Applications of Super-Resolution Techniques on Knee MRIs

Kevin Leung and Victor de Murat

May 13, 2018

1 Introduction

This project aims at enhancing the quality of knee MRI images from the Osteoarthritis Initiative using super-resolution methods. Over the past nine years, the Osteoarthritis Initiative (OAI) has been conducting a longitudinal study on knee osteoarthritis by observing 4796 patients. Periodically throughout these nine years, the patients have taken MRI scans over various parts of their knee which were then recorded by OAI. However, these images are of mixed resolution due to a variety of possible reasons, such as differences in scan time from the patient or the use of different MRI scanner settings since OAI used multiple locations across the country to conduct their study.

For our dataset, the resolution[1] on the $z-$axis is very low compared to the resolution on the $x-$ and $y-$axes, by around a factor of 10. So, the initial idea was to use super-resolution methods to recover the resolution on this $z-$axis to get it close to the resolutions on the other axes (in essence, to get isotropic 3-D MR images), but for practical reasons (explained in Section 3.1), we ended up downsampling on the $z-$axis and recovering the resolution to its original point.

In this next section, we introduce the main theoretical aspects of the methods we chose to compare. Next, we will explain the numerical experiments and their corresponding parameter details for the algorithms we implemented. Then, we compare the results of the methods we ran. Afterwards, we comment on the limitations of our experiments. And lastly, we will summarize our findings.

2 Theory Behind Super-Resolution Methods

The main idea of the project was to compare results obtained from an example-based method and a single image, self super-resolution method.

But first, let's introduce the super-resolution problem. Super-resolution is an ill-posed problem: we want to get HR information from LR information. With some simplifications, we can visualize the previous statement. Indeed, if we model the voxels of the LR image as averages of their surrounding voxels in the HR image (plus some noise), we clearly see that for each piece of information y_j we have N unknown pieces x_i:

$$y_j = \frac{1}{N}\sum_{i=1}^{N} x_i + n, \text{ where } n \text{ is the Rician noise}$$

One approach to the LR problem would be to compute a least squares HR estimator, but there will be infinite values meeting this minimization condition. One could then address this issue by adding a regularization term but one consequence of regularization is that it destroys HR patterns. Hence, other methods have to be developed in order to assess correctly the HR information.

[1]Notation: HR = high resolution, LR = low resolution

2.1 Interpolation with Cubic B-Splines

Interpolation is the most intuitive way to get higher resolution images. The idea is indeed to interpolate the actual voxels with some function to get new voxels in between, which results in a higher resolution image.

More formally, the interpolation method we used relied on cubic B-splines defined by the algorithm in [1]:

$$p(x, y, z) = \sum_{i=0}^{3}\sum_{j=0}^{3}\sum_{k=0}^{3} a_{ijk} x^i y^j z^k \tag{1}$$

where we must find the coefficients a_{ijk}. This spline is then used to generate intermediate voxel values to upsample our given low resolution image y_0 to the target resolution.

The main limitation of these simplistic methods is that they are non-local: they only provide a general transformation to compute over the image and so they cannot create new local, high resolution patterns. However, interpolation is used in both of the following super-resolution methods as a starting point.

2.2 Self similarity and Image priors (example-based)

For the example-based method, we followed [2], but also found information of the technique in a more general setting in [3]. This method uses a HR image as reference to help reconstruct a HR image from a LR image. It is an iterative method that aims at converging in a few steps to the HR reference image. The main idea behind it is to apply, at each step of the process, a transformation to each voxel. This transformation is computed via penalization terms reflecting the similarities with the corresponding neighborhood in the HR reference image and the self similarities in the image in reconstruction. Indeed, the underlying assumptions are that 1) similar voxels in the HR reference tend to be similar in the reconstructed data and 2) voxels with similar local context in the reconstructed image should be enhanced.

Now, let's be more concrete about the different steps of the method:

1. Denoising of the LR data and the HR data. By assumption, noise in a MRI is Rician.

2. Interpolating the LR image in order to put it in the same space as the HR reference.

3. Applying the iterative process: at each step t, apply the following transformations for every voxel p of the image in reconstruction:

$$x_p^{t+1} = \frac{1}{C_p} \sum_{\forall q \in \Omega} x_p^t P_1(z_p, z_q, h) P_2(x_p^t, x_q^t, k, h), \text{ where} \tag{2}$$

   - $P_1(z_p, z_q, h) = e^{-\frac{(z_p - z_q)^2}{h^2}}$ is a penalizing term on the similarities in the HR image
   - $P_2(z_p, z_q, k, h) = e^{-\frac{\|N(x_p^t) - N(x_q^t)\|^2}{k.h^2}}$ is a penalizing term on the reconstructed image local contexts
   - Ω is the search area around the voxel x_p^t
   - N the context window for local similarities computations
   - h and k are two parameters controlling the penalization strength
   - C_p is a normalization factor to keep the voxel values within some normalized range.

4. Once this voxels' transformation is achieved, a mean correction step is processed. Indeed, the algorithm imposes the following constraint: the downsampled version of the reconstructed data needs to be equal to the original LR image. In order to respect this constraint, the paper proposes the intuitive transformation:

$$x^{t+1} = x^t - \text{NearestNeighbors(downsampled(reconstructed image) - original)} \tag{3}$$

5. This iteration process ends by adding some tolerance on the improvements achieved at each iteration, such as the mean absolute error between the reconstructed image after two consecutive iterations.

2.3 Self Super-Resolution

2.3.1 Overview of the Method

The other algorithm we implemented is a single image self super-resolution algorithm adapted from [4]. This algorithm first takes an input low-resolution image y_0, upsamples the image using an interpolation method to achieve $\hat{y_0}$, then synthesizes new images $\hat{y_i}$ using a series of rotations, convolutions, and a method known as Anchored Neighborhood Regression (ANR). From these new images $\{\hat{y_0}, \hat{y_1}, \hat{y_2}, ..., \hat{y_N}\}$, we use a technique known as Fourier Burst Accumulation (FBA) that recovers a single high resolution image using a set of images (taken of the same object) that are blurred in independent directions. In the following section, we introduce the intuition behind FBA. Then, we go into the details of the image-generation algorithm that we implemented.

2.3.2 Rationale behind Fourier Burst Accumulation

FBA is a fast algorithm that was demonstrated to work on image deblurring by combining the Fourier coefficients of images (of the same scene), each of which are blurred independently of each other. FBA was first used on images of a specific scene that were blurred due to random hand tremors. Using this set of images, FBA takes the inverse Fourier transform of a weighted average of the Fourier coefficients of the generated images, where the weights depend on the magnitude of the Fourier spectrum. Using FBA, the new high resolution image $\hat{x}$ is created from the following equation:

$$\hat{x} = \mathcal{F}^{-1}\Big[\sum_{i=1}^{N} w_i[k]\mathcal{F}(y_i)[k]\Big]. \tag{4}$$

$\mathcal{F}(y_i)$ and $\mathcal{F}^{-1}(y_i)$ denote the Fourier and inverse Fourier transforms of the image y_i, respectively; k denotes the frequency and w_i denotes the weight for the i^{th} image. The weights w_i controls the contribution of image i at each frequency k and is determined with the follow equation:

$$w_i[k] = \frac{|\mathcal{F}(y_i)[k]|^p}{\sum_{i=1}^{N} |\mathcal{F}(y_i)[k]|^p}. \tag{5}$$

Images with a greater $|\mathcal{F}(y_i)[k]|$ contribute more for that particular frequency in the averaged image. In addition, p is a parameter that controls how much each image contributes in the Fourier domain. If $p = 0$ then the approximated image $\hat{x}$ is just the average (in the Fourier domain) of the input images y_i, while if $p \to \infty$ then for a given frequency it will take the maximum value for the frequency out of all the input images.

Jog et al. took the ideas demonstrated by Delbracio and extended it for the 3-D MRI case. By creating images, each of which are LR in a certain plane but HR in the plane perpendicular to the LR one, then combining these images with FBA, they created higher resolution MR images since each of these generated images contribute new information in different regions of the Fourier domain.

FBA, however, requires multiple images that each contribute different information in the Fourier domain (the algorithm does not work if there is only a single image) so we now explain how we adapted Jog et al.'s method for constructing new images.

2.3.3 Generating New Images

To generate new images, we first upsample our low-resolution MR image y_0 to the target resolution using the cubic spline interpolation method described in the prior section to get a new image $\hat{y_0}$. We can think of our attained image y_0 as $y_0 = h_0 * x + z$, where h_0 is a smoothing kernel, x is the true high resolution image, and z is noise. In the 2-D MRI acquisition case, h_0 is a truncated sinc in the image space with a cutoff frequency.

Using $\hat{y_0}$, a given rotation matrix R_i, and the smoothing kernel h_0, we now generate 2 new images y_{r0}, y^i_{a0}, and y^i_{s0} (in section 3, we will explain in further detail how we chose the rotation matrices).

- $y^i_{a0} = R_i(h_0) * y_0$, which is interpreted as the upsampled y_0 that is convolved by the rotated kernel $R_i(h_0)$.
- $y^i_{s0} = h_0 * R_i(y_0)$, which is interpreted as y_0 rotated by R_i that is then convolved with the smoothing kernel h_0.

Using these generated images, we then pair up y_{r0} and y^i_{a0} as a "training pair" to learn a transformation that will be applied to y^i_{s0}. Essentially, Jog et al. extracts voxel patches from transformed versions of y^i_{a0} to be used as features in order to learn a deconvolution transformation to attain y_{r0}. We then apply this learned transformation to y^i_{s0} to get our new image.

In order to attain this transformation, however, we must first use a method known as Anchored Neighborhood Regression (ANR). ANR is a fast method of super-resolution that learns a transformation for individual cluster groups of features. Our implementation of ANR is similar to the method used by Jog et al., however, we change some aspects of the algorithm due to a lack of familiarity with sub-algorithms used in ANR.

The first step of ANR is to create training features from y^i_{a0} by taking the first and second gradient images (in each direction) using the Sobel and Laplacian filters (in total, we get 6 gradient images per y^i_{a0}). Then we extract non-overlapping voxel patches j of size 4x4x38 from each gradient image; we chose this patch size based on the relative resolution of each direction – the x- and y-directions have roughly 10-times higher resolution than the z-plane. Next, we concatenate these patches into a flattened feature vector that we apply PCA to in order to reduce the feature space by a magnitude of 10^2; the original feature vector has size $6 \times 4 \times 4 \times 38 = 3648$, which is then reduced to size 36. We denote this compressed feature vector at voxel patch j as $f_j(y^i_{a0})$.

After extracting these features, we then calculate a difference image $y^i_d = y_0 - y^i_{a0}$. We pair each $f_j(y^i_{a0})$ with the corresponding voxel patch of y^i_d at position j, which is denoted as $g_j(y^i_d)$ to create our training pair; $f_j(y^i_{a0})$ are our features and $g_j(y^i_d)$ is the target patch. We then use the K-means algorithm to cluster our feature vectors into 128 groups (this is a value taken directly from Jog et al. since we already had many hyperparameters to optimize). In the original Self Super-Resolution paper, they use the K-SVD algorithm to learn the mapping dictionary for the features, however, we decided to use the K-Means algorithm due to its simplicity in implementation and interpretability.

Once each feature vector is assigned to a cluster K, we learn a transformation P_K that minimizes the least squares distance for $F_K P_K = G_K$, where F_K represents an array of all the features $f_j(y^i_{a0})$ that are in cluster K and G_K are the voxel patches that correspond to the features in F_K. To attain the transformation matrix P_K, we use the ridge regression estimate:

$$P_K = (F_K^T F_K + \lambda I)^{-1} F_K^T G_K. \tag{6}$$

We explain in more detail how we choose the regularization term λ in the following section.

Once we have the learned transformation P_K for each cluster K, we then compute the feature vector $f_j(y_{s0}^i)$ for y_{s0}^i using the same feature extraction method described above. We then assign each of these features to one of the clusters generated from the training set and apply the corresponding, learned P_K to each feature vector. We then combine each of these $g_j(y_{s0}^i)$ estimates at each voxel patch j to form our target difference image $\hat{y}_{ds}^i$. Since our original transformation learned a mapping from patches of y_{a0}^i to patches of y_d^i, our linear transformation of $g_j(y_{s0}^i)$ extends to a difference image $\hat{y}_{ds}^i$; we now extract a target image $\hat{y}_i$ by adding y_{s0}^i to $\hat{y}_{ds}^i$, so $\hat{y}_i = y_{s0}^i + \hat{y}_{ds}^i$.

2.3.4 Applying FBA on Generated Images

We generate N images using the above method, where we select N rotation matrices R_i. We then apply FBA on the set of images $\{\hat{y_0},\ R_1^{-1}(\hat{y_1}),\ R_2^{-1}(\hat{y_2}), ...,\ R_N^{-1}(\hat{y_N})\}$ to get our super-resolution image $\hat{x}$. As described in Jog et al., ANR is able to add information in a single direction of the Fourier space but if we are able to run the algorithm in multiple directions then combine all N images using FBA, we are theoretically able to add information across the entire Fourier space to synthesize a higher resolution image.

2.3.5 Summarizing the Self Super-Resolution Algorithm

1. Upsample low resolution image y_0 to the target resolution using spline interpolation (or some other non-example based method).

2. Create a smoothing kernel h_0 that is the slice select filter for the 2-D MRI acquisition.

3. For a given rotation matrix R_i:

   - Generate $y_{a0}^i = R_i(h_0) * y_0$
   - Generate $y_{s0}^i = h_0 * R_i(y_0)$
   - Create the first and second gradient images of y_{a0}^i
   - Extract voxel patches from the gradient images, concatenate into a feature vector, and apply PCA to get $f_j(y_{a0}^i)$
   - Compute the difference image $y_d^i = y_0 - y_{a0}^i$ and pair voxel patches $g_j(y_d^i)$ with corresponding $f_j(y_{a0}^i)$
   - Use the K-Means algorithm to cluster each $f_j(y_{a0}^i)$
   - For each cluster K, compute a transformation P_K that minimizes the least squares distance $F_K P_K = G_K$
   - Create feature vector $f_j(y_{s0}^i)$ for image y_{s0}^i (using the first and second gradient images)
   - Cluster $f_j(y_{s0}^i)$ into one of the K clusters from the training features
   - Apply P_K to each $f_j(y_{s0}^i)$ and combine these projections to form $\hat{y}_{ds}^i$
   - Add y_{s0}^i to $\hat{y}_{ds}^i$ to get target image $\hat{y}_i$
   - Repeat N times for different rotation matrices R_i

4. Apply FBA on the set $\{\hat{y_0},\ R_1^{-1}(\hat{y_1}),\ R_2^{-1}(\hat{y_2}), ...,\ R_N^{-1}(\hat{y_N})\}$ to get $\hat{x}$

3 Experiments

3.1 General Experiment Details

For comparison matters and because the images are 3-D (which means a lot of possible slices), we focused on one specific knee MRI. The resolution it has on the 3 axis corresponds to (444, 448, 38).

Because we didn't have closely related images of different level of resolution, we chose this image to be our HR reference and we built LR images from it using a block reduce method (downsampling using a function on local blocks, func = max was the one working best). For computational cost, we decided keep on applying our methods on the $z-$axis, so the resulting LR images after downsampling had $448 \times 448 \times 19$ resolution.

For both methods, we used numpy/skimage/dipy functions for the preprocessing steps, i.e denoising, downsampling and interpolating.

3.2 Example-Based Super-Resolution Implementation Details

For the example-based method implementation, we followed the process explained in Section 2. To get an interpolated version of the LR image, we used a cubic-spline interpolation. Then, because of the length of one experiment (2 days), we took the same hyper-parameters values as in the paper: $h = [32, 16, 8, 4, 2]$ (5 iterations), $k = 256$, a search volume of $7 \times 7 \times 7$ to compute similarities in the HR reference, and a $3 \times 3 \times 3$ neighborhood to compute local similarities in the reconstructed image.

3.3 Self Super-Resolution Implementation Details

For the self super-resolution algorithm there were a number of hyperparameters that had to be determined; these hyperparameters include the smoothing kernel h_0, the rotation matrices R_i, the regularization parameter λ in the ridge regression equation, the number of principal components, the number of clusters K in the K-means algorithm, and the weight factor p in our FBA implementation.

Due to computational and time constraints, we simply copied some hyperparameters from Jog et al.; in their paper, they reduced the number of features by a magnitude of approximately 10^2 so for our case we used 36 principal components (there were roughly 3600 features originally), and for the K-means implementation, we used 128 cluster groups since those were the number of dictionary mappings that Jog et al. used in their K-SVD implementation. We recognize that these parameters can be tuned further but we devoted our time to optimize the other parameters in the algorithm.

For the other hyperparameters R_i, h_0, λ, and p, we determined them either using a validation image that we call $\hat{V}$ and then testing various values for the parameter or we used prior knowledge related with the MRI acquisition process.

To determine the smoothing kernel h_0, we knew prior that the function (according to Jog et al.) was the slice selection pulse of the 2-D MRI acquisition; in the imaging space, this is a truncated sinc function. However, we had to determine the cutoff frequency for this truncated sinc, which we could not rigorously determine since we did not have further information on how the MRIs were acquired. As such, we simply used a truncated sinc, evaluated between -4 and 4 at 21 discrete points, that was cutoff to 0 at values less than -2 and values greater than 2. We had to empirically determine these cutoff values, which seemed to yield decent results for our experiments. To get more accurate cutoff values in the future, we would need to know more information about the MR acquisition process for these knees.

Similarly, for the rotation matrices we had to empirically determine the rotation angles. Since FBA was first used to remove the effects of camera shake and hand blurring from images of the same scene, we decided to use small rotations in our MR images around the Z-axis. We limited our rotations to be up to $5°$ in either the clockwise or counterclockwise directions and used 4 separate rotated matrices in our final SSR implementation. We also tested the use of much larger rotations of $\{90°,\ 180°,\ \text{and}\ 270°\}$; we highlight these results in the next section.

To get the values of our hyperparameters λ and p, we used $\hat{V}$ to perform hyperparameter optimization. We varied the values of the regularization term λ between $\{0,\ .5,\ 1,\ 1.5,\ 2,\ 2.5,\ 5,\ 10,\ 100\}$. For p, we tested the values $\{0,\ 1,\ 2,\ 3,\ 4,\ 5\}$. We determined the best values to use by visually observing the generated images and chose the parameters that yielded the sharpest image. Images of the generated images for each parameter value can be found in the corresponding section in the notebook *SSR_method.ipynb*. Due to computational and time bottlenecks, we chose the values for both parameters independently and did not test every combination between the two hyperparameters.

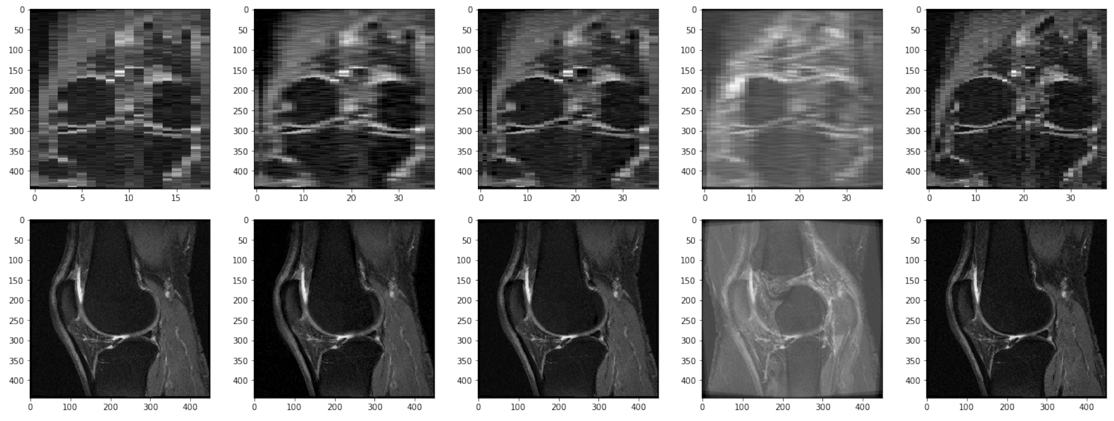
In the end, we decided to use $\lambda = 2.5$ and $p = 0$; with $p = 0$, we are essentially generating an average image (in the Fourier domain) of the input burst images, which makes sense since each of our generated images are only adding edge features in a specific direction (due to our specific construction of $\hat{y}_i = y^i_{s0} + \hat{y}^i_{ds}$).

4 Results

4.1 Evaluation & Comparisons

Since the images are in 3-D, we are going to show relevant slices to highlight improvements. We specifically chose 1 slice in the (x, z) plane and one in the (x, y) plane to understand the two types of changes that could happen in our 3-D image; the first image shows how the resolution improved along the z-axis, while the second image checks what happened on thw (x, y) planes.

First, the image generated from the example-based method shows clear improvements towards the HR reference. However, we feel some weaknesses in the algorithm for our examples. For instance, local neighborhoods were of the same size along all directions, whereas resolution is far



(a) Original LR (b) LR interpolated (c) Example-Based (d) SSR method (e) HR reference

Figure 1: Above are results for our experiments. Spline interpolation (b) already does a nice job, Example-based (c) improves images even more. However, SSR (d) doesn't perform as well where image artifacts and blurring leak into the generated image.

Super-Resolution Method	**Signal-to-Noise Ratio**
High-Resolution Reference	3.53
Low Resolution Image	1.02
Example-based Super-Resolution	3.73
Self Super-Resolution	2.59

Figure 2: Above is a table that highlights the Signal-to-Noise Ratio of the techniques we tested.

lower along the z-axis so this might have created some deformations. Concerning the preprocessing steps, we might not have denoised, downsampled and interpolated exactly as in the paper, as some parameters were unspecified and hard to tune knowing the running time of the method; although the method generated a very good HR approximation of the image, the method took over 3 days to run on a CPU.

Unlike the example-based super-resolution method, the SSR algorithm did not generate nearly as good results. Over the Z-plane, the generated high resolution image retains the main features with some edges becoming more refined but other aspects of the image exhibit significant blurring. In addition, the slice in the (x,y) direction acquires some artifacts that were never present in the original image; these artifacts may arise due to the rotated images combining information in the Fourier domain. Lastly, the generated image does not retain the original color gradient as the other images that were tested – the SSR image is much lighter in gradient than the other images. Even though the SSR algorithm does not yield great results compared to the example-based method, the method runs at a much faster speed; for 4 rotation matrices R_i, the method creates a high resolution approximation in roughly 10 minutes on a CPU. Lastly, the method only relies on a single LR image and does not require a HR reference image like the example-based method, however, the efficiency in run-time and data comes at the cost of performance for the SSR algorithm.

We also looked at the Signal-to-Noise Ratio (SNR) as a metric of evaluation. For this, we selected two regions of the slice in the (x,z) plane that is presented in Figure 1; one region contains no information about the image's signal which is used to estimate the standard deviation of the noise, and the other region that was chosen contains large information about the image's signal which was then used to compute the standard deviation of the signal. The signal-to-noise ratio is defined as:

$$SNR = \frac{StdDev(Signal\ Region)}{StdDev(Noise\ Region)}. \tag{7}$$

Table 2 summarizes the SNR of our tested methods. Unsurprisingly, the example-based method yields higher SNR compared to the SSR method; both methods, however, have higher SNR than the low resolution image, which makes sense and indicates that the techniques do generate some relevant HR information. What is interesting is that the example-based method also outperforms the HR reference in terms of SNR. This is a very peculiar observation and we must test the super-resolution method on multiple images before making any broader conclusions about the method. For now, we can conclude that the example-based method does better than the SSR algorithm that utilizes FBA.

4.2 Further experimentations based on SSR implementation

We ran some further experiments with the SSR algorithm along with the main algorithm from Jog et al. Apart from tuning the algorithm as much as we could, we tried a bunch of simpler ideas to try to get better results than the ones found with our SSR implementation.

First, we tried to replicate in a simpler way the setting in which FBA is used. We tried creating 4 images rotated with different angles from $-5°$ to $5°$ in order to feed them into FBA. Angles too big first induce an important loss of information that, when accumulated with the loss

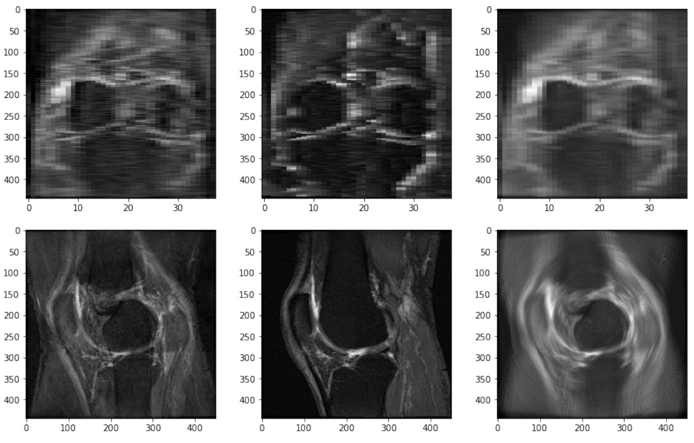

(a) (b) (c)

Figure 3: Further experimentations after the SSR failure. (a) corresponds the 8 rotations of the LR image fed to FBA, (b) corresponds to the sharpened image based on the Sobel gradients (along z-axis).

of information we have performing, yields even poorer results. Results are very noisy where the images are hard to denoise but some high resolution patterns are present.

Then, we saw that the gradient images were very good edge detectors, so we tried using them in order to sharpen the image. This would not improve resolution but should yield nicer images. It is actually the case, as you can observe in Figure 3. We added to the original image the Sobel gradient image (performed along the z-axis), weighted by some factor we tuned (10^{-1}) to get the best trade-off.

Finally, we tried using only the images created with the first pre-processing step of the algorithm. We took angles between $-5°$ and $5°$ and built 8 images to feed in FBA. Results were poorer than before.

5 Future Improvements

Our experiments were first constrained to very low resolution on the z-axis because of the dataset we had. Indeed, no images with higher resolution than 40 on the z-axis were available, so one immediate improvement would be to test these methods on more data with higher resolution across the through-plane; with these images, it would be easier to assess performance between different algorithms.

In addition, there are a number of immediate improvements we can apply to the SSR algorithm. First, we could acquire more information about the MRI acquisition process to get a better smoothing kernel h_0; as described earlier, we simply approximated a cutoff frequency for the sinc kernel. We could also try different rotations for the original image; right now, we only rotate across the (x,y) plane (e.g. tilting about the z-axis) but we could apply rotations that rotate along the other axes and the cross-diagonals of the axes. This would theoretically yield Fourier information in different areas of the Fourier space. We could also tune more hyperparameters in our method, such as the number of principal components to use and the number of clusters in our K-means algorithm. All of these adjustments could yield improved results in our method.

Finally, some other techniques could be tried, as the ones discussed in [5]. Right now, super-resolution is being tackled using deep learning based methods and has yielded state-of-the-art results [6]. Testing these learning-based super resolution methods on MR images could yield even better results than the methods we tested.

6 Conclusion

We compared three different approaches to super resolution: spline interpolation, self similarity and self super-resolution. The method that performed best is the self similarity algorithm, but this method is by far the most time-consuming and requires a HR reference image that might not always be available in a real setting, since one may only acquire a LR MR image. However, apart from the unknown reason destroying our results for SSR, this method seems pretty effective and has strong theoretical backing since the method relies on combining different information in the Fourier space; based on the results of Jog et al., SSR seems promising due to its fast run time, however, there are a number of hyperparameters (R_i, h_0, λ, p, the number of PCs, and the number of clustering groups) that must be intelligently chosen in order for the algorithm to attain great results. We seemed to not have attained the optimal hyperparameter combination to yield the best results using SSR.

References

[1] Hsieh Hou and H Andrews. Cubic splines for image interpolation and digital filtering. *IEEE Transactions on acoustics, speech, and signal processing*, 26(6):508–517, 1978.

[2] José V Manjón, Pierrick Coupé, Antonio Buades, D Louis Collins, and Montserrat Robles. Mri superresolution using self-similarity and image priors. *Journal of Biomedical Imaging*, 2010:17, 2010.

[3] William T Freeman, Thouis R Jones, and Egon C Pasztor. Example-based super-resolution. *IEEE Computer graphics and Applications*, 22(2):56–65, 2002.

[4] Amod Jog, Aaron Carass, and Jerry L Prince. Self super-resolution for magnetic resonance images. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 553–560. Springer, 2016.

[5] Eric Van Reeth, Ivan WK Tham, Cher Heng Tan, and Chueh Loo Poh. Super-resolution in magnetic resonance imaging: A review. *Concepts in Magnetic Resonance Part A*, 40(6):306–325, 2012.

[6] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *IEEE transactions on pattern analysis and machine intelligence*, 38(2):295–307, 2016.