Non-Blind Image Deblurring using Neural Networks

Andy Gilbert Shai Messingher Paul Westhoff

Abstract

Each year, people take over one trillion photographs. Most of those are taken on smartphones, which lend themselves to motion blur. In this project, we investigated whether a learning-based approach to image deblurring could enhance conventional deblurring techniques. We hypothesized that a non-linear combination of conventional reconstructions of a blurry image yield a sharper image. Our results show that the hypothesis was true. The proposed system outperformed conventional deblurring results, albeit at a higher computational cost.

1. Introduction

Single-image non-blind image deconvolution attempts to recover a sharp image from a blurred image and a blur kernel. Assuming that the camera motion was spatially invariant, this problem can be formulated as

$$y = k * x + n$$

where y is the blurred image, x is the sharp image, k is the blur kernel, and n is additive noise. Stated in terms of these values, our goal is to recover x from y and k. However, as n is unknown, this is an ill-posed problem.

Conventional non-blind deconvolution methods include the use of algorithms such as Wiener filtering and Richardson-Lucy. However, both of these methods suffer from ringing artifacts and are less effective at handling large motion outliers. Several methods attempt to find good priors to use in image restoration. These include Hyper-Laplacian Priors [1] and non-local means [2]. However, these methods require expensive computation costs to obtain top-quality sharp images.

More recently, neural networks have been utilized for image restoration. Though these methods work well, they are unpractical because they generally require re-training for each possible input kernel.[3]

In this project, we attempt to enhance conventional deblurring algorithms. We propose a system that takes in a blurry picture, forms 15 deblurred versions of it using Wiener filtering with different SNR assumptions, stacks them into a tensor, puts them through a deep neural network to non-linearly combine them, and outputs a new reconstructed sharp version.

1.1. Related Work

Recently, deep learning has been proposed as a solution to low-level image processing problems. These include denoising [4], super-resolution [5], and edge-preserving filtering [6]. Still more recently, researchers have attempted non-blind image deblurring. The two most relevant algorithms are discussed here.

Schuler et al. develop a multi-layer perceptron approach to remove artifacts caused by the deconvolution process.[3] Xu et al. go a step further and attempt to use a deep convolutional neural network (CNN) to restore images corrupted by outliers. They also use singular value decomposition (SVD) to reduce the number of parameters in the network. This approach cannot be generalized, however, and is limited by the fact that it needs to re-train the network for every kernel.

1.2. Dataset and Pre-Processing

Images from ImageNet were used to form the dataset. Specifically, the subset of images of drinks were used. The subset amounts to 1153 pictures. Each image was randomly cropped into a 256x256 sub-image. For each cropped image, a trajectory of motion was generated. The trajectory was estimated by modeling a particle's motion affected by inertial, impulsive and Gaussian perturbations. This model can represent a wide spectrum of motions, ranging from simple translations, to sudden movements that occur when camera users try to compensate the camera shake, to even abrupt motions that occur when users press the shutter button [1]. The trajectory of motion is then sampled into four different point-spread functions (PSFs), each one larger than the last. See Fig. 1 to see four PSFs generated from the same trajectory.

The PSFs were used as blur kernels. Each image from the dataset was convolved with its corresponding group of four blur kernels, resulting in 4612 blurry images. Gaussian and Poisson noise was subsequently added to them.

The train-test-validation split for this project was 80-10-10, meaning there were 3690 images in the training set, and 461 images in both the test and validation sets.

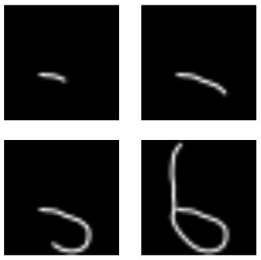

Figure 1. Generated PSFs from the same trajectory curve

For each blurry image from the training set, 15 reconstructions were made with Wiener filtering. Each reconstruction is computed using an assumed SNR value. The SNRs employed were the values in the range from 9 dB to 65 dB, in steps of 4dB. These partially deblurred images are then stacked into a (256, 256, 45) matrix, with the last dimension being 45 because the images are RGB. This stack serves as the input (features) to the next stage of the proposed system, the deep neural network.

The dataset and pre-processing pipeline is depicted in Figure 2

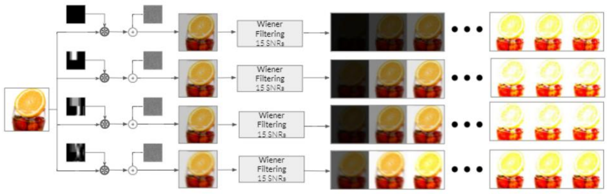

Figure 2. Data generation and pre-processing pipeline

2. Methodology

2.1. Architecture

We now have a feature set that is 256x256x45. We also found that stacking the original blurred image as an additional input feature helped train the network. This extra feature helped provide some of the original color information to the network, which helped it match the output image better. The final input dimension was then 256x256x48.

After the input layer the network had a variable number of ResBlock layers. Each Resblock layer had a 2D convolutional layer followed by a ReLu layer followed by another 2D convolution. A skip connection from the input was added to allow for building deeper networks. The number of ResBlocks was varied as a hyper-parameter, and will be discussed in the Training Details section. Each of the convolution layers had 64 filters and used a padding to maintain constant resolution (256x256). After the ResBlocks there is one final convolution that brings the channel size back to 3 to build an RGB output of the deblurred image. During the course of training many of the output images had sharper edges but had large areas that should have been colored more uniformly, and had Gaussian noise. To reduce this artifact, a bilateral filtering layer was added as the last layer of the network. This architecture is shown in Fig. 8 and was originally based off of the work done in [12].

A simpler system that consisted of one Wiener Filter was used as a metric for comparison. To enforce a fair comparison, the SNR parameter of the Wiener Filter was set based off of the average SNR of all images in our dataset. The SNR was not tuned individually for each image since part of the contribution made in this work is eliminating the need to tune this for each image. A bilateral filter was added to the output of the Wiener Filter with the same parameters as the one used in our network. This architecture is shown in Fig. 4.

2.2. Loss and Accuracy metrics

We originally used a L2 loss function shown in Equation 1 where n is the batch size, c is channels of image (3), and w & h are width and height of the image (256). y is the reference sharp image while $\hat{y}$ is the output of the network described above.

$$L_{L2} = \frac{1}{n * c * w * h} * \sum_{i=0}^{n} ||y - \hat{y}||_2^2 \tag{1}$$

This loss function worked decently and resulted in adequately deblurred images. However, one problem with using the L2 loss as a metric for images is that it does not correspond to perceptual differences as seen by the human visual system (HVS). Other loss functions for image loss in iterative algorithms have been previously analyzed by other groups and the SSIM loss has been experimentally found to be a better metric for perceptual difference following the HVS [10]. The SSIM Loss is defined in Equation 2 where P is a number of patches making up each image and p is the center pixel of each patch.

$$L_{SSIM} = \frac{1}{n * c * w * h} * \sum_{i=0}^{n} \sum_{p \in P} -SSIM(p) \tag{2}$$

Where:

$$SSIM(p) = \frac{2\mu_x \mu_{\hat{y}} + c_1}{\mu_x^2 + \mu_{\hat{y}^2 + c_1}} \times \frac{2\sigma_{x\hat{y}} + c_2}{\sigma_x^2 + \sigma_{\hat{y}}^2 + c_2} \tag{3}$$

$\mu_{\hat{y}}$ = average of first image window

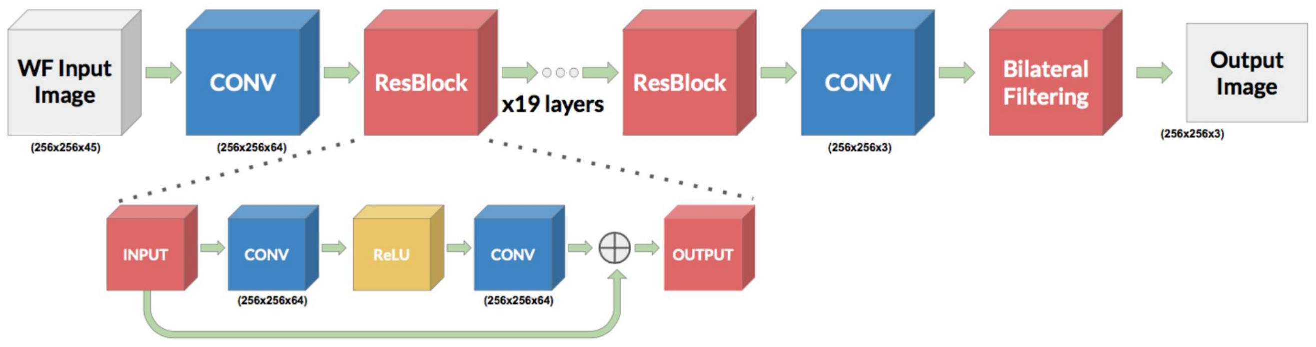

Figure 3. Architecture of our network. The number of Res Blocks was varied during the course of training from 19 to 12. The network is shown here with bilateral filtering added as the last layer.

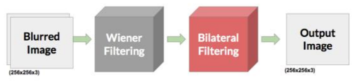

Figure 4. Architecture of the network we were comparing to. The network is shown here with bilateral filtering added as the last layer.

μ_y = average of second image window
$\sigma_{\hat{y}}$ = standard deviation of first image window
σ_y = standard deviation of second image window
$c_1 = (k_1 R)^2$
$c_2 = (k_2 R)^2$
R = dynamic range of pixels within the window

k_1 and k_2 are constants controlling the stability of the division and are set to 0.01 and 0.03 respectively. Although the loss is learned on these center pixels, the error is still back-propagated to each pixel within the support region (window) that contributes to the calculation of Equation 3 because those pixels are used in the mean and standard deviation results. This loss function is also well suited to this problem because it is differentiable with derivatives described in [10]. SSIM loss was implemented using the package pytorch_ssim [11]. Using SSIM loss actually reduced the Gaussian noise and thus the need for a bilateral filter on the output.

Peak signal noise ration (PSNR) was used as a measure of the accuracy of the output of the metric. However PSNR also does not directly correspond to the perceptual difference as seen by the HVS. It was a good metric to use as training progressed to see the status of training, but proved to be sub-optimal.

Using the SSIM loss perceptually improved the resulting sharp images.

2.3. Training Details

We used an adaptive learning process where, whenever possible, we would reuse the weights we had previously trained rather than starting over. When the loss stagnated for several epochs, a different hyperparameter combination would be tried and evaluated for several epochs to see how the loss reacted. This made it difficult to directly evaluate some of the tuning steps we used. Several optimizations, such as changing filter size and including batch normalization, could not rely on the pretrained network since the required parameters changed.

The hyperparameters varied are described below:

- **Learning rate**: This hyperparameter was varied frequently through the course of training in a range from $1e-5$ to $1e-2$. It was sometimes reduced after many epochs of training as well. The optimum learning rate was usually between $5e-4$ and $1e-3$.
- **Filter size:**: The filter size of the convolution layers was varied between 3 and 5. A filter size of 5 was found to be slightly better but did increase the number of parameters and thus training time. The final value was set at 5.
- **Dropout rate**: The network used dropout as a method for regularization. The dropout rate was varied form .8 to .9 and .8 was found to be optimum.
- **Number of channels**: This was kept constant at 64 following the work done in [12].
- **Number of Res Blocks**: The number of Res Blocks was varied from 19 to 12. There was not an appreciable difference between the results but it did help decrease

runtime since batch size could be increased. The final value was set at 12.

- **Batch Normalization**: Batch normalization was added between convolution layers in the Res Block but it did not help the results and increased the required parameters, slowing training, and so was removed.

- **Input Normalization**: The stacks of input images from Wiener Filtering also exhibited large color variations across the stack, so the input was normalized across the stack. However, this actually hurt the network and so was removed.

- **Bilateral filter params**: The bilateral filter uses window size = 7, $\sigma_{color} = 40$, and σ_{space} = 10.

- **Batch size**: The batch size was highly dependent on the other parameters. For the most part the batch size was set to be the maximum possible amount that would fit on the GPU to take advantage of vectorization. A batch size of 2 (following [12]) was tried, but this resulted in worse results and longer training times. Based off the other parameters above the final batch size was 32.

3. Results and Analysis

As previously mentioned, we compare to a baseline of using Wiener filtering with bilateral filtering. The results are summarized in Table 1

	Neural Net	Baseline
SSIM	0.62	0.51
Eval Time (sec)	0.347	0.023

Table 1. Summary of our results.

The ideal value for SSIM is 1, i.e. an image compared with itself has an SSIM of 1. We see that on average, our model performs better than the baseline on our test set. Our generated images have an SSIM of 0.62, a value 20% greater than the baseline. This is further substantiated by looking at the resulting images themselves. Figure 8 depicts a subset of images showing per image: a sharp, a blurred, a reconstructed version from our baseline, and a reconstructed version from our pipeline. As a specific example from this subset, the two reconstructions of the green Heineken bottle picture are shown in Figures 5 and 6. Our pipeline reconstructs high-frequency content like the letters and the star. The edges are also sharper, and the color is more realistic.

This improvement comes at the cost of runtime. Our algorithm runs 19 times slower than our baseline.

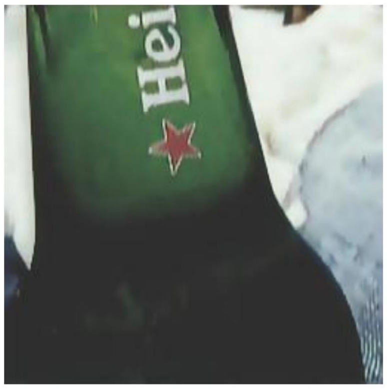

Figure 5. The output image from our network.

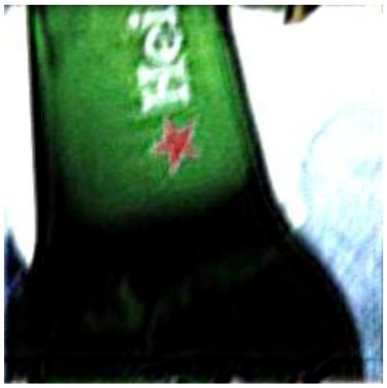

Figure 6. The output image from our baseline.

4. An Investigation Into Generative Adversarial Networks (GAN)

After gathering the results discussed above, we introduced a discriminator network to provide an adversarial loss to supplement the SSIM loss. This concept was introduced in [12] and we initially implemented the discriminator according to the architecture they described. This first GAN was unable to learn the difference between ground truth and generated images, and would output similar probability (around 0.5) for any image. We hypothesized that our network was too deep for the task and available data. By using a much shallower architecture shown in Figure 7 (similar to that proposed by [16] for a measure of sharpness), the discriminator could distinguish between generated and ground truth images with high accuracy.

Unfortunately, after training, we found this implementation to yield lower PSNR, lower SSIM, and images of lower quality perceptually in both sharpness and color shifting. We hypothesize that the GAN was learning to identify a color shift from the process of adding Gaussian and Poisson

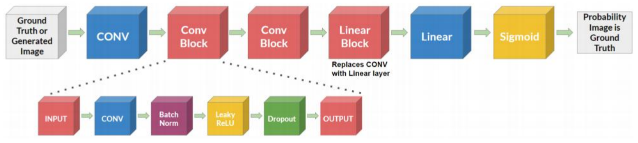
Ground Truth or Generated Image
CONV
Conv Block
Conv Block
Linear Block
Replaces CONV with Linear layer
Linear
Sigmoid
Probability Image is Ground Truth
INPUT
CONV
Batch Norm
Leaky ReLU
Dropout
OUTPUT

Figure 7. Shallow Discriminator Architecture.

noise, making the GAN too adept at classifying the images.

5. Conclusion and Future Work

We have shown that our model performs perceptually better than the baseline of Wiener filtering, but the runtime is an order of magnitude above the baseline.

An immediate next step for this project is the continued development of the GAN architecture. Weakening the discriminator by removing layers, increasing dropout, and adding normalization, might improve the adversarial loss term. Additionally, using a range of labels based on the magnitude of the original blur kernel during training might allow the degree of sharpness to be more easily learned by the discriminator. Finally, viewing the activations of each level could provide better insight to the information the discriminator is learning, and better allow us to determine what is causing the malfunction.

Finally, we want to further investigate whether our baseline is a good one to use. We hypothesize that using a reconstruction implemented with ADMM as our baseline, instead of a reconstruction using Wiener Filtering, would be more robust. We believe this because ADMM reconstructions maintain the color properties of the input image.

6. Acknowledgements

We would like to acknowledge the help of Vincent Sitzmann, who introduced to the idea of non-linearly combining simply filtered images to create a sharp reconstructed image. We would also like to acknowledge the starter code from CS230 [13] and the code we used for trajectory calculations [14] and ADMM with TV priors [15] on MATLAB.

7. References

[1] D. Krishnan and R. Fergus. Fast image deconvolution using hyper-laplacian priors. In NIPS, 2009.
[2] A. Buades, B. Coll, and J. Morel. A non-local algorithm for image denoising. In CVPR, pages 6065, 2005.
[3] C. J. Schuler, H. Christopher Burger, S. Harmeling, and B. Scholkopf. A machine learning approach for non-blind image deconvolution. In CVPR, 2013.
[4] H. C. Burger, C. J. Schuler, and S. Harmeling. Image de-noising: Can plain neural networks compete with bm3d? In CVPR, 2012
[5] C. Dong, C. C. Loy, K. He, and X. Tang. Learning a deep convolutional network for image super-resolution. In ECCV, 2014.
[6] Y. Li, J.-B. Huang, N. Ahuja, and M.-H. Yang. Deep joint image filtering. In ECCV, 2016.
[7] L. Xu, J. S. Ren, C. Liu, and J. Jia. Deep convolutional neural network for image deconvolution. In NIPS, 2014.
[8] http://home.deib.polimi.it/boracchi/Projects/PSFGeneration.html
[9] Z. Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, ”Image quality assessment: From error visibility to structural similarity,” IEEE Transactions on Image Processing, vol. 13, no. 4, pp. 600-612, Apr. 2004.
[10] Zhao, H, Gallo O, Frosio I, and Kautz J. ”Loss Functions for Neural Networks for Image Processing,” arXiv:1511.08861v2 [cs.CV] 14 Jun 2016.
[11] https://github.com/Po-Hsun-Su/pytorch-ssim
[12] Nah, Seungjun, Tae Hyun Kim, and Kyoung Mu Lee. ”Deep multi-scale convolutional neural network for dynamic scene deblurring.” arXiv preprint arXiv:1612.02177 3 (2016).
[13] https://github.com/cs230-stanford/cs230-code-examples
[14] http://home.deib.polimi.it/boracchi/Projects/PSFGeneration.html
[15] S. H. Chan, X. Wang and O. A. Elgendy, ”Plug-and-Play ADMM for image restoration: Fixed point convergence and applications,” IEEE Transactions on Computational Imaging, Nov. 2016. [16] Yu S, Wu S, Wang L, Jiang F, Xie Y, Li L (2017) A shallow convolutional neural network for blind image sharpness assessment. PLoS ONE 12(5): e0176632. https://doi.org/10.1371/journal.pone.0176632

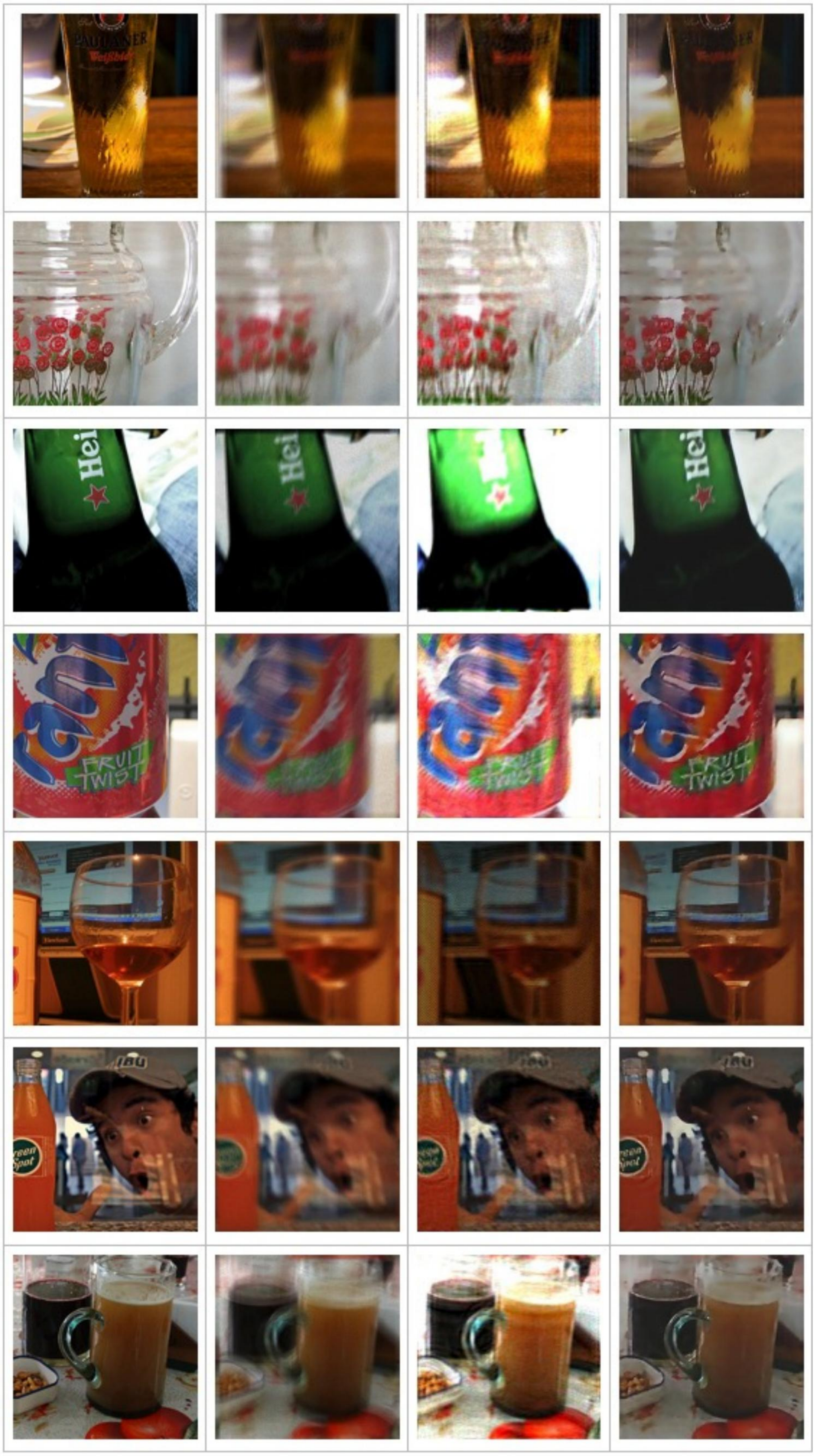

Figure 8. From left to right. The original, sharp image. The blurred image. The Wiener filtered image. The image generated from our model.