Introduction to machine learning methods for image deblurring

Jonathan King, ECE 901

1. Introduction

Image deblurring is a prominent and growing area of computational optics that can largely be boiled down to the solving the following equation:

$$I_B = k * I_s + N \qquad (1)$$

Where I_B is a blurred image, k is a blur kernel, I_s is the "ground truth" image, N is a noise term most often treated as additive which may or may not include saturation and compression artifacts. Here, the goal is to approximate I_s from I_B. The area of image deblurring can be further split into two areas: (1) "blind" deblurring where k is unknown and (2) "non-blind" deblurring where k is known. For "non-blind" image deblurring, a close approximation of the ground truth image can be recovered from a blurry image if the blur kernel is known, according to Eq. 2:

$$\mathbf{w}_{LS2} = \underset{\mathbf{v}\in R^n}{\operatorname{argmin}} \|\boldsymbol{I_B}\mathbf{v} - \boldsymbol{I_s}\|_2^2 + \lambda\|\mathbf{Bv}\|_2^2 \qquad (2)$$

Where $\mathbf{B}$ is an identity matrix, λ is a tuning parameter, and $\mathbf{w}_{LS2}$ are the L2-regularized weights that are used to return the recovered image $\boldsymbol{I_R}$ according to $\boldsymbol{I_R} = \boldsymbol{I_B}\mathbf{w}_{LS2}$, where $\boldsymbol{I}$'s are row vectors and $\mathbf{w}_{LS2}$ is a column vector. The solution to Eq. 2. is conveniently found through matrix inversion:

$$\mathbf{w}_{LS2} = \left(\boldsymbol{I_B}^{\mathrm{T}}\boldsymbol{I_B} + \lambda\mathbf{B}^{\mathrm{T}}\mathbf{B}\right)^{-1}\boldsymbol{I_B}^{\mathrm{T}}\boldsymbol{I_s} \qquad (3)$$

In our second homework set, we followed this method to recover a blurred image (Fig. 1)

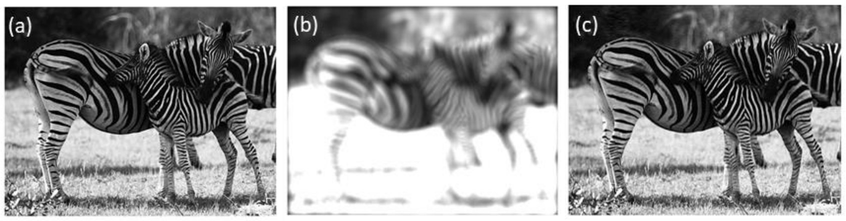

Fig. 1. L2-loss matrix inversion operation to recover a blurred image. (a) Original, "ground truth" image (b) blurred image generated by convolving the ground truth image with a disk blur kernel. (c) Image recovered using L2-loss matrix inversion.

However, in many cases, a close approximation of the blur kernel may not be known beforehand in which case blind image deblurring methods must be used. For the purposes of both non-blind and blind image deblurring, Machine learning (ML) has enabled an expanding zoo of powerful and diverse deblurring techniques. For the purposes of this project, I sought to gain holistic insight into the process of blind ML image deblurring. To do this, I set up an idealized scenario so that I could build multiple ML blind image deblurring codes "from scratch" with minimal complexity using a low-resolution data set to avoid prohibitive computational resource requirements. Throughout, I implemented individual-pixel deblurring, which, while unsuited for higher resolution applications, was straightforward to code and well-suited for my low-resolution data set. Overall, I developed three different machine learning algorithms: (1) L1-regularized least-squares loss ('LASSO') pixel regression, (2) L2-regularized least-squares pixel regression, and (3) 3-layer neural network. Each algorithm's performance was evaluated according to time-to-run and set-averaged mean squared pixel error from ground truth (MSPE), given in Eq. 4.

$$MSPE = \frac{1}{M}\sum_{j=1}^{M=1000}\frac{1}{N}\sum_{i=1}^{N=28^2}\left(I_{s_{ij}} - I_{R_{ij}}\right)^2 \qquad (4)$$

Where M is the number of images tested (1000), N is the number of pixels per image (784), $I_{s_{ij}}$ is ground truth pixel i in image j and $I_{R_{ij}}$ is recovered pixel i in image j. The mechanics and performance of my methods gave a baseline for insight into the sorts of techniques used by more-sophisticated models to improve accuracy and speed.

2. Datasets used

The dataset used in this project is the infamous MNIST dataset [1]. The MNIST dataset consists of 70,000 28x28 pixel images of handwritten integer numbers from 0 to 9. For this project, I only used the first 4,000 images of the 70,000 set to accommodate computational expediency. The first 1,000 were used as my testing images only to be used at the end, the next 2,000 were used for training and the next 1,000 were used as a validation set to tune models prior to testing. Blurred images were generated according to equation 1 where k is a 5x5 gaussian blur kernel (see Fig. 2), I_s is the original, unblurred, 28 x 28 image, and N = 0.

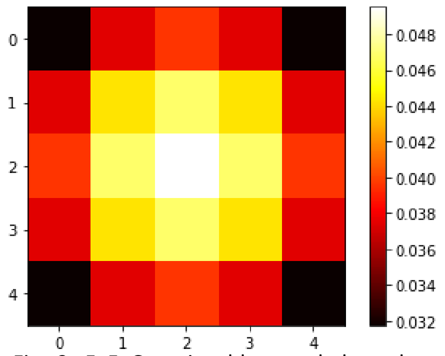

Fig. 2. 5x5 Gaussian blur used throughout project.

In the real world, the acquisition of "ground truth" images can be difficult (since an "ideal" image does not exist) and realistic blur kernals can be difficult to synthesize since real blur kernals can vary from image to image and can also vary within the same image [2][3]. A common approach to generating blurred images from real images is to use frame averaging with a video camera (e.g. for a 3-frame sequence, the blurred image would be an average of frames 1, 2, and 3 while the ground truth would be frame 2) [4]–[6]. For this project, the modeling process can still be simulated using ground truth images that are treated as ideal and blurred images that are generated by applying simple and globally consistent gaussian blur kernels to ground truth.

3. Methods and Results

3.1 Method 1: Non-blind deblurring via L2-regularized least squares loss matrix inversion

The first method employed was the method used in the homework to serve as a reference for the subsequent ML algorithms. Tuning parameter λ was varied across 1E-15<λ<1E1 to find minimum MSPE. Results are shown in Fig. 3. alongside an example of an original, blurred, and recovered digit. Runtime was 0.3 seconds.

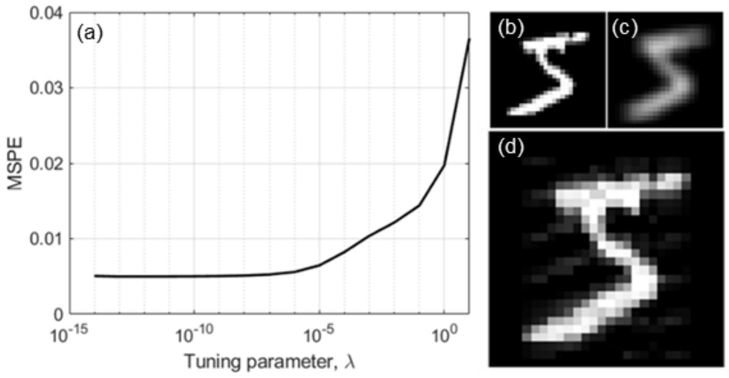

Fig. 3. (a) MSPE vs tuning parameter of non-blind L2-loss inversion deblurring method. **MSPE = 5E-3**. (b) Example of ground truth image. (c) Example of blurred image (d) recovered image.

3.2 Method 2: LASSO pixel regression

Both regression methods performed independent regressions for each single pixel within the 784-pixel space using the training images. First, all images were converted from 2-D 28x28 images into 1-D 784-element vectors with each element containing a pixel value from 0 to 1 corresponding to pixel brightness. A schematic for the problem setup is provided in Fig. 4. Each pixel *m* of the recovered image $\boldsymbol{I}_R$, was the dot product of a 784-element weight vector $\boldsymbol{w}_m$ and the blurry image pixel vector $\boldsymbol{I}_B$. During the solution process, $\boldsymbol{w}_m$ is adjusted for I_{R_m} to consistently and effectively approximate I_{s_m} across all training images.

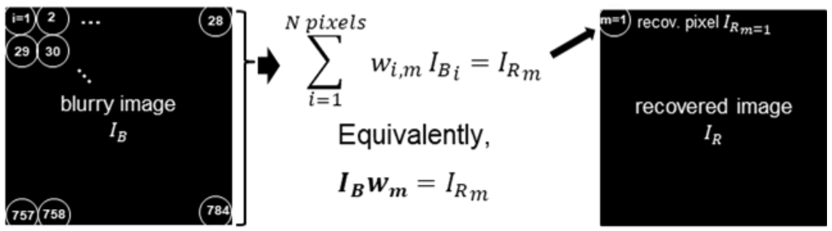

Fig. 4. Matrix operations and pixel weight vectors for recovering image pixel values.

Weight vector $\boldsymbol{w}_m$ was modeled using the training data and the L1-regularized least squares loss function (LASSO) for all training images (M = 2000 images):

$$\boldsymbol{w}_m = \underset{\mathbf{v}\in R^n}{\operatorname{argmin}} \sum_{j=1}^{M=2000} \left\| \boldsymbol{I}_{\boldsymbol{B}\,j}\mathbf{v} - \boldsymbol{I}_{\boldsymbol{s}\,j} \right\|_2^2 + \lambda \|\mathbf{Bv}\|_1 \quad (5)$$

$\boldsymbol{w}_m$ was solved using stochastic gradient descent where maximum iterations were set to 1E4 with a tolerance of 1E-3. Once the $\boldsymbol{w}_m$ vectors for all pixels (m=1…784) were solved, all 784 $\boldsymbol{w}_m$ column vectors were horizontally stacked into a 2-D array weighting array $\boldsymbol{W}$, where $W_{ij} \equiv$ weight of blur pixel i to recovered pixel j. This 784x784 matrix could then be directly applied to any blurry image row vector or vertical stack of blurry image row vectors to produce a predicted estimate of the ground truth via $\boldsymbol{I_R} = \boldsymbol{I_B W}$. The test MSPE could then be calculated accordingly:

$$MSPE = \frac{1}{M} \sum_{j=1}^{M=1000} \left\| \boldsymbol{I}_{\boldsymbol{B}\,j}\mathbf{W} - \boldsymbol{I}_{\boldsymbol{s}\,j} \right\|_2^2 \quad (6)$$

The validation error across tuning parameter λ for this method is provided in Fig. 5 along with an example of a ground truth, blurred, and recovered test image. Based on the validation error curve, the tuning parameter was selected to be 1E-3 and <u>a test MSPE of 2.38E-3 was acquired. Runtime to train per λ was 230.6 seconds.</u>

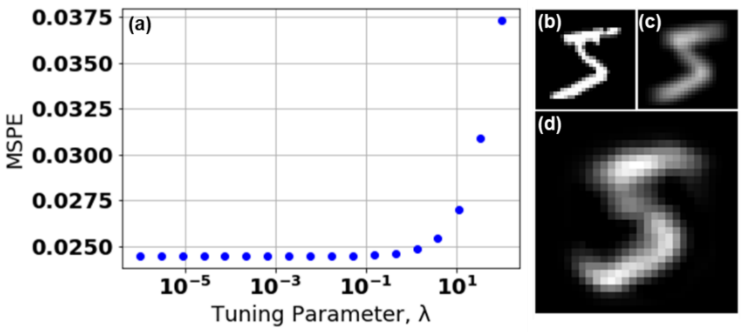

Fig. 5. (a) Validation MSPE vs tuning parameter of blind L1-loss LASSO pixel regression deblurring method. Testing **MSPE = 2.38E-2**. (b) Example of ground truth image. (c) Example of blurred image (d) recovered image.

3.3 Method 3: L2-loss pixel regression

Method 3 followed the same progression as Method 2, with the difference being the loss function and the process by which $\boldsymbol{w}_m$ was solved. This time, weight vector $\boldsymbol{w}_m$ was modeled using the L2-regularized least squares loss function for all training images (M = 2000 images):

$$\boldsymbol{w}_m = \underset{\mathbf{v}\in R^n}{\operatorname{argmin}} \sum_{j=1}^{M=2000} \left\| \boldsymbol{I}_{\boldsymbol{B}\,j}\mathbf{v} - \boldsymbol{I}_{\boldsymbol{s}\,j} \right\|_2^2 + \lambda \|\mathbf{Bv}\|_2^2 \quad (7)$$

Similar to Method 1, this particular matrix equation can be directly solved via inversion (Eq. 3). The validation error across tuning parameter λ for this method is provided in Fig. 6. along with an example of a ground truth, blurred, and recovered test image. Based on the validation error curve, the tuning parameter was selected to be 1E-6 and a <u>test MSPE of 1.5E-4 was acquired. Runtime to train per λ was 125.8 seconds.</u>

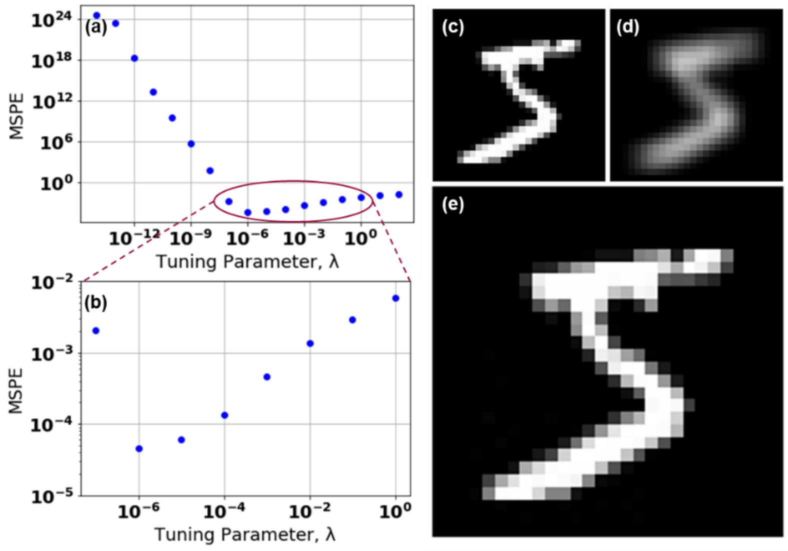

Fig. 6. (a) Validation MSPE vs tuning parameter of blind L2-regularized least squares pixel regression method. When applied to testing data, testing **MSPE = 1.5E-4**. (b) Zoomed-in selection of low-MSPE region. (c) Example ground truth image. (d) Example blurred image (e) recovered image

3.4 Method 4: 3-layer neural network

The final method was a convolution neural network built using Tenseorflow. The design degrees of freedom for a neural network are boundless and rather than attempt to rigorously optimize a model, I just wanted to build a functional one. While I did adjust model parameters such as number of hidden layers, number of nodes, optimizers, and activation functions, etc., I simply present the most-effective model.

A neural network with many layers (a.k.a. deep learning) allows for abstract feature selection which can compensate for irregular and non-uniform blurs. However, the blur that I use in this project is relatively straightforward and so I anticipated that a single hidden layer could work well. However, while the blur kernel is uniform across the image, the pixels that interact with the blur kernel to yield a given blurry pixel will be specific and localized. To me, this implied that weight mapping would be fairly disparate between different pixels and that the most intuitive feature selection would be the individual pixels themselves. So, to accommodate the anticipated uniqueness for pixel mapping, I added 784 nodes in my hidden layer. Based on common literature activation functions and trial and error, I used a relu activation function. My model was trained using an Adam optimizer over 200 epochs according to a mean squared error (MSE) loss function. A schematic of my model is provided in Fig. 7. along with an example of a blurry-and-recovered image pair from the testing data set. <u>The testing MSPE for the neural network was 5.82E-3. Runtime to train per λ was 55.9 seconds.</u>

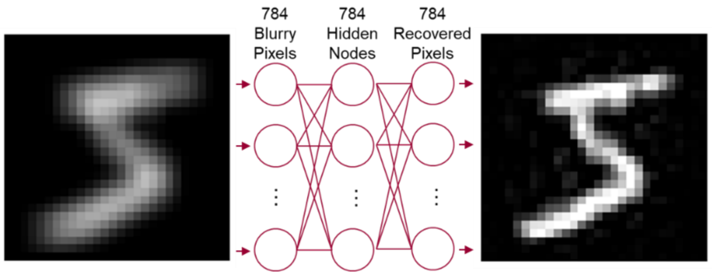

Fig. 7. Neural network architecture and example of blurry input test image and output recovered image. Model was trained using an Adam optimizer over 200 epochs according to an MSE loss function. **Testing MSPE = 5.82E-3**

4. Discussion and Conclusion

The L2-regularized regression approach significantly outperformed the other methods showcased in this project. Unlike the L2-regularization scheme, the L1-regulization scheme cannot be solved for directly (matrix inversion for the L2 scheme) and must utilize an iterative optimization approach like stochastic gradient descent to converge to a solution. For much larger data sets, the L1 scheme may be preferable to the L2 scheme since it would require less CPU memory, but here, L2 is much quicker. Furthermore, L1 regularization encourages sparse solutions and so the pixels that are poorly represented in the data set (e.g. pixels that are almost always dark in the ground truth training images) may lack proper weighting. The bottom left corner of ground truth images would be a region where one would expect very few instances of non-zero-pixel values in the training data. This lack of representation, or insufficient convergence criteria, may have resulted in the misrepresentative tail of the ‘5’ in the resultant recovered image in Fig. 5d. L2 regularization, on the other hand, is more inclusive and quicker to converge.

The shortcomings of my methods relative to published literature are readily apparent. First, while my methods were relatively straightforward to code, they were computationally inefficient. The weights in my modeling schemes effectively served as “reverse” blur kernels, or 784 different 28x28 kernels (one for each pixel), that, when applied to the blurred image, would return a recovered image to approximate ground truth. To repeat my algorithms for high definition images (1024x720 pixels), the process would take nearly 1E6 times more computations to complete, and runtime may occur on the scale of years.

However, the actual applied blur kernel was consistent across the image and only 5x5 pixels – a much smaller solution space than 784x28x28. A more computationally efficient scheme would be to flip the roles of the blurred and ground truth images and *then* perform regression to approximate the blur kernel rather than the “reverse”. However, while computationally efficient, coding would be much more complex since the images would need to be subdivided into smaller “patches” where patch size is proportional to maximum estimated size of the blur kernel. With the new blur kernel trained using an ML algorithm, a common deconvolution algorithm could then be used with the blurred image and the trained blur to produce recovered test images. This alternative approach, which relies on “patching”, is quite common in more-sophisticated models (see Gupta et al. for more detail [7]). I expect that it would yield orders-of-magnitude improvement in speed, but also decrease accuracy since errors would compound between the ML approximation step and the “non-blind” deconvolution step that applies the learned blur kernel.

In the case where blur can vary significantly between images, my methods would likely be ineffective due to their limited capacity for abstraction and for their computational inefficiency. Here again, the patchwork framework could be very useful. First, the vast improvements in computational efficiency would accommodate deeper and more-abstract models. As demonstrated in Svoboda et al., a model capable of deep abstractions can vastly outperform simple regression for real blurs which may be variant and complex [8]. Second, adjacent patches can be related to each other to further constrain the problem while still accommodating for irregularity across the image [2]. Another common technique involves using image statistics to constrain the solution for the blur kernel and noise [9][10]. While I did not employ these more-sophisticated techniques, these adjustments, and many others, would vastly improve overall performance.

5. References

[1] “MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges.” [Online]. Available: http://yann.lecun.com/exdb/mnist/. [Accessed: 22-Oct-2020].

[2] J. Sun, W. Cao, Z. Xu, and J. Ponce, “Learning a Convolutional Neural Network for Non-uniform Motion Blur Removal,” *IEEE Conf. Comput. Vis. Pattern Recognit.*, pp. 769–777, 2015.

[3] W. Z. Shao *et al.*, “DeblurGAN+: Revisiting blind motion deblurring using conditional adversarial networks,” *Signal Processing*, vol. 168, pp. 8183–8192, 2020.

[4] T. H. Kim, S. Nah, and K. M. Lee, “Dynamic Scene Deblurring using a Locally Adaptive Linear Blur Model,” pp. 1–14, 2016.

[5] M. Noroozi, P. Chandramouli, and P. Favaro, “Motion deblurring in the wild,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 10496 LNCS, pp. 65–77, 2017.

[6] A. Agrawal and R. Raskar, “Optimal single image capture for motion deblurring,” *2009 IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. Work. CVPR Work. 2009*, vol. 2009 IEEE, pp. 2560–2567, 2009.

[7] A. Gupta, N. Joshi, C. Lawrence Zitnick, M. Cohen, and B. Curless, “Single image deblurring using motion density functions,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 6311 LNCS, no. PART 1, pp. 171–184, 2010.

[8] P. Svoboda, M. Hradis, L. Marsik, and P. Zemcik, “CNN for license plate motion deblurring,” *Proc. - Int. Conf. Image Process. ICIP*, vol. 2016-Augus, pp. 3832–3836, 2016.

[9] M. Hradiš, J. Kotera, P. Zemčík, and F. Šroubek, “Convolutional Neural Networks for Direct Text Deblurring,” no. 1, pp. 6.1-6.13, 2015.

[10] A. Levin, “Blind Motion Deblurring Using Image Statistics.”

The code used in this project is partially based on code used for a classification project for ECE 532 (primarily MNIST data retrieval, vector initialization, and optimization loops). The purpose, implementation, and results of the two code sets are substantially different. Respective Githubs can be found here: ECE 901 - https://github.com/jlking2/Image_deblurring, ECE 532 - https://github.com/jlking2/Neural_net_numbers