# Restoring An Image Taken Through a Window Covered with Dirt or Rain

David Eigen  Dilip Krishnan  Rob Fergus
Dept. of Computer Science, Courant Institute, New York University
{deigen,dilip,fergus}@cs.nyu.edu

## Abstract

*Photographs taken through a window are often compromised by dirt or rain present on the window surface. Common cases of this include pictures taken from inside a vehicle, or outdoor security cameras mounted inside a protective enclosure. At capture time, defocus can be used to remove the artifacts, but this relies on achieving a shallow depth-of-field and placement of the camera close to the window. Instead, we present a post-capture image processing solution that can remove localized rain and dirt artifacts from a single image. We collect a dataset of clean/corrupted image pairs which are then used to train a specialized form of convolutional neural network. This learns how to map corrupted image patches to clean ones, implicitly capturing the characteristic appearance of dirt and water droplets in natural images. Our models demonstrate effective removal of dirt and rain in outdoor test conditions.*

## 1. Introduction

There are many situations in which images or video might be captured through a window. A person may be inside a car, train or building and wish to photograph the scene outside. Indoor situations include exhibits in museums displayed behind protective glass. Such scenarios have become increasingly common with the widespread use of smartphone cameras. Beyond consumer photography, many cameras are mounted outside, e.g. on buildings for surveillance or on vehicles to prevent collisions. These cameras are protected from the elements by an enclosure with a transparent window.

Such images are affected by many factors including reflections and attenuation. However, in this paper we address the particular situation where the window is covered with dirt or water drops, resulting from rain. As shown in Fig. 1, these artifacts significantly degrade the quality of the captured image.

The classic approach to removing occluders from an image is to defocus them to the point of invisibility at the time of capture. This requires placing the camera right up against the glass and using a large aperture to produce small depth-of-field. However, in practice it can be hard to move the camera sufficiently close, and aperture control may not be available on smartphone cameras or webcams. Correspondingly, many shots with smartphone cameras through dirty or rainy glass still have significant artifacts, as shown in Fig. 9.

Figure 1. A photograph taken through a glass pane covered in rain, along with the output of our neural network model, trained to remove this type of corruption. The irregular size and appearance of the rain makes it difficult to remove with existing methods. This figure is best viewed in electronic form.

In this paper we instead restore the image after capture, treating the dirt or rain as a structured form of image noise. Our method only relies on the artifacts being spatially compact, thus is aided by the rain/dirt being in focus — hence the shots need not be taken close to the window.

Image denoising is a very well studied problem, with current approaches such as BM3D [3] approaching theoretical performance limits [13]. However, the vast majority of this literature is concerned with additive white Gaussian noise, quite different to the image artifacts resulting from dirt or water drops. Our problem is closer to shot-noise removal, but differs in that the artifacts are not constrained to single pixels and have characteristic structure. Classic approaches such as median or bilateral filtering have no way

of leveraging this structure, thus cannot effectively remove the artifacts (see Section 5).

Our approach is to use a specialized convolutional neural network to predict clean patches, given dirty *or clean* ones as input. By asking the network to produce a clean output, regardless of the corruption level of the input, it implicitly must both detect the corruption and, if present, in-paint over it. Integrating both tasks simplifies and speeds test-time operation, since separate detection and in-painting stages are avoided.

Training the models requires a large set of patch pairs to adequately cover the space inputs and corruption, the gathering of which was non-trivial and required the development of new techniques. However, although training is somewhat complex, test-time operation is simple: a new image is presented to the neural network and it directly outputs a restored image.

### 1.1. Related Work

Learning-based methods have found widespread use in image denoising, e.g. [23, 14, 16, 24]. These approaches remove additive white Gaussian noise (AWGN) by building a generative model of clean image patches. In this paper, however, we focus on more complex structured corruption, and address it using a neural network that directly maps corrupt images to clean ones; this obviates the slow inference procedures used by most generative models.

Neural networks have previously been explored for denoising natural images, mostly in the context of AWGN, e.g. Jain and Seung [10], and Zhang and Salari [21]. Algorithmically, the closest work to ours is that of Burger *et al.* [2], which applies a large neural network to a range of non-AWGN denoising tasks, such as salt-and-pepper noise and JPEG quantization artifacts. Although more challenging than AWGN, the corruption is still significantly easier than the highly variable dirt and rain drops that we address. Furthermore, our network has important architectural differences that are crucial for obtaining good performance on these tasks.

Removing localized corruption can be considered a form of blind inpainting, where the position of the corrupted regions is not given (unlike traditional inpainting [5]). Dong *et al.* [4] show how salt-and-pepper noise can be removed, but the approach does not extend to multi-pixel corruption. Recently, Xie *et al.* [20] showed how a neural network can perform blind inpainting, demonstrating the removal of text synthetically placed in an image. This work is close to ours, but the solid-color text has quite different statistics to natural images, thus is easier to remove than rain or dirt which vary greatly in appearance and can resemble legitimate image structures. Jancsary *et al.* [11] denoise images with a Gaussian conditional random field, constructed using decision trees on local regions of the input; however, they too consider only synthetic corruptions.

Several papers explore the removal of rain from images. Garg and Nayar [7] and Barnum *et al.* [1] address airborne rain. The former uses defocus, while the latter uses frequency-domain filtering. Both require video sequences rather than a single image, however. Roser and Geiger [17] detect raindrops in single images; although they do not demonstrate removal, their approach could be paired with a standard inpainting algorithm. As discussed above, our approach combines detection and inpainting.

Closely related to our application is Gu *et al.* [9], who show how lens dust and nearby occluders can be removed, but their method requires extensive calibration or a video sequence, as opposed to a single frame. Wilson *et al.* [19] and Zhou and Lin [22] demonstrate dirt and dust removal. The former removes defocused dust for a Mars Rover camera, while the latter removes sensor dust using multiple images and a physics model.

## 2. Approach

To restore an image from a corrupt input, we predict a clean output using a specialized form of convolutional neural network [12]. The same network architecture is used for all forms of corruption; however, a different network is trained for dirt and for rain. This allows the network to tailor its detection capabilities for each task.

### 2.1. Network Architecture

Given a noisy image $x$, our goal is to predict a clean image $y$ that is close to the true clean image $y^*$. We accomplish this using a multilayer convolutional network, $y = F(x)$. The network $F$ is composed of a series of layers $F_l$, each of which applies a linear convolution to its input, followed by an element-wise sigmoid (implemented using hyperbolic tangent). Concretely, if the number of layers in the network is $L$, then

$$\begin{aligned} F_0(x) &= x \\ F_l(x) &= \tanh(W_l * F_{l-1}(x) + b_l), \quad l = 1, ..., L-1 \\ F(x) &= \frac{1}{m}(W_L * F_{L-1}(x) + b_L) \end{aligned}$$

Here, $x$ is the RGB input image, of size $N \times M \times 3$. If $n_l$ is the output dimension at layer $l$, then $W_l$ applies $n_l$ convolutions with kernels of size $p_l \times p_l \times n_{l-1}$, where $p_l$ is the spatial support. $b_l$ is a vector of size $n_l$ containing the output bias (the same bias is used at each spatial location).

While the first and last layer kernels have a nontrivial spatial component, we restrict the middle layers ($2 \leq l \leq L-1$) to use $p_l = 1$, i.e. they apply a linear map at each spatial location. We also element-wise divide the final output by the overlap mask[1] $m$ to account for different amounts of kernel overlap near the image boundary. The first layer

[1] $m = 1_K * 1_I$, where $1_K$ is a kernel of size $p_L \times p_L$ filled with ones, and $1_I$ is a 2D array of ones with as many pixels as the last layer input.

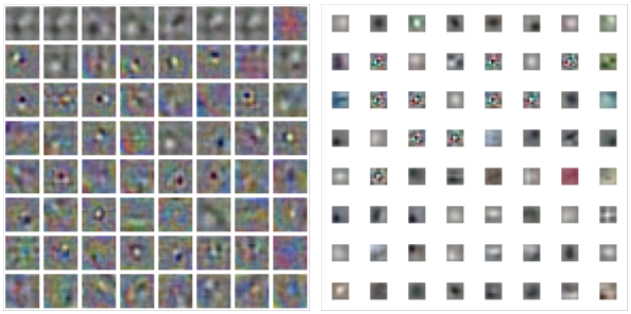


Figure 2. A subset of rain model network weights, sorted by $l_2$-norm. Left: first layer filters which act as detectors for the rain drops. Right: top layer filters used to reconstruct the clean patch.

uses a "valid" convolution, while the last layer uses a "full" (these are the same for the middle layers since their kernels have $1 \times 1$ support).

In our system, the input kernels' support is $p_1 = 16$, and the output support is $p_L = 8$. We use two hidden layers (i.e. $L = 3$), each with 512 units. As stated earlier, the middle layer kernel has support $p_2 = 1$. Thus, $W_1$ applies 512 kernels of size $16 \times 16 \times 3$, $W_2$ applies 512 kernels of size $1 \times 1 \times 512$, and $W_3$ applies 3 kernels of size $8 \times 8 \times 512$. Fig. 2 shows examples of weights learned for the rain data.

## 2.2. Training

We train the weights $W_l$ and biases $b_l$ by minimizing the mean squared error over a dataset $D = (x_i, y_i^*)$ of corresponding noisy and clean image pairs. The loss is

$$J(\theta) = \frac{1}{2|D|} \sum_{i \in D} ||F(x_i) - y_i^*||^2$$

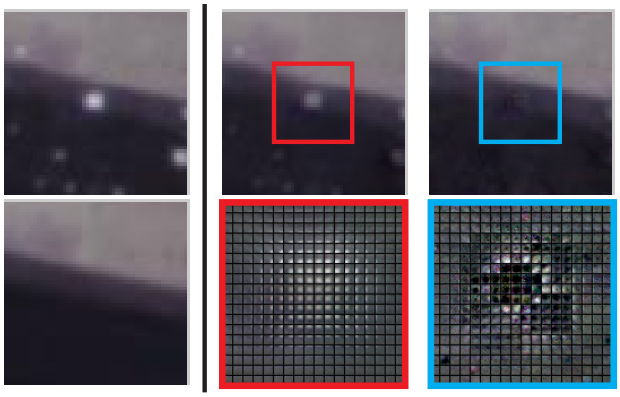
where $\theta = (W_1, ..., W_L, b_1, ..., b_L)$ are the model parameters. The pairs in the dataset $D$ are random $64 \times 64$ pixel subregions of training images with and without corruption (see Fig. 4 for samples). Because the input and output kernel sizes of our network differ, the network $F$ produces a $56 \times 56$ pixel prediction $y_i$, which is compared against the middle $56 \times 56$ pixels of the true clean subimage $y_i^*$.

We minimize the loss using Stochastic Gradient Descent (SGD). The update for a single step at time $t$ is

$$\theta^{t+1} \leftarrow \theta^t - \eta_t (F(x_i) - y_i^*)^T \frac{\partial}{\partial \theta} F(x_i)$$

where $\eta_t$ is the learning rate hyper-parameter and $i$ is a randomly drawn index from the training set. The gradient is further backpropagated through the network $F$.

We initialize the weights at all layers by randomly drawing from a normal distribution with mean 0 and standard deviation 0.001. The biases are initialized to 0. The learning rate is 0.001 with decay, so that $\eta_t = 0.001/(1 + 5t \cdot 10^{-7})$. We use no momentum or weight regularization.



(a) (b) (c)

Figure 3. Denoising near a piece of noise. (a) shows a $64 \times 64$ image region with dirt occluders (top), and target ground truth clean image (bottom). (b) and (c) show the results obtained using non-convolutional and convolutionally trained networks, respectively. The top row shows the full output after averaging. The bottom row shows the signed error of each individual patch prediction for all $8 \times 8$ patches obtained using a sliding window in the boxed area, displayed as a montage. The errors from the convolutionally-trained network (c) are less correlated with one another compared to (b), and cancel to produce a better average.

## 2.3. Effect of Convolutional Architecture

A key improvement of our method over [2] is that we minimize the error of the final image prediction, whereas [2] minimizes the error only of individual patches. We found this difference to be crucial to obtain good performance on the corruption we address.

Since the middle layer convolution in our network has $1 \times 1$ spatial support, the network can be viewed as first patchifying the input, applying a fully-connected neural network to each patch, and averaging the resulting output patches. More explicitly, we can split the input image $x$ into stride-1 overlapping patches $\{x_p\} = \texttt{patchify}(x)$, and predict a corresponding clean patch $y_p = f(x_p)$ for each $x_p$ using a fully-connected multilayer network $f$. We then form the predicted image $y = \texttt{depatchify}(\{y_p\})$ by taking the average of the patch predictions at pixels where they overlap. In this context, the convolutional network $F$ can be expressed in terms of the patch-level network $f$ as $F(x) = \texttt{depatchify}(\{f(x_p) : x_p \in \texttt{patchify}(x)\})$.

In contrast to [2], our method trains the full network $F$, *including patchification and depatchification*. This drives a decorrelation of the individual predictions, which helps both to remove occluders as well as reduce blur in the final output. To see this, consider two adjacent patches $y_1$ and $y_2$ with overlap regions $y_{o1}$ and $y_{o2}$, and desired output $y_o^*$. If we were to train according to the individual predictions, the loss would minimize $(y_{o1} - y_o^*)^2 + (y_{o2} - y_o^*)^2$, the sum of their error. However, if we minimize the error of their average, the loss becomes $\left(\frac{y_{o1}+y_{o2}}{2} - y_o^*\right)^2 = \frac{1}{4}[(y_{o1} - y_o^*)^2 + (y_{o2} - y_o^*)^2 + 2(y_{o1} - y_o^*)(y_{o2} - y_o^*)]$.

The new mixed term pushes the individual patch errors in opposing directions, encouraging them to decorrelate.

Fig. 3 depicts this for a real example. When trained at the patch level, as in the system described by [2], each prediction leaves the same residual trace of the noise, which their average then maintains (b). When trained with our convolutional network, however, the predictions decorrelate where not perfect, and average to a better output (c).

### 2.4. Test-Time Evaluation

By restricting the middle layer kernels to have $1 \times 1$ spatial support, our method requires no synchronization until the final summation in the last layer convolution. This makes our method natural to parallelize, and it can easily be run in sections on large input images by adding the outputs from each section into a single image output buffer. Our Matlab GPU implementation is able to restore a $3888 \times 2592$ color image in 60s using a nVidia GTX 580, and a $1280 \times 720$ color image in 7s.

## 3. Training Data Collection

The network has 753,664 weights and 1,216 biases which need to be set during training. This requires a large number of training patches to avoid over-fitting. We now describe the procedures used to gather the corrupted/clean patch pairs[2] used to train each of the dirt and rain models.

### 3.1. Dirt

To train our network to remove dirt noise, we generated clean/noisy image pairs by synthesizing dirt on images. Similarly to [9], we also found that dirt noise was well-modeled by an opacity mask and additive component, which we extract from real dirt-on-glass panes in a lab setup. Once we have the masks, we generate noisy images according to

$$I' = p\alpha D + (1 - \alpha)I$$

Here, $I$ and $I'$ are the original clean and generated noisy image, respectively. $\alpha$ is a transparency mask the same size as the image, and $D$ is the additive component of the dirt, also the same size as the image. $p$ is a random perturbation vector in RGB space, and the factors $p\alpha D$ are multiplied together element-wise. $p$ is drawn from a uniform distribution over (0.9, 1.1) for each of red, green and blue, then multiplied by another random number between 0 and 1 to vary brightness. These random perturbations are necessary to capture natural variation in the corruption and make the network robust to these changes.

To find $\alpha$ and $\alpha D$, we took pictures of several slide-projected backgrounds, both with and without a dirt-on-glass pane placed in front of the camera. We then solved a linear least-squares system for $\alpha$ and $\alpha D$ at each pixel; further details are included in the supplementary material.

[2]The corrupt patches still have many unaffected pixels, thus even without clean/clean patch pairs in the training set, the network will still learn to preserve clean input regions.

Figure 4. Examples of clean (top row) and corrupted (bottom row) patches used for training. The dirt (left column) was added synthetically, while the rain (right column) was obtained from real image pairs.

### 3.2. Water Droplets

Unlike the dirt, water droplets refract light around them and are not well described by a simple additive model. We considered using the more sophisticated rendering model of [8], but accurately simulating outdoor illumination made this inviable. Thus, instead of synthesizing the effects of water, we built a training set by taking photographs of multiple scenes with and without the corruption present. For corrupt images, we simulated the effect of rain on a window by spraying water on a pane of anti-reflective $MgF_2$-coated glass, taking care to produce drops that closely resemble real rain. To limit motion differences between clean and rainy shots, all scenes contained only static objects. Further details are provided in the supplementary material.

## 4. Baseline Methods

We compare our convolutional network against a non-convolutional patch-level network similar to [2], as well as three baseline approaches: median filtering, bilateral filtering [18, 15], and BM3D [3]. In each case, we tuned the algorithm parameters to yield the best qualitative performance in terms of visibly reducing noise while keeping clean parts of the image intact. On the dirt images, we used an $8 \times 8$ window for the median filter, parameters $\sigma_s = 3$ and $\sigma_r = 0.3$ for the bilateral filter, and $\sigma = 0.15$ for BM3D. For the rain images, we used similar parameters, but adjusted for the fact that the images were downsampled by half: $5 \times 5$ for the median filter, $\sigma_s = 2$ and $\sigma_r = 0.3$ for the bilateral filter, and $\sigma = 0.15$ for BM3D.


Figure 5. Example image containing dirt, and the restoration produced by our network. Note the detail preserved in high-frequency areas like the branches. The nonconvolutional network leaves behind much of the noise, while the median filter causes substantial blurring.

# 5. Experiments

## 5.1. Dirt

We tested dirt removal by running our network on pictures of various scenes taken behind dirt-on-glass panes. Both the scenes and glass panes were not present in the training set, ensuring that the network did not simply memorize and match exact patterns. We tested restoration of both real and synthetic corruption. Although the training set was composed entirely of synthetic dirt, it was representative enough for the network to perform well in both cases.

The network was trained using 5.8 million examples of $64 \times 64$ image patches with synthetic dirt, paired with ground truth clean patches. We trained only on examples where the variance of the clean $64 \times 64$ patch was at least 0.001, and also required that at least 1 pixel in the patch had a dirt $\alpha$-mask value of at least 0.03. To compare to [2], we trained a non-convolutional patch-based network with patch sizes corresponding to our convolution kernel sizes, using 20 million $16 \times 16$ patches.

### 5.1.1 Synthetic Dirt Results

We first measure quantitative performance using synthetic dirt. The results are shown in Table 1. Here, we generated test examples using images and dirt masks held out from the training set, using the process described in Section 3.1. Our convolutional network substantially outperforms its patch-based counterpart. Both neural networks are much better than the three baselines, which do not make use of the structure in the corruption that the networks learn.

| **PSNR** | Input | Ours | Nonconv | Median | Bilateral | BM3D |
|---|---|---|---|---|---|---|
| Mean | 28.93 | **35.43** | 34.52 | 31.47 | 29.97 | 29.99 |
| Std.Dev. | 0.93 | **1.24** | 1.04 | 1.45 | 1.18 | 0.96 |
| Gain | - | **6.50** | 5.59 | 2.53 | 1.04 | 1.06 |

Table 1. PSNR for our convolutional neural network, nonconvolutional patch-based network, and baselines on a synthetically generated test set of 16 images (8 scenes with 2 different dirt masks). Our approach significantly outperforms the other methods.

We also applied our network to two types of artificial noise absent from the training set: synthetic "snow" made from small white line segments, and "scratches" of random cubic splines. An example region is shown in Fig. 6. In contrast to the gain of +6.50 dB for dirt, the network leaves these corruptions largely intact, producing near-zero PSNR gains of -0.10 and +0.30 dB, respectively, over the same set of images. This demonstrates that the network learns to remove dirt specifically.

### 5.1.2 Dirt Results

Fig. 5 shows a real test image along with our output and the output of the patch-based network and median filter. Because of illumination changes and movement in the scenes, we were not able to capture ground truth images for quantitative evaluation. Our method is able to remove most of the corruption while retaining details in the image, particularly around the branches and shutters. The non-convolutional

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

Figure 6. Our dirt-removal network applied to an image with (a) no corruption, (b) synthetic dirt, (c) artificial "snow" and (d) random "scratches." Because the network was trained to remove dirt, it successfully restores (b) while leaving the corruptions in (c,d) largely untouched. Top: Original images. Bottom: Output.

network leaves many pieces of dirt behind, while the median filter loses much detail present in the original. Note also that the neural networks leave already-clean parts of the image mostly untouched.

Two common causes of failure of our model are large corruption, and very oddly-shaped or unusually colored corruption. Our $16 \times 16$ input kernel support limits the size of corruption recognizable by the system, leading to the former. The latter is caused by a lack of generalization: although we trained the network to be robust to shape and color by supplying it a range of variations, it will not recognize cases too far from those seen in training. Another interesting failure of our method appears in the bright orange cones in Fig. 5, which our method reduces in intensity — this is due to the fact that the training dataset did not contain any examples of such fluorescent objects. More examples are provided in the supplementary material.

## 5.2. Rain

We ran the rain removal network on two sets of test data: (*i*) pictures of scenes taken through a pane of glass on which we sprayed water to simulate rain, and (*ii*) pictures of scenes taken while it was actually raining, from behind an initially clean glass pane. Both sets were composed of real-world outdoor scenes not in the training set.

We trained the network using 6.5 million examples of $64 \times 64$ image patch pairs, captured as described in Section 3.2. Similarly to the dirt case, we used a variance threshold of 0.001 on the clean images and required each training pair to have at least 1 pixel difference over 0.1.

### 5.2.1 Water Droplets Results

Examples of our network removing sprayed-on water is shown in Fig. 7. As was the case for the dirt images, we were not able to capture accurate ground truth due to illumination changes and subject motion. Since we also do not have synthetic water examples, we analyze our method in this mode only qualitatively.

Figure 8. Shot from the rain video sequence (see supplementary video), along with the output of our network. Note each frame is processed independently, without using any temporal information or background subtraction.

As before, our network is able to remove most of the water droplets, while preserving finer details and edges reasonably well. The non-convolutional network leaves behind additional droplets, e.g. by the subject's face in the top image; it performs somewhat better in the bottom image, but blurs the subject's hand. The median filter must blur the image substantially before visibly reducing the corruption. However, the neural networks mistake the boltheads on the bench for raindrops, and remove them.

Despite the fact that our network was trained on static scenes to limit object motion between clean/noisy pairs, it still preserves animate parts of the images well: The face and body of the subject are reproduced with few visible artifacts, as are grass, leaves and branches (which move from wind). Thus the network can be applied to many scenes substantially different from those seen in training.

### 5.2.2 Real Rain Results

A picture taken using actual rain is shown in Fig. 8. We include more pictures of this time series as well as a video in the supplementary material. Each frame of the video was presented to our algorithm independently; no temporal filtering was used. To capture the sequence, we set a clean glass pane on a tripod and allowed rain to fall onto it, taking pictures at 20s intervals. The camera was placed 0.5m behind the glass, and was focused on the scene behind.

Even though our network was trained using sprayed-on water, it was still able to remove much of the actual rain. The largest failures appear towards the end of the sequence, when the rain on the glass is very heavy and starts to agglomerate, forming droplets larger than our network can handle. Although this is a limitation of the current approach, we hope to address such cases in future work.

Lastly, in addition to pictures captured with a DSLR, in Fig. 9 we apply our network to a picture taken using a smartphone on a train. While the scene and reflections are preserved, raindrops on the window are removed, though a few small artifacts do remain. This demonstrates that our model is able to restore images taken by a variety of camera types.

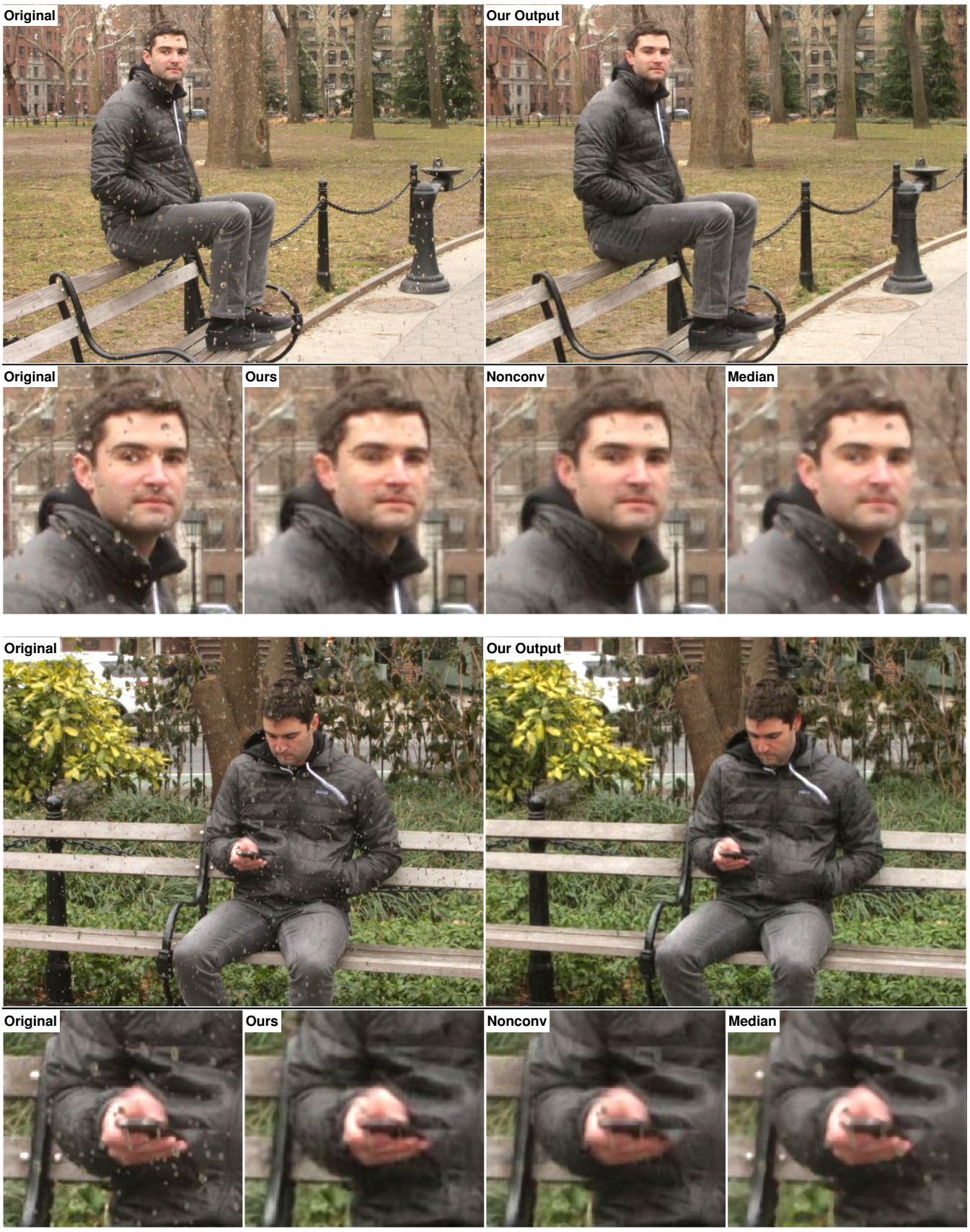



Figure 7. Our network removes most of the water while retaining image details; the non-convolutional network leaves more droplets behind, particularly in the top image, and blurs the subject's fingers in the bottom image. The median filter blurs many details, but still cannot remove much of the noise.

## 6. Summary

We have introduced a method for removing rain or dirt artifacts from a single image. Although the problem appears underconstrained, the artifacts have a distinctive appearance which we are able to learn with a specialized convolutional network and a carefully constructed training set. Results on real test examples show most artifacts being removed without undue loss of detail, unlike existing approaches such as median or bilateral filtering. Using a convolutional network accounts for the error in the final image prediction, providing a significant performance gain over the corresponding patch-based network.

The quality of the results does however depend on the statistics of test cases being similar to those of the training set. In cases where this does not hold, we see significant artifacts in the output. This can be alleviated by expanding the diversity and size of the training set. A second issue is that the corruption cannot be much larger than the training patches. This means the input image may need to be downsampled, e.g. as in the rain application, leading to a loss of resolution relative to the original.

Although we have only considered day-time outdoor shots, the approach could be extended to other settings such as indoor or night-time, given suitable training data. It could also be extended to other problem domains such as scratch removal or color shift correction.

Our algorithm provides the underlying technology for a number of potential applications such as a digital car windshield to aid driving in adverse weather conditions, or enhancement of footage from security or automotive cameras in exposed locations. These would require real-time performance not obtained by our current implementation. High-performance low-power neural network implementations such as the NeuFlow FPGA/ASIC [6] would make real-time embedded applications of our system feasible.

## Acknowledgements

The authors would like to thank Ross Fadeley and Dan Foreman-Mackay for their help modeling, as well as David W. Hogg and Yann LeCun for their insight and suggestions. Financial support for this project was provided by Microsoft Research and NSF IIS 1124794 & IIS 1116923.

## References

[1] P. Barnum, S. Narasimhan, and K. Takeo. Analysis of rain and snow in frequency space. *IJCV*, 86(2):256–274, 2010. 2

[2] H. Burger, C. Schuler, and S. Harmeling. Image denoising: Can plain neural networks compete with BM3D? In *CVPR*, 2012. 2, 3, 4, 5

[3] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian. Image denoising with block-matching and 3D filtering. In *Proc. SPIE Electronic Imaging*, 2006. 1, 4

[4] B. Dong, H. Ji, J. Li, Z. Shen, and Y. Xu. Wavelet frame based blind image inpainting. *Applied and Comp'l Harmonic Analysis*, 32(2):268–279, 2011. 2

[5] M. Elad and M. Aharon. Image denoising via learned dictionaries and sparse representation. In *CVPR*, 2006. 2

[6] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun. NeuFlow: A runtime reconfigurable dataflow processor for vision. In *IEEE Workshop on Embedded Computer Vision (ECV at CVPR)*, 2011. 8

[7] K. Garg and S. Nayar. Detection and removal of rain from videos. In *CVPR*, pages 528–535, 2004. 2

Figure 9. Top: Smartphone shot through a rainy window on a train. Bottom: Output of our algorithm.

[8] J. Gu, R. Ramamoorthi, P. Belhumeur, and S. Nayar. Dirty Glass: Rendering Contamination on Transparent Surfaces. In *Eurographics*, Jun 2007. 4

[9] J. Gu, R. Ramamoorthi, P. Belhumeur, and S. Nayar. Removing Image Artifacts Due to Dirty Camera Lenses and Thin Occluders. *SIGGRAPH Asia*, Dec 2009. 2, 4

[10] V. Jain and S. Seung. Natural image denoising with convolutional networks. In *NIPS*, 2008. 2

[11] J. Jancsary, S. Nowozin, and C. Rother. Loss-specific training of non-parametric image restoration models: A new state of the art. In *ECCV*, 2012. 2

[12] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, Nov 1998. 2

[13] A. Levin and B. Nadler. Natural image denoising: Optimality and inherent bounds. In *CVPR*, 2011. 1

[14] B. A. Olshausen and D. J. Field. Sparse coding with an overcomplete basis set: A strategy employed by V1? *Vision Research*, 37(23):3311–3325, 1997. 2

[15] S. Paris and F. Durand. A fast approximation of the bilateral filter using a signal processing approach. In *ECCV*, pages IV: 568–580, 2006. 4

[16] J. Portilla, V. Strela, M. J. Wainwright, and E. P. Simoncelli. Image denoising using scale mixtures of Gaussians in the wavelet domain. *IEEE Trans Image Processing*, 12(11):1338–1351, November 2003. 2

[17] M. Roser and A. Geiger. Video-based raindrop detection for improved image registration. In *ICCV Workshop on Video-Oriented Object and Event Classification*, Kyoto, Japan, September 2009. 2

[18] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *CVPR*, 1998. 4

[19] R. G. Willson, M. W. Maimone, A. E. Johnson, and L. M. Scherr. An optical model for image artifacts produced by dust particles on lenses. In *i-SAIRAS*, volume 1, 2005. 2

[20] J. Xie, L. Xu, and E. Chen. Image denoising and inpainting with deep neural networks. In *NIPS*, 2012. 2

[21] S. Zhang and E. Salari. Image denosing using a neural network based non-linear filter in the wavelet domain. In *ICASSP*, 2005. 2

[22] C. Zhou and S. Lin. Removal of image artifacts due to sensor dust. In *CVPR*, 2007. 2

[23] S. C. Zhu and D. Mumford. Prior learning and gibbs reaction-diffusion. *PAMI*, 19(11):1236–1250, 1997. 2

[24] D. Zoran and Y. Weiss. From learning models of natural image patches to whole image restoration. In *ICCV*, 2011. 2