Imperial College
London

Final Report for MSc Project

Machine learning based simultaneous reconstruction of spin-echo and stimulated echo diffusion cardiac MRI output to shorten the acquisition time

Author name(s):

Virginia Fernandez

Supervisor(s):

Andrew D. Scott

Submitted in partial fulfilment of the requirements for the award of MSc in Biomedical Engineering from Imperial College London

Word count: 5966

Table of contents

Abstract

Diffusion Tensor Cardiac Magnetic Resonance (DT-CMR) is a standalone tool to assess the heart microstructure *in vivo* and provide new insights into pathologies like hypertrophic cardiomyopathy (HCM). Two sequences, motion-compensated spin-echo (MC-SE) and Stimulated Echoes (STEAM) have been used in DT-CMR. Each of them is preferable under certain conditions and both potentially convey useful and independent information. Although it would be desirable to acquire both sequences in many patients this is clinically unfeasible as scans are long due to their low SNR which means that many averages must be acquired.

Therefore, we aim to reduce the number of averages required per sequence by denoising MC-SE and STEAM images with a U-NET-based neural network. We also hypothesized that there is common information between images from the two sequences, so we implemented a 2-sequence input network we called Y-NET. We trained both networks on simulated Gaussian noise, then real data from healthy volunteers (N=15) and HCM patients (N=11). We compared our results with those of an optimized Non-Local-Means (NLM) algorithm.

All three algorithms removed both Gaussian noise and real noise from our data. Based on a ground truth formed by averaging 4 images, both networks yielded a higher Structural Similarity index (SSIM) increase than NLM. However, NLM outperformed them in the estimation of DT-CMR-derived parameters characterizing the cardiac microstructure.

Although further refinement and validation are required, we have developed the first multi-sequence denoising neural network in DT-CMR, moving towards fewer averages and increased clinical applicability.

Acknowledgements

I thank my supervisor, Dr Andrew Scott, for his continuous support and guidance during this project. I also thank the members of the Cardiovascular Imaging group of the Royal Brompton Hospital, in particular, Dr Pedro Ferreira, Dr Guang Yang and PhD student Malte Roehl for their occasional, yet key assistance and valuable inputs.

Introduction

Diffusion Weighted Imaging (DWI) is a Magnetic Resonance technique that assesses the diffusion of water within a tissue, which depends on its microstructure [18]. The image is obtained with two symmetric diffusion encoding gradients added to the MR sequence. The second gradient cancels the effect of the first in a stationary medium, but water molecules in motion attenuate the signal. This can be modelled with a decaying exponential (equation 1), where S_0 is the original signal, D is the diffusion coefficient (units: mm^2/s), and b is the b-value, a parameter that depends on the diffusion gradients (equation 2) and is usually 0 for S_0 (b_0). However, a non-null reference b-value can be used to suppress the signal from blood flow.

$$S = S_0 e^{-bD} \quad (1)$$

$$b = (\gamma G \delta)^2 \left(\Delta - \frac{\delta}{3}\right) \quad (2)$$

δ and G: duration and magnitude of the gradient; γ:gyromagnetic ratio (42.57 MHz/T); and Δ: the time between gradients.

Equation 1 represents a scenario of isotropic diffusion, derived from the assumption that diffusion within the tissue follows a Gaussian pattern [40]. DWI fails to show anisotropy, although many body tissues, like the heart, are anisotropic. Diffusion Tensor Imaging (DTI) is a DWI technique that measures diffusion along multiple directions, accounting for anisotropy [6][18]. Instead of being a scalar, D is modelled by a symmetric tensor with different diffusion values in each encoding direction. Equation 4 shows a sample tensor with 6 encoding directions.

$$\mathbf{D} = \begin{pmatrix} D_{11} & D_{12} & D_{13} \\ D_{21} & D_{22} & D_{23} \\ D_{31} & D_{32} & D_{33} \end{pmatrix} \quad (4)$$

In Cardiac Magnetic Resonance imaging (CMR), tensor **D** characterizes the microstructure of the heart [37] and has been recently applied to *in-vivo* imaging [7].

From the eigensystem of the tensor, some important parameters can be extracted. One is the helix angle (HA), which is the angle between the projection of the first eigenvector into the local wall tangential plane and the local circumferential direction [7]. Another example is the second eigenvalue, E2A (see Figure 1). The cardiomyocytes are arranged in groups known as sheetlets, which vary their orientation during the cardiac cycle and are responsible for systolic wall thickening. E2A measures sheetlet rotation and is deranged in patients suffering from Hypertrophic Cardiomyopathy (HCM) [7], an inherited disease that affects one every 500 people in the United Kingdom [3]. This derangement can only be assessed *in-vivo*, which is why it is important to ensure the success of CMR techniques by endowing them with efficient and clinically usable tools.

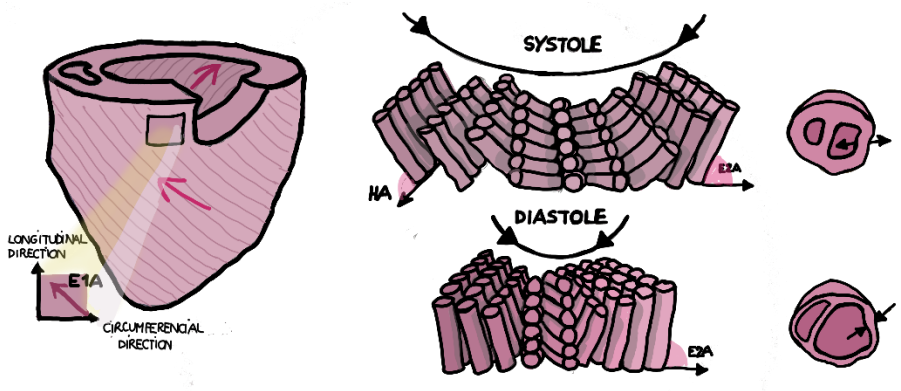


Figure 1. Schematic representation of the myocardium, and the orientation of the cardiomyocytes in the direction of E1A (left), and the sheetlets reorientating during the cardiac cycle in the direction of E2A (right) [7]

A key point in DT-CMR is the MR sequence used to acquire the images. During the respiratory cycle, the heart is affected by a considerable bulk motion, and the shape change of the myocardium during the cardiac cycle constitutes an additional artefact. The classical diffusion MRI sequence is represented in Figure 2. A high b-value is desirable to yield more sensitivity to diffusion, but the longer the diffusion gradient δ, the more sensitive to motion the signal is. Motion-Compensated Spin Echo sequence (MC-SE, Figure 5a) adds 2nd order motion compensation gradients to the sequence represented in Figure 2, which avoid signal loss artefacts when the imaged tissue is moving with constant acceleration [29].

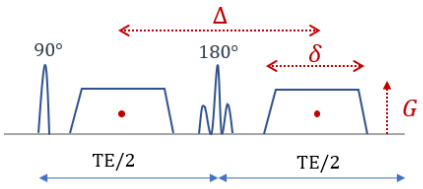


Figure 2. SE diffusion gradient representation. Both diffusion gradients of length δ and amplitude G are spaced by Δ [6].

Whereas MC-SE runs over one cardiac cycle, Stimulated Echo Acquisition Mode (STEAM) is acquired along two cycles and relies on the application of three 90° pulses [27][24][5]: the first two 90° pulses "store" the magnetization in the longitudinal direction during a time T_M (mixing time). The diffusion gradients take place after the first and the third 90° pulses (see Figure 5b) and allow for a long Δ interval. As a result, δ can be short without lowering the b-value, making this sequence more insensitive to motion.

In a previous study, both sequences were compared for a series of healthy volunteers [29]. STEAM is more reliable in diastolic acquisitions, whereas MC-SE outperforms STEAM in free-breathing or arrhythmia studies [29]. Furthermore, both MC-SE and STEAM sequences provide complementary information [6].

However, although it would be desirable to acquire both sequences within the same study, there are substantial downsides to it. First, acquiring both sequences takes a significant amount of time, since multiple averages must be recorded for the same encoding direction to yield an acceptable Signal-to-Noise Ratio (SNR). In addition, to prevent the introduction of breathing artefacts into the signal, the subject must hold his or her breath during an image acquisition, causing patient discomfort. The use of effective denoising methods is therefore a key point to make DT-CMR clinically applicable, as they would help reducing the acquisition time by decreasing the number of necessary averages per sequence.

Generative Adversarial Networks (GAN) and Convolutional Neural Networks (CNN) have been previously used for image denoising [13]. U-NET, a fully convolutional network [28] has been widely used in medical imaging to solve diverse problems such as segmentation, image reconstruction and denoising [39][20][28]. Behind its success are its ability to be trained on small datasets and the use of upsampling operators instead of pooling, which increases the output resolution [28][6].

In this project, we aim to reduce the acquisition time of SE and STEAM sequences with a denoising neural network. This can be done in three ways as shown in Figure 3.

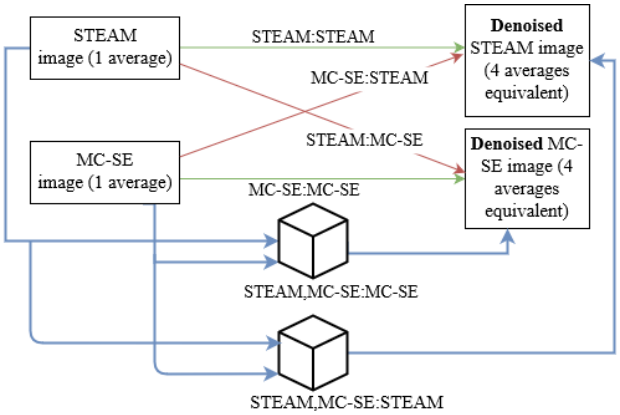


Figure 3. Diagram showing the possible Network configurations: a cross-sequence denoising network (red), a within-sequence denoising network (green) and a 2-input denoising network (blue).

In a first approach, we will deploy a within-sequence neural network. However, as this first approach disregards the fact that both sequences hold common information [6], we will then focus on the STEAM, SE: STEAM and SE, STEAM:SE approach, by implementing a 2-input neural network.

Methods

The diagram in Figure 4 summarizes the steps undertaken in this project.

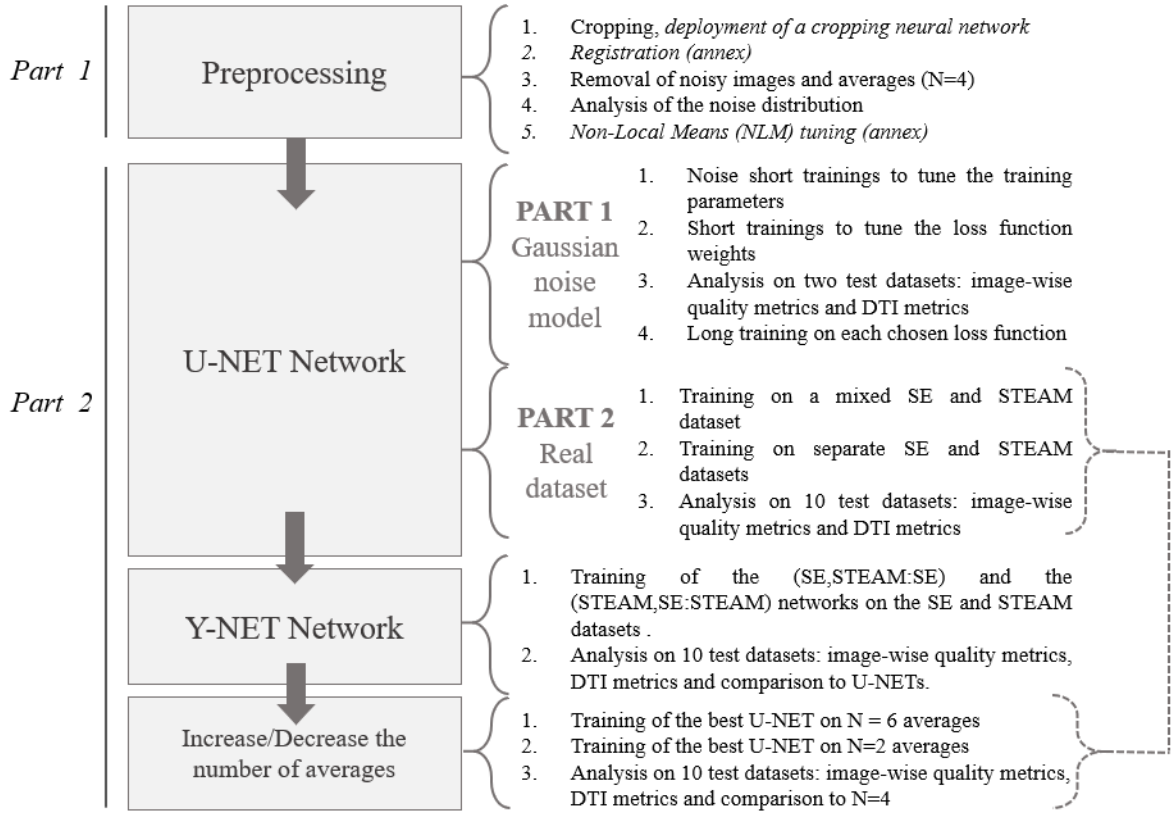


Figure 4. Steps followed in this project. Parts 1 and 2 refer to the respective "Methods" sections.

1. Preprocessing

a. Description of the dataset

The dataset consists of anonymized data from SE and STEAM sequences acquired in 15 healthy subjects in diastole, sweet-spot [29] and systole, and 11 HCM patients [7], acquired in diastole and systole. All the subjects gave written informed consent to be part of a study to develop new CMR methods. The study has been approved by the Research Ethics Committee (approval references 10/H0701/112 and 13/LO/1830) [6].

For each {subject, phase and sequence} sub-dataset, we have approximately 10 (STEAM) and 18 (SE) averages acquired in 6 encoding directions (see annex table A1) and the reference (Figure 5c).

Echo-Planar-Imaging (EPI) was used to acquire a whole image in one readout (single-shot imaging).

For each sub-dataset, a bounding box around the myocardium was defined by experts, and excessively noisy images or those containing too severe artefacts due to breathing or failed cardiac triggering were removed [29].

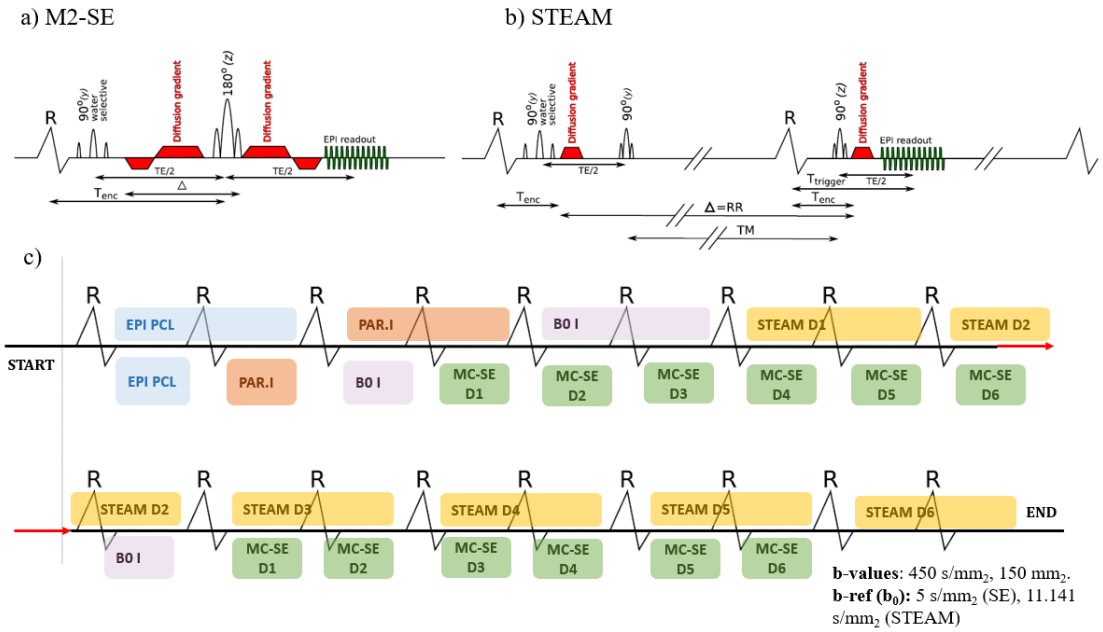


Figure 5. a) MC-SE sequence as a function of time; b) STEAM sequence as a function of time; T_{enc} = Time from R-wave to effective diffusion encoding; $T_{trigger}$ = time from R-wave to central K-space line; T_E = echo time; T_M = mixing time. c) diagram illustrating the acquisition of a {subject, phase} dataset as a function of the echocardiogram R-waves. Yellow/Green squares: 6 direction-encoded acquisitions (D_i) for STEAM/SE; purple squares: reference image "b0" aquisition; EPI PCL = echo planar imaging phase correction lines; PAR.I = Parallel imaging calibration.

b. Cropping

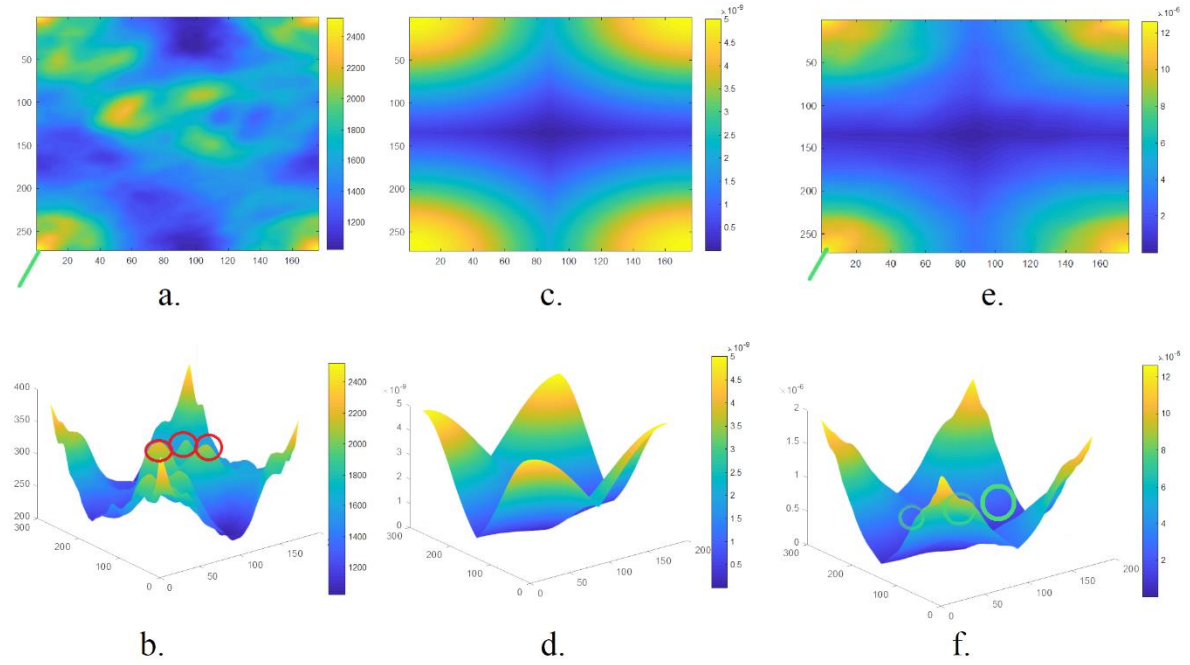
To avoid the network from being trained on elements different from the heart, such as the chest wall, the images were cropped using the 120x88 pixels bounding boxes manually defined and centred around the myocardium.

We compared the results from the manual segmentation with the results of an automatic segmentation algorithm based on the "*You Only Look Once"* YOLO network for object detection [25], which has proven to be very effective in object recognition (see annex C1).

c. Registration

Before the generation of our ground truth images, and as a mandatory step in DTI, a registration of the images within subjects and phases was applied. Because non-rigid rotation implied a modification of the diffusion encoding reference system that would alter the tensor results, we used a rigid registration based on upsampled discrete Fourier Transform (DFT) [29][9]. The DFTs of each image and the reference were multiplied, resulting in a 2-dimensional Fourier Space (Figure 6a). Because the chest wall generated a problematic high signal in the DFT space (Figure 6a), we modified the existing algorithm [9] by multiplying the DFT by a 2-dimensional Gaussian function (Figures 6b and 6d) that attenuated the distant peaks and preserved those that were close to the origin and corresponded to a correct registration.

Within each {subject, phase} pair, we attempted a within-sequence registration, and then used these registered images to perform a cross-sequence registration.



Notice that the change of scale does not affect the calculation of the maximum values.

Figure 6. a) Colormap of the product of the DFTs of the image and the reference of a healthy volunteer dataset. The green arrow points to the correct shift-peak; b) 3-d surface plot of the same DFT product, with the problematic peaks outlined in red; c) colormap of the 2-D gaussian function used; d) 3-d surface plot of the of the same gaussian function; c) result from multiplying a) and c), with the correct shift still at its place; f) 3-d surface plot of the same DFT product, with the previously problematic peaks of Figure b represented by the green circles.

d. Ground truth and noise analysis

We trained our networks to denoise a single image with an average of N images as a target. We called this the denoising ratio 1:N. The averaged images come from the same patient, phase, sequence, direction and b-value. We evaluated 1:4, and then 1:6 and 1:2. The SNR should be increased by $\sqrt{N}$ fold, thus 2, 2.4 and 1.7 respectively [32].

MRI noise is assumed to follow a Rician distribution [10], but over a certain SNR ratio, Gaussian noise is often assumed. To verify is this assumption could be done, we studied the noise profile of our DT-CMR data by selecting a random subset of SE and STEAM images (see annex A). We extracted the residuals R (R= Ground truth – Noisy Image) and plotted their joint histogram. Figure 7a) shows the global distribution of the noise along all phases and sequences. Both the global and sequence-specific histograms (Figures 7b and c) are comparable to that of a Gaussian distribution. Therefore, we adopted Gaussian noise (with a mean standard deviation σ of 0.1) as a model to validate our data. However, we are aware of its limitations: the standard deviations vary between sequence, and the tails are slightly less heavy in the real data histogram than in the Gaussian curve.

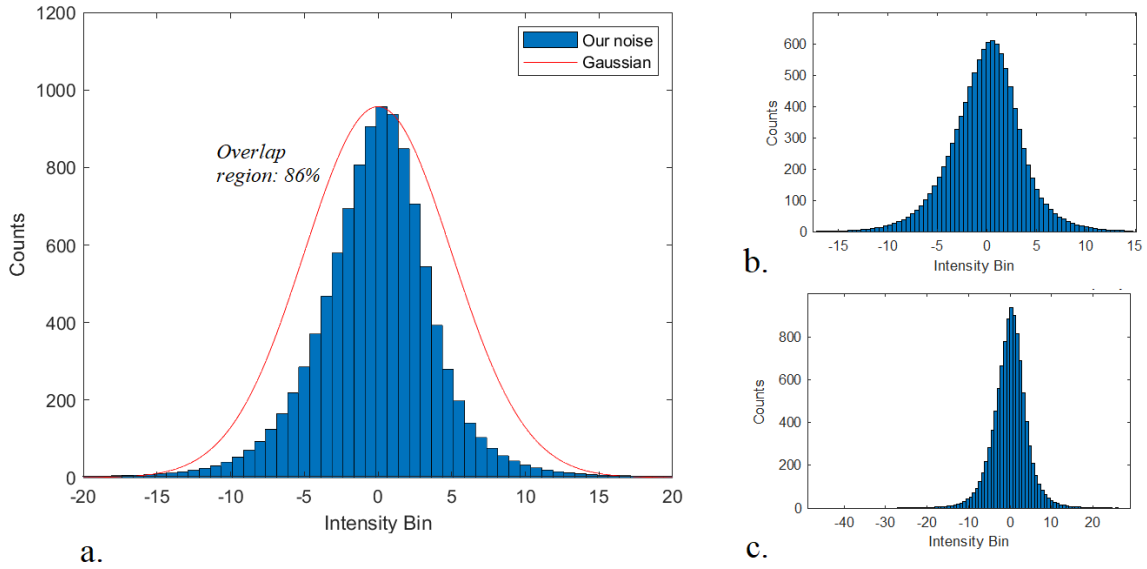


Figure 7. a) Histogram of the noise distribution for a subset of 600 SE and STEAM images. The red line shows a Gaussian distribution multiplied by the maximum of our histogram; b) Histogram of the noise distribution within a subset of randomly selected STEAM images (all phases); c) Histogram of the noise distribution within a subset of randomly selected SE images (all phases).

2. Denoising neural Networks deployment and training

a. Networks Structure

In the first stage, we used the original U-NET structure [28], adjusting the padding so that the concatenation layers had the same dimension. The structure of the network can be seen in Figure 8.

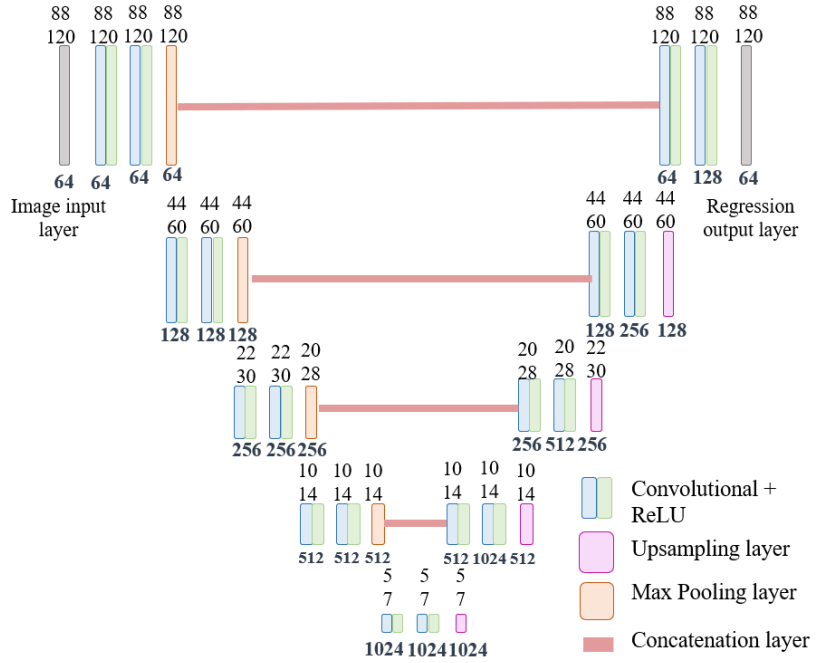


Figure 8. U-Net structure inspired from [28]. The numbers above each layer are the input size (X,Y) and the numbers below are the number of channels.

Our t-input neural network is based on U-NET, but we added a second down sampling branch that extracted features from the other sequence (see Figure 9). We named the resulting network "Y-NET". The pre-set weights for each down sampling branch were retrieved from those form each previously trained U-NET.

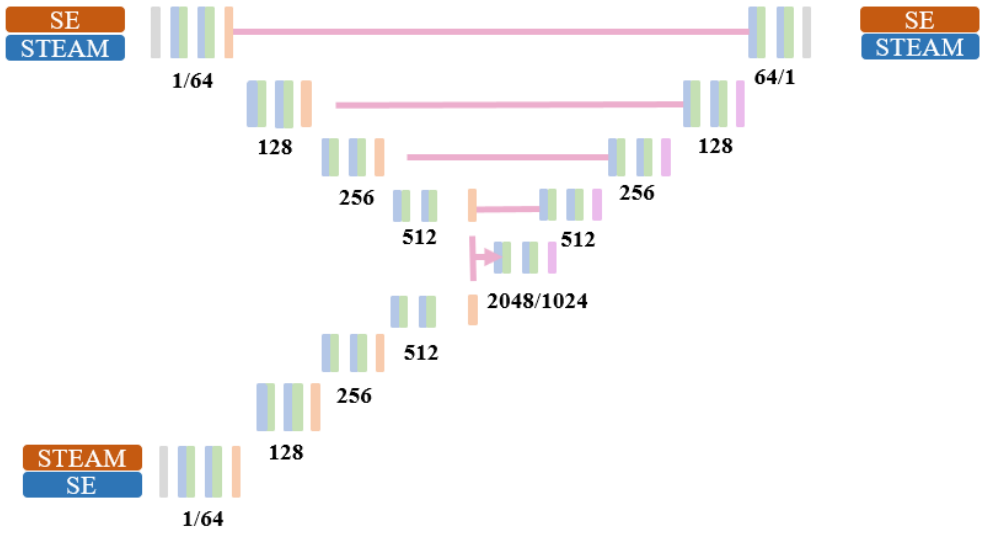


Figure 9. Y-NET structure. The numbers correspond to channels of the layer. The layers and input sizes are the sames as in Figure 8. The orange squares at the beginning and end of the network are the inputs/output of the SE,STEAM:SE Y-NET, and the blue ones are those of the STEAM,SE:STEAM Y-NET.

b. Metrics and assessment

We compared our network results with those of the Non-Local Means (NLM) algorithm, commonly used in image denoising [39], as it preserves the texture and edges better than classical denoising methods [4]. NLM calculates the intensity value of a pixel by computing a weighted average of its surrounding pixels within a search window, giving higher weights to pixels of the image that have a similar intensity surrounding (equation 5).

$$i(\mathrm{p}) = \frac{1}{\mathrm{C(p)}} \int_{\Omega} \mathrm{i(q)w(p,q)dq} \quad (5)$$

Where q are p's the neighbouring pixels, i(p) and i(q) the intensity of pixels p and q, C(p) is a normalization factor, Ω is the search window, , and w(p,q) is the Gaussian weighting function, based on the similarity between the neighbouring regions of p and q [4]

MATLAB NLM function *imnlmfilt* was optimized for each denoising ratio to get the best performance from NLM (see annex B for more details).

For each test image, we calculated the mean absolute error (MAE), the root mean square error (RMSE) and the structural similarity index (SSIM, equation 6) [39] between the output image and the ground truth denoised image.

$$\mathrm{SSIM} = \frac{(2\mu_t\mu_y + C_1)(2\sigma_{ty} + C_2)}{(\mu_t^2 + \mu_y^2 + C_1)(\sigma_t^2 + \sigma_y^2 + C_2)} \quad (6)$$

Where μ_t and σ_t and μ_y and σ_y are the mean and standard deviation of the target image and the output image respectively, and C_1 (2.55) and C_2 (7.65) are stability constants [35].

Additionally, we processed each test dataset in the Diffusion Analysis Tool [7] (see annex A) and retrieved key DT-CMR values: the mean helix angle (HA) of the epi and endocardium, the fractional anisotropy (FA, equation 7), the mean diffusivity (MD, equation 8), the transmural helix angle line profile (Lp) Gradient [7], the median E2A value, and the percentage of negative eigenvalues. A small percentage of negative eigenvalues translates into higher image quality. MD, FA, E2A and HA were compared with those of the ground-truth dataset.

$$\mathrm{FA} = \sqrt{\frac{3}{2}}\frac{\sqrt{(\lambda_1 - \bar{\lambda})^2 + (\lambda_2 - \bar{\lambda})^2 + (\lambda_3 - \bar{\lambda})^2}}{\sqrt{\lambda_1{}^2 + \lambda_2{}^2 + \lambda_3{}^2}} \quad (7)$$

$$\mathrm{MD} = \frac{\lambda_1 + \lambda_2 + \lambda_3}{3} \quad (8)$$

Where λ_i are the eigenvalues of the diffusion tensor, $\bar{\lambda}$ is their mean. A FA value of 0 corresponds to a sphere-shaped diffusion pattern, whereas a value close to 1 indicates a preferred direction.

To create the test datasets, two healthy volunteer and patient datasets were set apart and not used for the training or validation of the networks (see annex A2). The STEAM systolic sequence was ruled out for patient 2 because it lacked the minimum number of images required to produce the ground truth dataset.

c. Validation with Gaussian Noise

In section 1a, we assumed that our noise could be modelled with a Gaussian distribution with standard deviation $\sigma \approx 0.1$. As the STEAM dataset tended to have a lower σ, we chose to generate a Gaussian-noisy dataset with a σ varying between 0.1 and 0.001. The noise was added to our ground-truth images, creating a predictable noisy dataset that we used to train the U-NETs initially. We used the resulting weights and biases to re-train the U-NETs on our dataset, to optimize the training starting point.

Before performing the long training on the whole Gaussian dataset (11875 images), we performed 5-epochs trainings varying different training parameters. Based on the results detailed in annex D, we selected the following:

- SDGM optimizer.
- Mini-batch size of 15
- Learning rate of 0.1 (0.05 for the Y-NET).
- Drop factor of 0.1 every 5 epochs.

Even when training on residuals (difference between target and input), the results were always better leaving the last concatenation layer (see annex figure D1).

To avoid the blur caused by l_2 norm [39][34], we implemented a loss function L that combined l_1 norm (MAE) and the simplified implementation of a SSIM-based loss function developed in [39] (see annex D2) for further information) in a weighted sum (equation 9).

$$\mathrm{L} = \mathrm{W_{SSIM}} \times \mathrm{SSIM} + \mathrm{W_{MAE}} \times \mathrm{MAE} \quad (9)$$

We performed 5-epochs trainings varying the weights W_{SSIM} and W_{MAE} and calculated the metrics advanced in section 4b. We named the resulting networks using the nomenclature $SSIM_XMAE_X$ (where "X" are the weights). We trained one of the networks on residuals rather than images, for which we added the suffix "RES" to the name.

We performed a 40 epochs training on the same dataset with the optimum weights and training parameters. The resulting networks were then re-trained on the real data, using 90% of the images for training, and 10% for a 10-fold cross validation.

d. Platform and training

All the implementations were performed in MATLAB R2019a, using the Deep Learning Toolbox to generate the neural networks. We trained our networks on the Quadro P1000 available at Imperial CX1 Cluster, or the Quadro P6000 available at the Royal Brompton Hospital.

Results

Heart-detector

Although our classifier yielded a high accuracy (see annex figure C1), all the test on the YOLO resulted in empty bounding boxes. In annex C3, we analysed why the network could have failed and propose further research paths that could be followed in order to improve it. The manually delimited bounding-boxes of size 120x88 pixels were used to crop our images.

Registration

The results of the registration were quantified by taking the qualified as worst (patient, phase) and (healthy volunteer, phase) and computing the average distance between the coordinates of a landmark point labelled in all images, and its mean position across all images. The landmark point was chosen by visually seeing which structure identification was more reproducible: in the healthy subject, it was the top of the myocardium, and in the patient, it was the center of the blood pool.

The distribution of the distances (in pixels) is reflected in Figure 10. The mean distances between each SE and STEAM dataset differ by 1.87 pixels for the healthy subject, and 0.70 pixels for the patient.

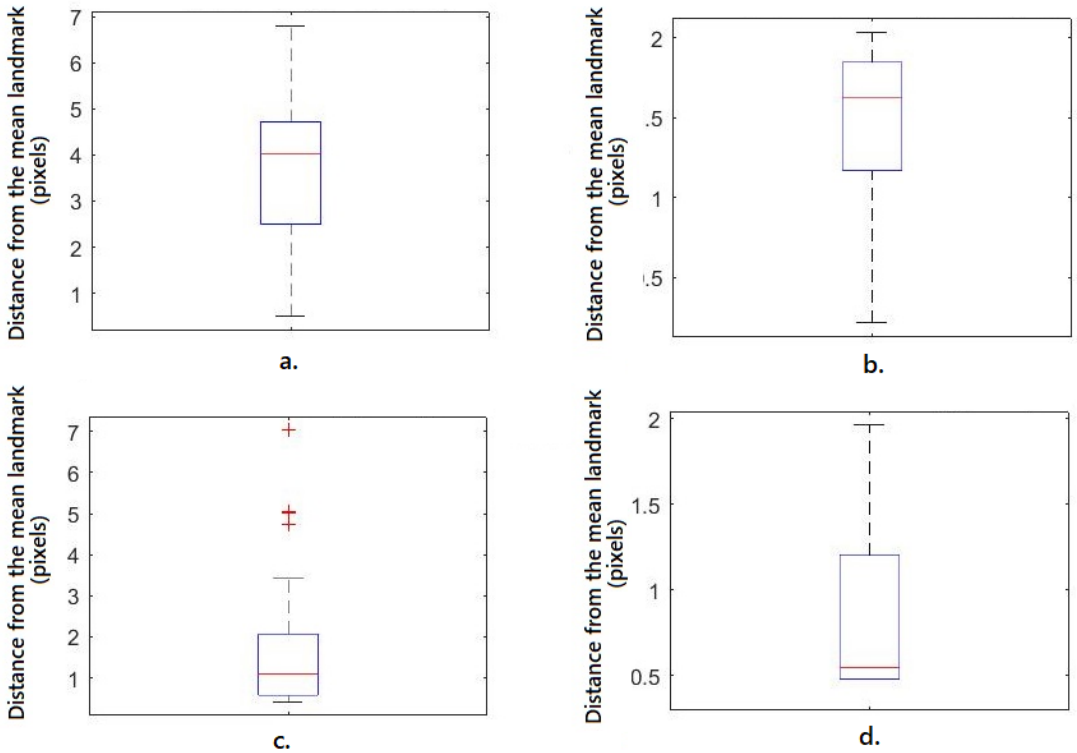


Figure 10. Top, box plots of the Euclidean distance distribution between images for healthy subject 5 within the SE dataset (a) and the STEAM dataset (b). Bottom, equivalent box plots for the SE (c) and STEAM (d) sequences for patient 7.

Denoising

Validation on Gaussian noise

The performance of the different loss functions was tested on a subject dataset for two phases. For the image-wise quality metrics advanced in Methods 4b), we compared each denoised figure with its equivalent noisy one by computing the relative difference percentage "% Diff", as in equation 10. These metrics can be found in annex table D3 and Figure 11. For the Diffusion Data Analysis tool advanced

in 4b as well, the relative difference percentage is between the ground truth value and the denoised value (equation 11). These values are collected in annex table D4 and represented in Figure 12.

$$\%\ \text{Diff} = \ 100\ \times \frac{F_{DENOISED} - F_{NOISY}}{F_{NOISY}}\text{, F: generic parameter (10)}$$

$$\%\ \text{Diff} = \ 100\ \times \left|\frac{F_{DENOISED} - F_{GROUND\ TRUTH}}{F_{GROUND\ TRUTH}}\right|\text{, F: generic parameter (11)}$$

Both the comparative images from Figure 13 and the values in Figure 12 show that the networks can remove the Gaussian Noise, some yielding better quality metrics than NLM, especially in the case of SSIM.

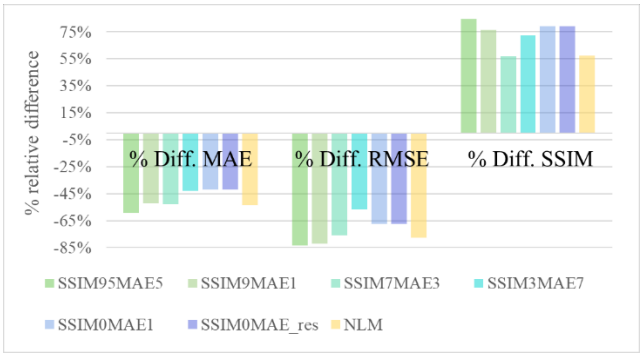


Figure 11. Relative differences of the MAE, RMSE and SSIM with regards to the noisy image for a series of SSIM and MAE weight configurations in the loss functions, and for the NLM algorithm.

The Diffusion Data Analysis Tool results (Figure 12) are more even. Yet, $SSIM_{95}MAE_5$, $SSIM_9MAE_1$, $SSIM_0MAE_1$ and $SSIM_0MAE_1RES$ outperformed NLM in more than 50% of the metrics. The best results are globally achieved by $SSIM_0MAE_1RES$.

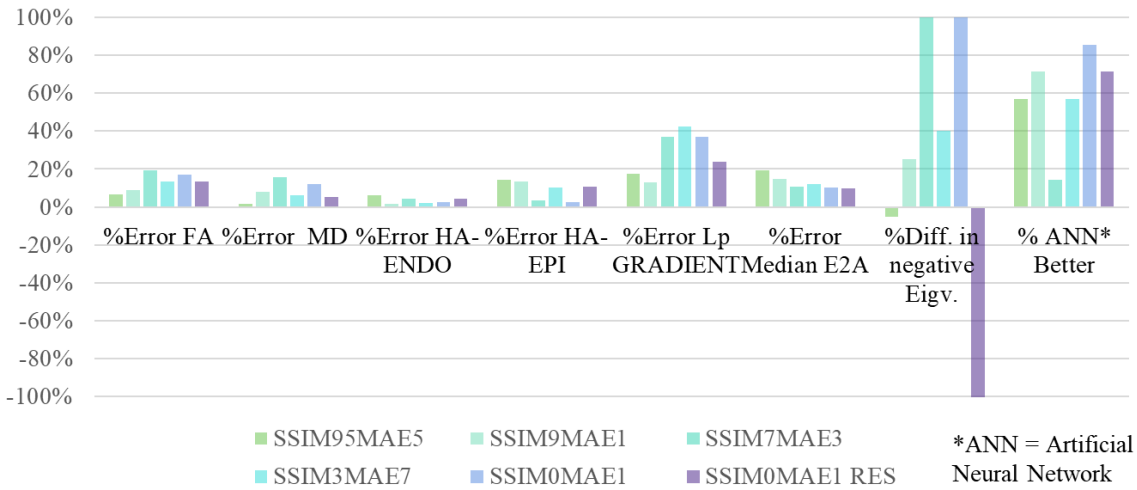


Figure 12. Mean relative differences for the DTI parameters obtained with different loss function weights and NLM. The column "% ANN Better" calculates the proportion of ANN outperforming NLM cases. Notice that a negative difference in the % of negative eigenvalues implies a reduction of that figure with regards to the ground truth, which is satisfactory. For the rest of the metrics, we calculated the absolute value and want them to be close to 0%; HA-ENDO: mean helix angle for the endocardium; HA-EPI: mean helix angle for the epicardium.

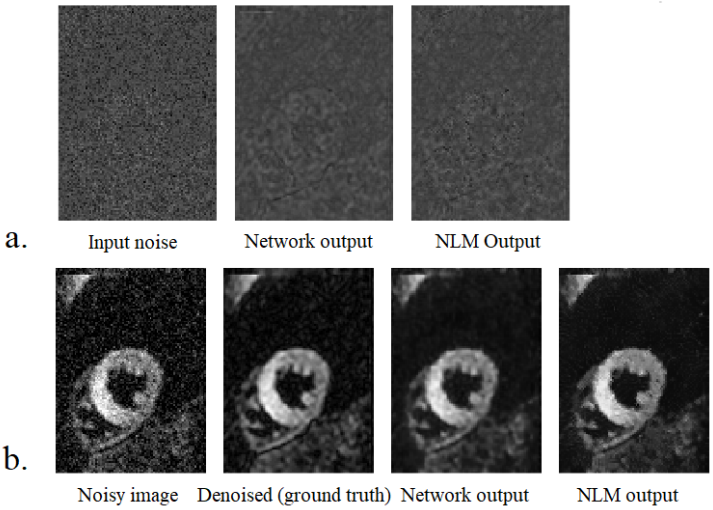


Figure 13: a) residuals map of the input Gaussian-noise, the output of the network and the NLM output; b) comparison between the input noisy image, the ground truth image and the output of the network and the NLM algorithm

Regarding the image-wise quality metrics, $SSIM_0MAE_1RES$ and $SSIM_{95}MAE_5$ gave the best results, followed by $SSIM_9MAE_1$ and $SSIM_0MAE_1$. The mean error in the DTI values was higher for $SSIM_0MAE_1$ than it was for the other three. Therefore, we decided to perform the trainings on the real data with $SSIM_{95}MAE_5$, $SSIM_9MAE_1$ and $SSIM_0MAE_1RES$.

Results on real dataset

We trained the pre-trained U-NETs on the real dataset, on which we performed data augmentation consisting on mild rotations, shearing and flipping. The dataset consisted of 5702 SE and 3090 STEAM images. We also performed trainings on separate SE and STEAM datasets.

The % differences for the image-wise metrics and the DTI ones (equations 10 and 11) are collected in annex tables D4 and D5. Both tables contain the output of the U-NETs trained on mixed and separate SE and STEAM datasets. Tables D6 and D7 are the equivalent of D4 and D5 for the Y-NET network.

The training on the mixed dataset resulted in unsatisfactory results, notably for the $SSIM_0MAE_1RES$ network, for which there was an increase on MAE and RMSE with regards to the noisy image, and a considerable reduction of SSIM (see Figure 14a). The results were better when the networks were trained on separate datasets (Figure 14b). Although NLM has the best loss of MAE and RMSE errors, the highest increase in SSIM (12%) is achieved by the $SSIM_0MAE_1RES$ U-NET.

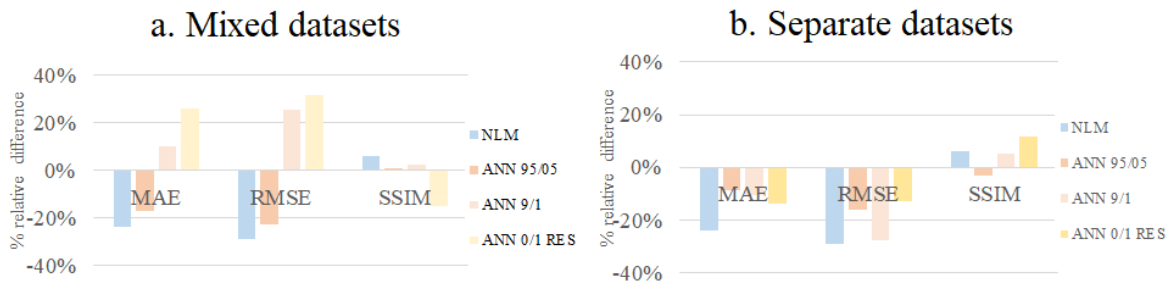


Figure 14. a) Relative difference of the MAE, RMSE, SSIM values between the noisy image and the output of the different U-NETs trained on the mixed dataset, and the NLM algorithm; b) Same values for the U-NETs trained on separate datasets.

We averaged the relative differences calculated for each parameter in the Diffusion Analysis Data Tool (Figure 15b), leaving apart the difference in the percentage of negative eigenvalues (Figure 15a),

due to its high values and variations along the different datasets. The U-NETs trained on mixed datasets lead to a higher increase in the % of negative eigenvalues, whereas all U-NETs trained on separate datasets do better than NLM in this aspect. When trained on the mixed dataset, $SSIM_{95}MAE_5$ and $SSIM_9MAE_1$ outperform NLM in 33 and 29% of the cases, whereas with the separate-datasets, the scores increase to 52 and 48% (see annex table D5). $SSIM_0MAE_1RES$ keeps a similar score in both cases.

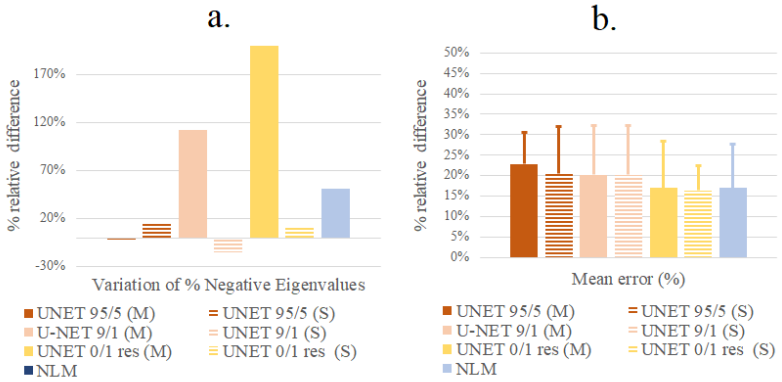


Figure 15. a) Difference in the % of Negative Eigenvalues achieved by the U-NETs trained on the mixed SE and STEAM dataset (M) and those trained on the separate datasets (S), and the NLM algorithm; b) Variation of the mean relative difference of the U-NETs and NLM in pooled for all DTI parameters. The needles over each bar represent the standard deviations.

From our results, we can conclude that the $SSIM_0MAE_1RES$ U-NET has the best performance. Thus, the following results focus on that specific weight combination.

Annex tables D6 and D7 show the image quality metrics and DTI metrics results for all the Y-NETs. We compared the results between the $SSIM_0MAE_1RES$ U-NET and its equivalent Y-NET. Figure 16 shows the relative differences for both networks.

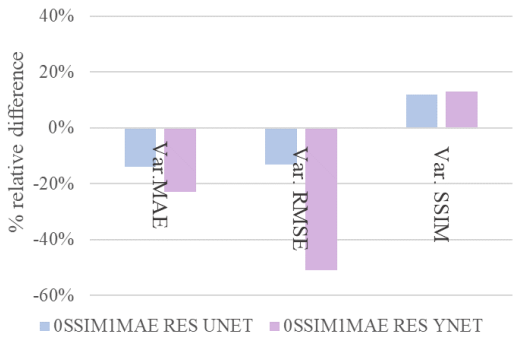


Figure 16. % relative differences in the image-wise quality metrics between the $SSIM_0MAE_1RES$ U-NET and the $SSIM_0MAE_1RES$ Y-NET.

Figure 17 compares the mean differences in the DTI metrics achieved by $SSIM_0MAE_1RES$ U-NET, Y-NET, and NLM. Depending on the category, U-NET outperforms or is outperformed by Y-NET. We tend to see a better performance of U-NET in what concerns the helix angle, but Y-NET manages to equal the performance of NLM for the mean diffusivity or the E2A median. Y-NET yields also the best reduction (29%) of the % of negative eigenvalues.

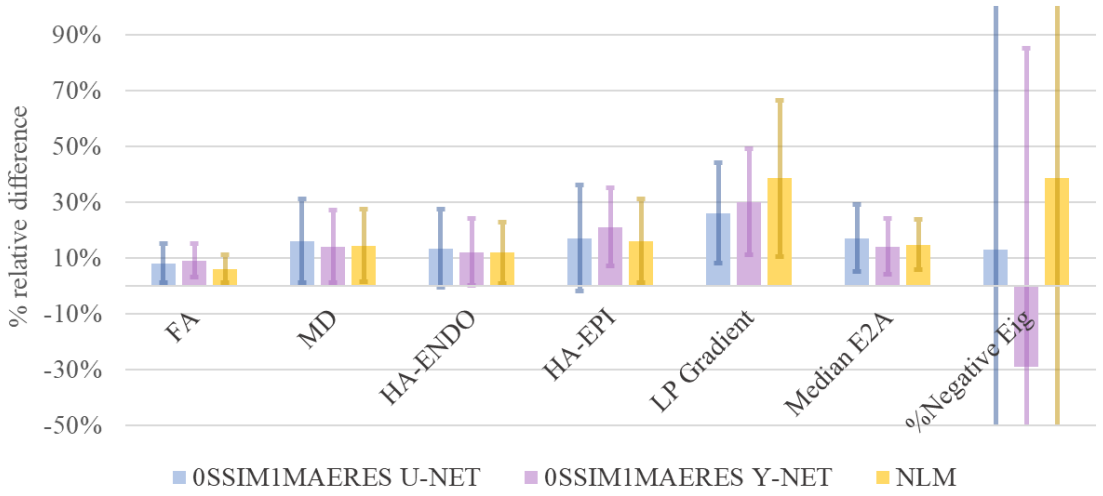


Figure 17. Mean relative differences of each calculated DTI parameter for the $SSIM_0MAE_1RES$ U-NET, Y-NET and the NLM algorithm. The standard deviations of the eigenvalues were too big to fit the graph. They take values of 210%, 150% and 114% for U-NET, NLM and Y-NET.

We also compared both networks $SSIM_0MAE_1RES$ U and Y-NET for each sequence (Figure 18a) and phase (Figure 18b, annex Figure D4). In systole, Y-NET achieves an 8% difference reduction over U-NET. The error is higher in Spin-Echo than STEAM for both networks and NLM. In every case, NLM outperforms both networks in every case, yielding substantially better results in STEAM over U-NET and Y-NET.

If we look at each category (Figure 17) NLM stays the best tool to predict both FA but is equalled in effectivity by Y-NET in the measurement of MD. On the other hand, U-NET is consistently the best to predict the HA and the Lp Gradient. Except for the Lp and the % of negative eigenvalues, NLM is the most robust for all the categories, having also a slightly lower variability than the networks.

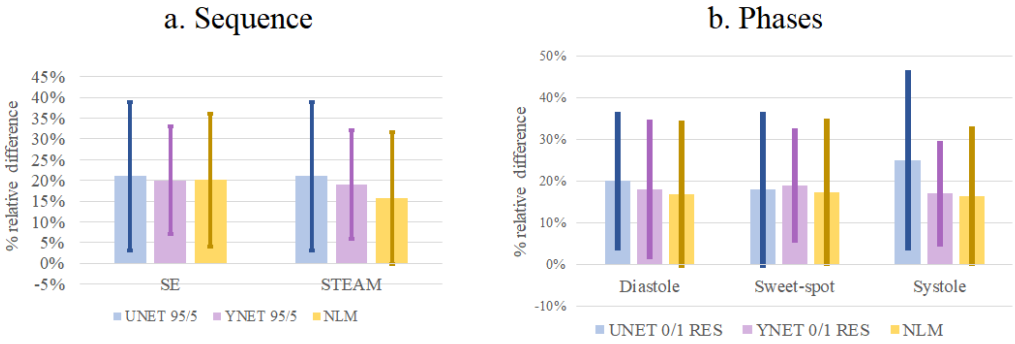


Figure 18. Comparison of the global DTI metrics error for the U-NET, Y-NET and NLM,f or each sequence and cardiac phase.

Figure 19 shows the HA, MD, FA and E2A (absolute value) maps for a SE systolic dataset. We can see how, whereas Y-NET achieves a better MD, FA and E2A than U-NET, the latter tends to be more effective than Y-NET when we tackle the HA measurement. The reason is that Y-NET causes a more aggressive blur, whereas both U-NET and NLM tend to preserve the angle variation more effectively (see yellow squares in Figure 19, upper row).

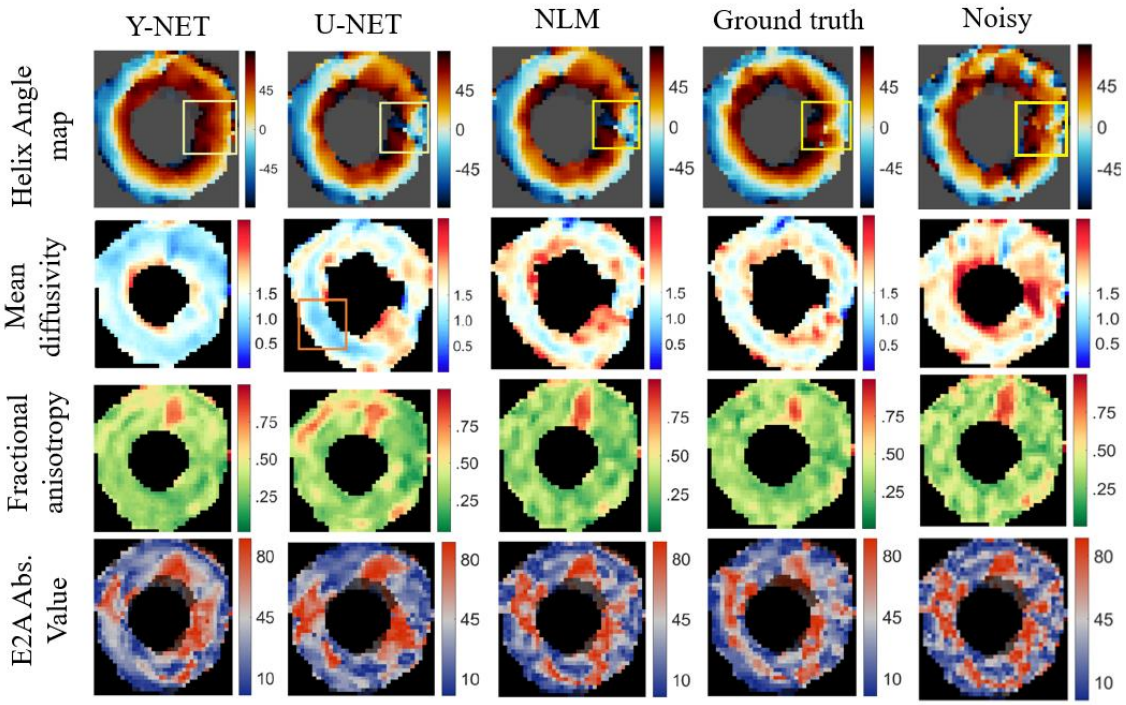


Figure 19. Parameter maps for a systolic healthy subject dataset. From top to bottom: HA map, the yellow squares point a region of interest where it is seen that U-NET outperforms Y-NET; MD map, the red square in U-NET points a region where the MD is severly underestimated; FA map; E2A absolute value map. From left to right: Y-NET, U-NET, NLM, ground truth and noisy dataset.

Increase and decrease of the number of averages

Until now, the denoising ratio (the number of averages that our network tries to aim with a single image) was 1:4. We trained 2 U-NETs on separate SE and STEAM datasets with ratios 1:6 and 1:2. The image-wise metrics are compared in Figure 20, whereas the DTI relative differences are represented in Figure 21. Although we tuned the NLM algorithm to adapt to 2 and 6 averages, its performance in the DTI metrics gets worse with the denoising ratio (annex figure D6). Notice that the SSIM gain decreases with the number of averages. For the DTI parameters, the error tends to be the bigger for the 1:6 ratio.

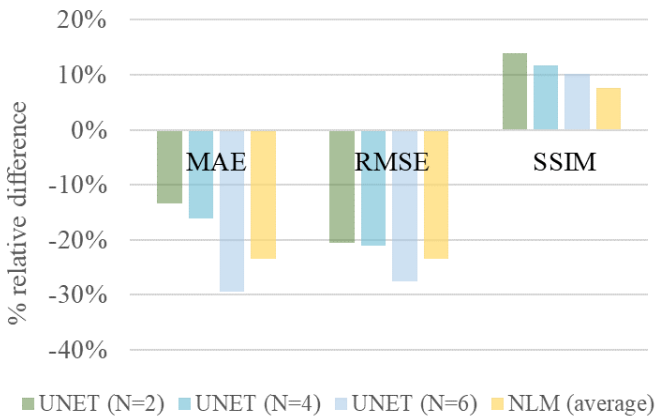


Figure 20. mean % of relative difference in image-quality metrics between the noisy and denoised image for ratios 1:1, 1:4 and 1:6, and the average difference for the three sets on NLM results.

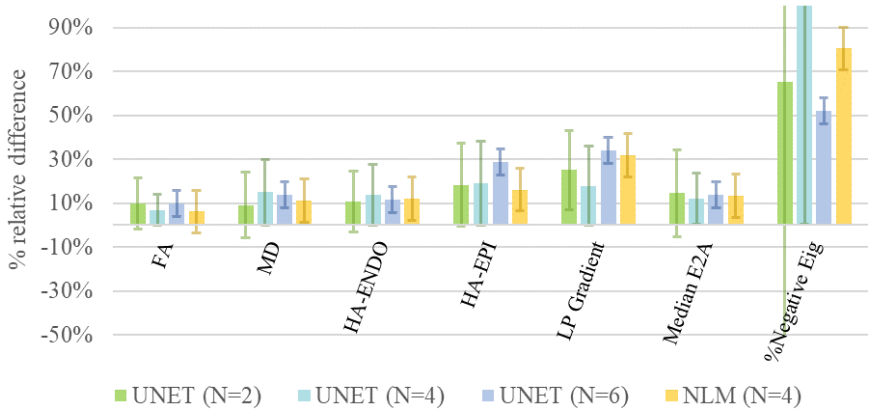


Figure 21. mean % of relative difference in the DTI values between the noisy and denoised image for ratios 1:1, 1:4 and 1:6, and the average difference for the three sets on NLM results.

Figure 22 compares the HA, MD, FA and E2A maps for the same dataset as in Figure 19, between the different denoising ratios. The best results are consistently those of N=2, followed by N=4.

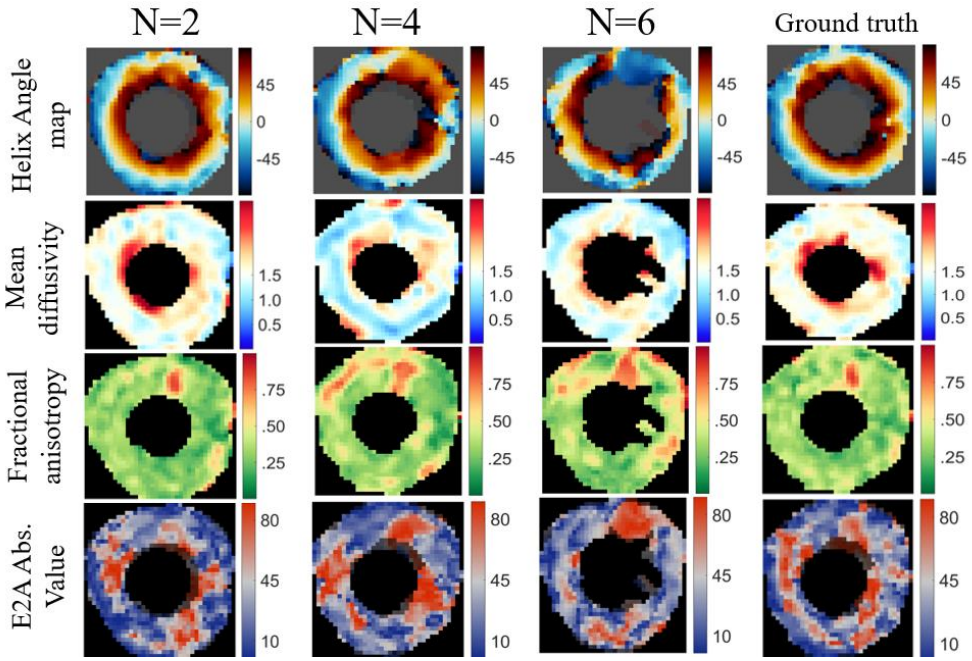


Figure 22. From top to bottom: HA map, MD map, FA map and E2A absolute value map. From left to right: 1:2 ratio, 1:4 ratio, 1:6 ratio and ground truth.

Convergence and performance

We did not notice a major difference in performance during our trainings. The average training time for 40 epochs was between 6 and 7 hours. Surprisingly, the Y-NETs, having 33% more parameters than the U-NETs, achieved a comparable training time. Figure 23 shows the training and validation RMSEs for our different trainings as a function of the iteration. The networks approach convergence at about 4000 iterations and stagnate around a RMSE between 300 and 500.

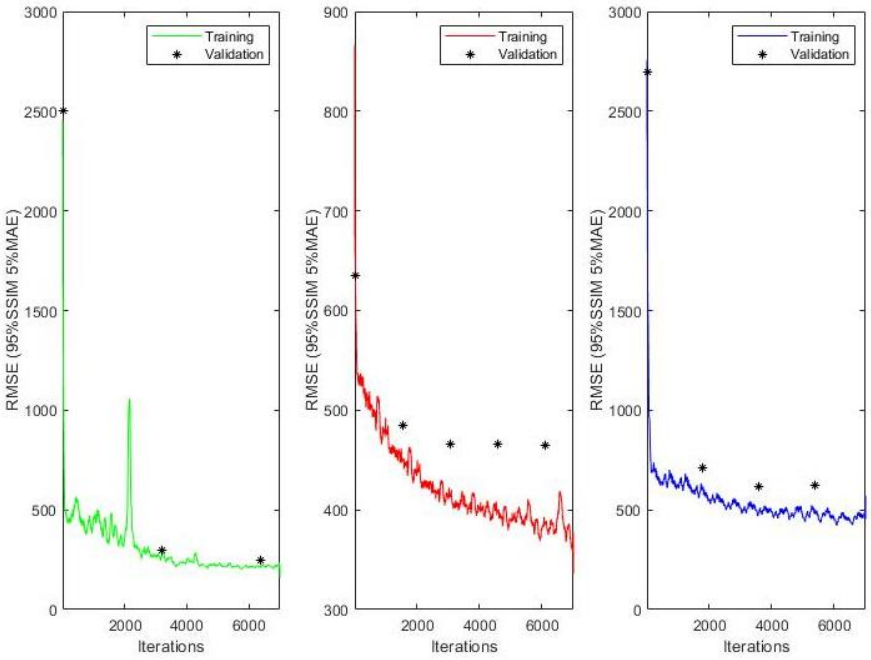


Figure 23. Variation of the training (lines) and validation (*) RMSE for a) the training on Gaussian noise (left); b) the U-NET $SSIM_0MAE_1RES$ retraining (middle); c) the equivalent Y-NET. Iterations were cut before destablization / overshooting / plateauing.

Table 1 contains the time (in seconds) taken to predict an image by the different U-NETs, the Y-NETs and the NLM algorithm. We can see that NLM is 75% fastest than the average U-NET, and up to 81% than Y-NET. We expected Y-NET to be slower than U-NET because of its additional down sampling branch, and indeed it is (about 35% slower).

	U-NET time (s)	Y-NET time (s)
95SSIM5MAE	0.602	0.716
9SSIM1MAE	0.532	0.711
0SSIM1MAE RES	0.516	0.712
NLM	0.1366	

Table 1. Time (in seconds) taken by each network/algorithm to perform a single prediction

Discussion

Both our neural networks achieve a denoising capability comparable to that of an optimized Non-Local Means algorithm. When we compare Figures 14b, 16 and 17, we see that there is a considerable gap between the results obtained with the image-wise quality metrics, where both networks tend to outperform NLM, and the DTI parameters calculated with the Diffusion Data Analysis Tool, where NLM yields the lowest error and variability, generally improving the metrics obtained with noisy data. Nonetheless, the results of our novel Y-NET are often comparable to those achieved by NLM, providing similar MD and E2A values. When it comes to estimating HA, however, U-NET looks more effective than the other two.

Even though statistical tests on our networks and NLM results would be necessary to compare their performances more robustly, the overall relative difference in the DTI values for a diastolic SE dataset is only 1% lower with the Y-NET than with the U-NET, although with the addition of STEAM data, we had hypothesized that the results would improve. Further improvements and analysis of our Y-NET network would be therefore required. We have seen from figures 20 to 22 that the results of our U-NET are substantially improved if instead of using a denoising ratio of 1:4, we use 1:2. With 1:2, U-NET performs slightly better than NLM (see Figure D7). A ratio 1:2 would still allow us to halve the acquisition time if the SNR is indeed decreased by $\sqrt{2}$-fold and seems to be a less ambitious approach.

Of course, the denoising methods could be applied to any number of averages, including the 8 or 16 averages typically acquired at present and may improve the results and allow higher spatial resolutions to be used.

Note that, as motion can cause differing signal loss in each of our multiple averages [29], and noise is not constant across the image due to coil sensitivity and the use of parallel imaging, we did not perform any SNR measurements to quantify the improvements achieved by our networks.

We hypothesized that the features extracted by the auxiliary branches of our Y-NETs would be different from the features extracted in their respective U-NETs. The reason is that, when embedded in the Y-NET as auxiliary branches, they must contribute to reconstruct the image from the other sequence. To account for this change, we looked at the activations of some intermediate layers of the down sampling branch. Figure 24 shows the activations of two channels of the second *maxpooling* layer of each branch of the SE,STEAM:SE Y-NET, compared to their equivalent output in each individual U-NET. While we hypothesized that the auxiliary branch should change between the networks, but the main branch would be similar, we can see in Figure 24 that both down-sampling branches change with regards to their equivalent U-NETs.

Further analysis would be required to understand how Y-NET combines the information from both sequences. Since we hypothesized that Y-NET would be more effective due to mutual information between the sequences, it would be interesting to validate this by comparing the results to those obtained with a 2-input Spin-Echo Y-NET and 2-input STEAM Y-NET.

A key point in building a neural network is choosing the correct loss function. Ours combines MAE and SSIM, and these metrics are indeed improved in the resulting individual images. However, these metrics might not suitable if what we want to improve are the DTI metrics. For that, we should ideally use a loss function that stresses the properties of diffusion images that are relevant for the tensor calculation, or directly optimizes the DTI parameters with regards to the ground truth. Optimizing this "semantic" information rather than the pixel values for denoising has worked for other techniques like CT [11], but the problem seems too complex in DT-CMR: on the one hand, multiple directions and the reference are necessary to calculate the tensor, on the other, backpropagate from the metrics to the denoised pixel values would be challenging. An alternative would be using a neural network to estimate the diffusion tensor and derived parameters directly from multiple directions and the reference. These images would contain all the information necessary to extract the tensor. Recent works in Deep Learning

applied to brain DTI have successfully extracted voxel-wise FA measures from raw DWI data [1], bypassing the tensor calculation. It would be interesting to see if such a network could be implemented in DT-CMR to extract the MD, HA, FA or E2A.

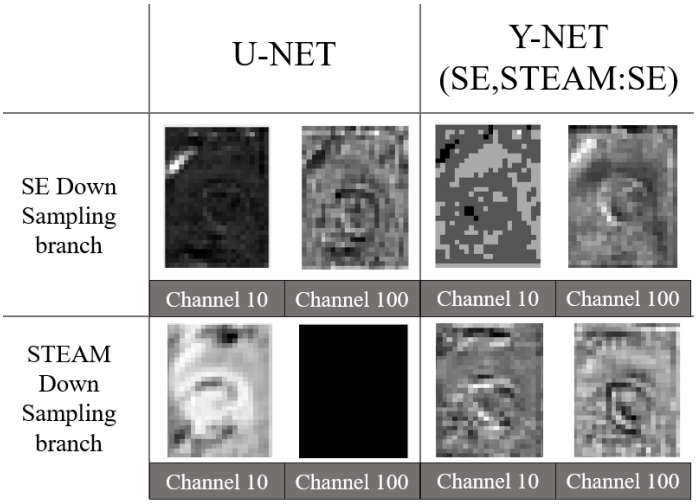


Figure 24. Activations of the 2nd *maxpooling* layer (channels 10 and 100) of both U-NETs, and of both branches of the SE,STEAM:SE Y-NET.

A less ambitious approach would be combining both images from the different sequences and from different b-values. Our dataset contained images with b = 150 s/mm^2 that were removed from the training dataset because they were too few to average. Low b-value images carry a high SNR and could assist a network to denoise an image with higher b-value. Wang et al. [36] demonstrated this approach in DWI, which outperformed other denoising algorithms such as BM3D (Block Matching), an algorithm widely used in denoising [39] and similar to NLM.

Wang et. al. [36] also replaced all *maxpooling* layers of the U-NET by convolutional layers. In other studies using U-NET for denoising [39][19], additional elements or modifications of the original structure are generally added. One example is the use of Wavelet transform layers instead of *maxpooling* layers to prevent loss of information at each contracting step [19]. In future developments, the effects of changing the *maxpooling* layers to prevent resolution loss should be assessed.

Another point to consider is that we used Gaussian noise to pre-train our model. Although we demonstrated that the assumption was acceptable given the noise histogram of our data, it would be interesting to see how U-NET works on the real dataset when pre-trained on spatially varying Rician noise, as the latter is the commonly adopted model for MRI noise [10]. Novel noise estimation algorithms for diffusion imaging have been able to model noise based on multiple b-value measurements. It would be interesting to use such an approach in the pre-processing steps [2]; statistically based denoising algorithms using this model have been developed in DTI with satisfactory results on FA measurements [16]. A neural network would be able to learn implicitly this underlying mathematics without being constricted by a rigid model.

GANs are being broadly used in Medical Imaging [38] and have proven effective in denoising tasks [13]. Maosong et al. developed a Residual GAN [21] to denoise various types of brain T1 weighted MR images, achieving a 2-fold increase in the PSNR. Although GANs are more difficult to train because they combine two sub-networks, a GAN with a 2-sequence input encoder and a discriminator network designed to optimize the DTI values measurements might improve our results in denoising cardiac DTI.

Conclusion

In this project, we have developed three algorithms capable of denoising Cardiac DTI images. The optimized NLM algorithm generally outperforms our within-sequence U-NET and cross-sequence 2-input Y-NET. Nonetheless, with no known precedent, our denoising Y-NET, that benefits from both SE and STEAM sequences, generally achieves better results than a regular U-NET. Even though further validation is needed and considerable improvements would be required to reduce the number of averages with a denoising Neural Network to lower the acquisition time in DT-CMR, our first approach reveals a promising prospect for the application of Deep Learning to Cardiac Diffusion Tensor Imaging.

References

1. Aliotta, E. , Nourzadeh, H. , Sanders, J. , Muller, D. and Ennis, D. B.. Highly accelerated, model-free diffusion tensor MRI reconstruction using neural networks. Medical Physics; 2019, 46: 1581-1591. doi:10.1002/mp.13400.

2. Alippor M, Maier S. Noise estimation and bias correction of diffusion signal decays: application to prostate diffusion imaging. International Society for Magnetic Resonance in Medicine 27th Annual Meeting and Exhibition; 2019.

3. British Heart Foundation. UK Factsheet. 2018.

4. Buades A, Coll B, Morel J M. A non-local algorithm for image denoising. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition; 2005; 2. 60- 65 vol. 2. 10.1109/CVPR.2005.38.

5. Burstein D. Stimulated echoes: description, applications, practical hints. Concepts Mag Reson 1996; 8:269-278.

6. Fernandez V. Machine learning based simultaneous reconstruction of spin-echo and stimulated echo diffusion cardiac MRI output to shorten the acquisition time. Planning report for the final project of the Master of Science in Bioengineering; Imperial College London (2019).

7. Ferreira PF, Kilner PJ, McGill LA, Nielles-Vallespin S, Scott AD, Ho SY, et al. In vivo cardiovascular magnetic resonance diffusion tensor imaging shows evidence of abnormal myocardial laminar orientations and mobility in hypertrophic cardiomyopathy. J Cardiovasc Magn Reson. 2014;16:87.

8. Froeling M, Strijkers GJ, Nederveen AJ, Chamuleau SA, Luijten PR. Diffusion Tensor MRI of the Heart – In Vivo Imaging of Myocardial Fiber Architecture. Current Cardiovascular Imaging Reports: Springer Science+Business Media New York; 2014.

9. Guizar-Sicairos M, Thurman ST, Fienup JR. Efficient subpixel image registration algorithms. Opt Lett. 2008;33(2):156-8.

10. Gudbjartsson, H, and S Patz. The Rician distribution of noisy MRI data. Magnetic resonance in medicine; 1995; vol. 34,6: 910-4. doi:10.1002/mrm.1910340618

11. Heinrich S, Stille M and Buzug T. Residual U-Net Convolutional Neural Network Architecture for Low-Dose CT Denoising. Current Directions in Biomedical Engineering; 2018; 4. 297-300. 10.1515/cdbme-2018-0072.

12. Immerkær, J.. Fast Noise Variance Estimation. Computer Vision and Image Understanding; 1996; 64, 300-302.

13. Kaplan S, Zhu YM. Full-Dose PET Image Estimation from Low-Dose PET Image Using Deep Learning: a Pilot Study. J Digit Imaging. 2018.

14. Khalique Z, Scott AD, Ferreira PF, Gorodezky M, Rick W, Nielles-Vallespin S, et al. An initial evaluation of STEAM and M012 spin echo diffusion tensor imaging in hypertrophic cardiomyopathy patients. 20th Annual SCMR Scientific Sessions Abstract Supplement,. 2017:93.

15. Kolařík M, Burget R, Uher V, Riha K, Kishore Dutta M. Optimized High Resolution 3D Dense-U-Net Network for Brain and Spine Segmentation. Applied Sciences; 2019; 9. 404. 10.3390/app9030404.

16. Lam F, Babacan SD, Haldar JP, Weiner MW, Schuff N, Liang ZP. Denoising diffusion-weighted magnitude MR images using rank and edge constraints. Magn Reson Med. 2014 Mar;71(3):1272-84. doi: 10.1002/mrm.24728. PMID: 23568755; PMCID: PMC3796128.

17. Le Bihan D, Iima M. Diffusion Magnetic Resonance Imaging: What Water Tells Us about Biological Tissues. PLoS Biol. 2015;13(7):e1002203.

18. Le Bihan D, Johansen-Berg H. Diffusion MRI at 25: exploring brain tissue structure and function. Neuroimage. 2012;61(2):324-41.

19. Liu P, Zhang H, Zhang K, Lin L, Zuo W. Multi-level Wavelet-CNN for Image Restoration. Computer Vision and Pattern Recognition NTIRE Workshop; Zurich 2018

20. Luk AYH, Yang G, Ferreira PF, Nielles-Vallespin N, Gorodezky M, Khalique Z, Pennell DJ, Firmin DN, Scott AD. Deep U-Net Reconstruction for Undersampled Spiral Diffusion Tensor Cardiovascular Magnetic Resonance.

21. Maosong R, Hu J, Chen Y, Chen H, Sun H, Zhou J, Zhang Y. Denoising of 3-D Magnetic Resonance Images Using a Residual Encoder-Decoder Wasserstein Generative Adversarial Network; 2019.

22. NHS. Cardiomyopathy [Available from: https://www.nhs.uk/conditions/cardiomyopathy/.

23 Nielles-Vallespin, S. , Scott, A. , Ferreira, P. , Khalique, Z. , Pennell, D. and Firmin, D. Cardiac Diffusion: Technique and Practical Applications. J Magn Reson Imaging; 2019; doi:10.1002/jmri.26912

24. Nielles-Vallespin S, Mekkaoui C, Gatehouse P, Reese TG, Keegan J, Ferreira PF, et al. In vivo diffusion tensor MRI of the human heart: reproducibility of breath-hold and navigator-based approaches. Magn Reson Med. 2013;70(2):454-65.

25. Redmon, Joseph & Divvala, Santosh & Girshick, Ross & Farhadi, Ali. (2016). You Only Look Once: Unified, Real-Time Object Detection. 779-788. 10.1109/CVPR.2016.91.

26. Redmon, Joseph & Farhadi, Ali. (2016). YOLO9000: Better, Faster, Stronger.

27. Reese TG, Weisskoff RM, Smith RN, Rosen BR, Dinsmore RE, Wedeen VJ. Imaging myocardial fiber architecture in vivo with magnetic resonance. Magn Reson Med. 1995;34(6):786-91.

28. Ronneberger O, Fischer P, Brox T, editors. U-Net: Convolutional Networks for Biomedical Image Segmentation 2015; Cham: Springer International Publishing. 18. Han Y, Ye JC. Framing U-Net via Deep Convolutional Framelets: Application to Sparse-View CT. IEEE Trans Med Imaging. 2018;37(6):1418-29.

29. Scott AD, Nielles-Vallespin S, Ferreira PF, Khalique Z, Gatehouse PD, Kilner P, et al. An in-vivo comparison of stimulated-echo and motion compensated spin-echo sequences for 3 T diffusion tensor cardiovascular magnetic resonance at multiple cardiac phases. Journal of Cardiovascular Magnetic Resonance. 2018;20.

30. Stoeck C, von Deuster C, Gener M, Atkinson D, Kozerke S. Second-order motion-compensated spin echo diffusion tensor imaging of the human heart. Magnetic Resonance in Medicine; 2016; 75:1669-1676.

31. Szegedy C, Vanhoucke V, Ioffe S, Shlens J and Wojna Z. Rethinking the Inception Architecture for Computer Vision. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, 2016, pp. 2818-2826.

32. Tkachenko NV. Optical spectroscopy : methods and instrumentations. Chapter 7.2.2: Signal Averaging. Amsterdam ; Oxford: Elsevier; 2006. Available at: https://www.sciencedirect.com/topics/chemistry/signal-to-noise-ratio

33. Xu H, Schneider JE, Young AA, Grau V. Fully Automated Segmentation of the Left Ventricle in Small Animal Cardiac MRI. 1st Conference on Medical Imaging with Deep Learning (MIDL 2018); Amsterdam, The Netherlands 2018.

34. Xu J, Gong E, Pauly J,Zaharchuk G. 200x Low-dose PET Reconstruction using Deep Learning . Computer Vision and Pattern Recognition; 2017.

35. Wang Z, Bovik AC, Sheikh HR, Simoncelli EP. Image quality assessment: from error visibility to structural similarity. IEEE Transactions on Image Processing; 2004; 13(4); 600-612.

36. Wang H, Zheng R, Dai F, Wang Q, Wang C. High-field MR diffusion-weighted image denoising using a joint denoising convolutional neural network. Journal of magnetic resonance imaging; 2019.

37. Wu MT, Su MY, Huang YL, Chiou KR, Yang P, Pan HB, et al. Sequential changes of myocardial microstructure in patients postmyocardial infarction by diffusion-tensor cardiac MR: correlation with left ventricular structure and function. Circ Cardiovasc Imaging. 2009;2(1):32-40, 6 p following

38. Yi X, Walia E, Babyn P. Generative Adversarial Networks in Medical Imaging: A Review; Medical Imaging Analysis; 2019.

39. Zhao H, Gallo O, Frosio I, Kautz J. Loss Functions for Image Restoration with Neural Networks. IEEE Transactions on Computational Imaging; 2017; 3(1).

40. Zhang G, Wang S, Wen D, Zhang J, Wei X, Ma W, Zhao W, Wang M, Wu G, Zhang J. Comparison of non-Gaussian and Gaussian diffusion models of diffusion weighted imaging of rectal cancer at 3.0 T MRI. Scientific Reports; 2016;9;6:38782. doi: 10.1038/srep38782. PMID: 27934928; PMCID: PMC5146921.

Annex

This section provides additional information about the steps undertaken during this project. Annexes are referenced all along the report, except for Annex E, which contains a detailed description of the scripts that were created or used for this Project. Most of the MATLAB scripts mentioned were created by the author of this report; scripts that were created by somebody else have the author's name specified whenever they are mentioned.

Contents

Annex A. Dataset

1. Number of averages per {subject, phase, sequence, direction}

HS 1	Diastole	SE	18	HS 6	Diastole	STEAM	10	HS 11	Sweet-Spot	SE	18	P 1	Systole	STEAM	12	P 9	Systole	SE	20
HS 1	Diastole	STEAM	10	HS 6	Sweet-Spot	SE	18	HS 11	Sweet-Spot	STEAM	10	P 2	Diastole	SE	20	P 9	Systole	STEAM	10
HS 1	Sweet-Spot	SE	18	HS 6	Sweet-Spot	STEAM	10	HS 11	Systole	SE	18	P 2	Diastole	STEAM	10	P 10	Diastole	SE	32
HS 1	Sweet-Spot	STEAM	10	HS 6	Systole	SE	18	HS 11	Systole	STEAM	10	P 2	Systole	SE	19	P 10	Diastole	STEAM	10
HS 1	Systole	SE	18	HS 6	Systole	STEAM	10	HS 12	Diastole	SE	18	P 2	Systole	STEAM	9	P 10	Systole	SE	22
HS 1	Systole	STEAM	10	HS 7	Diastole	SE	20	HS 12	Diastole	STEAM	10	P 3	Diastole	SE	30	P 10	Systole	STEAM	11
HS 2	Diastole	SE	18	HS 7	Diastole	STEAM	10	HS 12	Sweet-Spot	SE	22	P 3	Diastole	STEAM	16	P 11	Diastole	SE	18
HS 2	Diastole	STEAM	10	HS 7	Sweet-Spot	SE	18	HS 12	Sweet-Spot	STEAM	10	P 3	Systole	SE	26	P 11	Diastole	STEAM	14
HS 2	Sweet-Spot	SE	18	HS 7	Sweet-Spot	STEAM	10	HS 12	Systole	SE	18	P 3	Systole	STEAM	14	P 11	Systole	SE	18
HS 2	Sweet-Spot	STEAM	10	HS 7	Systole	SE	18	HS 12	Systole	STEAM	10	P 4	Diastole	SE	26	P 11	Systole	STEAM	11
HS 2	Systole	SE	18	HS 7	Systole	STEAM	10	HS 13	Diastole	SE	18	P 4	Diastole	STEAM	15				
HS 2	Systole	STEAM	10	HS 8	Diastole	SE	18	HS 13	Diastole	STEAM	10	P 4	Systole	SE	22				
HS 3	Diastole	SE	20	HS 8	Diastole	STEAM	10	HS 13	Sweet-Spot	SE	18	P 4	Systole	STEAM	13				
HS 3	Diastole	STEAM	10	HS 8	Sweet-Spot	SE	18	HS 13	Sweet-Spot	STEAM	10	P 5	Diastole	SE	18				
HS 3	Sweet-Spot	SE	18	HS 8	Sweet-Spot	STEAM	10	HS 13	Systole	SE	18	P 5	Diastole	STEAM	11				
HS 3	Sweet-Spot	STEAM	10	HS 8	Systole	SE	18	HS 13	Systole	STEAM	10	P 5	Systole	SE	18				
HS 3	Systole	SE	18	HS 8	Systole	STEAM	10	HS 14	Diastole	SE	18	P 5	Systole	STEAM	11				
HS 3	Systole	STEAM	10	HS 9	Diastole	SE	18	HS 14	Diastole	STEAM	10	P 6	Diastole	SE	22				
HS 4	Diastole	SE	18	HS 9	Diastole	STEAM	10	HS 14	Sweet-Spot	SE	18	P 6	Diastole	STEAM	10				
HS 4	Diastole	STEAM	10	HS 9	Sweet-Spot	SE	18	HS 14	Sweet-Spot	STEAM	10	P 6	Systole	SE	18				
HS 4	Sweet-Spot	SE	16	HS 9	Sweet-Spot	STEAM	10	HS 14	Systole	SE	18	P 6	Systole	STEAM	10				
HS 4	Sweet-Spot	STEAM	10	HS 9	Systole	SE	18	HS 14	Systole	STEAM	10	P 7	Diastole	SE	18				
HS 4	Systole	SE	18	HS 9	Systole	STEAM	10	HS 15	Diastole	SE	20	P 7	Diastole	STEAM	10				
HS 4	Systole	STEAM	10	HS 10	Diastole	SE	18	HS 15	Diastole	STEAM	10	P 7	Systole	SE	24				
HS 5	Diastole	SE	18	HS 10	Diastole	STEAM	10	HS 15	Sweet-Spot	SE	18	P 7	Systole	STEAM	10				
HS 5	Diastole	STEAM	10	HS 10	Sweet-Spot	SE	18	HS 15	Sweet-Spot	STEAM	10	P 8	Diastole	SE	18				
HS 5	Sweet-Spot	SE	18	HS 10	Sweet-Spot	STEAM	10	HS 15	Systole	SE	20	P 8	Diastole	STEAM	10				
HS 5	Sweet-Spot	STEAM	10	HS 10	Systole	SE	18	HS 15	Systole	STEAM	10	P 8	Systole	SE	18				
HS 5	Systole	SE	18	HS 10	Systole	STEAM	10	P 1	Diastole	SE	22	P 8	Systole	STEAM	10				
HS 5	Systole	STEAM	10	HS 11	Diastole	SE	18	P 1	Diastole	STEAM	12	P 9	Diastole	SE	20				
HS 6	Diastole	SE	18	HS 11	Diastole	STEAM	10	P 1	Systole	SE	23	P 9	Diastole	STEAM	11				

Table A1. Number of averages per {subject, phase, sequence} dataset. Program used: count_averages.m (folder: Preprocessing/Dataset Reporting).

2. Configuration and use of the test dataset

The test dataset was processed separately from the training/validation dataset (see Annex E for technical information), but undergone cropping, registration and removal of invalid images as well. The script *dicomtest.m* (folder: Dataset processing) takes the test dataset structures (cell array with the subjects, each subject containing the embedded phases, sequences and sub-datasets) and stores them in a folder system. For each subject, phase and sequence, the script creates 2 types of folder:

- *average_noisy*: Takes each average and stores a DICOM file for each direction (including the reference). It stores one set of 7 images with b=150 s/mm^2 as well (picking a random 150 index from the ones available – there should be 1 for STEAM and 2 for Spin-Echo).
- *average_denoised*: Takes each averaged image and stores a DICOM file for each direction (including the reference). It also adds 6 directions + reference of noisy 150 images (as there were no denoised ones due to unavailability). Makes sure that in these folders, there is no direction (or reference) missing. Deletes the folder otherwise.

From these folders, we manually created new folders containing the equivalent of 8 averages for the denoised ones (if the ground truth were averages of 4, then 2 sets of 6 directions + reference), labelled (2) in Figure A1 and the equivalent to 8/N (number of ground truth averages) for the noisy folders (labelled (2) in Figure A1). Sometimes, it was not possible to get the equivalent of 8 averages and had to use a smaller number.

These new denoised folders (a random one was picked for each dataset) were the ones processed with the Diffusion Data Analysis Tool to get the ground truth. A random noisy folder was picked up as well, and the testing programs (*process_results_integrated*.m*) pick up each DICOM and:

- Denoises it with NLM, and the input network.
- Stores it in a folder with the same name as the noisy folder in the directory where the network is, as shown in Figure A1.

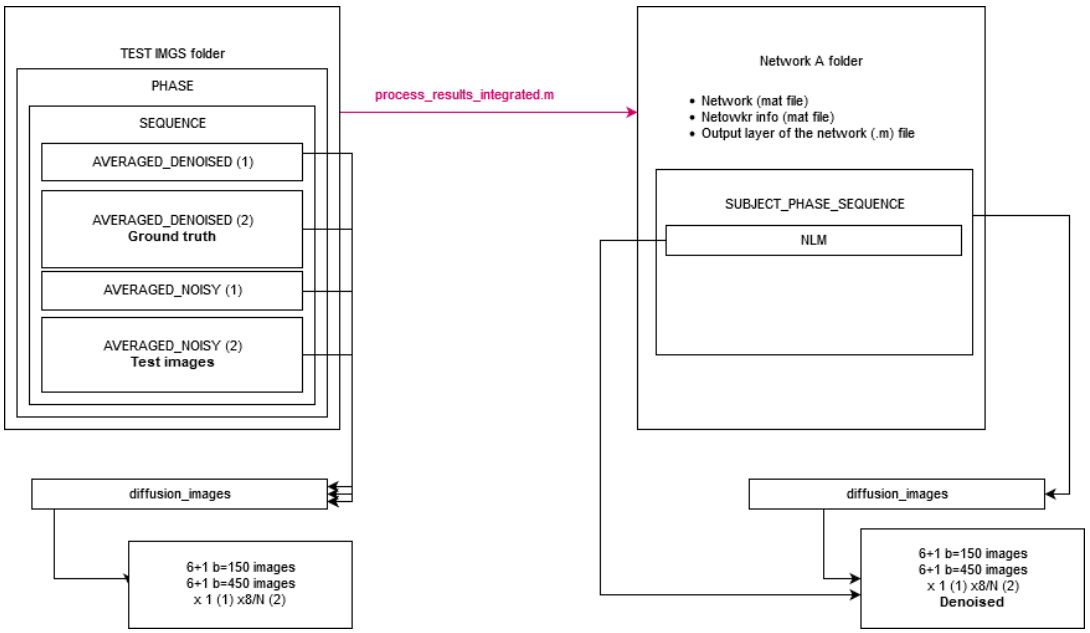


Figure A1. Diagram showing how the test images folder (left) is processed into the network results folder by script process_results_integrated (right).

Besides creating these denoised folders, on which we use the Diffusion Analysis Data Tool, the script also:

- Calculates the RMSE, MAE and SSIM between each direction and its equivalent direction in the denoised folder and puts these values in a table that is then saved in a folder "*results_SEQUENCE*" in the network folder.
- In the same folder, it stores the images (noisy, ground truth, network output and NLM output) in a single JPG file.
- Fetches the *matlab_data* folder (if there is one) from the denoised folder, and saves it on each processed folder containing the *diffusion_images* folder with the DICOM files. This ensures that the region of interest (ROI), thresholds etc. are the same for the ground truth and the network or NLM denoised data and makes the DTI values comparison more consistent.

Annex B. Non-Local Means Analysis

MATLAB NLM function *imnlmfilt* has three main parameters: the degree of smoothing, the comparison window size and the search window size [4]. The degree of smoothing controls how strong is the effect of the neighbouring pixels on each pixel that is being denoised. The search window size is the window that we look at to denoise a specific pixel p, whereas the comparison window size is the neighbouring area of each pixel q of the search window size that we evaluate to determine the weight associated to that pixel.

To find the optimal arguments for *imnlmfilt*, we applied the function to a randomly selected subset of SE and STEAM images. We calculated the mean absolute error (MAE), the root mean square error (RMSE) and the structural similarity index (SSIM) between the NLM-denoised image and the ground truth (average of 4). We quantified how these metrics varied with regards to those computed for the noisy image and the ground truth. The results of our analysis can be seen in figures B1, B2 and B3. We finally chose a degree of smoothing factor of 4, a search window size of 27, and a comparison window size of 3 pixels.

- Parameter 1: Degree of smoothing. The default degree of smoothing is calculated as per equation 2 in [12]. We multiplied this base degree of smoothing by an increasing factor between 0.1 and 10. The higher, the blurrier the resulting image is. Values smaller than 1 did not mark a difference with the noisy image. Figure B1 shows that a factor of 4 leads to the highest increase in SSIM and the highest decrease in RMSE.

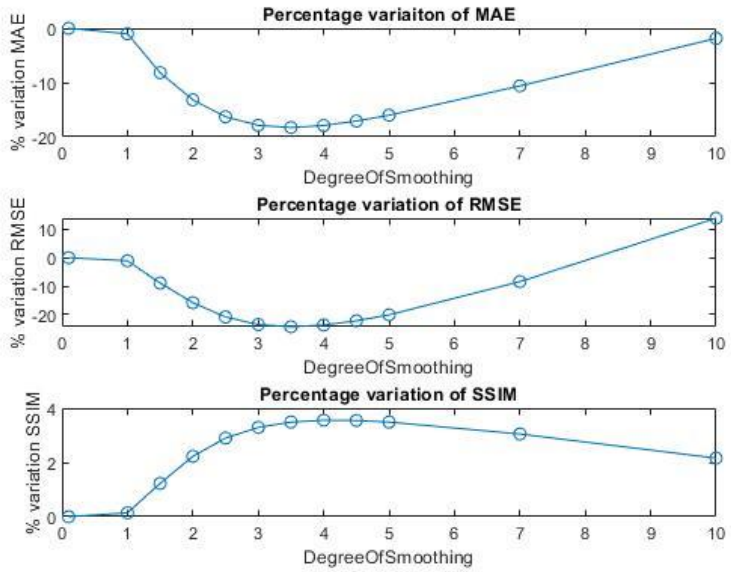


Figure B1. Mean MAE, RMSE, SSIM variations between ground truth and noisy image and groud truth and denoised image for a 200-image subset, as a function of the factor multiplying the base degree of smoothing.

- Parameter 2: Search Window Size. The neighbouring pixels are analysed within a window that is set up by this parameter. The highest the size, the better the results. Nonetheless, high window sizes increased the computation time considerably (see Figure B2).

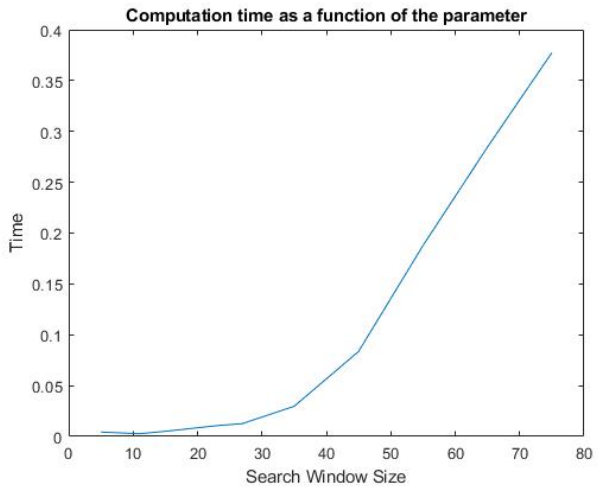


Figure B2. Time to process the NLM filtering of one image as a function of the search window size.

To yield a good compromise between time and image quality, we set the window size at 27, yielding an increase in SSIM of 0.2% and a 2% decrease in MAE and RMSE.

- Parameter 3: Comparison Window Size. The neighbourhood of each neighbouring pixel is analysed within a window set by this parameter. Better quality is obtained with smaller window sizes (Figure B3). Thus, we set up the size at 3 pixels.

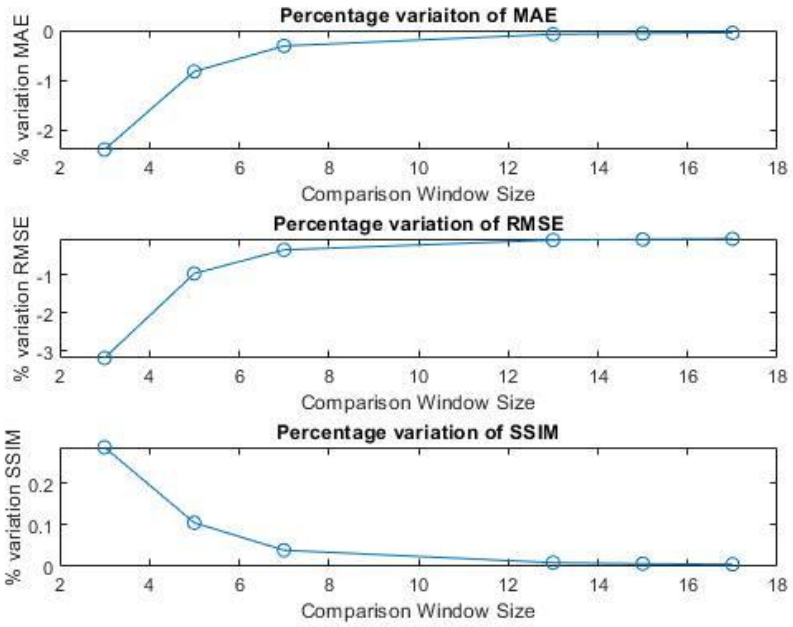


Figure A=B3. Mean MAE, RMSE, SSIM variations between ground truth and noisy image and groud truth and denoised image for a 200-image subset, as a function of the comparison window size.

Note: For the validation with Gaussian noise, the program reported a set of different values for the three parameters: search window size 13, comparison window size 3 and degree of smoothing factor 1. We applied them for the corresponding section. For the training on the N = 6 averages, the results were: search window size 35, comparison window size 3 and degree of smoothing factor 6; for the training on the N=2 averages, the results were: search window size 13, comparison window size 3, and degree of smoothing factor 2.

Program used: *NLM_reporting.m* (folder: *Preprocessing/NLM_tuning*).

Annex C. Cropping Network

1. Overview

We considered several algorithms: Hough-transform based segmentation, U-Net based segmentation [28] and the *You Only Look Once* "YOLO" network for object detection [25]. We chose YOLO because our ground truth is a bounding-box, and YOLO outputs a bounding-box instead of a segmented region, and because YOLO has proven to be very effective in object recognition [25].

To implement YOLO in MATLAB, we used its second implementation, YOLO9000 [26], that requires a feature-extraction sub-network as input. The feature extraction network was implemented from a pre-trained shortened *inceptionv3* [31] network re-trained on a dataset to detect the heart in an image.

We trained the network using 3 anchor boxes of sizes 120x88, 100x70 and 80x50 pixels. Our training dataset consisted on 1901 images. We trained the network with Adam Optimizer, with a learning rate of 0.01, $\beta_1 = 0.95$, $\beta_2 = 0.999$ and $\varepsilon = 10^{-7}$. The output is defined to have size 5: [height, width, center (X, Y) and probability of belonging to the object in question].

2. Classifier

MATLAB YOLO v2 implementation needs of a pre-trained classification network trained to identify whether the object in question is or is not in the image. Because our dataset contained only heart images, we had to use an open-source repository with non-heart images. The dataset was therefore constituted of:

The ground-truth of our denoising U-Net (average "denoised" images coming from the same {subject, phase, sequence, direction} dataset), labelled as "heart".

CT, MR images downloaded from an open medical image repository, Medpix (*https://medpix.nlm.nih.gov/topiclist*) . The images were cropped to fit the size of the size of the heart images and were labelled as "other" (see Figure C1).

Although we intendedly trained our network to overfit on the plane in which the heart has been acquired in our dataset, we allowed this as the purpose of this network is to identify the heart in our dataset. The network was re-trained for 10 epochs and yielded a classification accuracy of 99.36% (see Figure C1).

The classification network was based on a pre-trained *inceptionv3* network imported from MATLAB. As the input had to be 299x299 images, we had to fit our images to the required size by adding a frame of zeros. We did the same with the non-heart images; these were retrieved from Medpix and cropped manually to fit the initial size of our images (256x96). We used the training parameters used in [31], these are a learning rate of 0.045, *RMSprop* optimization with 0.9 decay and $\varepsilon = 1.0$, and exponential learning rate decay factor of 0.94. Due to the reduced size of our dataset without data augmentation (427 images without a heart present) and to the limitations of our computational resources, we decided to reduce the batch-size to 10. We re-trained the network for 10 epochs on our augmented dataset of 5498 images, yielding a validation accuracy of 99.36%. The training accuracy is plotted in C1 c).

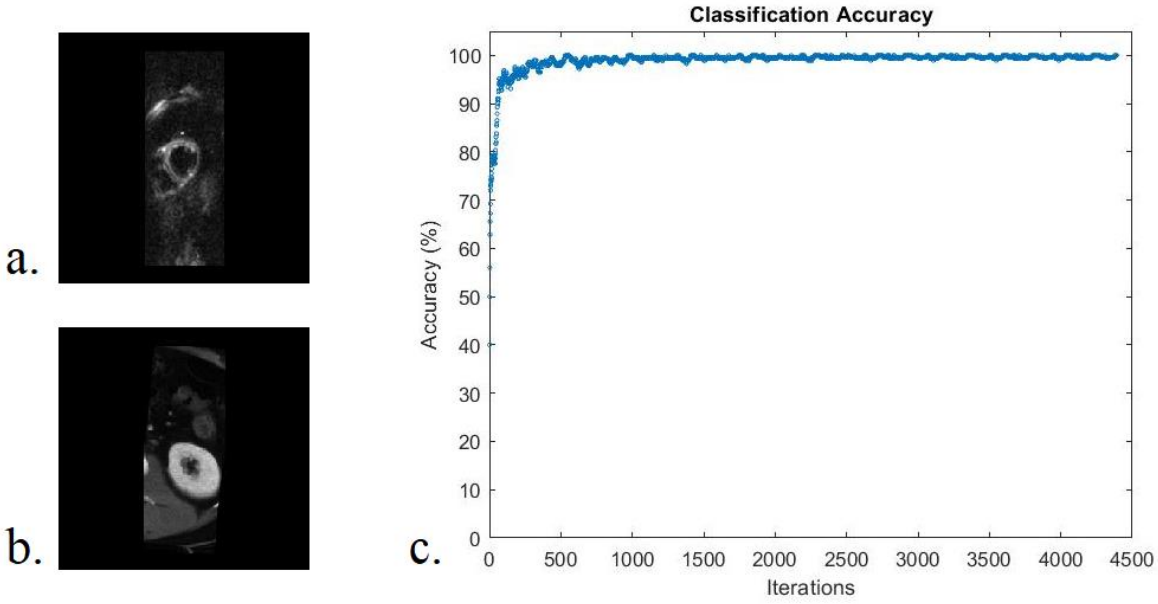


Figure C1. Classification Accuracy on the inceptionv3 network; a) heart image used for the training (category: heart), b) kidney image used for the training (category: non-heart); c) classification accuracy as a function of the number of iterations.

3. Results of YOLO Detector

Although the classification step filled its purpose, all the test on the trained YOLO network resulted in empty bounding-boxes. Surprisingly, the training loss and RMSE curves (see Figure C2) showed a sharp decrease during the first 50 iterations that resulted in a near-zero RMSE value. This YOLO implementation did not allow for validation during the training, and therefore we could not easily know if the network was simply overfitting on our data. We debugged MATLAB program *detect.m* and found out that the calculated probabilities for a potential bounding box were too low (between 15 and 40%) even when the heart was present in the bounding box. Since the positive detection threshold was 60%, no detected region of interest met the requirements and resulted in the heart not being detected. The threshold could not be modified because all the scripts from the YOLO9000 implementation in MATLAB were read-only.

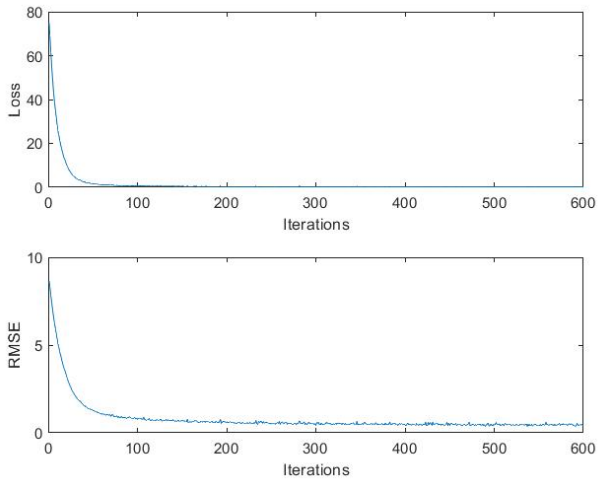


Figure C2. Upper image: training loss evolution during the YOLO training; Lower image: training RMSE evolution during the YOLO training.

A reason why YOLO failed could be our classification network. Although a high classification accuracy was yielded, we wonder whether our non-heart images were tricky enough to carry out a true learning from the network. We analysed some of its hidden layers (annex Figure C3) in debugging. Even though we achieved a satisfactory classification, our *inceptionv3* network might have been trained on features that are not structural, but rather more related to the intensity maps. Figure C3 shows a randomly selected channel of the activation of some layers of the network. We can see that in the deepest layers, there doesn't seem to be an emphasis on the structure, but rather a combination of structure and intensity values that give importance to pixels that are not part of the heart.

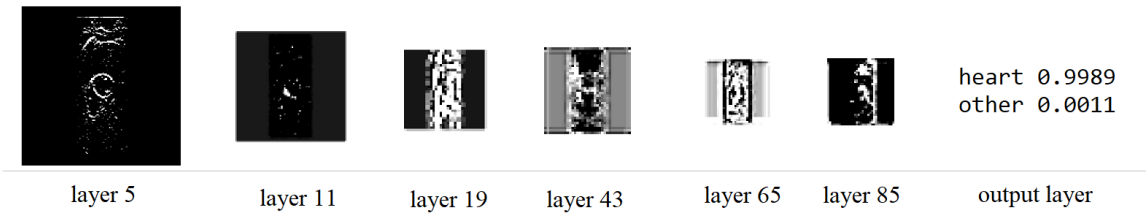


Figure C3. Activations of some layers of the fully trained classification inceptionv3 network (92 layers in total)

That is one reason why YOLO might have failed, and it is a direct consequence of the limitations of our dataset: the images labelled as "heart" had approximately the same intensity scales and composition, whereas the images labelled as "others" had completely different intensity scales. As a future modification, the images could be modified to match the pixel scale of ours and ensure that there is not only a structural likeliness, but also a composition similarity. Another solution would be using a dataset containing cardiac images from other sequences or even techniques that help our network to generalize better.

Annex D. Neural Network tuning

1. Training parameters

The defaults parameters were extracted from [11] , [19] and [26]. Although most consulted U-NET papers use a Stochastic Gradient Descent optimizer, we attempted a small training using Adam, as it is the most commonly used optimizer in Deep Learning. The parameters for Adam were extracted form [26]. Using a randomly selected subset of 2000 Gaussian-noisy images, we trained the network for 5 epochs varying the parameters reported in Table D1.

Parameter	Value	Def. LR	Def. DF	Def. Drop Period	Def. MB	Def. OPT	Def. CL	File name	Reduction of RMSE
Optimizer (OPT)	SDGM	0.1	0	1	10		With	UNETG2000_SDGM	**140%**
Optimizer (OPT)	ADAM	0.1	0	1	10		With	UNETG2000_ADAM	51%
Mini Batch Size (MB)	5	0.5	0	1		SDGM	With	UNETG2000_MB5	66%
Mini Batch Size (MB)	10	0.5	0	1		SDGM	With	UNETG2000_MB10	48%
Mini Batch Size (MB)	15	0.5	0	1		SDGM	With	UNETG2000_MB15	74%
Mini Batch Size (MB)	25	0.5	0	1		SDGM	With	UNETG2000_MB25	**509%**
Learning Rate (LR)	0.001		0	1	10	SDGM	With	UNETG2000_LR001	**49%**
Learning Rate (LR)	0.01		0	1	10	SDGM	With	UNETG2000_LR01	48%
Learning Rate (LR)	0.1		0	1	10	SDGM	With	UNETG2000_LR1	43%
Learning Rate (LR)	0.5		0	1	10	SDGM	With	UNETG2000_LR5	2%
Last Concatenation Layer (CL)	With	0.1	0	1	10	SDGM		UNETG2000_SDGM	**140%**
Last Concatenation Layer (CL)	Without	0.1	0	1	10	SDGM		UNETG2000_WOCL	31%
Drop Factor	0.9	0.1		1	10	SDGM	With	UNETG2000_DF9	43%
Drop Factor	0.5	0.1		1	10	SDGM	With	UNETG2000_DF5	46%
Drop Factor	0.1	0.1		1	10	SDGM	With	UNETG2000_DF1	**47%**
Drop Factor	0.01	0.1		1	10	SDGM	With	UNETG2000_DF01	40%

Table D1. %reduction of the RMSE between the first and the 200th iteration of the training as a function of the training parameters.

Figure D1 shows the results for the different tested parameters.

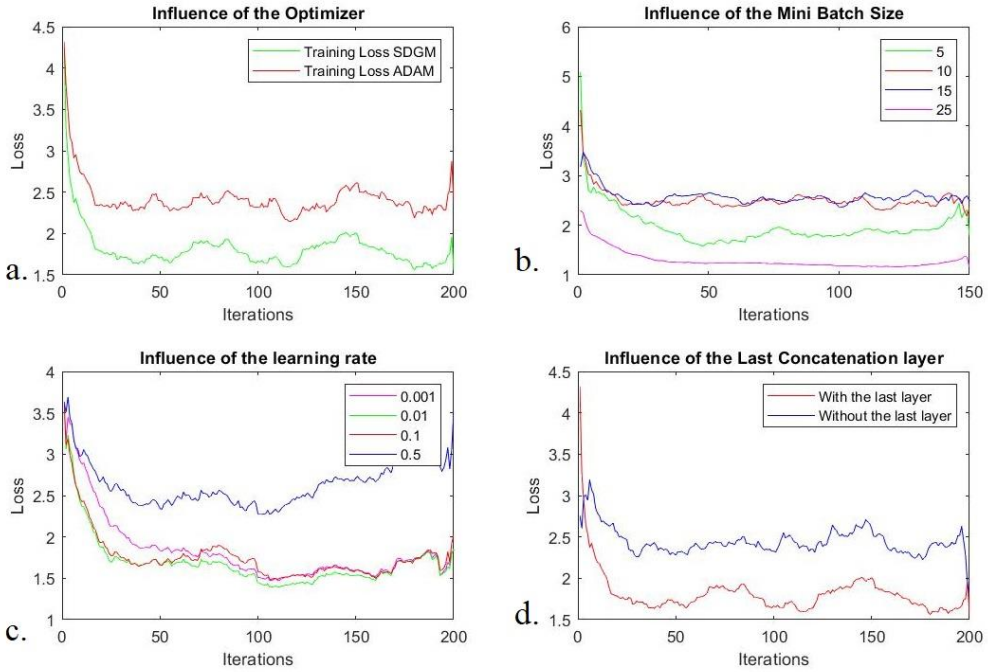


Figure D1, a) Evolution of the training loss using Adam or SDGM optimizer; b) influence of the mini-batch size on the training loss; c) influence of the learning rate on the training loss; d) influence of the removal of the last concatenation layer on the training loss

We only plotted and selected the first 200 iterations, as in some cases, there was a severe overfitting or even unexplainable destabilization of the convergence, coinciding a plateauing in other

cases. This narrowing seemed reasonable, as we had very few training images, and specially with high mini-batch sizes, the number of epochs was maybe excessive for a fixed learning rate.
Although mini-batch size provided the best performance, be chose size 15 because a size of 25 became computationally costly even using a GPU. Learning rates 0.1 and 0.01 have the fastest and most effective convergence. Since we intended to apply a drop factor during the training, we picked up 0.1. As expected, SDGM performs much better than Adam optimizer.

We performed all the trainings on residuals: we input the original image, but trained the network to output the predicted noise (the ground truth was the difference between the denoised and the noisy image). As the concatenation layers in U-NET are meant for image reconstruction, we hypothesized that removing the last concatenation layer would help the network to output a noise map, rather than a reconstructed denoised image. Nonetheless, as can be seen in Figure D1 d, the best results were obtained with the last concatenation layer.

Finally, we applied a varying exponential drop factor on the learning rate. There was no significant difference between the results in this case, although a drop factor of 0.1 gave slightly better results. Notice that the learning rate was reduced here every 1 epoch because our training consisted of 5 epochs. However, for the long trainings of 40 epochs, the period was increased to 5.

The program used to plot the parameters is *plot_parameter_tunings.m* (*folder*: U-NET/*GAUSSIAN_TRAININGS/PARAMETERS_TUNING).*

2. Output layer: loss functions

An important issue with the commonly used l_2 norm (equation D1) is that it gives excessive important to high intensity differences over more widespread and frequent small differences. This causes artifacts that are not seen with l_1 (equation D2). On the other hand, the Structural Similarity Index (see equation 6) takes into account the luminance (*l*), and the contrast-structure (*cs*), making it a more sharper estimator of the similarity between two images [39]. However, calculating a SSIM index on an image requires applying the formula pixel by pixel (as Gaussian filters centered on a single pixel are used to compute the mean and standard deviation). This is an excessively costly and complex problem. In [39], they bypassed this issue by calculating SSIM only on the central pixel of the image. Using a sufficiently big filter, we englobe the region of interest of our images, the myocardium. We manually implemented the SSIM-loss function in MATLAB by taking the forward and backpropagation formulas provided in [39]. We varied the standard deviation of our filter from σ=50 to σ=100 (Figure D2); the filter size that worked better was 70. Training on SSIM only generated a slight blur around the center of the image. We therefore decided to combine this loss value with that of the MAE loss by adding them up.

Figure D2. Main action area of a Gaussian filter with σ = 100.

$$l_1 = \sum_{i=1}^{N} |t_i - y_i| \quad (D1)$$

$$l_2 = \sum_{i=1}^{N} (t_i - y_i)^2 \quad (D2)$$

Where t_i is a pixel from the target image (ground truth), y_i is a pixel denoised by our network, N is the number of pixels.

Note that when training on residuals, only the MAE loss was used: the noise being close to a Gaussian or Rician distribution, does not seem to follow a strucutre, and thus SSIM was not relevant in this case.

The following tables contain the results for the training on different loss functions. They were created using Microsoft Excel ® and using program *exportdata2excel.m* (*folder: Neural_Networks*). This data corresponds to the raw data, from which we calculated relative errors.

	WEIGHTS	SEQUENCE	VAR. MAE GT-NLM	VAR. MAE GT-ANN	VAR. RMSE GT-NLM	VAR. RMSE GT-ANN	VAR. SSIM GT-NLM	VAR.SSIM GT - ANN	NUMBER OF POOLED IMAGES
On images									
SSIM	0.95	SE	-55%	**-66%**	-80%	**-89%**	43%	**113%**	24
MAE	0.05	STEAM	-52%	**-53%**	**-300%**	-78%	39%	**56%**	24
SSIM	0.90	SE	**-52%**	**-52%**	-80%	**-87%**	76%	**105%**	24
MAE	0.10	STEAM	**-52%**	**-52%**	-75%	**-77%**	39%	**48%**	24
SSIM	0.70	SE	-55%	**-60%**	-80%	**-85%**	43%	**54%**	24
MAE	0.30	STEAM	**-111%**	-45%	**-75%**	-67%	28%	**59%**	24
SSIM	0.30	SE	-55%	**-56%**	-23%	**-61%**	43%	**101%**	24
MAE	0.70	STEAM	**-52%**	-29%	**-75%**	-52%	39%	**43%**	24
SSIM	0.00	SE	**-55%**	**-55%**	-80%	**-82%**	76%	**100%**	24
MAE	1.00	STEAM	**-52%**	-30%	**-75%**	-52%	39%	**58%**	24
On residuals									
SSIM	0.00	SE	-55%	**-60%**	-80%	**-85%**	76%	**100%**	24
MAE	1.00	STEAM	**-53%**	**-53%**	-75%	**-78%**	39%	**58%**	24

Table D2. Individual image metrics for different MAE and SSIM weights; each column contains either the variation of the Mean Absolute Error (MAE), Root Mean Square Error (RMSE) and Structural Similarity Index (SSIM) between the noisy and denoised image. GT: Ground truth; NLM: Non-Local Means; ANN: Artificial Neural Network. The bold numbers highlight the denoising method (ANN or NLM) that scores the best variation.

	Weights	Sequence	%Error FA (ANN)	%Error MD (ANN)	%Error HA-ENDO (ANN)	%Error HA-EPI (ANN)	%Error Lp GRADIENT (ANN)	%Error Median E2A (ANN)	%Diff. in negative Eigv. (ANN)	% ANN Better	Mean Error (excl. Eigv.)	MEAN % ANN BETTER	Mean Error	Mean %Diff. In Eigv.
On images												CROSS SEQUENCE		
SSIM	0.95	SE	7%	6%	2%	14%	17%	19%	-5%	57%	11%	57%	**10%**	**-53%**
MAE	0.05	STEAM	1%	6%	11%	13%	11%	11%	-100%	57%	9%			
SSIM	0.90	SE	9%	8%	2%	13%	13%	15%	25%	71%	10%	**64%**	**9%**	**-30%**
MAE	0.10	STEAM	2%	11%	15%	15%	3%	8%	-85%	57%	9%			
SSIM	0.70	SE	19%	16%	4%	4%	37%	11%	370%	14%	15%	43%	13%	153%
MAE	0.30	STEAM	1%	5%	8%	17%	20%	14%	-65%	71%	11%			
SSIM	0.30	SE	14%	6%	2%	10%	42%	12%	40%	57%	14%	50%	13%	-20%
MAE	0.70	STEAM	2%	10%	13%	16%	13%	10%	-80%	43%	11%			
SSIM	0.00	SE	17%	12%	3%	2%	37%	10%	285%	86%	13%	**71%**	12%	128%
MAE	1.00	STEAM	2%	6%	13%	18%	18%	11%	-30%	57%	11%			
On residuals														
SSIM	0.00	SE	14%	5%	4%	10%	24%	10%	-170%	71%	11%	**64%**	**9%**	**-255%**
MAE	1.00	STEAM	2%	5%	8%	11%	3%	11%	-340%	57%	7%			
FA: Fractional Anisotropy; MD: Mean Diffusivity; ENDO: Endocardium; EPI: Epicardium; Diff: Difference; Eigv: Eigenvalues; excl: excluding														
NLM		SE	16%	5%	3%	10%	32%	13%	195%	N/A	13%	N/A	11%	-255%
		STEAM	2%	7%	11%	6%	9%	18%	-54%	N/A	9%			

Table D3. Table illustrating the mean relative errors of certain DTI metrics for both sequences. The metrics are calculated based on a subject dataset constituted of 2 phases: diastole and systole. The column "% ANN Better" calculates the proportion of ANN (Artificial Neural Network) outperforming NLM cases. The Mean Error excludes the Negative

Eigenvalues % because that metric aims to be as big and negative as possible, as opposed to the others that should get close to 0, and because it tended to have a too high standard variation. The last three columns average the % of outperformance and the error between the sequence-specific values. In bold are the networks yielding the best results. The NLM row is an average of all the NLM values obtained (we computed NLM results once per network).

3. Real dataset results

Table D4 shows a summary of the image quality metrics for the training of the U-NETs on the real dataset (training on the mixed and separated datasets). The three networks correspond to different weights of the loss function. The percentages show the variation with regards to the noisy image, a negative variation being desirable for MAE and RMSE, and a positive one being desirable for SSIM.

SINGLE DATASET							
Network	Sequence	MAE (ANN)	MAE (NLM)	RMSE (ANN)	RMSE (NLM)	SSIM (ANN)	SSIM (NLM)
SSIM = 0.95 MAE = 0.05	Spin-Echo	-21%	-22%	-29%	-23%	0%	8%
	STEAM	-13%	-26%	-17%	-35%	2%	4%
SSIM = 0.9 MAE=0.1	Spin-Echo	0%	-22%	4%	-22%	2%	9%
	STEAM	20%	-26%	47%	-35%	3%	5%
SSIM = 0 MAE=1 (res.)	Spin-Echo	6%	-22%	-2%	-23%	-14%	8%
	STEAM	45%	-25%	65%	-35%	-16%	4%
SEPARATED SPIN ECHO AND STEAM DATASETS							
Network	Sequence	MAE (ANN)	MAE (NLM)	RMSE (ANN)	RMSE (NLM)	SSIM (ANN)	SSIM (NLM)
SSIM = 0.95 MAE = 0.05	Spin-Echo	-17%	-22%	-44%	-24%	8%	8%
	STEAM	-1%	-25%	12%	-35%	-15%	4%
SSIM = 0.9 MAE=0.1	Spin-Echo	-15%	-21%	-47%	-19%	9%	7%
	STEAM	-6%	-25%	-9%	-35%	1%	4%
SSIM = 0 MAE=1 (res.)	Spin-Echo	-22%	-22%	-33%	-22%	15%	9%
	STEAM	-6%	-26%	7%	-35%	9%	5%

Table D4. Distance between the image quality metrics (MAE, RMSE, SSIM) of the U-NET (ANN) and NLM output and the noisy images, for a pool of 9 different datasets (18 images per dataset) in each case. The above part corresponds to the training on the mixed SE-STEAM dataset, and the below part corresponds to the SE and STEAM separate trainings.

Table D5 summarizes the results obtained with the Diffusion Data Analysis Tool for the datasets denoised with both NLM and the U-NET neural networks. We calculated the mean relative error with regards to the ground truth for each metric and calculated the percentage of times that the network outperformed the NLM algorithm (% Scores). We had to do an exception with the "% of negative eigenvalues": instead of calculating the relative error between the ground truth value and the denoised value in each case and then average all the cases (phase, sequence and subject), we calculated the mean % of negative eigenvalues for all phases and subjects and then calculated the relative error between the ground truth mean and the denoised mean. The reason of that is that the standard deviation of the relative error for this figure was sometimes above the hundred (for the U-NET rather than the Y-NET), caused by a few cases in which the percentage of the network output tended to be more than 3 times the percentage of the ground truth.

We also assumed that the results obtained with the Non-Local Means algorithm are constant along each training (with both U-NET and Y-NET).

SINGLE DATASET								
	SSIM = 0.95 MAE = 0.05		SSIM = 0.9 MAE = 0.1		SSIM = 0 MAE = 1 (res)		NLM	
	Spin-Echo	STEAM	Spin-Echo	STEAM	Spin-Echo	STEAM	Spin-Echo	STEAM
Fractional Anisotropy (FA)	12%	19%	11%	7%	27%	34%	9%	3%
Mean Diffusivity (MD)	11%	29%	14%	20%	19%	39%	11%	19%
Helix Angle (ENDO)	16%	18%	18%	8%	18%	16%	17%	6%
Helix Angle (EPI)	31%	20%	22%	21%	27%	15%	20%	11%
Lp Gradient	41%	30%	52%	37%	55%	27%	34%	44%
Median E2A	26%	21%	21%	12%	20%	12%	18%	11%
Variation of mean %Negative Eigenvalues	-13%	10%	33%	191%	352%	676%	80%	81%
Mean error (excl. Eigenvalues)	23%	23%	23%	18%	28%	24%	18%	16%
Error standard deviation (excl. Eigenval	11%	5%	14%	10%	13%	10%	8%	14%
% Scores*	40%	25%	31%	27%	49%	54%		
SEPARATED SPIN ECHO AND STEAM DATASETS								
	SSIM = 0.95 MAE = 0.05		SSIM = 0.9 MAE = 0.1		SSIM = 0 MAE = 1 (res)		NLM	
	Spin-Echo	STEAM	Spin-Echo	STEAM	Spin-Echo	STEAM	Spin-Echo	STEAM
Fractional Anisotropy (FA)	9%	8%	11%	7%	12%	5%	9%	3%
Mean Diffusivity (MD)	12%	19%	14%	20%	12%	20%	11%	19%
Helix Angle (ENDO)	13%	14%	18%	8%	17%	9%	17%	6%
Helix Angle (EPI)	21%	22%	22%	21%	17%	18%	20%	11%
Lp Gradient	36%	52%	52%	37%	23%	30%	34%	44%
Median E2A	19%	21%	21%	12%	20%	13%	18%	11%
Variation of mean %Negative Eigenvalues	-3%	33%	-10%	-21%	20%	5%	80%	81%
Mean error (excl. Eigenvalues)	18%	23%	23%	18%	17%	16%	18%	16%
Error standard deviation (excl. Eigenval	9%	14%	14%	10%	4%	8%	8%	14%
% Scores*	63%	40%	49%	46%	49%	46%		
*NLM scores are not displayed because they are relative to each ANN score and therefore, changes depending on the latter							NLM Values do not undergo changes with different networks, which is why we only display one set	

Table D5. Table illustrating the average Spin-Echo and STEAM results for the Diffusion Data Analysis Tool. The percentages correspond to the mean distance (average of results for all 9 cases) between the ground truth and the result from each trained U-NET (ANN) and the NLM filtering. For the variation of mean % of negative eigenvalues, we averaged the percentages along the 9 cases and then calculated the relative distance. The %Scores indicates the mean number of times (all cases and metrics pooled) in which ANN ouperformed NLM.

Tables D6 and D7 display the results obtained for the Y-NET training for the three loss function we tested. The tables follow the same format of D4 and D5 respectively. Again, the results for the 3 different loss function weights are portrayed.

SINGLE DATASET							
Network	Sequence	MAE (ANN)	MAE (NLM)	RMSE (ANN)	RMSE (NLM	SSIM (ANN)	SSIM (NLM)
SSIM = 0.95 MAE = 0.05	Spin-Echo	-25%	-25%	-56%	-21%	12%	7%
	STEAM	9%	-25%	13%	-35%	2%	3%
SSIM = 0.9 MAE=0.1	Spin-Echo	-92%	-26%	-36%	-22%	-3%	8%
	STEAM	-2%	-26%	4%	-36%	3%	3%
SSIM = 0 MAE=1 (res.)	Spin-Echo	-26%	-21%	-53%	-21%	18%	7%
	STEAM	-21%	-25%	-49%	-34%	7%	4%

Table D6. Distance between the image quality metrics (MAE, RMSE, SSIM) of the Y-NET (ANN) and NLM output and the noisy images, for a the same test dataset used for the U-NET (see Table 3).

	SSIM = 0.95 MAE = 0.05		SSIM = 0.9 MAE = 0.1		SSIM = 0 MAE = 1 (res)		NLM	
	Spin-Echo	STEAM	Spin-Echo	STEAM	Spin-Echo	STEAM	Spin-Echo	STEAM
Fractional Anisotropy (FA)	10%	7%	10%	8%	8%	9%	7%	3%
Mean Diffusivity (MD)	10%	24%	11%	20%	11%	17%	9%	19%
Helix Angle (ENDO)	18%	9%	13%	13%	13%	11%	16%	6%
Helix Angle (EPI)	24%	20%	19%	25%	20%	20%	19%	11%
Lp Gradient	38%	23%	35%	30%	29%	27%	32%	44%
Median E2A	21%	22%	16%	9%	14%	14%	15%	11%
Variation of mean %Negative Eigenvalues	-30%	-29%	-20%	-38%	0%	-56%	87%	16%
Mean error (excl. Eigenvalues)	20%	18%	17%	18%	16%	16%	16%	16%
Error standard deviation (excl. Eigenvalues)	10%	7%	8%	8%	7%	6%	8%	14%
% Scores*	59%	48%	51%	41%	56%	43%		
Mean error (excl. Eigenvalues)	19%		17%		16%		16%	
Error standard deviation (excl. Eigenvalues)	8%		8%		6%		11%	
% Scores*	54%		46%		50%			
*NLM scores are not displayed because they are relative to each ANN score and therefore, changes depending on the latter							NLM Values do not undergo changes with different networks, which is why we only display one set	

Table D7: Relative distances between the Y-NET and NLM data DTI metrics and the ground truth DTI metrics (the dataset is the same referenced in Table 4). For the variation of mean % of negative eigenvalues, we averaged the percentages along all cases and then calculated the relative distance. The % Scores indicates the mean number of times (all cases and metrics pooled) in which Y-NET ouperformed NLM.

Figure D3 shows the U-NET, Y-NET and NLM comparison for the $SSIM_{95}MAE_5$ network.

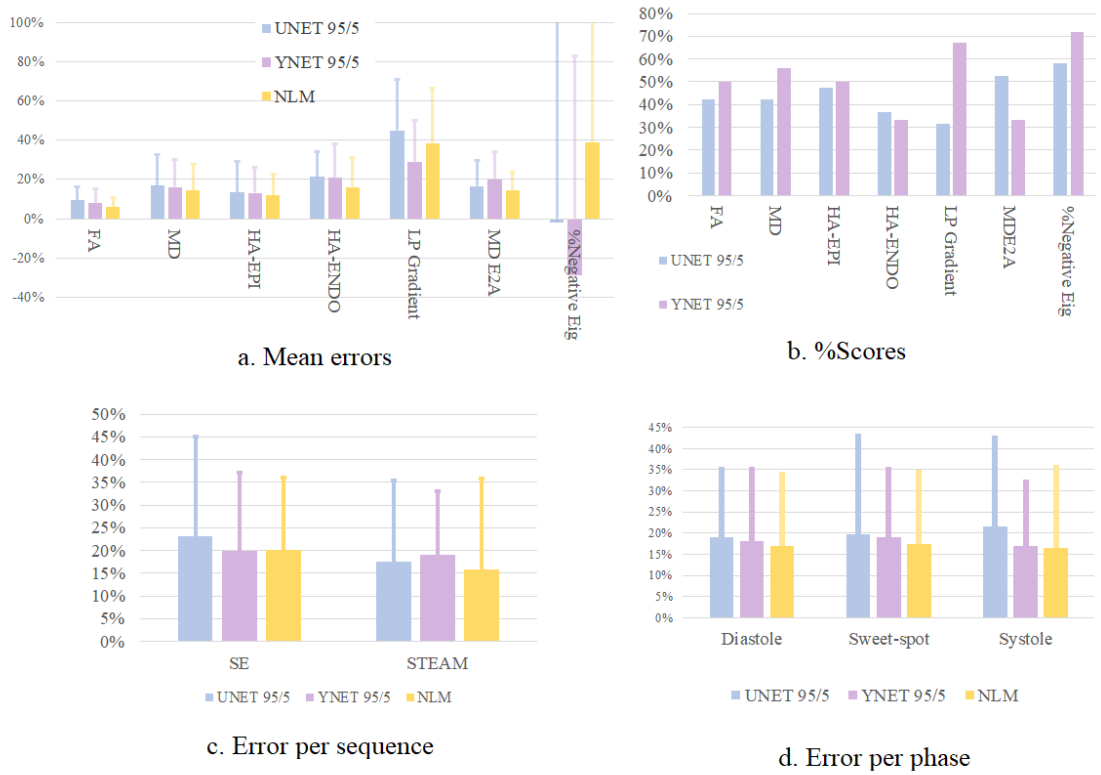


Figure D3. a) Mean relative error obtained for each DTI parameter related to the ground truth output for the $SSIM_{95}MAE_5$ U-NET, the $SSIM_{95}MAE_5$ Y-NET and the NLM algorithm; b) Percentage of scores yielded by the $SSIM_{95}MAE_5$ U-NET and Y-NET, meaning the mean number of times the networks outperformed NLM in each category; c) Mean error for each sequence of the $SSIM_{95}MAE_5$ U-NET and Y-NET and the NLM algorithm; d) Mean error for each cardiac phase

Whereas the behaviour for the sequence-specific and phase-specific errors are consistent with those of the $SSIM_0MAE_1RES$ network, we can see that in this case, the Y-NET outperforms the U-NET

in the Lp and helix angle assessment. The error obtained with the Y-NET is similar to that obtained with the $SSIM_0MAE_1RES$ U-NET, whereas the $SSIM_{95}MAE_5$ U-NET has a much higher error.

Figure D4 displays a comparison between the U-NET, Y-NET and NLM performance of $SSIM_0MAE_1RES$ in the different DTI parameters, for each cardiac phase.

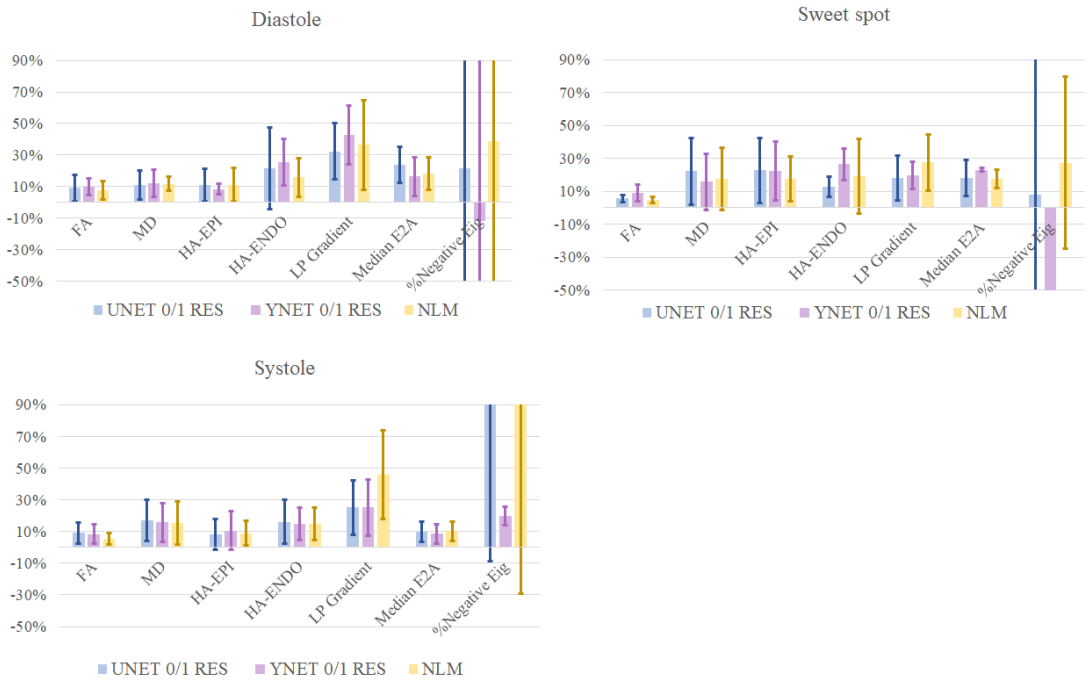


Figure D4. Comparison of the performance of $SSIM_0MAE_1RES$ U-NET (blue), Y-NET (purple) and NLM (yellow) in the different DTI categories approached, for each cardiac phase.

We can see that Y-NET tends to outperform the other two in systole, whereas U-NET and NLM appear more effective in the Sweet-Spot phase.

Figure D5 contains a sequence comparison similar to figure D4. We can see that whereas the error in STEAM is lower, Y-NET seems to outperform U-NET performance in SE sequence.

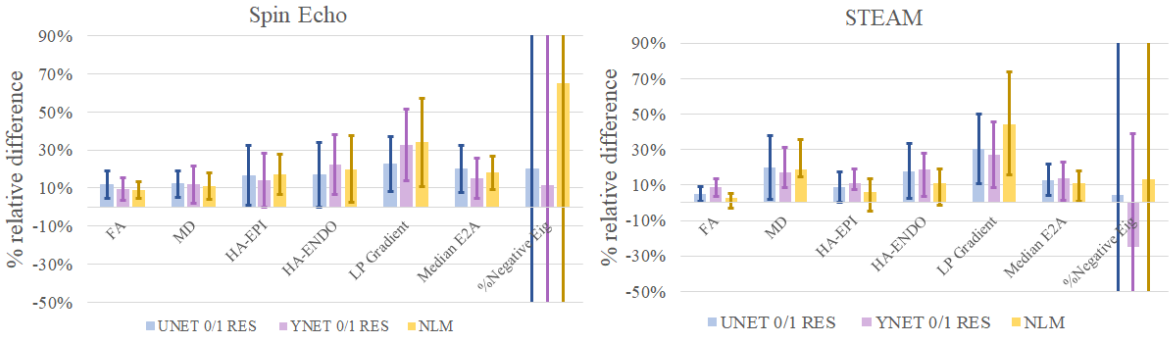


Figure D5: Comparison of the performance of $SSIM_0MAE_1RES$ U-NET (blue), Y-NET (purple) and NLM (yellow) in the different DTI categories approached, for each sequence.

4. Multiple Averages

We tuned NLM function (see annex B) to be optimized for the different denoising ratios we used (1:2, 1:4, and 1:6). Despite that, we can see in Figure D6 that NLM is more and more challenged as we increase the number of averages.

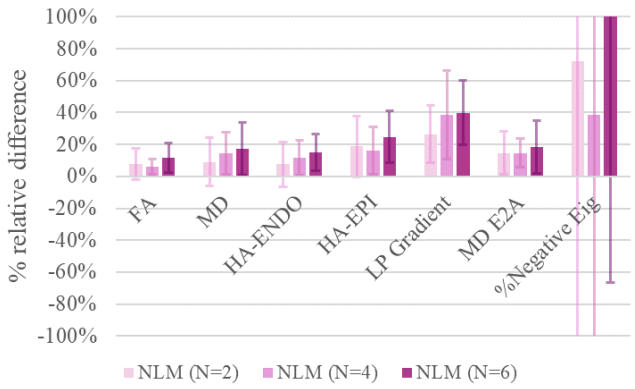


Figure D6. Mean relative difference output by NLM for the different DTI parameters considered, as we increment the number of averages (N) of the ground truth.

Figure D7 compares the DTI parameters for NLM and U-NET tuned/trained on denoising ratios of 1:2 (Figure D7a) and 1:4 (Figure D7b). We removed the % of negative eigenvalues, as the differences between our two ratios in the rest of the metrics are small and would not be perceived if the % of negative eigenvalues was included (the scale would be too big). Although with 1:2, we see a small increase in the error committed by U-NET on one of the helix angles, and that affects sensibly the Lp Gradient measure, we can see a reduction of the error for the MD, FA, E2A Median (although very small), and the other HA. For the U-NET, the difference in the percentage of negative eigenvalues stays the same for both ratios, although it increases severely from 12% to 65% with N=6. NLM has considerably bigger differences, and still gets its best value for N=4 (108%, 39% and 72% for N=2, 4 and 6 respectively). A statistical test should be run to check whether there is a significant difference between ratios 1:2 and 1:4.

Visually, the results obtained are better with 1:2 (see figure 22). We wonder if that could also be a consequence of handling 4 (2-averages) in the Diffusion Data Analysis Tool, rather than 2 (4-averages).

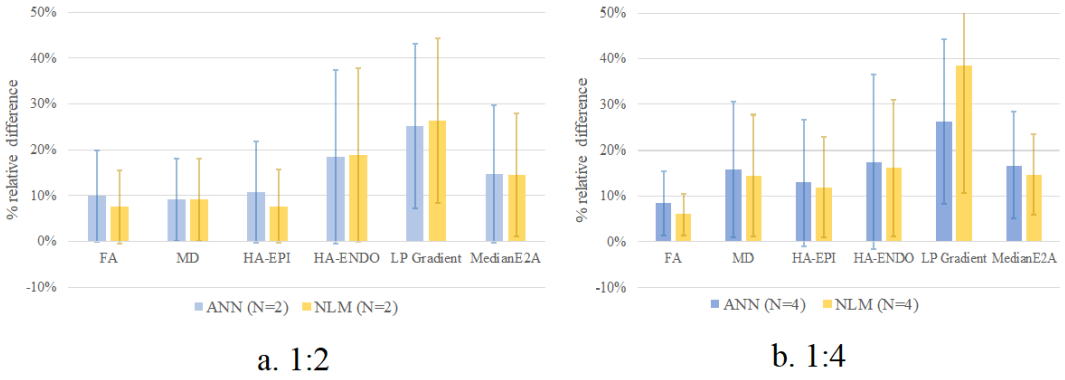


Figure D7. Comparison of the relative differences in the DTI values (except for the % of negative eigenvalues) between ANN and NLM a) optimized for a denoising ratio of 1:2; b) optimized for a denoising ratio of 1:4.

Annex E. Script structure

Our project repository contains the following folders:

- Pre-processing.
- Dataset processing; included sub-folders: Registration, Automated Heart Detector.
- U-NET; included sub-folders: Gaussian trainings, Retrained from Gaussian.
- Y-NET
- Post-processing

This annex aims to explain in detail the scripts we've developed, in case all or some of them are to be taken over in future work. All the scripts were written using MATLAB 2019a. Although rare, there are some incompatibilities with previous versions of MATLAB 2019a that we have outlined.

1. Pre-processing of the dataset

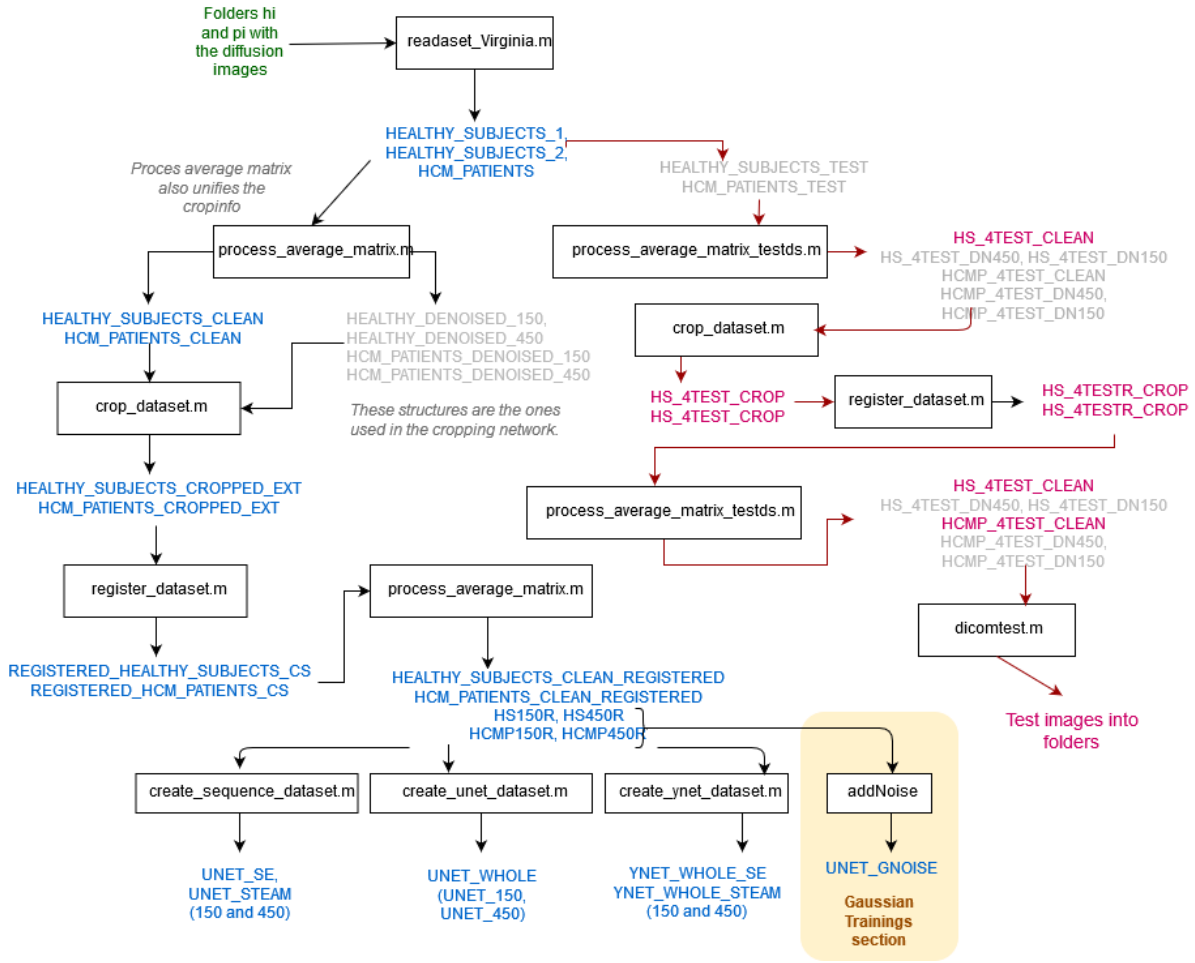
In this folder there are three sub-folders:

- **Dataset reporting**: contains *count_averages.m*, explained in Annex A1.
- **NLM_tuning**: contains *NLM_reporting.m* explained in Annex B, as well as the figures output by the program. To run this program, you need to import your data to the workspace. This data structure is that output by the programs listed in table E1. The variable *multiple_degrees* contains all the parameter values that the program is to test. It can be the Comparison Window Size, Search Window Size and Degree of Smoothing Factor (one at a time). The default ones are 15, 3 and 1 respectively. To output proper figures, the titles of the x-axis must be manually changed every time the parameter is switched. You can also specify the size of the sub-dataset that is tuned. This folder also contains function *degreeOfSmoothing.m*, embedded in *NLM_reporting.m* as well as in the *process_results.m* scripts of the network folders. This function calculates the degree of smoothing that is internally calculated by *imnlmfilt* when no 'Degree of Smoothing' is specified (see annex B for more details).
- **Analysis of the noise of the dataset**: this folder contains *noise_reporting.m*. This function takes as argument the sequence-specific datasets for U-NET (see table E1) and outputs the noise distributions for each sequence and phase, as well as the global distribution, fit to a Gaussian function. Under the section "Global parameters" of the script, you can control the size of the pooled sub-datasets, as well as the number of phase-specific images you want to pool separately. This program uses *histcounts*, as it allows to add the number of intensity counts across all images into the same structure, without the need to get an image average. The bin values (called "Edges") are those of the first image to be processed. Because the images are randomly selected, the first image is not always the same, and the "Edges" structures change every time you run the program. Although there are no big differences in the noise patterns, some images have more values within a certain range than others, and this causes the histogram to present – sometimes – almost empty bins if the first image has more sparse values than the rest. It is recommendable to run this program multiple times and save the best global histogram, and also play with the bin size to avoid empty bins in the count.

2. Processing of the dataset

This section offers an overview of how our training datasets are created, explaining step by step the different programs used and how they transform the data. Figure E1 is a representation of this pipeline.

Figure E1. Flow diagram of the dataset processing scripts. The squares contain the name of the scripts, and the blue text represents the relevant data structures. The grey structures are not relevant for the project. The red path related to the test dataset, processed separately, the red text pointing out the relevant structures.



a) *From the DICOM data to a Matlab Structure*

Our dataset is created from the output of program *readaset_Virginia.m* (Dataset Processing). This script organizes the diffusion data by calling script *organise_diff_dicoms.m* (author: P. Ferreira) for every subject, phase and sequence.

The outputs of *readaset_Virginia, process_average_matrix and process_average_matrix_testds* (the ones ending in _CLEAN) *crop_dataset* and *register_dataset* have the navigable structure outlined in Figure E2. The denoised structures output by *process_average_matrix* and *process_average_matrix_testds* have also a navigable structure, but lack the average matrix (as the bad images have already been removed), the crop information (as the images are already cropped).

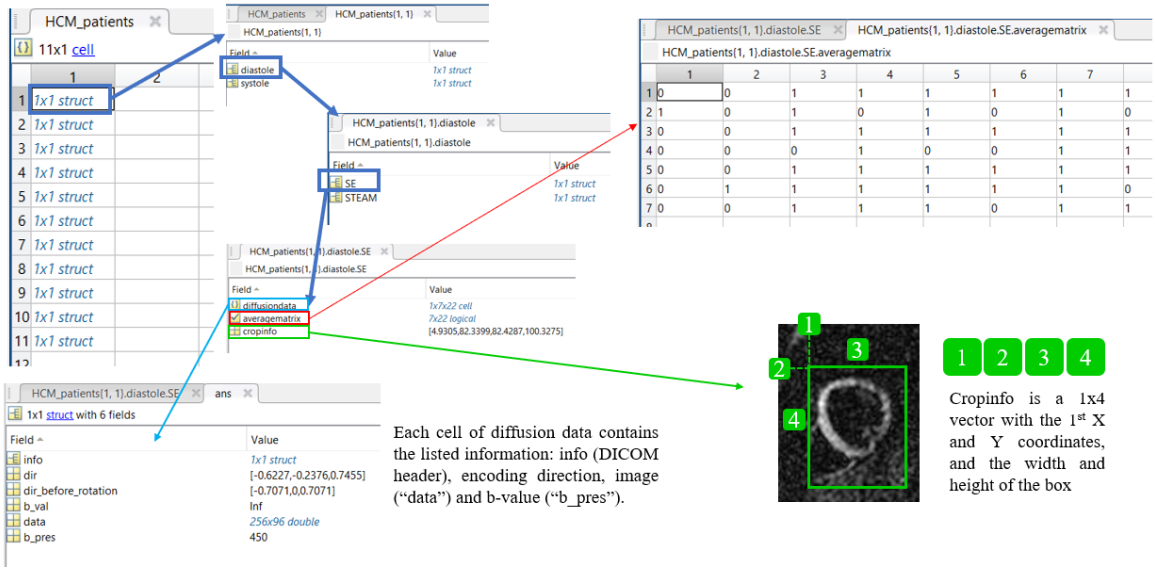


Figure E2. Structure of the subjects datasets. The upper-left part shows how we navigate in a subject structure (there are 11 patients in this case, and for each one, two phases, and for each, two sequences). Each sequence contains a *diffusiondata* structure, that we usually call "sub-dataset" along the report, with the content listed in the lower-left part of the figure, the average matrix, with a number per position of the *diffusiondata* structure, with "0" when the image is too noisy, and "1" when it's valid for use, and the *cropinfo* structure, which contains the coordinates and size of the manually delimited bounding box.

b) Creation of the ground truth

The program *process_average_matrix* (and its equivalent for the test dataset) removes the images that are labelled with 0 in the average matrix and creates the ground truth by averaging N images indicated at the beginning of the script. For a specific sub-dataset, all images belonging to encoding directions that do not have at least one average (N valid images), are also removed from the clean dataset. This script distinguishes as well between b-value=450 and b-value=150. Because each SE datasets generally contained only two averages with b-value=150, and STEAM only one, those images were generally ruled out from the clean dataset. The script *process_average_matrix_testds* differs from *process_average_matrix* in that it keeps noisy images with b-value = 150 untouched in the clean dataset. The reason behind this is that the Diffusion Analysis Data Tool required these 6 b=150 directions and the reference. We made sure as well that the average matrix did not have "0"s for these averages.

This script generated a clean dataset with our noisy images, and 2 datasets containing the averaged denoised images that would serve as ground truth (there were two structures, one for b=450, the other for b=150, although the latter did scarcely have data). The link between each noisy image and its denoised counterpart was an integer that identifies the subject, phase, sequence, direction, b-value and denoised-average a specific image is part of. That 9-digit integer, or tag (Figure E3), is added to the cell structure containing the image.

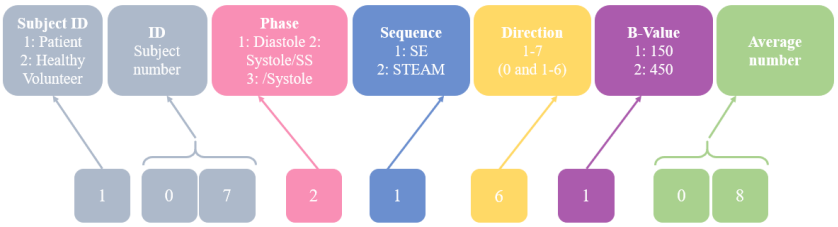


Figure E3. Structure of the tag. Each square represents a digit, the rightest ones (green) being the units.

Script *process_tag.m* retrieves all the information and is used by all the scripts that generate the training datasets for our neural networks.

c) Cropping

As mentioned in the report, we attempted an automatic heart detector based on a YOLO neural network, but due to the unsatisfactory results, we decided to crop the images based on the manually delimited bounding box. Annex C provides details of how YOLO was implemented. This section is meant to explain the program that was actually used to do the cropping, *crop_dataset.m*, as well as the structure of the 'Automatic Heart Detector' folder containing the YOLO implementation.

The script *crop_dataset.m* finds the maximum width and height along all the datasets (patients and healthy volunteers) and selects it as fixed size for the bounding box. This was necessary, since neural networks need a fixed input size.

Notice that in figure E1, *process_average_matrix* needs to be executed before and after the cropping. This is done because a few images were rotated by 90°, and *process_average_matrix* took care of setting them right and adjust the *cropinfo* structure as well. We did not include this task in *crop_dataset.m* because we created these scripts before the YOLO implementation. As YOLO would have needed the uncropped data as input, we needed to take care of this issue before the cropping.

YOLO was trained on the ground truth images (as they have more SNR) of size 256x96. To implement it, we followed the steps of MATLAB article "Create YOLO v2 Object Detection Network" (https://www.mathworks.com/help/vision/ug/create-yolo-v2-object-detection-network.html).

This article includes a series of functions belonging to MathWorks ®: *findLayerToReplace, createLgraphUsingConnections, freezeWeights*.

A pre-trained inceptionv3 network was imported and saved on the repository. This network was trained on "heart" and "no heart" images to be able to classify our images in two categories. The "non heart" images were downloaded at Medpix (https://medpix.nlm.nih.gov/topiclist) and manually cropped like the images on our dataset (256x96). They are stored in the "NON-HEART-IMAGES" folder in the "Classification Network" sub-folder.

Figure E4 shows the main steps of this section. The inceptionv3 network has too many parameters. We decided to remove a great part of its middle feature extraction layers, leaving only 96 layers and the final classification layers. The network was trained on a dataset created with function *create_classification_datastore.m*, that creates the relevant structures from a folder containing a "heart" folder and an "others" folder with the images. These folders are created with *store_images.m*.

The YOLO dataset is created with a program named *crop_denoised_dataset_YOLO.m*, that takes as input the denoised output of *process_average_matrix.m.* The output, stored in structure *yolo_dataset.m* is a Nx3 structure with:

- The uncropped image
- The bounding box (X1, Y1, Width, Height) with (X1, Y1) coordinates of the upper-left corner

- The sequence

This program calls program *image2inception.m* that rescales the images into a 299x299x3 size, mandatory for the *inceptionv3* network. The coordinates of the bounding box are adapted accordingly.

The program *generateYOLOdataset.m* stores the images from the resulting structure in folders and creates a groundTruth structure as required by YOLO, containing the image paths and the bounding boxes.

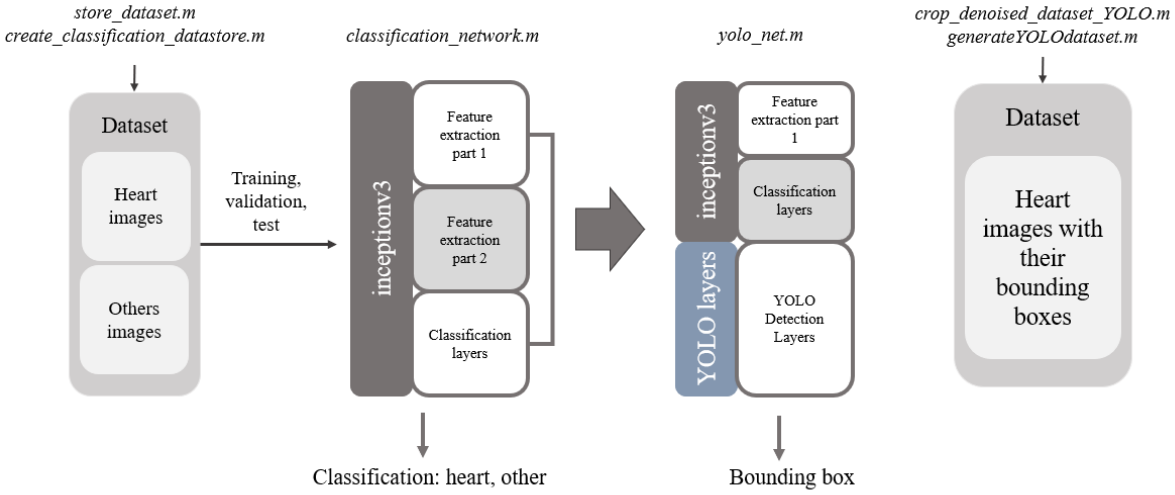


Figure E4. The middle diagrams show how the inceptionv3 network is processed by *classification_network*.m, and trained with the dataset created with *store_dataset.m* and *classification_network.m*. The resulting network is processed by *yolo_net.m*, where the final detection layers are added and the resulting network is trained on the dataset created by *crop_denoised_dataset_YOLO* and *generateYOLOdataset.m*.

The resulting classification network is named *"classification_network.mat*", and the resulting YOLO network is named "*trained_yolo.mat*". Its "info" file is also available under "*info_yolo.mat".*

Notice: if YOLO had worked, further scripts should have been written to: infer the bounding-box values for a 296x96 image from those extracted by the network for size 299x299, modified all the *cropinfo* of our original structure, and the re-run *crop_dataset.m.*

d) Registration

The script *register_dataset* is contained in the "*Registration*" folder. This script uses a DFT-based algorithm [9] implemented in program *dftregistration.m* (*author: M. Guizar Sicairos*). The program was modified by us to multiply the DFT with a 2-D Gaussian function. Our program *register_dataset*, takes a subject structure and performs a within-sequence and then a cross-sequence registration. The output preserves the original structure, but the images have been registered; both the within and the cross-sequence registered outputs are available. Figure E5 shows how the program uses the within-sequence registered data to set up the cross-sequence registration.

In addition to the registered datasets, the argument 'displaycode' allows you to choose a subject number and phase to output two movies playable with *implay* that stack the registered images for both sequences. A cross-registration movie shows the same results for all the subjects and a specific phase.

The reference image for the within-sequence registration can be either the first average, or the first image with b=150 the script can find. That is customizable by the user.

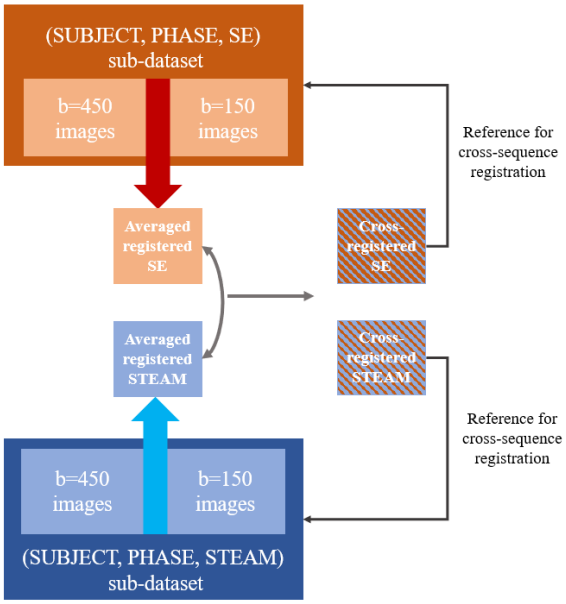


Figure E5. The within registration is performed within each sub-dataset, and then the average of the sub-dataset of each-sequence is registered with each other. The resulting registered averages are used to register the dataset again.

e) Generation of the network datasets

The datasets we used for all U-NETs and Y-NETs are cell-structures with N rows (N = number of images) and 3 columns (table E2).

Dataset purpose	**Script**	**First column (noisy image)**	**Second column (denoised image)**	**Third column**
Training of U-NET on Gaussian noise.	*addNoise.m* Receives the denoised 450 or 150 datasets as input.	Averaged image with a Gaussian noise with varying σ ([0.01; 0.001]. Size: 120x88x1	Same averaged image without the Gaussian noise. Size: 120x88x1	Tag
Training of U-NET on real data (mixed datasets).	*create_unet_dataset.m* Receives the clean "noisy" data and the 450 or 150 datasets as input. Stacks SE and STEAM data.	Noisy image. Size: 120x88x1	Averaged image. Size: 120x88x1	Tag
Training of U-NET on real data *(separate datasets)	*create_sequence_dataset.m* Receives the clean "noisy" data and the 450 or 150 datasets as input. Outputs SE and STEAM data.	Noisy image. Size: 120x88x1	Averaged image. Size: 120x88x1	Tag
Training of Y-NET on real data	*create_ynet_dataset.m* Outputs the SE and STEAM datasets.	Noisy image. Size: 120x88x3 (SE-STEAM-zeros or STEAM-SE-zeros stacked)	Averaged image. Size: 120x88x1 (SE or STEAM, sequence equivalent to channel 1 of the input)	Tag
Training of Y-NET on Gaussian noise	*addNoiseYnet.m* Outputs the SE and STEAM datasets.	Averaged image with Gaussian Noise. Size: 120x88x3 (SE-STEAM-zeros or STEAM-SE-zeros stacked)	Same averaged image without the Gaussian noise. Size: 120x88x1 (sequence equivalent to channel 1 of the input)	Tag

Table E1. Different scripts used to generate datasets fitting into the U-NET / Y-NET trainings.

**Script generateSequenceDataset.m can extract the sequence-specific datasets from the mixed one*

For Y-NET, the program needs to concatenate the SE and STEAM images. Because all images are then saved as .png or .jpg files, and they must either be 1 or 3 channels, it was necessary to add a channel of zeros at the end that is not used.

List of created structures:

- UNET_WHOLE: Mixed SE and STEAM datasets.
- YNET_WHOLE_SE, YNET_WHOLE_STEAM: SE and STEAM datasets for the Y-NET. The first contains the SE image in the first channel of the noisy image, whereas the second contains the STEAM image.
- UNET_ SE, UNET_STEAM: SE and STEAM datasets for the U-NET training on separate datasets.
- UNET_WHOLE_2, UNET_WHOLE_6, UNET_SE_2, UNET_SE_6, UNET_STEAM_2, UNET_STEAM_6: Same structures for denoising ratios 1:2 and 1:6.
- UNET_GNOISE: Trained on Gaussian noise added to the denoised images.

3. Neural Networks

This section overviews the scripts used to deploy and train the U-NET and Y-NET networks. Most of the files and structures are similar for U-NET and Y-NET. Thus, we will explain them once, writing a slash between the U-NET and Y-NET programs when they are mentioned.

From the previously mentioned structures, programs *generate_unet1_datastore.m* / *generate_ynet1_datastore.m* generate a ImageDatastore object that MATLAB uses to access images stored in folders. It was necessary to do so, since MATLAB consumed to much memory during the training and could not afford having such big structures in the Workspace.

Before that, program *zerobelow0.m* must be used to shuffle the dataset and eliminate all images that contain negative intensities. It also transforms the values to *uint*, necessary for the later storage in folders.

These programs perform data augmentation, store the data into folders, and generate these Datastores that are then used in the training scripts. They are called by the training programs *train_unet1.m* and *train_ynet.m*. The storage in folders is done once only. If an overwrite of the images is needed, you must manually delete the folders. Figure E6 contains an overview of the steps followed by *generate_unet1_datastore.m*.

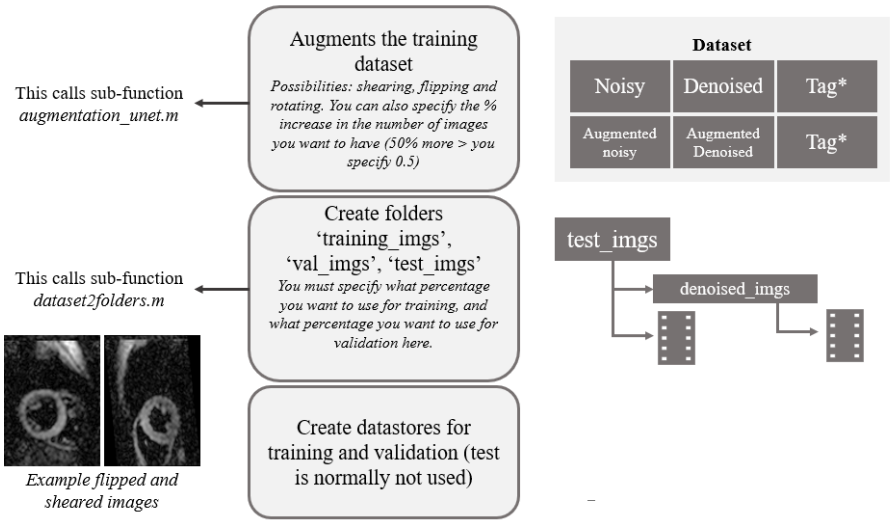


Figure E6: Scheme of the steps undertaken by *generate_unet1_datastore* (same for *ynet*). The steps are the grey rectangles of the center. The call sub-functions specified at the left, and modify the structures and folder systems as shown at the right.

The tag is modified during data augmentation: an additional index is added to identify if it's the original image "0" or if it has been flipped ("1"), sheared ("2") or rotated ("3").

We used the program *train_unet1.m* to train the U-NET on Gaussian noise. All the necessary scripts are stored in folder "TRAIN_SCRIPTS" of the U-NET – GAUSSIAN TRAININGS sub-section. Then, program *retrain_unet1.m* is used on the real datasets. These programs perform the folder storage as well, but do not contain the section where the network is created, as the pre-trained network is passed as an argument.

Program *rerun_unet1.m* imports the pre-trained function and the data, and calls *retrain_unet1.m*. Modifying the network and data name in this program is the easiest way to launch a training on the real dataset.

Program *train_unet1.m* is incompatible with previous versions of MATLAB, as the transposed convolution layer *transposedConv2Layer.m* was modified from 2018 to 2019.

For the Y-NET, we've got a file for SE trainings and a file for STEAM trainings. The reason is that the internal structure of the network changes depending on whether we are in a case or the other (the concatenation layers of the U-NET sub-structure must be between the main down sampling branch and the up sampling branch, and this changes from one case to the other). Therefore, to train the SE,STEAM:SE Y-NET, the file is *train_ynet_SE.m*. To train the STEAM,SE:STEAM Y-NET, the file is *train_ynet_STEAM.m*.

All the training scripts call a function named *unet1_datastore.m* / *ynet1_datastore.m* that read the ImageDatastore structures and create tables during the training in a dynamic way. This script controls whether the training is performed on the residuals (called *resid*) or on the actual images. The variable *data* created during function *read* contains a table with the noisy images, and a table with either the residuals or the denoised images, that can be modified by the user before the training.

The other important function named by our training scripts is *unet1output.m*. This is the output layer containing our forward and backpropagation functions. The loss function weights are specified both in the forward and in the back functions under "*weights_MAE"* and "*weights_SSIM"*, and can also be modified before the training. This layer was implemented thanks to MATLAB article "Custom Regression Layer" (https://www.mathworks.com/help/deeplearning/ug/define-custom-regression-output-layer.html).

Notice that when loading a pre-trained network, the *unet1output.m* file must be stored in the same folder as the network, otherwise it is not loaded properly.

Folder U-NET contains the following data:

- **GAUSSIAN_TRAININGS** folder. In here, you have:
  - ✓ Folder **TRAIN_SCRIPTS**, with the Gaussian-noisy dataset, and all the scripts necessary to run a training on it.
  - ✓ Folder **SSIMAE**, with the different loss-function short-trainings and the results.
  - ✓ Folder **PARAMETERS_TUNING**, with the different training-parameters short trainings and the results.
- **RETRAIN_SCRIPTS** folder, with the scripts necessary to run a re-training on the real dataset (the dataset is still missing, and needs to be input in program *rerun_unet.m*).
- **RETRAINED_NETWORKS** folder, with our functions retrained on real data, and the results obtained.

The structure of the results is outlined in section 4 of this annex. For Y-NET, we have a:

- **TRAIN_YSE** folder, with all the scripts necessary to run a training of the SE,STEAM:SE Y-NET.

- **TRAIN_YSTEAM** folder, with all the scripts necessary to run a straining of the STEAM, SE:STEAM Y-NET.
- **TRAINED_YNETS** folder, with the results of our trainings. Notice that, even though we have trained two networks to process our SE and STEAM test datasets, the result folders are shared by both Y-NETS, as at some point, we pool the results obtained for the STEAM and SE test datasets.

4. Test

The results folders contain sub-folders for the different loss function weights or parameters used. For each of those, we've got multiple test sub-folders with the different test cases. Annex A2 contains an explanation of how the test images folders are structured and how the scripts for test results processing named *process_results_integrated* work. There are various scripts that perform mostly the same tasks and are listed in Table E2.

Name of the file	Purpose	Important differences
process_results_gauss.m	Processes a single test sub-dataset.	Several prompts will ask you to manually select the denoised and storage folders. Receives only the "Denoised" folders because Gaussian noise is added on them for testing.
process_results_gauss_integrated.m	Processes all the specified test datasets	No noisy folder must be specified, as the noise (Gaussian) is added to the denoised files.
process_results.m	Processes a single dataset.	Several prompts will ask you to manually select the noisy, denoised and storage folders. Multiple noisy folders can be selected here, but manually.
process_results_integrated_SE.m	Processes all the SE datasets specified.	Meant for Spin-Echo folders. You need to change the names manually.
process_results_integrated_STEAM.m	Processes all the STEAM datasets specified.	Meant for STEAM folders. You need to change the names manually.
process_results_y_integrated.m	Processes all the STEAM and SE datasets specified.	Imports the SE,STEAM:SE and STEAM,SE:STEAM networks and processes both SE and STEAM datasets.

Table E2. List of the different *process_results* files used in this project, and their purpose and requirements.

These scripts process, as well, the *info* structures of the neural networks, and plot the Training Loss and Validation, smoothening the curves to reduce the peaks.

Once the Diffusion Analysis Data Tool has been executed for every test case, and every folder contains a *result_images* folder created by the tool, we can run script *exportdata2excel.m*. This script takes the relevant DTI metrics from each test folder contained in the folder selected by a prompt during the program and stores them in sheet 2 of a Microsoft Excel ® file that must be in that folder under name "BILAN.xls". This script assumes that all the Excels have the data in the exact same cells. Excel file "BILAN_BLANK.xls" in the TRAINED_YNETS and RETRAINED_NETWORKS are empty templates that can be copied into each results folder to work with *exportdata2excel.xls.* The ground truth raw data stored in folders named *test_imgs* (see Annex A2) must be entered manually into the Excel. Once the raw data is stored, the formulas in the Excel automatically calculate the relative differences between the denoised data and the ground truth.

Additionally, users must manually:

- Enter the image-wise quality metrics in the first sheet. This can be easily done with the Import Text Wizard tool, from the .txt.files stored in the results folder (see Annex A2).

- Compute the Scores in sheet 2. The %Scores are another measure besides the relative differences, that basically assess how many times (over all cases) the network outperforms NLM or NLM outperforms the network. This is done by looking at the relative differences.
- Compute the % of Negative Eigenvalues index. This is the next-to-last column of the negative differences table and contains 100% every time the % of negative eigenvalues is over that of the ground truth, and 0% every time it's equal or lower.

The last sheet of the Excels retrieves the global errors and creates bar charts to display them in a visual way.

During this report, we performed many different pooling and value retrievals, and thus the "BILAN" Excels of some result folders (specially in the case of the 0SSIM1MAERES case) might be slightly different from the template.

g. Post-processing

The "Post-processing" folder was meant to contain the programs for the statistical tests in future work on this project. It contains, as well, script *time_networks.m*. This scripts loads the SE U-NET and SE,STEAM:SE Y-NETs with the different loss function weightings, and calculates the time taken by each network, as well as NLM, to predict a denoised version of a test image stored in the folder.