

Blur Kernel Estimation by Deep Learning

Masters Thesis

in Computer Science

submitted by

Manuel Jüngst

born on May 29th, 1993 in Siegen

Written at

Computer Graphics and Multimedia Systems Group

Faculty 12

University of Siegen, Germany.

Advisors:

Prof. Dr. A. Kolb, Computer Graphics and Multimedia Systems Group, University of Siegen

M.Sc. T. M. Wong, Computer Graphics and Multimedia Systems Group, University of Siegen

Started on:

November 16th, 2020

Finished on:

May 17th, 2021

Eidesstattliche Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

A handwritten signature in dark ink, appearing to read 'M. Jengst', is written above a horizontal line.

Siegen, den 17. Mai 2021

Zusammenfassung

Blind-Deblurring ist die Aufgabe Unschärfe von einem Bild zu entfernen, ohne zu wissen welche Verunschärfung vorliegt, d.h. ohne die Punktverteilungsfunktion / den Blur-Kernel zu kennen. In dieser Arbeit schlage ich einen neuen Deep Learning Ansatz vor, genannt Kernel Prediction Network, welches den Blur-Kernel in natürlichen Bildern schätzt. Der geschätzte Kernel kann in Verbindung mit beliebigen existierenden Non-blind Deblurring Methoden verwendet werden, um daraus ein geschärftes Bild zu erhalten. Im Gegensatz zu früheren Verfahren kann das vorgeschlagene Netzwerk schwierige Bewegungsunschärfe schätzen, ohne ein scharfes Bild schätzen zu müssen. Die Architektur ist ebenfalls viel tiefer als frühere Verfahren für schwierige Bewegungsunschärfe. Eine einzige Instanz des Kernel Prediction Network kann auch andere Unschärfearten verarbeiten, beispielsweise Gaußsche Unschärfe. Ich evaluiere mein Verfahren gegen andere Blind-Deblurring Verfahren mit Deep Learning, auf drei neuen Testsätzen, und schneide vorteilhaft ab. Während Deep Blind-Deblurring immer noch hinter traditionellen Blind-Deblurring in einigen Fällen liegt, bietet die vorgeschlagene Methode noch viele Bereiche für Verbesserungen. Die vorgeschlagene Implementierung beabsichtigt nicht bis zur Grenze zu optimieren. Stattdessen legt sie viele Grundsteine. Diese Arbeit gibt einen Anfang bis Ende Methodologie um ein Kernel Prediction Network zu entwerfen. Sie fängt an bei theoretischen Grundlagen, geht über Datensatz Erstellung, Verlustfunktion, Architektur, Optimierung, Non-blind Deconvolution und schließlich Evaluierung.

Abstract

Blind deblurring is the task of removing blur from an image, without knowing how it was blurred, i.e. without knowing the point spread function / blur kernel. In this thesis I propose a novel deep learning approach, termed kernel prediction network, that estimates the blur kernel from natural images. The estimated kernel can be used in combination with any existing non-blind deblurring method in order to get a deblurred image. Unlike prior work, the proposed network can estimate difficult motion blur, without having to estimate a sharp image. The architecture is also much deeper than prior work on complex motion blur. A single instance of the kernel prediction network can also handle other blur types such as Gaussian blur. I evaluate my approach against other blind deblurring approaches with deep learning, on three new test sets, and perform favorably. While deep blind deblurring is still behind traditional blind deblurring in some cases, the proposed method offers many areas for improvement. The proposed implementation does not intend to optimize the approach to its limits. Rather, a lot of groundwork is laid. This thesis gives a start-to-finish methodology for designing a kernel prediction network. It starts at theoretical foundations, goes over dataset creation, loss function, architecture, optimization, non-blind deconvolution and finally evaluation.

Acknowledgements

I am especially grateful to Jannik Zuther, who provided me with the GPU that made this thesis possible, free of charge. They did so at their own expense, and whilst navigating through difficult times. This GPU ran tirelessly for months on end, and my old GTX 660 was not even close to make this happen. I would also like to thank my advisor M.Sc. Tak Ming Wong, who offered regular feedback and encouragement throughout the extensive duration of this thesis.

Manuel Jüngst

Contents

1	Introduction	1
2	Theoretical Foundations	3
2.1	Classical Estimation of Image and Kernel	3
2.2	Classical Estimation of Kernel	4
2.3	Non-blind Deconvolution	5
2.4	Blur Types	6
3	Related Work	8
4	Methodology	10
4.1	Selecting a MAP Estimator	10
4.2	Kernel Prediction Network by Solving MAP_k	12
4.3	Training Set	14
4.4	Loss Function	15
4.5	Network Architecture	17
4.5.1	Feature Extraction Network	17
4.5.2	Encoder-Decoder Network	18
4.6	Non-blind Deconvolution	19
5	Implementation	21
5.1	Architecture Details	21
5.2	Optimization	22
5.3	Dataloader	23
5.4	Motion Blur Parameters	25
6	Numerical Experiments	27
6.1	Experimental Setup	27
6.1.1	Test Sets	27
6.1.2	Non-blind Deconvolution	29
6.1.3	Blind Deblurring Overview	32

6.2	Evaluation on Gaussian Blur	34
6.3	Evaluation on Real Motion Blur	36
6.4	Evaluation on Synthetic Motion Blur	42
6.5	Understanding the Proposed Method	45
7	Conclusion and Future Work	48
A	Intuitive Derivation of a MAP_k Network	50
B	Crop Architecture	52
	References	54

List of Figures

4.1	$\text{MAP}_{x,k}$ vs. MAP_k	11
4.2	Deep MAP_k estimation	13
4.3	Examples of synthesized kernels	14
4.4	Centering kernels	15
4.5	Shift-tolerant loss function	16
4.6	Kernel Prediction Network	18
5.1	Architecture implementation	22
5.2	Convergence of optimization	23
5.3	Approximating real motion blur	25
6.1	Example kernels of created test sets	28
6.2	Non-blind deconvolution method comparison	30
6.3	Effect of kernel error on deblurring	35
6.4	Effects of different kernels vs. images on the <i>BSD-Levin</i> set	38
6.5	Qualitative comparison	39
6.6	Qualitative comparison for failure case	40
6.7	Assessment of error on motion kernels	44
6.8	Attribution via Integrated Gradients	47
B.1	Crop architecture	52

List of Tables

6.1	Blind deblurring method overview	33
6.2	Evaluation on <i>Gaussian3x3</i> set	34
6.3	Evaluation on <i>BSD-Levin</i> set	37
6.4	Evaluation on <i>Motion-Length</i> set	43
6.5	Train-val gap	45

Chapter 1

Introduction

The task of blind deblurring, or blind deconvolution, has been researched for at least half a century. Much like other tasks, the recent era of deep learning has also introduced new perspectives to this task. The desire to remove blur from images exists since images exist. Any conventional image can be blurred e.g. due to motion, out-of-focus regions, or optical properties of the sensor. In some cases, the blur kernel, or point spread function, is known. For instance when a calibration image is taken, or when sensor movements are recorded. In most cases however, the parameters of the blur are not known, so deblurring is said to be blind.

To understand how to deblur, which is an inverse problem, we should first understand how images are blurred. The following describes a simple model of blur

$$y = x * k + \eta. \tag{1.1}$$

The blurry image y is created from a sharp image x , and a blur kernel k . The blur kernel, or point spread function, describes the pattern in which every point in the sharp image is spread. This model assumes that every point in the image is spread by the same pattern, which is called spatially uniform blur. Mathematically this spread is described by the convolution operation $*$. Next to the aforementioned predictable components, there is also some random noise η in the blur process, which makes solving the inverse problem difficult. In natural images, noise is introduced by the random nature of photons, but also electronic components of the optical sensor.

To deblur an image, we need not necessarily know the blur kernel. If we have enough prior knowledge about how objects in an image should look, we can correct them as such. But a less restrictive approach is to estimate the blur pattern in some form, so as to undo the blur in a more systematic manner. Deep learning enables us to deblur without any explicit kernel estimates. But given that deep networks can generalize to unknown images, some implicit knowledge about the blur is also learned. When explicitly estimating a kernel, we can estimate the sharp image simultaneously. Intuitively, this allows some trial and error to see what produces desirable results. Alternatively, we can only focus on the kernel, then solve the easier non-blind deblurring problem.

The topic of this thesis is to estimate a mixture of blur types, as are found in natural images. Primarily, challenging motion blur is estimated. For one, this tests the estimation limits of the proposed method, and it also serves as the main point of comparison to other methods. Gaussian blur, a simple and common blur type, is also estimated. This is to ensure the proposed method can generalize to a mix of blur types. Moreover I find that Gaussian blur estimation is a good baseline for design of blind deblurring algorithms. One of the premises for the proposed algorithm is to use deep learning, due its new and less explored perspective on blind deblurring. Another premise for this work is to estimate just blur kernels, and focus on evaluation of blur kernels. Images will still be deblurred with a non-blind method, but mainly for comparison to other methods.

The main contribution of this thesis is a novel blind deblurring formulation. The proposed kernel prediction network implements a maximum a posteriori estimator on only kernels, namely MAP_k . To the best of my knowledge, this is the first deep learning implementation of a MAP_k estimator that is applied to non-trivial motion blur. Secondly, this is also the first *deep* architecture applied to non-trivial motion blur, to the best of my knowledge. By non-trivial or complex motion blur I mean motion paths that are not just straight lines or simple curves, and as such cannot be easily parametrized. Thirdly, the proposed kernel estimation is exceptionally fast (on GPU), while still giving fairly accurate estimates.

This thesis is organized as follows. Chapter 2 is an introduction to classical blind deblurring and different blur types. In chapter 3, existing approaches are mentioned for a range of categories. Namely estimator type, blur type, and deep vs. classical algorithms. Since we focus on complex motion blur, a close look at existing deep estimators for motion blur is taken. The proposed algorithm is then described in chapter 4 and chapter 5. First the key conceptual aspects are discussed. Then the implementation details required to replicate the results are mentioned. Comparisons to other methods are discussed in chapter 6, following an overview of the experimental setup. A few shortcomings of the proposed algorithm are also discussed. Finally chapter 7 concludes what was done and is yet to be done.

Chapter 2

Theoretical Foundations

One of the premises of this thesis is to employ deep learning to solve blur kernel estimation for blind deblurring. The idea to use deep learning to solve blind deconvolution, or inverse problems in general, is not new. In the case of image blur, the inverse problem is to get back a sharp image x , given a blurry image y . Traditionally such an inverse problem is solved by modeling a reconstruction problem. This requires modeling the forward operator, e.g. eq. (1.1) in the case of spatially invariant blur, cost function (e.g. ℓ_2), regularizer and optimizer [MJU17]. So it is tempting to train a deep learning algorithm \mathcal{N}_θ to skip all these steps. Instead of us, this algorithm then learns all the components of the classical reconstruction, leaving us with an operator to deblur $x = \mathcal{N}_\theta(y)$. Indeed many recent works do blind deblurring without explicitly modeling any traditional components like eq. (1.1) [NHKML17, TGS⁺18, ZDLK19, YLC20].¹

Often however, traditional knowledge can be incorporated into deep network design to increase performance. There is no one way to do this however. For instance layers of the network architecture can be inspired by a classical algorithm (i.e. deep algorithm unrolling) [LTME19, AKB⁺20]. Or for encoder-decoder architectures, the decoder can be a classical model, e.g. for Gaussian kernel estimation, the decoder can be the Gaussian function. In the case of blind deblurring, many decades of knowledge relating to classical algorithm design are available to inspire deep learning algorithm design.

2.1 Classical Estimation of Image and Kernel

A common blind deblurring formulation considers both image x and blur kernel k . Starting with some initialization (e.g. blurry image and a delta kernel²), the estimates x and k are iteratively refined (according to some cost and regularization function). When this optimization converges, the result

¹These works intentionally forego eq. (1.1) due to its spatially invariant blur assumption, that often does not hold in blurry images.

²The delta kernel is named after the Dirac delta, and is an empty kernel $\mathbf{0}$ with a value of one in the center. This is also called no-blur or identity kernel, since it does not modify the image it is convolved with.

should be $x * k \approx y$. This problem is formulated as finding the x and k that are most likely to explain the blurry image y , i.e. $\arg \max_{x,k} p(x,k|y)$. The solution is then formulated via Bayes' theorem as

$$\arg \max_{x,k} p(x,k|y) = \arg \max_{x,k} p(y|x,k)p(x)p(k). \quad (2.1)$$

The denominator that would follow on the right has been dropped since it is not relevant for a maximization problem. By convention of the theorem, the probability on the left side is called posterior probability, and the right-hand side has the likelihood and prior probabilities. In full, the left side is called the maximum a posteriori (MAP) estimate, and in this case $\text{MAP}_{x,k}$ to indicate that image and kernel are optimized.

Following Perrone and Favaro, a standard implementation of this is

$$\arg \min_{x,k} \|x * k - y\|_2^2 + \lambda J(x) + \gamma G(k), \quad (2.2)$$

where each summand roughly corresponds to one right-hand term of eq. (2.1) [PF15]. The first term contains the spatially invariant convolution operation of image and kernel as the forward model. The cost function is the ℓ_2 norm, measuring the difference between the blurry image and the output of the convolution. The regularizers J and G are smoothness priors for image and kernel, with some non-negative scalars λ and γ . Compared to a naïve deep learning formulation, which we may label MAP_x , the classical eq. (2.2) leverages what we know about blur, namely the convolution operation.

2.2 Classical Estimation of Kernel

An alternative blind deblurring formulation splits the problem into two disjoint subproblems. First only the blur kernel is estimated, and once obtained, it can be used to solve a simpler (convex) problem called non-blind deconvolution. This problem is formulated as

$$\arg \max_k p(k|y) = \arg \max_k \int p(y|x,k)p(x)p(k) dx, \quad (2.3)$$

where x is no longer optimized, by marginalizing over all possible images x . An implementation of this MAP_k estimator is only feasible as an approximation, because computing all possible images is intractable. The standard implementation is using a Variational Bayesian (VB) approach for this approximation [PF15]. In practice, this approximation often ends up similar to the $\text{MAP}_{x,k}$ approach. Levin et al. mention that several blind deconvolution algorithms can be viewed as approximations of MAP_k , even if not motivated that way [LWDF09]. Wipf and Zhang have shown that a common MAP_k formulation is equivalent to an unconventional $\text{MAP}_{x,k}$ approach [WZ14, PF15].

In principle, the motivation for MAP_k is solid. As identified by Levin et al., $\text{MAP}_{x,k}$ estimation suffers from a fundamental problem. For $\text{MAP}_{x,k}$ estimation, the number of measurements collected can never be enough, because the number of unknowns grows with image size. In other words, even if

the blurry image becomes large such that more samples can be collected, the sharp image to estimate also becomes larger. The motivation for MAP_k is that the kernel is relatively small, and does not grow with image size. Therefore it is possible for the blurry image to become large enough to estimate the kernel.

In practice however, $\text{MAP}_{x,k}$ remains the dominant approach (for classical deblurring, and also deep deblurring). This is despite the fact that for some $\text{MAP}_{x,k}$ formulations, it has been proven that they are unable to succeed (i.e. to avoid the no-blur solution). It is also known that $\text{MAP}_{x,k}$ optimization is more prone to get stuck in local minima [LWDF09, WZ14, PF15, FSH⁺06]. Perrone and Favaro investigated these issues for the case of $\text{MAP}_{x,k}$ deconvolution with a Total Variation (TV) prior. They found that an apparently harmless delayed normalization step causes signal scaling that allows the algorithm to succeed. They label this approach projected alternating minimization (PAM), and show that it can find the true solution, but does not minimize the original problem formulation. They also found that other optimization tricks were not needed for their PAM algorithm to succeed and be competitive. In general however, the $\text{MAP}_{x,k}$ toolbox also contains strategies like image filtering, blur kernel priors and edge enhancement. As Levin et al. also pointed out, $\text{MAP}_{x,k}$ optimization does favor sharp solutions near significant edges. Indeed, some form of salient edge detection is successfully used by many works, e.g. [XPZY17, XJ10, JSK08, SCWH13, YS16]. Perrone and Favaro also found that their PAM algorithm automatically mimics an edge selection scheme.

To summarize, an alternative MAP_k estimator for blind deblurring was presented. In principle it has some clear advantages, namely a more solid theoretical foundation and being easier to optimize. In practice $\text{MAP}_{x,k}$ still achieves good results, so long as appropriate optimization strategies are employed. The common implementation of MAP_k is also not fundamentally different, since an equivalent $\text{MAP}_{x,k}$ formulation exists.

2.3 Non-blind Deconvolution

If a MAP_k estimator is used, it outputs only a kernel. To obtain a deblurred image, a non-blind deconvolution is used. The term blind refers to whether a blur kernel (estimate) is available. Luckily the subproblem of non-blind deconvolution is easier, and many such algorithms already exist.

In the simplest case, deconvolution is done with an inverse filter. The easy way to do this is to transform both the blurry image and kernel into the frequency domain via (fast) Fourier transform (FFT). Let y_f and k_f be the blurry image and kernel in the frequency domain, respectively. The appeal of working in frequency domain is that basic convolution becomes multiplication, and with that deconvolution in essence just becomes division. Deconvolution is then $x_f = y_f \cdot \frac{1}{k_f}$, where $\frac{1}{k_f}$ is the inverse filter. The resulting sharp image is easily transformed back into spatial domain with the inverse FFT. One problem with convolution in the frequency domain is that it represents a circular convolution, which becomes incorrect at the image boundaries [KRS17, LFD07a]. As a result, the deblurred image inherits artifacts at the boundaries. These artifacts can be mitigated (but not removed)

by padding the image. The optimal mode and amount of padding largely depends on the deconvolution algorithm. To do deconvolution in the spatial domain, convolution is formulated as a linear operator, i.e. a matrix. Such a matrix is a special case of a Toeplitz matrix, that is typically very large and sparse, containing the rows of the kernel in increasing diagonal offsets. This matrix can also be inverted to create an inverse filter in the spatial domain. Ultimately both frequency and spatial domain algorithms remain in use, depending on the desired speed vs. quality trade-off.

An inverse filter is almost never used, because in the general case the kernel cannot be inverted, and due to its sensitivity to noise. A much more noise-robust filter is the Wiener filter $k_f^*/(k_f^*k_f + \lambda)$, where $*$ denotes the complex conjugate, and λ is a non-negative regularization parameter that is tuned to the amount of noise. A small variation of this, which in this thesis is labeled as Wiener-Hunt, uses $\lambda L_f^* L_f$ for the regularization component (in place of λ), where L is the 5-element Laplacian filter.

These early deblurring solutions are simple and easy to compute, but do not yield state-of-the-art results. Modern classical non-blind deconvolution algorithms are not dissimilar to blind deconvolution. Sparse image priors are used, and the deconvolution becomes an iterative optimization as sparse priors are no longer convex. In summary, non-blind deconvolution allows us to deblur an image, when a MAP_k estimator was used to obtain a kernel. One thing to note though is that generally non-blind deconvolution algorithms assume a kernel to be a perfect estimate.

2.4 Blur Types

Not every blur is the same, and there are some differences in the way they are estimated and deblurred. Depending on the field, the term used to describe the blur can also be point spread function (PSF). Most of the literature that primarily deals with deblurring is focused on motion blur. But other blur types are e.g. Defocus, Gaussian, or Sinc. Defocus blur is common in natural images, since generally there is a plane in the scene where contents are in focus, and anything in front or behind is somewhat out of focus. This is also called a disk/disc blur, due to the shape of the kernel which is just a constant-valued circle. Gaussian blur in its simplest form is similar to defocus blur, in the sense that it is circular and can be described by a single parameter, but the values are not constant. In the two-dimensional case, Gaussian blur can also have a different standard deviation (width) σ in each dimension, and its rotation also becomes apparent. There is no field that focuses specifically on sinc blur, but much like the Gaussian function, it can be found in a number of physical processes. For instance, a rectangle in spatial domain becomes a sinc function after Fourier transform. More subtle blur types can also be caused by intrinsic camera properties, such as small lens defects [JSK08].

Other than blur types, there are also two blur models. Although fundamentally identical, one may classify two types of motion blur. One is caused by dynamic movement in the scene, e.g. a crowd of people walking in different directions. Another is caused by movement of the optical sensor, which blurs the entire scene. Kernels of the former example tend to be straight or curved lines, with different directions depending on the movement of the person. In the latter case, the kernels can become

very complex, with sudden directional changes, sometimes forming loops and self-intersections. In practice, neither case produces a uniform blur kernel across the entire scene. In the crowded scene, each person gets a different kernel. This is called spatially variant or varying blur, or sometimes dynamic blur. But even if only the optical sensor moves, if there is any rotation around the Z-axis, it effects spatially variant blur [LWDF09]. For another blur type, e.g. defocus blur is also spatially variant, depending on the distance of objects to the sensor. Most classical blind deblurring literature uses the spatially uniform blur model. One simple way to rectify this is to apply a spatially uniform blur model to smaller image patches, so that different kernels may be estimated.

Lastly, note how blurs can be represented. In the general case, a blur is contained in a discrete image/matrix. This canvas that a kernel is contained in is called kernel support. By convention this kernel support has an odd width and height. This is because an even size kernel support has no center point. The size of the kernel support is usually selected depending on the context of the application, where it is set just large enough to contain the blurs. The larger the kernel support size, the more variables there are to estimate. Roughly speaking, a size of around 23×23 can contain a range of common blurs, while the largest kernel supports are in the order of a hundred. This is still substantially smaller than most images nowadays. In some contexts, kernels can also be represented just by parameters. For simple Gaussian, defocus or sinc blur, just one scalar value is required. For simple motion blur, one can just use length and rotation of a line. This is what makes complex motion blur estimation interesting, because it is challenging to estimate, as no simple set of parameters exists to describe it.

Chapter 3

Related Work

This chapter mentions existing blind deblurring solutions in a range of areas. First some classical *motion blur* estimators are mentioned, then *other blur* types with both classical and deep learning estimators. This is meant to give a general overview. Then a specific look at deep learning approaches for difficult kernels (i.e. complex motion blur) is taken.

A popular classical maximum a posteriori estimator for image and kernel, namely $\text{MAP}_{x,k}$, is the Robust Motion Deblurring (RMDb) approach by Xu and Jia [XJ10]. For a more recent comparison of blind motion deblurring methods, see [PF15, SCWH13]. A very recent classical motion blur estimation work is proposed by Wen et al. [WYL⁺20]. Li et al. propose a classical estimation algorithm, but imbued with a deep prior that is trained to classify sharp vs. blurry images [LPL⁺18]. The work that pioneered the MAP_k estimator for motion blur is by Levin et al. [LWDF09]. They also created a set of real motion blur kernels that remains a widely used test set for spatially uniform motion blur.

Outside of motion blur, there are several applications for Gaussian or defocus blur. Joshi et al. estimate composite motion, defocus and camera-specific blur kernels, to deblur real images [JSK08]. Yan and L. Shao classify and estimate Gaussian, defocus and parametric motion blur separately for image patches [YS16]. Lee et al. estimate defocus blur for every image pixel [LLCL19]. Long et al. estimate composite Gaussian, Airy and Lorentzian blur according to an astronomical imaging model [LSW⁺19].

Another application where Gaussian blur comes up is blind super-resolution. Michaeli and Irani estimate Gaussian blur prior to super-resolution, and point out the connection to MAP_k estimation [MI13]. Gu et al. iteratively improve high resolution image and Gaussian blur estimates [GLZD19]. W. Shao et al. present a recent classical approach for super-resolution and blind deblurring, for e.g. complex motion and elliptical Gaussian blur [SGW⁺19]. A specific context of super-resolution is THz imaging, where the optical sensors yield an unconventional degradation model and corresponding blur [LWY⁺19, LHZ⁺20].

To summarise, among these methods, many different blur types are estimated. Several methods estimate composite blur types and/or spatially variant blur in order to handle natural images. Outside of complex motion blur, the approaches favor MAP_k estimators.

Among deep learning approaches, there are only a few that can handle complex motion blur. There are several works that estimate only the sharp image x , namely MAP_x [HKZŠ15, NHKML17, TGS⁺18, ZDLK19, YLC20]. However these are only optimized for simple motion blurs, under the spatially variant blur model. The High-noise deblurring (HNDB) method by Anger et al. uses a deep denoising network to filter the blurry image, but has a classical $\text{MAP}_{x,k}$ estimator at its core [ADF19]. Ren et al. create the SelfDeblur method to do $\text{MAP}_{x,k}$ estimation in a generative approach [RZW⁺20]. Similar to a classical $\text{MAP}_{x,k}$ approach, they iteratively update networks that generate sharp image and kernel estimates, so as to match the blurry image after convolution. This iterative optimization is done from scratch for every blurry image. The NDeblur method by Chakrabarti is trained to predict blur kernels, or more accurately, coefficients of a Wiener-like inverse filter [Cha16]. The loss is computed on the deblurred image patches, so both sharp images and kernels are estimated, i.e. $\text{MAP}_{x,k}$ estimation is done. To obtain a global kernel estimate from patches, NDeblur uses a standard algorithm to infer the kernel based on the blurry image and stitched-together sharp image. This standard algorithm only uses strong edges, making the global kernel estimate robust to outliers. Both SelfDeblur and NDeblur are relatively shallow architectures, with up to 10 and 7 weighted layers respectively.

For non-blind deconvolution, there are many recent methods that apply deep learning [AKB⁺20, LTME19, GZS⁺20, RZM⁺18, DRS20, NJ20].

Chapter 4

Methodology

In the last few years, deep learning methods have been successfully applied to blind deblurring. Much like the trend for other inverse problems, advancements have not been as substantial as e.g. for image classification [MJU17]. For e.g. blind deblurring, decades of research have been invested into traditional algorithms, so outperforming them is not easy. Deep blind deblurring has been successfully applied to a number of deblurring tasks. For complex motion blurs, i.e. those that are long and have sudden movements, deep blind deblurring has found few successful applications so far. In this thesis, I propose a novel blind deblurring formulation to handle challenging blur types such as complex motion blur, using a MAP_k estimator. This type of maximum a posteriori estimator only outputs blur kernels, that can then be used with a non-blind deconvolution algorithm to obtain sharp images. Even since traditional blind motion deblurring, MAP_k estimation holds a solid theoretical justification, but $\text{MAP}_{x,k}$ estimation typically holds state-of-the-art results. Although deep MAP_k estimators have been successfully applied to other blur types, this is the first work to do so for complex motion blur, to the best of my knowledge. Moreover even the existing $\text{MAP}_{x,k}$ network architectures for complex motion blur are relatively shallow, with up to 10 weighted layers. To show that the proposed algorithm can be general and apply to different blur types, such as those found in natural images, it is trained for both motion and Gaussian blur. The training set can also be modified as needed, to suit desired blur types. The non-blind deconvolution method that is used for deblurring is freely interchangeable. The proposed method can be seen as a baseline for a deep MAP_k implementation. Because the kernel prediction network is a simple fully convolutional network, it is easy to adapt and optimize.

In this chapter the essential components are covered, namely the choice of estimator, training set, loss function and architecture. Then a final section covers how to go from kernels to deblurred images.

4.1 Selecting a MAP Estimator

Out of the three possibilities for the MAP estimator, the selection of MAP_k is justified as follows. One reason why MAP_x is not selected is because it does not leverage what we know from classical blind deblurring, namely the blur image formation model eq. (1.1). The motivation for MAP_k over $\text{MAP}_{x,k}$

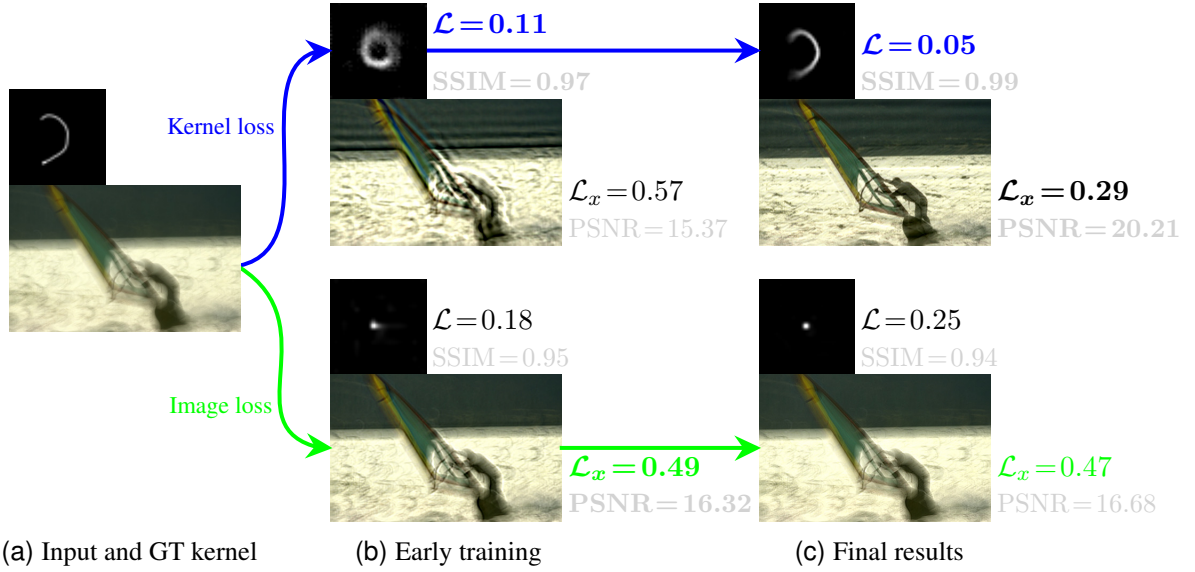


Figure 4.1. Deep Learning $\text{MAP}_{x,k}$ vs. MAP_k : Column (a) shows a ground truth kernel and a corresponding blurry image (i.e. the network input). The **upper path** shows training with a kernel-based loss function \mathcal{L} , i.e. a MAP_k network, while the **lower path** shows training with an image-based loss function \mathcal{L}_x , i.e. a $\text{MAP}_{x,k}$ network. See section 4.4 for their definition. Column (b) shows intermediate kernel and deblurred image predictions of the network, after an early 20,000 iterations. Column (c) shows the predictions after training has converged (70,000 iterations). The primary values to look out for are the kernel loss for the upper path and image loss for the lower path, as these are what is optimized for. Note how the non-optimized losses (e.g. \mathcal{L}_x for the upper path) change alongside. The SSIM and PSNR metrics are secondary, just to make sure the losses align with more objective measurements. Best scores in each column are marked in bold font.

is given by findings from estimation and signal processing theory: In order to approach true solutions, MAP estimation requires enough measurements. Levin et al. applied this finding to the problem of blind deblurring [LWDF09]. For $\text{MAP}_{x,k}$ estimation, the ratio between measurements (i.e. size of the blurry image) and unknowns (i.e. size of the sharp image and kernel) cannot be improved. For MAP_k estimation, only the kernel is estimated, so the ratio can be improved by larger blurry images. Despite this, classical MAP_k implementations in blind deblurring did not overcome practical shortcomings. But deep learning introduces us to new perspectives on MAP_k estimation.

A toy example in fig. 4.1 shows that the findings of Levin et al. and others can also be found in the field of Deep Learning. As mentioned in section 2.2, they found that simple $\text{MAP}_{x,k}$ favors the no-blur solution and that optimization is prone to get stuck in local minima, regardless of the image prior.

Note how the image quality in the figure starts off poor with the kernel-based loss, but ultimately ends up superior. The circular kernel shape approximately fits a number of different ground truth shapes, and can later be refined by culling it into a C-shape with the correct orientation. This is possible because the fierce deconvolution artifacts due to kernel error are not penalized. On the other hand, the image-based loss does penalize those artifacts, and thus cannot afford to make the risky—

yet meaningful—intermediate kernel predictions. Consequently image-based optimization ends up getting stuck at bad local minima, that represent essentially just generic image sharpening filters. The bad local minima here are not precisely the no-blur solution. But this is still approximately true, given that they yield small σ Gaussian kernels, that are certainly closer to the no-blur than the true solutions. This example also illustrates a challenge of $\text{MAP}_{x,k}$ optimization: Small kernel prediction errors are greatly amplified through the process of deconvolution.

To summarise, estimation theory tells us that MAP_k estimators are preferable. This intuition has also been validated empirically for the proposed method. For classical estimation, $\text{MAP}_{x,k}$ is preferable in practice, but often relying on optimization techniques that are not well-understood or easily applied to deep learning.

4.2 Kernel Prediction Network by Solving MAP_k

Equation (2.3) shows the maximum a posteriori formulation for a kernel, i.e. MAP_k . The following optimization procedure creates a network $\mathcal{N}_\theta(y)$ with trained parameters θ . I propose that this network learns the inverse problem of estimating the ground truth kernel \hat{k} from a blurry image \hat{y} , i.e.

$$\arg \max_{\theta} p(k|y) = \arg \min_{\theta} \mathbb{E}_{(k,y) \sim p_{\text{data}}} \left[\mathcal{L}(\mathcal{N}_\theta(\hat{y}_n), \hat{k}_n) + \alpha \Omega(\theta) \right], \quad (4.1)$$

given a set of training pairs $\{(\hat{k}_n, \hat{y}_n)\}_{n=1}^N$ sampled from the data-generating distribution p_{data} . In other words, the learned parameters θ are part of a parametric function \mathcal{N}_θ that solves the inverse problem. The loss function \mathcal{L} measures the error between the network’s prediction and the ground truth kernel. Moreover \mathcal{L} may be accompanied by a regularization term Ω on the network parameters, adjusted by a scalar value $\alpha \in [0, \infty]$. The regularization term is shown here for consistency with eq. (2.2), but is conventionally implemented by the optimizer (i.e. minimizer) in the form of weight decay, and thus will be omitted from here on out.¹ Minimizing the expected value \mathbb{E} of the prediction error results in a set of parameters θ that maximize $p(k|y)$.

For a more detailed argumentation on why eq. (4.1) implements eq. (2.3), see appendix A. In short, the network has to be able to somehow “pull apart” the kernel and the sharp image, for which some knowledge about kernels and images is required, i.e. $p(x)$ and $p(k)$ are learned. General knowledge about kernels and images (that does not pertain to a given y) is not enough to solve the task. So learning some concept of how kernels are imprinted on images is necessary. This is what the image formation prior $p(y|x,k)$ represents [PF15].²

Note how this implementation of MAP_k differs from a classical implementation. This implementation avoids sharp image estimates, which is the original intent of MAP_k estimation. In the classical setting, the backbone for estimation is the convolution operator, which in practice relates the kernel

¹Weight decay in an optimizer usually corresponds to ℓ_2 regularization of θ . See [LH17] for details.

²Commonly this imprinting is convolution, but as this is not specifically encoded, a network implementing eq. (4.1) can in principle learn more general imprinting, e.g. strided kernel addition.

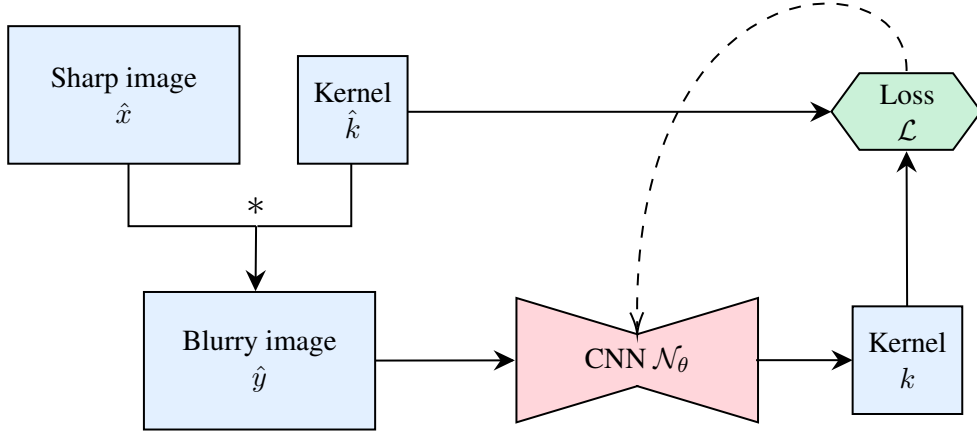


Figure 4.2. **Deep MAP_k estimation:** This shows a training step of the proposed kernel prediction network. First a sharp base image and generated kernel are convolved to create a blurry image. The kernel prediction network, which is a CNN architecture with a bottleneck, predicts the kernel for the blurry image. The loss is computed directly on the kernels, as opposed to e.g. adding a deblurring step and comparing to the sharp image. The loss is then used directly to update the network parameters θ via backpropagation. During inference, only the bottom three components are present.

estimate k to some variable x . Following the integration of eq. (2.3), the variable x ideally represents all possible sharp images, but in practice only a few estimates can be covered. The common VB approximation for MAP_k is thus equivalent to a $\text{MAP}_{x,k}$ formulation [PF15]. By eliminating the convolution operation between x and k in the proposed kernel estimation, the approach becomes more distinct from $\text{MAP}_{x,k}$. The integration over all possible x is still embedded into the network, as during training many images are observed, and in some sense deconvolved. During inference however, this learned knowledge over all observed images is then available without having to rely on specific estimates of x . In this sense the proposed method is closer to an ideal MAP_k estimator than the classical VB formulation. But unlike MAP_x estimators, the convolution image formation model is still leveraged explicitly, but only *after* kernel estimation is complete.

Lastly, in order to get an algorithm to optimize eq. (4.1), an approximation has to be made [GBC16]. Since the data generating distribution p_{data} is not available, an empirical distribution \hat{p}_{data} is selected as the training set. Consequently, the empirical risk is minimized by iterating over the finite and mini-batch-approximated training set, resulting in trained network parameters θ

$$\arg \max_{\theta} p(k|y) = \arg \min_{\theta} \frac{1}{N} \sum_{n=1}^N \mathcal{L}(\mathcal{N}_{\theta}(\hat{y}_n), \hat{k}_n). \quad (4.2)$$

Figure 4.2 illustrates a MAP_k training step, from data generation to loss computation, which subsequently is backpropagated to adjust the network parameters θ .

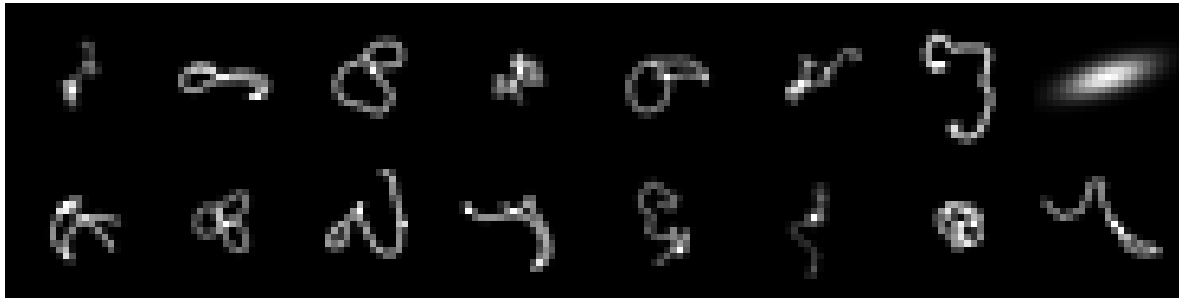


Figure 4.3. **Examples of synthesized kernels:** For training, a mixture of motion and Gaussian blur kernels of size 23×23 is used. The majority of kernels are motion kernels with an unstable continuity, meaning the line path is prone to abrupt directional changes. In addition, a small fraction of kernels are more continuous motion kernels, or Gaussian kernels with two axes and random rotation. In the aforementioned order, they follow a ratio of 15:2:1, guided by how difficult they are to learn.

4.3 Training Set

This task is fairly straightforward, but as (to the best of my knowledge) there are no public (\hat{k}, \hat{y}) training sets for motion blur kernel estimation, some considerations that have been taken will be outlined here. As is usually the case with inverse problems in imaging, given a natural blurry image y , there is no reliable way to determine the exact kernel \hat{k} that was used to create it (except for calibration images). But since the forward model is known, it is easy to start with sharp images x , then blur them with some kernel \hat{k} to obtain y . Thus there are conceptually two datasets to consider, one being a set of sharp images $\{\hat{x}_i\}_{i=1}^I$, and the other being a set of kernels $\{\hat{k}_j\}_{j=1}^J$. Each kernel can be paired up randomly with images $N=J$ times to create the training set $\{(\hat{k}_n, \hat{y}_n)\}_{n=1}^N$, where $\hat{y}_n = \hat{x}_l * \hat{k}_n + \eta$ and $l = n \bmod I$. Alternatively, e.g. each kernel could be paired up with every sharp image, but initial experiments showed that benefits of a large sharp image set are negligible, i.e. a good dataset has $J \gg I$ and should contain as many unique kernels as possible.

This also works out well for dataset creation. Images are hard to synthesize, but it is easy enough to collect a few. Meanwhile kernels are difficult to collect, but easy to synthesize. In this approach, all kernels are synthesized, meaning no training kernel is ever used twice. Even so, it is not possible to train a network on every possible kernel type, or even just e.g. every type of motion kernel. Thus any network trained on a dataset will always do well on kernels similar to those it has seen, but worse on kernels that are very unfamiliar. This is a blessing and a curse, as it allows to specialize well on kernel types for a given task, but makes it hard to design a dataset that generalizes well to any kernel that could be encountered in the real world.

For the image set, the BSDS500 dataset is used [AMFM11]. They are also augmented with horizontal flips, affine transformations and color jitter. A mixture of Gaussian and motion kernels is used for the kernel set, as shown in fig. 4.3. This selection is aimed at being able to handle some of the common, yet difficult kernels that may be found in the real world. It also enables comparison to other works. But the selection can also be substituted to better fit a particular task. The two-dimensional

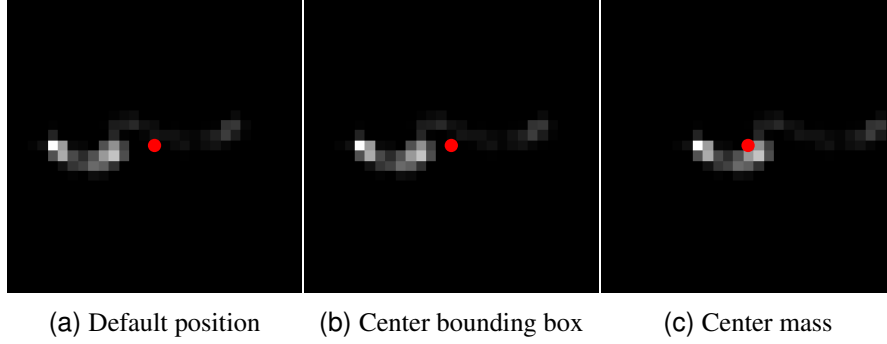


Figure 4.4. **Centering kernels:** Shown is a kernel from the Levin dataset (rotated by 135 degrees). Kernel (a) is without any modifications. Kernel (b) is centered by its bounding box, and (c) by center of mass. The red dot indicates the center of each image.

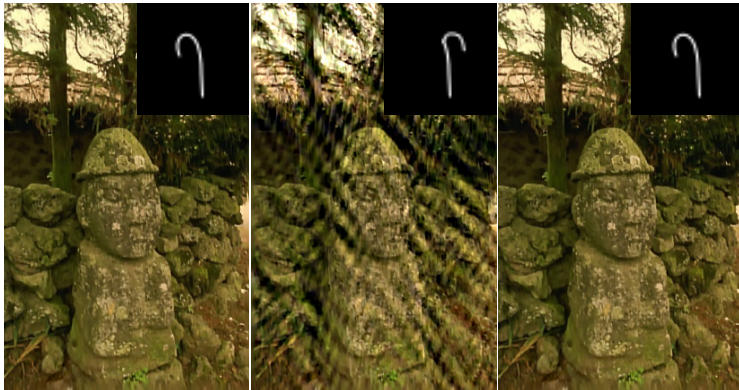
rotated Gaussian kernels $\hat{k}_\sigma(\sigma_x, \sigma_y, \phi)$ are randomly sampled from a gamma distribution $\sigma \sim \Gamma(\alpha, \beta)$ with concentration $\alpha = 2.5$ and rate $\beta = 1$, and rotation from a uniform distribution $\phi \sim U(0, 360)$. Motion kernels $\hat{k}_m(M, L_{max}, p_s)$ are synthesized based on the code of Kupyn et al., which follows that of Boracchi and Foi [KBM⁺18, BF12]. Primarily the parameters are set to $M = 100$, $L_{max} = 40$ and $p_s = 0.1$. See section 5.4 for information on how these parameters were chosen. In short, they are designed to be similar to the kernels by Levin et al., which are known real measured motion blur kernels [LWDF09]. All motion blur kernels used in this thesis are centered by mass, as shown in fig. 4.4. Presuming that coarse scale features are learned first during training (i.e. the thick part), fig. 4.4a and fig. 4.4b are not ideal because whenever the thick part is predicted and centered by the network, it hardly overlaps with its position in the ground truth. Note however, that fig. 4.4b allows for the smallest kernel support size.

4.4 Loss Function

As it turns out, finding an appropriate loss function is not just a means of improving the score (as it may be for tasks such as classification or denoising), but integral to a successful general kernel prediction network, i.e. one that can estimate non-parametric kernels. For instance, the standard MSE loss will not work for any motion kernels more difficult than lines and simple curves. The reason being, phase estimation of blur kernels in natural images is generally underdetermined, meaning the position of the kernel in the kernel support is not unique. For instance, if the kernel was a straight horizontal line, the blurry image is the same whether the sensor moved from left to right or vice versa.³ The phase of the kernel is thus usually constrained, e.g. by center of mass.

While centering the ground truth kernels is enough for the network to learn to approximately center its predictions, there is still some ambiguity to the exact placement. The problem is illustrated in the toy example of fig. 4.5. Even if the shape of a predicted kernel was perfect, if its position is

³Counterexamples are e.g. if the optical sensor position in space was recorded, or using prior knowledge on temporal movements in the scene.



	(a) GT kernel	(b) Bad kernel	(c) Shifted GT kernel
Deblur PSNR	25.757	15.799	20.314
Kernel ℓ_1	0	0.762	1.306
Kernel 1-SSIM _{0.5}	0	0.024	0.031
Kernel 1-SSIM _{1.5}	0	0.022	0.018
Kernel 1-SSIM ₃	0	0.017	0.015
Kernel DISTS	0	0.102	0.053

Figure 4.5. **Shift-tolerant loss function:** A blurry image is deblurred using different kernels: (a) the ground truth kernel, (b) a kernel with inaccurate shape (yet largely aligning with the ground truth), and (c) the ground truth kernel shifted to the right by one pixel (effecting the same shift in the deblurred image). Despite using PSNR, a metric that expects precise alignment, image scores still align with human perception in this example—that is, shifted kernel and image (c) look much closer to the ground truth than (b). This is because despite the majority of the kernel (b) aligning with the ground truth, the overall shape error resulted in glaring ringing artifacts.

When evaluating kernels, such as during training of a kernel prediction network, it is desirable to maintain this score alignment, i.e. an inaccurate shape should result in a *larger* error than a (slightly) inaccurate position. Loss functions that lack any shift-invariance, such as ℓ_p -norms do not meet this desire, and would thus favor the kernel (b). SSIM aligns correctly, as its underlying blur operation adds tolerance for small shifts, provided that its blur parameter σ is large enough. Perceptual losses like DISTS come with some shift-invariance that is inherent to CNNs.

off by just one pixel, conventional losses treat it as a very poor prediction. In practice, this means the network ends up being focused almost exclusively on getting the position correct (which has little impact on image quality, especially early on), instead of getting the shape correct. In order to make prediction of difficult kernels possible, the loss function used has to be tolerant to at least small shift errors.

One method to get full shift-invariance for network predictions is to have kernels be in the frequency domain (as opposed to spatial domain) [Cha16]. This approach naturally arises where (non-blind) deconvolution is part of an algorithm, because it is commonly done in frequency domain, via use of (fast) Fourier transform (FFT) on inputs y and k . This is not a necessity though, and as Levin et

al. showed, deconvolution in spatial domain also has its appeals (e.g. not having to deal with boundary issues) [LFD07b].

Previous works have represented motion kernels in both spatial and frequency domain, however only via shallow and largely linear architectures [Cha16, RZW⁺20]. In this thesis, kernel estimation is done in the spatial domain, with the idea that this is better suited for a deep convolutional network architecture (presuming a suitable loss function). Initial empirical results confirmed that prediction in two-dimensional frequency space is possible as well, but decidedly lacked in generalization.

Among a range of publicly available implementations for image quality metrics, those suitable as kernel loss functions were empirically determined [KZP19]. Future work can investigate loss functions designed for this task specifically, but for now the following is used:

$$\mathcal{L}(k, \hat{k}) = \sum_{\sigma \in \Sigma} \text{SSIM}_{\sigma}(k, \hat{k}) + \lambda \text{DISTS}(k, \hat{k}), \quad (4.3)$$

where $\Sigma = \{0.5, 1.5, 3.0\}$ and $\lambda = 0.6$ are selected. A single SSIM loss is enough to get the desired shift-tolerance and can provide acceptable results, but combining blur σ of different sizes better penalizes errors on smaller and larger scales. This is a similar idea to MS-SSIM, but the small kernel support size prevents its usage. The supplemental DISTS loss helps to make sharper kernel predictions, while still being shift-tolerant, which are both properties of perceptual loss functions [DMWS20].

4.5 Network Architecture

In the pursuit of creating a kernel prediction network, this thesis highlights four choices of that process. The first one was between applying a loss function to a kernel or an image, where the choice of kernel was made, illustrated in section 4.1. The second one was between working in spatial or frequency domain, where spatial domain was selected here, illustrated in section 4.4. The two remaining choices are within the architecture. Figure 4.6 breaks the proposed architecture down into two components, each corresponding to such a choice: a feature extraction network and an encoder-decoder network.

4.5.1 Feature Extraction Network

The main architectural task is to reshape the input that is generally a large image with variable aspect ratio, into a kernel that is generally a small square grayscale image. The first choice to consider is whether to use the full image or crop a small patch of it. Cropping the input into the desired output shape immediately solves the reshaping task, and also frees up a lot of the memory budget for training, (e.g. [Cha16]). The downside is that even if results of many input patches are averaged, the receptive field is limited to the size of the kernel support. Initial experiments with trainable crop architectures had suboptimal results (see appendix B). The proposed architecture opts to input full images, using a base network with pooling layers to approximately reshape the image features to the output kernel size.

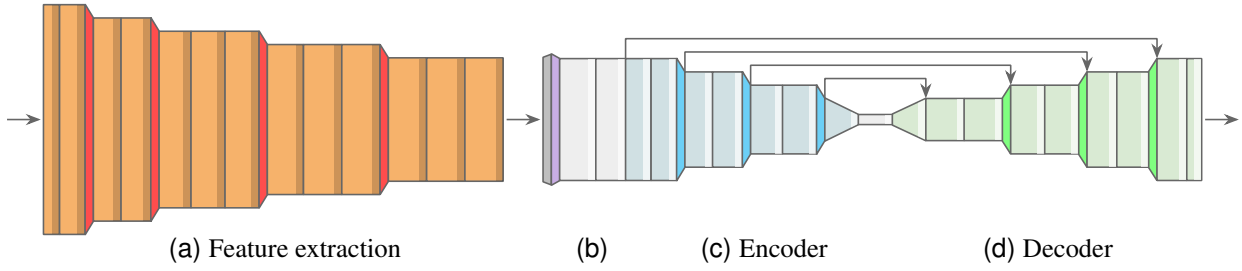


Figure 4.6. Kernel Prediction Network: The CNN architecture is comprised of a standard feature extraction network (a), such as VGG-16 (with the last pooling layer and linear layers removed), followed by a U-Net encoder-decoder network. First an adapter block (b) shapes features into a fixed size. The encoder then condenses the features into some hidden kernel parameters, that the decoder can use to construct a kernel output. The small bottleneck increases the receptive field, while the (concatenation) skip connections from (c) to (d) maintain availability of higher resolution features for reconstruction. For details refer to fig. 5.1.

The base network is labeled feature extraction network, as this is what the selected VGG architecture was designed for. Even though its task of image classification is different, the initial goals overlap, i.e. it also pools images into a small feature size. Although there is no semantic cutoff between feature extraction and encoder-decoder network, e.g. fig. 4.6a may already encode kernels, there is still a reason to designate this network as a separate component. It is modular and pretrained, meaning it can be substituted by other existing architectures and parameters, such as ResNet. Here pretrained parameters are used as a sensible initialization that speeds up training.

The exact version of the architecture/parameters is a VGG-16 variant with antialiased pooling (with blur filter size 2), that increase accuracy and robustness over the original VGG-16 [Zha19]. It is then trimmed for feature extraction by removing the final pooling layers and linear classification layers.

4.5.2 Encoder-Decoder Network

The second network has the role of turning image features into blur kernels. After having the features down to a size close to the desired output size, the remaining choice is whether to further reduce the size, and follow up by a later upscaling. As indicated by the bottleneck in the encoder-decoder network in fig. 4.6, this is the selected approach for the proposed architecture. Having a small bottleneck, here all the way down to a spatial size of 1×1 , causes the kernels to be encoded into (hidden) parameters. For instance, if the network had to merely estimate uniform Gaussian blur, only a single scalar σ would be needed to encode the kernel. The downside of this approach is that the learned encoding restricts diversity of kernels that can be decoded. For a toy example, if the sharp image was a black background with a white point, this would simply draw the blur kernel in the blurry image. In this case, encoding and decoding of that kernel would typically lose details, which an architecture without downsampling could avoid. The upside of encoding is that separation of the image background from

the kernel becomes simple, as e.g. for motion blur, the decoder figuratively just has to draw a white stroke over black background. This makes the parametric approach, such as an encoder-decoder network, the sensible choice for a first kernel prediction network.

One simple method to encode kernels is to flatten the two dimensional features into one (or more) linear layers, then reshape the output back into a two dimensional kernel [Cha16, LSW⁺19, RZW⁺20]. The problem with this approach is that the spatial structure of the features is discarded. To alleviate this, this thesis proposes to adapt a U-Net structure, an established CNN archetype originally developed for biomedical image segmentation [RFB15]. The primary path gradually pools features down into a small size, thereby increasing the receptive field and capturing context information, then scales the features back up. In addition, shortcut paths skip pooling layers, making higher resolution features available for precise reconstruction.

4.6 Non-blind Deconvolution

This section deals with going from the kernels, i.e. the network output, to deblurred/sharp images. Ideally comparison to other kernel estimation works, i.e. MAP_k or $\text{MAP}_{x,k}$ ⁴, is done directly on the kernels. A more accurate blur kernel estimate is the foundation towards restoring a sharp image, when using the convolution image formation model. Thus it is desirable to compare using this foundation where possible, rather than including the algorithms that do the actual deblurring, as they differ from method to method. In general however, comparison has to be done on deblurred images, such as when comparing to MAP_x estimation, or $\text{MAP}_{x,k}$ estimation where kernel estimates are specific to that method. For this reason, comparisons to other works are done on both deblurred images and kernels, despite the caveat that the chosen deblurring method is not a fixed part of this approach and can freely be substituted by other methods. As a whole, the algorithm of this thesis can be seen as blind deblurring, since a kernel is not known at the beginning. But broken down, the main algorithm is a kernel prediction network, followed by a non-blind deconvolution where kernels are known.

To do the deblurring for the kernel prediction network, a brief survey over available classic non-blind deconvolution implementations will be done, as the first numerical experiment. The one that performs best with the proposed approach will then be selected for all experiments.

For the $\text{MAP}_{x,k}$ variant of the proposed method (used in fig. 4.1), a custom Wiener deconvolution \mathcal{D}_λ is used. This Wiener implementation is done in PyTorch, to create a layer that can be used for backpropagation. The kernel output of the network is then simply fed into this layer, along with the blurry input image, to create a deblurred output. The regularization parameter λ of this layer is also

⁴Although $\text{MAP}_{x,k}$ estimation by design already outputs a sharp image x , it is possible to discard it, and use only the estimated kernel k for a separate deblurring method to obtain an alternative sharp image \tilde{x} (e.g. [ADF19]). In other words, even for some $\text{MAP}_{x,k}$ estimators, the kernel can be viewed as the primary output.

trainable. This corresponds to the $\text{MAP}_{x,k}$ formulation

$$\arg \max_{\theta, \lambda} p(x, k | y) = \arg \min_{\theta, \lambda} \frac{1}{N} \sum_{n=1}^N \mathcal{L}_x(\mathcal{D}_\lambda(\hat{y}_n, \mathcal{N}_\theta(\hat{y}_n)), \hat{x}_n).$$

To simplify, we can consider the layer \mathcal{D}_λ to be part of the network \mathcal{N}_θ , to get

$$\arg \max_{\theta} p(x, k | y) = \arg \min_{\theta} \frac{1}{N} \sum_{n=1}^N \mathcal{L}_x(\mathcal{N}_\theta(\hat{y}_n), \hat{x}_n). \quad (4.4)$$

The loss \mathcal{L}_x is now applied to an image, so a different loss function is used

$$\mathcal{L}_x(x, \hat{x}) = \gamma \text{MS-SSIM}(x, \hat{x}) + \mu \text{LPIPS}(x, \hat{x}), \quad (4.5)$$

where $\gamma = \mu = 0.5$. The perceptual LPIPS loss works similarly to DISTS, but is designed for structural images [ZIE⁺18].

Chapter 5

Implementation

This chapter goes over some details that are not necessary to understand the idea of the proposed method, but necessary to reproduce it. It consists of the exact layers of the architecture, optimization procedure and preparation of training data. In general, all of the implementations here are done in PyTorch.

Some of the topics of this chapter could also be discussed and justified, however the distinction is as follows. The methodology chapter is about choices that I propose are a good way to solve deep kernel estimation: MAP estimator, training set essentials, shift-tolerant loss, and a feature extraction plus parametric network. Choices in this chapter are just one particular way of driving the proposed approach to completion. The hyperparameters here have been optimized in some fashion, but no claims about their optimality or effectiveness are made. Although these choices do affect the quality of the results, they will not be evaluated in order to keep the thesis concise.

5.1 Architecture Details

The architectural implementation details are shown in fig. 5.1. The only custom layer is SquarePad, which simply adds zero-padding, either vertically or horizontally, such that the output becomes square. For the BlurPool and PixelShuffle layers, see the respective papers [Zha19, SCH⁺16]. To give an example for upscaling and concatenation, take the convolutional layer before the first PixelShuffle. It outputs 512 channels at spatial size 3×3 , where four output pixels in one feature map are taken from one position of four lower resolution input channels. The output size is thus $(128, 6, 6)$, i.e. spatial resolution is doubled while channel counts are quartered. Then 128 channels from an earlier feature map are concatenated, to create the 256 channels for the next convolutional layer. To get the final output of odd feature size, the last row and column are dropped after the last convolutional layer.

Finally note that no batch normalization is used throughout the entire architecture, even though 30+ weighted layers are used. This is because it is found to not be necessary, at least at this depth. Training with batch normalization works fine, but is slower, and creates a problem when transitioning into testing (due to the change in image statistics). Although batch normalization has been stan-

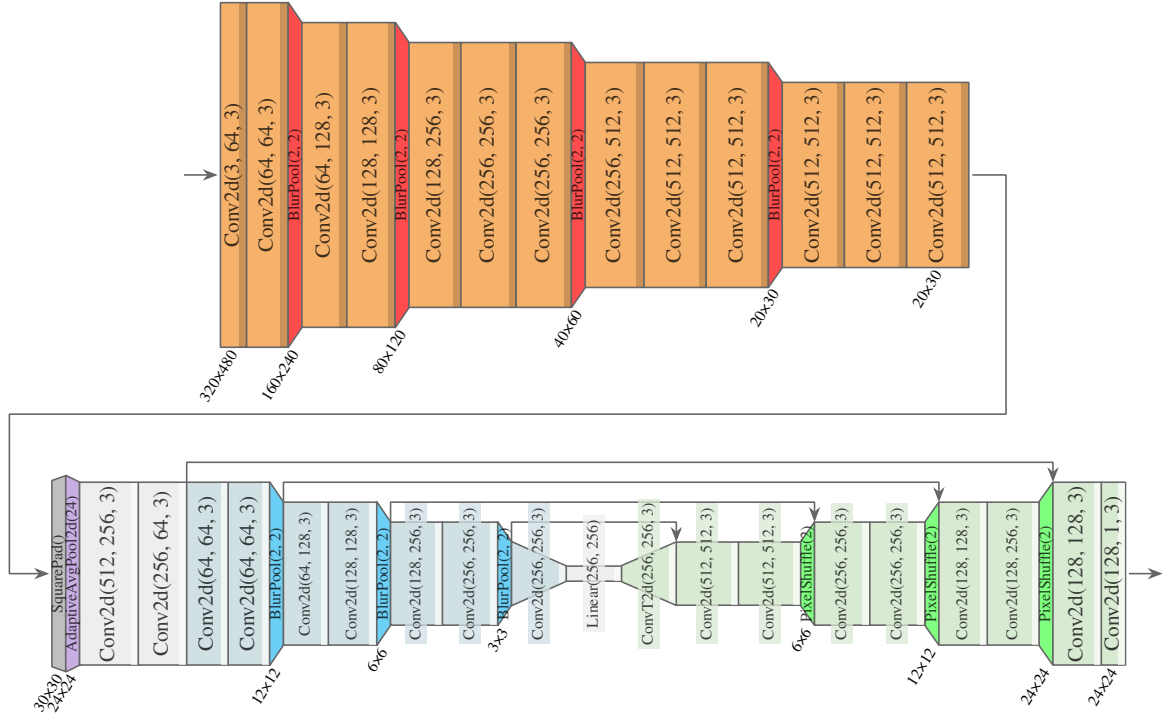


Figure 5.1. **Architecture implementation:** *Best viewed in the PDF version.* This shows the architecture of fig. 4.6 but with all the layer specifications. For weighted layers, the parameters are: input channels, output channels, and also kernel size if applicable. For BlurPool, the parameters are stride and filter size. For PixelShuffle it is upscaling factor. The bottom labels show output spatial feature size of the respective layers. Weighted layers are also followed by activation functions: ReLU (gray) or leaky ReLU with 0.01 slope (white). Note that the bottleneck has 1×1 spatial size, so a linear layer is used, followed by a transpose convolution to upscale back to 3×3 .

dard practice for a while, recent works have shown to be successful without it even for deep networks [BDSS21].

5.2 Optimization

This section covers how the kernel prediction network is optimized. The optimizer used is AdamW, which is just Adam with a fix to how weight decay is implemented [LH17]. A weight decay of 0.005 is used. As seen in fig. 5.2, the learning rate follows a one cycle schedule, that peaks at $\lambda = 1.3e-4$ ($= 0.00013$) and starts at $\lambda/7.5$ ($\approx 1.7e-5$). This schedule also inversely cycles momentum, which in the case of Adam are the β_1 and β_2 parameters, to reduce momentum at peak learning rate. The duration of training is 12,000 epochs at a batch size of 8. This is roughly one week of time on my machine with four CPU cores at 4.08 GHz and a GTX 1060 6GB for GPU. Using the training set of 480 base images per epoch (61 batches), this totals at 732,000 optimization steps (see section 5.3 for an explanation of the number of batches). Towards the last 500 epochs, stochastic weight averaging

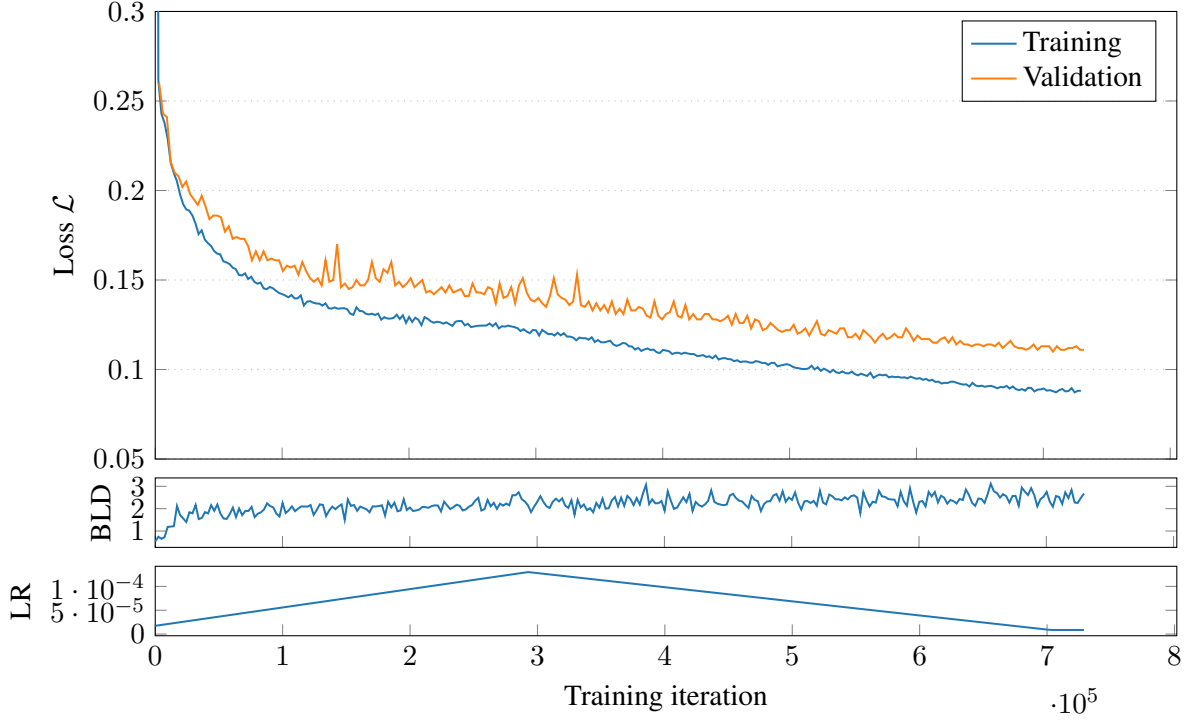


Figure 5.2. **Convergence of optimization:** Primarily the convergence of the loss \mathcal{L} (see eq. (4.3)) during training of the proposed kernel prediction network (with output size 23×23) is shown. Two complementary metrics are shown that can be used to get a better understanding of the loss behavior. The baseline difference (BLD) is a (proposed) metric that tracks the difference of a predicted kernel for a blurred image and the predicted kernel for a baseline input, that is just a black image $\mathbf{0}$. (Here, this metric is modified by taking the standard deviation and scaling by the size of the kernel support.) Ideally this metric is nonzero, otherwise the network is making constant predictions (that are irrespective of the input). Third, the learning rate (LR) is shown. Here a one-cycle schedule is used, that intentionally peaks at very high values where convergence is slow or impossible (but importantly does not explode). Note that for both visual clarity and due to technical limitations, the plotted values are heavily smoothed and subsampled, respectively.

(SWA) is used every 5 epochs at a learning rate of $\lambda/15$ ($\approx 8.7\text{e-}6$) [IPG⁺18].

5.3 Dataloader

So far the base image set, kernel set, and batch size was mentioned, but not how they are combined and augmented to create batches of blurred images. This section aims to close this gap, as it has some important details that are easy to overlook.

For data augmentation, horizontal flips (50%) and color jitter (brightness from 0.9 to 1.3, contrast from 0.9 to 1.4, saturation from 0.9 to 1.4, hue of ± 0.03) are easy to apply. But the affine transformation (rotation of ± 25 , shear of ± 8 , scale of ± 0.25) raises the question of where to apply it.

For instance, take a rotation transformation. It does not output a leveled/straight image. Instead it

leaves empty space when surrounding it by a leveled bounding box to sample it into an image again. This empty space has to be filled, typically by a single value like zeros. When applied after blurring, this altered image is no longer consistent with the kernel, which would have to be rotated as well. When applied before blurring, the output image does not lose synchronization with the kernel, but the fill value boundary now gets blurred. Conceptually this is as if the original image had four large, perfectly sharp edges, where it becomes easy to detect the blur kernel. This would be an advantage over natural images, which generally do not contain large predictable boxes of constant color. Initial results confirmed that such borders were the primary factor in network predictions (using attribution [STY17]). For simplicity the proposed approach does not attempt to keep kernels synchronized, and opts for augmenting base images.¹ To mitigate blurring of the borders, the same affine transformation that is applied to the base image is applied to a white image **1**. This becomes a rotated white image with black borders, and serves as a mask. The mask is multiplied with the blurred image to remove all blur lying outside of the border. The fill value bleeding inwards to the base image boundary still remains. But if desired, the mask can be made smaller, e.g. by an erosion operation, to mitigate the inward bleed as well (which sacrifices some of the base image). Note that validation or test performance is unaffected by data augmentation. The training-validation gap is later investigated in section 6.5.

Once images are augmented, they are collated into batches of 8, i.e. tensor shapes of $(8, 3, H, W)$ with 3 color channels, and some height H and width W . Note that the base images have different spatial dimensions, so random images cannot be collated natively. Instead of padding or cropping the images to a common size, which would either be inefficient or compromise the ability of the network to handle images of varying aspect ratios, the proposed algorithm implements a custom random image index sampler. Once a random sample has been drawn, this sampler only samples images that lie in the same group, which is defined by the image size here. This works well for this dataset because there are only two distinct sizes: a vertical and a horizontal orientation. Just like with standard sampling, the final non-full batches can either be dropped or accepted as smaller batches. The latter is chosen here, resulting in epochs of the aforementioned 61 batches. The mean and standard deviation of the blurry inputs are also normalized, which is standard practice in deep learning. Here normalization is implemented as the first network layer, which makes life easier, e.g. original images remain available for comparison to ground truth images and for a non-blind deconvolution layer.

The final noteworthy implementation choice is the ground truth kernel orientation *after* convolution. Here the orientation is flipped in both axes (i.e. rotated by 180 degrees) such that a point source in the base image traces out a blurred shape matching the ground truth kernel. This alignment is more natural for kernel prediction as the network has no need to flip the kernels that it observes in blurry images. For the choice of ground truth normalization, it is kept power-normalized (summing up to one), but range-normalized kernels are also conceivable.

¹For example, one challenge with keeping kernels synchronized is that they need to be resampled, which is not necessarily synchronized with the resampling of the blurred image.

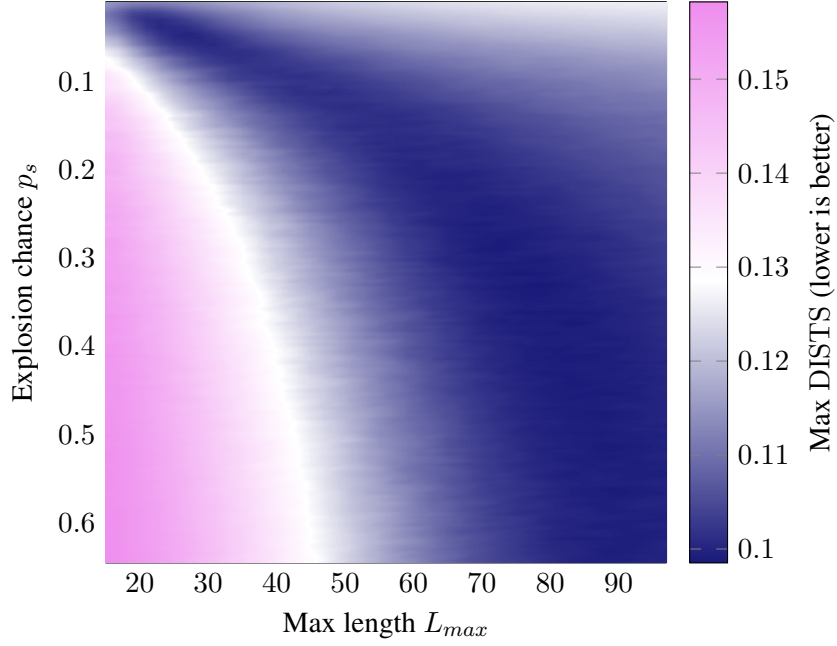


Figure 5.3. **Approximating real motion blur:** Two parameters of the kernel generation method L_{max} and p_s are shown (while the M parameter is fixed to 100). This height map is created by randomly sampling kernels at the different parameters, then measuring their similarity to the motion kernels by Levin et al. (the target set). To measure similarity, the DISTS metric is used (higher means less similar). For each random kernel, there is one similarity score per target set kernel. Instead of taking the average score, only the least similar target score is used. In other words, a reduction by max operation is performed.

5.4 Motion Blur Parameters

This section briefly details how the synthetic motion blur parameters $\hat{k}_m(M, L_{max}, p_s)$ are selected. Figure 5.3 illustrates the parameter search space of two parameters. For simplicity and due to time constraints, the simulation steps M are set to 100, which roughly effects that kernels can be long enough, but without looping too much.

In this parameter landscape, there are two mountains where generated kernels are very dissimilar to the desired motion blur. With a very high explosion chance (left side), kernels tend to not go far in any direction, essentially creating just a shape like a dot. With very low explosion chance (top side), kernels just trace out straight lines or curves. Between the mountains emerges a canyon, that is narrow in the top left corner, briefly extending slightly upwards and then becoming very wide and flat.

Brief preliminary experiments were done to find a good parameter setting. Empirically the narrow canyon is the most interesting region, offering the best deblurring scores in initial experiments. The large flat area, even though deep, actually does not yield optimal results, suggesting that this similarity landscape is only part of the picture. Note however that this choice of metric and reduction (i.e. DISTS and max) still provided better alignment with deblurring quality than SSIM or ℓ_1 metrics

and min/mean/median reductions or mixtures thereof. For the proposed training set, the selected parameters are $L_{max} = 40$ and $p_s = 0.1$, which is not the deepest part of the canyon, but yields good results empirically. Different parameter positions, as well as mixtures of them, did not outperform this simple choice. As a reminder, the kernels generated by these parameters are shown in fig. 4.3.

Lastly consider some validity concerns. A good thing is trying to match real motion blur as best as possible. A bad thing is trying to match a specific set of kernels to perform good a particular test set. Unfortunately in this case, both of these overlap, as the availability of real motion blur kernels is very sparse. I argue that the visual similarity to the kernels by Levin et al. is not to a concerning degree in this instance (given the random nature of motion blur, as well as limitations of the kernel generation method). But note that there may still be some bias for that particular dataset.

Chapter 6

Numerical Experiments

This chapter primarily deals with comparing the proposed method to other deep blind deblurring methods. To do this, the experimental setup is first outlined, including the choice of non-blind deconvolution and creation of test sets. Comparisons are then done on the three test sets created, each of which being focused on a different family of blur kernels. Next to assessing deblurred images, the experiments emphasize assessment of blur kernels, which are an important (yet sometimes overlooked) factor in deblurring performance. Lastly characteristics and flaws of the proposed method are examined.

6.1 Experimental Setup

To start off, some test sets are created, which the first subsection deals with. For two reasons the next subsection compares some non-blind deconvolution methods. First out of necessity, since some method needs to be selected. Secondly to get a sense of how much lifting for the proposed method is done by the network vs. the non-blind deconvolution method. Finally the third subsection contains an overview for all the blind deblurring algorithms that are evaluated here, and describes how they are parametrized and modified.

For image evaluations, first the standard PSNR metric is used. Second, the MS-SSIM metric is used. This metric is less common, but Yang et al. find that it better correlates with visual perception than PSNR or SSIM [YMY14]. There are no standard metric for kernels, so a selection of three different metrics is offered.

6.1.1 Test Sets

The goal of the numerical experiments is to evaluate both blur kernel estimation and blind deblurring performance of the proposed method. Although most blind deblurring works are focused on motion blur, one interest here is to create a method that can handle general blur. Furthermore even if the focus were on just motion blur, due to the nature of the deep learning algorithm, a good performance on one

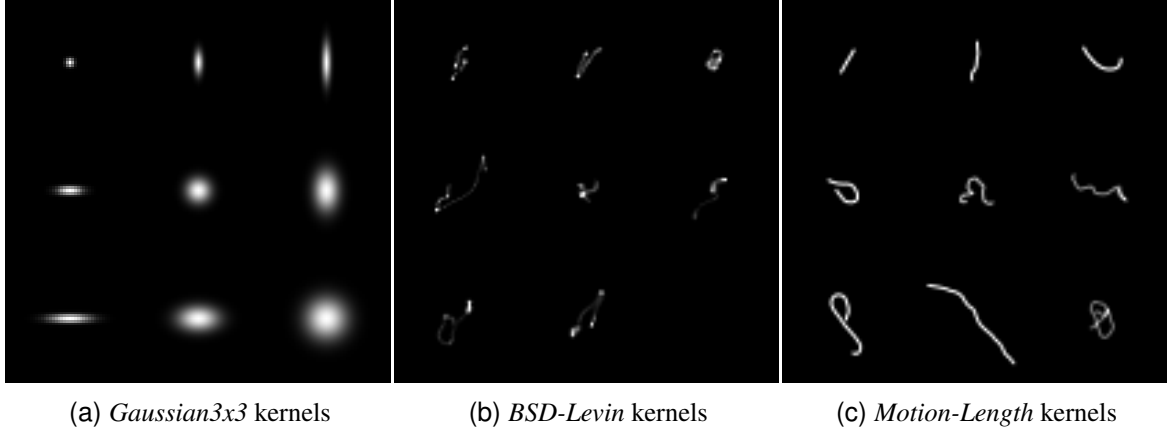


Figure 6.1. Example kernels of created test sets: Set (a) has $3 \times 3 = 9$ Gaussian blur kernels, that are combinations of three blur sizes in two axes $\sigma_x, \sigma_y \in \{1, 3, 5\}$. Set (b) has the 8 real measured motion blur kernels from Levin et al. [LWDF09]. The shown kernels (a) and (b) are all that are used for their respective datasets, each being convolved with all base images, and only the rotation of the kernels changes. Set (c) has synthetic motion blur kernels in 9 different lengths $L_{max} \in [10, 50]$, where again 9 kernels are convolved with each base image, but here each base image gets a unique set of such kernels. In (c) the length increases in order left to right, then top to bottom. Note that the other parameters M and p_s are also varied alongside L_{max} , such that they are neither too straight nor too self-intersecting/small. Specifically they are set $M = 2 \cdot L_{max}$ and $p_s = 0.001 \cdot L_{max}$. Motion kernels are centered as shown in fig. 4.4. The base images used with these kernels are the same for each test set, 20 images from the BSDS500 dataset (also part of the BSD100 set) [AMFM11].

motion blur dataset does not guarantee good performance on any motion blur dataset, or motion blur in the wild. This issue can exist in the classical deblurring setting, but definitely has to be considered for learning-based approaches.

Consequently multiple test sets are used in the following experiments, as shown in fig. 6.1. These are created because existing datasets are either unfit, or at least hard to find, especially with real motion blur kernels in mind.¹ The images blurred with each kernel set are 20 images from the BSDS500 dataset [AMFM11]. The first test set has Gaussian blur kernels. For each of the 20 base images, the set of 9 kernels is first rotated and then convolved, for a total of 180 images. Additive white Gaussian noise (AWGN) is added to the blurred images, parametrized with a signal-to-noise ratio (SNR) of 70. This is a *very* low amount of noise, because large Gaussian blur estimation is more sensitive to noise. The second set follows this approach, but using the 8 measured motion blur kernels by Levin et al., and with 35 SNR of noise, for a total of 160 images. Although the number of base kernels is still low, at least they are used in 20 different rotations. Note that AWGN noise is commonly parametrized by its standard deviation σ , and the amount used here translates to about 1.1%. The last test set uses synthetic motion blur, in 9 varying lengths/difficulties. Unlike before, there is no kernel set for every

¹One candidate was the test set by Sun et al. [SCWH13], that uses the 8 real motion blur kernels by Levin et al., and convolves each with 80 base images (the original set consists of only 4 base images). Unfortunately it only contains grayscale images, and 8 kernels.

base image. Instead, each base image gets a unique set of 9 kernels synthesized at different lengths. The added noise remains AWGN at 35 SNR. So in total there are again 180 blurred images, with 180 unique kernels.

One thing to keep in mind for the experimental results is that the synthetic kernels were generated with the same method that was used to create the training set here. Although the exact parameters differ for both, and every kernel synthesized is unique, this is still a subtle advantage for the proposed method. In fairness, some other methods use the same code for kernel generation of their training sets as well. In regards to the noise levels, the training set uses 40 SNR, which is a lower amount compared to the motion blur test sets. As a final side note, the base images used for the test sets are natural images, and as such sometimes have areas that are out of focus. To be very precise, the blurred images can be seen as having somewhat spatially varying blur, where the true blur kernels would be a mixture of motion and defocus blur. Here it is presumed that this deviation does not significantly affect scores, and in any case a higher ground truth than inevitable natural blurs is not available.

6.1.2 Non-blind Deconvolution

In order for the proposed kernel prediction network to be able to deblur, a non-blind deconvolution algorithm has to be used. This choice is made among a set of available implementations, and is empirically selected based on the results of fig. 6.2. This is necessary because the primary way of comparison to other works is via deblurred images, rather than predicted kernels. Furthermore this experiment gives a sense of how much performance can be attributed to the choice of non-blind deconvolution method.

To better understand fig. 6.2, first note the representativeness of the selection of methods. Most of the methods are well-established classical methods, that have been explored for decades. Wiener-Hunt deconvolution for example has been in use for over 50 years [Hun71]. Since then (up until recent years), advancements are moderate, with about 1.5 dB increase in PSNR for ground truth kernels, and even less for the selection of estimated kernels here. As such, even though not every single classical method is shown here, these results can be seen as fairly representative of what is possible with classical methods. On the flip side, the recent deep learning methods look very promising. These are still actively being researched, but even just the one method shown here pushes the boundary for ground truth kernels by 2 dB PSNR.

The curves in fig. 6.2 show the scores that can be obtained by varying the primary parameter of each method. Any other parameters, both intrinsic and extrinsic were optimized for best results. Intrinsic parameters are the number of iterations of matrix inversion (ℓ_1 and sparse prior), or arguably the choice of noise model (Total Variation prior). Extrinsic parameters are those that are outside of the methods, namely the padding that is applied to (and later removed from) the images. This is relevant for methods operating in the frequency domain, where the image boundary needs special consideration in order to mitigate ringing artifacts originating from the boundary [LFDF07a, KRS17]. Thus every method shown here was extended by an optimized padding mode and amount. This is

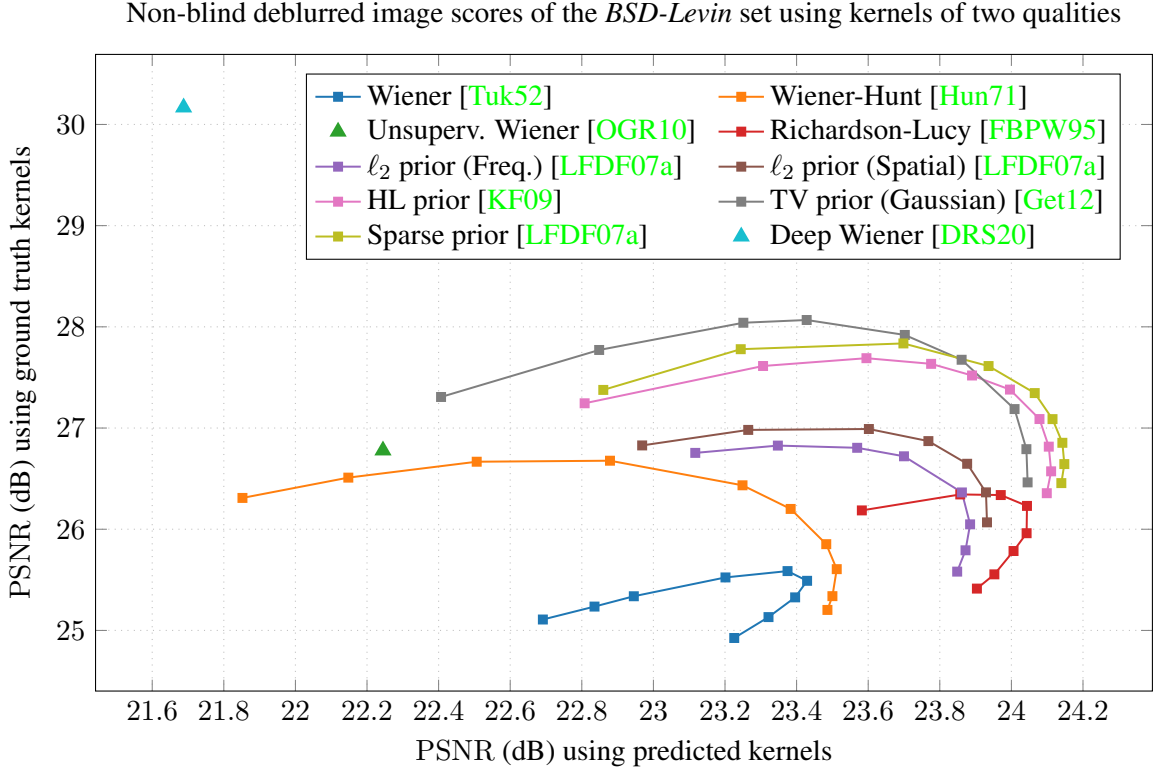


Figure 6.2. **Non-blind deconvolution method comparison:** Each marker represents the averaged results of two dataset evaluations—one for the Y-axis using ground truth kernels for deblurring, one for the X-axis using those from the Kernel Prediction Network. The markers within a method are obtained by varying the regularization parameter λ (or iterations for RL), where only the most interesting range is shown for better clarity. Conversely if a method does not have a parameter, it is represented by a single marker. Results near the top right are best, meaning average deblurring quality is good for perfect kernels, as well as estimated kernels with some errors. Generally regularization offers a trade-off, allowing to increase tolerance to estimation error at the cost of peak quality. Note that spatial domain methods ℓ_2 and sparse prior have an additional parameter for the number of iterations to compute matrix inversion, that have been set to 50 and 15 respectively.

either no padding, replication padding (i.e. repeating the boundary value), or edgetaper padding. For the latter case, I implemented a version (for Python), that in itself has several parameters that were all optimized for best fit with each method. Lastly the Deep Wiener Deconvolution was augmented by padding, not to improve results but because input sizes are required to be a multiple of 8.

Discussion. The baseline method for comparisons here is plain Wiener deconvolution, which is the earliest method listed here (and is only preceded by a simple inverse filter). It receives the lowest PSNR scores for both quality levels of kernels. Its meaningful parameter range is also almost just a point, i.e. it offers little regularization trade-off to better handle kernel errors. The method designated as Wiener-Hunt (WH) here differs in the implementation by adding a (3×3) Laplacian kernel to the regularization step (whereas Wiener would use an identity kernel). This simple change greatly

increases the quality for ground truth kernels (1 dB), and also slightly for imperfect kernels (less than 0.1 dB). The unsupervised Wiener (UW) variant scores even higher for perfect kernels, despite the fact that it does not even require tuning of a parameter (although it requires very close attention to padding). The lack of regularization parameter for UW becomes a major downside (over 1 dB) for imperfect kernels. This shows regularization not only counters noise but also kernel error. Next in line is Richardson-Lucy (RL), which also originated in the 1970's. While RL does not improve scores for perfect kernels, it improves them for imperfect kernels over WH (around 0.6 dB). Surprisingly RL is not far behind even the most modern methods on that axis (with less than 0.2 dB difference to the best). One downside is that in this experiment it was the slowest method.

From here on, regularization methods aim to model natural image statistics. The ℓ_2 prior is known to not model those accurately, but since it is convex, a closed-form solution still exists, making it very efficient. Despite its efficiency, the method scores higher PSNR than WH for perfect kernels. But especially for imperfect kernels, both ℓ_2 variants are easily within 0.2 dB of RL. Note that ℓ_2 comes in a frequency domain and a spatial domain implementation. The main difference is that the spatial domain ℓ_2 does not suffer from artifacts due to an incorrect boundary assumption, that frequency domain methods typically suffer from. Here we see that (for this particular image and kernel size) there is a small PSNR advantage for the spatial domain implementation. This advantage holds equally for both kernel qualities. The remaining three classical methods use image priors that are *sparse*, which requires iterative optimization to solve. These all outperform the previous methods on PSNR, especially for perfect kernels, by up to 1 dB. For imperfect kernels the PSNR advantages are moderate, with the Total Variation (TV) prior being on par with RL, and the Sparse prior (SPS) implementation by Levin et al. gaining the aforementioned less than 0.2 dB over RL. The Hyper-Laplacian (HL) prior method falls short of getting higher scores in either category (but it is faster than TV and SPS). Among TV and SPS there is no objective best, as each just offers a different trade-off for the different kernel qualities. Here TV scores higher for perfect kernels, and SPS for imperfect kernels. Note that SPS operates in the spatial domain, which offers two advantages over the similar HL: It does not suffer from boundary issues, and its conjugate gradient (CG) algorithm for convolution matrix inversion only approximates the inverse kernel. The approximation actually becomes an advantage when the kernels are imperfect. Here the number of CG iterations was set to 15, while convergence here occurs at around 50 iterations (the selected value for ℓ_2). If desired, a kernel filtering/degradation effect can also be achieved for other methods explicitly. One way to do this is e.g. thresholding or quantization. Nonetheless the method that scores highest PSNR out-of-the-box with the kernel prediction network here is SPS. SPS is thus used after the kernel prediction network for all following experiments.

In the deep learning category, the Deep Wiener deconvolution method is shown. This method easily outperforms all of the classic methods in PSNR when using perfect kernels. Much like unsupervised Wiener however, its lack of regularization parameter makes it unsuited for use with imperfect kernels. This does make deep non-blind deconvolution look very promising though, especially if a

regularization parameter was added.

Overall the main PSNR differences between non-blind deconvolution methods are largely with respect to ground truth kernels, with a quality range of around 5 dB. The range of the X-axis is about half of that (2.5 dB), although the amount of error that the axis represents is just one particular choice. To get a sense of the magnitude of the kernel errors displayed here, see the kernel metrics in table 6.3. Furthermore the types of kernel errors of the proposed method are not the same errors as e.g. classical $\text{MAP}_{x,k}$ estimation. The X-axis cannot be used to determine the non-blind deconvolution method that gives the highest PSNR scores for all use cases. One important insight from fig. 6.2 is that kernel estimation error has to be taken into account when selecting a method for use in blind deblurring.

As it stands, the choice of non-blind deconvolution method ends up being not highly consequential for the proposed method. Among the modern classical methods the PSNR differences are subtle. Consequently the performance of the proposed method is not strongly tied to any particular non-blind method. What matters most is that the regularization parameter is set higher than what is optimal for perfect kernels, to counteract some of the estimation error.

6.1.3 Blind Deblurring Overview

Next up, table 6.1 gives an overview of all the methods that are compared in the subsequent evaluations. The selection of methods is focused on deep learning methods. But as a reference for a classical method, the popular RMDB method by Xu and Jia is included. For every method, only one variant is selected for all future experiments. Often different architectures or training sets are provided, in which case the ones that give the best results across all test sets are selected. The proposed network is also the same instance through all experiments. It is trained on just one training set, and uses one non-blind deconvolution with fixed parameters. Although my results can be improved by optimizing for particular test sets (especially changing the training set), the aim here is to create a method that can generalize well, and to make a fair comparison. This table contains all the references for the methods, and lists some characteristics as context for interpreting the subsequent results.

Due to time constraints, only pretrained parameters of other methods are used, but a more fair comparison would also train each method from scratch on the same training set. (The exception is SelfDeblur, which is necessarily trained on every single image.) The main modifications to other methods are bug fixes, compatibility fixes (operating system, updating deprecated library code) and adding support for arbitrary input sizes (padding inputs to a multiple of 8 or 16 where required). For the SelfDeblur method, an additional postprocessing step is added, where the returned image and kernel are shifted to center the kernel by mass. This step helps improve its PSNR and MS-SSIM scores. This modification is labeled as SelfDeblur+ in the subsequent experiments.

As a final but important detail regarding the evaluations, note that all metrics are computed on deblurred images that have been shifted by integer amounts to get the best possible alignment with the ground truth images. Kernels are also shifted alongside images by the same amounts, to maintain their relation to the images. Aligning images is common practice, because the phase of kernel signals

	Deep Learning	RGB support	Spatially variant blur	MAP estimator type	Total running time (seconds)	Kernel estimation (seconds)
DeepDeblur (REDS)*	Yes	✓	✓	MAP _x	0.61	-
SRN (LSTM) [†]	Yes	✓	✓	MAP _x	2.80	-
DMPHN (1_2_4) [‡]	Yes	✓	✓	MAP _x	0.02	-
SIUN [YLC20]	Yes	✓	✓	MAP _x	0.20	-
RMDB [XJ10]	No	✓	✗	MAP _{x,k}	1.61	same
HNDB [ADF19]	Partial	✓	✗	MAP _{x,k}	7.30	1.73
SelfDeblur (23) [§]	Yes	✗	✗	MAP _{x,k}	856.21	same
NDeblur [Cha16]	Yes	✗	✗	MAP _{x,k}	161.40	12.95
My results	Yes	✓	✗	MAP _k	1.21	0.03

*[NHKML17] [†][TGS⁺18] [‡][ZDLK19] [§][RZW⁺20]

Table 6.1. Blind deblurring method overview: The table lists the selection of methods that are compared against in this thesis, which is focused on Deep Learning. The selected method variants are indicated in round brackets (e.g. architecture, training set). As the number of available implementations suited for general blur kernel estimation is limited, four methods that do not produce kernels are included. HNDB is categorized partial, because the deblurring itself is a classical method, but a denoising network is used to preprocess the input. The last column shows just the time for kernel estimation (if applicable), whereas total running time also includes deblurring. Time measurements were done on the *BSD-Levin* dataset.

is generally not uniquely determined. The alignment algorithm used here is very simple. It explores each cardinal direction so long as the PSNR score improves. Although it does not explore every possible alignment (due to practical limitations), results for all methods are greatly improved this way (in the order of 2-3 dB). (Note that this helps other methods more so than my results.)

Discussion. As a disclaimer, note that this is not an even ground for comparisons, because some methods are designed for spatially variant blur. Spatially variant deblurring focuses on simpler kernels, but the difficulty comes from the fact that there are different kernels across a single image. With that in mind, it is still interesting to see how they can keep up for challenging kernels.

So far there is only the running time to compare. For total running time, DMPHN is fastest, at least for the variant (out of five) that was selected. SelfDeblur is easily the slowest method, because two networks are trained for 5000 iterations for every input image. Its time also depends on the selected size of the kernel support, which is 23×23 here. The majority of the total running time of NDeblur is due to the use of the EPLL method for the non-blind deblurring, which performs well but is also slow [ZW11].² Among all the methods that explicitly consider kernels, the fastest running time is for the proposed method, even when using a spatial domain non-blind deconvolution. It is also faster than

²The reason why EPLL is not included in fig. 6.2 is because it is too slow to run on a large dataset several times (in my time budget), especially if RGB outputs are obtained by naïvely running it thrice.

	Deblur PSNR \uparrow	Deblur MS-SSIM \uparrow	Kernel ℓ_1 \downarrow	Kernel $\overline{\text{SSIM}}$ \downarrow	Kernel DISTS \downarrow
DeepDeblur (REDS)	22.991	0.832	-	-	-
SRN (LSTM)	23.265	0.840	-	-	-
DMPHN (1_2_4)	23.104	0.833	-	-	-
SIUN	23.307	0.841	-	-	-
● RMDB	23.830	0.862	0.537	0.012	0.013
● HNDB	23.293	0.842	0.537	0.010	0.025
● SelfDeblur+ (23)	22.017	0.801	0.865	0.024	0.052
● NDeblur	23.517	0.841	0.632	0.015	0.038
● My results	24.049	0.872	0.180	0.001	0.007

Table 6.2. **Evaluation on Gaussian3x3 set:** The second kernel metric is viewed as a loss metric for easier comparison, where $\overline{\text{SSIM}} = 1 - \text{SSIM}$. The parameters are the default for SSIM, i.e. $\sigma = 1.5$.

the classical RMDB method, although that is comparing CPU against GPU implementations. When considering just kernel estimation, the proposed method is by far the fastest. This is because there is no iterative refinement of a kernel estimate, and one forward pass is all that is needed (although even with 50 passes it would still be the fastest). Note that this also applies to the MAP_x methods, that are all easily below one second, except for SRN which also does iterative refinement.

6.2 Evaluation on Gaussian Blur

The first evaluation is on the Gaussian blur test set, shown in table 6.2 (and is done as described in section 6.1). Other than its practical applications, Gaussian blur is a good experimental baseline for blur kernel estimation. Gaussian blur is easier to *estimate* than motion blur. This is also why only one out of 16 kernels in the training set is Gaussian. On the other hand, Gaussian blur is harder to *deblur*. Such a baseline is especially relevant for the proposed method. The proposed method has no explicit convolutional model or any theoretical guarantee of sensible predictions. Even though the methods compared against were not designed or trained for Gaussian blur, the fact that it is easier to estimate should still make it a useful comparison. And further, Gaussian blur with a small σ in one axis (e.g. top-right or bottom-left in fig. 6.1a) is similar to a basic motion blur.

The proposed method scores best on every image and kernel metric, with e.g. over 24 dB PSNR. It is also the only deep learning method that scores above the classical method RMDB. In terms of deblurring performance, the other methods are not far behind though, ranging mostly from 23 to almost 24 dB PSNR. The only exception is SelfDeblur with 22 dB, which generally struggles in cases where the kernel support is larger than the actual blur bounding box. Even the spatially variant methods are performing well, with SIUN scoring 23.3 dB PSNR. Since Gaussian blur is hard to

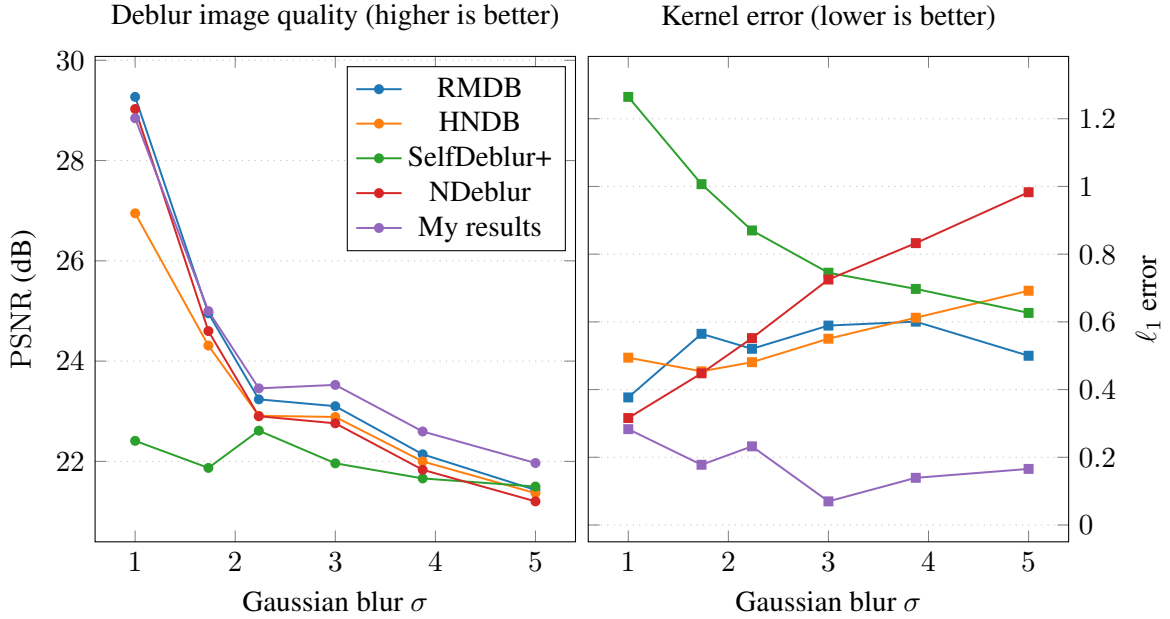


Figure 6.3. **Effect of kernel error on deblurring:** Shown are evaluation results of the *Gaussian3x3* test set, broken down by projected blur size σ (the dataset blur kernels each have a σ_x and σ_y value, which is projected as $\sigma = \sqrt{\sigma_x \sigma_y}$). Unlike motion blur, Gaussian blur can be easily broken down into one parameter this way, enabling one to judge kernel estimation error. The left plot shows deblurring scores for the methods that output kernels. In general, deblur quality is expected to decrease with larger σ values, since Gaussian blur is a low-pass filter that aggressively (compared to motion blur) discards information. It can be difficult to judge how well a method performs based on deblurring score, because deblurring quality depends on many factors. The right plot thus shows the same data, but looking at the estimated kernel, using ℓ_1 as a quality metric.

deblur, the PSNR score differences are more subtle.

The kernel metrics on the other hand show a greater variance. They also generally correlate with deblurring performance, but have subtle deviations. For instance ℓ_1 kernel scores rank RMDB and HNDB equally, but RMDB has notably higher deblur scores (0.5 dB PSNR difference). This could be attributed to differences in deblurring rather than kernel estimates. But at least the DISTS metric points towards RMDB indeed having better kernel estimates. Kernel SSIM is not better at predicting deblur scores either, as it puts RMDB and HNDB into the incorrect order. In this evaluation, the one outlier with all kernel metrics is NDeblur, which they rank fourth, while deblurring metrics rank it third. Even though fig. 4.5 showed that ℓ_1 is not an ideal kernel error metric in general, Gaussian blur is simple enough for it to work well. In fact the earliest versions of the proposed method did only Gaussian blur estimation, using ℓ_1 as a loss function (and it was not until motion blur that this became problematic).

To get a more in-depth view on the characteristics of each method, and the quality of image and kernel metrics, fig. 6.3 breaks them down by the Gaussian blur width σ .

The broad picture of the two plots mirrors that of the table: Deblur scores show less differences

than kernel metrics. Here the left plot shows only methods with kernel outputs, for simplicity. As in table 6.2, we see that the methods perform similarly in terms of PSNR, with the exception of SelfDeblur. This provides confirmation that SelfDeblur struggles when the kernel support is too large, because at small blur sizes the scores are the lowest in comparison. For the largest blur size $\sigma = 5$, where the kernel support size is tight, it actually moves from fifth to second rank. Other than this, rankings of the methods do not change significantly along with blur size. The proposed method ranks first at most stages, except for the very first, where it still remains close to the top score.

Looking at the kernel error, the subtle differences of deblur PSNR are amplified. SelfDeblur starts off with a high kernel error as expected, but also overtakes two methods towards the end. The proposed method has the lowest kernel ℓ_1 error at all stages. Its error also generally decreases with a larger σ . Note that the kernel support of the proposed method (and also SelfDeblur) of 23×23 cannot fully contain the largest blurs. This makes some error starting around 4σ unavoidable. To revisit the deblur/kernel ranking incongruity from earlier, the kernel plot indicates one issue. There NDeblur stands out as well, being the only method that has a steady increase in estimation error. On average, the kernel error of NDeblur is higher than that of HNDB, but not at the smallest blur size. Knowing that larger σ values are harder to deblur (as they are wider low-pass filters), kernel estimation error becomes more important at smaller Gaussian blurs. For images with very large Gaussian blur, most high-frequency information has been lost and so there is not much to reconstruct. The averaged kernel scores of table 6.2 fail to encompass that smaller σ scores are more important. This can also be used to explain how RMDB scores higher than HNDB even though their kernel error curves are similar: RMDB has a lower kernel ℓ_1 error at the smallest σ , giving it an edge for deblurring scores.

6.3 Evaluation on Real Motion Blur

With the easier baseline experiment out of the way, the primary evaluation is on real motion blur, shown in table 6.3. Each of the kernel estimation methods, including the proposed method, has done evaluations on the kernels by Levin et al. in some shape or form. These evaluation results for the kernel estimation methods are thus on an even playing field. Compared broadly, the top deblur scores are higher, and the kernel errors are higher than those of table 6.2. Even though the kernel estimates are less accurate by the metrics, the overall deblur scores are higher, showing that motion blur is easier to deblur. The performance of the spatially variant blur methods has dropped. They are no longer competitive with methods that model a single blur kernel (which is to be expected). The highest scoring method by almost all metrics is the classical RMDB. The overall second rank goes to HNDB, which uses classical estimation that is augmented by deep denoising. The proposed method scores higher in PSNR than HNDB, but not in any of the other image and kernel metrics. None of the deep learning methods are quite state-of-the-art yet. The closest one is the proposed method, with a gap of about 0.5 dB PSNR. In terms of kernel error they are notably behind as well.

It still remains desirable to find a kernel metric that correlates with deblur quality. For Gaussian

	Deblur PSNR \uparrow	Deblur MS-SSIM \uparrow	Kernel ℓ_1 \downarrow	Kernel SSIM \downarrow	Kernel DISTS \downarrow
DeepDeblur (REDS)	22.239	0.799	-	-	-
SRN (LSTM)	22.407	0.810	-	-	-
DMPHN (1_2_4)	19.406	0.605	-	-	-
SIUN	22.727	0.819	-	-	-
● RMDB	24.899	0.906	0.718	0.008	0.037
● HNDB	24.345	0.890	0.739	0.010	0.030
● SelfDeblur+ (23)	22.825	0.821	1.271	0.028	0.086
● NDeblur	24.027	0.871	0.873	0.014	0.038
● My results	24.405	0.882	0.827	0.013	0.032

Table 6.3. Evaluation on *BSD-Levin* set

blur, one issue we identified was that estimation error is ideally influenced by blur size, to match deblur scores. But this missing aspect is less of a problem for motion blur, since motion blurs are not low-pass filters. Another issue for motion blur metrics is that e.g. one-pixel misalignments between ground truth and predicted kernels cause high error scores in most directions. These high error scores do not translate to deblur error scores, as shown in fig. 4.5. The issue is mitigated in the evaluations here since images, and kernels alongside them, are shifted to their best alignment with the ground truth. It is still not entirely removed since only integer shift amounts are employed.

In this instance the ℓ_1 and SSIM kernel metrics match deblur scores more closely. The ℓ_1 metric matches the deblur PSNR ranking, except for one instance with a subtle error of 0.005. The SSIM metric matches completely. The DISTS metric on the other hand correlates much less with deblur scores. It scores three methods equally around 0.037, while the corresponding deblur scores differ by up to 1 dB PSNR.

Similarly to the Gaussian evaluation, the results of the table are broken down further. As a result of the real motion blur kernels lacking design parameters such as σ , the box plots in fig. 6.4 just show the score distributions. This will give insights into how consistent each method performs. Note that methods that do not output kernels could be shown in both plots as well, but are omitted for clarity since they occupy a different PSNR range.

The test set blurs the same base images with different kernels, and vice versa. The left box plot is thus a summary of the average PSNR scores for each of the 20 base images. One interesting finding is that there is one outlier base image in the test set that is particularly easy to deblur (this is the “aquarium” image with a black background). Every method scores easily above 30 dB PSNR on this image. None of the remaining images are deblurred upwards of 28 dB by any method. Overall the listed blind deblurring methods tolerate different base images well, since differences are small. This eliminates the possible concern that the learning-based methods (NDeblur and the proposed

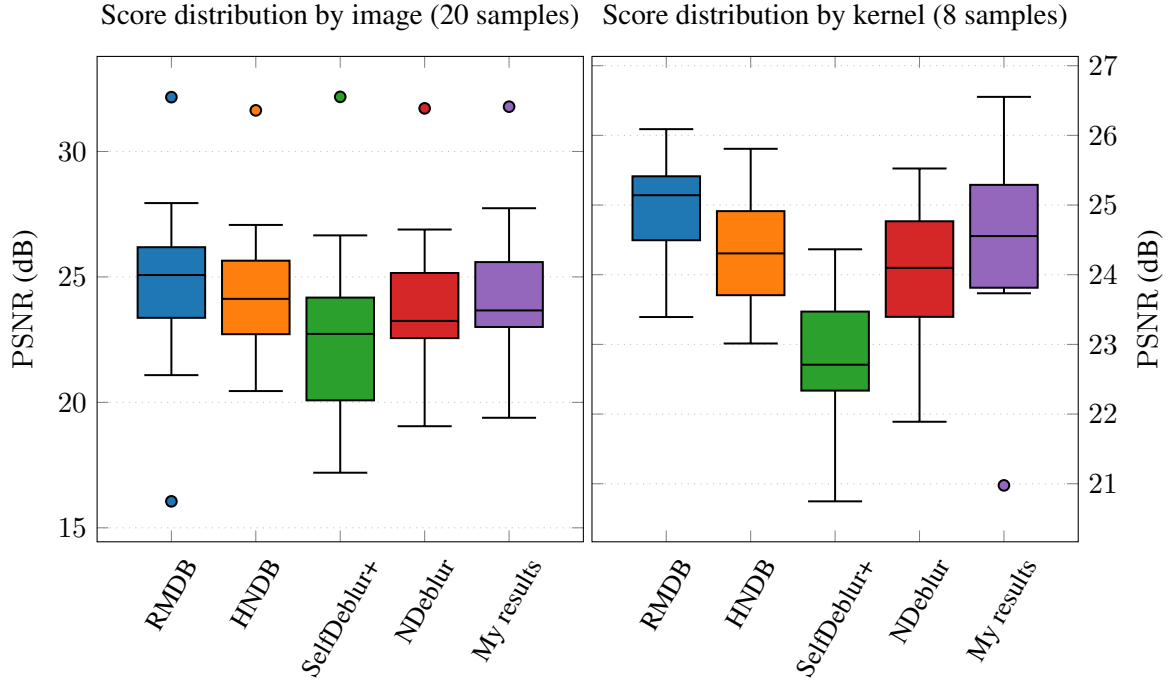


Figure 6.4. Effects of different kernels vs. images on the *BSD-Levin* set: This dataset pairs each of the real motion blur kernels by Levin et al. with 20 base images. The left plot shows how each method performs when viewing all the scores averaged per image. The right plot also shows the dataset scores but instead averaged per kernel. Since the kernels of this dataset cannot be ranked (like the other two test sets), this plot still gives some sense of how the methods perform for the kernel set as a whole. The box plots consist of upper and lower quartiles Q_3 and Q_1 (75th/25th percentiles), and median (50th percentile), that are computed using a standard type 7 estimator [HF96]. Whiskers are drawn for points that lie in the standard 1.5 interquartile range ($IQR = Q_3 - Q_1$) beyond the upper/lower quartiles, e.g. the upper whisker is within $Q_3 + 1.5 \cdot IQR$. Any points outside are drawn individually as outliers.

method) overfit on the images they were trained on. There is one outlier image at which RMDB scores consistently low PSNR at around 16 dB, but it still ranks first.

The right box plot shows the average scores for each of the 8 kernels. This perspective shows a lot more variation between the methods. RMDB not only scores highest on PSNR, it also does so consistently for each of the 8 kernels by Levin et al. The proposed method has the highest ceiling for a single kernel (over 26 dB). But it also fails completely on one kernel (below 21 dB). This is the kernel that can be seen in fig. 6.6. Outside of this failure kernel, the proposed method scores close to 24 dB PSNR with all other kernels.

Following the quantitative analysis are now two qualitative examples. These are done on the real motion blur set since it makes the most fair comparison. First fig. 6.5 shows a case where the proposed method performs well.

The proposed method scores over 26 dB PSNR, with second rank method SelfDeblur at just above

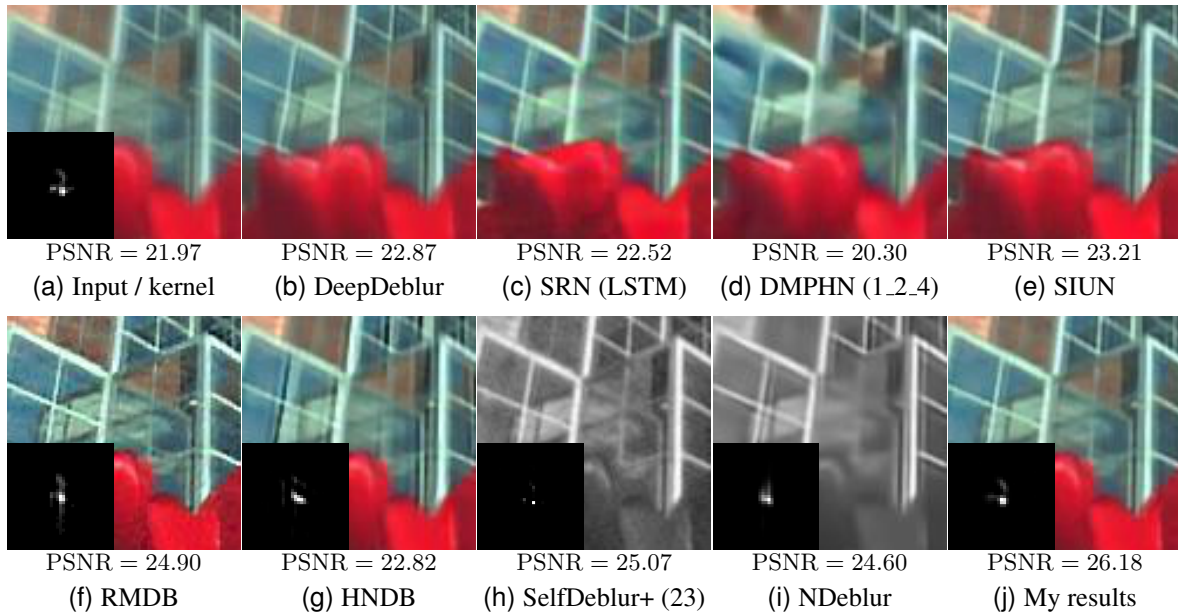


Figure 6.5. **Qualitative comparison:** This example was taken from the evaluation on the *BSD-Levin* set. The image has been cropped for better visibility.

25 dB. The highest score among the methods without kernel output (upper row) is SIUN at 23.21 dB. For a more simple and small blur kernel, those methods are 1 to 2 dB PSNR above the blurry image. In the images, note the somewhat horizontal white window edge at the top-left. In the blurry image it is dragged downward to create a half-opaque wider beam. For some methods this beam is reduced to be thinner, but not entirely removed. RMDB removes it, but goes too far, creating a dark blue ringing artifact. This corresponds to a kernel estimation error where RMDB adds an additional downward stroke. The RMDB image also contains more high frequency noise. For HNDB the estimated kernel does not closely resemble the ground truth, giving it one of the lower PSNR scores of below 23 dB. The kernel of SelfDeblur shows its characteristically sparse estimates. The SelfDeblur kernel is essentially just three pixels. They correspond to the center point and the two main stroke directions of the ground truth kernel. The estimated center pixel by SelfDeblur also has the heaviest weight, matching the heavy center dot in the ground truth. Despite the fact that this kernel looks visually dissimilar to the ground truth, it evidently catches important characteristics, because SelfDeblur ranks second in PSNR. The NDeblur kernel estimate is also visually dissimilar to the ground truth. In the NDeblur image, one window pane and the foreground tulips are even more blurry than in the input image. But the PSNR score is still almost 2 dB above HNDB, and less than 0.5 dB behind SelfDeblur. For the proposed method, the kernel estimate is visually very similar to the ground truth. The main missing feature in its kernel estimate is a subtle stroke starting at the center point and going to the right. This good kernel estimate is what leads to the high PSNR score. The aforementioned white beam in the deblurred image is removed better than with the other methods, without adding ringing artifacts or noise.

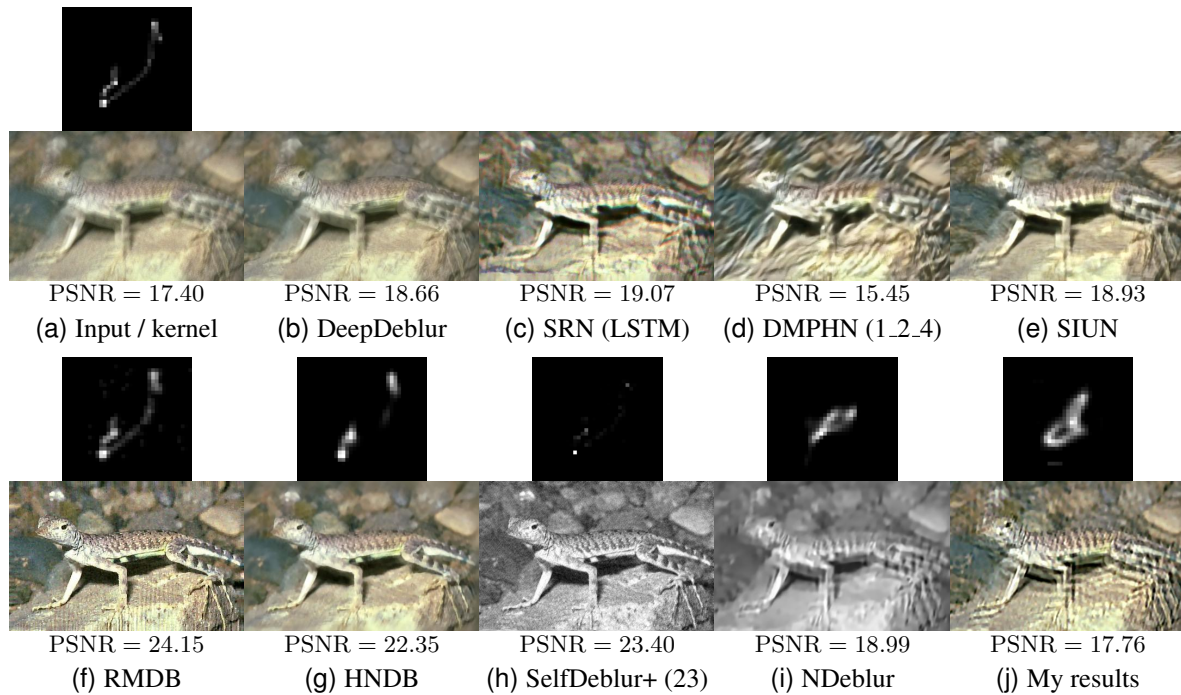


Figure 6.6. **Qualitative comparison for failure case:** This example was taken from the evaluation on the *BSD-Levin* set.

On the flip side, fig. 6.6 shows a failure case for the proposed method. The blur kernel here is what caused the low outlier point for the proposed method in the right box plot in fig. 6.4. On this image, the proposed method just barely increases the PSNR score (17.40 vs. 17.76 dB). Due to the inaccurate kernel prediction, the output of the proposed method has prominent ringing artifacts throughout. Note that the shown blur kernel is also the largest kernel from the *BSD-Levin* test set (e.g. in terms of bounding box size).

In this instance, we may consider the methods to fall into a successful or unsuccessful group, because there is a wide 3+ dB PSNR gap between 19.07 and 22.35 dB. Three methods are above this threshold, the remaining six are below. Methods designed for spatially variant blur (upper row) are at a heavy disadvantage, due of the large blur size. Here they all fall into the lower PSNR group. DeepDeblur sharpens the blurry image, but does not remove the blur shadow (i.e. the duplicate lizard). SRN does remove blur shadow, but has RGB color ringing artifacts. DMPHN aggressively removes any blurred edges, but retains little of the original sharp image. SIUN also removes the blur shadow, but is overall still not very sharp.

The highest score is obtained by RMDB with 24.15 dB PSNR. Its deblurred image is generally sharp and free of ringing artifacts due to kernel estimation error. It is noisy however and has boundary artifacts. The RMDB kernel estimate matches the ground truth in structure and weight. There is some noise in the kernel it estimated though (this is only visible with good contrast in the PDF version). The line thickness is different to the ground truth, but as SelfDeblur showed, this is not necessarily

problematic. HNDB has the lowest PSNR in the successful group. Its kernel estimate gets the weight proportions correct, but lacks a finer resolution that is needed to accurately represent the ground truth kernel. The deblurred image by HNDB thus also lacks the most fine details. SelfDeblur again estimates a very sparse kernel, but not greatly detrimental to its PSNR score of 23.4 dB, which puts it second. Despite its sparsity, the estimated kernel does contain the fine line between the heavier upper and lower parts of the ground truth kernel (again only visible with good contrast). The most prominent flaw of the SelfDeblur image is that it is noisy. Both NDeblur and the proposed method fail to estimate the blur kernel with the correct proportions. In other words, they lack a heavy upper and lower component with a largely empty center (by weight). As a result of bad proportions, both methods fall into the unsuccessful group. Especially the proposed method scores a low PSNR value, ranking even below three spatially variant blur methods.

Summary. We compared several blind deblurring methods on a test set with real motion blur. Although the proposed method had the highest PSNR among the deep learning methods, there is still an upward gap toward classical blind deblurring methods. A box plot of the score distribution per kernel revealed one bad outlier. In a qualitative analysis, this outlier points to a first shortcoming of the proposed method. Before mentioning a proposed cause for the poor kernel proportions of fig. 6.6j, let us first consider some alternative causes. One initial suspicion is that the kernel generation method is unable to create similar enough training examples. However, considerable effort has already been invested into optimizing kernel generation parameters, as shown in section 5.4. It is still possible that the kernel generation method itself, no matter how optimal the chosen parameters are, is unable to create similar enough kernels. In regards to this possibility, consider that the motion blur kernel generation is one of the most sophisticated implementations available, to the best of my knowledge. Another possibility is that the blur kernel parameter space (i.e. the bottleneck) of the proposed kernel prediction network is too small. When too small, the diversity of the output kernels is limited. I argue that parameter space limitations alone cannot explain the difference of the kernel in fig. 6.6j to the ground truth. As seen in fig. 6.5j, predictions can be detailed despite this limitation. For some quantitative evidence, see table 6.4, where the proposed method generalizes to unseen kernels.

I propose that one weakness for kernels predicted by the network, such as the one in fig. 6.6, is attributable to a lack of shift-invariance in the loss function. The aforementioned kernel has, broadly speaking, a dumbbell shape: two heavier ends connected by a thin center stroke. Now consider that above all, kernels that the proposed network predicts need be centered. No matter how accurate the shape, proportions and rotation of a prediction may be, if it does not align well with the ground truth, it effects a high loss value. Secondly kernel predictions have to incrementally improve over time, as training progresses. Incrementally learning a dumbbell shape kernel poses a challenge. The first possible solution is that the heaviest end is learned first. Due to the centering rule, this heavy end would be predicted in the center. Unfortunately the alignment with the ground truth is then poor. This is because the ground truth center is almost empty, and the heavy ends lie on the outside. Secondly if both heavy ends were predicted simultaneously, alignment with the ground truth would be achieved.

But this solution puts the cart before the horse. In order to obtain a modest loss value, intricacies of the kernel shape need already be known (e.g. that there are two heavy ends, and their proportions). Moreover motion kernels are not disconnected, and most of the time also not approximately disconnected, as is the case with a thin center stroke for a dumbbell shape. This strategy would thus not fare well for most kernel types. The third incremental learning possibility would be to start with the thin center stroke, but fine details are not learned first. To conclude, some types of motion blur kernels are difficult to learn, due to the lack of shift-invariance in the loss function. Specifically these are kernels that have most of their mass off-center.

6.4 Evaluation on Synthetic Motion Blur

The third test set where comparisons are done is using synthetic motion blur. The kernels of this set are designed specifically to be broken down by some parameter, namely the motion path length. This allows one to gain some insight into the strengths and weaknesses of each method. Moreover a second test set for motion blur is very desirable. Several motion blur test sets exist, but using the motion blur by Levin et al., which has only 8 kernels. In this thesis, the same kernels were also used for the second test set, except for rotating those kernels to add a little variation. But 8 unique kernels is still a small number, especially in the age of deep learning, where overfitting is a key concern. The synthetic test set created here contains 180 unique motion blur kernels.

As for the validity of the results, note that the proposed method was trained on kernels that use the same kernel generation implementation. Although the parameters on this test set differ from those that were used for the training set. None of the parameter combinations of this test set were used in the training set. For instance, the maximum path lengths L_{max} in the training set are set to 40 (and one out of 16 is set to 100). In this test set they range from 10 to 50. The NDeblur method was also trained on synthetic motion blur, but using their own custom implementation. There are no known concerns for the other blind deblurring methods.

The summary of the evaluation is shown in table 6.4. Looking at the score ceiling, deblur PSNR here is roughly 24 dB, compared to 25 dB in the second test set. The floor for kernel ℓ_1 is around 0.7 for both test sets. So broadly speaking, this test set is slightly harder to deblur and kernel estimation difficulty is similar. The proposed method ranks first on all image and kernel metrics. NDeblur ranks second on PSNR, while RMDB is second on MS-SSIM (with a small margin of 0.002 MS-SSIM between RMDB and NDeblur). It is interesting to note that RMDB is notably lower in PSNR than NDeblur, with a difference of over 0.6 dB. Looking at the kernel metrics, RMDB scores higher errors on all of them. So on this set PSNR correlates higher with kernel errors than MS-SSIM. Surprisingly even though RMDB scored highest on the previous test set, here it ranks on the lower end in PSNR with 22.81. It is closer to the spatially variant methods in PSNR, the closest being SIUN with 22.586. HNDB ranks second in terms of kernel metrics (even lower errors than NDeblur), but places third in PSNR. SelfDeblur ranks 8th in PSNR, so even lower than most spatially variant methods. One

	Deblur PSNR \uparrow	Deblur MS-SSIM \uparrow	Kernel ℓ_1 \downarrow	Kernel $\overline{\text{SSIM}}$ \downarrow	Kernel DISTS \downarrow
DeepDeblur (REDS)	21.827	0.788	-	-	-
SRN (LSTM)	22.326	0.808	-	-	-
DMPHN (1_2_4)	19.767	0.626	-	-	-
SIUN	22.586	0.813	-	-	-
● RMDB	22.810	0.850	0.982	0.011	0.047
● HNDB	23.243	0.845	0.791	0.011	0.031
● SelfDeblur+ (23)	21.419	0.793	1.212	0.024	0.067
● NDeblur	23.450	0.848	0.805	0.011	0.032
● My results	23.900	0.875	0.716	0.008	0.025

Table 6.4. Evaluation on *Motion-Length* set

aspect to consider here is that the shorter kernel lengths of this test set are just straight lines. This is exactly the simple type of motion blur that spatially variant methods are designed for.

The obvious next step is to do another break down of image and kernel metrics, this time by the motion length parameter. Figure 6.7 shows deblur PSNR for the image metric, and ℓ_1 error for kernels. Unlike with Gaussian blur in fig. 6.3, the PSNR differences are more varied between the methods. As a general trend, we also see difficulty increase with motion blur length. This applies not only for deblurring, but also for kernel estimation. For deblur PSNR, there are no major ranking changes between the methods. In other words, there is no method which has a pronounced weakness or strength tied to motion blur length L_{max} . The proposed method has its highest PSNR at length 20, so it has a subtle weakness on the smaller lengths of 10 and 15. On the test set in general, lengths 15 and 25 have lower PSNR scores compared to the neighboring values. Due to the random nature of the test set, there cannot be a perfectly linear increase in difficulty, so this is to be expected. Some methods handle this increase in deblur difficulty better than others. RMDB has an especially noticeable drop at 25. SelfDeblur handles those two values the best in relation to its own performance. Surprisingly SelfDeblur has the highest PSNR at the smallest motion lengths, and then drops off faster than the other methods. This is despite the fact that kernel support size is tightest as the larger lengths. Overall the proposed method has the highest deblur PSNR at all lengths.

For kernel error, the ℓ_1 metric was selected because it had the highest (absolute) correlation with deblur PSNR among the three kernel metrics of table 6.4. Note that in comparison to fig. 6.3, ℓ_1 errors start higher, from near 0.6 (compared to 0 for Gaussian blur). This raised floor can explain why there is more variation in the deblur PSNR between the methods. Generally the rankings in this metric are matching deblur PSNR. For instance, SelfDeblur has universally the highest ℓ_1 errors and the lowest deblur PSNR. This is followed by RMDB with matching rankings. Going further though, there are ranking changes between the remaining methods in the ℓ_1 error, that are not present in deblur PSNR.

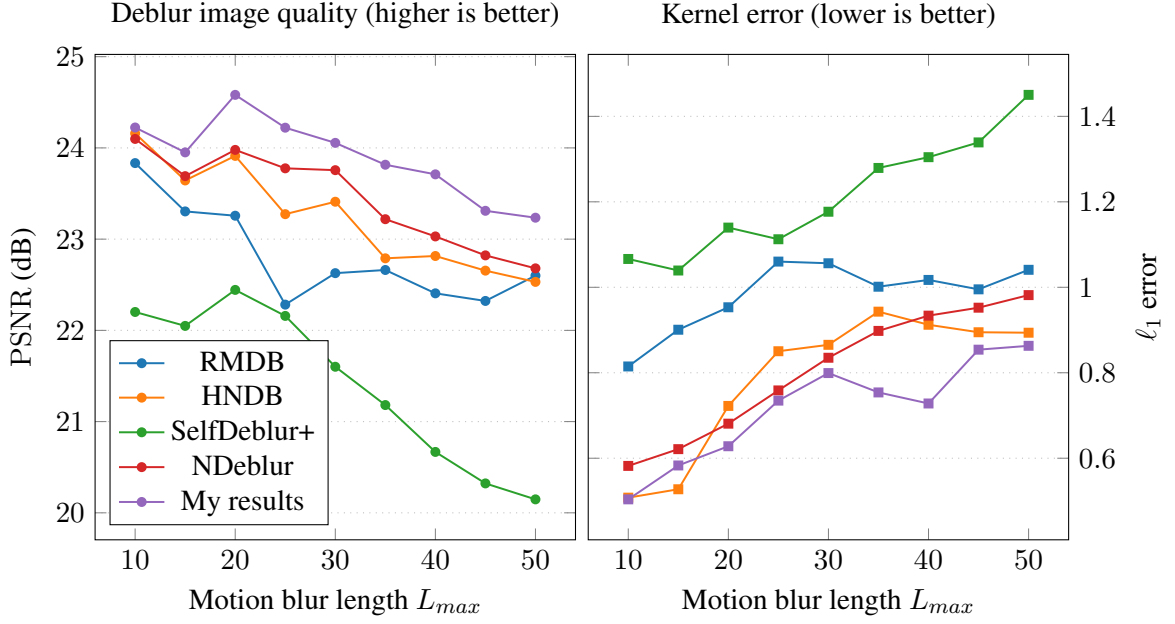


Figure 6.7. **Assessment of error on motion kernels:** Similarly to fig. 6.3, this shows a breakdown of deblur and kernel error by a parameter of the blur kernel. Here the test set was created in such a fashion that motion kernel length L_{max} increases, along with the probability for the motion path to curve. This is designed such that difficulty increases with length. Among the kernel metrics of table 6.4, ℓ_1 is shown here, as it had the best correlation with deblur PSNR.

For instance, HNDB has the lowest ℓ_1 error at length 15, but does not overtake the proposed method or NDeblur in deblur PSNR. The RMDB ℓ_1 error at 25 and 30 is almost the same, but in PSNR there is a notable difference. This shows that the ℓ_1 metric for kernel error cannot perfectly predict PSNR deblur scores for motion blur. Lastly we are able to analyse the kernel errors of the proposed method. It has a peak in ℓ_1 error at length 30, then a valley at length 40. A lower ℓ_1 error at length 40 makes sense, because this is exactly the value that almost all training set examples use. In regards to the lower lengths, these are similar to Gaussian blur where one σ component is small. The reason why lengths 45 and 50 produce higher ℓ_1 error is the same as in the Gaussian test set. The kernel support size of the proposed method is 23×23 , which is too small to contain the largest blur sizes (in cases where the blur is straight). Overall the proposed method is able to score lower ℓ_1 kernel errors than other methods, even at lengths where its generalization gap is the largest.

To put these results into perspective, note the differences to the first motion blur test set. The performance of the proposed method does not change significantly (24.405 dB PSNR vs. 23.9). The proposed method scores lower on this third test set, which is to be expected as all other methods score lower as well. Compared to other methods however, the proposed method does have higher deblur PSNR scores on this third test set. RMDB specifically has the largest drop in deblur PSNR, which is unexpected. Looking at it quantitatively, there are some failure cases and noisy estimates for RMDB, but these are not tied to specific motion blur lengths. As stated in the previous section, some of the

	Training	Validation
Augmented	26.62	25.99
Not augmented	26.08	25.22

Table 6.5. Train-validation gap: Shown are the deblur PSNR (dB) scores of the kernel prediction network on four validation set variations. The kernels are the same as in the training set, i.e. synthetic motion and Gaussian blur. The top left cell corresponds to the final measurement of standard training, and the bottom right analogously to standard validation (both marked in bold font). The non-standard measurements are done to isolate the compound effects of base image and augmentation changes. Each cell is the mean score over 4800 images with random kernels.

kernels by Levin et al. are particularly challenging for the proposed method. Conversely the kernels of this test set are more challenging for other methods. Both NDeblur and the proposed method are robust, losing just over 0.5 dB in PSNR between the first and second motion blur test set. The other methods with kernel estimation have a PSNR drop of at least 1 dB.

6.5 Understanding the Proposed Method

This last section investigates the proposed method in isolation. One possible issue that stood out during optimization of the parameters in fig. 5.2 was the gap between training and validation loss. Another issue that was previously mentioned was generalization to new kernels. This is a distinct issue from that, because both training and validation use the same kernel generation. This leaves only two differences between those datasets: base images and data augmentation. For most of the optimization, validation loss decreases at the same rate as training loss. But there is still a possibility for very early overfitting that remains constant from thereon. To investigate this, table 6.5 isolates the two varying factors between training and validation loss.

The lowest PSNR score of over 25 dB is for the standard validation set, i.e. without augmentation. On the other end, the standard training set scores 26.62 dB. In other words, the loss difference between training and validation comes down to 1.4 dB in terms of deblur PSNR. This gap can be broken down into two factors. For one, the validation and test sets have images that are unknown to the trained network. This isolated factor is shown in the upper-right table cell, and results in a deblur PSNR drop of 0.63 dB. To improve on this aspect, the base image set can be expanded. With more base images, the network is less able to overfit on particular image features.

The more interesting factor is data augmentation, as a unique concern with the implementation of the proposed method. This is isolated in the bottom-left table cell, and results in a PSNR *drop* of 0.54 dB. In other words, data augmentation improves kernel estimates. As mentioned in the implementation chapter, this is because data augmentation is done prior to blurring. Blur is then easier to estimate on the black borders from augmentation padding. To improve on this aspect, data augmentation could be removed. But this would increase the performance drop for the first factor, since that makes the base images less diverse. To get more fine grained control, the mask that is used to cull the

blur on the augmentation border can be adjusted. The two factors have a compounding effect, as the combined drop in PSNR is greater than the sum of the individual factors. If no augmentation borders are available, the network can focus on known image features, and vice versa.

As a closing thought on this table, note that even with the lowest score of 25.22, the proposed method would still rank first on the three test sets. So regardless of the training-validation gap, there is definitely room to improve motion blur generation and trainability.

The final experiment is a qualitative analysis of how the kernel prediction network makes its predictions. The analysis is done using integrated gradients, that show which input features are used to create a (scalar) network output [STY17]. Figure 6.8 shows the gradients overlaid onto the inputs, for two blurred images. Images are blurred by Gaussian blur because it is easier to analyse. The first output is the mean activation of the bottleneck layer, the second one the mean of the predicted kernel. Outputs at those two stages are made using different features in the inputs. Conceptually the bottleneck is estimating blur parameters, while the kernel output entails constructing the kernel from the blur parameters.

For the first blurred image, the bottleneck gradients are sparse. The strongest gradients are at the top left on the rock, and top right on a thin piece of debris. More gradients are subtly spread throughout the image. The directions of the gradients tend to align with the axes of the Gaussian kernel. In other words, image edges are favored that align with the kernel. On the other hand, the kernel output for the first image has a lot more gradients than the bottleneck. The gradients are also no longer strongly aligning with the kernel. Instead they adhere to contours in the image. For the second blurred image with a larger Gaussian kernel, the gradient intensities of the two rows are flipped. For the bottleneck activation, gradients are spread throughout the whole image and more prominent. Then for the network output, they are more sparse again, and also adhere to image contours.

Although such gradients are difficult to interpret, we gain some insights into how the kernel prediction network makes its decisions. For one, there are no universal points that are used to estimate a kernel. In the same base image, different features are prioritized depending on the blur kernel. Those features are also generally diverse, rather than being focused on a few points in the images. This is consistent with the experimental finding that the proposed method is robust to different images.

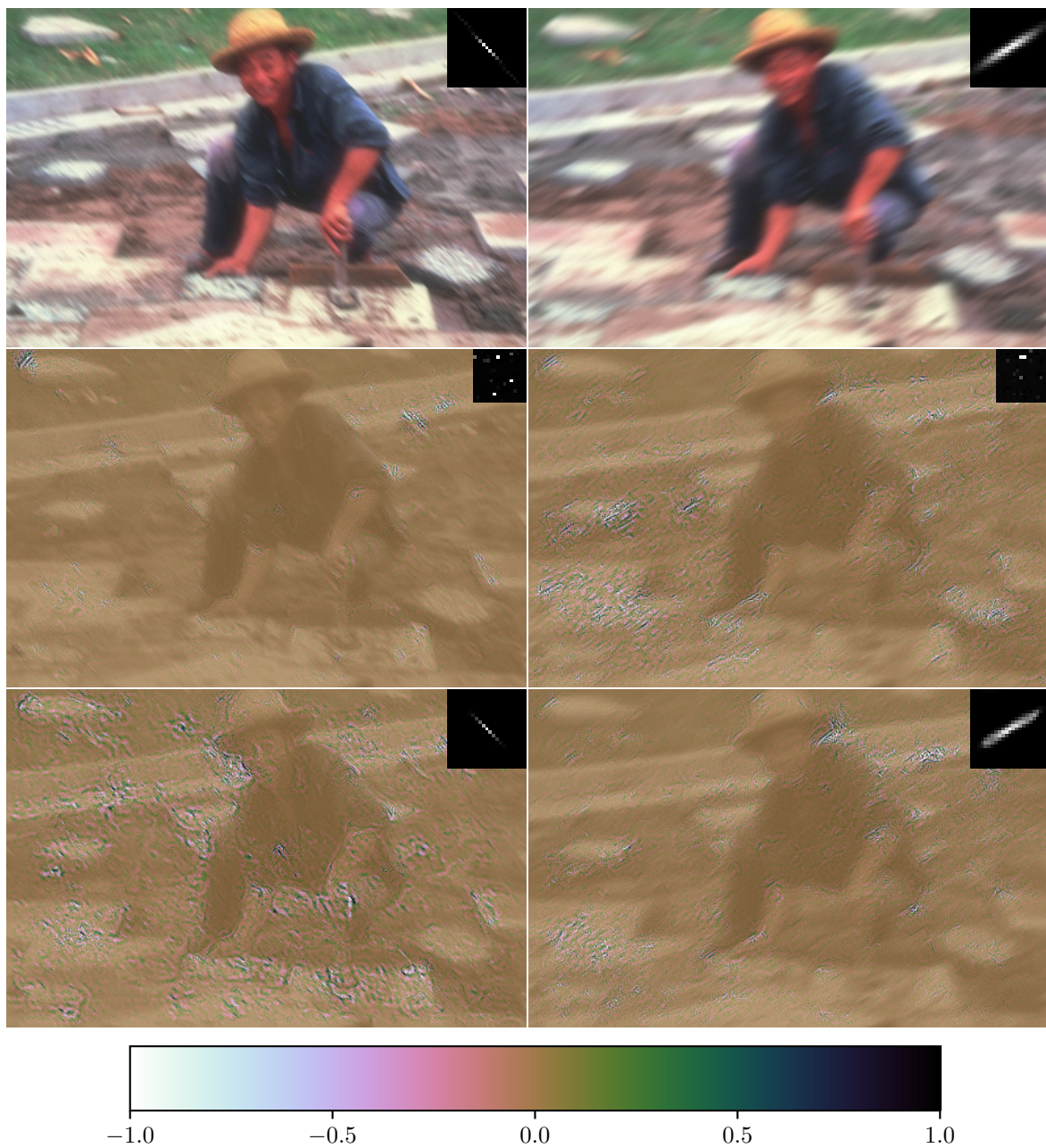


Figure 6.8. Attribution via Integrated Gradients: Shown is one base image blurred with two different (Gaussian) kernels (top right corner), and the corresponding gradients attributed to the input. The range-normalized gradients are overlaid on the input images using the color map shown below. The second row shows attribution for the mean output of the linear bottleneck layer of the kernel prediction network. The linear layer output is shown in top right corner, reshaped to two dimensions for ease of viewing. The third row shows attribution for the mean of the predicted kernel output. Following the idea of attribution techniques, the aim of this figure is to give a glimpse into the inner workings and reasoning of the network.

Chapter 7

Conclusion and Future Work

The aim for this thesis was to create a deep blur kernel estimation network, using a MAP_k estimator. This type of estimator predicts only a kernel at its core, as opposed to a deblurred image. The choice of estimator was inspired by the findings of Levin et al., who showed that simple MAP_k is easier to optimize than $\text{MAP}_{x,k}$. In this thesis this finding is also empirically validated in a deep learning setting. Since a deep learning MAP_k estimator has not been done before for challenging blurs like motion blur, a lot of foundational work was required. Training datasets, loss function and network architecture that are suitable for this task had to be devised. Moreover non-blind deconvolution and test sets were needed to evaluate the proposed algorithm. Hyperparameter optimization is not the focus of this thesis, given the focus on foundations. Yet hyperparameters are still required to be somewhat optimized in order to get a functional deep learning algorithm. Training set synthesis and augmentation, architectural choices, optimization schedules and loss functions were thus also evaluated behind the scenes.

Still there is ample room for formal hyperparameter optimization and ablation studies in future work. The lack of such optimization means there is room to grow. In this thesis, a few specific weaknesses to improve upon are also identified. First and foremost, the lack of a loss function designed specifically for kernel estimation makes training less effective. The ideal loss would have an increased shift-tolerance, but remain intolerant to rotation and scaling, which would help to learn more challenging blurs. Secondly the training set is not perfect. More base images should be added, and data augmentation and motion blur synthesis need further optimization. Specifically a new parameter—mask size—for post-blur treatment of the augmentation boundary was conceptualized. The kernel orientation in the dataset was optimized, but not how kernels should be normalized. For instance, kernels could be range-normalized, and a softmax layer can be placed at the end of the network. For motion blur synthesis, e.g. an additional parameter to vary line thickness could be created in order to simulate micro-stutter during image capturing. The higher the velocity in the motion blur simulation, the thinner the line should be, to more accurately simulate real motion blur. While the proposed architecture is simple, future architectures may specialize more heavily on kernel estimation, as e.g. attempted in appendix B.

In the evaluations, the proposed algorithm performs favorably against state-of-the-art deep learning blind deblurring methods. It scores highest in both image metrics in two test sets, and second/third in a real motion blur test set where it is still the best deep deblurring approach. These rankings are also mirrored in terms of kernel error metrics. The performance differences between the two motion blur test sets also highlights an importance of larger test sets in future evaluations, especially for deep learning approaches.

A selection of three kernel metrics are used in evaluations, because kernel estimation error is the primary factor of interest in deblur quality. Where kernel estimates are available, this is preferable to comparing deblurred images, which is complicated by additional factors (e.g. choice of non-blind deblurring algorithm). Among the three kernel metrics, the very simple ℓ_1 error is found to correlate best with deblur metrics. An ideal kernel metric however would mirror properties of deblurring. For instance, translations or line width differences should be weakly penalized, but not rotation or scaling. This task has strong overlap with that of finding a good kernel loss function.

The proposed algorithm is also very efficient, requiring in the order of 30 milliseconds to make a prediction. This opens up interesting possibilities to boost performance of slower methods, e.g. by using predicted kernels as initial estimates. One such application could be the SelfDeblur method, which is complementary to the proposed method in several ways. SelfDeblur is very slow, the proposed method very fast. SelfDeblur is accurate when it works, but struggles to consistently center kernels and get the rough kernel shapes correct. The proposed method is good at centering and getting the rough kernel shapes correct. In combination both algorithms may achieve the best of both worlds: fast, consistent and accurate kernel predictions. The implementation may be as simple as substituting the random kernel initialization in SelfDeblur by the proposed kernel prediction network.

Appendix A

Intuitive Derivation of a MAP_k Network

As a reminder, a common blind deblurring formulation is to estimate the kernel and marginalize over all sharp images, i.e. to maximize the posterior distribution MAP_k

$$\arg \max_k p(k|y) = \arg \max_k \int p(y|x,k)p(x)p(k) dx. \quad (\text{A.1})$$

In other words there is a probability to encounter some pair x, k , given the blurry image y that we have. Maximizing that probability means adjusting the pair x, k to have a better chance of creating the blurry image (through the convolution image formation model). This task is broken down by Bayes' theorem into three probabilities: First the likelihood to get the blurry image, given the *known* sharp image and kernel. It may seem pointless to model how the blurry image is created, if we already have x and k , but this term represents not simply the convolution model, but also captures the noise model that is imposed on y in parallel to the convolution of x and k . Then we have the second and third probabilities, that when maximized, yield a sharp image and blur kernel (respectively) that are likely to be found. For instance, there may be images that fit the convolution model, but have a lot of noise in them, so this second term aims to ensure that “natural” images are optimized for, e.g. those that do not commonly have very different colors from one pixel to the next. For kernels, prior knowledge can also be included, e.g. one kernel that always works with the convolutional model in eq. (1.1) is the delta (also called identity/no-blur) kernel $k = \delta$ (with $x = y$), however such a kernel is hardly ever the correct solution. Further properties of probable kernels can be included in this third term as well, depending on the desired group of blur types (e.g. motion and Gaussian kernels).

The following section will attempt to build up those three MAP probability terms, starting from a neural network implementation of eq. (A.1). At first glance, a kernel prediction network neither has sharp image priors, nor a convolution model, so assume they are not implemented and remove those terms from eq. (A.1)

$$\arg \max_k p(k|y) = \arg \max_k p(k).$$

This term $p(k)$ does not depend on the blurry input though. This could at best produce constant

predictions, so we can try adding a conditional on y

$$\arg \max_k p(k|y) = \arg \max_k p(k|y).$$

Here we failed to break down the problem entirely. So a successful kernel prediction network clearly has to be more capable than this initial assumption.

We know it has a kernel prior $p(k)$, but it stands to reason it also has an image prior $p(x)$, so that it can differentiate the two. Then it can “pull apart” the two components. This leaves us with the intermediate equation

$$\arg \max_k p(k|y) = \arg \max_k p(x)p(k),$$

which has two components of the original MAP_k equation. But still some term is missing since, as we already established, this problem cannot simply be solved with just general priors that do not factor in y .

Ignoring the convolution model for a moment, there is a simple case where it is actually easy to figure out the kernel from an image. Identify a point source in the blurry image, i.e. a small dot in the sharp image with a flat-colored background, and then it will trace out the (flipped) kernel in the blurry image. This case is e.g. found in the case of astronomical imaging of stars, which are indeed point sources in front of a dark background [LSW⁺19]. A network architecture without explicit knowledge about convolution could use this fact to gradually (with each layer) filter out the background, so that only the kernel remains at the end.

If we try to define what “pull apart” means, we find that it basically requires learning the convolutional model, or at least an approximation of a generalized form of it. A different way of imprinting kernels on images would be e.g. to simply overlay the kernel with some stride over the sharp image. This can also in principle be solved by the network, since it is very similar to the simple point-source case. So we know that even the same architecture could solve convolution and other methods of merging images and kernels. While this shows that a convolutional model is not required, we have to submit that a generalized form of it has to be learned, so that image and kernel can be separated at all. So we add back the image formation term

$$\arg \max_k p(k|y) = \arg \max_k \int p(y|x,k)p(x)p(k) dx.$$

Thus we arrive back at the start of eq. (A.1), meaning that the kernel prediction network, at least in one specific case (i.e. an ideal point source), learns a form of a convolution model.

Note that this first probability term (that can encode knowledge about the convolutional model) is general enough, so that it may also be implemented with more general image/kernel merging methods. This will never translate into an actual explicitly encoded convolution, as it does in the classical MAP_k sense, but it can be expected that at least an approximation of the convolutional model has to be learned.

Appendix B

Crop Architecture

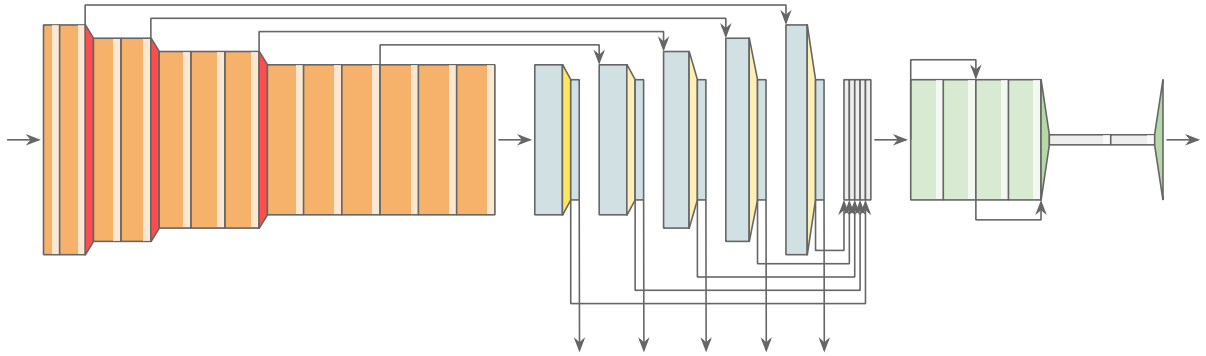


Figure B.1. **Crop architecture:** Shown is a base network (orange), followed by side outputs (gray) and novel cropping layers (yellow), and some residual blocks (green) and linear layers. Instead of reconstructing a kernel from parameters, this architecture attempts to crop a kernel-sized patch in the input and filter it.

This shows a very different architecture idea, that was much less successful (the best test set score was 19 dB PSNR, and on BSD-Levin 17 dB). As stated in chapter 4, the proposed architecture has an encoder-decoder network with a bottleneck. Take as a calibration image as an example, that contains a black patch with a white point. In such an image, the most efficient kernel estimation would merely crop in on this patch, which can be directly output as the kernel. This would avoid having to encode and decode the kernel, which is prone to lose details. In a more general case, in principle it is still possible to find the most point-like source in an image and *filter* some trace of a kernel by removing the background. This architecture is inspired by this idea. Instead of downsampling and upsampling, all this architecture does is crop, while retaining the original image scale. Differentiable cropping is not trivial, but spatial transformer networks have already laid the foundation for this. Here only a subset of the spatial transformation is needed, namely translation, and scaling to a predetermined size.

The data flow in this architecture is non-linear. First the base network processes the blurry image as usual, except that one more pooling layer is removed from the end. Initial experiments showed that

learning where to crop without supervision is very challenging. To alleviate this, the basic idea with this architecture was to crop at the same relative position in the features, at all resolutions. This way, all of the base network layers can flow into one layer that decides the crop position. After the base network (orange), the uncropped features of the five base network stages are fed into convolutional layers that keep the resolution and just output a fixed channel count, here 16. Then a novel hotspot crop layer (dark yellow) decides the crop position simply based on the brightest kernel-sized feature window, emulating a soft argmax. The motivation behind this is that a bright features are likely to be point sources, and the crop layer hopefully finds the best point source. The light yellow layers simply crop at the same relative feature position that was just determined, and upscale features to the kernel support size. Now each of the five stages has two outputs that have the spatial size of the output kernel support. The primary outputs are all concatenated and fed into a short network with residual blocks (green). As a final step they are flattened and put through two linear layers, and unflattened again, giving the kernel output. The secondary outputs of the stages are first reduced to a single channel (thin grey blocks), and represent side outputs (downward arrows). This makes them usable for deep supervision, i.e. the loss can be applied not just to the main output, but intermediate outputs as well. The motivation behind this is that each stage should already try to produce the clearest kernel output possible, only filtering out more with each stage.

To summarise, this architecture could not learn to find good point sources, as the crop positions did not vary greatly. The general idea was to first find a good window, then filter image background out from the kernel. Future work may find a method that can successfully locate good patches to estimate a kernel from, and benefit from retaining the full feature resolution to restore much more accurate kernels. To the best of my knowledge, an architecture that crops instead of downsamples features has not been done before. Another takeaway from this is that architectures that heavily specialize on kernel estimation are conceivable.

References

- [ADF19] J  r  my Anger, Mauricio Delbracio, and Gabriele Facciolo. Efficient blind deblurring under high noise levels. In *2019 11th International Symposium on Image and Signal Processing and Analysis (ISPA)*, pages 123–128. IEEE, 2019.
- [AKB⁺20] Chirag Agarwal, Shahin Khobahi, Arindam Bose, Mojtaba Soltanalian, and Dan Schonfeld. Deep-url: A model-aware approach to blind deconvolution based on deep unfolded richardson-lucy network. In *2020 IEEE International Conference on Image Processing (ICIP)*, pages 3299–3303. IEEE, 2020.
- [AMFM11] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(5):898–916, May 2011.
- [BDSS21] Andrew Brock, Soham De, Samuel L Smith, and Karen Simonyan. High-performance large-scale image recognition without normalization. *arXiv preprint arXiv:2102.06171*, 2021.
- [BF12] Giacomo Boracchi and Alessandro Foi. Modeling the performance of image restoration from motion blur. *IEEE Transactions on Image Processing*, 21(8):3502–3517, 2012.
- [Cha16] Ayan Chakrabarti. A neural approach to blind motion deblurring. In *European conference on computer vision*, pages 221–235. Springer, 2016.
- [DMWS20] Keyan Ding, Kede Ma, Shiqi Wang, and Eero P Simoncelli. Image quality assessment: Unifying structure and texture similarity. *arXiv preprint arXiv:2004.07728*, 2020.
- [DRS20] Jiangxin Dong, Stefan Roth, and Bernt Schiele. Deep wiener deconvolution: Wiener meets deep learning for image deblurring. *Advances in Neural Information Processing Systems*, 33, 2020.
- [FBPW95] DA Fish, AM Brinicombe, ER Pike, and JG Walker. Blind deconvolution by means of the richardson-lucy algorithm. *JOSA A*, 12(1):58–65, 1995.

- [FSH⁺06] Rob Fergus, Barun Singh, Aaron Hertzmann, Sam T Roweis, and William T Freeman. Removing camera shake from a single photograph. In *ACM SIGGRAPH 2006 Papers*, pages 787–794. Association for Computing Machinery, 2006.
- [GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <https://www.deeplearningbook.org>.
- [Get12] Pascal Getreuer. Total variation deconvolution using split bregman. *Image Processing On Line*, 2:158–174, 2012.
- [GLZD19] Jinjin Gu, Hannan Lu, Wangmeng Zuo, and Chao Dong. Blind super-resolution with iterative kernel correction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1604–1613, 2019.
- [GZS⁺20] Dong Gong, Zhen Zhang, Qinfeng Shi, Anton van den Hengel, Chunhua Shen, and Yanning Zhang. Learning deep gradient descent optimization for image deconvolution. *IEEE transactions on neural networks and learning systems*, 31(12):5468–5482, 2020.
- [HF96] Rob J Hyndman and Yanan Fan. Sample quantiles in statistical packages. *The American Statistician*, 50(4):361–365, 1996.
- [HKZŠ15] Michal Hradiš, Jan Kotera, Pavel Zemčík, and Filip Šroubek. Convolutional neural networks for direct text deblurring. In *Proceedings of BMVC*, volume 10, 2015.
- [Hun71] B Hunt. A matrix theory proof of the discrete convolution theorem. *IEEE Transactions on Audio and Electroacoustics*, 19(4):285–288, 1971.
- [IPG⁺18] Pavel Izmailov, Dmitrii Podoprikin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.
- [JSK08] Neel Joshi, Richard Szeliski, and David J Kriegman. Psf estimation using sharp edge prediction. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2008.
- [KBM⁺18] Orest Kupyn, Volodymyr Budzan, Mykola Mykhailych, Dmytro Mishkin, and Jiří Matas. Deblurgan: Blind motion deblurring using conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8183–8192, 2018.
- [KF09] Dilip Krishnan and Rob Fergus. Fast image deconvolution using hyper-laplacian priors. *Advances in neural information processing systems*, 22:1033–1041, 2009.

- [KRS17] Jakob Kruse, Carsten Rother, and Uwe Schmidt. Learning to push the limits of efficient fft-based image deconvolution. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4586–4594, 2017.
- [KZP19] Sergey Kastryulin, Djamil Zakirov, and Denis Prokopenko. PyTorch Image Quality: Metrics and measure for image quality assessment, 2019. Open-source software available at <https://github.com/photosynthesis-team/piq>.
- [LFDF07a] Anat Levin, Rob Fergus, Fredo Durand, and William T Freeman. Deconvolution using natural image priors. *Massachusetts Institute of Technology, Computer Science and Artificial Intelligence Laboratory*, 3, 2007.
- [LFDF07b] Anat Levin, Rob Fergus, Frédo Durand, and William T Freeman. Image and depth from a conventional camera with a coded aperture. *ACM transactions on graphics (TOG)*, 26(3):70–es, 2007.
- [LH17] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.
- [LHZ⁺20] Yade Li, Weidong Hu, Xin Zhang, Zhihao Xu, Jiaqi Ni, and Leo P Ligthart. Adaptive terahertz image super-resolution with adjustable convolutional neural network. *Optics Express*, 28(15):22200–22217, 2020.
- [LLCL19] Junyong Lee, Sungkil Lee, Sunghyun Cho, and Seungyong Lee. Deep defocus map estimation using domain adaptation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12222–12230, 2019.
- [LPL⁺18] Lerenhan Li, Jinshan Pan, Wei-Sheng Lai, Changxin Gao, Nong Sang, and Ming-Hsuan Yang. Learning a discriminative prior for blind image deblurring. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6616–6625, 2018.
- [LSW⁺19] Ma Long, Yang Soubo, Ni Weiping, Xiao Feng, and Yu Jun. Point-spread function estimation for adaptive optics imaging of astronomical extended objects. *The Astrophysical Journal*, 888(1):20, 2019.
- [LTME19] Yuelong Li, Mohammad Tofighi, Vishal Monga, and Yonina C Eldar. An algorithm unrolling approach to deep image deblurring. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7675–7679. IEEE, 2019.
- [LWDF09] Anat Levin, Yair Weiss, Fredo Durand, and William T Freeman. Understanding and evaluating blind deconvolution algorithms. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1964–1971. IEEE, 2009.

- [LWY⁺19] Zhenyu Long, Tianyi Wang, ChengWu You, Zhengang Yang, Kejia Wang, and Jinsong Liu. Terahertz image super-resolution based on a deep convolutional neural network. *Applied optics*, 58(10):2731–2735, 2019.
- [MI13] Tomer Michaeli and Michal Irani. Nonparametric blind super-resolution. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 945–952, 2013.
- [MJU17] Michael T McCann, Kyong Hwan Jin, and Michael Unser. A review of convolutional neural networks for inverse problems in imaging. *arXiv preprint arXiv:1710.04011*, 2017.
- [NHKML17] Seungjun Nah, Tae Hyun Kim, and Kyoung Mu Lee. Deep multi-scale convolutional neural network for dynamic scene deblurring. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3883–3891, 2017.
- [NJ20] Yuesong Nan and Hui Ji. Deep learning for handling kernel/model uncertainty in image deconvolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2388–2397, 2020.
- [OGR10] François Orieux, Jean-François Giovannelli, and Thomas Rodet. Bayesian estimation of regularization and point spread function parameters for wiener–hunt deconvolution. *JOSA A*, 27(7):1593–1607, 2010.
- [PF15] Daniele Perrone and Paolo Favaro. A clearer picture of total variation blind deconvolution. *IEEE transactions on pattern analysis and machine intelligence*, 38(6):1041–1055, 2015.
- [RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.
- [RZM⁺18] Wenqi Ren, Jiawei Zhang, Lin Ma, Jinshan Pan, Xiaochun Cao, Wangmeng Zuo, Wei Liu, and Ming-Hsuan Yang. Deep non-blind deconvolution via generalized low-rank approximation. *Advances in neural information processing systems*, pages 297–307, 2018.
- [RZW⁺20] Dongwei Ren, Kai Zhang, Qilong Wang, Qinghua Hu, and Wangmeng Zuo. Neural blind deconvolution using deep priors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3341–3350, 2020.
- [SCH⁺16] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings*

- of the IEEE conference on computer vision and pattern recognition*, pages 1874–1883, 2016.
- [SCWH13] Libin Sun, Sunghyun Cho, Jue Wang, and James Hays. Edge-based blur kernel estimation using patch priors. In *IEEE International Conference on Computational Photography (ICCP)*, pages 1–8. IEEE, 2013.
- [SGW⁺19] Wen-Ze Shao, Qi Ge, Li-Qian Wang, Yun-Zhi Lin, Hai-Song Deng, and Hai-Bo Li. Nonparametric blind super-resolution using adaptive heavy-tailed priors. *Journal of Mathematical Imaging and Vision*, 61(6):885–917, 2019.
- [STY17] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *International Conference on Machine Learning*, pages 3319–3328. PMLR, 2017.
- [TGS⁺18] Xin Tao, Hongyun Gao, Xiaoyong Shen, Jue Wang, and Jiaya Jia. Scale-recurrent network for deep image deblurring. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8174–8182, 2018.
- [Tuk52] John W Tukey. The extrapolation, interpolation and smoothing of stationary time series with engineering applications., 1952.
- [WYL⁺20] Fei Wen, Rendong Ying, Yipeng Liu, Peilin Liu, and Trieu-Kien Truong. A simple local minimal intensity prior and an improved algorithm for blind image deblurring. *IEEE Transactions on Circuits and Systems for Video Technology*, 2020.
- [WZ14] David Wipf and Haichao Zhang. Revisiting bayesian blind deconvolution. *Journal of Machine Learning Research (JMLR)*, 15(1):3595–3634, January 2014.
- [XJ10] Li Xu and Jiaya Jia. Two-phase kernel estimation for robust motion deblurring. In *European conference on computer vision*, pages 157–170. Springer, 2010.
- [XPZY17] Xiangyu Xu, Jinshan Pan, Yu-Jin Zhang, and Ming-Hsuan Yang. Motion blur kernel estimation via deep learning. *IEEE Transactions on Image Processing*, 27(1):194–205, 2017.
- [YLC20] Minyuan Ye, Dong Lyu, and Gengsheng Chen. Scale-iterative upscaling network for image deblurring. *IEEE Access*, 8:18316–18325, 2020.
- [YMY14] Chih-Yuan Yang, Chao Ma, and Ming-Hsuan Yang. Single-image super-resolution: A benchmark. In *European conference on computer vision*, pages 372–386. Springer, 2014.
- [YS16] Ruomei Yan and Ling Shao. Blind image blur estimation via deep learning. *IEEE Transactions on Image Processing*, 25(4):1910–1921, 2016.

-
- [ZDLK19] Hongguang Zhang, Yuchao Dai, Hongdong Li, and Piotr Koniusz. Deep stacked hierarchical multi-patch network for image deblurring. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5978–5986, 2019.
- [Zha19] Richard Zhang. Making convolutional networks shift-invariant again. In *International Conference on Machine Learning*, pages 7324–7334. PMLR, 2019.
- [ZIE⁺18] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018.
- [ZW11] Daniel Zoran and Yair Weiss. From learning models of natural image patches to whole image restoration. In *2011 International Conference on Computer Vision*, pages 479–486. IEEE, 2011.