Learning to Deblur

Christian J. Schuler[1], **Michael Hirsch**[1], **Stefan Harmeling**[1,2] **and Bernhard Schölkopf**[1]
[1] Max Planck Institute for Intelligent Systems, Tübingen, Germany
firstname.lastname@tuebingen.mpg.de
[2] Institut für Informatik, Heinrich-Heine-Universität Düsseldorf, Germany
firstname.lastname@uni-duesseldorf.de

Abstract

We show that a neural network can be trained to blindly deblur images. To accomplish that, we apply a deep layered architecture, parts of which are borrowed from recent work on neural network learning, and parts of which incorporate computations that are specific to image deconvolution. The system is trained end-to-end on a set of artificially generated training examples, enabling competitive performance in blind deconvolution, both with respect to quality and runtime.

1 Introduction

Blind image deconvolution is the task of recovering the underlying sharp image from a recorded image corrupted by blur and noise. Examples of such distortions are manifold: In photography, long exposure times might result in camera shake, often in combination with image noise caused by low light conditions. Lens aberrations can create blur across images in particular for wide apertures. In astronomy and remote sensing, atmospheric turbulence blurs images. All these image reconstruction tasks are examples of *inverse problems*, which are particularly hard to solve since not only the underlying image is unknown, but also the blur. We assume that the blurred image $\mathbf{y}$ is generated by linearly transforming the underlying image $\mathbf{x}$ (sometimes called the "true image" or "latent image") by a convolution (denoted by $*$), and additive noise $\mathbf{n}$,

$$\mathbf{y} = \mathbf{k} * \mathbf{x} + \mathbf{n}. \tag{1}$$

The task of blind image deconvolution is to recover $\mathbf{x}$ given only the blurry image $\mathbf{y}$, without knowing $\mathbf{k}$ and $\mathbf{n}$. A number of approaches have recently been proposed to recover the true image from blurred photographs, e.g., [1, 2, 3]. Usually these methods assume some sparsity inducing image prior for $\mathbf{x}$, and follow an iterative, multi-scale estimation scheme, alternating between blur and latent image estimation.

The idea of the proposed method is to "unroll" this reconstruction procedure and pose it as a nonlinear regression problem, where the optimal parameters are learned from artificially generated data. As a function approximator, we use a layered architecture akin to a deep neural network or multilayer perceptron. Some of the layers are convolutional, as popular in many recent approaches, while others are non-standard and specific to the problem of blind deconvolution. Overall, the system is inspired by [4] who formulated the idea of neural networks (NNs) as general processing schemes with large numbers of parameters.

Using extensive training on pairs of sharp and blurry images generated by applying simulated camera shakes on images from a large image dataset, we train the blind deconvolution NN to automatically obtain an efficient procedure to approximate the true underlying image, given only the blurry measurement.

A common problem with NNs, and in particular with large custom built architectures, is that the devil is in the details and it can be nontrivial to get systems to function as desired. We thus put particular emphasis on describing the implementation details, and we make the code publicly available.

Main contributions: We show how a trainable model can be designed in order to be able to learn blind deconvolution. Results are comparable to the state of the art of hand-crafted approaches and go beyond when the model is allowed to specialize to a particular image category.

2 Related work

Some recent approaches to blind deconvolution in photography have already been mentioned above. We refer the interested reader to [5] for an overview of the subject, and focus only on how NNs have been previously used for deconvolution.

Neural networks have been used extensively in image processing. Comprehensive reviews are [6, 7], both of which present broad overviews of applying NNs to all sorts of image processing operations, including segmentation, edge detection, pattern recognition, and nonlinear filtering. Image deconvolution, often also called image reconstruction, has been approached with NNs for a long time. However, these approaches are quite different from our proposed work:

- *NN to identify the type of blur:* [8] and similarly [9] apply NNs to identify the type of blur from a restricted set of parametrized blurs, e.g. linear-motion, out-of-focus and Gaussian blur, and possibly their parameters.
- *NN to model blurry images:* [10] models the blurry image as the result of a neural network where at the different layers the blur kernel and the true image appear.
- *NN to inversely filter blurry images:* [11] learns an inverse filter represented by a NN to deblur text.
- *NN to optimize a regularized least squares objective:* [12] proposes also the common two-stage procedure to first estimate the blur kernel and then to recover the image. For both tasks Hopfield networks are employed to solve the optimization problems.
- *NN to remove colored noise:* [13] presents a method for non-blind deblurring that starts with a straight-forward division in Fourier space and then removes the resulting artifacts (mainly colored noise) with large neural networks.

Other learning-based approaches for blind deconvolution try to learn the deconvolution solution for a single image, in particular they attempt to learn an appropriate sparse representation for a given blurry image, e.g. [14]. In our work, we follow a different strategy: instead of learning the solution or part of a solution for a single fixed image, we use a neural network to learn a general procedure that is directly applicable to other images and to different blurs. Closest to our approach is the work of [15], who train a deblurring procedure based on regression tree fields. However, even though their approach is not limited to a specific blur or a specific image, they consider only the problem of *non-blind* deblurring, i.e. their method assumes that the blur kernel is known. This is in contrast to our contribution, which demonstrates how to train a NN to solve the *blind* deconvolution problem, which is a much harder problem as both the blur and the latent image are unknown.

Finally we note that *deconvolutional networks*, introduced in [16] and further extended in [17], are not architectures for image deconvolution. Instead, they use convolutions to link sparse features to given images. Their goals are good image representations for tasks such as object recognition and image denoising.

3 Blind deconvolution as a layered network

Existing fast blind deconvolution methods work by alternating between the following three steps [18, 3, 2]:

1. *Feature extraction* computes image representations that are useful for kernel estimation. These representations may simply be the gradient of the blurry image and the gradient of the current estimate, possibly thresholded in flat regions [3]; they may also be preliminary estimates of the sharp image, for instance computed by heuristic nonlinear filtering [2].
2. *Kernel estimation* uses the extracted features to estimate the convolution kernel.
3. *Image estimation* attempts to compute an approximation of the sharp image using the current kernel estimate.

We will represent these steps by a trainable deep neural network, thus adding more flexibility to them and allowing them to optimally adapt to the problem. The layers of the network alternate between (1) a local convolutional estimation to extract good features, (2) the estimation of the blur kernel, globally combining the extracted features, and finally (3) the estimation of the sharp image. Parts (2)

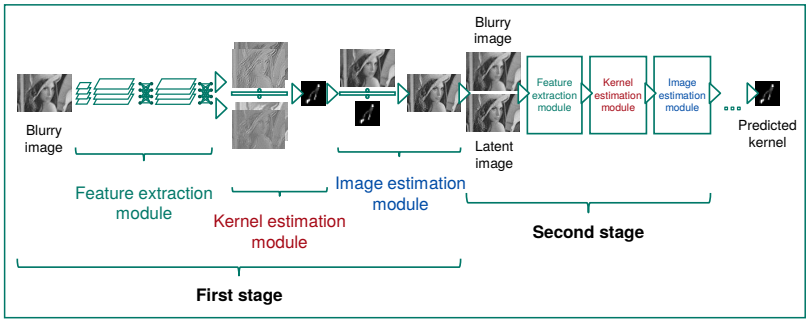


Figure 1: Architecture of our proposed blind deblurring network. First the *feature extraction module* transforms the image to a learned gradient-like representation suitable for kernel estimation. Next, the kernel is estimated by division in Fourier space, then similarly the latent image. The next stages, each consisting of these three operations, operate on both the blurry image and the latent image.

and (3) are fixed (having only one hyper-parameter for regularization). The free parameters of the network appear in part (1), the feature extraction module. Thus, instead of having to learn a model on the full dimensionality of the input image, which would not be doable using realistic training set sizes, the learning problem is naturally reduced to learning filters with a limited receptive field.

3.1 Architecture layout

3.1.1 Feature extraction module

What makes a good feature for kernel estimation can reasonably be assumed to be a translation invariant problem, i.e., independent from the position within the image. We therefore model the feature extractors using shared weights applying across all image locations, i.e., as a convolutional NN layer,[1] creating several feature representations of the image. This is followed by two layers that introduce nonlinearity into the model. First, every value is transformed by a tanh-unit, then the feature representations are pixel-wise linearly combined to new hidden images, formally speaking

$$\tilde{\mathbf{y}}_i = \sum_j \alpha_{ij} \tanh(\mathbf{f}_j * \mathbf{y}) \text{ and } \tilde{\mathbf{x}}_i = \sum_j \beta_{ij} \tanh(\mathbf{f}_j * \mathbf{y}) \tag{2}$$

where $\mathbf{f}_j$ are the filters of the convolution layer (shared for $\tilde{\mathbf{x}}_i$ and $\tilde{\mathbf{y}}_i$), the function $\tanh$ operates coordinate-wise, and α_{ij} and β_{ij} are the coefficients to linearly combine the hidden representations. Note that we usually extract several gradient-like images $\tilde{\mathbf{x}}_i$ and $\tilde{\mathbf{y}}_i$. Depending on the desired non-linearity, these two layers can be stacked multiple times, leading to the final image representations based on features useful for kernel estimation.

3.1.2 Kernel estimation module

Given $\tilde{\mathbf{x}}_i$ and $\tilde{\mathbf{y}}_i$ which contain features tuned for optimal kernel estimation, the kernel $\tilde{\mathbf{k}}$ can be estimated by minimizing

$$\sum_i \|\tilde{\mathbf{k}} * \tilde{\mathbf{x}}_i - \tilde{\mathbf{y}}_i\|^2 + \beta_k \|\tilde{\mathbf{k}}\|^2 \tag{3}$$

for $\tilde{\mathbf{k}}$ given the results from the previous step $\tilde{\mathbf{x}}_i$ and their blurry counterparts $\tilde{\mathbf{y}}_i$. Assuming no noise and a kernel without zeros in its power spectrum, the true gradients of the sharp image $\mathbf{x}$ and its blurred version $\mathbf{y}$ would return the true kernel for $\beta_k = 0$. Typically, in existing methods $\tilde{\mathbf{y}}_i$ are just the gradients of the blurry image, while here these can also be learned representations predicted from the previous layer. The minimization problem can be solved analytically and computed in one step in Fourier space if we assume circular boundary conditions [2]:

$$\tilde{\mathbf{k}} = F^{\mathsf{H}} \frac{\sum_i \overline{F\tilde{\mathbf{x}}_i} \odot F\tilde{\mathbf{y}}_i}{\sum_i |F\tilde{\mathbf{x}}_i|^2 + \beta_k}. \tag{4}$$

[1] Note that this convolution has nothing to do with the convolution appearing in our image formation model (Eq. (1)) — causally, it goes in the opposite direction, representing one step in the inverse process turning the blurry image into an estimate of the underlying image.

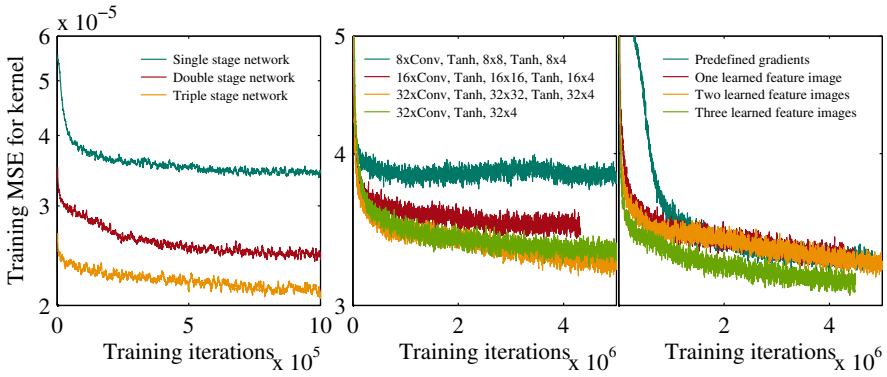


Figure 2: *Left:* Deeper networks are better at kernel prediction. One stage for kernel prediction consists of a convolutional layer, two hidden layers and a kernel estimation module. *Middle:* Performance depends on the number of predicted feature images used in the kernel estimation module. Predefining $\tilde{\mathbf{y}}_i$ to x- and y-gradients and not learning these representations slows down training.

Here F is the discrete Fourier transform matrix, F^{H} its Hermitian conjugate (i.e. inverse for F being a discrete Fourier transform matrix), $\odot$ the Hadamard product, $\overline{\mathbf{v}}$ the complex conjugate of a vector $\mathbf{v}$, and the division and absolute value are performed element-wise. The one step solution is only possible because we use a simple Gaussian prior on the kernel. We call this step the *quotient layer*, which is an uncommon computation in NNs that usually only combine linear layers and nonlinear component-wise thresholding units. The final kernel is returned by cropping to the particular kernel size and thresholding negative values. To reduce artifacts from the incorrect assumption of circular boundary conditions, the borders of the image representations are weighted with a Barthann window such that they smoothly fade to zero.

Due to varying size of the input image, we set $\beta_k = 10^{-4}$ for numerical stability only. This forces the network to not rely on the prior, which would lose importance for a larger input image relative to the likelihood term (since the kernel size is fixed).

3.1.3 Image estimation module

Before adding another *feature extraction module*, the estimated kernel is used to obtain an update of the sharp latent image: analogously to Eq. (3), we solve

$$\|\tilde{\mathbf{k}} * \tilde{\mathbf{x}} - \mathbf{y}\|^2 + \beta_x \|\tilde{\mathbf{x}}\|^2 \tag{5}$$

for $\tilde{\mathbf{x}}$, which can also be performed with a quotient layer. This can be done in one step (which would not be possible when using a sparse prior on $\tilde{\mathbf{x}}$). The following convolution layer then has access to both the latent image and the blurry image, which are stacked along the third dimension (meaning that the learned filters are also three-dimensional). The hyper-parameter β_x is also learned during training.

3.2 Iterations as stacked networks

The *feature extraction module*, *kernel estimation module* and the *image estimation module* are stacked several times, corresponding to multiple iterations in non-learned blind deconvolution methods. This leads to a single NN that can be trained end-to-end with back-propagation by taking the derivatives of all steps (see Appendix A for the derivatives of the solutions to Eq. (3) and the analogous Eq. (5)), increasing the performance as shown in Fig. 2 (but at the same time increasing runtime).

Similar to other blind deconvolution approaches, we also use a multi-scale procedure. The first (and coarsest) scale is just a network as described above, trained for a particular blur kernel size. For the second scale, the previous scale network is applied to a downsampled version of the blurry image, and its estimated latent image is upsampled again to the second scale. The second scale network then takes both the blurry image and the result of the previous scale as input. We repeat these steps until the final scale can treat the desired blur kernel size on the full resolution image.

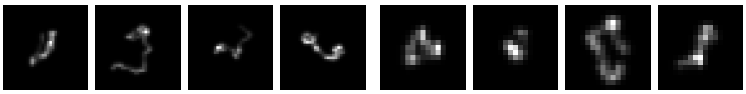


Figure 3: Examples of blurs sampled from a Gaussian process (left: 33 px, right: 17 px).

3.3 Training

To train the network, we generate pairs of sharp and blurry images. The sharp image is sampled from a subset of about 1.6 million images of the ImageNet [20] dataset, randomly cropped to a size of 256×256. Next, a blur trajectory is generated by sampling both its x- and y-coordinates separately from a Gaussian Process $f_x(t), f_y(t) \sim \mathcal{GP}\big(0, k(t,t')\big)$ with

$$k(t,t') = \sigma_f^2 \left(1 + \frac{\sqrt{5}|t-t'|}{l} + \frac{5(t-t')^2}{3l^2}\right) \cdot \exp\left(-\frac{\sqrt{5}|t-t'|}{l}\right), \tag{6}$$

which is a Matérn covariance function with $\nu = 3/2$ [21]. The length scale l is set to 0.3, the signal standard deviation σ_f to $1/4$. The coordinates are then scaled to a fixed blur kernel size and the final kernel is shifted to have its center of mass in the middle. This simple procedure generates realistic looking blur kernels, examples for both small and large kernel sizes are illustrated in Fig. 3. For every setting, we generate 1 million noise-free training examples, and add Gaussian noise during training (by default, $\sigma = 0.01$).

To avoid that the training process is disturbed by examples where the blur kernel cannot be estimated, e.g., a homogeneous area of sky without any gradient information, we reject examples where less than 6% pixels have gradients in x- and y-direction with an absolute value 0.05 or above.

As described in the previous subsection, a network for a certain blur kernel size, i.e., a particular scale, consists of several stages, each iterating between the *feature extraction*, *kernel estimation* and the *image estimation module*.

We use pre-training for our network: we start by training only one stage minimizing the L2 error between the estimated and the ground truth kernel, then add a second stage after about 1 million training steps. For the next 1000 steps, the parameters of the first stage stay fixed and only the parameters of the second stage are updated. After that, the network is trained end-to-end, until a potential next stage is added.

For the update of the parameters, convergence proved to be best with ADADELTA [19], a heuristic weighting scheme of parameter updates using gradients and updates from previous training steps. For our experiments, we choose a learning rate of 0.01 and a decay rate of 0.95. Moreover, it makes training more robust to outliers with strong gradients since it divides by the weighted root-mean-square of the seen gradients, including the current one. Responsible for the mentioned strong gradients are typically images dominated by abrupt step-edges, which create ringing artifacts in the deconvolution Eq. (5). In ImageNet these often are photos of objects with trimmed background. To make the training even more robust, we don't back-propagate examples with an error above 10 times the current average error.

4 Implementation

We make the code for both training and testing our method available for download on our project webpage[2]. For training, we use our own C++/CUDA-based neural network toolbox. After training once for a certain blur class (e.g., camera shake), which takes about two days per stage, applying the network is very fast and can be done in Matlab without additional dependencies. The runtime on an Intel i5 using only Matlab ranges between 1.5 sec for an image of size 255×255 pixels to 90 sec for a 4 MP image and a blur kernel size of 33×33 pixels.

5 Experiments

If not otherwise stated, all experiments were performed with a multi-scale, triple stage architecture. We use up to three scales for kernels of size 17×17, 25×25, 33×33. On each scale each *feature*

[2] `http://webdav.is.mpg.de/pixel/neural_blind_deconvolution/`

Valley img. (Flickr ID)	Xu et al. [18] (dB)	Cont. *agnostic* training (dB)	Cont. *specific* training (dB)
fCetj6	24.13	24.61	**24.96**
hR7YPb	**28.59**	27.96	28.32
i1GWi8	20.70	20.17	**20.91**
nz5BxB	**22.97**	22.28	22.80
nRuiRC	23.32	23.20	**23.44**
Average	23.94	23.64	**24.09**

Blackboard img. (Flickr ID)	Xu et al. [18] (dB)	Cont. *agnostic* training (dB)	Cont. *specific* training (dB)
bkkdz3	26.37	23.63	**27.53**
btKo6w	20.71	23.87	**26.32**
dbQBYX	20.88	22.36	**24.25**
f9jSxK	23.63	21.60	**24.41**
fK3V16	**25.32**	23.77	25.22
Average	23.38	23.04	**25.55**

Table 1: PSNRs of content *agnostic* vs. content *specific* training on images of the category *valley* (top) and *blackboard* (bottom). All images were downloaded in resolution "medium" and have been blurred with blur kernel number 2 from [22].

extraction module consists of a convolution layer with 32 filters, a tanh-layer, a linear recombination to 32 new hidden images, a further tanh-layer, and a recombination to four gradient-like images, two for $\tilde{\mathbf{x}}_i$ and $\tilde{\mathbf{y}}_i$ each (in the third stage: eight gradient-like images). In the case of the network with blur kernels of size 33×33, we deconvolve the estimated kernels with a small Gaussian with $\sigma = 0.5$ to counter the over-smoothing effect of the L2 norm used during training. For the specific choice of the architecture, we refer to the influence of model parameters on the kernel estimation performance in Fig. 2.

5.1 Image content specific training

A number of recent works [23, 24, 25] have pointed out the shortcoming of state-of-the-art algorithms [2, 3] to depend on the presence of strong salient edges and their diminished performance in the case of images that contain textured scenes such as natural landscape images. The reason for this is the deficiency of the so-called image prediction step, which applies a combination of bilateral and shock filtering to restore latent edges that are used for subsequent kernel estimation.

In this context, learning the latent image prediction step offers a great advantage: by training our network with a particular class of images, it is able to focus on those features that are informative for the particular type of image. In other words, the network learns *content-specific* nonlinear filters, which yield improved performance.

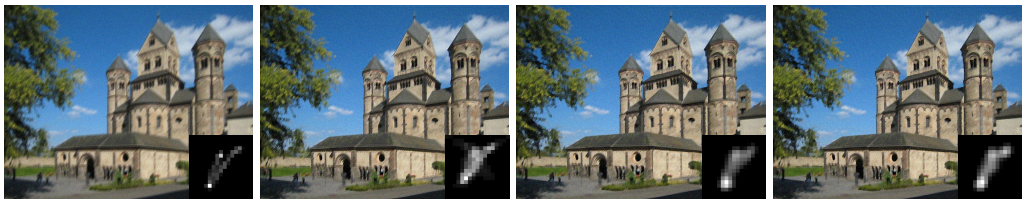
To demonstrate this, we used the same training procedure as described above, however, we reduced the training set to images from a specific image category within the ImageNet dataset. In particular, we used the image category *valley*[3] containing a total of 1395 pictures. In a second experiment, we trained a network on the image category *blackboard*[4] with a total of 1376 pictures. Table 1 compares the two networks on a random selection of photos taken from Flickr. We see that content specific training outperforms content agnostic training for all examples, and demonstrates on par performance with [18] for the category *valley*. For category *blackboard*, which is more distinct from generic natural images, it surpasses its competitor by a large margin.

5.2 Noise specific training

Typically, image noise impedes kernel estimation. To counter noise in blurry images, current state-of-the-art deblurring algorithms apply a denoising step during latent image prediction such as bilateral filtering [2] or Gaussian filtering [3]. However, in a recent work [26], the authors show that for increased levels of noise current methods fail to yield satisfactory results and propose a novel robust deblurring algorithm. Again, if we include image noise in our training phase, our network is able to adapt and learn filters that perform better in the presence of noise. In particular, we trained a network on images with Gaussian noise of 5% added during the training phase. Figure 4 compares the results for an image taken from [26] with 5% Gaussian noise for a network trained with 1% and 5% of added Gaussian noise during training, respectively. We also show the result of [26] and compare peak signal-to-noise ratio (PSNR) for objective evaluation. All results use the same non-blind deconvolution of [26]. The noise specific training is most successful, but even the noise agnostic NN outperforms the non-learned method on this example.

[3] ImageNet 2011 Fall Release > Geological formation, formation > Natural depression, depression > Valley, vale (`http://www.image-net.org/synset?wnid=n09468604`)

[4] ImageNet 2011 Fall Release > Artifact, artifact > Sheet, flat solid > Blackboard, chalkboard (`http://www.image-net.org/synset?wnid=n02846511`)



	PSNR 23.17 dB	PSNR 23.28 dB	**PSNR 23.52 dB**
Blurry image with ground truth kernel	Deblurring result of of Zhong et al. [26]	Deblurring result w. noise *agnostic* training	Deblurring result w. noise *specific* training

Figure 4: Comparison of deblurring results for NNs that have been trained with different amounts of noise added to the sample images during training. The network that has been trained with the same amount of noise as the input blurry image (5% noise) performs best. We also show the results of a recently proposed deblurring method tailored for increased levels of noise [26].

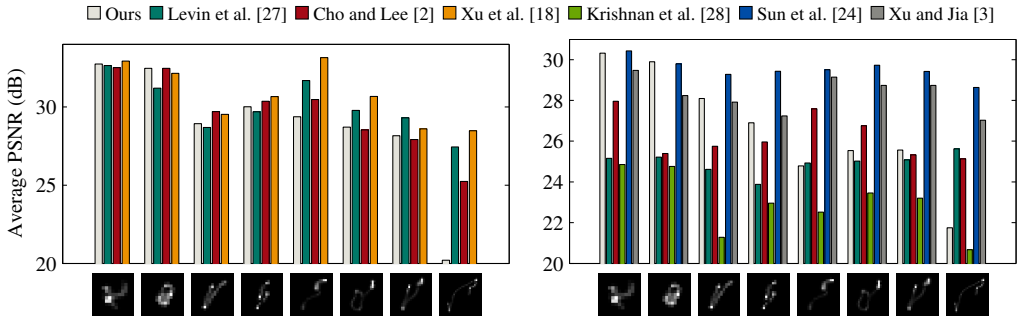


Figure 5: Results of the benchmark dataset of Levin et al. [22] and the extended benchmark of Sun et al. [24]. The results are sorted according to blur kernel size. While for kernels up to a size of 25×25 pixels, our approach yields comparable results, it falls short for larger blur kernels.

5.3 Benchmark Datasets

We evaluate our method on the standard test sets from [22, 24]. The four images of [22] are 255×255 pixels in size and are artificially blurred each with eight different blur kernels and contain 1% additive Gaussian noise. The performance is illustrated in Fig. 5 on the left, with blur kernels sorted according to increasing size. We compare with Levin et al. [27], Cho and Lee [2], and Xu et al. [18]. While our method is competitive with the state of the art for small blur kernels, our method falls short in performance for blur kernel sizes above 25×25 pixels. The second benchmark from [24] extends this dataset to 80 new images with about one megapixel in size each, using the same blur kernels as in [22]. Results are shown in Fig. 5 on the right. Here we compare with Levin et al. [27], Cho and Lee [2], Krishnan et al. [28], Sun et al. [24], and Xu and Jia [3], where however [24] has runtime in the order of hours. Again, we see competitive performance for small blur kernels.

6 Discussion

6.1 Learned filters

The task of the *Feature Extraction Module* is to emphasize and enhance those image features that contain information about the unknown blur kernel. Figure 6 shows the learned filters of the convolution layer for each of the three stages within a single scale of a trained NN for kernel size 17×17 pixels. While the first stage takes a single (possibly down-sampled) version of a blurry image as input, the subsequent stages take both the restored latent image (obtained by non-blind deconvolution with the current estimate of the kernel) and the blurry image as input. The outputs of each stage are nonlinearly filtered versions of the input images. In Fig. 7 we visualize the effect of the first stage of a NN with two predicted output images on both the Lena image and a toy example image consisting of four disks blurred with Gaussians of varying size. Note that our feature extraction module outputs nonlinearly filtered images for both the blurry and the latent sharp image, both of which serve as

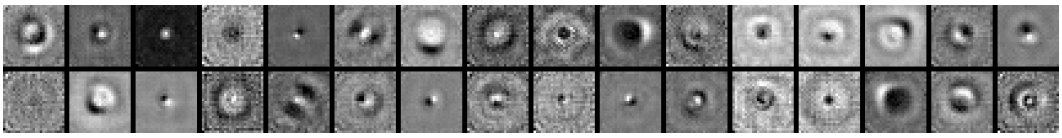


Figure 6: Learned filters of the convolution layer for the first iteration within a single scale of a trained NN for kernel size 17×17 pixels. See text for details.

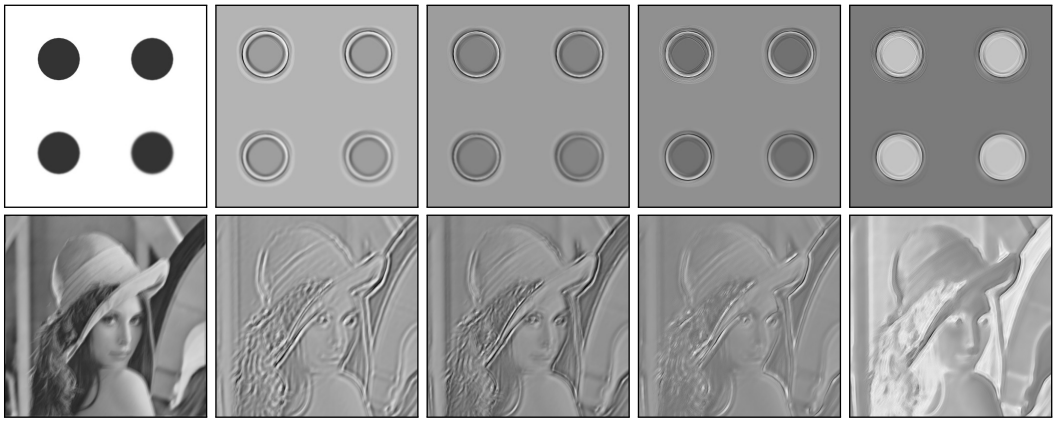


Figure 7: Visualization of the effect of the first stage of a network with two predicted output images on toy example with disks blurred with Gaussians of varying size (top row) and motion blurred Lena image (bottom row). While for the circles in $\tilde{\mathbf{y}}_i$ the different sizes of the Gaussian blurs are clearly visible, the NN replaces them in $\tilde{\mathbf{x}}_i$ with shapes of comparable sharpness.

input to the subsequent quotient layer, which in turn computes an estimate of the blur kernel. This is in contrast to other existing approaches [2, 3], which apply a *nonlinear* filter to the current estimate of the latent image, but use only a *linearly* filtered version of the blurry input image for kernel estimation. Once these feature images have passed the subsequent tanh and recombination layer, they serve as input to the *quotient layer*, which computes an estimate of the unknown blur kernel.

7 Conclusion

We have shown that it is possible to automatically learn blind deconvolution by reformulating the task as a single large nonlinear regression problem, mapping between blurry input and predicted kernel. The key idea is to incorporate the properties of the generative forward model into our network, namely that the image is convolved with the same blur kernel everywhere. While features are extracted locally in the image, the *kernel estimation module* combines them globally. Next, the *image estimation module* propagates the information to the whole image, reducing the difficulty of the problem for the following iteration.

Our approach can adapt to different settings (e.g., blurry images with strong noise, or specific image classes), and it could be further extended to combine deblurring with other steps of the imaging pipeline, including over-saturation, Bayer filtering, HDR, or super-resolution.

The blur class also invites future research: instead of artificially sampling from a stochastic process, one could use recorded spatially-varying camera shakes, or a different source of unsharpness, like lens aberrations or atmospheric turbulences in astrophotography. Additionally, the insights gained from the trained system could be beneficial to existing hand-crafted methods. This includes using higher-order gradient representations and extended gradient filters.

Scalability of our method to large kernel sizes is still an issue, and this may benefit from future improvements in neural net architecture and training.

References

[1] R. Fergus, B. Singh, A. Hertzmann, S. T. Roweis, and W. T. Freeman. Removing camera shake from a single photograph. *ACM Trans. Graphics*, 25(3):787–794, 2006.

[2] S. Cho and S. Lee. Fast motion deblurring. *ACM Trans. Graphics*, 28(5):145, 2009.

[3] L. Xu and J. Jia. Two-phase kernel estimation for robust motion deblurring. In *Computer Vision ECCV 2010*, Lecture Notes in Computer Science, pages 157–170. Springer, 2010.

[4] L. Bottou and P. Gallinari. A framework for the cooperation of learning algorithms. In *Advances Neural Information Processing Systems*, pages 781–788, 1991.

[5] D. Wipf and H. Zhang. Revisiting Bayesian Blind Deconvolution. *ArXiv e-prints*, 2013.

[6] M. Egmont-Petersen, D. de Ridder, and H. Handels. Image processing with neural networksa review. *Pattern recognition*, 35(10):2279–2301, 2002.

[7] D. De Ridder, R. P. Duin, M. Egmont-Petersen, L. J. van Vliet, and P. W. Verbeek. Nonlinear image processing using artificial neural networks. *Advances Imaging and Electron Physics*, 126:352–450, 2003.

[8] I. Aizenberg, D. Paliy, C. Moraga, and J. Astola. Blur identification using neural network for image restoration. In *Computational Intelligence, Theory and Applications*, volume 38, pages 441–455. Springer, 2006.

[9] Ch. Khare and K. K. Nagwanshi. Image restoration in neural network domain using back propagation network approach. *Int. J. Computer Information Systems*, 2(5):25–31, 2011.

[10] C. M. Cho and H. S. Don. Blur identification and image restoration using a multilayer neural network. In *IEEE Int. Joint Conf. Neural Networks*, pages 2558–2563, 1991.

[11] J. E. Tansley, M. J. Oldfield, and D. J. MacKay. Neural network image deconvolution. In *Maximum Entropy and Bayesian Methods*, Fundamental Theories of Physics, pages 319–325. Springer, 1996.

[12] R. J. Steriti and M. A. Fiddy. Blind deconvolution of images by use of neural networks. *Optics letters*, 19(8):575–577, 1994.

[13] Ch. J. Schuler, H. Ch. Burger, S. Harmeling, and B. Schlkopf. A machine learning approach for non-blind image deconvolution. In *IEEE Conf. Computer Vision and Pattern Recognition*, pages 1067–1074, 2013.

[14] Zh. Hu, J.-B. Huang, and M.-H. Yang. Single image deblurring with adaptive dictionary learning. In *IEEE Int. Conf. Image Processing*, pages 1169–1172, 2010.

[15] U. Schmidt, C. Rother, S. Nowozin, J. Jancsary, and S. Roth. Discriminative non-blind deblurring. In *IEEE Conf. Computer Vision and Pattern Recognition*, pages 604–611, 2013.

[16] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus. Deconvolutional networks. In *IEEE Conf. Computer Vision and Pattern Recognition*, pages 2528–2535, 2010.

[17] M. D. Zeiler, G. W. Taylor, and R. Fergus. Adaptive deconvolutional networks for mid and high level feature learning. In *IEEE Int. Conf. Computer Vision*, pages 2018–2025, 2011.

[18] L. Xu, Sh. Zheng, and J. Jia. Unnatural l0 sparse representation for natural image deblurring. In *IEEE Conf. Computer Vision and Pattern Recognition*, pages 1107–1114, 2013.

[19] M. D. Zeiler. ADADELTA: An Adaptive Learning Rate Method. *ArXiv e-prints*, 2012.

[20] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-fei. Imagenet: A large-scale hierarchical image database. In *IEEE Conf. Computer Vision and Pattern Recognition*, 2009.

[21] C. E. Rasmussen and Ch. K. I. Williams. *Gaussian Processes for Machine Learning*. 2006.

[22] A. Levin, Y. Weiss, F. Durand, and W. T. Freeman. Understanding and evaluating blind deconvolution algorithms. In *IEEE Conf. Computer Vision and Pattern Recognition*, pages 1964–1971, 2009.

[23] Zh. Hu and M.-H. Yang. Good regions to deblur. In *Computer Vision ECCV 2012*.

[24] L. Sun, S. Cho, J. Wang, and J. Hays. Edge-based blur kernel estimation using patch priors. In *IEEE Int. Conf. Computational Photography*, 2013.

[25] C. Wang, Y. Yue, F. Dong, Y. Tao, X. Ma, G. Clapworthy, H. Lin, and X. Ye. Nonedge-specific adaptive scheme for highly robust blind motion deblurring of natural images. *IEEE Trans. Image Processing*, 22(3):884–897, 2013.

[26] L. Zhong, S. Cho, D. Metaxas, S. Paris, and J. Wang. Handling noise in single image deblurring using directional filters. In *IEEE Conf. Computer Vision and Pattern Recognition*, pages 612–619, 2013.

[27] A. Levin, Y. Weiss, F. Durand, and W. T. Freeman. Efficient marginal likelihood optimization in blind deconvolution. In *IEEE Conf. Computer Vision and Pattern Recognition*, 2011.

[28] D. Krishnan, T. Tay, and R. Fergus. Blind deconvolution using a normalized sparsity measure. In *IEEE Conf. Computer Vision and Pattern Recognition*, 2011.

[29] Kaare Brandt Petersen and Michael Syskind Pedersen. *The matrix cookbook*. 2012.

A Appendix: Quotient Layer

As introduced in Eq. 4, the quotient layer performs the operation

$$\tilde{\mathbf{k}} = F^{\mathsf{H}} \frac{\sum_i \overline{F\tilde{\mathbf{x}}_i} \odot F\tilde{\mathbf{y}}_i}{\sum_i |F\tilde{\mathbf{x}}_i|^2 + \beta_k} \tag{7}$$

to estimate the kernel $\tilde{\mathbf{k}}$ from images $\tilde{\mathbf{x}}_i$ and their blurry counterparts $\tilde{\mathbf{y}}_i$, both predicted by the previous layers of the NN.

The quotient layer also includes a learned regularization parameter β_k. To train the NN, we need the gradients of the output with respect to its parameters (in this case $\tilde{\mathbf{x}}_i$, $\tilde{\mathbf{y}}_i$ and β_k) in every layer. From these we obtain the gradient steps $\Delta\tilde{\mathbf{x}}_j$, $\Delta\tilde{\mathbf{y}}_k$ and $\Delta\beta_k$ in terms of $\Delta\tilde{\mathbf{k}}$, where $\Delta\tilde{\mathbf{k}}$ is determined by the loss for the current training example, back-propagated through the layers subsequent to the quotient layer.

A.1 Derivative with respect to sharp images

To obtain $\Delta\tilde{\mathbf{x}}_j$, we first determine the differential form in numerator layout, which means for vectors $\mathbf{u}$, $\mathbf{v}$ and a matrix M that

$$d\mathbf{u} = \begin{pmatrix} du_1 \\ du_2 \\ \vdots \end{pmatrix}, \quad \frac{d\mathbf{u}}{d\mathbf{v}} = \begin{pmatrix} \frac{du_1}{dv_1} & \frac{du_1}{dv_2} & \cdots \\ \frac{du_2}{dv_1} & \frac{du_2}{dv_2} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} \quad \text{and} \quad dM = \begin{pmatrix} dm_{11} & dm_{12} & \cdots \\ dm_{21} & dm_{22} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}. \tag{8}$$

Therefore, assuming that only $\tilde{\mathbf{x}}_j$ is variable, with the rules of matrix calculus [29] we arrive at

$$\begin{aligned} d\tilde{\mathbf{k}} &= F^{\mathsf{H}} \frac{\overline{F d\tilde{\mathbf{x}}_j} \odot F\tilde{\mathbf{y}}_j}{\sum_i |F\tilde{\mathbf{x}}_i|^2 + \beta_k} - F^{\mathsf{H}} \frac{(\sum_i \overline{F\tilde{\mathbf{x}}_i} \odot F\tilde{\mathbf{y}}_i) \odot d(F\tilde{\mathbf{x}}_j \odot \overline{F\tilde{\mathbf{x}}_j})}{(\sum_i |F\tilde{\mathbf{x}}_i|^2 + \beta_k)^2} = \\ &= \underbrace{F^{\mathsf{H}} \operatorname{Diag}\left(\frac{F\tilde{\mathbf{y}}_j}{\sum_i |F\tilde{\mathbf{x}}_i|^2 + \beta_k}\right) \overline{F}}_{A} d\tilde{\mathbf{x}}_j \\ &\quad - \underbrace{F^{\mathsf{H}} \operatorname{Diag}\left(\frac{(\sum_i \overline{F\tilde{\mathbf{x}}_i} \odot F\tilde{\mathbf{y}}_i) \odot \overline{F\tilde{\mathbf{x}}_j}}{(\sum_i |F\tilde{\mathbf{x}}_i|^2 + \beta_k)^2}\right) F}_{B} d\tilde{\mathbf{x}}_j - \underbrace{F^{\mathsf{H}} \operatorname{Diag}\left(\frac{(\sum_i \overline{F\tilde{\mathbf{x}}_i} \odot F\tilde{\mathbf{y}}_i) \odot F\tilde{\mathbf{x}}_j}{(\sum_i |F\tilde{\mathbf{x}}_i|^2 + \beta_k)^2}\right) \overline{F}}_{C} d\tilde{\mathbf{x}}_j, \end{aligned} \tag{9}$$

with the Hadamard product $\mathbf{a} \odot \mathbf{b} = \operatorname{Diag}(\mathbf{a})\mathbf{b}$ and the operation "Diag" that creates a diagonal matrix from a vector. Additionally, $\tilde{\mathbf{x}}_i$ is real, and therefore $\overline{d\tilde{\mathbf{x}}_j} = d\tilde{\mathbf{x}}_j$.

With the differential $d\tilde{\mathbf{k}} = M d\tilde{\mathbf{x}}_j$ (where $M = A + B + C$) in numerator layout, we note that the derivative $\frac{d\tilde{\mathbf{k}}}{d\tilde{\mathbf{x}}_j} = M$ and the gradient step $\Delta\tilde{\mathbf{x}}_j = M^{\mathsf{T}}\Delta\tilde{\mathbf{k}}$. Additionally, we use that $B^{\mathsf{T}}\Delta\tilde{\mathbf{k}}$ is real since it is the inverse Fourier transform of a point-wise product of vectors all with Hermitian symmetry (they are themselves purely real vectors that have been Fourier transformed). Hermitian symmetry for a vector $\mathbf{v}$ of length n means $\mathbf{v}_{n-i+1} = \overline{\mathbf{v}_i}$ for all its components i. Therefore,

$$\begin{aligned} \Delta\tilde{\mathbf{x}}_j &= A^{\mathsf{T}}\Delta\tilde{\mathbf{k}} - B^{\mathsf{T}}\Delta\tilde{\mathbf{k}} - C^{\mathsf{T}}\Delta\tilde{\mathbf{k}} = A^{\mathsf{T}}\Delta\tilde{\mathbf{k}} - \overline{B^{\mathsf{T}}\Delta\tilde{\mathbf{k}}} - C^{\mathsf{T}}\Delta\tilde{\mathbf{k}} = \\ &= F^{\mathsf{H}} \frac{\overline{F\Delta\tilde{\mathbf{k}}} \odot F\tilde{\mathbf{y}}_j}{\sum_i |F\tilde{\mathbf{x}}_i|^2 + \beta_k} \\ &\quad - F^{\mathsf{H}} \frac{\overline{\sum_i \overline{F\tilde{\mathbf{x}}_i} \odot F\tilde{\mathbf{y}}_i} \odot F\tilde{\mathbf{x}}_j \odot F\Delta\tilde{\mathbf{k}}}{(\sum_i |F\tilde{\mathbf{x}}_i|^2 + \beta_k)^2} - F^{\mathsf{H}} \frac{(\sum_i \overline{F\tilde{\mathbf{x}}_i} \odot F\tilde{\mathbf{y}}_i) \odot F\tilde{\mathbf{x}}_j \odot \overline{F\Delta\tilde{\mathbf{k}}}}{(\sum_i |F\tilde{\mathbf{x}}_i|^2 + \beta_k)^2} \\ &= F^{\mathsf{H}} \left(\frac{\overline{F\Delta\tilde{\mathbf{k}}} \odot F\tilde{\mathbf{y}}_j}{\sum_i |F\tilde{\mathbf{x}}_i|^2 + \beta_k} - \frac{2\Re\left(\overline{\sum_i \overline{F\tilde{\mathbf{x}}_i} \odot F\tilde{\mathbf{y}}_i} \odot F\Delta\tilde{\mathbf{k}}\right) \odot F\tilde{\mathbf{x}}_j}{(\sum_i |F\tilde{\mathbf{x}}_i|^2 + \beta_k)^2} \right), \end{aligned} \tag{10}$$

since the transpose has no effect on a diagonal matrix. The real part of a vector $\mathbf{v}$ is denoted as $\Re(\mathbf{v}) = \frac{1}{2}(\mathbf{v} + \overline{\mathbf{v}})$.

A.2 Derivative with respect to blurry images

The derivation for the gradient $\Delta\tilde{\mathbf{y}}_j$ is similar. We again start with the differential form

$$d\tilde{\mathbf{k}} = F^{\mathsf{H}} \frac{\overline{F\tilde{\mathbf{x}}_j} \odot F d\tilde{\mathbf{y}}_j}{\sum_i |F\tilde{\mathbf{x}}_i|^2 + \beta_k} = \underbrace{F^{\mathsf{H}} \operatorname{Diag}\left(\frac{\overline{F\tilde{\mathbf{x}}_j}}{\sum_i |F\tilde{\mathbf{x}}_i|^2 + \beta_k}\right) F}_{A} d\tilde{\mathbf{y}}_j, \tag{11}$$

this time assuming $d\tilde{\mathbf{x}}_j$ and $d\beta_k$ to be zero. Next, we use again that a real vector is unchanged by conjugation, and we obtain

$$\Delta\tilde{\mathbf{y}}_j = A^{\mathsf{T}}\Delta\tilde{\mathbf{k}} = \overline{A^{\mathsf{T}}\Delta\tilde{\mathbf{k}}} = F^{\mathsf{H}}\overline{\operatorname{Diag}\left(\frac{\overline{F\tilde{\mathbf{x}}_j}}{\sum_i |F\tilde{\mathbf{x}}_i|^2 + \beta_k}\right) F\Delta\tilde{\mathbf{k}}} = F^{\mathsf{H}}\left(\frac{F\tilde{\mathbf{x}}_j \odot F\Delta\tilde{\mathbf{k}}}{\sum_i |F\tilde{\mathbf{x}}_i|^2 + \beta_k}\right). \tag{12}$$

A.3 Derivative with respect to the regularization parameter

Following the previous procedure for the regularization parameter β_k we arrive at

$$d\tilde{\mathbf{k}} = F^{\mathsf{H}} \underbrace{\frac{\sum_i \overline{F\tilde{\mathbf{x}}_i} \odot F\tilde{\mathbf{y}}_i}{\left(\sum_i |F\tilde{\mathbf{x}}_i|^2 + \beta_k\right)^2}}_{A} d\beta_k, \tag{13}$$

assuming only β_k to be variable. Then, multiplying $\Delta\tilde{\mathbf{k}}$ by the transpose of $\frac{d\tilde{\mathbf{k}}}{d\beta_k} = A$ we get

$$\Delta\beta_k = A^{\mathsf{T}}\Delta\tilde{\mathbf{k}} = \left(F^{\mathsf{H}} \frac{\sum_i \overline{F\tilde{\mathbf{x}}_i} \odot F\tilde{\mathbf{y}}_i}{\left(\sum_i |F\tilde{\mathbf{x}}_i|^2 + \beta_k\right)^2}\right)^T \Delta\tilde{\mathbf{k}}. \tag{14}$$