Bachelor's project

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Anders Wiik

Noise removal in synthetically generated diffusion tensor imaging data using a denoising autoencoder

Bachelor's project in Mathematical Studies

Supervisor: Elena Celledoni

December 2020

NTNU
Norwegian University of Science and Technology

Anders Wiik

Noise removal in synthetically generated diffusion tensor imaging data using a denoising autoencoder

Bachelor's project in Mathematical Studies
Supervisor: Elena Celledoni
December 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

NTNU
Norwegian University of
Science and Technology

Abstract

Diffusion Tensor Imaging (DTI) is a popular medical imaging technique that maps the diffusion of water molecules in biological tissue. Measured DTI data is contaminated by noise, and the topic of this project was to explore ways of removing this noise. A denoising autoencoder is an algorithm that has proven to be effective at denoising images, and has been used successfully for medical image denoising. This report investigates whether denoising autoencoders can be used to remove noise from DTI data as well. We describe how we implemented a simple example of a denoising autoencoder in a computer program, and how it was used to denoise synthetically generated DTI data. The performance of the denoising autoencoder was compared to total variation denoising. Our results agree with the statement that denoising autoencoders are effective at removing noise from medical images.

Acknowledgements

I would like to thank my thesis advisor, Elena Celledoni, for making this project possible and always offering help.

Data were provided [in part] by the Human Connectome Project, WU-Minn Consortium (Principal Investigators: David Van Essen and Kamil Ugurbil; 1U54MH091657) funded by the 16 NIH Institutes and Centers that support the NIH Blueprint for Neuroscience Research; and by the McDonnell Center for Systems Neuroscience at Washington University.

1 Introduction

Diffusion Tensor Imaging (DTI) is a medical imaging technique used to map the diffusion of water molecules in biological tissue. The technique has been used in many different neuroscientific studies, and DTI has been used for planning neurosurgery on patients, with good results [1]. DTI data is measured using magnetic resonance imaging, resulting in a tensor which can be represented by a 3×3 symmetric positive definite matrix at each point. Measured DTI data is contaminated by noise, and so one might be interested in ways of removing noise from this type of data. This has been done before, by using total variation denoising [2] [3]. Another type of algorithm that can be used to remove noise from images is a denoising autoencoder [4] [5] [6]. Denoising autoencoders have been used to denoise medical images in [7] and [8], and the goal of this project was to investigate whether denoising autoencoders can be used to denoise DTI data as well. This report describes an implementation of a denoising autoencoder algorithm in a computer program, and the results of using this algorithm to denoise synthetically generated DTI data. Total variation denoising was also implemented, and the results of the two methods were compared.

First off, we will describe DTI data in more detail, as well as how this type of data can be visualized. Then we will give a description of the denoising autoencoder algorithm, and of how total variation denoising is done. Next, we explain how we implemented the two algorithms in a computer program. The final part of the report contains an explanation of the experiment that was performed and a presentation of the results. Topics from linear algebra such as eigenvectors, eigenvalues, and symmetric positive definite matrices, will be important for understanding this manuscript, and we assume the reader is familiar with these concepts.

2 Diffusion Tensor Imaging

Diffusion Tensor Imaging is used to map the diffusion of water in biological tissue. DTI images are represented as a tensor, i.e. a multidimensional array. This tensor is calculated from data that has been acquired through magnetic resonance imaging (MRI) [9]. The MRI measurements of the diffusion tensor will often be contaminated by noise [10], and the next paragraphs will explore two ways of reducing noise in the measured DTI data. After the tensor is calculated, the DTI data can be considered to be a grid of voxels [3]. Each voxel is a 3x3 symmetric positive definite (SPD) matrix, and these matrices describe the diffusion of water at that point in the brain, where the eigenvalues of the matrix describe the diffusion rates along the direction of each corresponding eigenvector [1].

If one wishes to visualize these types of images, one way is to represent the SPD matrix at each voxel as an ellipsoid. The correspondence between SPD matrices and ellipsoids is given by the eigenvalues and eigenvectors of the matrix. Since the matrix is real and symmetric, there will be three orthogonal

eigenvectors. An ellipsoid can be constructed by taking its three axes to be the directions of the eigenvectors, and make the lengths along each axis depend on the corresponding eigenvalues [1]. One could then pick a rectangular region in the grid of SPD matrices that make up the DTI data, and plot the ellipsoid corresponding to each matrix in a plane. This results in a visualization of the diffusion tensor in that region. The ellipsoids can also be colored to make the plot more informative, by for example having color correspond to the direction of the major axis [11]. Visualizations like this will be used in the later parts of this report.

3 Denoising Autoencoders

A denoising autoencoder (DAE) is a type of artificial neural network that has proven to be effective at denoising images [4], so it is possible that they could be used to remove noise from DTI data as well. Denoising autoencoders are defined in [4], and we will restate the simple case of a denosing autoencoder that we are going to implement in a computer program. Denoising autoencoders are a variant of traditional autoencoders, which in turn consists of two parts: An encoder, that transforms an input vector into a lower dimensional representation, and a decoder, which transforms the lower dimensional representation back into a vector in the input space. If $\mathbf{x} \in \mathbb{R}^d$ is an input vector, the encoder $f_\theta : \mathbb{R}^d \to \mathbb{R}^{d'}$ can be written as

$$f_\theta(\mathbf{x}) = s(\mathbf{W}\mathbf{x} + \mathbf{b})$$

Here, $\theta = \{\mathbf{W}, \mathbf{b}\}$ is the set of parameters. $\mathbf{W}$ is a $d' \times d$ matrix, and $\mathbf{b}$ is a vector. s is a non-linear function, for example the sigmoid function $s(x) = \frac{1}{1-e^{-x}}$, where $s(\mathbf{x})$ is defined to be $(s(\mathbf{x}_1), \ldots, s(\mathbf{x}_d))^T$. The form of the decoder $g_{\theta'} : \mathbb{R}^{d'} \to \mathbb{R}^d$ is similar to the encoder:

$$g_{\theta'}(\mathbf{y}) = s(\mathbf{W}'\mathbf{y} + \mathbf{b}')$$

with $\theta' = \{\mathbf{W}', \mathbf{b}'\}$ being the set of parameters. In this case, the nonlinearity is optional, even though it is necessary for the encoder. $\mathbf{y}$ should be thought of as the output from the encoder, and since we want the result of the decoder to be in the input space, $\mathbf{W}'$ must be a $d \times d'$ matrix. Since traditional autoencoders have $d' < d$ [4], one can think of the autoencoder as first reducing the number of dimensions of $\mathbf{x}$ to get $\mathbf{y} = f_\theta(\mathbf{x})$, and then restoring the dimensions to get $\mathbf{z} = g_{\theta'}(\mathbf{y}) = g_{\theta'}(f_\theta(\mathbf{x}))$.

The goal of the autoencoder is to have $\mathbf{z} = g_{\theta'}(\mathbf{y})$ be a reconstructed version of $\mathbf{x}$. But $\mathbf{z}$ will not be an exact reconstruction, since the reduction of dimension forces the autoencoder to discard some of the information in the input vector. The purpose of an autoencoder is to separate useful information from noise, i.e. to figure out what to discard, and what to encode into the compressed representation $\mathbf{y}$, while still being able to recover the original data as well as possible.

To achieve this one begins by defining a loss function $L(\mathbf{x}, \mathbf{z})$. This function should describe the reconstruction error, that is, when $\mathbf{z}$ is a good reconstruction of $\mathbf{x}$, $L(\mathbf{x}, \mathbf{z})$ should be small. So the process of 'training' the autoencoder consists of finding the parameter sets θ and θ' that minimize the loss function when it is averaged over a set of input data.

A denoising autoencoder has a similar structure to a traditional autoencoder, but the goal is to learn a mapping that removes noise. The difference is that the input vectors are corrupted before they are encoded, meaning that $\mathbf{x}$ is turned into $\tilde{\mathbf{x}}$ by adding random noise. Now, the hidden representation becomes $\mathbf{y} = f_\theta(\tilde{\mathbf{x}})$, and the reconstructed vector is $\mathbf{z} = g_{\theta'}(\mathbf{y})$ like before. When the autoencoder is trained to minimize the loss $L(\mathbf{x}, \mathbf{z})$, the parameters will be chosen so as to reconstruct the original input vector from the corrupted one. In this way, the denoising autoencoder is trained to remove noise from the input vector.

4 Total variation denoising

Another way of approaching this problem is with Total Variation (TV) denoising. To perform TV denoising on an $l \times m$ image $s = (s_{11}, \ldots, s_{lm})$, where s_{ij} are the pixels of the image, one has to find a new image $u = (u_{11}, \ldots, u_{lm})$ that minimizes the total variation functional:

$$V(u) = \frac{1}{\beta} \sum_{i,j=1}^{l,m} d(u_{ij}, s_{ij})^\beta + \lambda \left(\sum_{i,j=1}^{l-1,m} d(u_{ij}, u_{i+1,j})^\gamma + \sum_{i,j=1}^{l,m-1} d(u_{ij}, u_{i,j+1})^\gamma \right)$$

Here, $d(x, y)$ is a distance function defined for pixels x, y, and β, λ, γ are parameters [3]. To minimize this functional, it is possible to use iterative methods like gradient descent. In the case of DTI data, where each pixel is an SPD matrix, this would not guarantee that the matrices stay SPD. If one wanted to guarantee that each iteration only consists of SPD matrices, one could use the algorithms described in [2] or [3]. By considering the manifold of symmetric positive definite matrices $\mathrm{Sym}^+(3)$, these articles presents iterative methods that can be used to minimize the TV functional so that each iteration stays on the manifold.

One key difference between total variation denoising and denoising autoencoders is that the parameters of an autoencoder only have to be trained once. After determining the parameters, the denoising autoencoder can be used to denoise an arbitrary amount of images, without much computational cost. In contrast, when using TV denoising, one would have to minimize the total variation functional for each image separately.

A disadvantage of using a denoising autoencoder over total variation denoising is that a data set of images without noise is needed to learn good choices for the parameters. Depending on the situation, this type of data might be impractical to acquire. On the other hand, total variation denoising does not

require any data. The only prerequisite for minimizing the TV functional is the single image one wants to denoise.

5 Experiments

5.1 Computer program

5.1.1 Generating synthetic data

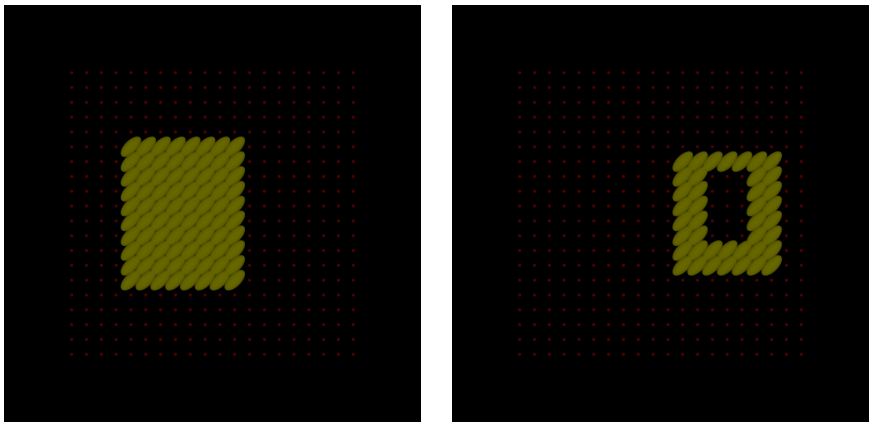
To help investigate the performance of denoising autoencoders, it would be helpful to implement a simple example of an autoencoder in a computer program. The prerequisite for the autoencoder to work, is having a data set that can be used to train the parameters. In our experiments this problem was solved by generating the data set synthetically. Since DTI data can be considered to be a grid of voxels, where each voxel is a 3×3 SPD matrix, one can construct a data set by making a program output a large number of such images, along with versions of the same images with random noise added to them. For the purposes of testing, it is useful to construct the data in a way that makes it obvious whether the denoising is working properly. In this case, we did this by generating the data so that the ellipsoid plot would display predetermined geometric shapes. We wrote code to fill a 20 x 20 grid with SPD matrices having small eigenvalues, and picked a random position to place a randomly sized rectangle of SPD matrices with larger eigenvalues, and with eigenvectors slightly rotated about the z-axis. Code was also written to create data sets where the rectangles had holes in them, creating a slightly different geometric shape in the ellipsoid plot. We will refer to this figure as a box. This was useful because it let me generate two different, but similar, types of data sets. By doing this, we could later test how the autoencoder would perform when used to remove noise from data that was not exactly equal to what was contained in the training data set. Figure 1 shows an example of an ellipsoid plot of an image with a rectangle shape, and of an image with a box shape. All the ellipsoid plots in this report are made with the DIPY Python library, using the diffusion imaging functionality from that library [12]. The colors of the ellipsoids depend on the direction of the eigenvector corresponding to the largest eigenvalue [11], and the brightness of the colors depend on the fractional anisotropy at each point. Fractional anisotropy is a function of the eigenvalues and is given by

$$FA = \sqrt{\frac{1}{2}\frac{(\lambda_1 - \lambda_2)^2 + (\lambda_1 - \lambda_3)^2 + (\lambda_2 - \lambda_3)^2}{\lambda_1^2 + \lambda_2^2 + \lambda_3^2}}$$

It can be viewed as a measure of how far away the ellipsoid is from a sphere [1].

5.1.2 Generating random noise

Different ways of adding noise to the generated data were considered. In the case of our experimentation, we chose to generate the noise in a way that kept



Rectangle shape Box shape

Figure 1: An example of each of the two types of images used in the synthetically generated data sets in this report.

the program simple instead of in a way that was guaranteed to be realistic. One goal was to keep all the matrices symmetric and positive definite after random noise was added. This was done by calculating the singular value decomposition (SVD), and adding noise to each of the singular values in a way that kept them positive. Note that this keeps the resulting matrix product symmetric positive definite. To see this let A be SPD, and its SVD be $A = U\Sigma V^T$. Then $A^2 = A^T A = (V\Sigma V^T)^2$, since V is orthogonal. This means that A and $V\Sigma V^T$ are both positive definite matrix square roots of the SPD matrix A^2, but this square root is unique [13], so $V\Sigma V^T = A = U\Sigma V^T$. Which shows that the $U = V$ in the SVD of positive definite matrices. So the SVD is of form $A = V\Sigma V^T$, where V is an orthogonal matrix. Replacing Σ by any diagonal matrix S with positive elements, makes VSV^T symmetric positive definite. Knowing this, we picked two different ways of adding noise for our experiments. The noise was generated by adding uniformly distributed random numbers from the interval $[0, a]$ where a is some constant, or from the interval $[-r\sigma_i, r\sigma_i]$ where σ_i is the singular value, and r is some number in $(0, 1)$. The first interval gives noise with positive expected value, and the second gives noise with mean 0. It should be noted that this is not necessarily realistic in terms of how noise occurs in real world DTI data, but instead that it was done this way to keep the code simple.

After generating the synthetic training and testing data sets, the data sets were saved for use with the autoencoder. It should also be mentioned that instead of representing the SPD matrices as 3x3 arrays, they were represented

as vectors with 6 elements, by letting $(a_1, a_2, a_3, a_4, a_5, a_6)$ correspond to

$$\begin{pmatrix} a_1 & a_2 & a_3 \\ a_2 & a_4 & a_5 \\ a_3 & a_5 & a_6 \end{pmatrix}$$

This could be useful for reducing the file size of the data sets, but most importantly it guaranteed that the output of the computer program will consist of only symmetric matrices, since no matter how $(a_1, a_2, a_3, a_4, a_5, a_6)$ is changed by the code, reconstructing the matrix will always lead to a symmetric matrix.

5.1.3 Implementing the denoising autoencoder

The autoencoder was then implemented in the Python programming language using TensorFlow, and the Keras API [14]. The Sequential model from the Keras API can be used to implement a simple autoencoder. In this model, a neural network is viewed as a stack of layers. A layer can be viewed as a function having one input and one output, and possibly some parameters. In the Sequential model, the output of one layer is the input to the next one. The most important type of layer for this simple example was the Dense layer, which implements the operation $s(\mathbf{W}\mathbf{x} + \mathbf{b})$. Two Dense layers were used to implement the encoder and the decoder operations, and in both cases, the $\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ function was used as the nonlinearity. Since the x in the expression is a vector of real numbers, and the input image is a multidimensional array, the input had to be converted into a vector before being encoded. The simplest solution to this was to use a Flatten layer to move the values from the input image into a vector of length $20 \cdot 20 \cdot 6$. To make the output of the autoencoder be an image instead of a vector, the final layer of the model needed to be a Reshape layer. This layer converts a vector into a multidimensional array of given dimensions, and is used here as the inverse operation of the Flatten layer. All the data was normalized before being used with the denoising autoencoder so that every element was contained in the interval $[-1, 1]$.

In summary, the input image, which is a $20 \times 20 \times 6$ array, is moved into a vector $\mathbf{x}$ of length 2400 after every element is normalized to the interval $[-1, 1]$. The operation $\mathbf{y} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b})$ is applied, producing an encoded representation $\mathbf{y}$. The dimension of $\mathbf{y}$ is determined by d', the number of rows of the parameter matrix $\mathbf{W}$. In our experiments, $d' = 400$ seemed to give good results. $\mathbf{z} = \tanh(\mathbf{W}'\mathbf{y} + \mathbf{b}')$ is then calculated, and has dimension 2400. Finally this vector is converted back into an array of dimensions $20 \times 20 \times 6$.

5.1.4 Defining a loss function

The next step is to define a loss function, and use Keras to find parameters that minimize this loss function over the training data set. Different loss functions

were considered, and the one used in the results below was

$$L(u,s) = \frac{1}{\beta}\sum_{i,j=1}^{l,m} d(u_{ij},s_{ij})^{\beta} + \lambda\left(\sum_{i,j=1}^{l-1,m} d(u_{ij},u_{i+1,j})^{\gamma} + \sum_{i,j=1}^{l,m-1} d(u_{ij},u_{i,j+1})^{\gamma}\right),$$

where s is the original image, and u is the output of the autoencoder. This was inspired by the total variation functional, with the key difference being that the first term compares the uncorrupted version of the image with the result of the denoising. In classical total variation denoising, this term compares the first image in the sequence with the next iterations instead. β, γ and λ are constant parameters. The function $d(u_{ij}, s_{ij})$ is a distance function defined for voxels u_{ij} and s_{ij}. This function was picked to be the 2-norm: $||u_{ij} - s_{ij}||_2 = \sqrt{\sum_{k=1}^{6}(u_{ij}^{(k)} - s_{ij}^{(k)})^2}$, since $u_{ij} = (u_{ij}^{(1)}, \ldots, u_{ij}^{(6)})$ and $s_{ij} = (s_{ij}^{(1)}, \ldots, s_{ij}^{(6)})$ are vectors in $\mathbb{R}^6$. Another distance function that was considered is

$$d(A,B) = \sqrt{\sum_{i=1}^{3}\log(\kappa_i)^2},$$

where κ_i are the eigenvalues of $A^{-\frac{1}{2}}BA^{-\frac{1}{2}}$. This function is derived by viewing $\mathrm{Sym}^+(3)$, the space of 3×3 SPD matrices, as a Riemannian manifold with Riemannian metric

$$g_A(X,Y) = \mathrm{tr}(A^{-\frac{1}{2}}XA^{-1}YA^{-\frac{1}{2}}),$$

in which case $d(A, B)$ is the geodesic distance on $\mathrm{Sym}^+(3)$ [2]. Implementing this function in the program led to some problems. While the output of this implementation is guaranteed to consist of symmetric matrices, the matrices are not guaranteed to be SPD when the parameters have not yet been trained. This happens because the parameters are initialized to random values in the Dense layers of the Keras API. During the training of the parameters, the loss function is used to measure the distance between the original image and the result from the autoencoder, but since the parameters are random values, this results in the function $d(A, B)$ being used to measure the distance between matrices that might not be positive definite. One way to try to solve this problem is by finding the eigendecompositions $A = Q_1\Lambda_1Q_1^{-1}$, $B = Q_2\Lambda_2Q_2^{-1}$, and defining $d(A, B)$ to be $d(Q_1|\Lambda_1|Q_1^{-1}, Q_2|\Lambda_2|Q_2^{-1})$. Where $|\Lambda_i|$ means to take the absolute value of each element of Λ_i. This did not solve all the problems because after this, the optimization did not seem to converge, and each iteration was impractically slow. Still, it is not possible to conclude that this distance function is a bad choice, only that we could not find a good way to make it work. Using this function might give better results, but one might have to structure the computer program in a different way than what we have done.

5.1.5 Comparing results with total variation denoising

Since one goal of the project was to study the performance of a denoising autoencoder, it would be useful to compare the autoencoder to another way of

denoising DTI data. To do this, we implemented total variation denoising, by minimizing the TV functional $V(u)$ from section 4. This was done by using the gradient descent functionality from TensorFlow, performing gradient descent on u. $d(x, y)$ was defined to be the 2-norm, the same way it was for the autoencoder.

One way to compare the two denoising algorithms is by using the Structural Similarity Index (SSIM), which is an algorithm that measures the similarity of two images [15]. The SSIM is a function $S(x, y)$, where x, y are images. We have $S(x, y) \leq 1$, where higher values correspond to more similarity, and $S(x, y) = 1$ if and only if $x = y$. SSIM is originally defined for grayscale images, containing one value per pixel, but in the case of this experiment, each voxel contains 6 values instead of 1. By default, the TensorFlow implementation of SSIM solves this problem by taking the average of the SSIM values. So in this case, SSIM is calculated by separating the data into 6 images, with scalar values at each pixel, and then finding the average SSIM. The TensorFlow implementation of SSIM assumes that the values are in $[0, a]$, with a being some given max value. Since the data we want to apply the SSIM algorithm to has elements in $[-1, 1]$, we added 1 to each element, to shift every value to be in the interval $[0, 2]$. In the results presented in the following sections, SSIM was used to measure to what extent the denoised image was similar to the original, uncorrupted image.

5.2 Denoising results

5.2.1 Description of the experiment

The two denoising methods were tested on different combinations of data sets. In all cases the training data set consisted of 6000 images of rectangle shapes, as well as versions of the same images with added noise. The training data set contained 100, separately generated, images of either rectangle shapes or box shapes, with the same type of noise added. The noise was generated as described in section 5.1.2. Experiments were performed with uniformly distributed random noise with both positive and zero expected value. Two different lengths of intervals were tried for each uniform distribution: For the noise on the interval $[0, a]$, values of $a = {}^{\lambda_{max}}/_{2}$ and $a = {}^{\lambda_{max}}/_{4}$ were tried, where λ_{max} is the largest eigenvalue of all matrices in the image. For noise on the interval $[-r\sigma_i, r\sigma_i]$, we considered $r = 1/2$, and $r = 1/4$. The parameters of the autoencoder were trained with the training data set. This was done by using the Keras API, with the choice of the Adam optimizer [16] to minimize the loss function. After this, each image in the corresponding testing data set was denoised using these parameters. The similarity was calculated using SSIM as described above, and the average similarity over the 100 images in the testing data set is presented in Table 1. Total variation denoising was also used to denoise each of the 100 images in the testing data set. The average similarities are presented in the table, with the best performance highlighted in bold. Similarity between original images and the untreated, noisy version were also calculated, and this is presented in the second column. In addition to this table, Figure 2 gives an example of how

the denoised images might look.

5.2.2 Results of the experiment

Both denoising methods resulted in the denoised images having increased similarity to the original image, except for when the images had small amounts of noise with 0 expected value. Here, the denoising autoencoder resulted in unchanged or slightly reduced similarity. In most cases, total variation denoising seems to perform better, and it seems like the type of noise has an impact on the denoising performance of each method. Since only a simple example of each of the two denoising methods were implemented, these results are not sufficient to claim that total variation denoising outperforms denoising autoencoders in general. In [4], the authors describe ways that the basic autoencoder algorithm can be improved. In the same manner, total variation denoising might give better results when minimized with a method like the ones described in [2] or [3]. Furthermore, notice that the denoising autoencoder performs worse when the testing data is of a different type from the training data, while the performance of total variation denoising remains close to unchanged. Because the training set consisted of rectangle images, one might expect good denoising performance on other rectangle images, since the parameters were minimized to denoise this exact type of image. The box images are slightly different from the type of image used to train the parameters, which makes the DAE perform a bit worse when denoising box images instead of rectangle images. On the other hand, TV denoising does not depend on the training data set, so one might expect it to perform similarly in both cases.

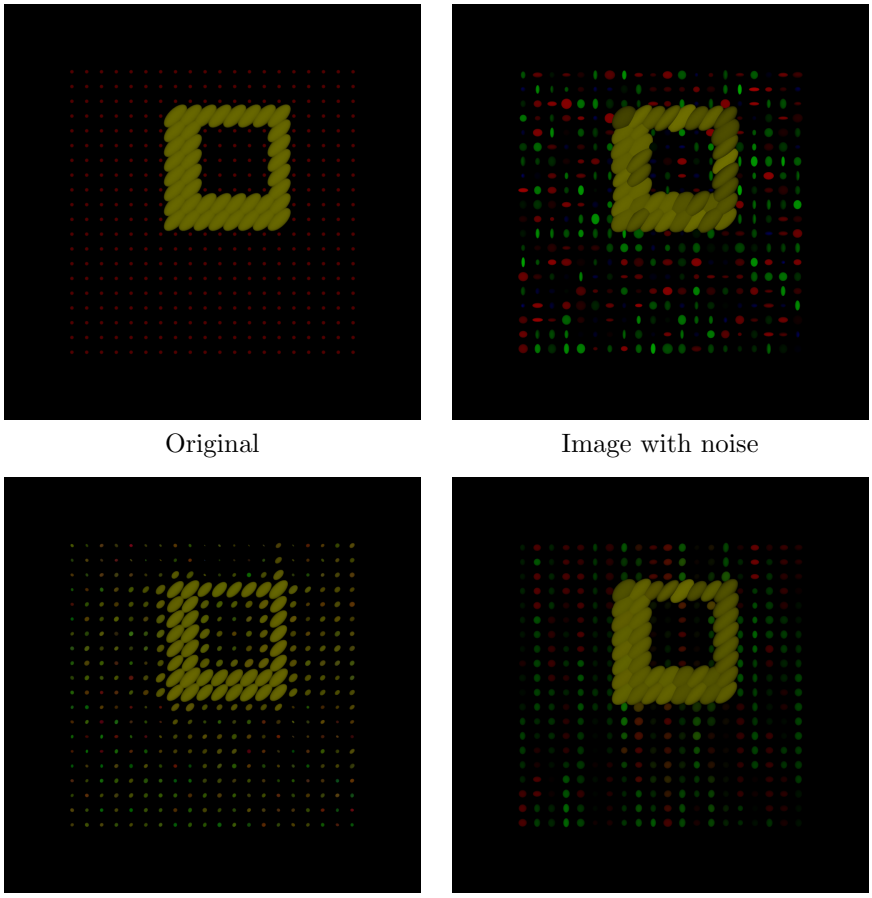
5.3 Remarks about using real world DTI data

Everything in this project could easily be made to work with real DTI data. Since DTI data sets are large, and reconstructing the diffusion tensor from the MRI data takes time, it was impractical to do this, but with more time and computing power one could do the same experiments with real DTI data. As a demonstration, preprocessed MRI data was downloaded from the Human Connectome Project [17], and the diffusion tensor was reconstructed using DIPY for 10 of the images. This resulted in 10 three-dimensional grids of SPD matrices. We then copied a 20×20 region of each z-slice into a new data set, and added noise like before. We could then use this data set made from 20×20 regions of real DTI data with our existing autoencoder implementation. Figure 3 shows an example of the region of the DTI data, and the version with noise, as well as the resulting denoised versions. Because of the way the data set was constructed, no conclusions should be made about what denoising method is better from this demonstration, but instead it shows that the implemented autoencoder could easily be applied to real DTI data.

Training data: rectangles, Testing data: rectangles			
Noise interval	Untreated, avg. SSIM	DAE, avg. SSIM	TV, avg. SSIM
$[0, \lambda_{max}/2]$	0.6945558	**0.9383318**	0.8298421
$[0, \lambda_{max}/4]$	0.7888118	**0.9449149**	0.9372864
$[-\sigma_i/2, \sigma_i/2]$	0.9170565	0.9555395	**0.9740480**
$[-\sigma_i/4, \sigma_i/4]$	0.9661060	0.9580894	**0.9886879**

Training data: rectangles, Testing data: boxes			
Noise interval	Untreated, avg. SSIM	DAE, avg. SSIM	TV, avg. SSIM
$[0, \lambda_{max}/2]$	0.6968152	**0.9126226**	0.8264212
$[0, \lambda_{max}/4]$	0.8012284	0.9327329	**0.9398828**
$[-\sigma_i/2, \sigma_i/2]$	0.9269519	0.9422862	**0.9765552**
$[-\sigma_i/4, \sigma_i/4]$	0.9730638	0.9520538	**0.9898018**

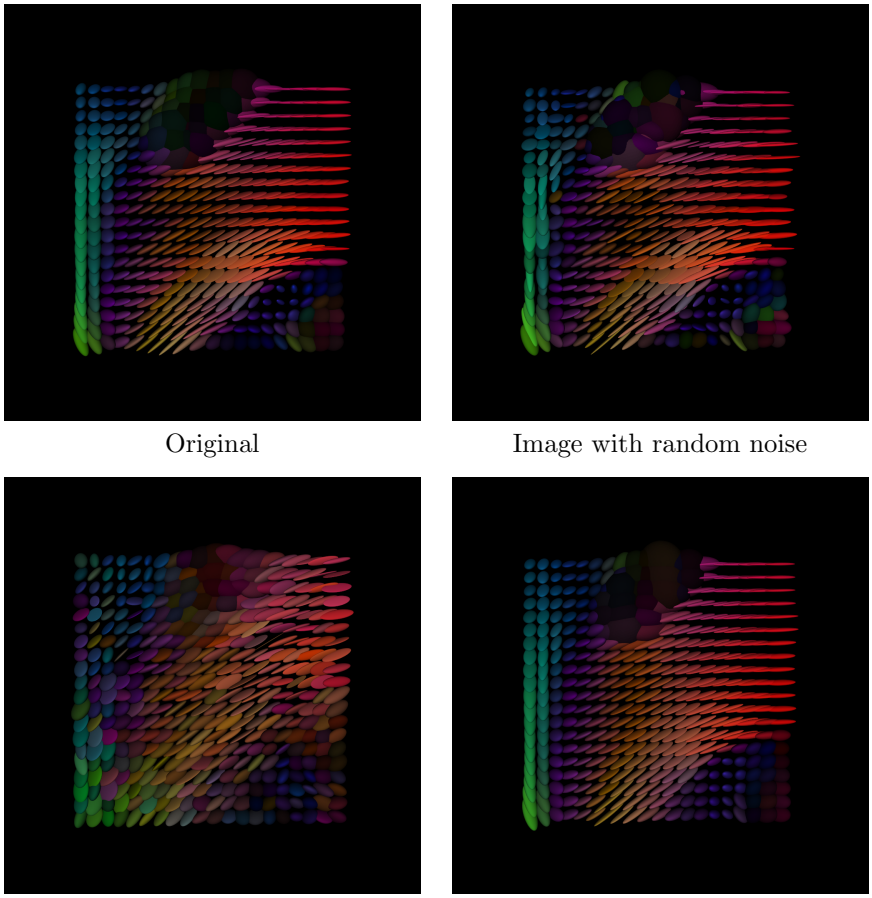
Table 1: Results of denoising. The two tables show the results of the experiments when performed with testing data sets consisting only of box images, or only of rectangle images. In both cases, the training data sets consisted solely of rectangle images. The first column shows the interval of the uniform distribution used to add noise to the singular values σ_i of each matrix. The second column shows the average similarity between the original, uncorrupted images and the noisy images, and the third and fourth column shows the similarity between original images and the results of the two denoising methods that were considered. All averages are over the 100 images in the testing data set. The parameters of the denoising autoencoder were trained on the 6000 images in the training data set.



Original | Image with noise

Denoising autoencoder | Total variation denoising

Figure 2: An example of denoising an image of a box shape, with noise from a uniform distribution on the interval $[0, \lambda_{max}/4]$. The autoencoder was trained on rectangle shapes, and the resulting SSIM value was 0.9257586. The SSIM value of the total variation denoising was 0.9646640. In both cases this improved the similarity, which was 0.7995793 for the untreated, noisy image.



Original | Image with random noise

Denoising autoencoder | Total variation denoising

Figure 3: The result of using the computer program on publicly available DTI data from the Human Connectome Project. This is an example of denoising applied to a single element in the testing data set, i.e. a 20×20 region of some z-slice of the DTI data. The untreated, noisy image had SSIM value 0.9024593, and the result of the denoising autoencoder had a reduced SSIM value of 0.7190125. Total variation denoising resulted in an SSIM value of 0.9487670.

6 Conclusion

We have considered a simple implementation of denoising autoencoders for diffusion tensor imaging and seen that it achieves good performance when compared to TV denoising. Denoising autoencoders have been shown to perform well for denoising images before, as seen in [4], [5] and [6], while [7] and [8] show that denoising autoencoders can be used for medical image denoising. The results of this project seem to agree with their findings. Depending on the situation, one might prefer to use a DAE since most of the computational cost comes in advance. This is useful because the parameters of the denoising autoencoder can be trained once, and then used to denoise an arbitrary amount of images. The drawback of this model is the prerequisite of having a data set that can be used to train the parameters. It could be impractical to acquire DTI data without any noise at all. Since the signal to noise ratio might depend on the MRI scanner hardware [18], a possibly more realistic scenario would be using a set of DTI data with small amounts of noise, and training the autoencoder to reduce noise, instead of trying to remove it completely. Whether this would result in good noise reduction needs further investigation. Another possibility for future work could be exploring more realistic ways of generating synthetic DTI data, and using this to train a denoising autoencoder. An alternative way of acquiring uncorrupted DTI data could be to use total variation denoising to preprocess DTI data, and use the denoised results as a starting point for training the DAE. Finally, it might be possible to improve the performance of denoising autoencoders by taking advantage of the manifold description of $\mathrm{Sym}^+(3)$, as was done in [2] and [3].

In this report we gave a brief description of the simplest form of a denoising autoencoder. We considered total variation denoising as an alternative approach for denoising images, and used it in our experiments for comparison. By creating a computer program, we applied both techniques to synthetically generated DTI data, and compared the results. Finally, it was demonstrated that our computer program could easily be made to work with real world DTI data.

Supplementary material

The computer code for this project is available at https://github.com/awiik/bachelor-project-dti-denoising.

References

[1] L. J. O'Donnell and C. F. Westin, "An introduction to diffusion tensor image analysis," *Neurosurg Clin N Am*, vol. 22, no. 2, pp. 185–196, Apr. 2011.

[2] A. Weinmann, L. Demaret, and M. Storath, "Total variation regularization for manifold-valued data," *SIAM Journal on Imaging Sciences*, vol. 7, no. 4, pp. 2226–2257, 2014. DOI: 10.1137/130951075. eprint: https://doi.org/10.1137/130951075. [Online]. Available: https://doi.org/10.1137/130951075.

[3] E. Celledoni, S. Eidnes, B. Owren, and T. Ringholm, "Dissipative schemes on riemannian manifolds," Apr. 2018.

[4] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of Machine Learning Research*, vol. 11, no. 110, pp. 3371–3408, 2010. [Online]. Available: http://jmlr.org/papers/v11/vincent10a.html.

[5] K. Cho, *Boltzmann machines and denoising autoencoders for image denoising*, 2013. arXiv: 1301.3468 [stat.ML].

[6] J. Xie, L. Xu, and E. Chen, "Image denoising and inpainting with deep neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., vol. 25, Curran Associates, Inc., 2012, pp. 341–349. [Online]. Available: https://proceedings.neurips.cc/paper/2012/file/6cdd60ea0045eb7a6ec44c54d29ed402-Paper.pdf.

[7] L. Gondara, "Medical image denoising using convolutional denoising autoencoders," *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, Dec. 2016. DOI: 10.1109/icdmw.2016.0041. [Online]. Available: http://dx.doi.org/10.1109/ICDMW.2016.0041.

[8] Y. Liu and Y. Zhang, "Low-dose ct restoration via stacked sparse denoising autoencoders," *Neurocomputing*, vol. 284, pp. 80–89, 2018, ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2018.01.015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925231218300316.

[9] D. Le Bihan, J.-F. Mangin, C. Poupon, C. A. Clark, S. Pappata, N. Molko, and H. Chabriat, "Diffusion tensor imaging: Concepts and applications," *Journal of Magnetic Resonance Imaging*, vol. 13, no. 4, pp. 534–546, 2001. DOI: https://doi.org/10.1002/jmri.1076. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/jmri.1076. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/jmri.1076.

[10] H. Gudbjartsson and S. Patz, "The Rician Distribution of Noisy MRI Data," *Magn Reson Med*, vol. 34, pp. 910–914, 1995.

[11] S. Pajevic and C. Pierpaoli, "Color schemes to represent the orientation of anisotropic tissues from diffusion tensor data: application to white matter fiber tract mapping in the human brain," *Magn Reson Med*, vol. 42, no. 3, pp. 526–540, Sep. 1999.

[12] E. Garyfallidis, M. Brett, B. Amirbekian, A. Rokem, S. Van Der Walt, M. Descoteaux, and I. Nimmo-Smith, "Dipy, a library for the analysis of diffusion mri data," *Frontiers in Neuroinformatics*, vol. 8, p. 8, 2014, ISSN: 1662-5196. DOI: 10.3389/fninf.2014.00008. [Online]. Available: https://www.frontiersin.org/article/10.3389/fninf.2014.00008.

[13] R. A. Horn and C. R. Johnson, *Matrix Analysis*. Cambridge University Press, 2013, p. 439, ISBN: 978-0-521-54823-6.

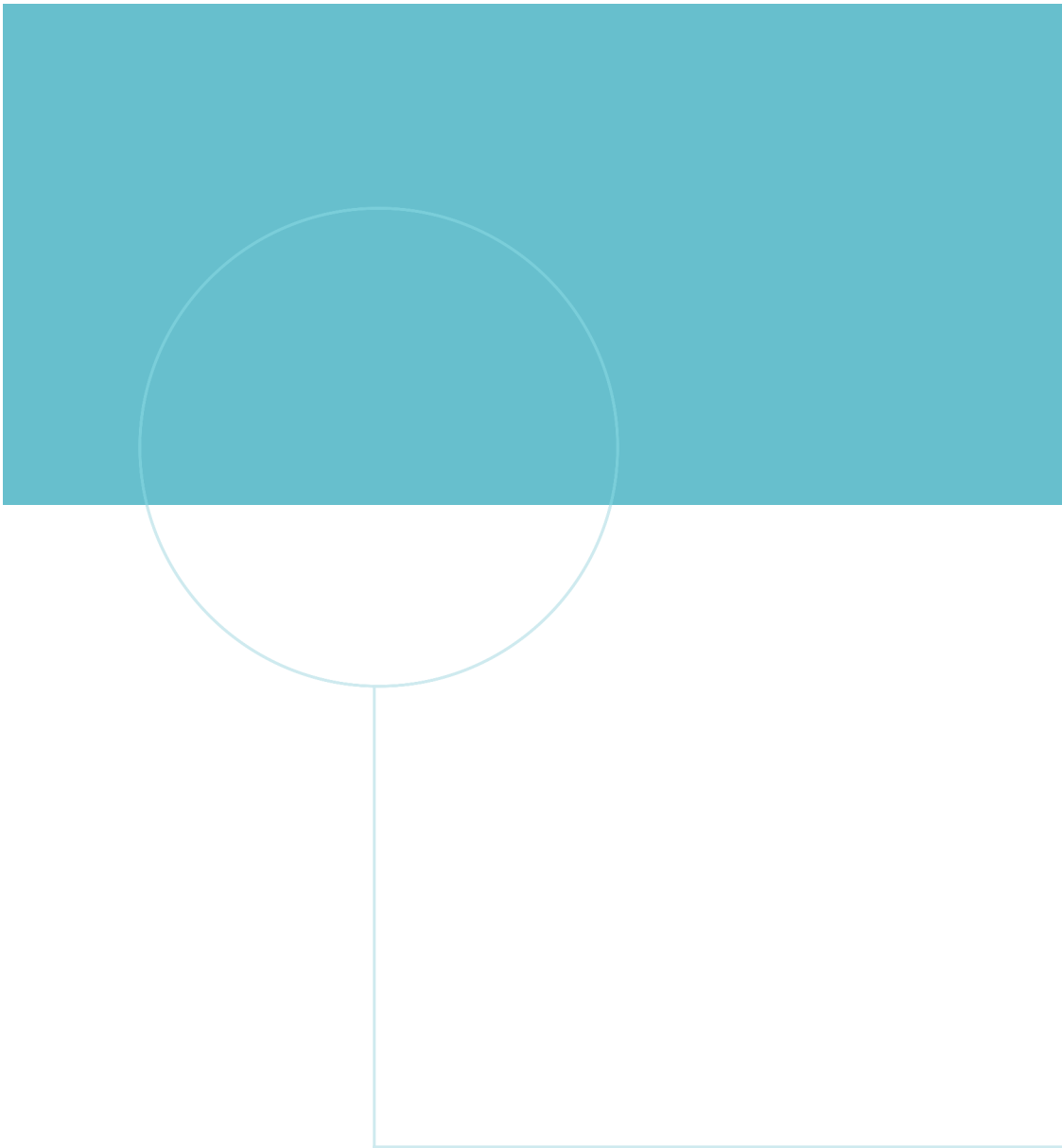
[14] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: https://www.tensorflow.org/.

[15] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Trans Image Process*, vol. 13, no. 4, pp. 600–612, Apr. 2004.

[16] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017. arXiv: 1412.6980 [cs.LG].

[17] D. C. Van Essen, K. Ugurbil, E. Auerbach, D. Barch, T. E. Behrens, R. Bucholz, A. Chang, L. Chen, M. Corbetta, S. W. Curtiss, S. Della Penna, D. Feinberg, M. F. Glasser, N. Harel, A. C. Heath, L. Larson-Prior, D. Marcus, G. Michalareas, S. Moeller, R. Oostenveld, S. E. Petersen, F. Prior, B. L. Schlaggar, S. M. Smith, A. Z. Snyder, J. Xu, and E. Yacoub, "The Human Connectome Project: a data acquisition perspective," *Neuroimage*, vol. 62, no. 4, pp. 2222–2231, Oct. 2012.

[18] D. L. Polders, A. Leemans, J. Hendrikse, M. J. Donahue, P. R. Luijten, and J. M. Hoogduin, "Signal to noise ratio and uncertainty in diffusion tensor imaging at 1.5, 3.0, and 7.0 Tesla," *J Magn Reson Imaging*, vol. 33, no. 6, pp. 1456–1463, Jun. 2011.



NTNU

Norwegian University of
Science and Technology