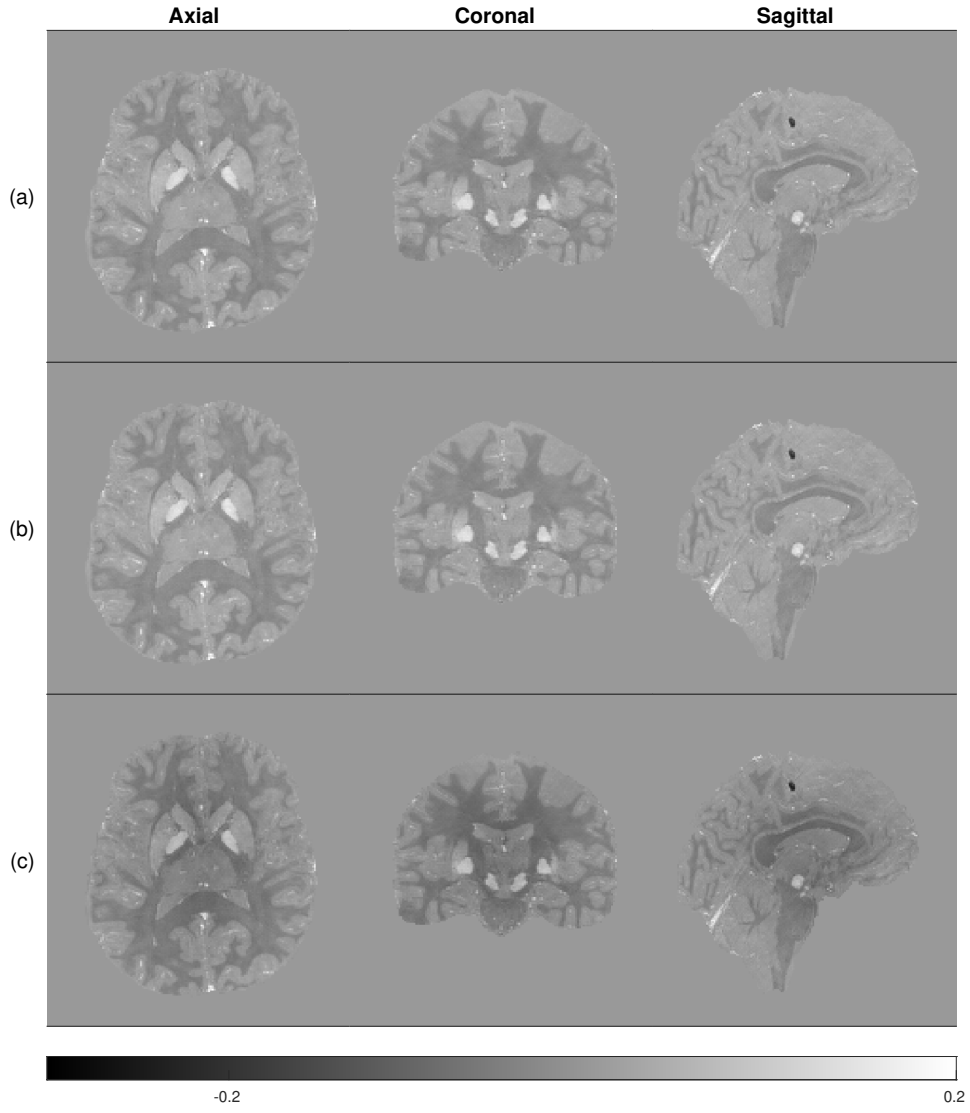


Figure 3.17. Predicted susceptibility maps with calcification for models trained on (a) $128 \times 128 \times 128$-sized and (b) $64 \times 64 \times 64$-sized data with complicated shapes. (c) Shows the case with non-complicated shapes.

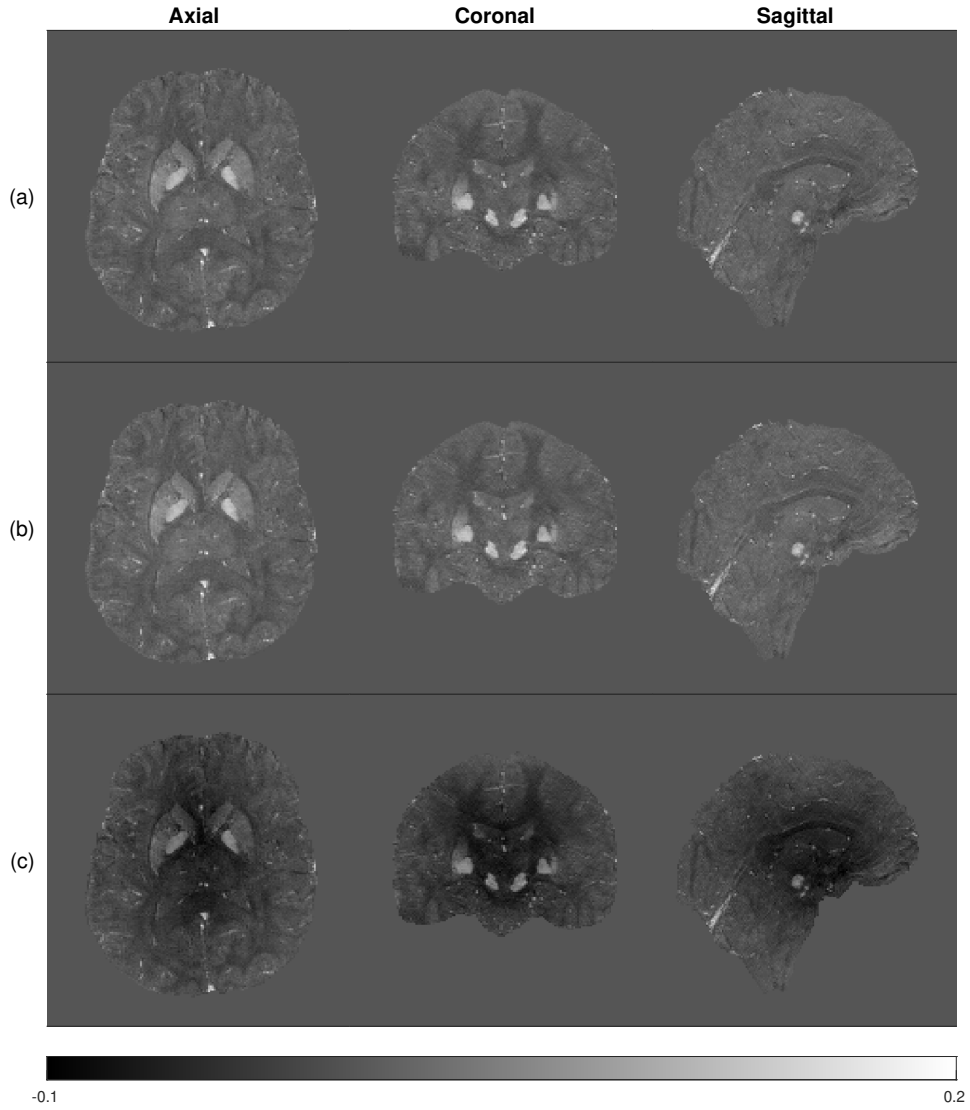


Figure 3.18. Predicted susceptibility maps in the case without calcification for models trained on (a) $128\times128\times128$-sized and (b) $64\times64\times64$-sized data with complicated shapes. (c) Shows the case with non-complicated shapes.

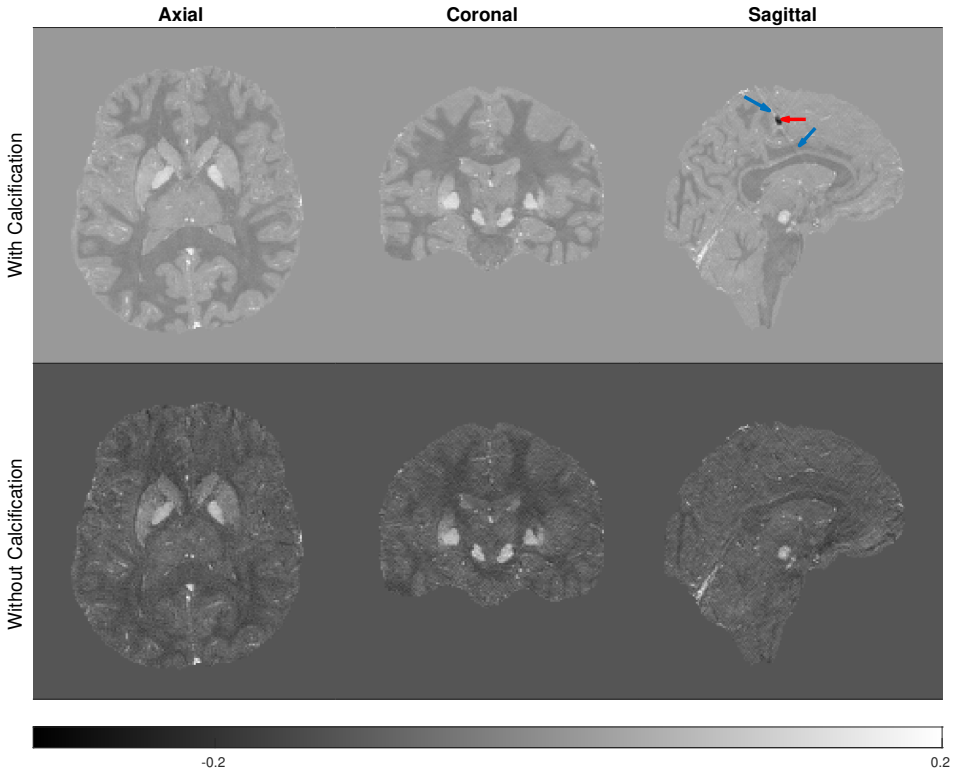


Figure 3.19. TKD susceptibility maps with and without calcification. Red arrow points at calcification and blue arrows at streaking artifacts.

Brain phantom suceptibility maps obtained from models trained on more complicated shapes, described here, and the model trained on the simple shapes, described in Section 3.4.1, are shown in Figures 3.17 – 3.19. For comparison, the susceptibility map was also obtained via TKD and the predicted map is shown as well. The NRMSE for all these models are shown in Table 3.1. One reason the NRMSE for the model with simple shape looks extremely off with respsect to the others may be how the data was generated. To generate the phase for the brain phantom and the data with complicated shape, the susceptibility maps were padded to twice their size and the filtering with the dipole kernel was then done in the Fourier domain to obtain the phase image. This was done to follow the method prescribed in the challenge paper. For the simple shapes, we just carried out the filtering without doing the padding, hence the irregularity. The obtained average NRMSE from the model trained on the $128\times128\times128$ data is shown in Table 3.2 where we see that it ranks among the top 10 models, in terms of NRMSE,

Table 3.1. Model NRMSE values.

Model Name	**Calcification NRMSE**	**No Calcification NRMSE**	**Average NRMSE**
Model with 64×64× 64 data	25.22	27.61	26.42
Model with 128×128× 128 data	23.36	30.38	26.87
TKD	19.91	37.07	28.49
Model without complicated shapes	115.69	180.44	148.06

from the QSM challenge. The acronyms in the table are the names of the different models submitted for the challenge. The NRMSE reported was the average over data with calcification and data without calcification. Further details about the results from the challenge can be found in [13].

Table 3.2. Top-scoring NRMSE—Stage 2 (Acronym and NRMSE only).

Acronym	**NRMSE**
This study	**26.87**
FANSI	28.9
wL1L2TV	29.2
L1-QSM	30.3
FINE	30.4
WH-nlQSM	30.7
L1-QSM	31.7
WH-QSM	32.3
WH-QSM	32.8
TFIPC	35.5
FANSI-TGV	36.5

For the TKD result, streaking artifacts can be seen in the calcification region in the reconstruction of the brain with calcification as indicated by the blue arrows in the diagram. These artifacts are not present in the reconstruction from our models in general which means this model does a good job of getting rid of QSM streaking artifacts in this case. Additionally, we see from the table that TKD performs signifcantly

worse with higher noise levels in the output whereas our models do much better, even though we did not explicitly train them with noisy inputs.

Compared to the FINE algorithm described in Section 2.2.4, the algorithm described in this work uses a large dataset to learn a prior which enables it to predict the correct QSM values in the Fourier domain in the dipole kernel's singularity regions. It does not use a posttraining iterative algorithm during inference which means inference is fast. For the brain phantom described in this section, inference on this matrix of size $192 \times 160 \times 192$ took about $16.6s \pm 468ms$ measured the system running the Intel Xeon Silver CPU already described. In addition to this, the FINE algorithm was one of the participating models in the QSM challenge referred to in this work. It is named FINE in Table 3.2. From this table, its average NRMSE over the two dataset from the challenge can be seen to be 30.4, much worse than that obtained in this work. Another recent work also described in this work is MoDIP. This is also an iterative model, which does not learn any prior at all to correct errors in the singularity but instead relies on prior capture by the nature of the architecture used. The authors ran their algorithm on brain phantom from the QSM challenge but only did so on the one without calcification. They additionally did not record the NRMSE they obtained from their experiments [26].

4. CONCLUSION AND FUTURE WORKS

In this work we have shown that replacing transposed convolution in UNet can speed up the reconstruction of susceptibility maps from phase maps. In addition to this, we have presented a method to learn a deep learning based prior for QSM reconstruction. Our QSM reconstruction from this prior then follows the standard method for QSM reconstruction namely: maps are generated to obey the QSM forward equation in stable regions in k-space and for the unstable regions reconstruction is done with the learned prior. This is inline with both traditional reconstruction methods and deep learning based methods. Unlike previous methods, we directly only learned the required prior from data as opposed to learning to reconstruct the entire map. Our proposed method would place in first position in the QSM reconstruction challenge 2.0.

Data used for training our models were entirely generated by simulation. While this may be advantageous in the sense that it is the only way to have a truly correct ground truth, it has the limitation that the phase obtained may not represent real world data. Real world phase data may be anisotropic, and in addition generally requires preprocessing steps such phase unwrapping and background removal which may lead to these data type not exactly obeying the QSM forward equation. This means that even if we have a model that learns to truly perform QSM deconvolution, we may still have incorrect result on real world data. Additionally, as we are learning from data, even though the model may be learning to perform a deconvolution, it may also be learning (or memorizing) the structure of the input data. For this reason it would be advantageous to train models on the true data distribution expected in real life scenarios.

The main problem with training deep learning models for QSM reconstruction is the lack of real ground truth data. Traditional training methods propose a prior to guide the reconstruction but they generally are time consuming and require modifying the regularisation parameter for different data. There is an abundance of phase data.

A future research direction could be to train a deep learning model purely on such phase data and with a loss function comprised of the usual data fidelity term and a term from one of the traditional QSM priors like the one in MEDI, for example. Doing this has several advantages. If the prior (or regularisation) term used is good enough, we would be able to train on a really large and diverse corpus of data. This means we can train on phase images obtained from healthy individuals and individuals with different neurodegenerative diseases. The advantage of this over traditional MEDI would be the timesaving provided (since the deep learning method will not be iterative during inference) and it would get rid of the need to be fiddling with regularisation parameters.

REFERENCES

1. Langkammer, C., L. Pirpamer, S. Seiler, A. Deistung, F. Schweser, S. Franthal, N. Homayoon, P. Katschnig-Winter, M. Koegl-Wallner, T. Pendl, E. M. Stoegerer, K. Wenzel, F. F. Fazekas, S. Ropele, J. R. Reichenbach, R. Schmidt and P. Schwingenschuh, "Quantitative Susceptibility Mapping in Parkinson's Disease", *PloS One*, Vol. 11, No. 9, pp. 1–13, 2016.

2. Eskreis-Winkler, S., Y. Zhang, J. Zhang, Z. Liu, A. Dimov, A. Gupta and Y. Wang, "The Clinical Utility of QSM: Disease Diagnosis, Medical Management, and Surgical Planning", *NMR in Biomedicine*, Vol. 30, No. 4, p. e3668, 2017, e3668 NBM-15-0296.R3.

3. Milovic, C., C. Tejos, J. Acosta-Cabronero, P. S. Özbay, F. Schwesser, J. P. Marques, P. Irarrazaval, B. Bilgic and C. Langkammer, "The 2016 QSM Challenge: Lessons Learned and Considerations for a Future Challenge Design", *Magnetic Resonance in Medicine*, Vol. 84, No. 3, pp. 1624–1637, 2020.

4. Nishimura, D. G., *Principles of Magnetic Resonance Imaging*, Stanford University, Stanford, CA, 1996.

5. Haacke, E., S. Mittal, Z. Wu, J. Neelavalli and Y.-C. Cheng, "Susceptibility-Weighted Imaging: Technical Aspects and Clinical Applications, Part 1", *American Journal of Neuroradiology*, Vol. 30, No. 1, pp. 19–30, 2009.

6. Wang, Y. and T. Liu, "Quantitative Susceptibility Mapping (QSM): Decoding MRI Data for a Tissue Magnetic Biomarker", *Magnetic Resonance in Medicine*, Vol. 73, No. 1, pp. 82–101, 2015.

7. Schofield, M. A. and Y. Zhu, "Fast Phase Unwrapping Algorithm for Interferometric Applications", *Opt. Lett.*, Vol. 28, No. 14, pp. 1194–1196, 2003.

8. Wharton, S., A. Schäfer and R. Bowtell, "Susceptibility Mapping in the Human Brain Using Threshold-Based K-Space Division", *Magnetic Resonance in Medicine*, Vol. 63, No. 5, pp. 1292–1304, 2010.

9. Tian, L., *Quantitative Susceptibility Mapping Using Magnetic Resonance Imaging*, Master's Thesis, Cornell University, 2011.

10. Liu, T., P. Spincemaille, L. de Rochefort, B. Kressler and Y. Wang, "Calculation of Susceptibility Through Multiple Orientation Sampling (COSMOS): A Method for Conditioning the Inverse Problem From Measured Magnetic Field Map to Susceptibility Source Image in MRI", *Magnetic Resonance in Medicine*, Vol. 61, No. 1, pp. 196–204, 2009.

11. Jung, W., S. Bollmann and J. Lee, "Overview of Quantitative Susceptibility Mapping Using Deep Learning: Current Status, Challenges and Opportunities", *NMR in Biomedicine*, Vol. 35, No. 4, p. e4292, 2022, e4292 NBM-19-0283.R1.

12. Shmueli, K., J. A. de Zwart, P. van Gelderen, T.-Q. Li, S. J. Dodd and J. H. Duyn, "Magnetic Susceptibility Mapping of Brain Tissue in Vivo Using MRI Phase Data", *Magnetic Resonance in Medicine*, Vol. 62, No. 6, pp. 1510–1522, 2009.

13. Committee, Q. C. . O., B. Bilgic, C. Langkammer, J. P. Marques, J. Meineke, C. Milovic and F. Schweser, "QSM Reconstruction Challenge 2.0: Design and Report of Results", *Magnetic Resonance in Medicine*, Vol. 86, No. 3, pp. 1241–1255, 2021.

14. Deistung, A., F. Schweser and J. R. Reichenbach, "Overview of Quantitative Susceptibility Mapping", *NMR in Biomedicine*, Vol. 30, No. 4, p. e3569, 2017, e3569 NBM-15-0326.R2.

15. Wei, H., R. Dibb, Y. Zhou, Y. Sun, J. Xu, N. Wang and C. Liu, "Streaking Artifact Reduction for Quantitative Susceptibility Mapping of Sources With Large Dynamic

Range", *NMR in Biomedicine*, Vol. 28, No. 10, pp. 1294–1303, 2015.

16. Li, W., B. Wu and C. Liu, "Quantitative Susceptibility Mapping of Human Brain Reflects Spatial Variation in Tissue Composition", *NeuroImage*, Vol. 55, No. 4, pp. 1645–1656, 2011.

17. Liu, C., "Susceptibility Tensor Imaging", *Magnetic Resonance in Medicine*, Vol. 63, No. 6, pp. 1471–1477, 2010.

18. Gkotsoulias, D. G., C. Jäger, R. Müller, T. Gräßle, K. M. Olofsson, T. Møller, S. Unwin, C. Crockford, R. M. Wittig, B. Bilgic and H. E. Möller, "Chaos and COSMOS—Considerations on QSM methods with multiple and single orientations and effects from local anisotropy", *Magnetic Resonance Imaging*, Vol. 110, pp. 104–111, 2024.

19. Yoon, J., E. Gong, I. Chatnuntawech, B. Bilgic, J. Lee, W. Jung, J. Ko, H. Jung, K. Setsompop, G. Zaharchuk, E. Y. Kim, J. Pauly and J. Lee, "Quantitative Susceptibility Mapping Using Deep Neural Network: QSMnet", *NeuroImage*, Vol. 179, pp. 199–206, 2018.

20. Jung, W., J. Yoon, S. Ji, J. Y. Choi, J. M. Kim, Y. Nam, E. Y. Kim and J. Lee, "Exploring Linearity of Deep Neural Network Trained QSM: QSMnet+", *NeuroImage*, Vol. 211, p. 116619, 2020.

21. Bollmann, S., K. G. B. Rasmussen, M. Kristensen, R. G. Blendal, L. R. Østergaard, M. Plocharski, K. O'Brien, C. Langkammer, A. Janke and M. Barth, "DeepQSM - Using Deep Learning to Solve the Dipole Inversion for Quantitative Susceptibility Mapping", *NeuroImage*, Vol. 195, pp. 373–383, 2019.

22. Liu, J., A. S. Nencka, L. T. Muftuler, B. Swearingen, R. Karr and K. M. Koch, "Quantitative Susceptibility Inversion Through Parcellated Multiresolution Neural Networks and K-Space Substitution", *arXiv preprint arXiv:1903.04961*, 2019.

23. Goodfellow, I., Y. Bengio and A. Courville, *Deep Learning*, MIT Press, Cambridge, MA, 2016.

24. He, J., L. Wang, Y. Cao, R. Wang and Y. Zhu, "Learn Less, Infer More: Learning in the Fourier Domain for Quantitative Susceptibility Mapping", *Frontiers in Neuroscience*, Vol. 16 - 2022, p. 837721, 2022.

25. Zhang, J., Z. Liu, S. Zhang, H. Zhang, P. Spincemaille, T. D. Nguyen, M. R. Sabuncu and Y. Wang, "Fidelity Imposed Network Edit (FINE) for Solving Ill-Posed Image Reconstruction", *NeuroImage*, Vol. 211, p. 116579, 2020.

26. Xiong, Z., Y. Gao, Y. Liu, A. Fazlollahi, P. Nestor, F. Liu and H. Sun, "Quantitative Susceptibility Mapping Through Model-Based Deep Image Prior (MoDIP)", *NeuroImage*, Vol. 291, p. 120583, 2024.

27. Ulyanov, D., A. Vedaldi and V. Lempitsky, "Deep Image Prior", *International Journal of Computer Vision*, Vol. 128, No. 7, p. 1867–1888, 2020.

28. Ashley, S., "QSM-Forward GitHub Repository", `https://github.com/astewartau/qsm-forward`, accessed on June 12, 2025.

29. Dumoulin, V. and F. Visin, "A Guide to Convolution Arithmetic for Deep Learning", *arXiv preprint arXiv:1603.07285*, 2016.

30. Odena, A., V. Dumoulin and C. Olah, "Deconvolution and Checkerboard Artifacts", *Distill*, 2016, `http://distill.pub/2016/deconv-checkerboard`, accessed on June 12, 2025.

31. Gong, E., B. Bilgic, K. Setsompop, A. Fan, G. Zaharchuk and J. Pauly, "Accurate and Efficient QSM Reconstruction Using Deep Learning", *Proceedings of the 26th Annual Meeting of International Society for Magnetic Resonance in Medicine*, p. 0189, Paris, France, 2018.

32. Wu, B., W. Li, A. Guidon and C. Liu, "Whole brain susceptibility mapping using compressed sensing", *Magnetic Resonance in Medicine*, Vol. 67, No. 1, pp. 137–147, 2012.

33. Smith, L. N. and N. Topin, "Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates", *arXiv preprint arXiv:1708.07120*, 2018.

34. Ȯzbay, P., *Quantitative Susceptibility Mapping (QSM): Progress and Brain Applications*, Ph.D. Thesis, ETH Zurich, 2016.

35. Schweser, F., A. Deistung, K. Sommer and J. R. Reichenbach, "Toward Online Reconstruction of Quantitative Susceptibility Maps: Superfast Dipole Inversion", *Magnetic Resonance in Medicine*, Vol. 69, No. 6, pp. 1581–1593, 2013.

36. LeCun, Y., Y. Bengio and H. G., "Deep Learning", *Nature*, Vol. 521, No. 7553, pp. 436 – 444, 2015.

37. Ronneberger, O., P. Fischer and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation", N. Navab, J. Hornegger, W. M. W. III and A. F. Frangi (Editors), *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2015 - 18th International Conference Proceedings, Part III*, Vol. 9351 of *Lecture Notes in Computer Science*, pp. 234–241, Springer, Munich, Germany, 2015.

38. Bollmann, S., M. H. Kristensen, M. S. Larsen, M. V. Olsen, M. J. Pedersen, L. R. Østergaard, K. O'Brien, C. Langkammer, A. Fazlollahi and M. Barth, "SHARQnet – Sophisticated Harmonic Artifact Reduction in Quantitative Susceptibility Mapping Using a Deep Convolutional Neural Network", *Zeitschrift für Medizinische Physik*, Vol. 29, No. 2, pp. 139–149, 2019, special Issue: Deep Learning in Medical Physics.

39. Schweser, F., A. Deistung, B. W. Lehr and J. R. Reichenbach, "Quantitative Imag-

ing of Intrinsic Magnetic Tissue Properties Using MRI Signal Phase: An Approach to in Vivo Brain Iron Metabolism?", *NeuroImage*, Vol. 54, No. 4, pp. 2789–2807, 2011.

40. Ioffe, S. and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", F. Bach and D. Blei (Editors), *Proceedings of the 32nd International Conference on Machine Learning*, Vol. 37 of *Proceedings of Machine Learning Research*, pp. 448–456, PMLR, Lille, France, 2015.

41. Santurkar, S., D. Tsipras, A. Ilyas and A. Madry, "How does batch normalization help optimization?", *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, Vol. 31 of *NIPS'18*, p. 2488–2498, Curran Associates Inc., Red Hook, NY, USA, 2018.

42. Micikevicius, P., S. Narang, J. Alben, G. F. Diamos, E. Elsen, D. García, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh and H. Wu, "Mixed Precision Training", *6th International Conference on Learning Representations, ICLR*, pp. 1–12, OpenReview.net, Vancouver, BC, Canada, 2018.

43. Nwankpa, C., W. Ijomah, A. Gachagan and S. Marshall, "Activation Functions: Comparison of Trends in Practice and Research for Deep Learning", *arXiv preprint arXiv:1811.03378*, 2018.

44. Sun, R.-Y., "Optimization for Deep Learning: An Overview", *Journal of the Operations Research Society of China*, Vol. 8, No. 2, pp. 249–294, 2020.

45. Kingma, D. P. and J. Ba, "Adam: A Method for Stochastic Optimization", Y. Bengio and Y. LeCun (Editors), *3rd International Conference on Learning Representations, ICLR*, pp. 1–15, San Diego, CA, USA, 2015.

46. Loshchilov, I. and F. Hutter, "SGDR: Stochastic Gradient Descent with Warm

Restarts", *5th International Conference on Learning Representations, ICLR*, pp. 1–11, OpenReview.net, Toulon, France, 2017.

APPENDIX A: NEURAL NETWORKS AND DEEP LEARNING

Machine learning techniques have been used to automate detection and estimation tasks. Usually Machine Learning algorithms are used after useful discriminative features have been extracted from raw data. These features are then used for classification or regression, as necessary. Deep neural network methods simplify this pipeline by removing the need to manually encode useful representations or features. In deep learning, useful features are learned in an end-to-end manner with the task of interest and the learning is done by using different layers to learn increasingly complicated features which can then be used to classify the test pattern into the correct class [36]. We aim to use these networks to obtain features and perform regression, using those features, on a voxel by voxel basis to calculate our susceptibility at every voxel, hence obtaining our susceptibility map. In this chapter we examine different optimization strategies employed in training the deep learning models proposed in this work. These optimizations were necessary to be able to fit the model and training data on the available GPU memory.

A.1. Batch Normalization

When training a deep neural network, the output of every layer changes with every update of the weights of that layer. This phenomenon, generally referred to as internal covariate shift, is one of the numerous complications of training deep neural networks [40]. Batch normalization is one of the methods proposed to solve this issue. Batch normalization introduces a normalizing step into the architecture by replacing the network output for a given batch with its zero mean and unit variance version. To obtain the zero mean and unit variance version, the mean and the variance of a batch are first obtained then normalization is done by subtracting the mean and dividing by the variance. A small value is added to the variance to prevent division by zero.

Mathematically, this is written as

$$\hat{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma^2{}_B + \epsilon}} \tag{A.1}$$

where x_i is the $i-th$ sample of the minibatch, μ_B is the mean of the batch of samples with B samples, σ_B^2 is the batch variance and ϵ is a small value to prevent zero division. The resulting $\hat{x_i}$ is the normalized version of x_i. As restricting the output to always be zero mean and unit variance can limit the network's expressivity, a pair of parameters are further introduced to scale and shift the normalized output. This scaling and shifting restores the model's representation power [40]. The scaling and shifting is written mathematically as

$$y_i = \gamma \hat{x_i} + \beta \tag{A.2}$$

where γ and β are learnable parameters. These parameters can help to restore the mean and variance to any arbitrary values, it can be thought of as introducing a linear layer with weights γ and bias β. According to the authors, batch normalization not only deals with internal covariate shift, it can also speed up training by allowing the use of bigger learning rates and reduce the problems of too small or too large gradients. While the authors of the original paper have claimed that batch normalization works by getting rid of internal covariate shift, the authors in [41] have made the claim that this is not necessarily true. They went further to claim that the reason batch normalization helps speed up training is because it makes the training loss landscape smoother.

Introducing batch normalisation into the model described later in this work did speed up training but it also led to serious overfitting on the training set. To deal with this, regularisation techniques had to be used. These techniques are discussed in Section A.3.

A.2. Mixed Precision Training

Deep learning models are usually trained in 32 bits precision. This means that the weights, activations and gradients are stored in 32 bits, and the matrix operations are also done in 32 bits both in the forward calculation and in the backward gradient calculation. To reduce the memory requirement of training these models, training in a mixture of 16 bits and 32 bits has been proposed as a solution. This type of training is called Mixed Precision training. Here, the weights and activations are stored in 16 bits, certain operations like non-linearities and element-wise matrix multiplication are performed in 16 bits and others such as calculating batch normalization statis are done in 32 bits [42]. It has been shown that this training method nearly halves memory requirements while simultaneously maintaining model accuracy matching that of models trained with 32 bits precision. Training with mixed precision was employed in this work so as to save memory. To ensure that the model capability is maintained while training, the authors in [42] introduced the following training algorithms.

A.2.1. Maintaining Master Copy of Weights

The authors in [42] found that in some models, it was necessary to maintin a copy of the weights in 32 bits to match the performance of a 32 bits model. They maintain this weight copy by performing forward and backward operations in 16 bits, then during the weight updates, that is gradient descent step, 32 bits version of the gradients are used. This is useful because some gradients can become zero, after multiplication by the learning rate, as a result of underflow in 16 bits. Performing the weights update in single precision prevents this underflow. There is still a reduction in memory usage in this case because most memory consumption is dominated by activations from forward and backward propagations, and those are still stored 16 bits.

A.2.2. Loss Scaling

Most gradients obtained in many deep learning models are tiny in magnitude, this means that those gradients will underflow to zero when calculated in 16 bits precision. It has been observed that most of those gradient values do not make use of the entire range of 16 bits precision. It therefore makes sense to scale those gradient values up and push them into the full range that can be represented by 16 bits [42]. In practice, when using Python's PyTorch package, after the loss value is calculate in 16 bits, that value is then scaled by some scaling factor before the gradients are calculated. Obviously, the gradients obtained will be scaled equally as the loss and the original gradients can be obtained by scaling by the gradients by the inverse of the scaling factor. This new gradient is then used for the gradient update. The scaling factor can be chosen to simply be a large constant, or it could be chosen such that an overflow does not occur in 16 bits. If an overflow occurs at any interation, the gradients become NaNs and the weight update is skipped for that iteration. Following the overflow, the scaling factor is then reduced and the non-NaN gradients can be obtained which can then be used for weight updates normally.

A.2.3. Arithmetic Precision

The final training method introduced in [42] involves the arithmetic precision of neural networks operations. According to those authors, vector dot-products can be done in 16 bits then accumulated into 32 bits, reduction operations that occur in operations such as batch normalization and softmax should be done in 32 bits and pointwise operations can be done in either 16 bits or 32 bits. In PyTorch, the precision for each operation arre automatically chosen once the option is enabled. As an example, Fourier transforms are automatically done in 32 bits and attempts to force it to be done 16 bits can lead to NaN outputs.

A.3. Optimization, Schedulers and Regularisation

A.3.1. Optimization

Given a model architecture, deep learning aims to learn model weights such that a loss function is minimized. The process of iteratively improving on the loss function by modifying the learned weights is called model optimization. Several algroithms have been developed for deep learning model optimization and some of them are explained here. All algorithms described here are based on the gradient descent algorithm.

A.3.1.1. Stochastic Gradient Descent (SGD). Stochastic Gradient Descent (SGD) minimizes the loss function by moving the weight in a direction opposite that of the gradient of the loss with respect to the model weights. It is stochastic in the sense that at every iteration, a different, random batch is used for loss and gradient calculation. Mathematically, it is written as

$$\theta = \theta - \epsilon \nabla_\theta L(x, \theta) \tag{A.3}$$

where θ are the model weights, ϵ represents the learning rate and L represents the loss per batch of samples. The learning rate is a parameter that controls how much the weights are updated at any iteration, and it is one of the most important hyperparameters in deep learning training.

A.3.1.2. RMSProp and Adam. Both RMSProp and Adam are examples of training algorithms with adaptive learning rates [23, pp. 302–306]. In RMSProp, the gradient used for gradient updates is normalized by the square root of the historical sum of its own squares. It is adaptive learning rate in the sense that this normalization is weight-dependent and is modifying the learning rate for each weight individually. The normalization essentially means that only the sign of the weight, not the magnitude, is used for weight updates. Normalization in this manner means larger gradients would

have smaller learning rates, hence ensuring a smoother training and convergence. In addition to normalizing the gradients, Adam also normalizes the momentum updates as well [45]. Adam is one of the most popular training algorithms currently in use because it is robust and does not require too much tuning to perform decently well [44]. As a result of its robustness, the Adam optimizer was used during training for all models described in this work.

A.3.1.3. Gradient Accumulation. Gradient accumulation is not an optimization algorithm but a way to "increase" a training batch size without running into GPU memory issues. For all the optimization algorithms disscussed, a weight update step is normally taken at every iteration. Each iteration uses a batch size, which is the number of samples one uses at that iteration. The loss and gradient are calculated using only those samples in the batch. Sometimes the batch size is limited by the amount of GPU memory available for training. A limited batch size means we have a small number of samples to calculate the gradient which can make training unstable. To remedy this, one can instead take n steps and add up the gradients over those steps then perform the weight update with the average gradient over those n steps. This has effectively increased the batch size to $n \times batchsize$ without increasing the amount of memory consumed.

A.3.2. Learning Rate Schedulers

Although the learning rate is written in Equation A.3 as a fixed value, in practice, it is often necessary to decrease its value over the course of training a neural network. This is because gradients in SGD are inherently noisy and will not become zero even at a minimum [23, pp. 290]. Several methods for reducing the learning rates have been proposed, and they are collectively known as learning rate scheduling algorithms. Some of those algorithms are described here.

A.3.2.1. Linear Schedule. In a linear schedule, the learning rate is made to vary linearly with the number of iterations. An example schedule can be written as [23, pp. 291]:

$$\epsilon_k = (1 - \beta)\epsilon_0 - \beta\epsilon_\tau \tag{A.4}$$

where k is the iteration number, and $\beta = \frac{k}{\tau}$. It can be seen here that the learning rate will vary from an initial value of ϵ_0 to a value ϵ_τ after τ iterations. The learning rate value can then be fixed at ϵ_τ, which is lower than the initial learning rate, for the remaining number of iterations.

A.3.2.2. OneCycleLR Schedule. This method was introduced in [33]. Unlike the previous methods where we start from a maximum LR and anneal to a minimum, here, the LR starts at a small initial value. From this initial value, the learning rate is annealed to a maximum value then annealed back to the initial starting learning rate. From this value, it is then annealed to a minimum value. A subtle variant, which is implemented in PyTorch, has the learning rate annealed directly from the maximum value to a minimum value (which is different from the initial value) directly. A choice can be made as to whether to increase and decrease the learning rate linearly or using some other form of schedule. Linear annealing is used in this work.

A.3.3. Regularisation

Deep learning models are able to memorise their training data as a result of having a large number of parameters. Memorising training data leads these models to perform poorly on unseen validation data. Several strategies have been developed to mitigate this phenomenon and those strategies are collectively known as regularisation [23, pp. 224]. Some popular regularisation methods are explained in this section.

A.3.3.1. Weight Norm Penalties. Weight norm penalties are a class of regularisation strategies in which a weighted norm of the model parameters is added to the loss of function of interest. Given a loss function, L, a weight norm regularised loss function is defined as

$$J(x, \theta) = L(x, \theta) + C\Omega(\theta) \tag{A.5}$$

where C is the weighting that controls how much regularisation is used. For large values of C, more regularisation is used while smaller regularisation is used for smaller values. As we can see from Equation A.5, minimizing this new loss function requires minimizing the weight values which would then reduce model capacity and hence reduce overfitting. Ω specifies the type of norm used for defining the regularisation. It may be L^2 regularisation, in which case it is defined as

$$\Omega(\theta) = \frac{1}{2}||\theta||_2^2 \tag{A.6}$$

and it may be L^1 in which case it is defined as

$$\Omega(\theta) = ||\theta||_1. \tag{A.7}$$

The L^1 regularisation is said to be sparsity-promoting which means that it can set the unimportant weights to exactly zero, unlike the L^2 [23, pp. 232].

A.3.3.2. Data Augmentation. As regularisation is about reducing a model's generalisation error, one of the best ways to regularise a model is therefore to train with as much data as possible. Training with more data may be impractical, therefore it is often necessary to generate fake or synthetic data that try to represent real world distribution of data. This process of generating fake training data is called data augmentation. In deep learning based QSM, rotation has been used as a data augmentation method [19] and the same trend was followed in this work.

APPENDIX B: DATA LICENSE/PERMISSION

In addition to the data generated in-house, the methods in this work were tested on phantom brain data obtained from [13]. Permission to use this data is freely given as can be seen in the Data Availability section of the work where the authors say "The code and data of this QSM reconstruction challenge are openly available".