Blur Invariant Feature Learning for Deep Blind Deblurring

A Project Report

submitted by

AKASH KUMAR SINGH

in partial fulfilment of requirements
for the award of the dual degree of

BACHELOR OF TECHNOLOGY AND MASTER OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS

MAY 2017

THESIS CERTIFICATE

This is to certify that the thesis titled **Blur Invariant Feature Learning for Deep Blind Deblurring**, submitted by **Akash Kumar Singh**, to the Indian Institute of Technology, Madras, for the award of the degree of **Master of Technology**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

A N Rajagopalan

Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: Chennai

Date: 10th April 2017

ACKNOWLEDGEMENTS

I take this opportunity to express my deepest gratitude to my project guide Dr. A.N. Rajagopalan for his valuable guidance and motivation throughout the project. I am very grateful to him for providing his valuable time to guide me during the project. My sincere thanks to Nimisha , Green Rosh and all my lab mates who shared their experience and helped me whenever faced with an obstacle.

It is a privilege to be a student in IIT Madras. I express special thanks to all my teachers for all the academic insight obtained from them. I also acknowledge the excellent facilities provided by the institute to the students. I would continue to extend my gratitude to all friends in the institute to make my stay in IIT Madras the most memorable time of my life.

ABSTRACT

KEYWORDS: Blind Deblurring; Autoencoder; Dictionary Learning.

In this work, we investigate deep neural networks for blind motion deblurring. Instead of regressing for the motion blur kernel and performing non-blind deblurring outside of the network (as most methods do), we propose a compact and elegant end-to-end deblurring network. Inspired by the data-driven sparse-coding approaches that are capable of capturing linear dependencies in data, we generalize this notion by embedding non-linearities into the learning process. We propose a new architecture for blind motion deblurring that consists of an autoencoder that learns the data prior, and an adversarial network that attempts to generate and discriminate between clean and blurred features. Once the network is trained, the generator learns a blur-invariant data representation which when fed through the decoder results in the final deblurred output. We also compare our work with recent similar deep network based deblurring techniques and analyze effects of various network hyper-parameters. Furthermore, We analyze effects of cost functions like mean square error, gradient loss, adversarial loss and perceptual loss functions for our task of blind deblurring.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

PSNR	Peak Signal to Noise Ratio
PSF	Point Spread Function
CNN	Convolutional Neural Network
FFT	Fast Fourier Transform
GPU	Graphics Processing Unit
TV	Total Variation
GAN	Generative Adversarial Network
KLD	Kullback-Leibler Divergence
MMD	Maximum Mean Discrepancy
JSD	Jensen-Shannon Divergence
ReLU	Rectified Linear Unit
MSSIM	Mean Structural Similarity
MSE	Mean Squared Error
ResNet	Residual Network

NOTATION

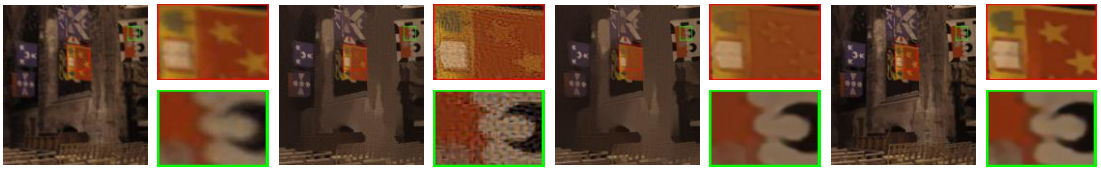
$\mathcal{G}$	Generator
$\mathcal{D}$	Discriminator
$\mathcal{D}e$	Decoder
$\mathcal{E}$	Encoder
P_d	Probability distribution of training data
P_g	Probability distribution of Generated data
x	Input vector
z	Random noise vector
y	Output or generated data
adv	Adversarial

CHAPTER 1

INTRODUCTION

Motion blur is an inevitable phenomenon caused due to relative motion between camera and scene/object while capturing, especially under long exposure times and fast moving object. In general, motion blur can be caused by camera shake, scene depth as well as multiple object motion. Motion blur in these scenarios can be reduced by changing camera settings like reducing exposure time but this comes in compromise with other unwanted image degradation effects like noise, low light in the image. With the increasing use of handheld imaging devices, there is an increasing need to invert the blurring process to recover the underlying clean image. However, it is well-known that deblurring is ill-posed as both the image and the blur are unknown. Due to this many methods exist Zhang and Carin (2014) that rely on information from multiple frames captured using video or burst mode and work by harnessing the information from these frames to solve for the underlying original (latent) image. But most often we don't have this extra information from multiple frames and and deblurring has to be done with just a single image which is considerably more challenging as the blur kernel as well as the latent image must be estimated from just one observation. It is this problem that we attempt to solve here.

Early works Cho and Lee (2009); Shan *et al.* (2008); Levin *et al.* (2011) assumed space-invariant blur and iteratively solved for the latent image and blur kernel. Although these convolutional models are simple and straight forward to analyze using FFTs, they fail to account for space-variant blur caused by non-linear camera motion or dynamic objects or depth-varying scenes. Nevertheless, even in such situations local patch-wise convolutional model can be employed to achieve deblurring. Instead of using a patch-wise model, works such as Whyte *et al.* (2012); Joshi *et al.* (2008) take the space-variant blur formation model itself into consideration. But the deblurring process becomes highly ill-posed as it must now estimate blur kernel at each pixel position along with the underlying image intensities. For planar scenes or under pure camera rotations, the methods in Whyte *et al.* (2012); Gupta *et al.* (2010) circumvent this issue by modeling the global camera motion using homographies.



(a) (b) (c) (d)

Figure 1.1: Effect of image prior. (a) Input blurred image. (b-d) Deblurred result using Pan *et al.* (2016) with ($\lambda = 0.04$, $\mu = 0.007$, $\lambda_{TV} = 0.001$), ($\lambda = 0.001$, $\mu = 0.01$, $\lambda_{TV} = 0.01$) and ($\lambda = 0.004$, $\mu = 0.005$, $\lambda_{TV} = 0.01$), respectively.

Major efforts have also gone into designing priors that are apt for the underlying clean image and the blur kernel to regularize the inversion process and ensure convergence during optimization. The most widely used priors are total variational regularizer Chan and Wong (1998); Perrone and Favaro (2014), sparsity prior on image gradients, l_1/l_2 image regularization Krishnan *et al.* (2011), the unnatural l_0 prior Xu *et al.* (2013) and the very recent dark channel prior Pan *et al.* (2016) for images. Even though such prior-based optimization schemes have shown promise, the extent to which a prior is able to perform under general conditions is questionable Krishnan *et al.* (2011). Some priors (such as the sparsity prior on image gradient) even tend to favor blurry results Levin *et al.* (2011). In a majority of situations, the final result requires a judicious selection of the prior, its weightage, as well as tuning of other parameters. Depending on the amount of blur, these values need to be adjusted so as to strike the right balance between over-smoothening and ringing in the final result. Such an effect is depicted in Fig. 1.1. Note that the results fluctuate with the weightage selected for the prior. These results correspond to the method of Pan *et al.* (2016) with varying weights for dark channel prior (λ), l_0 prior (μ) and the TV prior (λ_{TV}). Furthermore, these methods are iterative and quite time-consuming.

Dictionary learning is a data-driven approach and has shown good success for image restoration tasks such as denoising, superresolution and deblurring Aharon *et al.* (2006); Yang *et al.* (2010). Research has shown that sparsity helps to capture higher-order correlations in data, and sparse codes are well-suited for natural images Mairal *et al.* (2009). Lou et al. sparsedeb have proposed a dictionary replacement technique for deblurring of images blurred with a Gaussian kernel of specific variance. The authors of Xiang *et al.* (2015) adopt this concept to learn a pair of dictionaries jointly from blurred as well as clean image patches with the constraint that the sparse code be invariant to

blur. They were able to show results for space-invariant motion deblurring but were again constrained to a single kernel. For multiple kernels, they learn different dictionaries and choose the one for which the reconstruction error is the least. Even though sparse coding models perform well in practice, they share a *shallow* linear structure and hence are limited in their ability to generalize to different types of blurs.

Recently, deep learning and generative networks have made forays into computer vision and image processing, and their influence and impact are growing rapidly by the day. Neural networks gained in popularity with the introduction of Alexnet Krizhevsky *et al.* (2012) that showed a huge reduction in classification error compared to traditional methods. Following this, many regression networks based on Convolutional Neural Networks (CNNs) were proposed for image restoration tasks. With increasing computational speeds provided by GPUs, researchers are investigating deep networks for the problem of blur inversion as well. Xu et al. Xu *et al.* (2014) proposed a deep deconvolutional network for non-blind single image deblurring (i.e, the kernel is fixed and known apriori). Schuler et al. Schuler *et al.* (2014) came up with a neural architecture that mimics traditional iterative deblurring approaches. Chakrabarti Chakrabarti (2016) trained a patch-based neural network to estimate the kernel at each patch and employed a traditional non-blind deblurring method in the final step to arrive at the deblurred result. Since these methods estimate a single kernel for the entire image, they work for the space-invariant case alone. The most relevant work to handle space-variant blur is a method based on CNN for patch-level classification of the blur type Sun *et al.* (2015), which focuses on estimating the blur kernel at all locations from a single observation. They parametrize the kernels (using length and angle) and estimate these parameters at each patch using a trained network. However, such a parametric model is too restrictive to handle general camera motion blur.

The above-mentioned methods attempt to estimate the blur kernel using a deep network but finally perform non-blind deblurring exterior to the network to get the deblurred result. Any error in the kernel estimate (due to poor edge content, saturation or noise in the image) will impact deblurring quality. Moreover, the final non-blind deblurring step typically assumes a prior (such as sparsity on the gradient of latent image) which again necessitates a judicious selection of prior weightage; else the deblurred result will be imperfect as already discussed (Fig. 1.1). Hence, kernel-free approaches are very much desirable.

In this work, we propose a deep network that can perform single image blind-deblurring without the cumbersome need for prior modeling and regularization. The core idea is to arrive at a blur-invariant representation learned using deep networks that facilitates end-to-end deblurring. Performance-wise, our method is at par with conventional methods which use regularized optimization, and outperforms deep network-based methods. While conventional methods can only handle specific types of space-variant blur such as blur due to camera motion or object motion or scene with depth variations, our network does not suffer from these limitations. Most importantly, the run-time for our method is very small compared to conventional methods. The key strength of our network is that it does end-to-end deblurring with performance quality at par or better than competing methods while being computationally efficient.

CHAPTER 2

Blur-invariant Feature Learning

2.1 Blur-invariant Feature Learning

It is well-known that most sensory data, including natural images, can be described as a superposition of small number of atoms such as edges and surfaces Mairal *et al.* (2009). Dictionary-based methods exploit this information and learn the atoms that can represent data in sparse forms for various image restoration tasks (including deblurring). With an added condition that these representations should be invariant to the blur content in the image, dictionary methods have performed deblurring by learning coupled dictionaries Xiang *et al.* (2015). However, constrained by the fact that dictionaries can capture only linearities in the data and blurring process involves non-linearities (high frequencies are suppressed more), their deblurring performance does not generalize across blurs.

In this work, we extend the notion of blur-invariant representation to deep networks that can capture non-linearities in the data. We are not the first one to approach deep learning as a generalization of dictionary learning for sparse coding. The work in Xie *et al.* (2012) combines sparse coding and denoising encoders for the task of denoising and inpainiting. Deep neural networks, in general, have yielded good improvements over conventional methods for various low-level image restoration problems including super-resolution Dong *et al.* (2016), inpainting and denoising Xie *et al.* (2012); Pathak *et al.* (2016). These networks are learned end-to-end by exposing them to lots of example-data from which the network learns the mapping to undo distortions. We investigate the possibility of such a deep network for the task of single image motion deblurring.

For blind-deblurring, we first require a good feature representation that can capture image-domain information. Autoencoders have shown great success in unsupervised learning by encoding data to a compact form Hinton and Salakhutdinov (2006) which can be used for classification tasks. This motivated us to train an autoencoder on clean

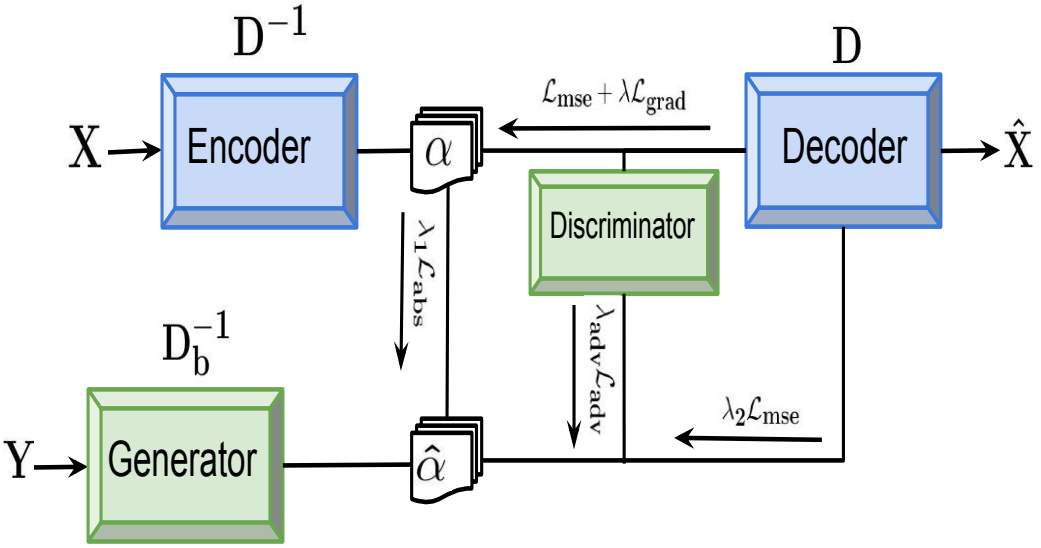


Figure 2.1: Illustration of our architecture.

image patches for learning the feature representation. Once a good representation is learned for clean patches, the next step is to produce a blur-invariant representation (as in Xiang *et al.* (2015)) from blurred data. We propose to use a generative adversarial network (GAN) for this purpose which involves training of a generator and discriminator that attempt to compete with each other. The purpose of the generator is to confuse the discriminator by producing clean features from blurred data that are similar to the ones produced by the autoencoder so as to achieve blur-invariance. The discriminator, on the other hand, tries to beat the generator by identifying the clean and blurred features.

A schematic of our proposed architecture is shown in Fig. 2.1. Akin to dictionary learning that represents any data X as a sparse linear combination of dictionary atoms D i.e, $X = D\alpha$, our encoder-decoder module performs this in non-linear space. Hence, the encoder can be thought of as an inverse dictionary D^{-1} that projects the incoming data into a sparse representation. The decoder acts as the dictionary D that reconstructs the input from the sparse representation. Generator training can be treated as learning the blur dictionary that can project the blurred data Y into the same sparse representation of X i.e, $\alpha = D^{-1}X = D_b^{-1}Y$. Once training is done, the input blurry image (Y) is passed through the generator to get a blur-invariant feature which when projected to the decoder yields the deblurred result as $\hat{X} = D\alpha = DD_b^{-1}Y$.

Thus, by associating the feature representation learned by the autoencoder with GAN training, our model is able to perform single image blind deblurring in an end-to-end manner. Ours is a kernel-free approach and does away with the tedious task of selecting priors, a serious bottleneck of conventional methods. Unlike other deep learning methods, our network directly regresses for the clean image.

2.2 Summary of Contributions

This is a collaborative work done by me and one of my lab mate. For completeness entire work is mentioned in this report. My contributions to this work as follows :

- Included residual blocks in the autoencoder (Section 3.1) and generator architecture (Section 3.2) which helped in implementing deeper networks. Including residual blocks increased the $PSNR$ of reconstructed output of autoencoder by $15dB - 20dB$.
- Implemented codes of various networks including the ones described in section 4.1 with different loss functions in torch and analyzed effects of hyper-parameters like weights given to loss functions, batch size etc before arriving at the final architecture and parameter setting.

CHAPTER 3

Network Architecture

3.1 Network Architecture

Our network consists of an autoencoder that learns the clean image domain and a generative adversarial network that generates blur-invariant features. We train our network in two stages. We first train an autoencoder to learn the clean image manifold. This is followed by the training of a generator that can produce clean features from a blurred image. This is then fed to the decoder to get the deblurred output. Note that instead of combining the task of data-representation and deblurring into a single network, we relegate the task of data-learning to the autoencoder and use this information to guide image deblurring. Details of the architecture and the training procedure are explained next.

3.1.1 Encoder-Decoder

Autoencoders were originally proposed for the purpose of unsupervised learning Hinton and Salakhutdinov (2006) and have since been extended to a variety of applications. An autoencoder projects the input data into a low-dimensional space and recovers the input from this representation. When not modeled properly, it is likely that the autoencoder learns to just compress the data without learning any useful representation. Denoising encoders Vincent *et al.* (2008) were proposed to overcome this issue by corrupting the data with noise and letting the network undo this effect and get back a clean output. This ensures that the autoencoder learns to correctly represent clean data. Deepak et al. Pathak *et al.* (2016) extended this idea from mere data representation to context representation for the task of inpainiting. In effect, it learns a meaningful representation that can capture domain information of data.

We investigated different architectures for the autoencoder and observed that including residual blocks (ResNet) He *et al.* (2016) helped in achieving faster convergence and

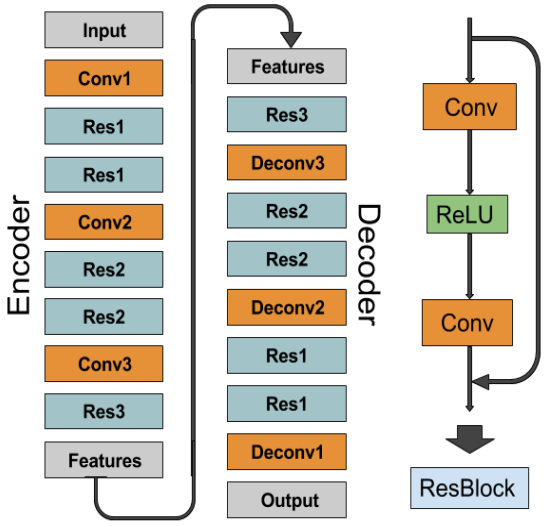


Figure 3.1: Autoencoder architecture with residual networks.

in improving the reconstructed output. Residual blocks help by by-passing the higher-level features to the output while avoiding the gradient vanishing problem. This way ResNets help in building deeper architecture for extracting finer features. The training data was corrupted with noise (30% of the time) to ensure encoder reliability and to avoid learning an identity map. The architecture used in our work along with the ResNet block is shown in Fig. 3.4. A detailed description of the filter and feature map sizes along with the stride values used are as given below.

Encoder: $C^{5}_{3\to 8} \downarrow 2 \to R^{5(2)}_{8} \to C^{5}_{8\to 16} \downarrow 2 \to R^{5(2)}_{16} \to C^{3}_{16\to 32} \downarrow 2 \to R^{3}_{32}$

Decoder: $R^{3}_{32} \to C^{2}_{32\to 16} \uparrow 2 \to R^{5(2)}_{16} \to C^{4}_{16\to 8} \uparrow 2 \to R^{5(2)}_{8} \to C^{4}_{8\to 3} \uparrow 2$

where $C^{c}_{a\to b} \downarrow d$ represents convolution mapping from a feature dimension of a to b with a stride of d and filter size of c, $\downarrow$ represents down-convolution, $\uparrow$ stands for up-convolution, and $R^{b(c)}_{a}$ represents the residual block which consists of a convolution and a ReLU block with output feature size a. Filter sizes b and c represent the number of repetitions of residual blocks.

Fig. 3.2 shows the advantage of the ResNet block. Fig. 3.2(a) is the target image and Figs. 3.2(b) and (c) are the output of autoencoders with and without ResNet block for the same number of iterations. Note that the one with ResNet converges faster and preserves the edges due to skip connections that pass on the information to deeper layers.

(a) (b) PSNR= 29.5 (c) PSNR= 23.1

Figure 3.2: Effect of ResNet on reconstruction. (a) The target image. (b) Result of encoder module of Fig. 3.4. (c) Result obtained by removing ResNet for the same number of iterations. (Enlarge for better viewing).

3.2 GANs and Objective Functions of Generative Networks

Generative models can be used for wide range of vision applications like denoising, deblurring, inpainting, compression and various other supervised/unsupervised learning tasks. Given this huge range of application there exists a lot of heterogeneity in modeling and training of such models. Although different generative models are trained with different objective function, all of these are more or less related to log-likelihood, such as Kullback-Leibler Divergence (KLD) Shlens (2014), Maximum Mean Discrepancy (MMD) $\big($(Gretton *et al.*, 2007), (Li *et al.*, 2015)$\big)$, Jensen-Shannon Divergence (JSD) (Goodfellow *et al.*, 2014), contrastive divergence Hinton (2002). Brief discussion about these objective functions:

3.2.1 Maximum Mean Discrepancy (MMD)

Given, generated and data samples MMD answers the question weather $P_g = P_d$. MMD is a statistical hypothesis testing technique which leads to a simple objective that can be interpreted as matching all orders of statistics between the generated samples and data samples. MMD simply compares all the statistics of given generated samples $Y = \{y_j\}_{j=1}^{M}$ and data samples $X = \{x_j\}_{j=1}^{N}$. If the statistics are similar then the samples are likely to come from the same distribution. The following formula computes the mean squared difference between the statistics of both set of samples:

$$\mathcal{L}_{MMD^2} = \left|\left| \frac{1}{N}\sum_{i=1}^{N}\phi(x_i) - \frac{1}{M}\sum_{j=1}^{M}\phi(y_j) \right|\right|^2 \tag{3.1}$$

$$= \frac{1}{N^2}\sum_{i=1}^{N}\sum_{i'=1}^{N}\phi(x_i)^\top\phi(x_i') - \frac{2}{MN}\sum_{i=1}^{N}\sum_{j=1}^{M}\phi(x_i)^\top\phi(y_i) + \frac{1}{M^2}\sum_{j=1}^{M}\sum_{j'=1}^{M}\phi(y_i)^\top\phi(y_i') \tag{3.2}$$

Different choice of function ϕ leads to match different moments of statistics. It is shown that MMD is 0 if and only if P_g = P_d. As equation (3.4) only involves inner products between the functions ϕ and hence kernel trick can be applied which is computationally cheaper. The kernels like Gaussian kernel which is defined as $k(x, x') = exp(-\frac{1}{2\sigma}\|x - x'\|^2)$, implicitly lifts the data into infinite dimensional space. Minimizing MMD using these function is then equivalent to minimizing distance between all the moments of the two distributions. Generative convolutional neural networks trained via simple MMD cost function shows promising results. Some works combine MMD loss with other losses to boost up their performance.

3.2.2 Variational Autoencoders and Kullback-Leibler Divergence (KLD)

Traditional methods of generative modeling rely on maximizing likelihood or equivalently minimizing KLD between our unknown data distribution P_d and generator's distribution P_g. One such method is Variational autoencoder, which consists of two networks an encoder and a decoder. Encoder encodes the data sample x to a latent representation z. Decoder decodes z to map it back into the data space.

$$z \sim Enc(x) = q(z|x) \quad , \hat{x} \sim Dec(z) = p(x|z) \tag{3.3}$$

Variational autoencoder (VAE) try to minimize KLD between the generator distribution and the data distribution. The KLD quantifies the closeness between two probability distributions. The KLD between continuous distribution P_d and P_g is given by:

$$KL(P_d\|P_g) = \int_x P_d(x) log\left(\frac{P_d(x)}{P_g(x)}\right)dx \tag{3.4}$$

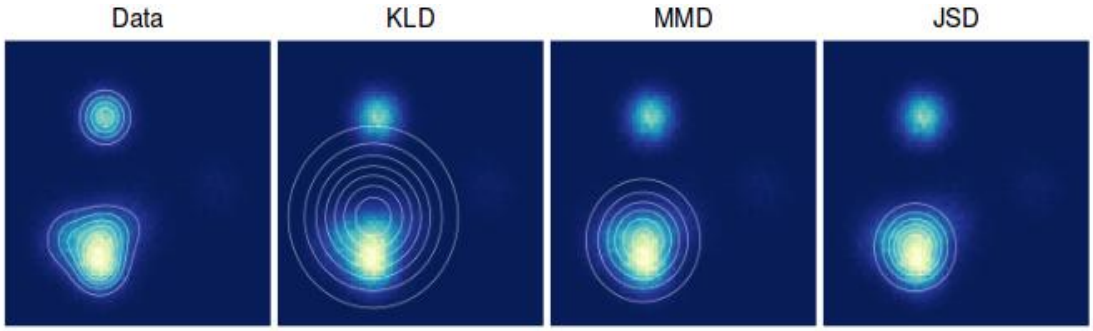


Figure 3.3: The above figure shows the three different fits to the data drawn from the mixture of Gaussians by either minimizing Kullback-Leibler Divergence (KLD), Maximum Mean Discrepancy (MMD) or Jensen-Shannon Divergence (JSD). It is clearly seen that using different distance measure between the distributions results in different fit on the data.

This cost function is non-negative $\big(KL(P_d\|P_g) \geq 0\big)$, non-symmetric in P_d and P_g and has a unique minimum at zero when P_g matches P_d exactly. For a training example x if $P_d(x) > P_g(x)$ then x has higher probability coming from data samples than a generated sample. It is important to note that when $P_d(x) > 0$ and $P_g(x) \to 0$ the integrand quickly grows to infinity, meaning this cost function assigns an extremely high cost to a generator's distribution for not covering parts of the data. But if $P_d(x) < P_g(x)$ then this cost pays extremely low cost to the samples when $P_d(x) \to 0$ and $P_g(x) > 0$. This means that this cost function will pay extremely low cost for generating fake samples. Clearly, if we would minimize $KL(P_g\|P_d)$ instead, the weighing of these errors would get reversed. The cost function would pay high cost for generating fake looking samples but it will pay extremely low cost for the generating not covering parts of data distribution.

3.2.3 GANs and Jensen-Shannon Divergence (JSD)

Jensen-Shannon Divergence (JSD) overcomes the difficulties of KLD as it minimizes the symmetric middle ground of the two cost functions $KL(P_g\|P_d)$ and $KL(P_d\|P_g)$. Hence, JSD is symmetrized and smoothed version of KLD. Formally, JSD can be written as:

$$JSD(P_d\|P_g) = \frac{1}{2}KL(P_d\|P_A) + \frac{1}{2}KL(P_g\|P_A) \tag{3.5}$$

Where P_A is the average distribution with density $\frac{P_d+P_g}{2}$.

Generative adversarial networks (GANs) have been shown to optimize an approximation to the JSD. It is indeed conjectured that the reason for success of GANs is due to the switch from traditional likelihood approaches. GANs were first introduced by Goodfellow Goodfellow *et al.* (2014) in 2014. Since then, they have been widely used for various image related tasks. GANs consists of two models: a Generator ($\mathcal{G}$) and a Discriminator ($\mathcal{D}$) which play a two-player mini-max game. $\mathcal{D}$ tries to discriminate between the samples generated by $\mathcal{G}$ and training data samples, while $\mathcal{G}$ tries to fool the discriminator by generating samples close to the actual data distribution for input random vector z. Goodfellow proved that there exists a unique solution to this, with $\mathcal{G}$ modeling the training data distribution and $\mathcal{D}$ equal to $\frac{1}{2}$ everywhere. The mini-max cost function Goodfellow *et al.* (2014) for training GANs is given by

$$\min_{\mathcal{G}}\max_{\mathcal{D}} C(\mathcal{G},\mathcal{D}) = E_{x\sim P_{data}(x)}[\log \mathcal{D}(x)] + E_{z\sim P_z(z)}[\log(1-\mathcal{D}(\mathcal{G}(z))] \tag{3.6}$$

where $D(x)$ is the probability assigned by the discriminator to the input x for discriminating x as a real sample. P_{data} and P_z are the respective probability distributions of data x and the input random vector z.

Minimizing equation 3.6 would result in minimizing the JSD between the generator's distribution and data distribution if the discriminator is optimal. So, In theory one would expect to first train the discriminator near to the optimum then optimize equation 3.6 so the cost function better approximates JSD. But this doesn't work, In practice the gradients from $\mathcal{D}$ required for updating parameters of $\mathcal{G}$ becomes close to zero for near to optimum discriminator. Hence, In practice the mini-max game between $\mathcal{G}$ and $\mathcal{D}$ is implemented iteratively. First k steps of $\mathcal{D}$ is optimized to minimize $\log \mathcal{D}(x) + \log(1-\mathcal{D}(\mathcal{G}(z))$ keeping $\mathcal{G}$ constant, which is followed by one step of $\mathcal{G}$ to minimizing $log(1-\mathcal{D}(\mathcal{G}(z))$ while keeping $\mathcal{D}$ constant.

The above technique also doesn't solve the problem completely, As in early learning $\mathcal{G}$ is poor and $\mathcal{D}$ can rejects samples produced by $\mathcal{G}$ with high confidence as they are clearly not from training data. Due to this $\mathcal{G}$ doesn't get sufficient gradients for updates as the term $log(1-\mathcal{D}(\mathcal{G}(z))$ of the cost function saturates. Hence, to avoid this gradient vanishing problem especially in the early learning Goodfellow suggested to train $\mathcal{G}$ to maximize $log(\mathcal{D}(\mathcal{G}(z))$ rather than minimizing $log(1-\mathcal{D}(\mathcal{G}(z))$. Figure 3.4 shows a more pedagogical explanation of the training procedure.

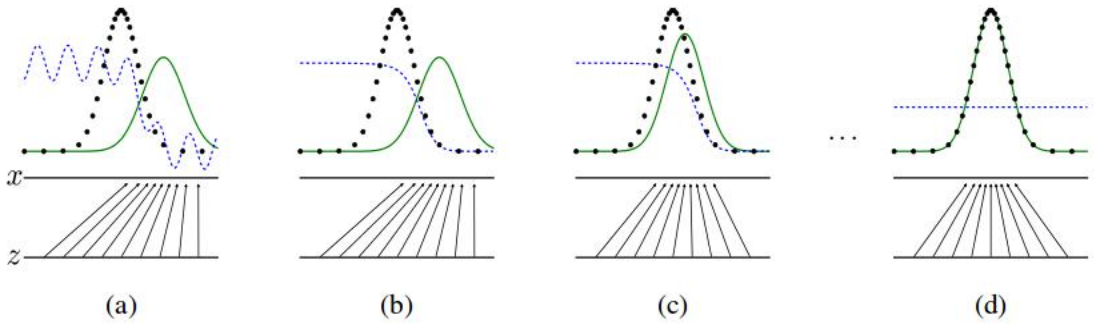


Figure 3.4: Above figure shows hows how the distributions of Generator (green line) and Discriminator (dotted blue line) changes as the GAN training proceeds to make Generator's distribution equal to data distribution. The lower horizontal line is the domain from which z is sampled, in this case uniformly. The horizontal line above is part of the domain of data x. The upward arrows shows how the mapping $x = \mathcal{G}(z)$ imposes the non-uniform distribution on transformed samples. As the training proceeds the gradients of $\mathcal{D}$ (significant in the transition region where $\mathcal{D}$ is less confident) helps shift the distribution of $\mathcal{G}$ towards data distribution.

GANs that just accept random noise and attempt to model the probability distribution of data over noise are difficult to train. Sometimes their instability leads to artifacts in the generated image. Hence, instead of a vanila network for GAN, we used conditional GAN which was introduced by Mirza et al. Mirza and Osindero (2014) and which enables GANs to accomodate extra information in the form of a conditional input. The inclusion of adversarial cost in the loss function has shown great promise Pathak *et al.* (2016), Isola *et al.* (2016). Training conditional GANs is a lot more stable than unconditional GANs due to the additional guiding input. The modified cost function Isola *et al.* (2016) is given by

$$\min_{\mathcal{G}} \max_{\mathcal{D}} C_{cond}(\mathcal{G}, \mathcal{D}) = E_{x,y\sim P_{data}(x,y)}[\log \mathcal{D}(x, y)] + E_{x\sim P_{data}(x),z\sim P_z(z)}[\log(1 - \mathcal{D}(x, \mathcal{G}(x, z))] \quad (3.7)$$

In our case y is the clean target feature, x is the conditional image (the blurred input), and z is the input random vector. In conditional GANs, the generator tries to model the distribution of data over the joint probability distribution of x and z. When trained without z for our task, the network learns a mapping for x to a deterministic output y which is the corresponding clean feature.

Isola *et al.* (2016) proposes an end-to-end network using a generative model to

(a) (b) (c)

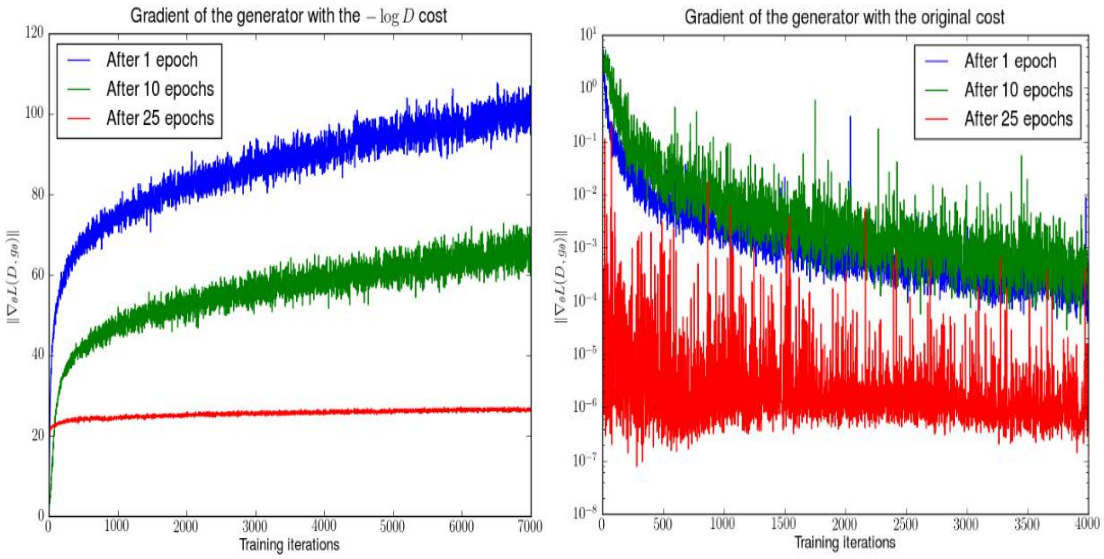
Figure 3.5: Effect of direct regression using generative networks. (a) Input blurred image. (b-c) Output of the network and the expected output.

perform image-to-image translation that can be used in multiple tasks. Following this recent trend, we initially attempted regressing directly to the clear pixels using off-the-shelf generative networks. However, we observed that this can lead to erroneous results as shown in Fig. 3.6. The main reason for this could be that the network becomes unstable when trained on higher-dimensional data. Also GANs are quite challenging to train and have mainly shown results for specific class of images. When trained for large diverse datasets, training does not converge Warde-Farley and Bengio (2017). Hence, we used the apriori-learned features of the autoencoder for training GAN.

Training a perfect discriminator requires it's weights to be updated simultaneously along with the generator such that it is able to discriminate between the generated samples and data samples. This task becomes easy and viable for the discriminator in the feature space for two reasons:
i) In this space, the distance between blurred features and clean features is higher as compared to the image space. This helps in faster training in the initial stage.
ii) The dimensionality of the feature-space is much lower as compared to that of image-space. GANs are known to be quite effective in matching distributions in lower-dimensional spaces Donahue *et al.* (2017).

Wasserstein GANs

Recently (Arjovsky and Bottou, 2017) showed that training GANs optimizing $\mathcal{G}$ by maximizing $log(\mathcal{D}(\mathcal{G}(z))$ rather than minimizing $log(1-\mathcal{D}(\mathcal{G}(z))$ is also not desirable. It was proved that as the $\mathcal{G}$ gets better, either we see vanishing gradients $log(1-\mathcal{D}(\mathcal{G}(z))$ when we optimize by minimizing $log(1-\mathcal{D}(\mathcal{G}(z))$ or the massively unstable behaviour



(a) (b)

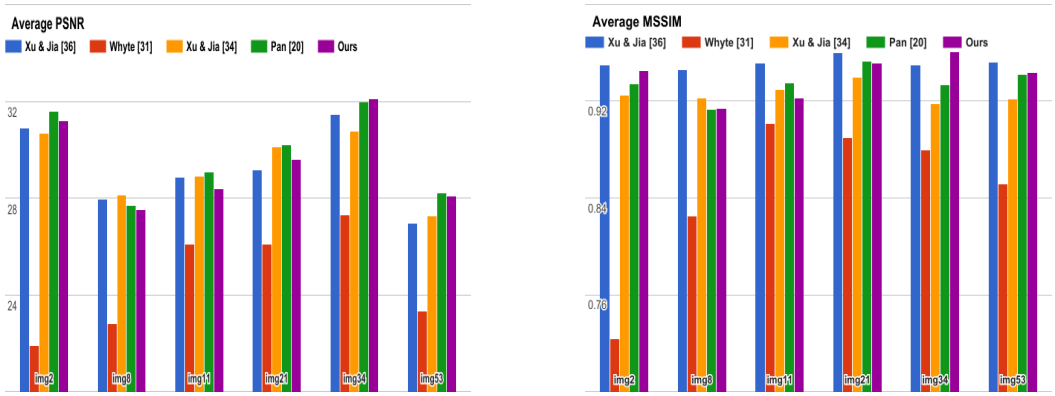
Figure 3.6: First a DCGAN was trained for 1, 10 and 25 epochs. Then with the generator fixed the discriminator was trained from scratch. (a) shows discriminator's norms quickly going to 0 when trained with $log(1 - \mathcal{D}(\mathcal{G}(z))$ cost function. (b) shows discriminator's norms quickly growing when trained with $log(1 - \mathcal{D}(\mathcal{G}(z))$ cost function. As gradients grows their variance increases and the they gradient update becomes noisy.

(noisy updates with $log(\mathcal{D}(\mathcal{G}(z))$) when we maximize $log(\mathcal{D}(\mathcal{G}(z))$. Figure 3.6 shows the discriminators error rates on logarithmic scale trained with these the two different criteria. (Arjovsky and Bottou, 2017) showed that this noisy and vanishing gradient behavior is due to the supports of P_d and P_g lying in different manifolds.

To overcome these issues (Arjovsky and Bottou, 2017) suggests to train GANs with a softer metric known as Wasserstein metric, Which he calls them Wasserstein GANs. In Wasserstein GANs, first we train the discriminator by assigning continuous labels to the generated samples and data samples. The labels can be random noise with different mean (for generated and data samples) and small variance. After training the $\mathcal{D}$ till close to optimum, the generator is trained with $log(1 - \mathcal{D}(\mathcal{G}(z))$ cost function. But this time the gradients will not be zero due to use of continuous labels in training of $\mathcal{D}$.

3.2.4 Loss function

We trained our network using the following loss functions. For autoencoder training, we used $\mathcal{L}_{\text{mse}} + \lambda\mathcal{L}_{\text{grad}}$. Adding the gradient-loss helps in preserving edges and re-



(a) (b)

Figure 3.7: Quantitative evaluation on dataset of Sun *et al.* (2013). (a) and (b) correspond to average PSNR and MSSIM values, respectively.

covering sharp images as compared to $\mathcal{L}_{\text{mse}}$ alone. We use normalized l_2 distance on the expected and observed image as our loss function i.e.

$$\mathcal{L}_{\text{mse}} = \|\mathcal{D}e(\mathcal{E}(I + \mathcal{N})) - I\|_2^2 \tag{3.8}$$

where $\mathcal{D}e$ is the decoder, $\mathcal{E}$ the encoder, N is noise and I is the target (clean) image. The MSE error captures overall image content but tends to prefer a blurry solution. Hence, training only with MSE loss results in loss of edge details. To overcome this, we used gradient loss as it favours edges as discussed in Mathieu *et al.* (2015) for video-prediction.

$$\mathcal{L}_{\text{grad}} = \|\nabla\mathcal{D}e(\mathcal{E}(I + \mathcal{N})) - \nabla I\|_2^2 \tag{3.9}$$

where ∇ is the gradient operator.

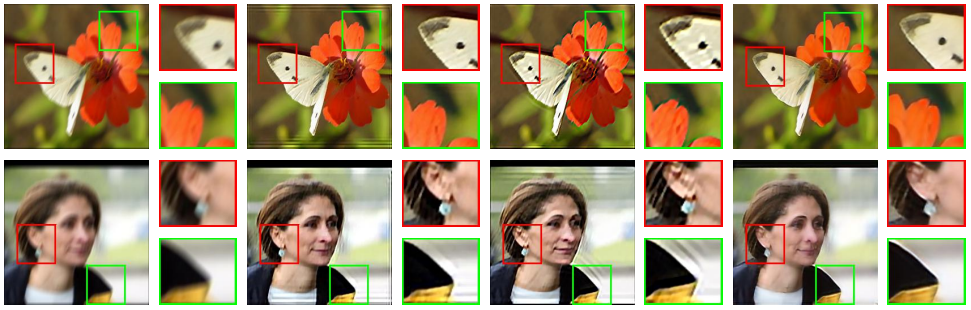
GAN is trained with the combined cost given by $\lambda_{\text{adv}}\mathcal{L}_{\text{adv}} + \lambda_1\mathcal{L}_{\text{abs}} + \lambda_2\mathcal{L}_{\text{mse}}$ in the image and feature space. Even though l_2 loss is simple and easy to back-propagate, it under performs on sparse data. Hence, we used l_1 loss for feature back-propagation i.e.

$$\mathcal{L}_{\text{abs}} = \|\mathcal{G}(B) - \mathcal{E}(I)\|_1 \tag{3.10}$$

where B is the blurred image. The adversarial loss function $\mathcal{L}_{\text{adv}}$ (given in Eq. (3.7)) requires that the samples output by the generator should be indistinguishable to the discriminator. This is a strong condition and forces the generator to produce samples that are close to the underlying data distribution. As a result, the generator outputs features

Method	Run time
Ours (Torch, GPU)	3.4 sec
Xu & Jia Xu and Jia (2010) (Executable)	3 min (800 $\times$ 600 pixels)
Xu & Jia Xu *et al.* (2013) (Matlab, CPU)	34 sec
Pan Pan *et al.* (2016) (Matlab, CPU)	40 min
Whyte Whyte *et al.* (2012) (Matlab, CPU)	4 min

Table 3.1: Run-time for each method for average image size of 1024 $\times$ 700.



(a) (b) (c) (d)

Figure 3.8: Comparisons for space-invariant deblurring. (a) Input blurred image. (b-c) Deblurred output using methods in Xu and Jia (2010) and Pan *et al.* (2016), respectively. (d) Our result.

that are close to the clean feature samples. Another advantage of this loss is that it helps in faster training (especially during the initial stages) as it provides strong gradients. Apart from adversarial and l_1 cost on the feature space, we also used MSE cost on the recovered clean image after passing the generated features through the decoder. This helps in fine-tuning the generator to match with the decoder. Fig. 2.1 shows the direction of error back-propagation along with the network modules.

3.2.5 Final Training

We train GAN using normal procedure but instead of asking the discriminator to discern between generated images and clean images, we ask it to discriminate between their corresponding features. The generator and the discriminator architectures are as given below.

Generator: $C^5_{3\to8} \downarrow 2 \to R^{5(2)}_8 \to C^5_{8\to16} \downarrow 2 \to R^{5(2)}_{16} \to C^3_{16\to32} \downarrow 2 \to R^{5(2)}_{32} \to C^3_{32\to128} \downarrow 2 \to R^{3(2)}_{128} \to C^3_{128\to32} \uparrow 2$

Discriminator: $C^5_{32\to32} \to C^5_{32\to32} \downarrow 2 \to C^5_{32\to16} \to C^5_{16\to16} \downarrow 2 \to C^5_{16\to8} \to C^3_{8\to8} \downarrow 2 \to C^3_{8\to1}$

Each convolution is followed by a Leaky ReLU and batch-normalization in the discrim-

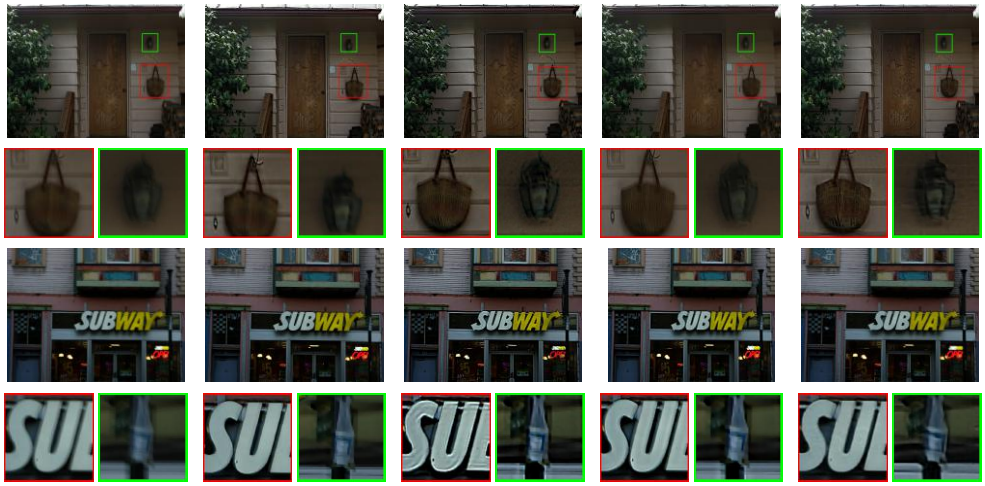


Figure 3.9: Examples for space-variant deblurring and comparisons with conventional state-of-the-art methods. (a) Input blurred image. (b-e) Results obtained by the methods in Xu *et al.* (2013); Whyte *et al.* (2012); Pan *et al.* (2016) and our result, respectively.

inator, and ReLU and batch-normalization in the generator.

Once the second stage is trained, we have a generator module to which we pass the blurred input during the test phase. The generator produces features which correspond to clean image features which when passed through the decoder deliver the final deblurred result. It may be noted that our network is compact with 34 convolutional layers (generator and decoder put together) despite performing end-to-end deblurring. The details of choice of parameters for both stages is given below:

We trained our autoencoder using clean patches of size 128 $\times$ 128 from the Pascal VOC 2012 dataset Everingham *et al.* (2012). The inputs were randomly corrupted with Gaussian noise (standard deviation = 0.2) 30% of the time to ensure learning of useful data representation. For learning, we used Adam Kingma and Adam (2015) with an initial learning rate of 0.0002 and momentum 0.9 with batch-size of 8. The training took around 10^5 iterations to converge. The gradient cost was scaled by λ = 0.1 to ensure that the final results are not over-sharpened.

The second stage of training involved learning a blur-invariant representation from blurred data. For creating the blurred data, we used around 10^5 kernels generated using the code provided by Chakrabarthi et al. Chakrabarti (2016). The input images from PASCAL dataset were blurred using these kernels and patches of size 128 $\times$ 128 were

extracted. Around 35×10^5 training data was used to train the generator-discriminator pair. The Adam optimizer with initial setting as before was used with a batch-size of 16. To improve GAN stability, we also used smooth labeling of blur and clean features as discussed in Arjovsky and Bottou (2017). For around 2×10^5 iterations, the training was done with feature costs alone with $\lambda_{\text{adv}} = 0.01$ and $\lambda_1 = 1$. Fine tuning of the generator was subsequently done by adding the MSE cost and weighing down the adversarial cost ($\lambda_2 = 1$, $\lambda_1 = 1$ and $\lambda_{\text{adv}} = 0.001$).

CHAPTER 4

Tried Network Architectures and Recent Related Works

4.1 Tried Network Architectures

Initially our first architecture for autoencoder had only convolutional layers with ReLU and max-pooling layers in between. We trained this network with both MSE as well as gradient loss criteria.The PSNR of the recovered image with this network was less even after using only one max-pooling layer in the architecture. Also, this architecture has gradient vanishing problem, so going deep for extraction features for encoding is difficult. Using strided convolution along with ResNet solved these problems. Max-pooling layers provide invariance to small translations of the input due to which these features are lost in the image due to max-pooling. But in case of autoencoders ideally we want the complete image to be recovered as it is. Hence, we trained the autoencoder whose architecture is shown in figure 3.1 that sidesteps above problems.

After autoencoder training the encoded features of the blurry image should be matched with corresponding clean image features, which when passed through the decoder should produce deblurred image. The first obvious idea came to our mind was to match features by using fully connected layers and propagate MSE and gradient loss during training. But by introducing fully connected layers in the network, the training becomes extremely slow making this technique computationally infeasible. Next we tried CNN for feature correction which takes features of a clean image as input and outputs corresponding clean image features. But training this network with MSE (on features) and gradient loss itself only works for very less blur and encourages some blur in the output.

Using adversarial loss along with MSE proved to be very effective for feature matching task. With adversarial loss network started to give sharp from the first epoch itself. This is due to strong gradients propagated back to generator using adversarial loss function. Recent works has have shown that high-quality images can be generated by optimizing perceptual loss functions based on high-level features extracted from pretrained networks. They define perceptual loss as MSE loss on the features encoded by standard

networks like VGG, AlexNet. Although MSE loss on features encoded by our autoencoder is similar to this but we also tried explicitly including perceptual loss (feature loss from trained VGG network) in our loss function. But it didn't show any significant improvement. Below are some of the tried networks and results. Architecture wise these networks are not much different from the our final network.

- $Network^1$: Autoencoder of figure 3.1 with adversarial and MSE loss on features We trained this network on space invariant synthetically blurred images from celebrity face dataset with image size of 256 x 256. The network was trained for 200 epochs on 17000 images. The training was done with two different values of $\mathcal{L}_{\text{adv}}$, 0.01 (very high.) and 0.0001. Due to high weightage to adversarial cost (0.01) the network was trying to output extra sharp output resulted in artifacts especially for heavily blurred images but for less weightage the network was not able to handle heavily blurred images. Figure 4.1 shows result of this network with two different settings.
- $Network^2$: Autoencoder of figure 3.1 trained with adversarial , MSE loss on features and explicitly adding feature loss on features encoded by pretrained VGG network. We trained this network by giving different weightage to adversarial loss and compared with $Network^1$. Addition of features from VGG network didn't show any significant improvement.
- $Network^3$: Autoencoder of figure 3.1 trained with adversarial , MSE loss on features and on final reconstructed image when passed through decoder. Here we tried two variants, In first we only update weights of Generator while keeping decoder unchanged. In second we decoder also with only the MSE loss between reconstructed and ground truth image. These two networks were tried by giving different weightage to all the three losses. The second one shows slight improvement over the first but it slows down the training process.

4.2 Recent Related Works

Recently two more end-to-end single image deep blind deblurring came in parallel to our work [kyoung,parmanand]. Their network architecture being similar to each other significantly differs from ours. Both of them use three stage multiscale approach for deblurring, as blur is less visible at smaller scales. Both of the papers proposed datasets that provides pairs of realistic blurry image and the corresponding ground truth sharp image. To obtain such data they use high frame rate video camera (Gopro camera) and keep one frame as the sharp image and average of frames as the corresponding blurred image.

(a) (b) (c)

(d) (e) (f)

Figure 4.1: (b) is the result of $Network^1$ on a blurry image when trained with $\mathcal{L}_{\text{adv}} = 0.0001$. (a) and (c) are the corresponding input and target image respectively. (e) shows the output of same network when trained with $\mathcal{L}_{\text{adv}} = 0.01$, (d) and (f) are the corresponding input and target image respectively.

First stage of [kyoung]'s network takes blurred image scaled to $\frac{1}{4}$ as input and produces deblurred image at $\frac{1}{4}$ scale. Second stage takes blurred image scaled to $\frac{1}{2}$ and output of first image upconvolved to $\frac{1}{2}$ as input and produces deblurred image at $\frac{1}{2}$ scale. Similarly, output of second stage upconvolved to $\frac{1}{2}$ and blurred image is fed to third stage to get the final deblurred output. For optimizing the network parameters, they use combination of two losses MSE and adversarial loss at each stage. [kyoung] in his paper shows that using multiscale approach doesn't show much improvement.

[parmanand]'s first stage of network takes original blurred image as input and produces sharp image at $\frac{1}{4}$ scale of the original. Second stage takes output of first stage and blurred image scaled to $\frac{1}{4}$ as input and produces clean output at $\frac{1}{2}$ of original scale. Third stage takes out from second network and blurred image scaled to $\frac{1}{2}$ as input and produces clean final deblurred output. Unlike [kyoung]'s this network doesn't have separate losses for each stage and has only one discriminator for discriminating at final output. One major difference is that [kyoung]'s network is very large (120 layers) where as [parmanand] uses only 17 layers.

CHAPTER 5

Experiments

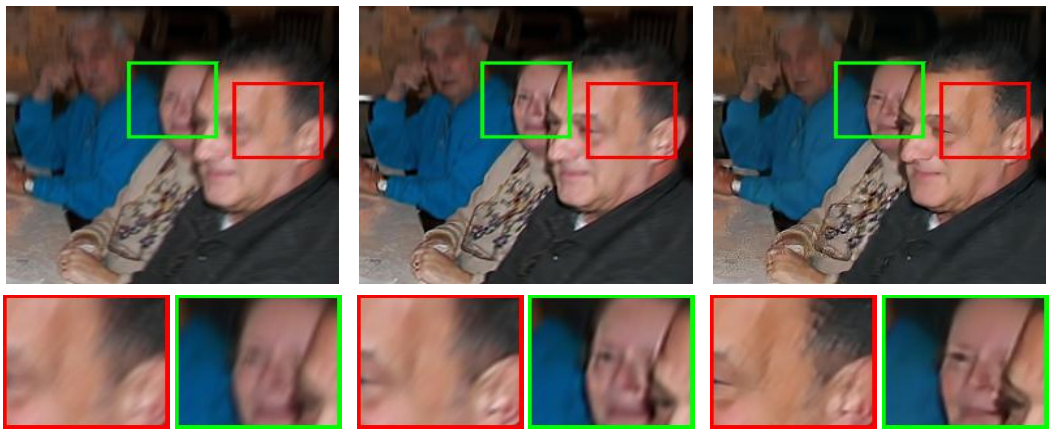
We evaluated the efficacy of our network for deblurring, both quantitatively and qualitatively. We also compared performance with conventional as well as deep networks. For conventional methods, we selected the very recent dark channel prior of Pan *et al.* (2016), l_0 unnatural prior by Xu et al. Xu *et al.* (2013), deblurring method of Whyte et al. Whyte *et al.* (2012) and the two-phase kernel method of Xu Xu and Jia (2010). Codes were downloaded from the authors' websites. We also did comparisons with deep learning-based approaches Chakrabarti (2016) and Sun *et al.* (2015). In addition to the above, we also tested on images affected by object motion and compared our results with the generalized video deblurring method of Hyun Kim and Mu Lee (2015).

5.1 Quantitative Analysis

We performed quantitative comparisons of our method with state-of-the-art methods Pan *et al.* (2016); Xu *et al.* (2013); Whyte *et al.* (2012); Xu and Jia (2010) on the dataset of Sun *et al.* (2013). The dataset consists of 80 images and 8 kernels. We did a comparative study by picking 5 kernels from the set and using them to blur 6 randomly chosen images (i.e. 30 blurred image in all). The average PSNR and MSSIM (Mean SSIM) measures obtained on each of these images is provided in Figs. 3.7(a) and

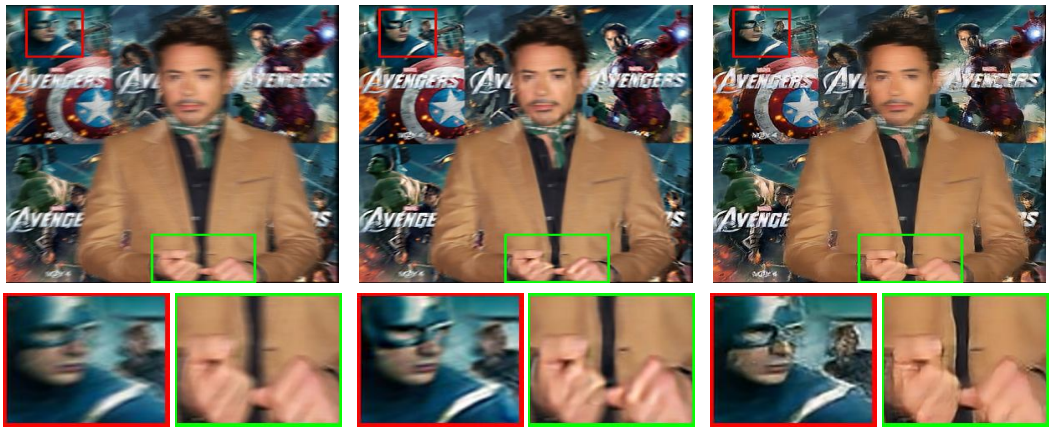
(a) (b) (c) (d)

Figure 5.1: Comparison with Chakrabarti (2016). (a) Input blurred image. (b) Output of our network. (c) Network output of Chakrabarti (2016) and (d) final non-blind deblurred output of Chakrabarti (2016).



(a) (b) (c)

Figure 5.2: Comparison with Sun *et al.* (2015). (a) Space variantly blurred input image. (b) Deblurred output of Sun *et al.* (2015). (c) Output of our network.



(a) (b) (c)

Figure 5.3: Comparison with Sun *et al.* (2015). (a) Space variantly blurred input image. (b) Deblurred output of Sun *et al.* (2015). (c) Output of our network.

(a) (b) (c)

Figure 5.4: Comparison with Hyun Kim and Mu Lee (2015). (a) Input blurred image of dynamic scene. (b) Output of our network. (c) Network output and final non-blind deblurred output of Hyun Kim and Mu Lee (2015).

(b), respectively. Although we do not always outperform the above listed conventional methods, our performance is at par with them and sometimes even better (**img34**). Importantly, our method offers a significant advantage in terms of run-time. The images tested here had an average size of 1024×700 and we have reported the run-time for each method (run on Intel I-7 3.4GHz CPU with 8 cores) in Table 3.1. Note that our method implemented in Torch and run using a Titan-X GPU is at least an order faster than the fastest competitive method Xu *et al.* (2013).

We also provide quantitative evaluation of our network on the Köhler dataset Köhler *et al.* (2012) which is commonly used for evaluating non-uniform deblurring techniques. We have compared against conventional methods Whyte *et al.* (2012); Xu *et al.* (2013) and the deep network approach of Sun *et al.* (2015). The dataset consists of 4 images and 12 non-uniform camera motions totaling to 48 blurred images. The average MSSIM values are provided in Table 5.1. These numerical values are calculated by using the evaluation code provided in the webpage of Köhler *et al.* (2012) and the results for competing methods are obtained by running the code provided by the authors (on their webpage) with default parameter settings for the respective works. Note that the performance of our network is quite comparable with competing methods on this dataset as well. The highlight of our method is that it is quite fast and does not involve any parameter tuning.

5.2 Qualitative Comparisons

Figs. 3.8 and 3.9 provide qualitative performance of our network compared to conventional methods on space-invariant and space-variant blur, respectively. The results in Fig. 3.8 clearly reveal that our method (Fig. 3.8(d)) produces results devoid of any ringing artifacts when compared to Xu and Jia (2010) and Pan *et al.* (2016). The butterfly's wing and the lady's collar look sharp and clear. Though the result in Fig. 3.8(c) also appears to be sharp, there is ringing at the boundaries. The same issue is present in Fig. 3.8(b) as well.

The efficacy of our method to deal with space-variant blur due to non-uniform camera motion is given in Fig. 3.9. Here, the input images (first column) are affected by

different blurs at different locations. We compared our method with that of Xu *et al.* (2013); Whyte *et al.* (2012); Pan *et al.* (2016). It can be clearly observed that our method outperforms all others. The zoomed-in patch of the bag in the second row as well as the bottle in the fourth row are quite clear and sharp in our result (Fig. 3.9(e)), in comparison with other outputs. It has to be noted here that we ran all the comparisons using the default settings provided by the authors in their codes. The result of these methods could perhaps improve with parameter tuning (although the 'subway' comparison results are taken directly from Pan *et al.* (2016)); but as already explained earlier section, this is very cumbersome exercise; more so, given the fact that the run-time for the competing methods is quite high. In stark contrast, our method elegantly avoids these pitfalls.

We provide additional qualitative comparisons on publicly available Lai et al. Lai *et al.* (2016) real dataset. Figs. 5.5 - 5.12 show results of our network along with the outputs obtained from prior-based state-of-the-art conventional methods Levin *et al.* (2011); Whyte *et al.* (2012); Xu *et al.* (2013); Sun *et al.* (2013); Pan *et al.* (2014) on randomly chosen images from Lai et al. Lai *et al.* (2016) dataset. The dataset consists of both synthetic and real images collected from various conventional prior works on deblurring. Comparative results are directly taken from Lai *et al.* (2016) and hence it is reasonable to assume that the results for competing methods correspond to their fine-tuned parameters and priors. Figs. 5.5 - 5.7 are synthetic examples from this dataset generated using non-uniform camera motions and affected with several common degradations. Rest of the examples are real and are taken from Lai *et al.* (2016). Fig. 5.13 contains additional visual results on images captured by us as well as real images taken from Lai *et al.* (2016). From the exhaustive evaluation given in the main paper and in this supplementary material, it is amply evident that our method yields output that is visually comparable or even better than methods that painstakingly employ different priors on the underlying clean image and kernel for deblurring.

5.3 Comparision with Deep networks

We also compared our deblurring results with that of Chakrabarti (2016) and Sun *et al.* (2015). These are deep network-based approaches but perform the task of kernel es-

timation. The deblurred results for these methods are obtained by using a non-blind deblurring scheme in the final step. For comparisons with Chakrabarti (2016) (shown in Fig. 5.1), we ran our network on the images provided in their paper. The results are compared here with the network output of Chakrabarti (2016) (Fig. 5.1 (c)) and the final output (Fig. 5.1 (d)) obtained post non-blind deblurring. It must be noted here that our network output (Fig. 5.1 (b)) is significantly better than the network output of Fig. 5.1 (c). Moreover, the method Chakrabarti (2016) can only handle space-invariant blur.

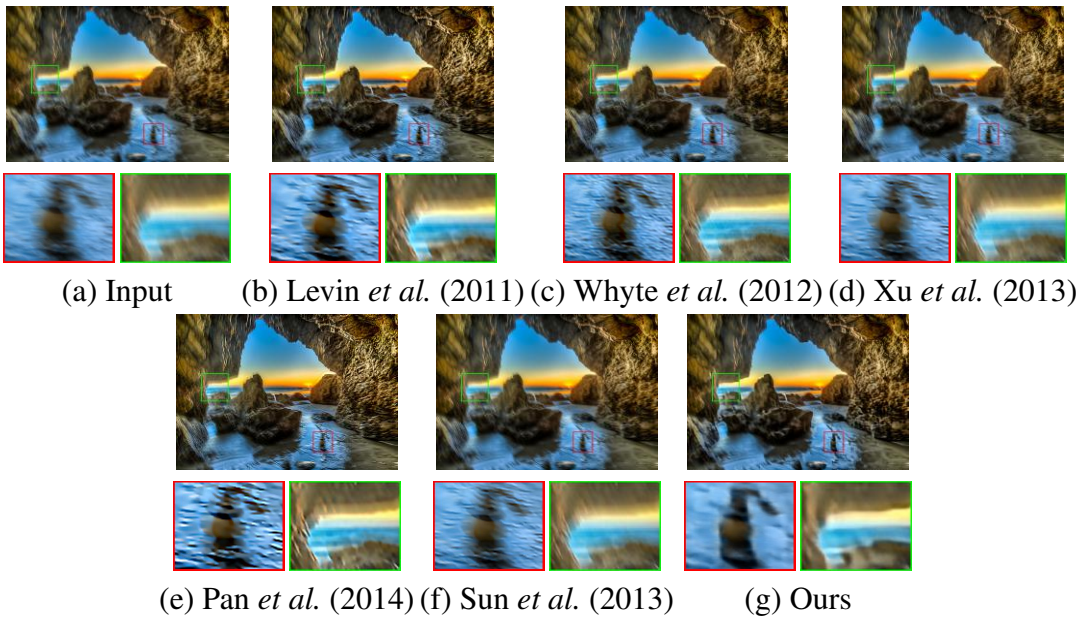
We also compared our network with Sun *et al.* (2015) that performs parametrized estimation of space-variant blur using deep networks and then uses a conventional method for deblurring. We again ran our network on the images provided by the authors in their paper (Fig. 5.2 & Fig.5.3). Note that our model produces results that are comparable to that produced by final deblurring in Sun *et al.* (2015). The zoomed-in patches of the man and the woman's face in the second row as well as the masked man's face and reporter's hands in the forth row are much sharper in our result.

5.3.1 Object Motion Deblurring

We also tested on images with blur due to object motion and observed that our network is able to handle even such cases to a resonable extent although it was not trained with such examples. In contrast, conventional deblurring methods Pan *et al.* (2016); Xu *et al.* (2013); Whyte *et al.* (2012); Xu and Jia (2010) that model blur due to camera motion (alone) cannot handle blur due to object motion. Fig. 5.4 depicts examples of object motion deblurring where the inputs in Fig. 5.4(a) have blur in the background due to camera motion while the foreground (which is at a differnt depth from the background) incurs blur due to object as well as camera motion. We compared our results (Fig. 5.4(c)) with the video-based dynamic scene deblurring method of Hyun Kim and Mu Lee (2015) (Fig. 5.4(b)). The results reveal that our method (Fig. 5.4(c)) (although single image based) is able to perform at par with the video-based method that uses information from multiple frames.

	Whyte *et al.* (2012)	Xu *et al.* (2013)	Sun *et al.* (2015)	Ours
MSSIM	0.8405	0.8340	0.7932	0.8102

Table 5.1: Quantitative comparisons on Köhler dataset Köhler *et al.* (2012).



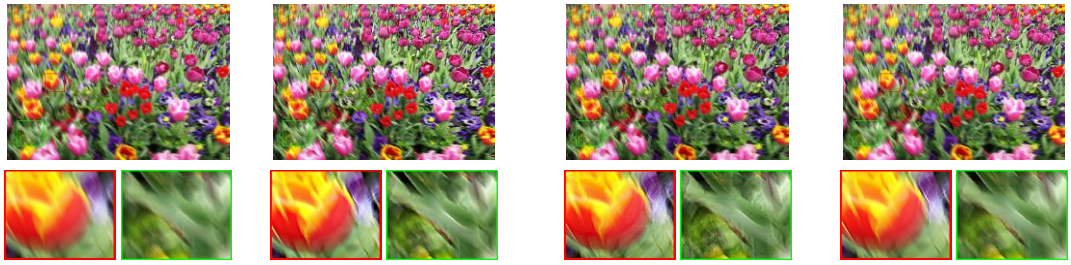
(a) Input (b) Levin *et al.* (2011) (c) Whyte *et al.* (2012) (d) Xu *et al.* (2013)

(e) Pan *et al.* (2014) (f) Sun *et al.* (2013) (g) Ours

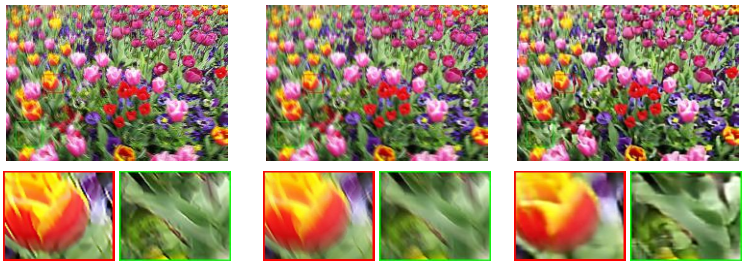
Figure 5.5: Synthetic example taken from dataset Lai *et al.* (2016). (a) Input blurry image. (b-f) Deblurred result corresponding to methods in Levin *et al.* (2011); Whyte *et al.* (2012); Xu *et al.* (2013); Pan *et al.* (2014); Sun *et al.* (2013) in order respectively, and (g) is the result obtained by our network. Zoomed-in patches are shown below each of the corresponding result for better viewing.

5.4 Conclusions

In this work, we analyze various deep network architectures for blind deblurring and proposed an end-to-end deep network for single image blind-deblurring using autoencoder and GAN. Instead of directly regressing for clean pixels, we perform regression over encoder-features to arrive at a blur-invariant representation which when passed through the decoder produces the desired clean output. Our network is kernel-free, does not require any prior modeling, and can handle both space-invariant and space-variant blur. When evaluated on standard datasets as well as on our own examples, our network performs at par or even better than competing methods while being faster by at least an order. Inspired from traditional dictionary technique we believe that this kind of architecture can be extended for other tasks like super-resolution and image inpainting. Where traditional dictionary learning techniques has already shown good results.

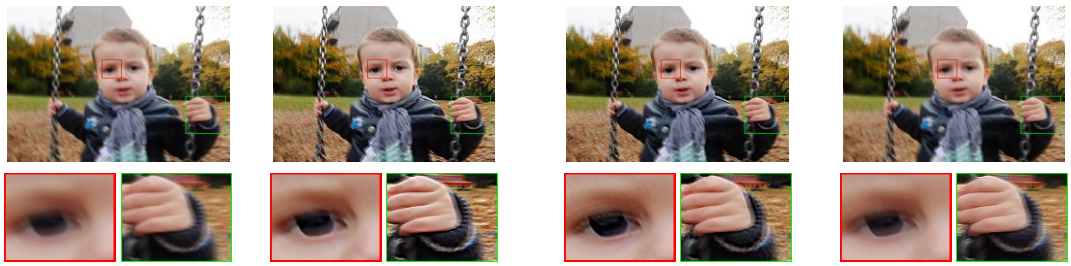


(a) Input (b) Levin *et al.* (2011) (c) Whyte *et al.* (2012) (d) Xu *et al.* (2013)

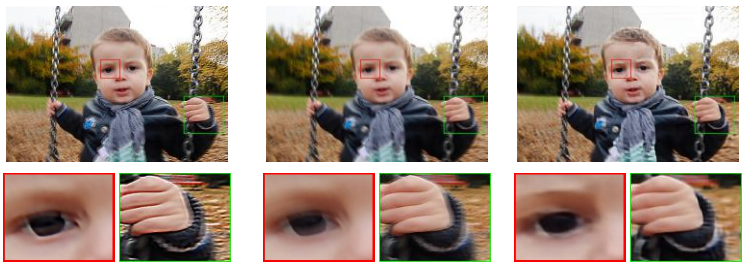


(e) Pan *et al.* (2014) (f) Sun *et al.* (2013) (g) Ours

Figure 5.6: Another synthetic example from Lai *et al.* (2016) dataset. (a) Input blurry image. (b-f) Deblurred result corresponding to methods in Levin *et al.* (2011); Whyte *et al.* (2012); Xu *et al.* (2013); Pan *et al.* (2014); Sun *et al.* (2013) in order respectively, and (g) is the result obtained by our network. The zoomed in patch of leaves is much clear in our result.



(a) Input (b) Levin *et al.* (2011) (c) Whyte *et al.* (2012) (d) Xu *et al.* (2013)



(e) Pan *et al.* (2014) (f) Sun *et al.* (2013) (g) Ours

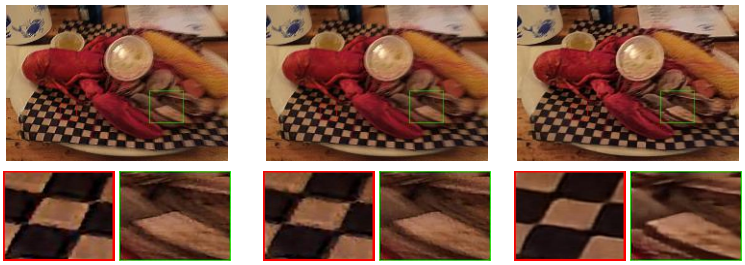
Figure 5.7: Synthetic example taken from dataset Lai *et al.* (2016). (a) Input blurry image. (b-f) Deblurred result corresponding to methods in Levin *et al.* (2011); Whyte *et al.* (2012); Xu *et al.* (2013); Pan *et al.* (2014); Sun *et al.* (2013) in order respectively, and (g) is the result obtained by our network. Here the kid's eye and hand look more clear in our result.

(a) Input (b) Levin *et al.* (2011) (c) Whyte *et al.* (2012) (d) Xu *et al.* (2013)

(e) Pan *et al.* (2014) (f) Sun *et al.* (2013) (g) Ours

Figure 5.8: Visual comparison with conventional methods on real example. Our method generates output that are at par or better than the state-of-the-art methods.

(a) Input (b) Levin *et al.* (2011) (c) Whyte *et al.* (2012) (d) Xu *et al.* (2013)



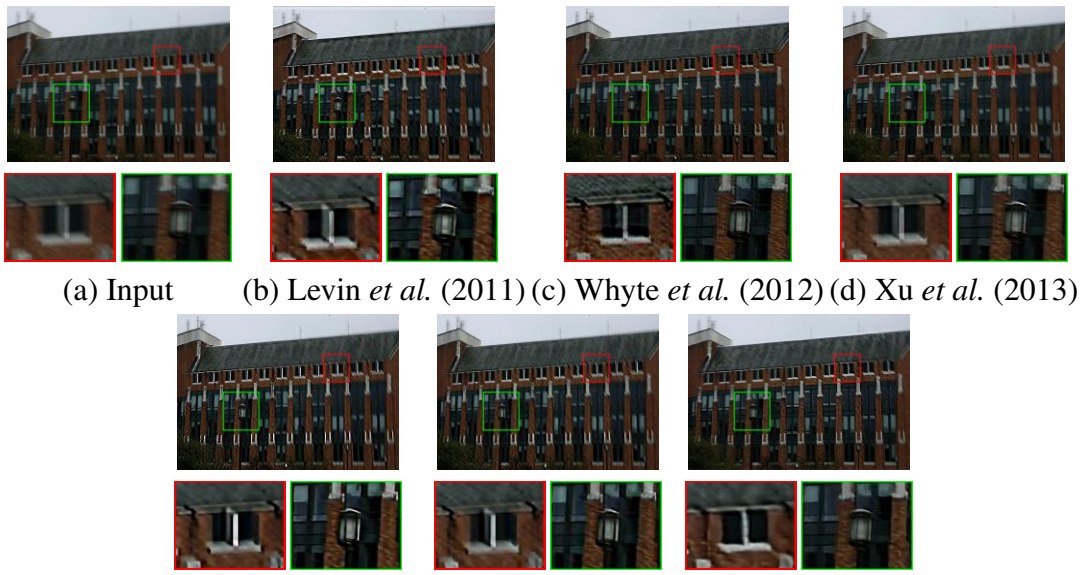
(e) Pan *et al.* (2014) (f) Sun *et al.* (2013) (g) Ours

Figure 5.9

(a) Input (b) Levin *et al.* (2011) (c) Whyte *et al.* (2012) (d) Xu *et al.* (2013)

(e) Pan *et al.* (2014) (f) Sun *et al.* (2013) (g) Ours

Figure 5.10

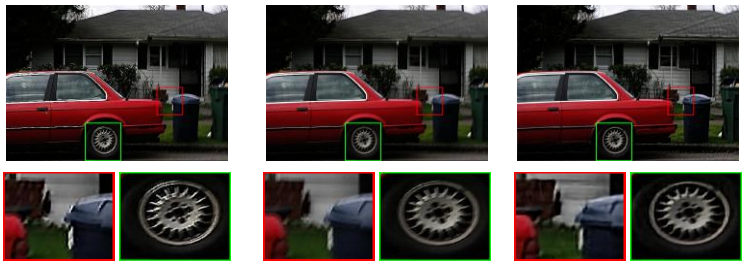


(a) Input (b) Levin *et al.* (2011) (c) Whyte *et al.* (2012) (d) Xu *et al.* (2013)

(e) Pan *et al.* (2014) (f) Sun *et al.* (2013) (g) Ours

Figure 5.11

(a) Input (b) Levin *et al.* (2011) (c) Whyte *et al.* (2012) (d) Xu *et al.* (2013)



(e) Pan *et al.* (2014) (f) Sun *et al.* (2013) (g) Ours

Figure 5.12

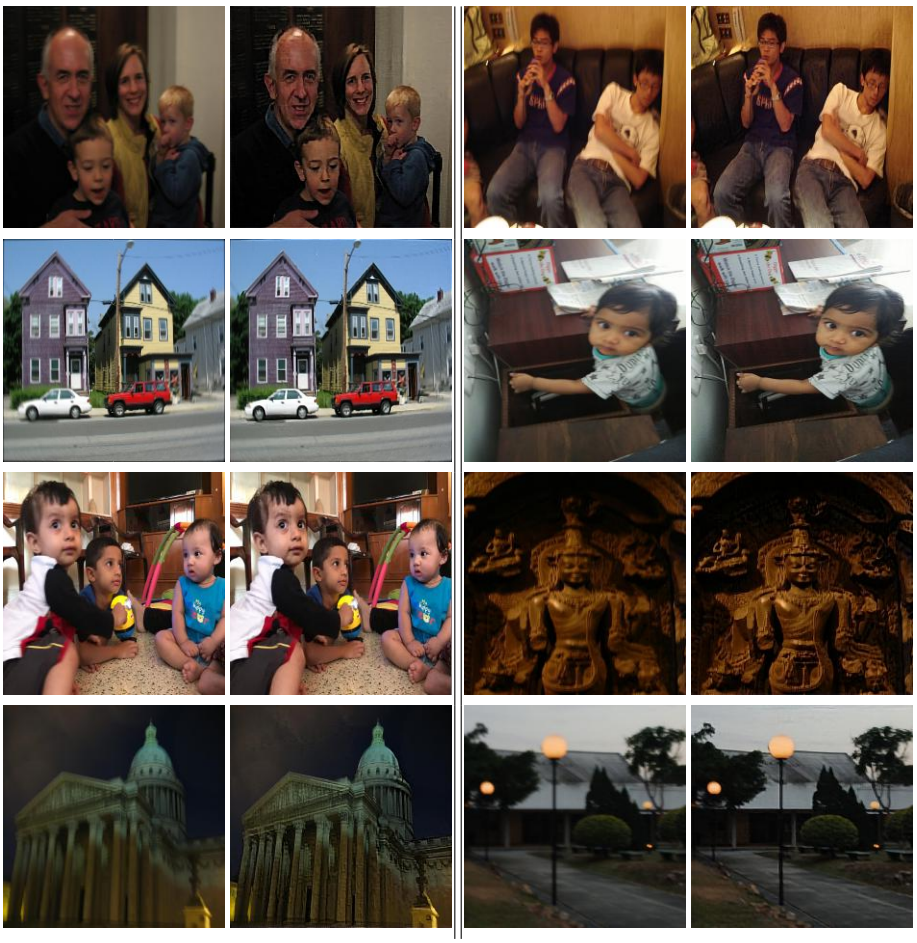


Figure 5.13: Additional qualitative results for images picked randomly from Lai *et al.* (2016), and captured by ourself. Input in the left side and corresponding output in the right.

REFERENCES

1. **Aharon, M.**, **M. Elad**, and **A. Bruckstein** (2006). rmk-svd: An algorithm for designing overcomplete dictionaries for sparse representation. *Transactions on signal processing*, **54**(11), 4311–4322.

2. **Arjovsky, M.** and **L. Bottou**, Towards principled methods for training generative adversarial networks. *In NIPS 2016 Workshop on Adversarial Training*, volume 2016. 2017.

3. **Chakrabarti, A.**, A neural approach to blind motion deblurring. *In ECCV*. Springer, 2016.

4. **Chan, T. F.** and **C.-K. Wong** (1998). Total variation blind deconvolution. *TIP*, **7**(3), 370–375.

5. **Cho, S.** and **S. Lee**, Fast motion deblurring. *In TOG*, volume 28. ACM, 2009.

6. **Donahue, J.**, **P. Krähenbühl**, and **T. Darrell**, Adversarial feature learning. *In ICLR*. 2017.

7. **Dong, C.**, **C. C. Loy**, **K. He**, and **X. Tang** (2016). Image super-resolution using deep convolutional networks. *TPAMI*, **38**(2), 295–307.

8. **Everingham, M.**, **L. Van Gool**, **C. K. I. Williams**, **J. Winn**, and **A. Zisserman** (2012). The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

9. **Goodfellow, I.**, **J. Pouget-Abadie**, **M. Mirza**, **B. Xu**, **D. Warde-Farley**, **S. Ozair**, **A. Courville**, and **Y. Bengio**, Generative adversarial nets. *In NIPS*. 2014.

10. **Gretton, A.**, **K. M. Borgwardt**, **M. Rasch**, **B. Schölkopf**, **A. J. Smola**, *et al.* (2007). A kernel method for the two-sample-problem. *Advances in neural information processing systems*, **19**, 513.

11. **Gupta, A.**, **N. Joshi**, **L. Zitnick**, **M. Cohen**, and **B. Curless**, Single image deblurring using motion density functions. *In ECCV*. 2010.

12. **He, K.**, **X. Zhang**, **S. Ren**, and **J. Sun**, Deep residual learning for image recognition. *In CVPR*. 2016.

13. **Hinton, G. E.** (2002). Training products of experts by minimizing contrastive divergence. *Neural computation*, **14**(8), 1771–1800.

14. **Hinton, G. E.** and **R. R. Salakhutdinov** (2006). Reducing the dimensionality of data with neural networks. *science*, **313**(5786), 504–507.

15. **Hyun Kim, T.** and **K. Mu Lee**, Generalized video deblurring for dynamic scenes. *In CVPR*. 2015.

16. **Isola, P.**, **J.-Y. Zhu**, **T. Zhou**, and **A. A. Efros** (2016). Image-to-image translation with conditional adversarial networks. *arXiv preprint arXiv:1611.07004*.

17. **Joshi, N.**, **R. Szeliski**, and **D. J. Kriegman**, Psf estimation using sharp edge prediction. *In CVPR*. IEEE, 2008.

18. **Kingma, D.** and **J. B. Adam**, A method for stochastic optimisation. *In ICLR*. ICLR, 2015.

19. **Köhler, R.**, **M. Hirsch**, **B. Mohler**, **B. Schölkopf**, and **S. Harmeling**, Recording and playback of camera shake: Benchmarking blind deconvolution with a real-world database. *In European Conference on Computer Vision*. Springer, 2012.

20. **Krishnan, D.**, **T. Tay**, and **R. Fergus**, Blind deconvolution using a normalized sparsity measure. *In CVPR*. IEEE, 2011.

21. **Krizhevsky, A.**, **I. Sutskever**, and **G. E. Hinton**, Imagenet classification with deep convolutional neural networks. *In NIPS*. 2012.

22. **Lai, W.-S.**, **J.-B. Huang**, **Z. Hu**, **N. Ahuja**, and **M.-H. Yang**, A comparative study for single image blind deblurring. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016.

23. **Levin, A.**, **Y. Weiss**, **F. Durand**, and **W. T. Freeman** (2011). Understanding blind deconvolution algorithms. *TPAMI*, **33**(12), 2354–2367.

24. **Li, Y.**, **K. Swersky**, and **R. Zemel** (2015). Generative moment matching networks, 1718–1727.

25. **Mairal, J.**, **J. Ponce**, **G. Sapiro**, **A. Zisserman**, and **F. R. Bach**, Supervised dictionary learning. *In Advances in NIPS*. 2009.

26. **Mathieu, M.**, **C. Couprie**, and **Y. LeCun** (2015). Deep multi-scale video prediction beyond mean square error. *arXiv preprint arXiv:1511.05440*.

27. **Mirza, M.** and **S. Osindero** (2014). Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*.

28. **Pan, J.**, **Z. Hu**, **Z. Su**, and **M.-H. Yang**, Deblurring text images via l0-regularized intensity and gradient prior. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014.

29. **Pan, J.**, **D. Sun**, **H. Pfister**, and **M.-H. Yang**, Blind image deblurring using dark channel prior. *In CVPR*. 2016.

30. **Pathak, D.**, **P. Krahenbuhl**, **J. Donahue**, **T. Darrell**, and **A. A. Efros**, Context encoders: Feature learning by inpainting. *In CVPR*. 2016.

31. **Perrone, D.** and **P. Favaro**, Total variation blind deconvolution: The devil is in the details. *In CVPR*. 2014.

32. **Schuler, C. J.**, **M. Hirsch**, **S. Harmeling**, and **B. Schölkopf**, Learning to deblur. *In NIPS*. 2014. URL `http://arxiv.org/abs/1406.7444`.

33. **Shan, Q.**, **J. Jia**, and **A. Agarwala**, High-quality motion deblurring from a single image. *In TOG*, volume 27. ACM, 2008.

34. **Shlens, J.**, Notes on kullback-leibler divergence and likelihood. 2014.

35. **Sun, J.**, **W. Cao**, **Z. Xu**, and **J. Ponce**, Learning a convolutional neural network for non-uniform motion blur removal. *In CVPR*. IEEE, 2015.

36. **Sun, L.**, **S. Cho**, **J. Wang**, and **J. Hays**, Edge-based blur kernel estimation using patch priors. *In ICCP*. IEEE, 2013.

37. **Vincent, P.**, **H. Larochelle**, **Y. Bengio**, and **P.-A. Manzagol**, Extracting and composing robust features with denoising autoencoders. *In Proceedings of the 25th iICML*. ACM, 2008.

38. **Warde-Farley, D.** and **Y. Bengio**, Improving generative adversarial networks with denoising feature matching. *In ICLR*. ICLR, 2017.

39. **Whyte, O.**, **J. Sivic**, **A. Zisserman**, and **J. Ponce** (2012). Non-uniform deblurring for shaken images. *IJCV*, **98**(2), 168–186.

40. **Xiang, S.**, **G. Meng**, **Y. Wang**, **C. Pan**, and **C. Zhang** (2015). Image deblurring with coupled dictionary learning. *IJCV*, **114**(2-3), 248–271.

41. **Xie, J.**, **L. Xu**, and **E. Chen**, Image denoising and inpainting with deep neural networks. *In NIPS*. 2012.

42. **Xu, L.** and **J. Jia**, Two-phase kernel estimation for robust motion deblurring. *In ECCV*. Springer, 2010.

43. **Xu, L.**, **J. S. Ren**, **C. Liu**, and **J. Jia**, Deep convolutional neural network for image deconvolution. *In NIPS*. 2014.

44. **Xu, L.**, **S. Zheng**, and **J. Jia**, Unnatural l0 sparse representation for natural image deblurring. *In CVPR*. 2013.

45. **Yang, J.**, **J. Wright**, **T. S. Huang**, and **Y. Ma** (2010). Image super-resolution via sparse representation. *TIP*, **19**(11), 2861–2873.

46. **Zhang, H.** and **L. Carin**, Multi-shot imaging: joint alignment, deblurring and resolution-enhancement. *In CVPR*. 2014.