Robust Single Image Super-Resolution via Deep Networks with Sparse Prior

Ross Lawrence

1 Introduction

Single image super-resolution (SR) is the process of recovering an original high-resolution (HR) image from a low-resolution (LR) observed image. The applications for this process are as varied as the use of images, be it in the medical, security, entertainment, or scientific fields [1] [2]. However, because of the known variables in LR images being greatly outnumbered by the unknowns in the desired HR image, the problem of SR is ill-posed, limiting the ability for SR techniques to be used across a wide variety of images. A multitude of methods have been proposed for achieving SR, each leveraging different domain knowledge. Inspired by the success achieved by deep learning in computer vision tasks, the use of deep neural networks has begun to be used for image SR. With multiple layers of auto-encoders stacked together to make a robust model which performs SR, deep convolutional neural networks (CNN) and deconvolutional networks are designed to directly learn the non-linear mapping from LR space to HR [3].

Because these deep networks are built with generic architectures, they fail to utilize any previous knowledge about SR such as the natural image prior (i.e. the image is not random and has repeating patterns/components at different orientations and scales), and must instead learn everything from the training data. That is where the model created by the authors of this paper comes into play. In their paper *Robust Single Image Super-Resolution via Deep Networks with Sparse Prior* [4] they propose a simple model, which they named "sparse coding based network" (SCN), which works to combine a sparse representation prior with the capabilities of deep neural networks. They show that the proposed model is capable of notable improvement over the generic CNN model in recovery accuracy as well as required number of parameters.

2 Theory

The main reasoning behind the use of sparse representation in SR is the idea that the process which degrades a HR image to its LR state can be considered essentially linear. Let $Y \in \mathbb{R}^{n \times n}$ be a LR image upscaled by bicubic interpolation, and let $X \in \mathbb{R}^{n \times n}$ be the corresponding HR image. Let $y \in \mathbb{R}^{m_y}$ be a patch taken from Y which corresponds to the patch $x \in \mathbb{R}^{m_x}$ taken from X. The dimension m_y does not have to be the same as m_x when image features other than raw pixel values is used to represent patch y. It can be assumed that both x and y can be represented with respect to overcomplete dictionaries $D_x \in \mathbb{R}^{m_x \times n}$ and $D_y \in \mathbb{R}^{m_y \times n}$ and sparse linear coefficients $\alpha_x, \alpha_y \in \mathbb{R}^n$ respectively [5]. Because of our assumption that the degradation process from X to Y (and by extension x to y) is nearly linear, both patches can have the same sparse linear coefficients if D_x and D_y are defined correctly. With this in mind, we can define x in terms of y:

$$\alpha_x = \alpha_y = \alpha$$
$$x = D_x\alpha_x,\ y = D_y\alpha_y$$
$$x = D_x\alpha \text{ s.t. } \alpha = \underset{z}{argmin}||y - D_yz||_2^2 + \lambda||z||_1$$

where $||\cdot||_1$ is the l_1 norm and λ is the regularization coefficient. Thus, with this inclusion of sparse representation, the goal of SR becomes finding a suitable dictionary pair which minimizes the recovery error of x and y. The corresponding loss function L is defined as:

$$L = \tfrac{1}{2}(\gamma||x - D_xz||_2^2 + (1-\gamma)||y - D_yz||_2^2) \text{ where } 0 < \gamma \leq 1$$

The optimal dictionary pair can be found by minimizing the loss function over all the training LR and HR pairs:

$$\underset{D_x, D_y}{min}\ \tfrac{1}{N}\sum_{i=1}^{N} L(D_x, D_y, x_1, y_i) \text{ s.t.}$$
$$z_i = \underset{\alpha}{argmin}||y_i - D_y\alpha||_2^2 + \lambda||\alpha||_1,\ i = 1, 2, ..., N$$
$$||D_x(:,k)||_2 \leq 1,\ ||D_y(:,k)||_2 \leq 1,\ k = 1, 2, ..., K$$

However, because this minimization of the loss function is highly nonconvex, it is imperative that other methods be used in order to find suitable values for D_x and D_y. This is where the model proposed in the paper comes into play, as it attempts to automate the process of finding the values of α, D_x, D_y. One of the key components to this is the use of the Learned Iterative Shrinkage Threshold Algorithm (LISTA).

2.1 LISTA network

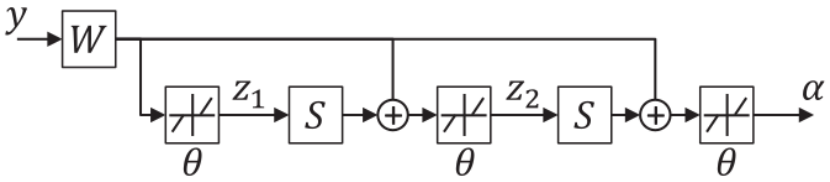

Fig. 1: Time-unforlded LISTA network with 2 iterations.

LISTA is an algorithm, based off of the Iterative Shrinkage Algorithm (ISTA), which serves to take an input vector $y \in \mathbb{R}^{m_y}$ and return an estimate of a sparse vector α given dictionaries $W \in \mathbb{R}^{n \times m_y}, S \in \mathbb{R}^{n \times n}$ and some coordinate-wise shrinkage function with threshold $\theta \in \mathbb{R}^n$. LISTA uses a time-unfolded version of ISTA truncated to a fixed number of iterations [6]. This method allows for the imposition of restrictions on S and W so as to minimize the approximation error to the optimal sparse code on a given input. For any input vector y, the output estimated sparse representation at each iteration k is:

$$z_{k+1} = h_\theta(Wy + Sz_k)$$

where:

$$[h_\theta(a)]_i = sign(a_i)\theta_i(|a_i|/\theta - 1)_+$$

for an input vector $a \in \mathbb{R}^n$ and index $i = \{1, ..., n\}$. The model for a LISTA network with 2 iterations can be seen in Figure 1. This limited number of iterations drastically reduces the computational complexity involved with training a model compared to the ISTA method, while still being able to give a good sparse estimation α.

2.2 Threshold Neuron Decomposition

A key way that the authors simplify the LISTA network is through the decomposition of the nonlinear neuronal layer with activation function h_θ. When training a model with the LISTA network, the two linear layers parameterized by $W \in \mathbb{R}^{n \times m_y}$ and $S \in \mathbb{R}^{n \times n}$ have to be updated along with the activation thresholds $\theta \in \mathbb{R}^n$ for the nonlinear neuronal layer. This complicates the learning algorithm. To restrict the number of adjustable parameters in the linear layers, the authors rewrote the activation function as:

$$[h_\theta(a)]_i = sign(a_i)\theta_i(|a_i|/\theta - 1)_+ = \theta_i h_1(a_i/\theta_i)$$

This allows for the decomposition of a neuronal layer into two linear scaling layers and a unit-threshold neuron. The weights of the two scaling layers are defined by the diagonal matrices θ (the $\times$ layer) and its element-wise reciprocal (the $\div$ layer). This allows for the combination of linear layers when training, which is discussed in the next section.

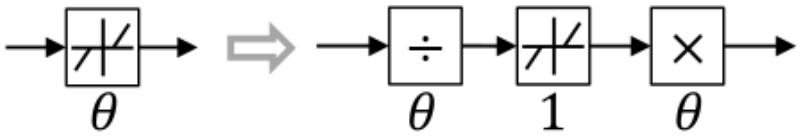

Fig. 2: A neuron with adjustable threshold θ decomposed into two linear scaling layers and a unit-threshold neuron.

3 Model

Utilizing the assumption of linear degradation between HR and LR images, along with the ability of the LISTA network to estimate a suitably sparse representation of input patch y, the authors of the paper were able to construct and deep network model which succeeded in SR over other comparable networks with the same number of parameters(see Figure 3).

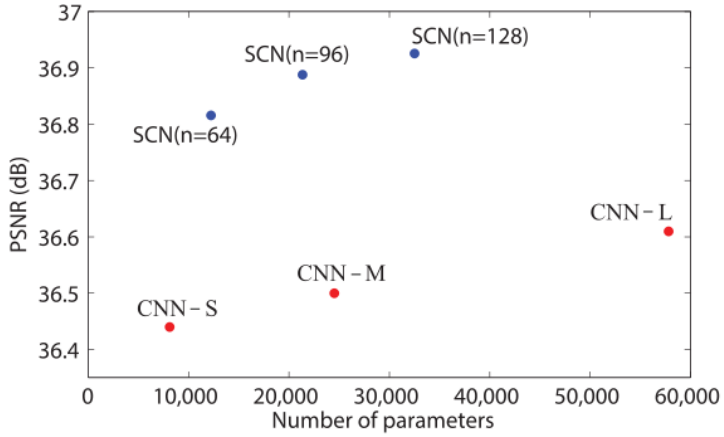

Fig. 3: PSNR comparison between the proposed model (SCN) at different D_x dictionary sizes and CNNs with equal number of parameters. SCN outperforms the generic CNN models with the same number of parameters on the Set5 [7] image dataset.

3.1 Configuration

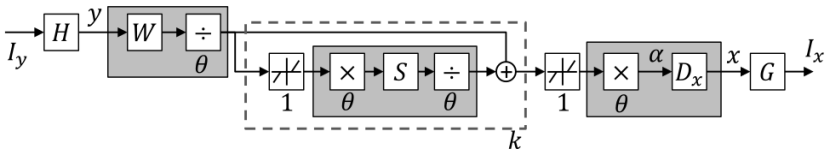

Fig. 4: The SCN model, where k is the number of iterations to perform for LISTA and each grey box represents consecutive linear layers which were combined to increase training efficiency.

Because the SCN model takes a LR image of $\mathbb{R}^{h\times w}$ and returns an SR image of size $\mathbb{R}^{r\times c}$, where $r < h$ and $w < c$, reflective padding is applied to the LR image such that the LR image before padding and the SR image are the same size. The input image I_y (i.e. bicubicly interpolated to be the same dimensions as the HR image) is padded and goes through a trainable convolutional layer H, which has m_y filters of kernel size $s_y \times s_y$. This results in the feature representation y of each $s_y \times s_y$ patch to have m_y dimensions. The LR patch y is normalized by its mean and variance, and the same mean and variance are used to restore the final HR patch x. Each y is then sent through the LISTA network with k iterations in order to estimate a sparse representation $\alpha \in \mathbb{R}^n$. The linear layers $W \in \mathbb{R}^{n\times m_y}$ and $S \in \mathbb{R}^{n\times n}$ and the nonlinear neuron layer with activation function h_θ (where $\theta \in \mathbb{R}^n$ is the activation threshold) are utilized in the LISTA network. Through the threshold neuron decomposition, however, the linear layers associated with θ are combined with layers W, S, and D_x (shown in Figure 4 as the gray boxes). The sparse representations α are then put through the linear layer D_x to get the HR vector x. Finally the recovered patches are put in the corresponding positions in the HR image I_x by the convolutional layer G which has a convolutional filter with spatial size $s_g \times s_g$ to combine all remaining patches by taking their weighted average, assigning appropriate weights to the overlapping areas. Thus the final prediction towards an HR image is output as I_x.

There are 5 trainable layers in the SCN network: 2 convolutional layers H and G, and 3 linear layers shown in the gray boxes. Each of the linear layers were implemented as convolutional layers with filters of spatial size 1×1.

3.2 Initialization

The authors proposed educated initialization weights for each layer in the model by understanding each layer's role in sparse coding. As the input to the model came from natural images, Harr-like gradient filters were used to initialize each of the filters of H which were then shifted into fixed positions (explained in Fixed Positioning section). As the linear layer component of G served to combine multiple channels together, uniform weights were used to initialize each layer. The remaining three linear layers (the gray boxes in Figure 4) are related to the dictionary pair (D_x, D_y) in sparse coding, and the relationship between LISTA and ISTA could be utilized. Both D_x and D_y were randomly set with Gaussian noise and the corresponding weights were calculated as they would be with ISTA:

$$w_1 = C \cdot D_y^T,\ w_2 = I - D_y^T D_y,\ w_3 = (CL)^{-1} \cdot D_x$$

where w_1, w_2, and w_3 correspond to the first, second, and third gray boxes in Figure 4 and $C, L \in \mathbb{R}$. According to the authors' empirical testing, they found that $L = C = 5$ performed best.

3.3 Training

The mean square error (MSE) was used as the cost function to train the network. Training consisted of mini-batches of image patches, which were rotated, scaled, and translated randomly. Parameters for the trainable layers were optimized using the standard back-propagation algorithm, with the optimization objective expressed as:

$$\min_{\Theta} \sum_i ||SCN(I_y^{(i)}; \Theta) - I_x^{(i)}||_2^2$$

where $I_y^{(i)}$ and $I_x^{(i)}$ are the i-th pair of LR/HR images in the training data, and $SCN(I_y; \Theta)$ denotes the predicted HR image for I_y with parameter set Θ. Each layer in the model had zero biases.

4 Implementation

4.1 Model Construction

In the paper, the models were trained using the CUDA ConvNet package. The authors provided a github repository with a file containing the weights of one of their models, as well as code to test that the weights generated by their model worked (`https://github.com/huangzehao/Super-Resolution.Benckmark`). However as the github repository did not include the actual model they trained, and due to the lack of response to my emails enquiring about the code, in order to implement and test the code I had to construct it myself. To this end, I constructed the proposed model in python using the Pytorch library [8]. The jupyter notebook containing my model and a copy of the parameters I found through training can be found here: `http://github.com/lawreros/SCN`.

4.1.1 Fixed Positioning

One seemingly unique method was used by the authors of this paper when constructing the layer H in their model, which they explained as:

"...we implement the LR patch extraction layer H as the combinations of two layers: the first layer has 4 trainable filters each of which is shifted to 25 fixed positions by the second layer. Similarly, the patch combination layer G is also split into a fixed layer which aligns pixels in overlapping patches and a trainable layer which aligns pixels in overlapping patches and a trainable layer whose weights are used to combine overlapping pixels."

While I searched for other such implementations of this tactic in other works, as well as discussing with individuals more experienced with convolutional networks, I was unsuccessful in getting a meaningful explanation. It was only when I looked at the weights the provided on their github repository that I found the answer. As the convolutional kernel used in their model for H was 9×9, if you were to instead use a 5×5 kernel and shift it one step along either the height or width of a 9×9 kernel, you could get 25 unique 9×9 kernels. For example, one 9×9 kernel would have the first 5 rows and columns have nonzero values $b \in \mathbb{R}^{5 \times 5}$, while the next would have rows 2 through 6 and the first 5 columns have the values b. Thus, if you were to preform this operation to 4 different 5×5 kernels, you would get 100 unique filters shifted to "fixed position" while only changing the weights of the 4 original filters. Then for the G layer, you would perform the opposite of the shifting to fixed positions of each channel, so for each channel with dimensions 48×48 a 44×44 subimage was taken (see the `reassemble2` function in the code for clarification). The $25 \times 44 \times 44$ channel stack was then run through a 1×1 convolutional filter to get a 44×44 HR image with one channel (see Figure 5). The authors stated that this did not affect the performance of the model.

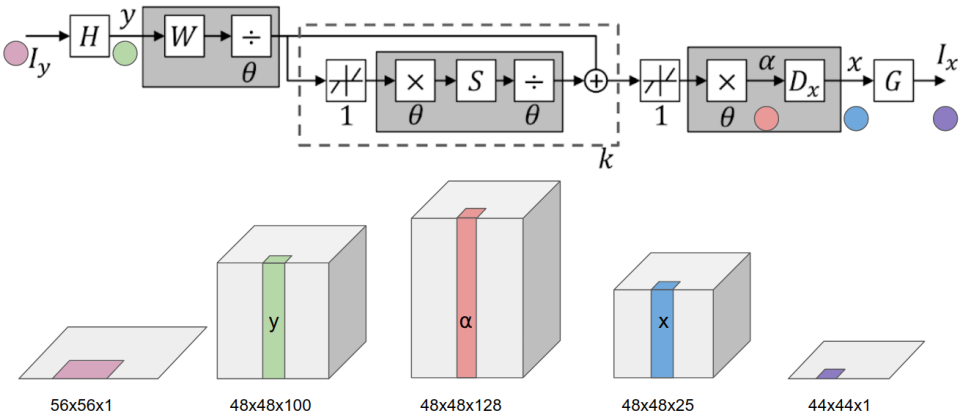

Fig. 5: A visual aid for understanding the dimensions of the outputs at various stages of the model. The dimensions for the cubes are written as $height \times width \times channels$, which is contrary to the conventions in Pytorch, but was chosen for a more clear visual aid.

4.1.2 Parameters

In an effort to replicate the results of the paper, I decided to reuse many of the parameters set by the authors. These included:

- Convolutional layer H with 4 Haar-like gradient filters of spatial size 5×5 shifted to 25 fixed positions, resulting in $y \in \mathbb{R}^{100}$
- $D_x \in \mathbb{R}^{128 \times 25}$ and $D_y \in \mathbb{R}^{100 \times 128}$ initialized with Gaussian noise with mean 0 and variance 1
- The convolutional filter component of G being paramaterized by a vector $[0.04, 0.04, ..., 0.04]$ of length 25. This resulted in equal weighting of each channel as the sum was take for the output image I_x
- The number of iterations in the LISTA network was set to 1, a change that the authors of the paper stated did not have a significant effect on the model's accuracy

4.2 Training and Testing Data

4.2.1 BSD100, Set5, Set14, and BSD200

In order to recreate the model discussed in the paper, I trained my model on the images from the BSD100 dataset [9] (as I couldn't find the set of Set91 images that was discussed) and tested with the Set5 [7], Set14 [10], and BSD200 [9] datasets. As the model only works to estimate the SR of the luminance layer of an image, each LR and corresponding HR image from the datasets were first converted from RGB to YCbCr and had their luminance layer isolated and saved as a separate image. These luminance images were then randomly rotated, scaled, and translated in order to increase the robustness of the model. The resulting transformed images were then saved as png files for training and testing, with each original luminance image having 15 different copies that underwent different transformations.

After the LR images were bicubicly interpolated by a factor of 2 to match the size of the HR images, 44×44 pixel patches of each image were used in inputs into the model for training. As per the protocol discussed in the paper, a standard stochastic gradient descent algorithm was used with mini-batches of 64 images were used for training.

4.2.2 Structural MRI Data

In order to both train and test the model on MRI data, I used structural MRI images of my brain from three different studies that I have participated in. The choice of only using my MRI was to serve as a proof of concept, as the images used in the training and testing would be "similar" to eachother. If the model failed to perform SR on this dataset, then it would most likely fail on a more diverse MRI dataset.

Using python's `imageio` library [11], png images were created of the sequential "slices" along the sagittal, coronal, and axial axes of each MRI. Because the data from the MRI files consisted of intensity values ranging from 0 to $> 50,000$, in order to get the values into the [0,255] range of values found in a png I divided each pixel by the maximum intensity value present in that slice of the MRI file and multiplied the resulting value by 255. After that, the images were put through the same process as the BSD100 dataset and training image patches were created with randomized transformations.

5 Results

Due to time are resource constraints, I was unable to replicate all of the different evaluations discussed in the paper. However, I was able to replicate the performance of the SCN model in SR estimation of bicubicly interpolated LR images from common SR datasets. I also expanded on the use-case of the model by applying it to structural MRI images, in an attempt to explore the sparse representation of brain images.

5.1 Paper Replication

After training the model on the BSD100 dataset until it the decrease in MSE plateaued across all mini-batches, the model was used to estimate SR images for the 3 other datasets. Instead of inputting 44×44 patches from the LR image, the whole LR image was up-scaled by a factor of 2 using bicubic interpolation and passed through the model. Peak signal to noise (PSNR) was used as the metric to measure performance of the estimated HR image compared to the actual HR image. As a metric for comparison, I also calculated the PSNR for the input LR image compared to the actual HR image, as any improvement could be viewed as a success. The mean PSNR scores for each dataset can be found in Table 1.

Dataset	Bicubic PSNR (dB)	SCN PSNR (dB)
Set5	33.93	38.89
Set14	31.91	33.29
BSD200	29.98	31.68

Tab. 1: PSNR results for the SR using the SCN model trained on BSD100 images. The SCN model resulted in a higher PSNR than the original bicubic interpolated images for the vast majority of images.

The estimated HR images from the trained model displayed higher PSNR over the up-scaled LR images not only on average, but individually as well. There were only 2 images in the BSD200 dataset that had PSNR values < 0.02 dB below their bicubic counterparts. These results were in line with what was found by the authors of the paper (see Table 3 from the paper).

5.2 MRI Super Resolution

5.2.1 BSD100 Model

To test the generalizability of the SCN model trained on the BSD100 dataset, I tested its ability to estimate HR images from the MRI data. While it did perform slightly better than the bicubic interpolation benchmark, it did not perform as well as it did with the other image datasets (Table 1), instead having a mean PSNR of 32.92 dB compared to the bicubic benchmark of 32.46. While the mean PSNR was greater, not all estimated HR images had higher PSNR than their bicubic counterparts, with some having PSNR values significantly lower. This indicated that the improvements were not consistent from image to image and the model as a whole could not be trusted to reliably estimate HR MRI images.

5.2.2 MRI Trained Model

With the SCN model trained on the BSD100 dataset not performing as well as expected, I decided to train a new model using MRI coronal images. The idea behind this was to create a model more sensitive to the patterns found in the structure of the brain. Using the same initialization parameters that were used when training the SCN on the BSD100 dataset, I attempted to train the SCN model using image patches from the coronal images. However, despite training on over $3,500$ mini-batches of 64 images, the model was unable of attaining a PSNR value consistently greater than that of the bicubic interpolation benchmark (32.2145 vs 32.463).

With the SCN model using the parameters specified in the paper not succeeding, I decided to disregard two computation-saving steps proposed by the authors. Instead of only training 4 filters with 5×5 kernel size in layer H and shifting them to 25 fixed positions each, I instead had layer H be 100 filters with kernels of size 9×9 initialized randomly by Pytorch. I also increased the number of iterations of the LISTA network from 1 to 2 in an attempt to get a more sparse representation of y. With these two changes the SCN model was able to outperform the bicubic benchmark on all coronal images of the training and testing set.

Curious to the generalizability of the SCN model with regard to images of the MRI taken from different orientation (i.e. sagittal and axial), I tested the model. What I found was that the SCN model successfully estimated an SR image with a higher PSNR than the bicubic benchmark for both orientations, with axial performing better than sagittal (Table 2). This finding is intuitive, as there are consistent patterns that exist across MRI slice orientation, such as the curves

of the gyri and sulci or the cylindrical structure of the brain stem. Example output SR images from each orientation

Orientation	Bicubic PSNR (dB)	SCN PSNR (dB)
Coronal	32.463	34.562
Axial	37.5634	38.0625
Sagittal	32.2128	32.46735

Tab. 2: Mean PSNR results for the SCN model trained on coronal slices in estimating SR images for slices from other orientations.

can be seen in Figures 6-8. Upon visual inspection, the output images from the SCN model look noticeably more clear and like the target HR images.

6 Discussion

It's clear that the model proposed by the authors, which leveraged the strengths of sparse coding with the capabilities of deep networks, is successful in producing good SR results. In my limited experience, I was able to train a model which successfully increased the PSNR of bicubicly interpolated images consistently across multiple datasets. My success with MRI SR also shows that there is potential is use of the model in the medical imaging field.

6.1 The Many Mistakes in Development

The biggest difficulty in the completion of this project came from my on inexperience with machine learning software. While I had worked with pre-made datasets and simplified neural networks before, this was the first time I had coded something non-standard. As such, there were a multitude of mistakes in development which drastically hindered the amount I was able to accomplish with my limited time and resources. Everything from learning rate to properly randomizing training data had to be done through trial and error, where testing took hours if not days. It's the main reason why I haven't provided a figure which shows the learning rate of the SCN models, as I a continually changed the learning rate until I reached sufficient performance. Because of the delays I was unable to explore and experiment with the models as much as I had originally intended. This includes the SCN cascade models, which involved having several SCN models in sequence, each of which magnifies its input by a fixed amount. The authors of the paper found that consecutive SCN models whose cumululative scaling was some factor f would output SR images with significantly better PSNR values than a single SCN model whose scaling factor was f. Exploration the benefits of SCN cascades on MRI data would have required significant time to train that I didn't have. There was also the exploration of the performance of the SCN model with images with more complex degradation than just bicubic interpolation.

6.2 Paper Shortcomings

The authors of the paper were fairly clear and thorough in their model's description and the tests that were run to measure its performance. The only glaring issues were through the lack of providing the code they used for their model. A lot of difficulties that I had with understanding the fine details of their model would have been immediately alleviated with the ability to look at their code directly, such as the shifting of filters into fixed positions that occurs in layer H, or the shifting of channels in layer G. Access to the CNN models (or just the weights) that they trained would have also been useful in comparing the performance of the SCN model. Allowing access to their code lowers the barrier for verification of the paper and expansion of its use case as others can easily train it on new data.

6.3 Future Ideas/Development

Given more time, there are a multitude of ideas which I would like to explore with regard to the SCN model. Aside from the effect of different training parameters, like the number of iterations in the LISTA network, the most pressing is the exploration of increasing the resolution of raw intensity values for data that has been trilinearly interpolated. As I discussed in the Structured MRI Data section, I chose to normalize the MRI intensity values and convert the data into pngs before using them as inputs in the model. This was done as a simplification method for my initial model training, attempting to keep it closer to the model described in the paper, however there is information loss that occurs due to pngs requirements to have integer pixel values. The choice to normalize across each slice instead of across the whole 3D structural image was also done for simplification sake. Given more time, I would like to experiment with training a SCN which has inputs consisting of raw intensity values.

With regards to training different models, I would like to experiment further with the inclusion of patches from different orientations on the overall performance of the SCN. By training on only coronal images, I may have hindered

the ability of my model to estimate SR images of MRI data. The use of data from different orientations may increase the robustness of the SCN model, as patterns in MRI data along the sagittal plane may appear in the coronal or axial planes (and vice-versa). This idea was given some validity from the ability of the SCN model I trained on coronal MRI data performing well on axial and sagittal images.

The effect on the scanner resolution would also be very interesting to investigate. While the data I trained on had voxel sizes of $1mm^3$, there are a variety of voxel sizes used for MRI images. I think it would be interesting to see how a model could handle voxels which are not perfect cubes. On top of this would be the exploration of the SCN models performance on both function and diffusion MRI data, instead of just structural data. Would there be enough overlap in the different imaging modalities that a single SCN could be trained to perform effectively on all of them?

Regardless, I do not intend on stopping my experimentation with the SCN model after this assignment. There is plenty of work still left to do that I plan on pursuing in my own time, and I look forward to having the time to experiment with my SCN model.

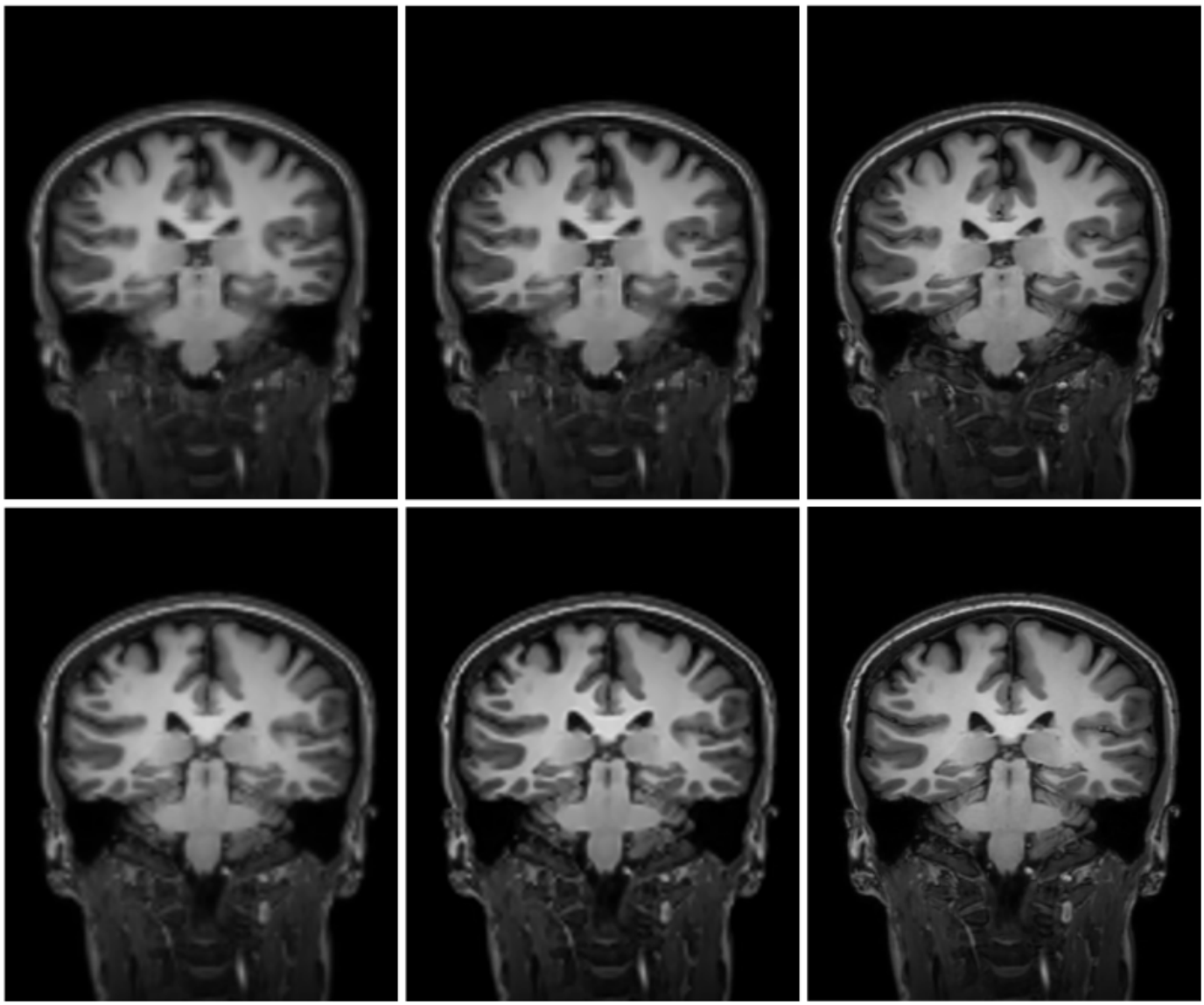

Fig. 6: Two examples of the results from running coronal images through the coronally trained SCN model. (Left) The bicubicly interpolated LR image by a magnitude of 2, (Center) the output of model when the bicubicly interpolated image is used as an input, (Right) the ground-truth HR image. The PSNR values for the images are (bicubic = 34.8923, SCN = 33.4427) and (bicubic = 33.835, SCN = 34.8736) for the top and bottom row, respectively.

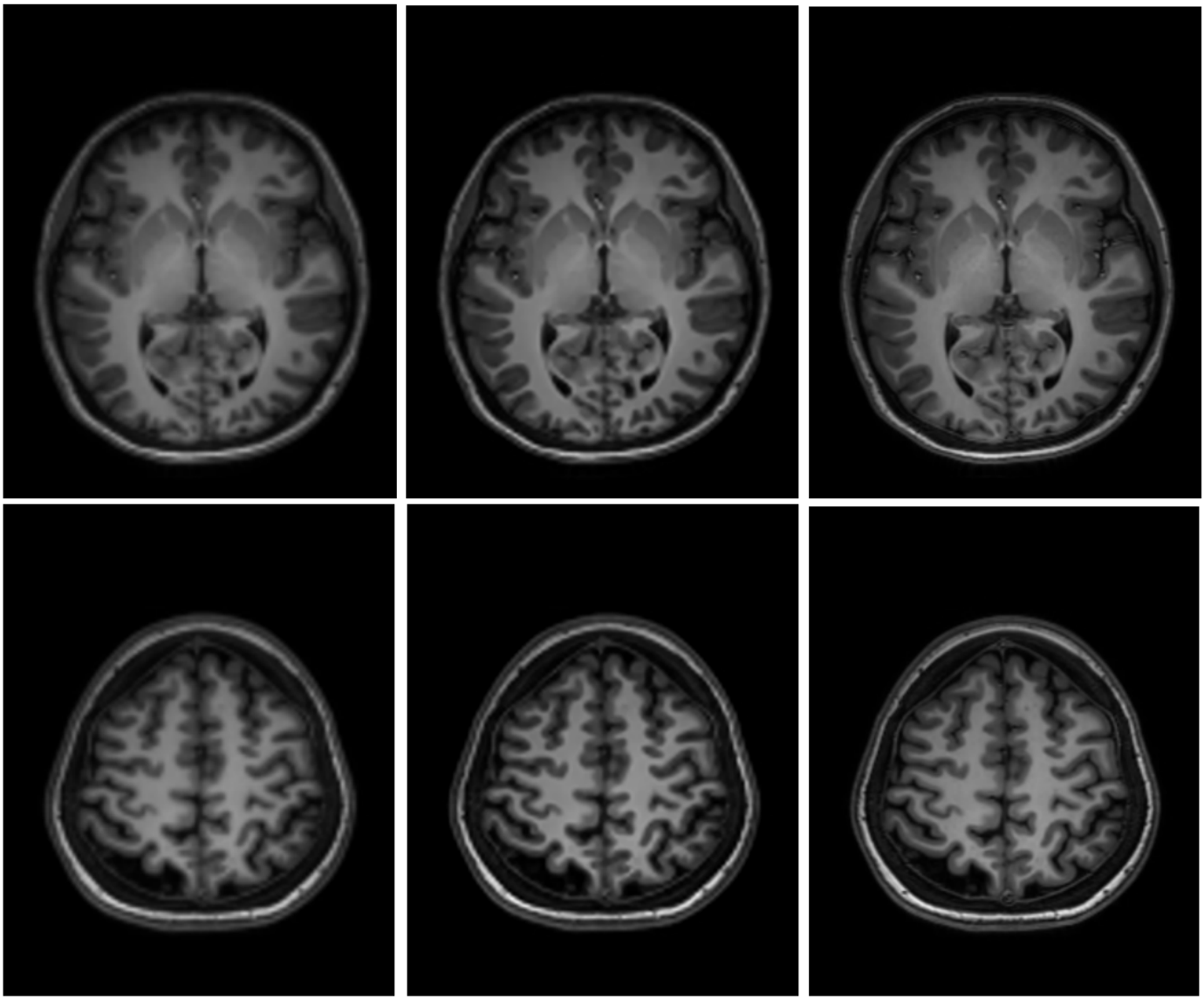

Fig. 7: Two examples of the results from running axial images through the coronally trained SCN model. (Left) The bicubicly interpolated LR image by a magnitude of 2, (Center) the output of model when the bicubicly interpolated image is used as an input, (Right) the ground-truth HR image. The PSNR values for the images are (bicubic = 36.1841, SCN = 37.0517) and (bicubic = 34.8884, SCN = 36.5133) for the top and bottom row, respectively.

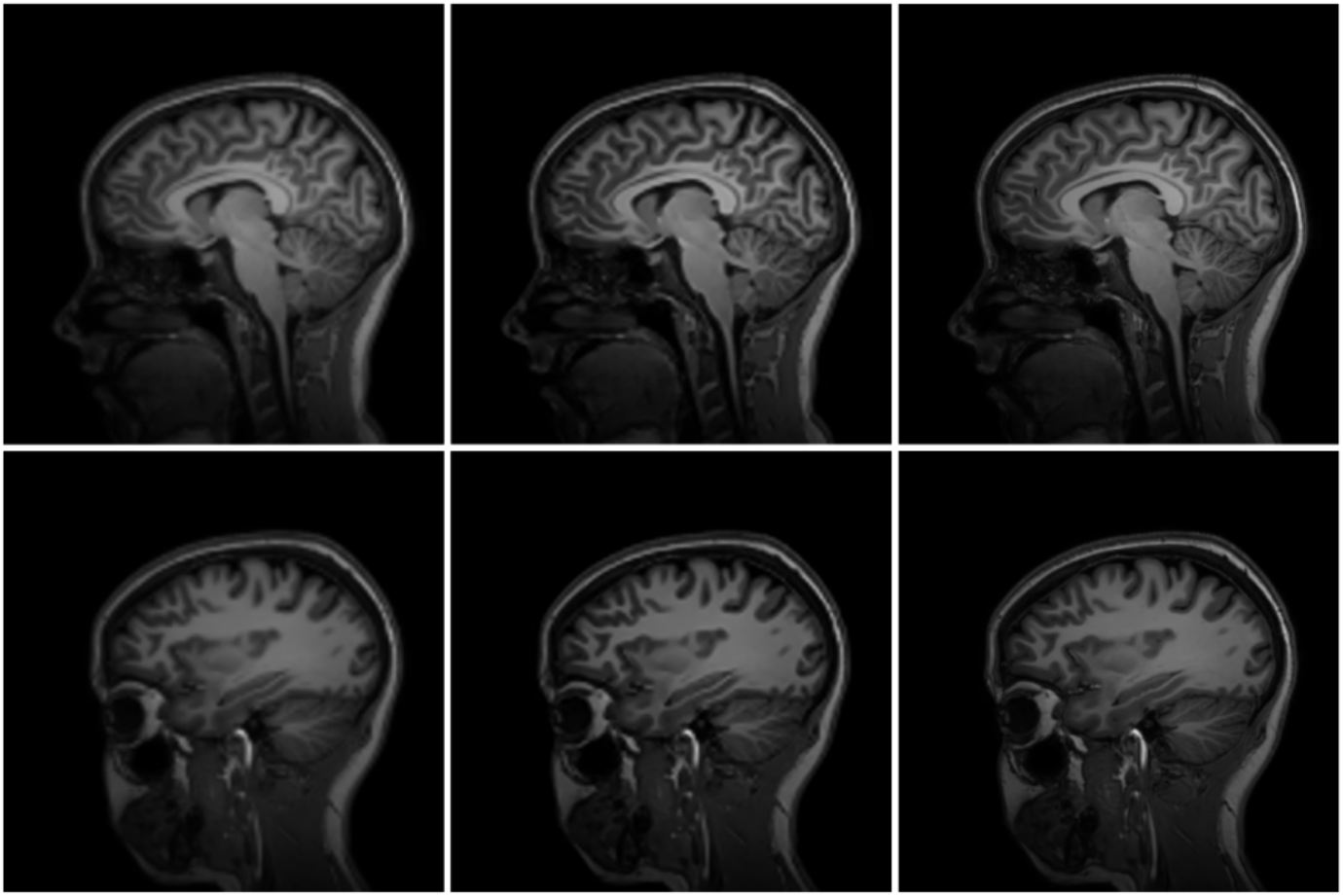

Fig. 8: Two examples of the results from running sagittal images through the coronally trained SCN model. (Left) The bicubicly interpolated LR image by a magnitude of 2, (Center) the output of model when the bicubicly interpolated image is used as an input, (Right) the ground-truth HR image. The PSNR values for the images are (bicubic = 36.1970, SCN = 37.2298) and (bicubic = 38.3143, SCN = 39.3143) for the top and bottom row, respectively.

References

[1] S. Baker and T. Kanade. Limits on super-resolution and how to break them. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(9):1167–1183, 2002. doi: 10.1109/TPAMI.2002.1033210.

[2] Zhouchen Lin and Heung-Yeung Shum. Fundamental limits of reconstruction-based superresolution algorithms under local translation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(1):83–97, 2004. doi: 10.1109/TPAMI.2004.1261081.

[3] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Learning a deep convolutional network for image super-resolution. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 184–199, Cham, 2014. Springer International Publishing. ISBN 978-3-319-10593-2.

[4] Ding Liu, Zhaowen Wang, Bihan Wen, Jianchao Yang, Wei Han, and Thomas S Huang. Robust single image super-resolution via deep networks with sparse prior. pages 3194–3207, 2016. doi: 10.1109/TIP.2016.2564643.

[5] Jianchao Yang, John Wright, Thomas S. Huang, and Yi Ma. Image super-resolution via sparse representation. *IEEE Transactions on Image Processing*, 19(11):2861–2873, 2010. doi: 10.1109/TIP.2010.2050625.

[6] Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 399–406, Madison, WI, USA, 2010. Omnipress. ISBN 9781605589077.

[7] Marco Bevilacqua, Aline Roumy, Christine Guillemot, and Marie line Alberi Morel. Low-complexity single-image super-resolution based on nonnegative neighbor embedding. In *Proceedings of the British Machine Vision Conference*, pages 135.1–135.10. BMVA Press, 2012. ISBN 1-901725-46-4. doi: http://dx.doi.org/10.5244/C.26.135.

[8] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[9] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, volume 2, pages 416–423 vol.2, 2001. doi: 10.1109/ICCV.2001.937655.

[10] Roman Zeyde, Michael Elad, editor="Boissonnat Jean-Daniel Protter, Matan", Patrick Chenin, Albert Cohen, Christian Gout, Tom Lyche, Marie-Laurence Mazure, and Larry Schumaker. On single image scale-up using sparse-representations. In *Curves and Surfaces*, pages 711–730, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[11] Almar Klein, Sebastian Wallkötter, Steven Silvester, Anthony Tanbakuchi, Paul Müller, Juan Nunez-Iglesias, actions user, Mark Harfouche, Antony Lee, Matt McCormick, OrganicIrradiation, Arash Rai, Ariel Ladegaard, Tim D. Smith, Ghislain Antony Vaillant, jackwalker64, Joel Nises, Miloš Komarčević, rreilink, lschr, Hugo van Kemenade, Maximilian Schambach, Chris Dusold, DavidKorczynski, Felix Kohlgrüber, Ge Yang, Graham Inggs, Joe Singleton, Michael, and Niklas Rosenstein. imageio/imageio: v2.13.3, December 2021. URL `https://doi.org/10.5281/zenodo.5768245`.