A Comparison of Super-Resolution Techniques Applied to MRIs

Jiarun Li, Jaqueline Lu, Daniel Levi-Minzi

I. Introduction

Magnetic resonance imaging (MRI) is a widely used tool for diagnosis in the medical field. However, it can take a long time to generate a clear high resolution MRI. This is because the process requires a period of waiting for the protons to resume precessing. It is not possible for a hospital to take a full day to scan a patient. In severe cases, scans must be done as quickly as possible. An example of such a severe case would be getting scans for potentially lethal head traumas. There are many ailments that could be treated more reliably with imaging data available, but that cannot spare the time required to produce such imaging data. Another common problem is generating quality MRIs of children. Children are not naturally inclined or easily convinced to remain motionless for long stretches of time. Thus, MRIs of children are difficult for practitioners. The common issue in both of the aforementioned cases is the lack of the time required to generate high resolution, reliable MRIs. Time is not malleable, but pixels are.

Super resolution (SR) is the the process of increasing the resolution of a given image via transformations. The goal is to create a reliable mapping of low resolution images to their higher resolution counterparts. Certain SR methods, like the ones used to enhance pictures taken with mobile phones[3], use an array of images for input during scaling. In this paper, only single image super resolution (SISR) methods will be discussed. This means that the mapping will only consider a single image for its input.

There are many super resolution techniques. Some common ones are nearest neighbor, bilinear, and bicubic interpolation. The strategy for each of these methods is to create new pixels with reasonable intensity or color by using the values of surrounding pixels. The quickest of these methods is nearest neighbors. This algorithm only considers the pixel nearest to the interpolation point. Bicubic and bilinear interpolation consider the surrounding 2x2 or 4x4 (respectively) grid of pixels[7]. They use the weighted average of these pixels to generate the value of the newly added pixel. These algorithms are functional, but limited by artifacts[6]. Interpolation techniques do not differentiate between features in an image; they apply an identical transformation to every part of the image. Such a uniform approach is unable to account for intricacies in an image and thus is limited in effectiveness.

In recent years, deep learning techniques have been applied to super resolution. These approaches to super resolution have outperformed the previously mentioned techniques. In this domain, performance is measured in peak signal-to-noise ratio (PSNR). This metric measures the difference in true pixel value and computed pixel value, weighted by the maximum value a pixel could take. The most common deep learning models applied to super resolution are based on the ResNet[10] architecture. This architecture allows for very deep networks to be trained. This is done by passing the activation of shallower layers to deeper layers in the network via skip connections. In this paper, we will explore enhanced deep residual networks (EDSR)[9] and wide activation deep residual networks (WDSR)[8]. We will also compare them to rapid and accurate imaging super-resolution (RAISR)[2] method. All of these networks will be trained and validated on a dataset of MRI images.

II. Related Work

Pixel Shuffling

During interpolation, filler pixels with value zero are added to the image and then replaced with values inferred from the surrounding pixel values. This process is not compatible with deep learning models because the addition of zero is not a differentiable operation. Thus, this process cannot be applied within a deep learning model. Several deep learning architectures still use interpolation on the image before any learning is done. The RAISR algorithm applies interpolation at the beginning[2]. As did early super resolution networks. As the authors of the WDSR paper note, this choice is computationally inefficient because it increases the size of the input to the model by the square of the scaling factor[8]. Instead, both EDSR and WDSR rely on pixel shuffling (or sub-pixel convolution). For some scaling factor s, this takes the HxWxC image and performs a convolution to get a HxWxs^2C result. This result can then be reshaped to a sHxsWxC image[16]. Thus, the image has increased by a factor s from a learnable operation.

Loss Functions

Both EDSR and WDSR opted to use the mean absolute error functions (L1). This decision was made as a result of the findings of H. Zhao et al [11], which claimed that the L1 loss performed better for image processing than the L2 counterpart. We tested the validity of this claim in the implementations of both EDSR and WDSR.

Batch Normalization

Batch normalization forces each layer to have mean zero and a variance of one. This is meant to improve the stability of training in deep learning networks. EDSR's authors found that this process contributed to worse performance. They attribute this finding to less flexible feature ranges imposed by normalization[9]. WDSR's authors were in agreement with this conclusion. However, they find that weight normalization is a strong alternative to batch normalization and no normalization[8]. We explore different normalization techniques for both EDSR and WDSR.

III. Methods

Dataset

It was necessary to train the models on a dataset that was consistent with the potential use case we chose to explore. Thus, the IXI brain development dataset [11] was used. This dataset consists of sets of 600 MR scans from three different hospitals. The data comes in the NIFTI format. For each of the images in the dataset, three slices where taken from the middle of the head. The resulting slices were then formatted as 256x256 images. From those images, roughly $1,000$ images were taken from the three hospitals, evenly distributed. These images were then resized to be 64x64, using bicubic interpolation. Finally, 900 of these images are used in our training set and 100 are used in our testing set.

RAISR

The *RAISR* method generates a hash table of patch filters by solving a least squares minimization between a cheaply upscaled low resolution (LR) patch and the true high resolution (HR) patch. The "cheap" upscaling method is up to the implementer, but the authors chose bilinear interpolation. This method allows for a filter to be constructed without considering every possible patch in the LR image. This feature combined with some algebraic tricks improves training time. A hash table is used to efficiently cluster the filter patches. The keys are generated by considering the direction, strength, and coherence of a patch via its gradient. To prevent over sharpening a locally weighted blending method is used on the image at the end of the upscaling process[2]. To summarize, traditional methods apply cheap upscaling, which smooths the sharp pixel contrast on edges, and creates more natural lines. RAISR applies this traditional method and then learns additional filters to be applied to patches of the image. It does this efficiently by utilizing clustering of similar patches and algebraic tricks to increase parallelization. In our implementation, we referenced previous work from Jalali Laboratory at UCLA [15].

Hashing in RAISR

Hashing is the core of RAISR. A well established hash table in RAISR contains functions of local gradients as keys, and corresponding pre-learned filters as entries. We use eigenanalysis to compute the attributes of the gradient keys. For every pixel value k, the neighboring $\sqrt{n}$ pixels in each of the four directions are considered for calculation. We then combine the horizontal and vertical $\sqrt{n}$ neighbors, and form a nx2 matrix G_k, where n denotes the span of surrounding pixels and 2 denotes number of axis. Next, we decompose the eigenvector by performs matrix multiplication on $G_k^T G_k$ to create an 2x2 output matrix. In addition, we uses a Gaussian 2D filter [13] to generate a weight matrix and apply the weight matrix to form a new matrix $G_k^T W_k G_k$. From this matrix, we can then obtain the largest eigenvector to compute the three attributes of gradient key: gradient angle (θ_k), gradient strength (λ_k), as well as coherence (μ_k) from equations described in [2] and [14]. In out implementation, we used 24 for quantization factor of angle, 3 for quantization factor of strength, and 3 for quantization factor of coherence.

Residual Blocks

Both EDSR and WDSR are residual networks. This means that both networks use the residual block structure. Residual networks are an evolution of convolutional networks. In both cases, each layer feeds into the subsequent layer, but residual networks also feed directly into deeper layers of the network. In a typical network, the goal is to learn the the true output Y. Given some input X it will produce an approximate of the truth Y'. In a residual network, the goal is to learn the residual R. Given an input X the network will produce an approximate of the truth in the form $X + R$.

Image Normalization

For the EDSR and WDSR models, normalization is applied to images before they are put into the network. This is to ensure that the range of output values is consistent for any given input in the dataset. Without this step, a difference in scanning intensity has the potential to inhibit learning. This is done for every input x as follow:

$$x = (x - \hat{I})/127.5$$

where Î denotes average intensity. The reverse operation (denormalization) is applied at the end of the network.

EDSR

The EDSR approach uses a convolutional layer followed by a variable number of residual blocks. The structure of the residual blocks used in EDSR is identical to that of ResNet with the exception that batch normalization is removed. The residual block ends with the input weights being added to the output of a final convulational layer. In order to upscale the image, an additional convolution is performed and then pixel shuffling is used as a learnable interpolation alternative.

In our initial training attempts, weights pre-trained on div2k were used[16]. For these trials, we used 16 residual blocks and 64 feature maps per convolutional layer. After this, the model was trained without any starting weights. We followed the specifications used in by the EDSR paper's authors and used 32 residual blocks with 256 feature maps per convolutional layer[9]. Of course, we expected better performance because of the increased size. Below are the details of the different EDSR specifications that were trained (provided by keras model summary function).

batch norm	residual blocks	filters	parameters
no	16	64	1.5M
no	32	256	43.1M
yes	32	256	43.2M

The author's of the EDSR paper do not use batch normalization because they found it performed worse. We wished to confirm that batch normalization had a detrimental effect. We ran training sessions with and without batch normalizaiton. With batch normalization we had to use a batch size of 10 because of memory limitations. Without it we could use a batch size of 16. However, we opted to use a batch size of 10 because it would make plotting both together easier. Both training sessions were 500 epochs. One final session was run to generate the results to compare to WDSR. This model had a batch size of 16, 32 residual blocks, and 256 filters per convolutional layer. This model was trained for 1,000 epochs.

Wide Activation

The idea that the authors of the WDSR paper had was to expand the features more before the relu activation in the residual blocks. They suspected that information from the shallow layers was not flowing to deeper layers. It is not clear why relu is restricting the flow of information. However, the solution is to simply supply a higher volume of information so that more gets through. To do this, the author's increased the number of filters before activation. This can be implemented by multiplying the number of filters by some expansion factor. For our trials we kept this expansion factor constant at six. However, this expansions factor is reported to decrease the accuracy of the model unless the large convolutional kernel is broken into two lower rank convolutional kernels[8]. Due to time constraints, we could not test differences in performance with variable expansion factors.

Weight Normalization

One of the original problems with neural networks was the vanishing gradient problem. Weight normalization is one of several methods used to address this issue. This process sets the weights according to the following:

$$w = \frac{g}{||v||} \cdot v$$

This has the effect of separating the magnitude of the weight vectors from their direction. Then instead of minimizing loss with respect to the weight w, we minimize with respect to g and v are optimized using gradient descent. This separation causes the norm of the activation to remain somewhat constant; an effect similar to that of batch normalization.

WDSR

The WDSR model uses weight normalization. The structure is similar to the edsr model. There is a convolutional layer followed by a series of residual blocks. Then there are interwoven convolutions and pixel shuffling. Expansion factors are applied to the convolutions in its residual blocks. As previously mentioned these expansion factors necessitate the split of large convolutions into two low rank convolutions. The difference in these two residual block structures can be observed in figure 1.

In initial training attempts, weights from a model trained on div2k were used[16]. For these trials, we used 32 residual blocks and 32 feature maps per convolutional layer. Both papers claimed that mean absolute error performed better than mean squared error. To confirm this finding, we trained our 32B, 32FM model with each loss. The model was also trained from scratch using 16 residual blocks with 128 feature maps per convolutional layer[9]. Below are the details of the different WDSR specifications that were trained (provided by keras model summary function).

residual blocks	filters	parameters
32	32	615K
16	128	4.8M

Consistent with the EDSR paper, the author's of WDSR found batch normalization had negative effects on training. However, unlike EDSR, the model does use weight normalization. We ran training sessions with weight normalization, batch normalization, and no normalization to confirm the validity of their decisions. Each of these sessions was run with 16 residual blocks and 128 feature maps. They were trained for 500 epochs each. A final training session of 1,000 epochs was run to compare against the EDSR 1,000 epoch trained model.

IV. Results

MSE and MAE

Mean absolute error (L1 norm) is one of two error types we will use. With this method, error is calculated according to the following equation:

$$MAE = \frac{\sum_{M,N} |I_1(m,n) - I_2(m,n)|}{M * N}$$

Mean squared error (L2 norm) is the second error type used. With this method, error is calculated according to the following equation:

$$MSE = \frac{\sum_{M,N} [I_1(m,n) - I_2(m,n)]^2}{M * N}$$

PSNR Computation

We derive our PSNR measurement from the MSE value. PSNR is the ratio between a signal's maximum power and its noise. Higher PSNR values represents larger signal power and thus higher reconstruction quality. PSNR is computed using:

$$PSNR = 10 * log_{10}(\frac{M^2}{MSE})$$

where M represents the maximum potential fluctuation from the type the input image. For example, for types such as double and float, the max fluctuation would be 1, whereas more commonly for 8 bit unsigned int type, the M value would be $2^8 - 1 = 255$.

Batch Normalization vs. No Normalization (EDSR)

The first training sessions were used to confirm the claim that batch normalization was not suitable for super resolution networks. The model trained without batch normalization achieved greater accuracy than its counterpart (figure 2). The model trained with batch normalization did have an unexpectedly poor start as well. It is possible that the keras implementation uses parameter initialization that effected its performance for this task. As a result of this experimentation, we opted not to use batch normalization for our final implementation of EDSR.

L1 vs. L2 Norm (WDSR)

When training with L1 and L2 norms we did not find a clear better option. The L2 norm achieves slightly higher accuracy (figure 3). However, this question should be explored more thoroughly. More trials are needed to determine whether the L2 norm is consistently more accurate. Because our results here were not decisive, we opted to default to the L1 norm for our final EDSR model because it has a smaller computational footprint and was the method used by the paper's authors.

Weight Normalization vs. Batch Normalization vs. No Normalization (WDSR)

Our results when testing different normalization methods were consistent with the findings detailed in the WDSR paper. As visible in figure 4, the model trained with weight normalization is more accurate than the others trained with batch or no normalization. Also, similar to our finding for EDSR, batch normalization performs worse than no normalization. However, batch normalization begins at a much lower accuracy level. Again, we suspect this is a result of some accidental parameter initialization in keras. Nonetheless, the results were encouraging, so for the final training model of WDSR, we opted to use weight normalization instead of the alternatives.

EDSR vs. WDSR

EDSR won the NITRE 2017 award for super resolution and WDSR took its place in 2018. It was reasonable to expect that WDSR will outperform EDSR in our own independent trials. In these training sessions, we ran for 1,000 epochs (57,000 steps) with a batch size of 16. WDSR did achieve slightly better accuracy. With our validation test we found that EDSR's psnr to be 32.93 and WDSR's psnr to be 33.17. Also, as visible in figure 5, the WDSR model managed to achieve higher accuracy with far fewer training epochs than EDSR.

RAISR

We tried to change various parameters to obtain better results. During our attempts, there are some interesting results worth mentioning. As figure 6 indicates, with the current implementation, lower upscaling factor creates higher PSNR value. Higher PSNR value indicates that the model is performing better. Thus, it agrees with the result in the sense that the higher upscaling factor applied, the worse accuracy it will have. An upscaling factor of 2 performs the best with PSNR value of 28.5, as indicated in Figure 6. For consistency with EDSR and WDSR, we kept the upscaling factor as 4 in our further experiments. Compared with the traditional methods, RAISR computed PSNR value is better than the bicubic counterpart.

Additionally, changing gradient size or modifying quantization factor for strength does not produce a significant effect in the output, evidenced through small change in the amplitude of PSNR values.

Summary of Different Super Resolution Techniques

method	train time	PSNR
Bicubic	NA	21.74
RAISR	30 minutes	22.92
EDSR	48 hours	32.93
WDSR	14 hours	33.17

V. Discussion

When testing the L1 and L2 norms, we did not find a significant difference in performance between the two. We expected that the L1 norm would outperform the L2 norm because that is the finding reported by B. Lim et al[9]. However, we had limited time to run trials and so our results here should not be considered conclusive. More models should be trained using the different norms to confirm this finding for both WDSR and EDSR.

One of the key differences between the WDSR model and the EDSR model is the use of weight normalization in the former. The authors of both papers advise against batch normalization and the authors of the WDSR paper present weight normalization as an alternative . When performing batch normalization, the mean and standard deviation of the pre-activations in a batch is found at every step. These values are then normalized. This solves the problem of exploding and vanishing gradients. However, it has the undesirable effect of causing each image's output be a function of all the other images in the batch. This is okay in some settings because it might allow the model to generalize more easily. However, for super resolution this can cause features to be lost, hurting accuracy. As stated in our results, for EDSR we found that batch normalization did have a negative effect on accuracy. For WDSR, we found that weight normalization performed best. Moving forward, weight normalization seems to be a good alternative to batch normalization for image restoration. It would be interesting to try applying weight normalization to the EDSR model to see if performance is improved.

We found that the WDSR model achieves the greatest accuracy. Followed by EDSR, and then RAISR. This was expected because WDSR had surpassed EDSR in the paper from Yu et al[8]. However, RAISR performed worse than expected. In previous tests, RAISR had typically performed 4x upscaling with PSNR in the range of 26 to 29 [2]. This is likely due to dataset selections, as using different datasets to train a model can result in different PSNR test values [15]. The datasets used previously consisted of more typical images. In such images, there is likely a greater variety of common structures. MRIs tend to have curvature and very subtle details. Additionally, the ground truth high resolution images were only 256x256 pixels. Thus, the number of learnable patches was far lower than typical training sets for RAISR and mistakes in a few patches had the potential to significantly derail results. In the future, we should rerun the RAISR training on higher resolution MRI data to see how an increase in information might improve results.

VI. Conclusion

Through our trials we tested two different deep learning super resolution techniques. Both of these techniques outperformed RAISR. However, both EDSR and WDSR achieved similar levels of accuracy. The main difference between the two methods was in time to train. From figure 5, we observe that the training for EDSR is more gradual than that of WDSR.

It is hard to recommend that EDSR be used in place of WDSR. The widened pre-activation layer combined with weight normalization allowed WDSR to achieve state of the art results with far fewer parameters than EDSR. However, both of these models are not realistic for computationally limited hospitals or users. In order to upscale an MRI using WDSR, a doctor might have to wait for a central computer to compute the upscaled image and foward it to their device. However, if RAISR was the technique of choice than the computation could be done locally. RAISR is shown to have worked far worse on our dataset, which is at a normal resolution for MRIs and was sampled from three different hospitals. This suggests that while RAISR might be the more computationally feasible option, it seems to be more susceptible to oddities in the dataset than EDSR or WDSR.

All three of the options for super resolution output results that outperform classic interpolation techniques. Yet, the most computationally feasible option is only slightly more accurate. If the resources are available, then WDSR is a good model for this task. In the future it would be interesting to test other super resolution techniques that are specifically designed to work under computational constraints. For instance, there are models for mobile phone super resolution tasks. These types of models could make MRI super resolution much more accessible.

VII. Appendix

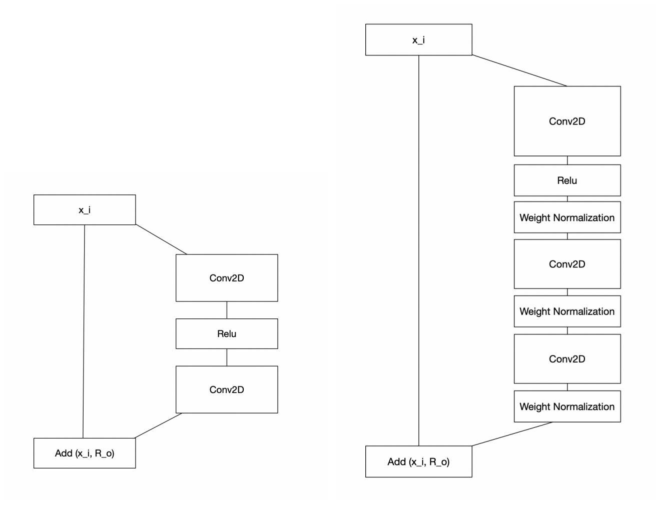

Fig. 1. A comparison of the residual blocks used by EDSR and WDSR. On the left is the residual block of the EDSR model. One the right is the residual block of the WDSR model.

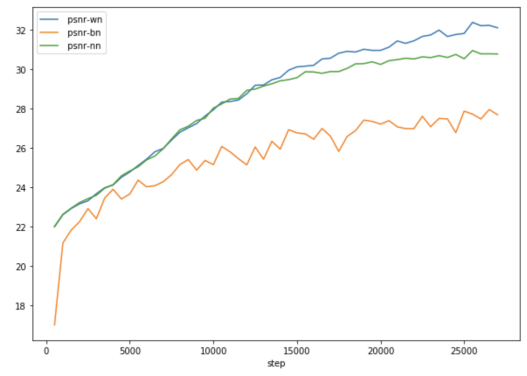

Fig. 4. PSNR during training with batch, weight, and no normalization on the WDSR model. These values were calculated on a sample of 10 images from the validation set at half epoch intervals. psnr-wn is weight, psnr-bn is batch, and psnr-nn is none.

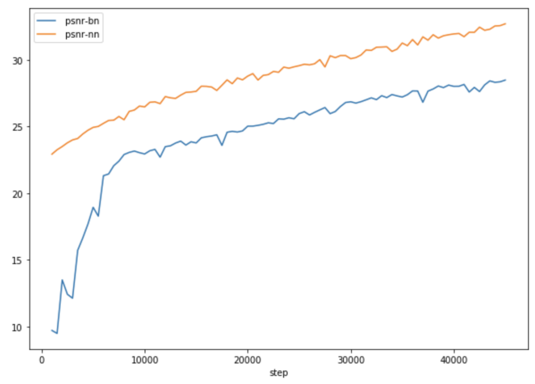

Fig. 2. PSNR during training with/without batch normalization for the EDSR model. These values were calculated on a sample of 10 images from the validation set at half epoch intervals. psnr-bn has batch normalization and psnr-nn does not.

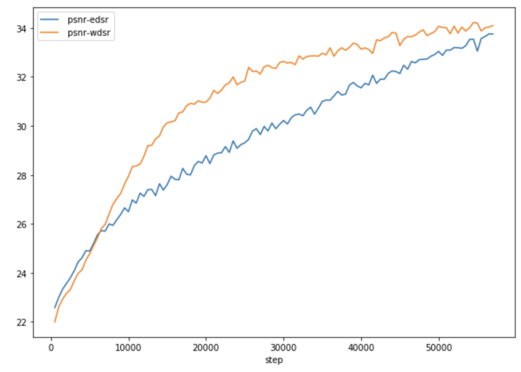

Fig. 5. PSNR during training for EDSR and WDSR. These values were calculated on a sample of 10 images from the validation set at half epoch intervals.

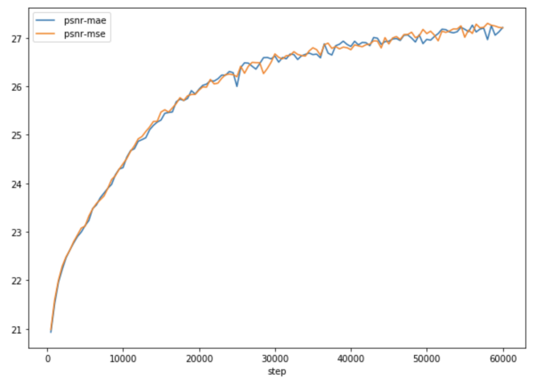

Fig. 3. PSNR during training with L1 norm vs L2 norm for the WDSR model. These values were calculated on a sample of 10 images from the validation set at half epoch intervals. psnr-mae is L1 norm and psnr-mse is L2.

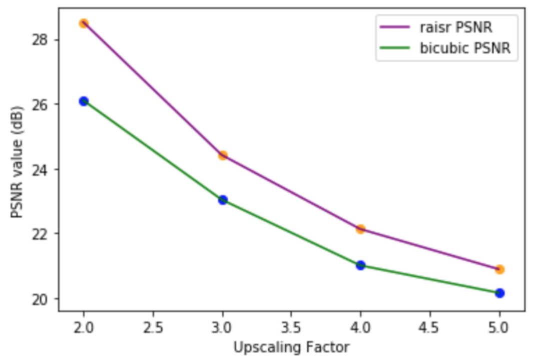

Fig. 6. Relationship between upscaling factor and PSNR values, and a comparison between RAISR PSNR and bicubic PSNR.

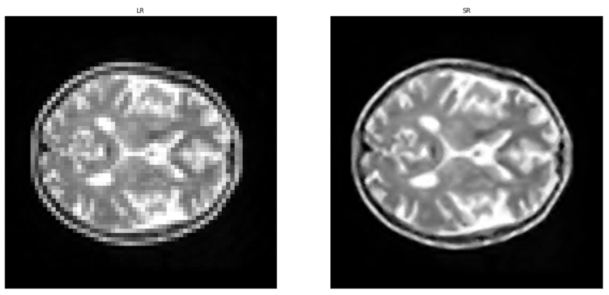

Fig. 7. Example upscaling using the RAISR filters (4x scaling). Low resolution image is on the left and the upscaled image is on the right

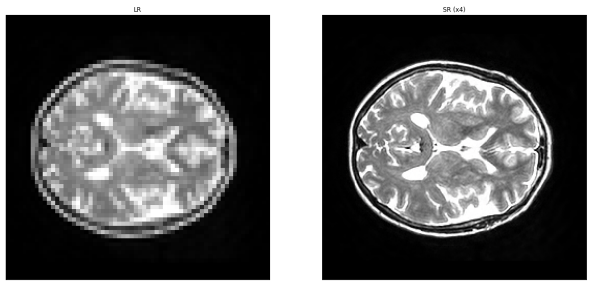

Fig. 8. Example upscaling using the EDSR model. Low resolution image is on the left and the upscaled image is on the right

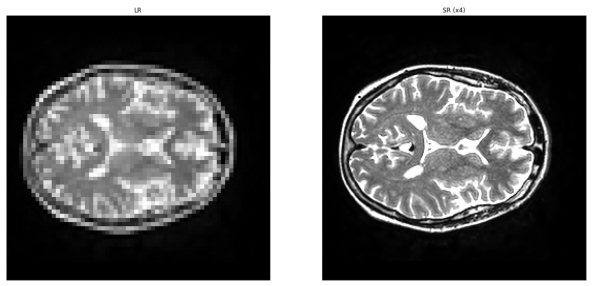

Fig. 9. Example upscaling using the WDSR model. Low resolution image is on the left and the upscaled image is on the right

References

[1] "Enhance! RAISR Sharp Images with Machine Learning,", P. Milanfar, Google AI Blog, 14-Nov-2016. [Online]. Available: https://ai.googleblog.com/2016/11/enhance-raisr-sharp-images-with-machine.html.

[2] "RAISR: Rapid and Accurate Image Super Resolution - IEEE Journals Magazine", Y. Romano, J. Isidoro, and P. Milanfar, IEEE Xplore, 15-Nov-2016. [Online]. Available: https://ieeexplore.ieee.org/document/7744595.

[3] "Handheld Multi-Frame Super-Resolution", B. Wronski, I. Garcia-Dorado, M. Ernst, D. Kelly, M. Krainin, C.K. Liang, M. Levoy, and P. Milanfar*, in ACM Transactions on Graphics, Vol. 38, No. 4, Article 28, July 2019

[4] "A+: Adjusted anchored neighborhood regression for fast super-resolution", R. Timofte, V. De Smet, and L. Van Gool, in ACCV, 2014

[5] A Fully Progressive Approach to Single-Image Super-Resolution", Yifan Wang, Federico Perazzi, Brian McWilliams, Alexander Sorkine-Hornung, Olga Sorkine-Hornung, Christopher Schroers, " CoRR, 2018

[6] "Interpolation Methods", *Dartmouth College*, 2005, link

[7] "A Brief Tutorial On Interpolation for Image Scaling", University of Wisconsin-Madison, 1999, link

[8] "Wide Activation for Efficient and Accurate Image Super-Resolution",Yu, Jiahui and Fan, Yuchen and Yang, Jianchao and Xu, Ning and Wang, Xinchao and Huang, Thomas S, arXiv preprint arXiv:1808.08718, 2018

[9] "Enhanced Deep Residual Networks for Single Image Super-Resolution", Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, Kyoung Mu Lee, arXiv:1707.02921, 2017

[10] "Deep Residual Learning for Image Recognition", Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, arXiv:1512.03385, 2015

[11] IXI Brain Development Dataset, Imperial College London,

[12] "Loss functions for neural networks for image processing", H. Zhao, O. Gallo, I. Frosio, and J. Kautz, arXiv:1511.08861, 2015

[13] A. M., "How to obtain a gaussian filter in python," Stack Overflow, 19-Jun-2013. Available: https://stackoverflow.com/questions/17190649/how-to-obtain-a-gaussian-filter-in-python.

[14] "Multiscale principal components analysis for image local orientation estimation", X. Feng and P. Milanfar, in Proc. 36th AsilomarConf. Signals, Syst. Comput., Nov. 2002, pp. 478–482.

[15] JalaliLabUCLA, "JalaliLabUCLA/Jalali-Lab-Implementation-of-RAISR," GitHub, 19-Aug-2018. [Online]. Available: https://github.com/JalaliLabUCLA/Jalali-Lab-Implementation-of-RAISR.

[16] M. Krasserm, Super-Resoltion, https://krasserm.github.io/2019/09/04/super-resolution/

[17] "psnr; Compute peak signal-to-noise ratio (PSNR) between images", Simulink https://www.mathworks.com/help/vision/ref/psnr.html