IMAGE DEBLURRING

November 14, 2016

Team : Deep Archers

Members :

Mahankali Saketh

Ravi Bansal

Kumar Ayush

Ramanth Gajula

Siva Kesava Reddy Kakarla

Aditya Mantha

Varun Rawal

Gaurav Jain

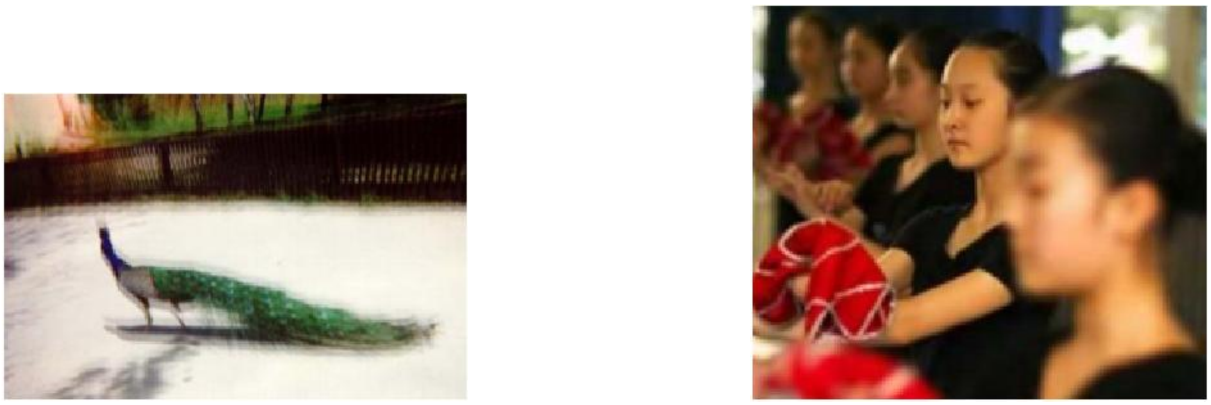
Raghav Agarwal

Abhishu Khekre

INTRODUCTION - PROBLEM DEFINITION

When we use a camera, we want the recorded image to be a faithful representation of the scene that we see - but every image is more or less blurry. Scene motion, Defocusing , camera shake might be some of the causes for this blurry effect. Figure 1 shows some examples of blurred images corresponding to the above mentioned causes.Handling these problems could raise a broad set of artifacts related to image content. One way to remove these artifacts is via generative models. These models are usually built upon strong assumptions, such as identical and independently distributed noise.But these assumptions seem to be fragile in the real world images.

Luckily , many of these image and video degradation processes can be modeled as translation-invariant convolution. To restore the quality of the images , the inverse process, i.e., deconvolution, becomes an important tool.With this motivation , we try to address the problem of image deblurring by using a convolutional neural network to learn this blurring process and then we proceed to use this knowledge to deblur some images and evaluate our model.



(a)

(b)

Figure 1: Examples of blurred images

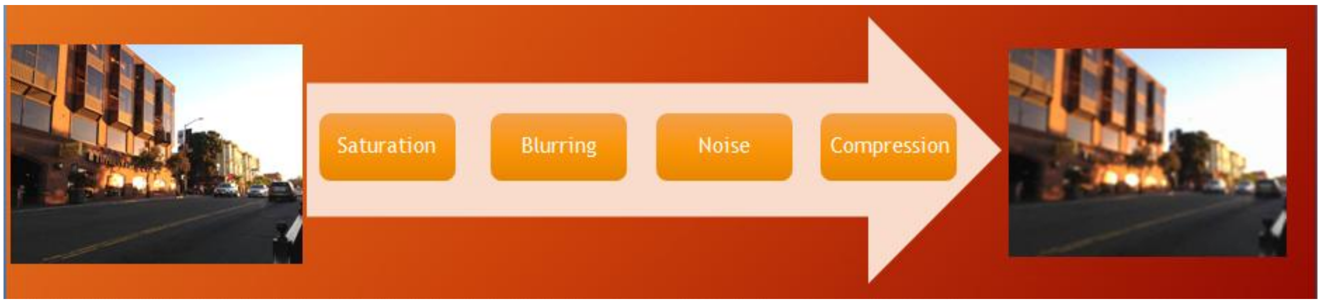


Figure 2

DATASET GENERATION

Before we can deblur an image, we must have a mathematical model that relates the given blurred image to the unknown true image.To generate the images that constitute our dataset, we simulate different type of blurring processes such as saturation, camera noise, and compression artifacts on flickr dataset.

$$y = \psi_b(\phi(\alpha x \times k + n))$$

where αx represents the latent sharp image , k represents the convolutional kernel , $\phi(.)$ represents the clipping function to model saturation and $\psi_b(.)$ represents a nonlinear compression operator.This process in summarized in the Figure 2.

Even with y and k , getting the value of αx is not tractable .So our goal is to restore the clipped input

DEBLURRING AS A DECONVOLUTION TASK

The deconvolution task can be approximated by a convolutional network by nature. To see how , consider the following simple linear blur model

$$y = x \circledast k$$

This convolution can be transformed into multiplication in frequency domain.

$$\mathscr{F}(y) = \mathscr{F}(x).\mathscr{F}(k)$$

where $\mathscr{F}(.)$ is the fourier transform

Now the equation can be rewritten as

$$x = \mathscr{F}^{-1}(\mathscr{F}(y)/\mathscr{F}(k)) = \mathscr{F}^{-1}(1/\mathscr{F}(k)) \circledast y$$

here $\mathscr{F}^{-1}(.)$ is the inverse fourier transform The above expression is equivalent to

$$x = k^i \circledast y$$

where k^i is the pseudo inverse kernel.

METHODOLOGY

The deconvolution task can be approximated by a convolutional network as seen in previous section.The complete neural network shown in figure 3 . The whole neural network can be thought of as 2 parts , first 2 layers are for actual deconvolution and next 2 layers are for denoising .The first part of network is a deconvolution convolutional neural network(DCNN) and the second part of neural network is the denoising network or outlier rejection convolutional neural network(ODCNN). This network can be expressed as $h_0 = y$, $h_3 = W_3 \ast h_2$ and

$$h_i = (W_i \ast h_{i-1} + b_{i-1}), i \in \{1,2\}$$

The CNN used majorily consists of four hidden layers.First 2 layers consitute the DCNN and second next two constitute ODCNN. In an abstract view , DCNN takes an input patch of size 184 x 184 and produces an output map of size 49 x 49 x 512 . This is fed as input to the ODCNN network which gives an output of size 56 x 56.Below is the elaborate description of each hidden layer.

Initially we have a 184 x 184 representation of image .The first hidden layer h1 is generated by applying 38 large-scale one-dimensional kernels of size 121 x 1 . After application of these kernels, we obtain 38 maps of size 64 x 184 i.e. we obtain 64 x 184 x 38 sized data as input for second layer h2.

Now we apply 38 one-dimensional kernels of size 1 x 121 x 38 to each of the 38 maps in h1. So now we obtain 38 maps of size 64 x 64 . Now we use 512 kernels each of size 16 x 16 x 38. Note that we have 64 x 64 x 38 map and by using a 16 x 16 x 38 kernel , we obtain a 49 x 49 map . Since 512 such kernels are used , we obtain a 49 x 49 x 512 map.

The next layer is generated by applying 512 kernels of size 1 x 1 x 512 . So we get a 49 x 49 x 512 map again . The next layer is generated by applying a 8 x 8 x 512 kernel to the 49 x 49 x

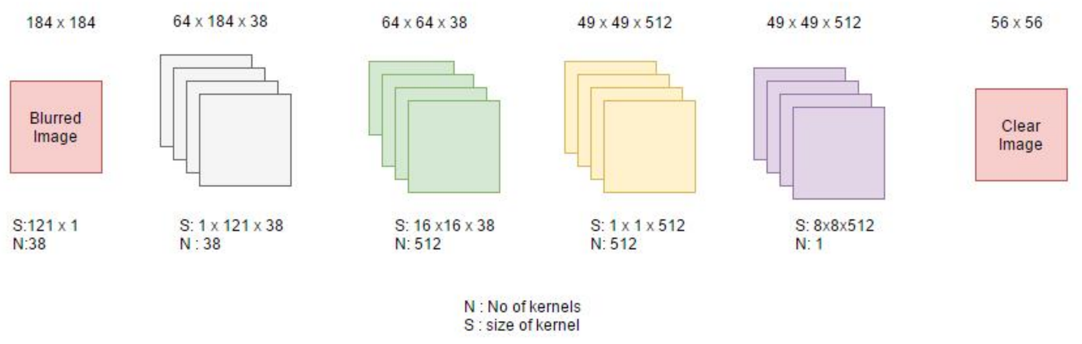


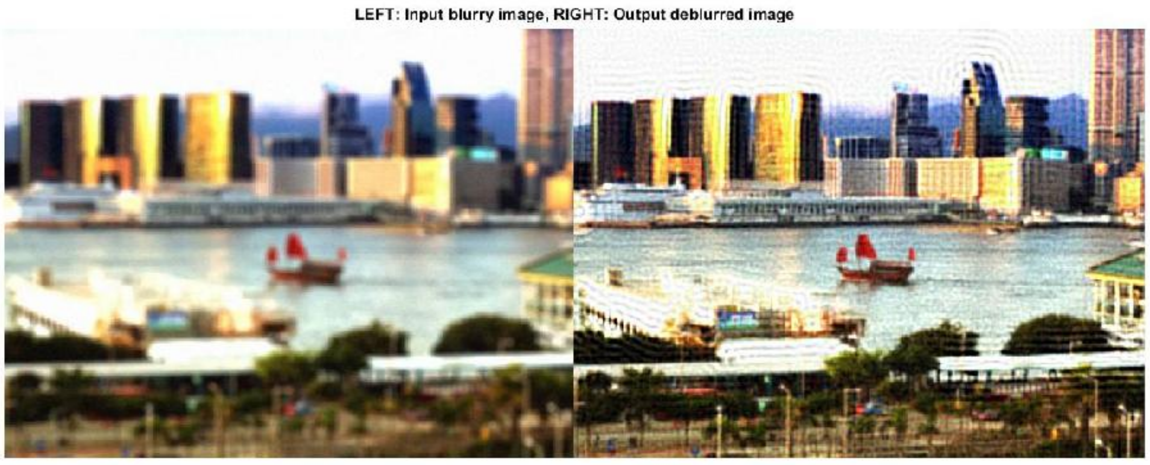
Figure 3

512 image and we zero pad the obtained image to get 56 x 56 image (zero padding is done by adding 8 rows before first row and after last row and adding 8 columns before first column and after last column)

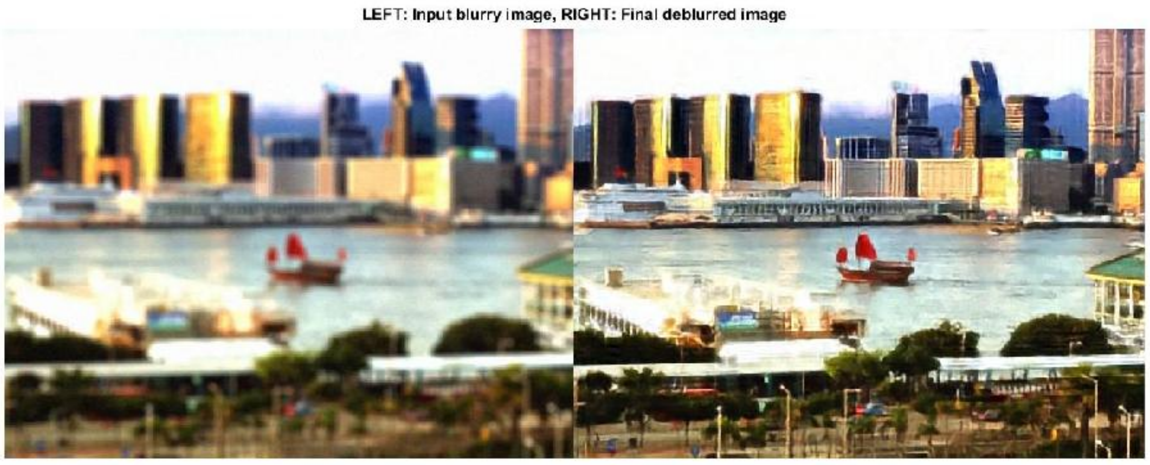
TRAINING AND RESULTS

We blurred the natural images for training , thus it is easy to obtain a large number of data.Even with the huge flickr data , different sizes of images pose a problem. So patches of some particular size(184 x 184) are taken from the images. Specifically, we use 2000 natural images downloaded from Flickr. 0.8 million patches are generated from these 2,500 images. We train the sub-networks separately. The CNN is trained using the initialization from separable inversion .The training samples contains all patches possibly with noise, saturation, and compression artifacts.

The training state consists of two main phases - training DCNN and training ODCNN.First we tried training only the DCNN network and .Figures [4-9](a) show the blurred and deblurred images after using just DCNN. Still some noise effects seem to exist after DCNN.So we proceed to train the ODCNN network with input as the output of DCNN . Figures [4-9](b) show the blurred and deblurred images of the whole network after addition of 2 hidden layers of ODCNN.In all the cases we observes that using ODCNN along with DCNN outperforms the output obtained using just DCNN.

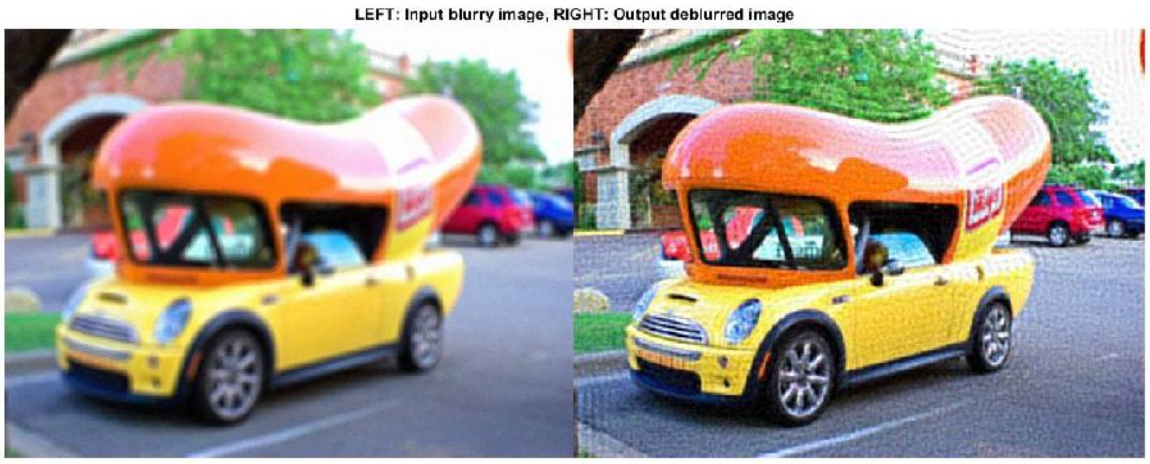


(a)

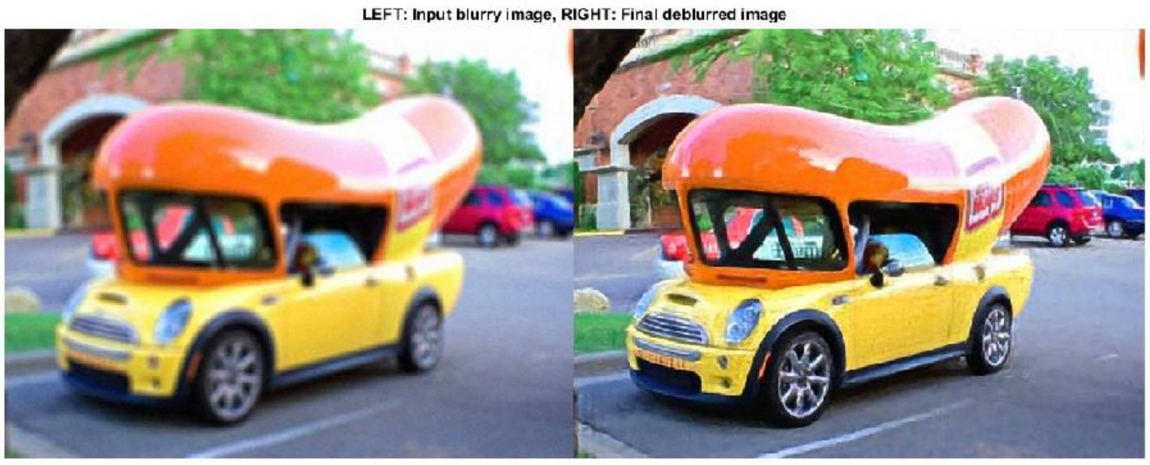


(b)

Figure 4: (a) input image and image obtained after DCNN (b) input image and image obtained after DCNN combined with ODCNN

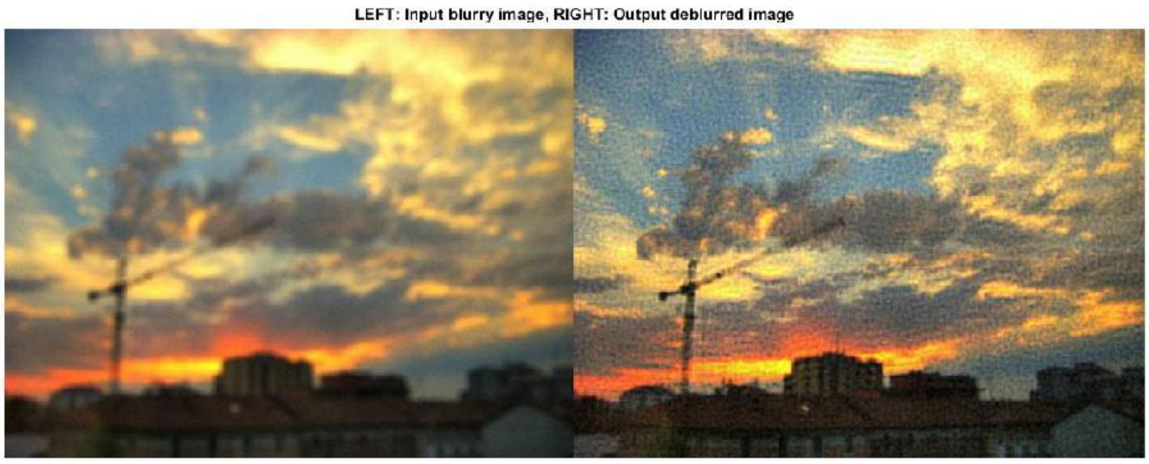


(a)

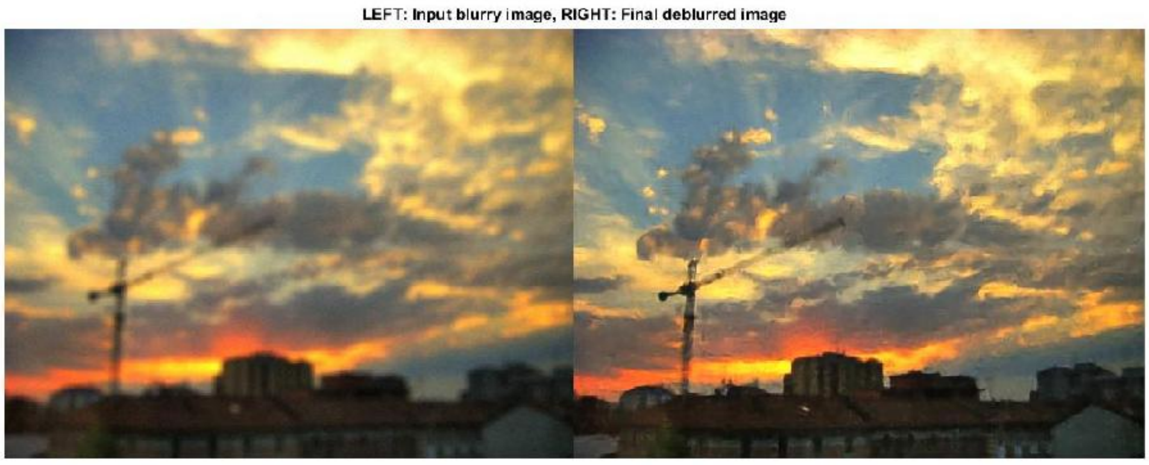


(b)

Figure 5: (a) input image and image obtained after DCNN (b) input image and image obtained after DCNN combined with ODCNN

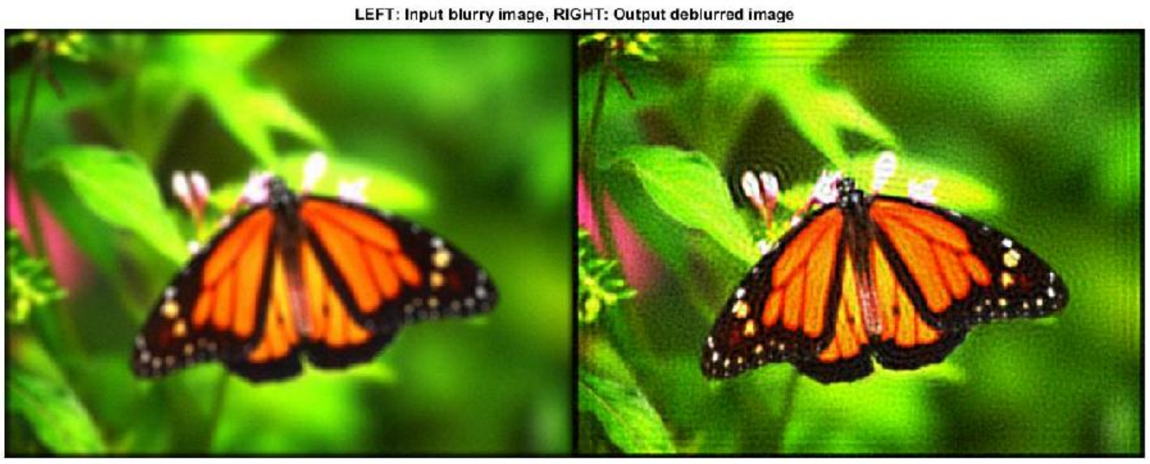


(a)

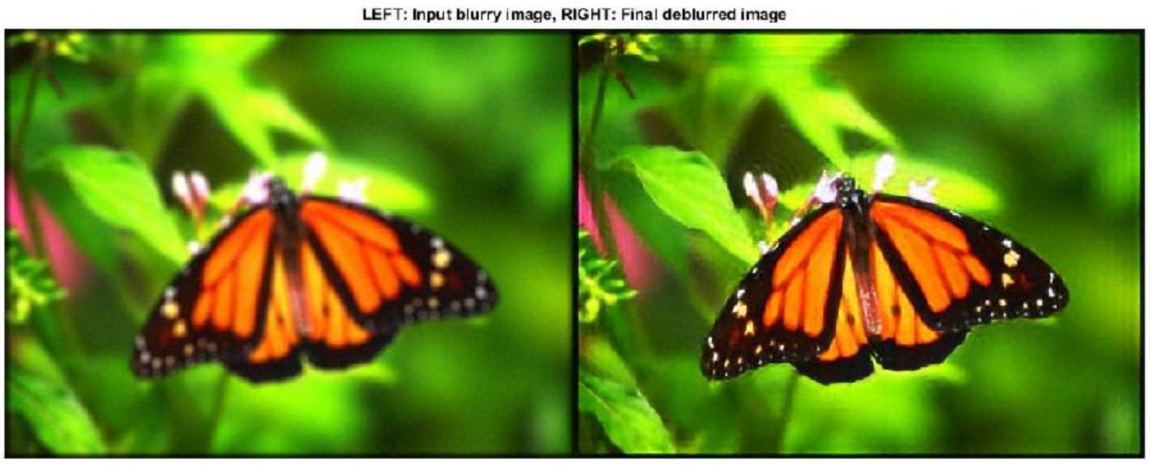


(b)

Figure 6: (a) input image and image obtained after DCNN (b) input image and image obtained after DCNN combined with ODCNN

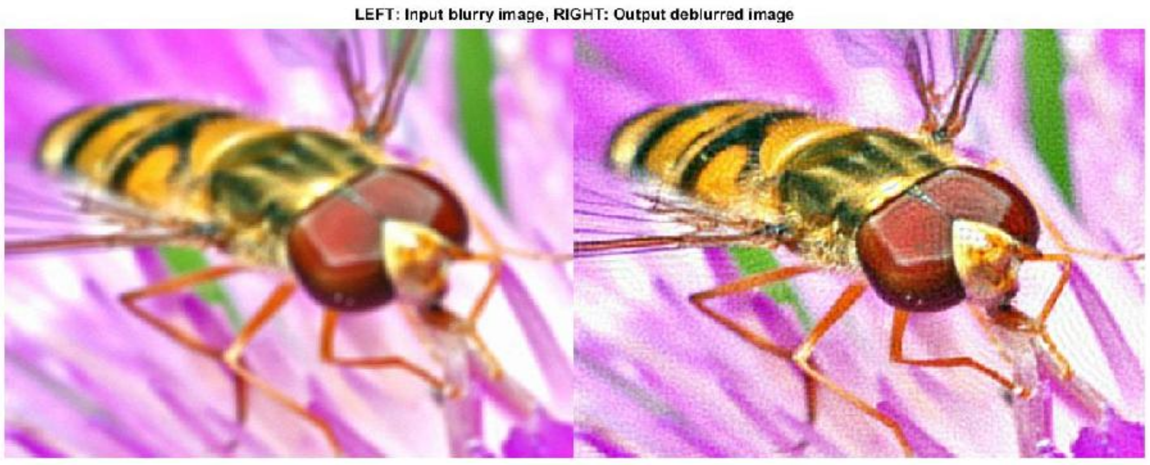


(a)

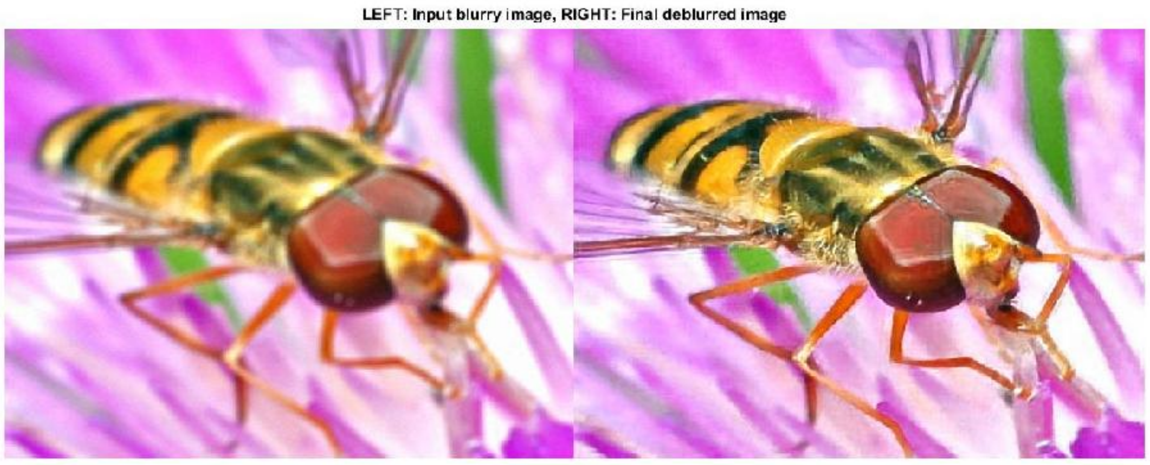


(b)

Figure 7: (a) input image and image obtained after DCNN (b) input image and image obtained after DCNN combined with ODCNN

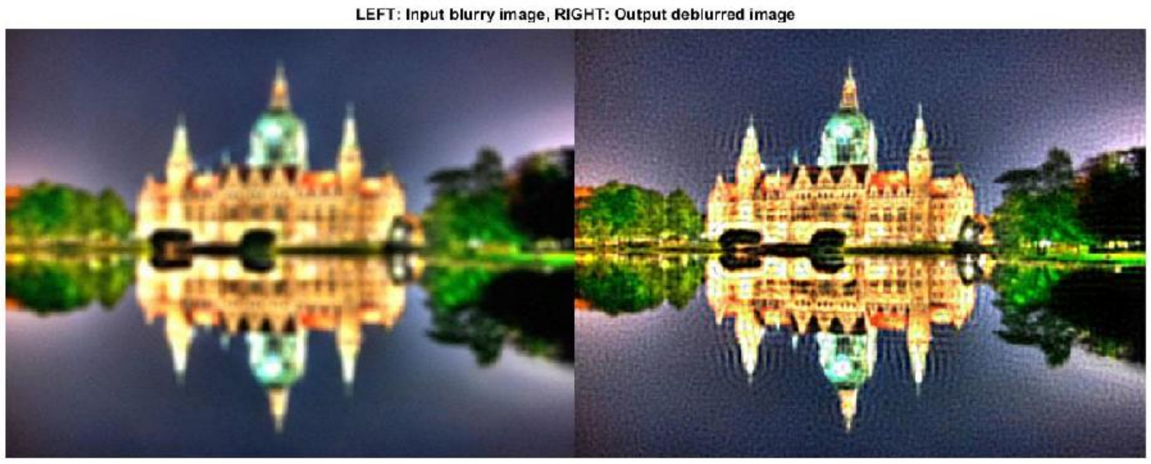


(a)

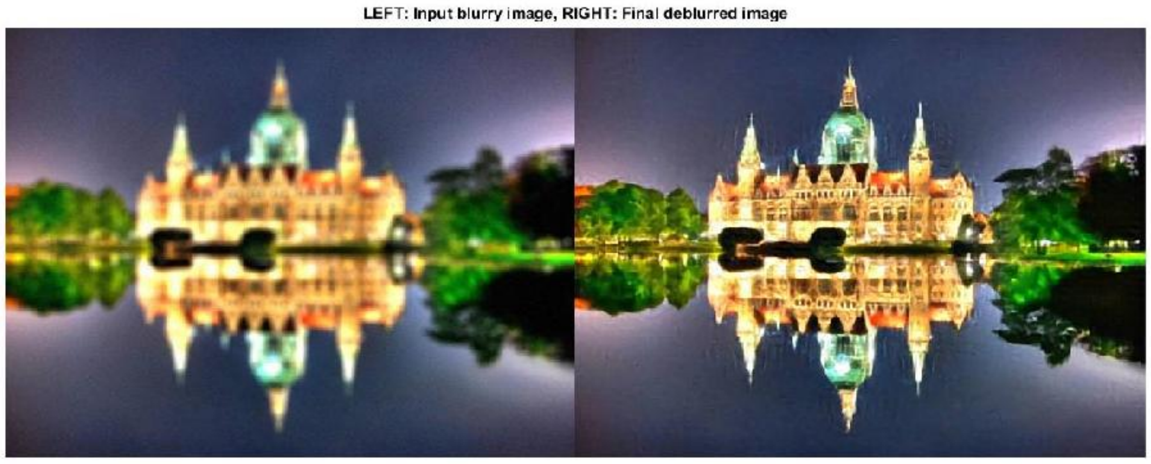


(b)

Figure 8: (a) input image and image obtained after DCNN (b) input image and image obtained after DCNN combined with ODCNN



(a)



(b)

Figure 9: (a) input image and image obtained after DCNN (b) input image and image obtained after DCNN combined with ODCNN