Re-implementation of Paper Called "Unrolled Optimization Using Deep Priors"

Shashank Balla 105349909[a], Ege Cetintas 405429037[a], Berkin Durmus 105429086[a], and Dilek Mirgun Yalcinkaya 105332027[a]

[a]University of California, Los Angeles

The aim of the project was to reproduce the results of (1). The empirically derived insights presented in the paper, specific for denoising and deblurring, were implemented as per the proposed framework. The performance of these implementations was evaluated in terms of the PSNR of the result vs. its groundtruth.

1. Introduction

Most solutions to common problems in computational imaging involve solving an inverse problem which assumes the model follows a known distribution pattern. In general, predetermined priors were implemented along with iterative optimization methods.

This work proposes a general framework which uses prior knowledge of the image formation within the optimization method, by integrating knowledge both from classical and deep approaches (1). In (1), they formulate their approach as a maximum-a-posteriori (MAP) estimation under a Bayesian model. The proposed framework assumes the formation of the original image (x) is through an unknown that is drawn from a prior distribution $\Omega(x)$ over empirically derived parameters θ. The various optical processes effecting the image during capture is modeled as a linear operation Ax. Moreover, the sensor noise in the measurement of the resulting image at the sensor y is modeled as an output of a noise distribution $w(Ax)$.

Iterative algorithms are commonly used to solve problems modeled this way. In these algorithms, at the end of each iteration, in the data-step, a mapping incorporating the prior information into the result generated from the network is applied as long as the stopping conditions for convergence are met.

Citing low requirements on accuracy, this paper proposes fixing the number of iterations so that this iterative process can be modeled as an explicit function of the initial point to speed up the optimization process.

The MAP estimation is based on selecting x so as to maximize the likelihood probability $P(y;x)$, given information of the posteriori probability $P(y|x)$. Their relation is given by the Bayes' rule as, $P(y,x) = P(y|x)P(x)$. Since x is modeled as a realization of a distribution $\Omega(\theta)$ over the hyper-parameters (θ), $P(x) = P(x;\theta)$. Thus, the problem is formulated as:

$$argmax_x(P(y|Ax).P(x;\theta))$$

taking negative log likelihoods of the same, we have,

$$argmin_x(f(y,Ax) + r(x,\theta))$$

The model hyper-parameters used in the data-step represent the prior information on image formation. These parameters give the network a superficial understanding of the task at hand. Thus, the model capacity is limited by the number of unique representations of this prior, and determines the range of the application of the proposed framework.

2. Related Work

The papers we reviewed for this work are, according to their motivations, as follows:

Specific applications: Unrolled optimization networks can be used in specific applications such as MRI (2) where architecture discussed in this paper is taken as basis and incorporated for increasing and extending the flexibility of neural networks in reconstruction using a band-pass filter. We have reviewed this paper to get an idea on how the architecture we implemented can be extended to other applications. Another application proposed in (3) applies only a single deconvolution step with the aim of deblurring which can be considered as a one-iteration version of the network being discussed in this paper.

Other unrolled optimization networks: In (4), authors propose a method to train a specific architecture with a fixed depth to approximate sparse

code. This can be thought as a preliminary application of the work that was presented by (1). Finally, we also reviewed (5), which is also cited by (1), in which proximal methods are solved with deep recurrent neural network and the proposed method is also shown with image denoising, depth refinement and optical flow estimation.

3. Approach/Algorithm

A. Training Approach. We went through three different steps while deciding how to train our network. Firstly, we inputted the complete image in RGB to the network. In other words, we trained our network with RGB images in deep layers. The problem here was due to the nature of natural images where blue channel is generally suppressed and not very common. As a result, we obtained results which is denoised for specific values. In other words, when we looked at the channels separately we observed that red and green channel is denoised but blue channel was not denoised at all. This was due to using same filters and activation functions for all channels although the intensity of blue color is not that much when compared to red and green. Such differences between channels caused learning of the parameters to be more oriented towards red and green channels. As a result, blue channel has remained under the activation threshold.

Having realized this issue, we decided to train 3 different networks for each channel to avoid this and we continued with separating the channels and inputted each channel to a different deep network. However, we firstly did the mistake that we added the noise to each channel independently. Having noticed that, we added the noise beforehand in the RGB format and then separated it to different channels. We trained 3 different networks to get good results.But this time the problem turned out to be computationally very expensive since we were using computers having single GPU and our training time was three times of before. Trying it with around 500 epochs (this was the maximum our computers could handle at once), we obtained results with which we were not satisfying. Another issue was implementing the problem in MATLAB which has a slower speed than Python. Then, we have reviewed (6) and realized that it is a broader problem than we can handle in the scope of this project and denoising different channels is not an easy task. Eventually,

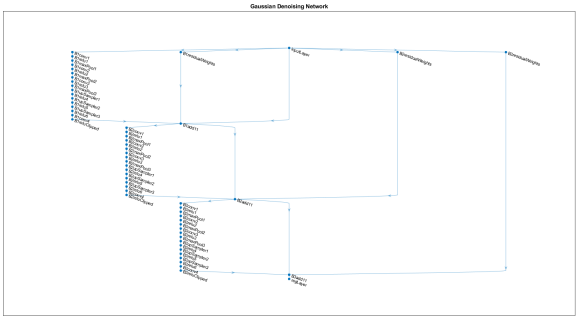

Fig. 1. Structure for the Denoising Task

we proceeded with using gray-scale images which is indeed the format that the results are produced in our reference paper and obtained results without any problem.

B. Denoising. For this task, we first generated our data-set by adding noise to the BSDS500 image set using the *imnoise* function in MATLAB. The parameter for standard deviation was set to 0.01. In our first attempt, we worked on full colour images. Here, we constructed our CNN prior, similar to an autoencoder structure, as delineated in Table 1. Our implementation of the CNN follows a common structure in this field, used to extract features from images. The general idea behind the auto encoder CNN is simple. In general, one filter cannot incorporate the complexity to represent all patterns emerging in the learning process. Hence, multiple filters are implemented to account for more complex pattern recognition. This will help us to extract more aspects of the images, namely features. We first decrease the size of the images, which is encoding, and then increase their size in the decoding process. This issue can be handled with the help of the up samplers and this can be worked out by the input padding. So, at the output we have a result with the same size as the original image, this result is used as a prior for the data-step in the iteration (See Fig.1).

On constructing the CNN following this architecture, a copy of it was made for every iteration. We then made the required connections between them as suggested by the framework in Fig.2 which is proposed in (1). Some of the important ones among these were the residual connections to forward the results across iterations, so the block in every iteration has access to the output of the preceding block and the original input to the network. The main idea behind this is to inculcate memory into our network.

Algorithm 5 ODP proximal gradient denoising network

1: Initialization: $x^0 = y, \alpha_k = C^{-k}, C_0 > 0, C > 0$
2: **for** $k = 0$ to $N - 1$ **do**
3: $\quad x^{k+1/2} \leftarrow \text{CNN}(x^k, \theta^k)$.
4: $\quad x^{k+1} \leftarrow (\alpha_k y + x^k + x^{k+1/2})/(\alpha_k + 1)$.
5: **end for**

Fig. 2. Algorithm used in denoising (1)

Table 1. Deep Network Structure for Denoising

Position	Layer Type	Features
1	Conv. Layer	3-input channels, 16 filters, edge padding
2	Activation Layer	ReLU function
3	MaxPooling Layer	Pool size 2, edge padding, stride 2
4	Conv. Layer	3-input channels, 8 filters edge padding
5	Activation Layer	ReLu function
6	MaxPooling Layer	Pool size 2, edge padding, stride 2
7	Conv. Layer	3-input channels, 8 filters, edge padding
8	Activation Layer	ReLU function
9	MaxPooling Layer	Pool size 2, edge padding, stride 2
10	Upsampler Conv. Layer	2-input channels, 8 filters
11	Activation Layer	ReLU function
12	Upsampler Conv. Layer	2-input channels, 8 filters
13	Activation Layer	ReLU function
14	Upsampler Conv. Layer	2-input channels, 16 filters
15	Activation Layer	ReLU function
16	Conv. Layer	3-input channels, 1 filters, edge padding
17	Activation Layer	clipped ReLu 1.0

Thus, these residual connections will, add memory to our network, as well as allow the backward gradients to flow from these connections allowing for easier updates of weights. This is very important considering the long computation time due to slow updates in weights, which is a common issue in such deep networks.

The paper provides few empirically derived values for these network hyperparameters, but we chose to implement them as learnable parameters within our network. Our intention behind this was to increase the complexity of the network by providing leeway for different weights to settle in different residual connections. Few other notable connections in our network are the output-input connections between every iteration and the ones that give the input image to all iterations and make. Every iteration must give a weighted summation of its input, original input image to the network and output of the CNN block. Table 1 is an image of the final network constructed for the denoising task.

C. Deblurring. The approach implemented in the paper is similar to one of the techniques discussed in class for deconvolution which is known as Wiener Deconvolution. This is nothing but an extension for the deconvolution idea. In other words, if we directly set deconvolution result as in the regular deconvolution (FT of output of the system/FT of impulse response), it will be too sensitive to noise. Furthermore, the zeros in the point spread function (PSF) will negatively affect the result. To overcome this, we can use Wiener filtering, which we did after the CNN priors output. Normally, setting the α_k parameter as learnable gives better results, but it makes the network too complicated to implement and it is computationally too expensive, so this time we incorporated them as constants and assigned the values proposed in the paper. Our network structure is slightly changed for this operation. Rather than using 4 iterations(which we did in denoising) we used 8 iterations for motion deblurring. We also changed the structure of our CNN. It is still an auto-encoder structure but with fewer layers. Our CNN structure can be seen in Table 2 and we followed the structure in Fig. 4. In this structure before giving summation of iteration output, iteration input and original input to the proceeding iteration we need to make some operations with them, which is also known as the deconvolution operation. This operation can be seen in Fig. 6. The overall network structure can be seen in Fig. 5. The training process was different for the different types of blur, which is explained in detail in the following subsections. Information on few notable layers from our structure are explained as follows.

Residual Nodes - These nodes are serving for the same purpose as the denoising. In other words, this is for memory and easy gradient propagation.

Deconvolution Node (DataStep as in paper (1)) - This nodes takes the fourier transform of the input make the wiener deconvolution operation about which we talked above and which can also be seen in Fig.5.

Output Node(RegLayer) - This is calculating the Mean Squared Error between original image and the prediction.We prefered using MSE as a measure of loss.

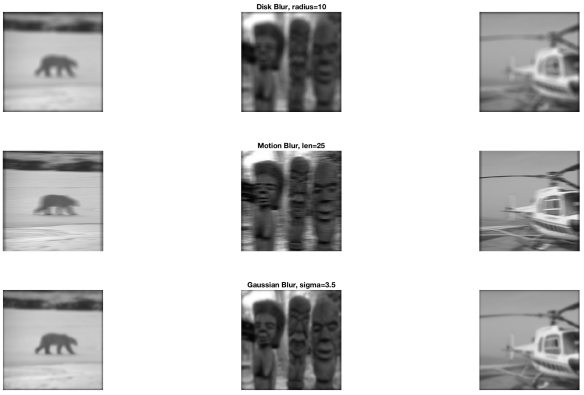

Fig. 3. Examples from blurred images

Algorithm 6 ODP proximal gradient deblurring network. $\mathcal{F}$ is the DFT, k is the blur kernel, and $\mathbf{K}$ is the DFT of k

1: Initialization: $x^0 = k^T * y$, $\alpha_k = C^{-k}$, $C_0 > 0$, $C > 0$
2: **for** $k = 0$ to $N - 1$ **do**
3: $\quad x^{k+1/2} \leftarrow \text{CNN}(x^k, \theta^k)$.
4: $\quad x^{k+1} \leftarrow \mathcal{F}^{-1}\text{diag}(\alpha_k|\mathbf{K}|^2 + 1)^{-1}\mathcal{F}(\alpha_k k^T * y + x^k + x^{k+1/2})$.
5: **end for**

Fig. 4. Algorithm used in deblurring (1)

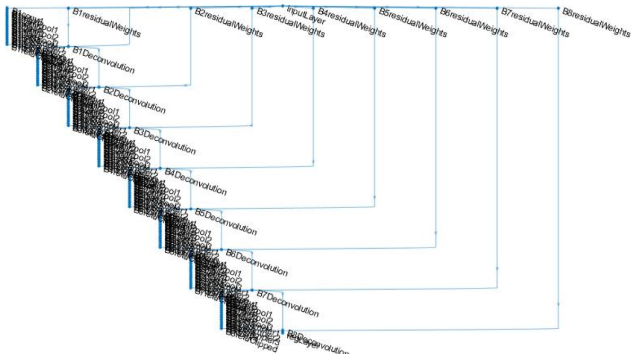

Fig. 5. Our deblurring structure

Table 2. Deep Network Structure for Deblurring

Position	Layer Type	Features
1	Conv. Layer	1-input channels, 16 filters, edge padding
2	Activation Layer	ReLU function
3	MaxPooling Layer	Pool size 2, edge padding, stride 2
4	Conv. Layer	3-input channels, 8 filters edge padding
5	Activation Layer	ReLu function
6	MaxPooling Layer	Pool size 2, edge padding, stride 2
7	Upsampler Conv. Layer	2-input channels, 8 filters
8	Activation Layer	ReLU function
9	Upsampler Conv. Layer	2-input channels, 8 filters
10	Activation Layer	ReLU function
11	Upsampler Conv. Layer	2-input channels, 16 filters
12	Activation Layer	ReLU function
13	Conv. Layer	3-input channels, 1 filters, edge padding
14	Activation Layer	clipped ReLu 1.0

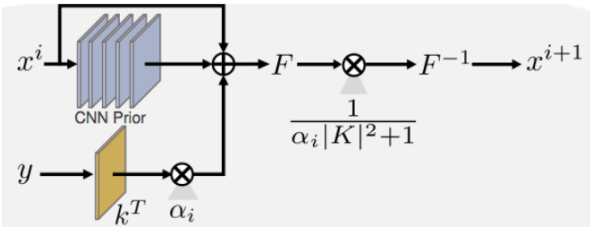

Fig. 6. Deblurring Structure (1) followed from (1)

C.1. Motion Deblurring. We created two sets of blurred images using the disk and line kernels from MATLAB. The *fspecial* functions were used to generate these Kernels and the blurred images were generated using the MATLAB funtion *imfilter* along with these kernels over images from the BSDS500 Dataset (7). The radius for the disc kernel was set to 10, and the length for the line kernel was set to 25. Few examples from the generated dataset are given in Fig.3. In addition to this, we also included some data augmentation steps in which every image was rotated 90° and flipped horizontally before they were reused for training.

For the network design, we changed certain connections before giving the summation of the iteration output and the original input to the following iteration. The structure can be seen in Table 2.

Here, we trained our ODP network for 4500 epochs using BSDS500 dataset with data augmentation. The training options were: batchsize = 50, initial learning rate = 0.001, Adam $\beta_1 = 0.9$, $\beta_2 = 0.95$ and weight decay = 0.0001. For initialization of the parameters we used the values given in the supplement paper of our main reference paper(1): $\alpha_k = C_0 C_k$ with C_0 and C_k learnable, initialized to $C_0 = 1000$ and $C_k = 2k$.

C.2. Gaussian Deblurring. We first created our dataset by applying blur on the BSDS500 image set (7). We used the MATLAB function *fspecial* with parameter sigma set to 3.5 to generate the blur kernel. Our network structure was also changed slightly for this task; rather than using 4 iterations we used 8 iterations for motion deblurring. We also changed the structure of our CNN.

Here, we trained our ODP network for 3000 epochs using BSDS500 dataset with data augmentation. The training options were: batchsize = 50, initial learning rate = 0.001, Adam $\beta_1 = 0.9$, $\beta_2 = 0.95$ and weight decay = 0.0001. For initialization of the parameters we used the values given in the supple-

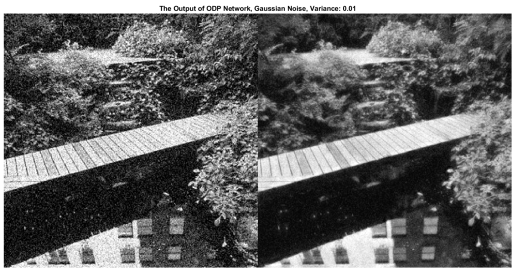

Fig. 7. Example 1 from Gaussian denoising

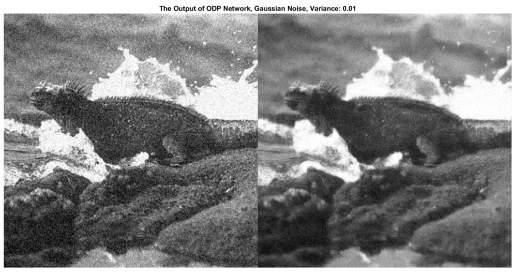

Fig. 8. Example 2 from Gaussian denoising

ment paper of our main paper (1): $\alpha_k = C_0 C_k$ with C_0 and C_k learnable, initialized to $C_0 = 1000$ and $C_k = 2^{-k}$. (2).

4. Experimental Results

A. Denoising Results. In the following (Figures 7,8,9,10,11,12) our results for the denoising case can be seen. Our PSNR values 13 turned out to be in line with the results obtained in our reference paper, from which we deduce that our results can be regarded as compatible with the reference paper. Thus, based on our implementation we observe that the proposed framework in the paper (1) is more robust for the denoising task when compared to deblurring.

B. Deblurring Results. Here, the most dominant artifacts we encounter are the ringing effects. We

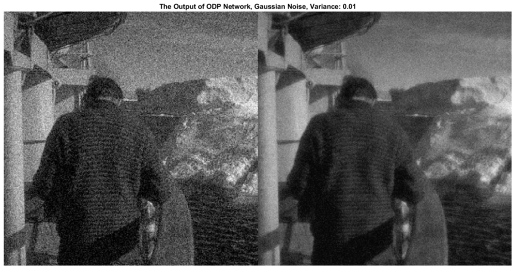

Fig. 9. Example 3 from Gaussian denoising

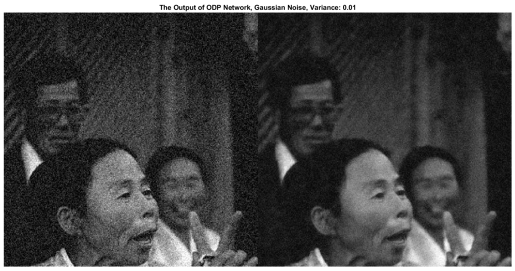

Fig. 10. Example 4 from Gaussian denoising

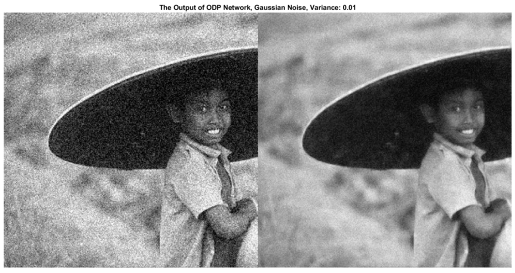

Fig. 11. Example 5 from Gaussian denoising

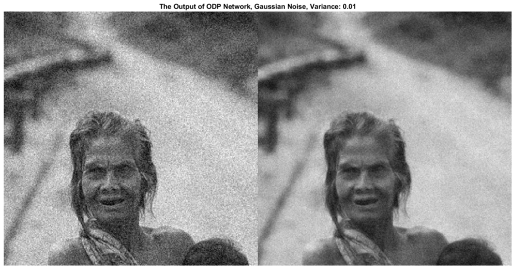

Fig. 12. Example 6 from Gaussian denoising

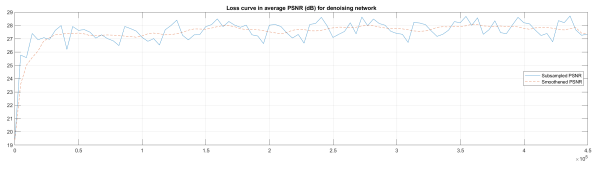

Fig. 13. PSNR values for denoising (dB)

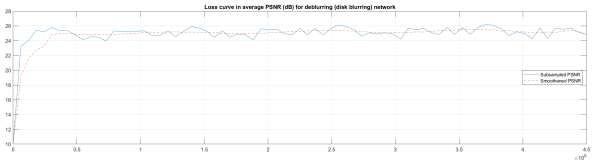

Fig. 14. PSNR values for deblurring from Disk Motion Blurring (dB)

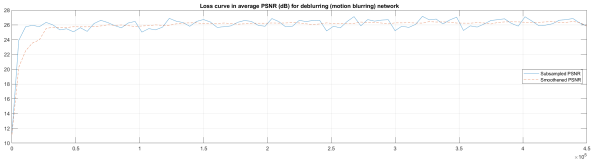

Fig. 15. PSNR values for deblurring from Linear Motion Blurring (dB)

believe that this arises from the frequency tolerance of our filter (Weiner filter) in the fourier domain. The filter captures the frequency of the edges and assumes that the images have a natural periodicity at this frequency in the fourier domain. Due to the statistical features of natural images, it is expected to have a sudden change in the frequency domain. This discontinuity causes the effect called boundary related ringing (See Fig.24).

We solved this issue by tapering the edges, further details are mentioned in the novelty section. All results presented ahead in the deblurring case are processed accordingly.

5. Novelty

As stated in training approach part, we initially trained three different network structures for three different colors then combined the all channels to reconstruct the RGB denoised version. Consequently, training process took three times longer, as expected. As seen in Fig. 22, red and the green channel denoising process are performed better whereas denoising in blue channel is not as good as the other channels. We wanted to bring a novel approach to solve this problem: We trained blue channel 500 epochs more while keepeing the other parameters unchanged and we obtained the result in Fig. 23 with a PSNR of 28.3, which is the best result we

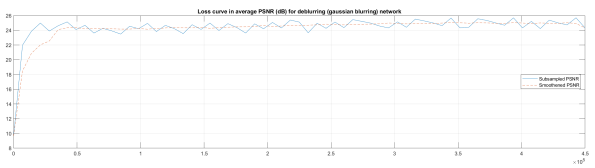

Fig. 16. PSNR values for deblurring from Gaussian Motion Blurring (dB)

Fig. 17. Deblurring Artifacts - without taper

Fig. 18. Deblurring Artifacts - without taper

Fig. 19. Motion Deblur Results - taper added as novelty

Fig. 20. Disk Deblur Results - taper added as novelty

Fig. 21. Gaussian Deblur Results - taper added as novelty

Fig. 22. Separation of the rightmost below image to its channels where blue is noisy

Fig. 23. Output obtained when blue channel is trained more

obtained so far in the 3-channel case.

For the deblurring case, the Discrete Fourier Transform (DFT) that is used in the network assumes that the images have a periodicity in the frequency domain. In order to eliminate the resulting ringing effects, the images were tapered down towards the edges. This was achieved using a built-in MATLAB function called *edgetaper*. Also, we note that the techniques of edge-detection with Bilateral filtering discussed in the lectures could also be applied to solve this problem.

6. Conclusions

In this project we have implemented the results of an existing paper. This process was instructive for us since we realized that a complete project consists of many components which should be planned and divided among team members carefully. Moreover, the paper (1) offers many insights to the inverse problem often tackled in Computational Imaging. We

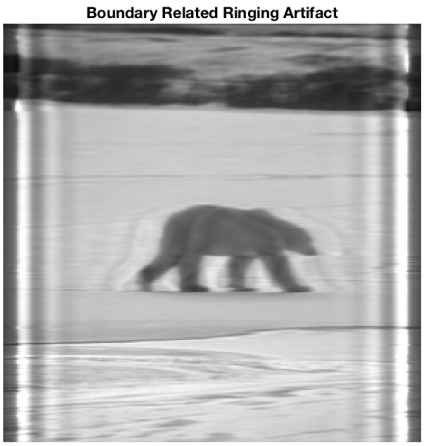

Fig. 24. Ringing artifact

learnt of various approaches to solving such problems specific to certain tasks in Imaging and the nuances in the various approaches with respect to the task at hand. We experienced first hand how extensions to theoretical techniques from classes, such as Weiner deconvolution, are implemented in code.

Furthermore, through this project we got some exposure to state-of-art techniques being developed in Machine Learning and CNNs in particular. It was a very enriching experience implementing such advanced techniques to solving common problems in Computational Imaging. The concepts of Residual Networks and melding classical approaches within the Deep Learning paradigm offer great insights into the functioning of a Neural Network.

7. Work Division

Shashank Balla: trained the network, implemented debluring part, report writing, preparing the figures

Ege Cetintas: Implemented the denoising network, trained it, handled the ringing artifacts in motion deblur, report writing, preparing the figures.

Berkin Durmus: Implemented the denoising network, trained it, handled the ringing artifacts in motion deblur, report writing, preparing the figures.

Dilek Mirgun Yalcinkaya: Implemented the debluring network, trained it, report writing, preparing the results.

8. References

1. Diamond S, Sitzmann V, Heide F, Wetzstein G (2017) Unrolled optimization with deep priors. *ArXiv* abs/1705.08041.
2. Cheng JY, Chen F, Alley MT, Pauly JM, Vasanawala SS (2018) Highly scalable image reconstruction using deep neural networks with bandpass filtering. *CoRR* abs/1805.03300.
3. Xu L, Ren JSJ, Liu C, Jia J (2014) Deep convolutional neural network for image deconvolution in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'14. (MIT Press, Cambridge, MA, USA), pp. 1790–1798.
4. Gregor K, Lecun Y (year?) Learning fast approximations of sparse coding.
5. Wang S, Fidler S, Urtasun R (2016) Proximal deep structured models in *Advances in Neural Information Processing Systems 29*, eds. Lee DD, Sugiyama M, Luxburg UV, Guyon I, Garnett R. (Curran Associates, Inc.), pp. 865–873.
6. Jaroensri R, Biscarrat C, Aittala M, Durand F (2019) Generating training data for denoising real RGB images via camera pipeline simulation. *CoRR* abs/1904.08825.
7. Arbelaez P, Maire M, Fowlkes C, Malik J (2011) Contour detection and hierarchical image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.* 33(5):898–916.