Non-Blind Image Deblurring using Neural Networks

Andy Gilbert Shai Messingher Anirudh Patel

Abstract

Each year, people take over one trillion photographs. Most of those are taken on smartphones, which lend themselves to motion blur. Our goal in this paper is to try to implement a neural network architecture to deconvolve the blur kernel out of the motion-affected image so that we can recover a sharp image. We show that we can do this with reasonable results, at least better than our baseline comparison. However, there is still much room for improvement.

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] among others. However, most of these methods require expensive computation costs to obtain top-quality sharp images.

More recently, neural networks have been utilized for image restoration. Though many of these methods work well, quite a few require re-training on each different possible input kernel.[3] This makes them laborious and not very useful in practice.

In this project, it was investigated whether a learning-based approach to non-blind image deconvolution could enhance conventional deconvolution techniques. The hypothesis was that a non-linear combination of conventional reconstructions of the same blurry image yields a sharper image. The system proposed 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 version.

1.1. Related Work

Over the years, numerous algorithms have been proposed for non-blind image deconvolution. The most relevant algorithms are discussed here. Because non-blind deconvolution is an ill-posed problem, assumptions were made at the offset of the project to constrain the solution space. For instance, Wiener filtering assumes that the value of each pixel should follow a Gaussian distribution. However, this is untrue for most natural images, in which the pixels are more likely to be heavily-tailed. The method of using Hyper-Laplacian priors attempts to circumvent this problem by assuming a heavily-tailed distribution. However, this is very time-consuming.

Recently, deep learning has also been proposed as a solution to low-level image processing problems like denoising [4], super-resolution [5], and edge-preserving filtering [6]. Still more recently, researchers have attempted non-blind image deblurring. In their paper, 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 randomply cropped into a 256x256 sub-image. For each cropped image, a trajectory of motion was generated. The 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.

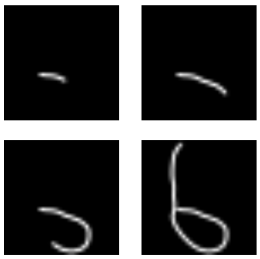

Figure 1. Generated PSFs from the same trajectory curve

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.

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

As described before, we now have a feature set that is 256x256x45, where the 45 comes from having 15 images split into RGB. As an input to the network we also found that stacking the original blurred image as an additional layer helped train the network. This appeared to provide some of the original color information to the network which helped it match the output image better. Therefore the final input dimension was 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 in to allow for building deeper networks. The number of Res Blocks was varied as a hyper-parameter and will be covered in Training Details. Each of the convolution layers had 64 filters and used a padding to maintain constant resolution (256x256). After the Res Blocks 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, but 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. 7 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 have a fair comparison the SNR 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
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 did not lead to a great numerical comparison of the effectiveness of the metric.

Using the SSIM loss did perceptually improve the results of the image.

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. This made it difficult to directly evaluate some of the tuning steps we used. Rather the loss stagnated for several epochs a different hyperparameter combination would be tried and evaluated for several epochs to see if the loss would increase, remain constant or decrease. However, 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.

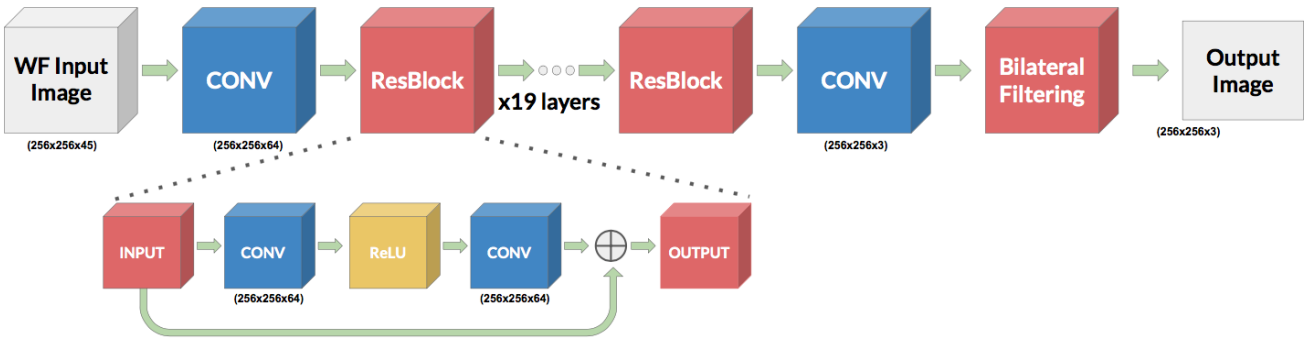

Figure 3. Architecture of our network. The number of Res Blocks was varied during the course of training from 19 to 12. The network is shown here with bilateral filtering added as the last layer.

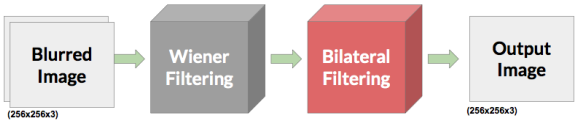

Figure 4. Architecture of the network we were comparing to. The network is shown here with bilateral filtering added as the last layer.

- **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 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 for noise reduction. It is important to note that for the Wiener filtering, we assumed the SNR to be equal to the average SNR of the entire set of test images. The results are summarized in Table 1

	Neural Net	Baseline
PSNR (dB)	26.1	16.6
Eval Time (sec)	0.347	0.023

Table 1. Summary of our results.

We see that our model performs almost 10dB better than the baseline on our test set. This is further substantiated by looking at the images themselves. For example, we take two pictures with the third largest kernel shown in Figures 5 and 6. However, as mentioned above this may be largely due to color differences.

You can see that in the image from our network (Figure 5), the lettering and star are significantly cleaner. That being said, there are very significant color artifacts that appear in the the black region of the bottle in our image. We believe these to be due to the fact that our inputs (shown in Fig. 2) were not normalized. We therefore run into the issue in which some of our inputs have really low dynamic range

Figure 5. The output image from our network.

Figure 6. The output image from our baseline.

(appear all black or close to all white). Imagine, for sake of argument, that one of our input images has values ranging from 0-0.05 and another has values ranging from 0-1. In this scenario, the contribution from the input with values ranging from 0-1 will dominate our loss function. This makes it vital that we re-scale our inputs so that their variability reflect their importance to our model. Since we don't know these a priori, a good step would have been to normalize all of our inputs to the same standard deviation beforehand so that we guarantee that their variability are at least not the inverse of their importance. Since the very black and very white regions lie at the extremities of the spectrum, it is very likely that the input images they correspond with most strongly have low dynamic resolution.

4. Conclusion and Future Work

In general, we have shown that our model performs slightly better than the baseline of Wiener filtering. However, there are significant color artifacts that can arise from it. Our output is also still not a completely sharp image reconstruction. For these reason, we believe that there are several modifications we would like to check in our next iteration of the project.

One of these is adding a Generative Adversarial Network (GAN) term into our loss function. We believe that adding this information that tells the networks what a sharp image should look like, the network would have a better idea about what to modify in order to get to this point.

As opposed to a modification, another aspect we'd like to further investigate is our baseline comparison. We believe that using ADMM as a comparison would be a fairer metric, and hope to use that to judge our modifications.

5. Acknowledgements

We'd 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'd 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.

6. 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

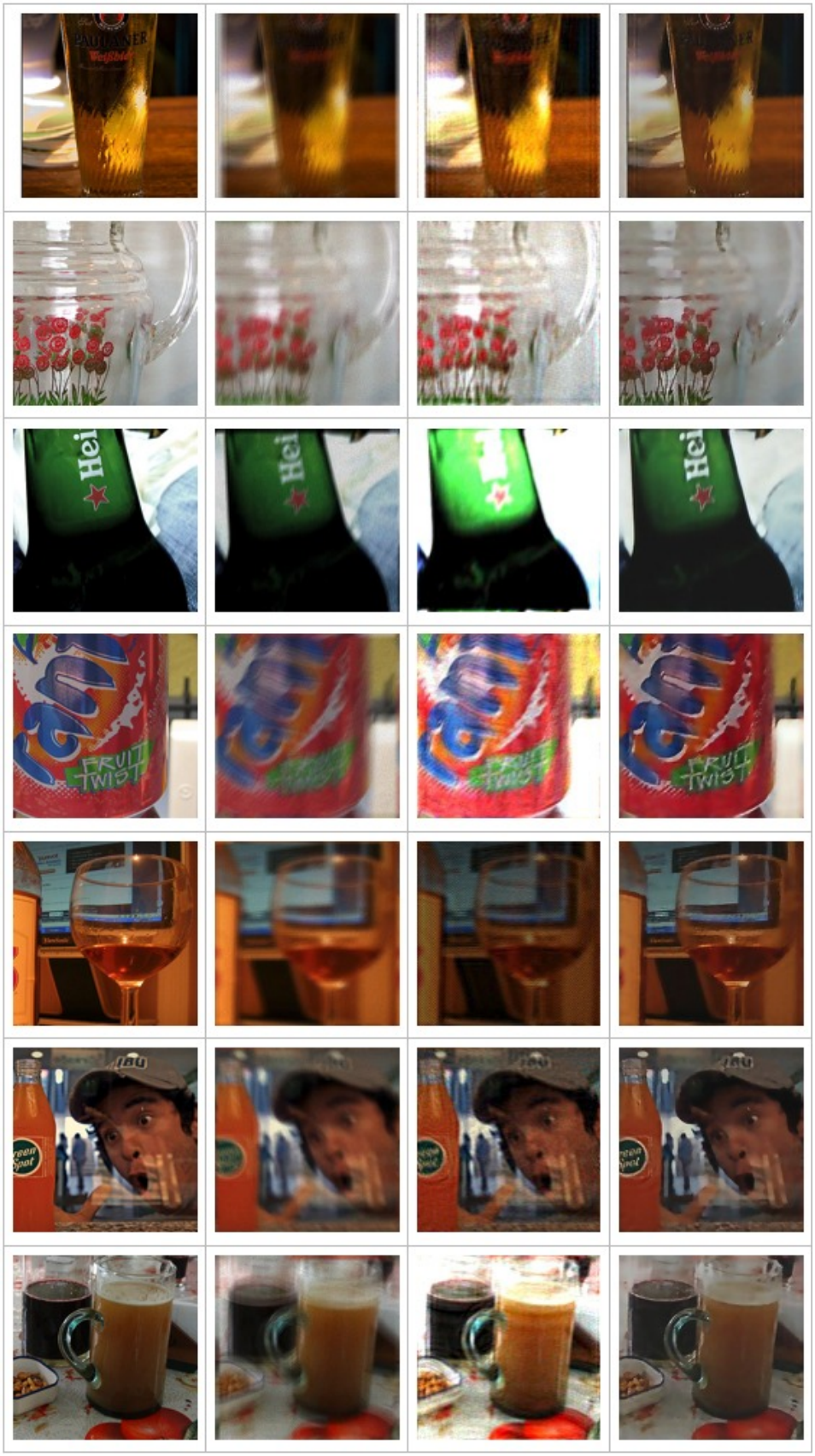

Figure 7. From left to right. The original, sharp image. The blurred image. The Wiener filtered image. The image generated from our model.

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.