Image Smoothing via Unsupervised Learning Supplemental Material

1 Detection of the salient edge points

Our edge detector works by selecting edge points as described in Algorithm 1 and then filtering noisy edges with Algorithm 2.

To be specific, we select the salient edge points whose intensity is *relatively larger* than *most of its nearby points*. Such a design is based on the observation that what makes a point salient is its relative intensity compared to its local neighbourhood, instead of its absolute intensity. Because of this simple strategy, we are able to extract the points that may be weak in absolute value, but noticeable among its nearby points.

We further filter the noisy edge points by considering its *neighbourhood connectivity*. This is based on the principle that an edge is usually composed of lines or curves, thus an edge point should be connected with others to form some shape of lines. To implement this process, given a selected edge point, we iterate over 4 directions (horizontal, vertical and diagonal) to seek the connection with its neighbourhood. And if the current point fails to be related with enough points in all the directions, we decide to remove this candidate point.

Note that two examples of our detected edge maps B are demonstrated in the application section of image abstraction of the main paper. They conform with the image structures of the smoothed images produced by our network.

Algorithm 1 Edge selection for salient image points

procedure EDGESELECTION($E(I)$)
  Initialize:
  $t_1 = 25, t_2 = 20, t_3 = 200, w = 21$
  for $E_i(I) \in E(I)$ **do**
    $B_i = 0$
    if $E_i(I) > t_1$ **then**
      count = 0
      for $E_j(I) \in \mathcal{N}_w(E_i(I))$ **do**
        if $E_i(I) - E_j(I) > t_2$ **then**
          count++
      if count $> t_3$ **then**
        $B_i = 1$
  return B

Algorithm 2 Edge filtering based on neighbour connectivity

procedure EDGEFILTERING($B(I)$)
  Initialize:
  $t_1 = 51, t_2 = 20$
  for $B_i \in B$ **do**
    if $B_i == 1$ **then**
      failureNum = 0
      repeat
        count = 0
        for $j = i + 1$ to $i + t_1$ **do**
          if $B_j == 1$ **then**
            count++
          else
            break
        if count $< t_2$ **then**
          failureNum++
      until Iterate over 4 directions ▷ x, y, diagonal
      if failureNum == 4 **then**
        $B_i = 0$
  return B

2 Detection of the texture image structures

Our texture extraction method is based on an important observation that the most common texture structures usually share similar colors on its two sides, such as those many texture images shown in the main paper and supplemental material. Therefore, we simply designed our texture detection method as following three steps.

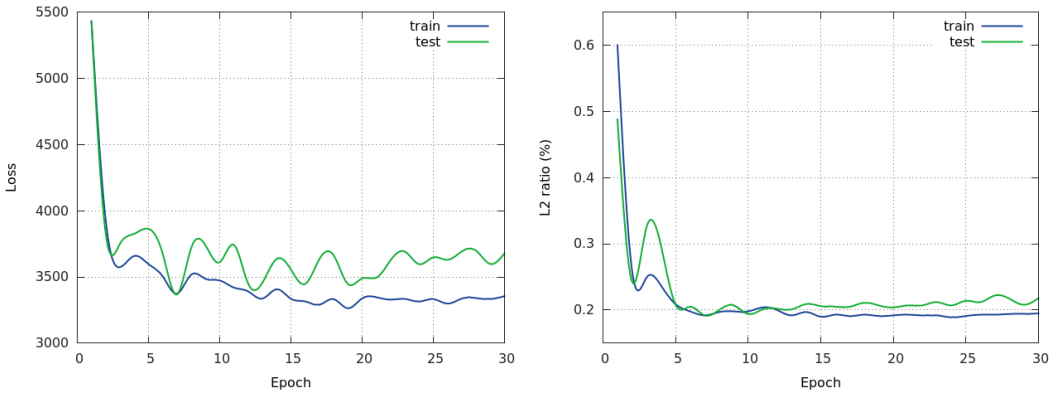
- Calculate the edge direction of the current point.
- Select two points on the perpendicular direction of current edge direction.
- Set current point as texture point if the above two points share similar colors.

The degree of edge direction can be calculated by computing the arc tangent of the y and x image gradient [5]. Note a few examples of detected texture structures are further filtered with Algorithm 2 and shown in the application section of texture removal of the main paper.

Note we don't argue for the optimal performance of the salient edge detection and texture extraction tasks, since which are not the focus of this paper. Moreover these tasks are very subjective, we believe the ideal solution should be manual labeling training images according to user preference, however such an edge-labeling work is prohibitively labor-intensive to be conducted with limited human resource. Hence we adopt these two simple algorithms to deal with it automatically, and find they work well enough for the majority of natural images. We believe more sophisticated and advanced approaches can be leveraged to obtain potentially better results.

3 Energy Minimization with Deep Learning

Our proposed whole objective function is very complex; it is highly non-convex and difficult to optimize. In this paper, we propose to use a deep learning technique to optimize it. A deep network can run very fast with the aid of modern GPUs, facilitating real-time interaction with users. Moreover, as a data-driven approach, deep networks can leverage a large volume of natural images for learning to optimize the complex energy function.



loss L_2 ratio

Figure 1: Illustration of the loss curve and L_2 ratio curve of the deep learning solver along the training procedure.

Specifically, we train a deep convolutional neural network which takes as input the raw images and produces smoothed images. In this section, we present how the energy function can be minimized using back-propagation in a neural work.

We implement the energy function as three differentiable loss layers in the deep learning framework to enable back-propagation for network training. Implementing the data term is relatively easy: the loss is the mean squared difference between the output image T predicted the input image I, and the derivate is simply

$$\frac{\partial \mathcal{E}_d}{\partial T_i} = \frac{2}{N} \cdot (T_i - I_i). \tag{1}$$

Note that the gradient is computed for each channel of T. We omit the channel index here and hereafter for brevity.

The flattening term calculates the weighted sum of the L_p norm difference between each pixel and its neighbouring pixels in the output image T, which is very resource-intensive. To tackle this challenge, we compute the corresponding loss in parallel for all the points in the image using GPU. When calculating the derivative of this loss with respect to each point i in T, we need to consider not only the loss it induced as the center pixel of the local window $\mathcal{N}_h(i)$, but also the loss it contributed as the neighbourhood of other pixels in their local windows. The complete derivate can be written as

$$\frac{\partial \mathcal{E}_f}{\partial T_i} = \frac{1}{N} \sum_{j \in \mathcal{N}_h(i)} (w_{i,j} \cdot g_{i,j} - \ w_{j,i} \cdot g_{j,i}), \tag{2}$$

where

$$g_{i,j} = p_i \cdot |T_i - T_j|^{p_i - 1} \cdot sign(T_i - T_j), \tag{3}$$

and $g_{j,i}$ can be computed accordingly. The edge-preserving term computes the mean squared differences of guidance image values on selected important edge points. The corresponding back propagation can be derived as

$$\frac{\partial \mathcal{E}_e}{\partial E_i(T)} = \frac{2}{N_e} \cdot B_i \cdot (E_i(T) - E_i(I)). \tag{4}$$

According to the chain rule, the gradients backpropagated to the smooth image are

$$\frac{\partial \mathcal{E}_e}{\partial T_i} = \sum_{j \in \mathcal{N}(i)} sign(\sum_c (T_{i,c} - T_{j,c})) \cdot (\frac{\partial \mathcal{E}_e}{\partial E_i(T)} + \frac{\partial \mathcal{E}_e}{\partial E_j(T)}) \tag{5}$$

Since our neural network only uses above the training signals, it does not require any ground truth images except for possibly some selected edge points in the edge-preserving term. The deep network is learned in an unsupervised (or possibly weakly-supervised) fashion.

4 Convergence of deep learning solver on our objective function

To explore the convergence of our algorithm optimized over the dynamic objective function, we show the training and test loss curve along the optimization process of the deep network in Figure 1. Along the 30 training epoches, we observe the convergence of the deep neural network, even if the energy is not guaranteed to decrease as the optimized objective function is continuously changing.

We also calculate the L_2 ratio as the rate of the area covered by L_2 norm to the entire image in Figure 1, whose trend is in accordance to the loss curve. The L_2 norm ratio is averaged among the whole training and testing datasets, which can be considered as a hyper-parameter that indicates the amount of potentially over-sharpened regions covered by L_2 norm among the entire dataset. It is observed that the ratio curve converges and becomes stable in the end instead of keeping fluctuating along the whole training process.

In a more general and bigger sense, the above two curves demonstrate that the deep neural network learns an implicit combination of L_2 and $L_{0.8}$ norm from the large corpus of training data.

5 Detailed network structure

A detailed network structure of the proposed network is shown in the above table. The source codes of our network implementation as well as the trained models will be released upon publication.

6 Effect of different L_p norms

One critical factor influencing the smoothness in the basic criterion is the choice of p for the L_p norm in $\mathcal{E}_f$. Applying an L_p norm with $p \leq 1$ promotes sparse solutions assuming Laplacian or hyper-Laplacian distributions. Larger p with $1 < p < 2$ steers the assumption towards Gaussian distribution ($p = 2$). The effects of different p values are illustrated in Figure 2, where the results are obtained by minimizing Equation 1 in the main paper. Here we observe that using a small p ($0 < p < 1$) flattens many image regions while preserving sharp edges (*e.g.* the rope the climber is holding). However, it also tends to generate stair-casing artifacts within moderately smooth regions by segmenting them into regions separated by spurious edges (see the regions around the climber with smooth gradation). In contrast, using a large p (*e.g.*, $p = 2$) blurs the image and wipes out more edges (*e.g.*, the rope disappears and boundaries between objects become less clear). Discussions about the L_p-norm regularization can also be found in [9, 4, 1].

Many variants of the basic criterion have been introduced to achieve different smoothing effects [6, 8, 10, 12, 13, 2]. One property shared by these previous works is that there is one single, fixed regularization across the whole image in their smoothing terms. However, as mentioned previously using a single regularization (*i.e.* choice of p) can result in obvious artifacts. It may either over-sharpen the moderately smooth regions and give rise to spurious edges (with a small p), or miss some important slender edges (with a large p). An ideal algorithm would both preserve the structures of interest and avoid generating spurious edges.

On the other hand, in distinct image smoothing tasks the image structure of interest may differ. For instance, texture smoothing aims at diminishing high-contrast small-scale edges, while traditional edge-preserving image smoothing methods which are based only on difference of brightness values tend to keep them. It is difficult to incorporate all the different smoothing effects into a single framework with a uniform regularizer.

Based on the above observations, we believe that a reasonable combination of different L_p norms over image regions characterising distinct degrees of smoothness is highly desirable, which motivates the design of our spatially-variant L_p flattening criterion. Such a combination should not only avoid the side-effects induced by a single regularization, but also facilitate different smoothing effects.

7 More visual comparison with state-of-the-art approaches

In Figure 3, we demonstrate one more example to compare our proposed algorithm with the previous state-of-the-art approaches, as a complement to Figure 2 in the main paper.

8 More visual results

In this section, we present the qualitative results on more than 100 images across multiple applications. Specifically,

- Figure 4 and 5 demonstrates 18 pairs of **edge-preserving smoothing** results.
- Figure 6, 7, 8, 9 and 10 shows 28 groups of edge-preserving smoothing results accompanied with its **image abstraction** effects.
- Figure 11, 12, 13, 14, 15, 16 and 17 displays 26 groups of **detail magnification** results.
- Figure 18 and 19 exhibit 20 pairs of **texture removal** results.
- Figure 20, 21, 22, 23 and 24 shows 24 groups of **content-aware image manipulation** results.

Table 1: The detailed network structure of the proposed network in Figure 7 of the main paper. "conv", "bat", "deconv","relu" represent convolution layer, batch normalization layer, deconvolution layer and ReLU activation respectively. each "residual block" consists of two convolution layers, both followed by batch normalization, the first of which is further followed by ReLU.

Name	Kernel	Stride	Dilation.	Ch I/O	InpRes	OutRes
conv1 + bat + relu	3×3	1	1	3/64	448×448	448×448
conv2 + bat + relu	3×3	1	1	64/64	448×448	448×448
conv3 + bat + relu	3×3	2	1	64/64	448×448	224×224
residual block1	3×3	1	2	64/64	224×224	224×224
residual block2	3×3	1	2	64/64	224×224	224×224
residual block3	3×3	1	4	64/64	224×224	224×224
residual block4	3×3	1	4	64/64	224×224	224×224
residual block5	3×3	1	8	64/64	224×224	224×224
residual block6	3×3	1	8	64/64	224×224	224×224
residual block7	3×3	1	16	64/64	224×224	224×224
residual block8	3×3	1	16	64/64	224×224	224×224
residual block9	3×3	1	1	64/64	224×224	224×224
residual block10	3×3	1	1	64/64	224×224	224×224
deconv4 + bat + relu	4×4	2	1	64/64	224×224	448×448
conv5 + bat + relu	3×3	1	1	64/64	448×448	448×448
conv6	3×3	1	1	64/3	448×448	448×448

References

[1] F. Bach, R. Jenatton, J. Mairal, G. Obozinski, et al. Structured sparsity through convex optimization. *Statistical Science*, 27(4):450–468, 2012. 3

[2] S. Bi, X. Han, and Y. Yu. An L_1 image transform for edge-preserving smoothing and scene-level intrinsic decomposition. *ACM Transactions on Graphics (TOG)*, 34(4):78, 2015. 3, 5

[3] H. Cho, H. Lee, H. Kang, and S. Lee. Bilateral texture filtering. *ACM Transactions on Graphics (TOG)*, 33(4):128, 2014. 5

[4] G. Chung and L. A. Vese. Image segmentation using a multilayer level-set approach. *Computing and visualization in science*, 12(6):267–285, 2009. 3

[5] Q. Fan, F. Zhong, D. Lischinski, D. Cohen-Or, and B. Chen. Jumpcut: non-successive mask transfer and interpolation for video cutout. *ACM Trans. Graph.*, 34(6):195, 2015. 1

[6] Z. Farbman, R. Fattal, D. Lischinski, and R. Szeliski. Edge-preserving decompositions for multi-scale tone and detail manipulation. *ACM Transactions on Graphics (TOG)*, 27(3), 2008. 3, 5

[7] B. Ham, M. Cho, and J. Ponce. Robust image filtering using joint static and dynamic guidance. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4823–4831, 2015. 5

[8] D. Min, S. Choi, J. Lu, B. Ham, K. Sohn, and M. N. Do. Fast global image smoothing based on weighted least squares. *IEEE Transactions on Image Processing*, 23(12):5638–5653, 2014. 3, 5

[9] V. S. Prasath, D. Vorotnikov, R. Pelapur, S. Jose, G. Seetharaman, and K. Palaniappan. Multiscale tikhonov-total variation image restoration using spatially varying edge coherence exponent. *IEEE Transactions on Image Processing*, 24(12):5220–5235, 2015. 3

[10] L. I. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 60(1-4):259–268, 1992. 3

[11] C. Tomasi. Bilateral filtering for gray and color images. In *Computer Vision, 1998. Sixth International Conference on*, pages 839–846. IEEE, 1998. 5

[12] L. Xu, C. Lu, Y. Xu, and J. Jia. Image smoothing via L_0 gradient minimization. In *ACM Transactions on Graphics (TOG)*, volume 30, page 174, 2011. 3, 5

[13] L. Xu, Q. Yan, Y. Xia, and J. Jia. Structure extraction from texture via natural variation measure. *ACM Transactions on Graphics (TOG)*, 2012. 3, 5

[14] F. Zhang, L. Dai, S. Xiang, and X. Zhang. Segment graph based image filtering: fast structure-preserving smoothing. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 361–369, 2015. 5

[15] Q. Zhang, X. Shen, L. Xu, and J. Jia. Rolling guidance filter. In *European Conference on Computer Vision (ECCV)*, pages 815–830, 2014. 5

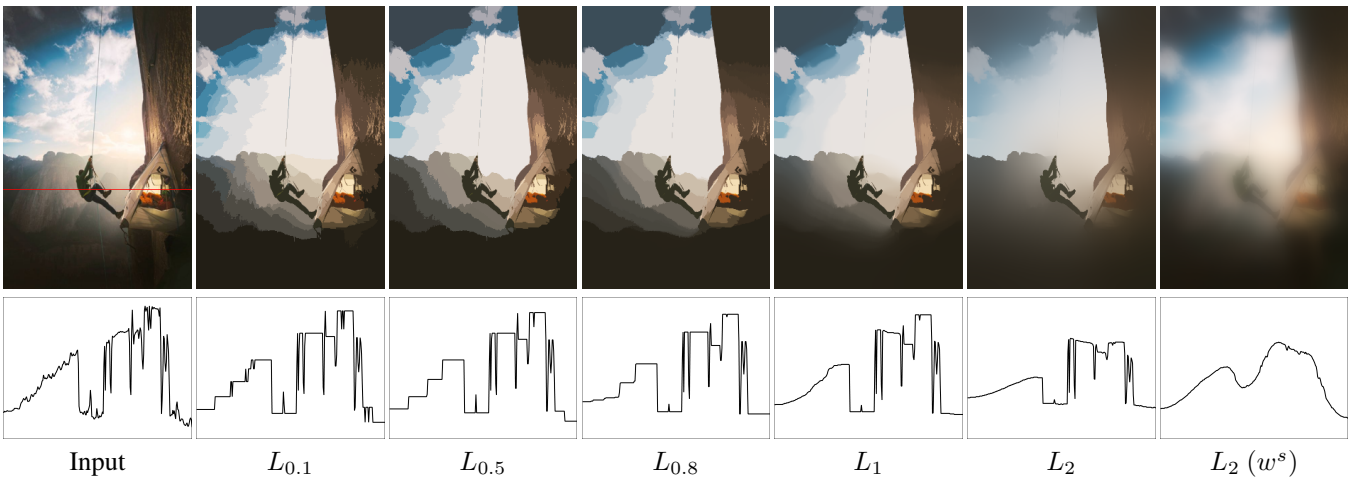


Figure 2: Image smoothing results with different L_p regularization in the smoothness term. Gaussian weights with color affinity is used for these results except for the last one which uses spatial affinity. The bottom row show the resultant brightness values on the 1D image slice indicated by the red line. Smaller p values can well flatten the details and preserve the important structures, but spurious edges may arise (known as the staircasing artifact). A large p value such as $p = 2$ may lead to blurry results and miss some important image structures. The smoothing results are optimized by the basic criterion.

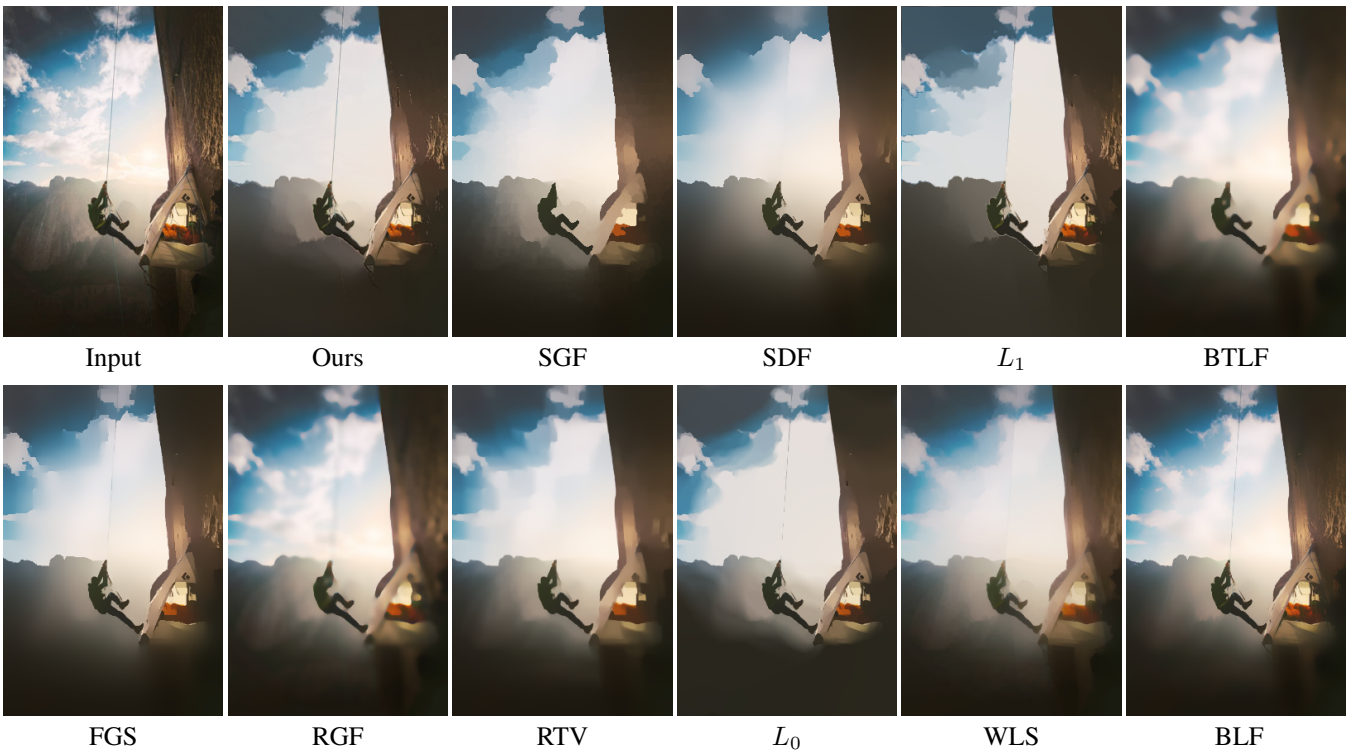


Figure 3: Visual comparison between our method and previous image smoothing methods, abbreviated as SGF [14], SDF [7], L_1 [2], BTLF [3], FGS [8], RGF [15], RTV [13], L_0 [12], WLS [6] and BLF [11]. Our smooth image is generated by depressing the low-amplitude details and preserve the high-contrast structures. As can be seen, in addition to achieving pleasing flattening effects, the slender rope is also maintained much better in our result than the others.

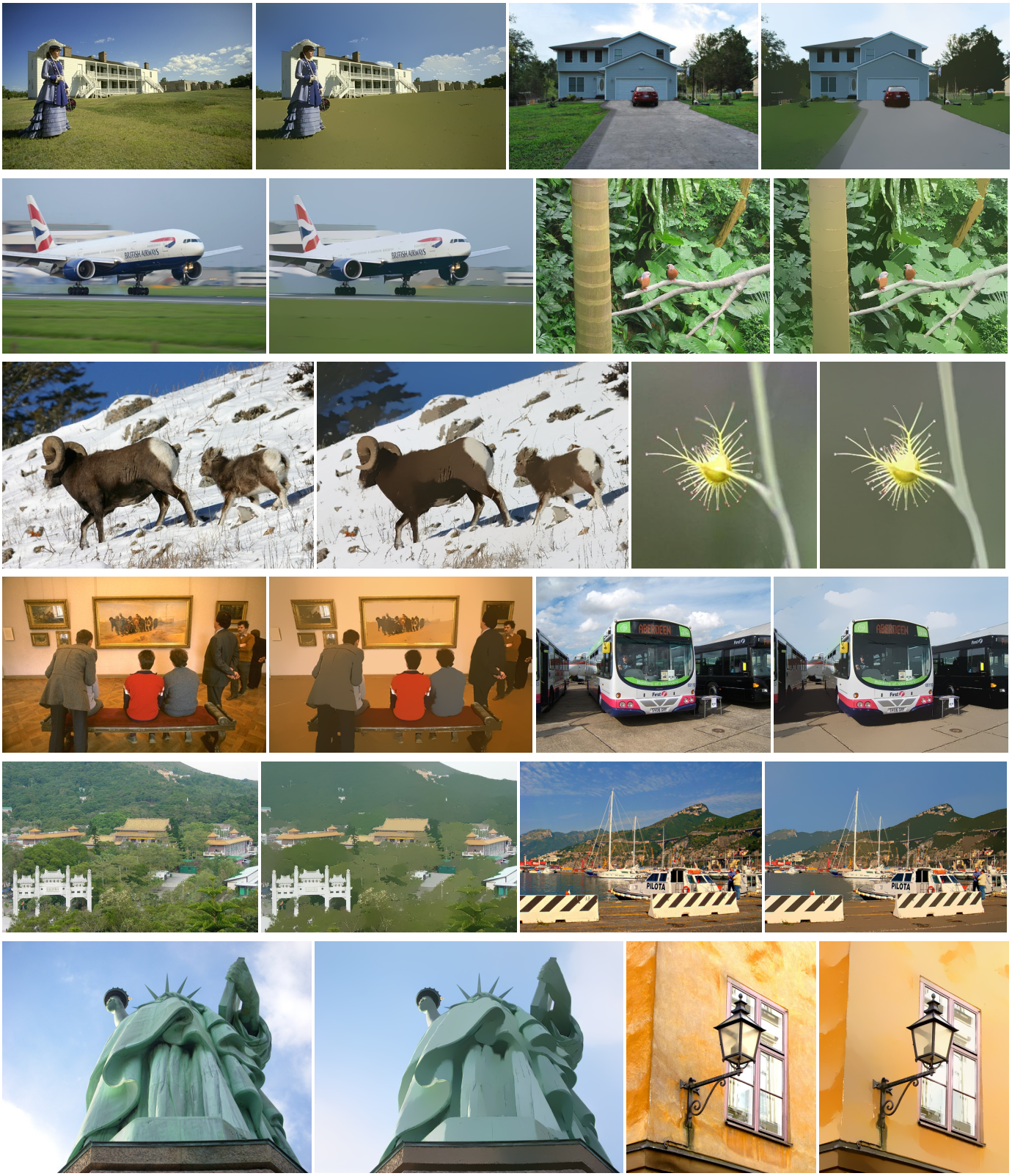


Figure 4: Edge-preserving smoothing results. Note all these results are generated without any parameter tweaking for this application.

Figure 5: Edge-preserving smoothing results. Note all these results are generated without any parameter tweaking for this application.

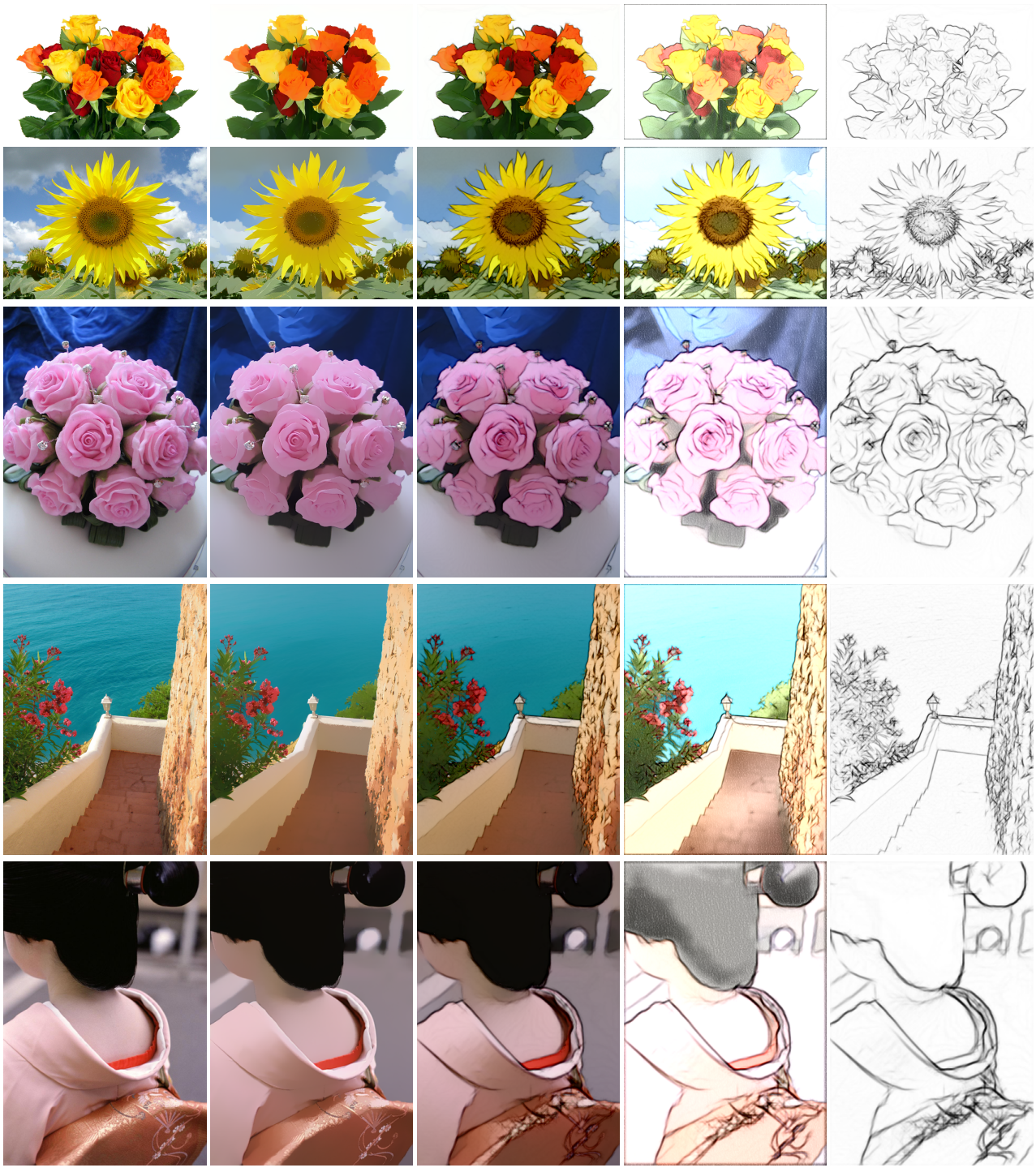


Figure 6: Image abstraction and pencil drawing results. Note all these results are generated without any parameter tweaking for this application.

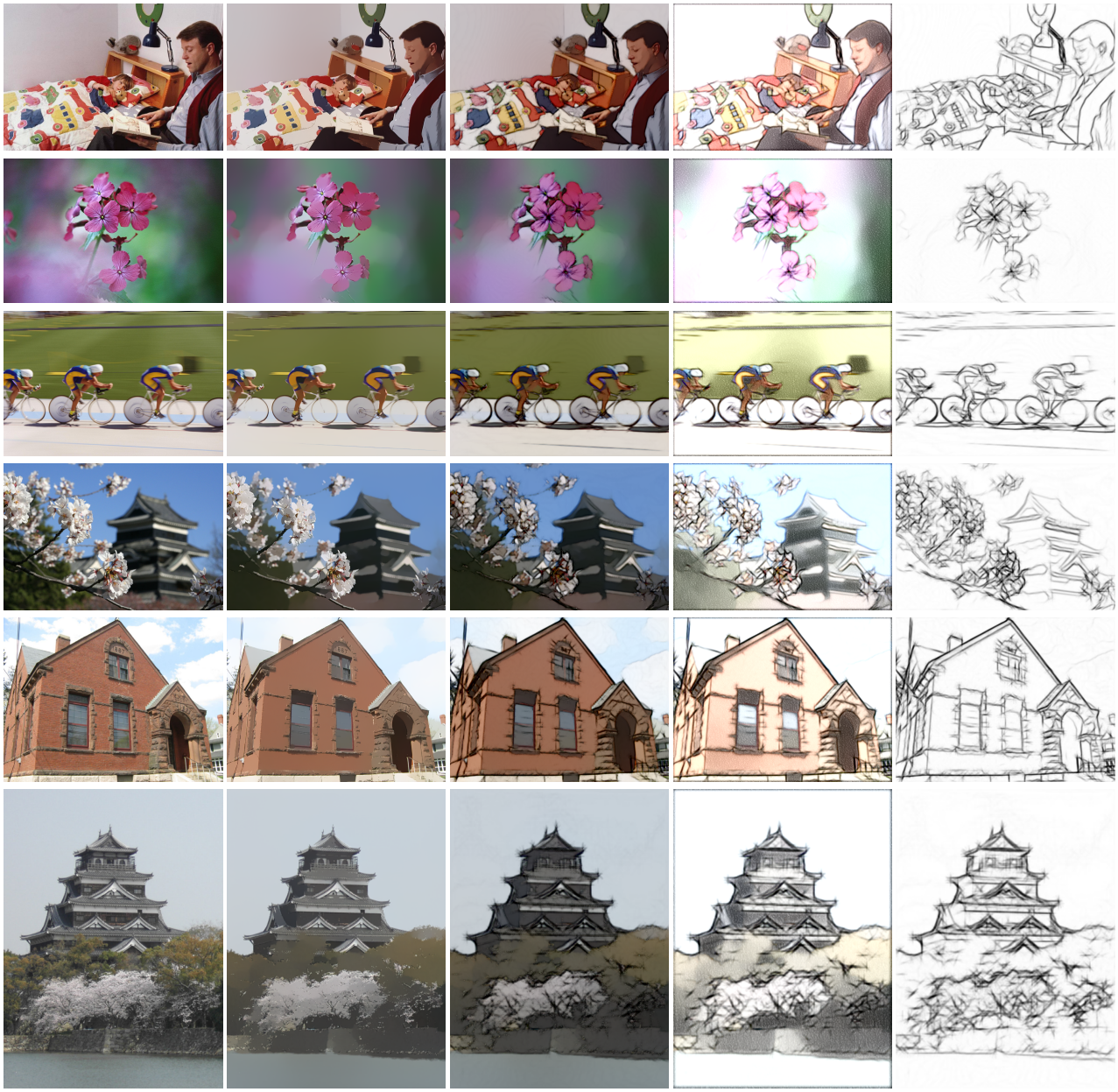


Figure 7: Image abstraction and pencil drawing results. Note all these results are generated without any parameter tweaking for this application.

Input | Smooth image | Abstraction (w/o Tone) | Abstraction | Pencil Drawing

Figure 8: Image abstraction and pencil drawing results. Note all these results are generated without any parameter tweaking for this application.

Figure 9: Image abstraction and pencil drawing results. Note all these results are generated without any parameter tweaking for this application.

Figure 10: Image abstraction and pencil drawing results. Note all these results are generated without any parameter tweaking for this application.

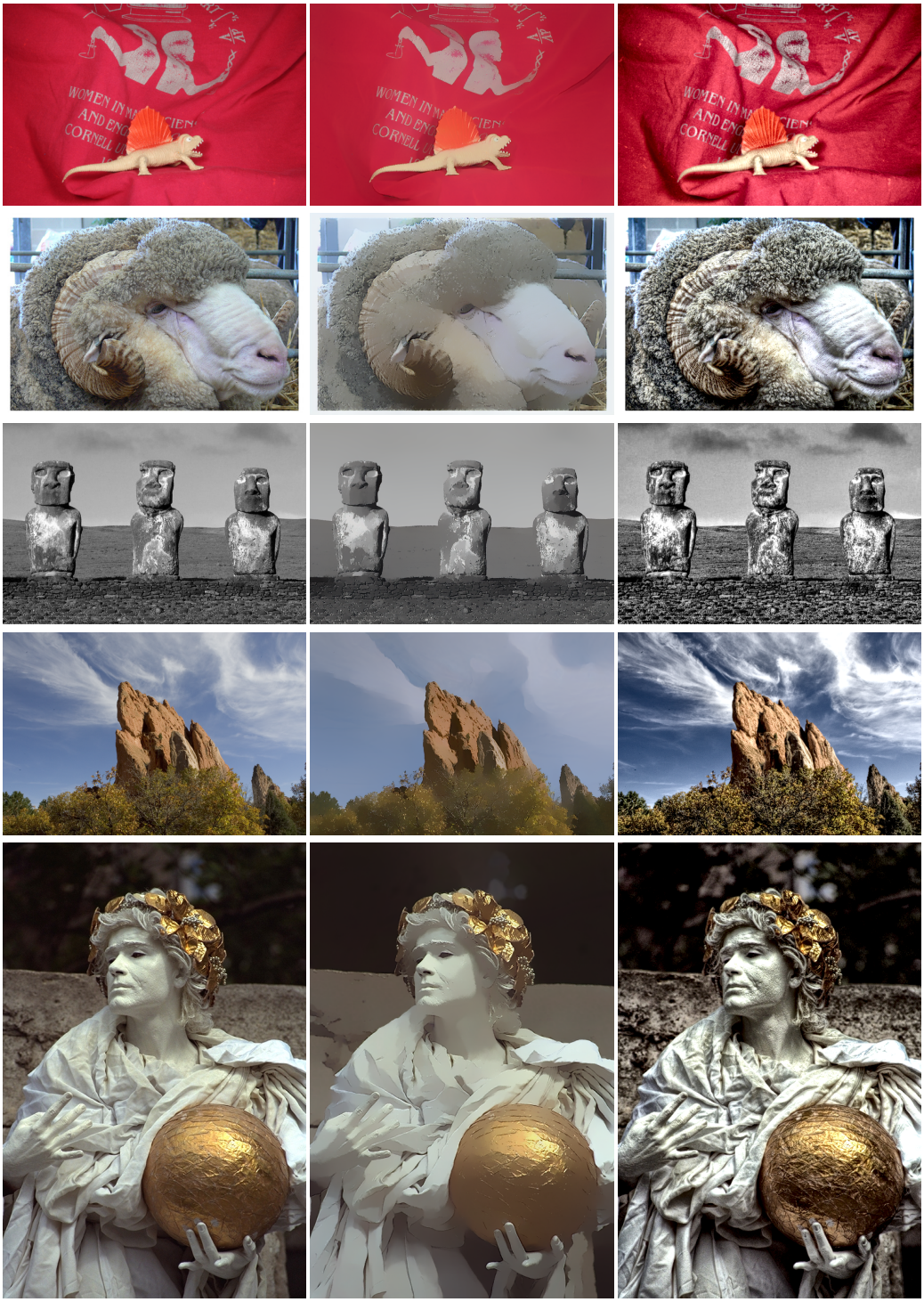


Figure 11: Detail enhancement results. Note all these results are generated without any parameter tweaking for this application.

Input Smooth image Detail Enhancement

Figure 12: Detail enhancement results. Note all these results are generated without any parameter tweaking for this application.

Input Smooth image Detail Enhancement

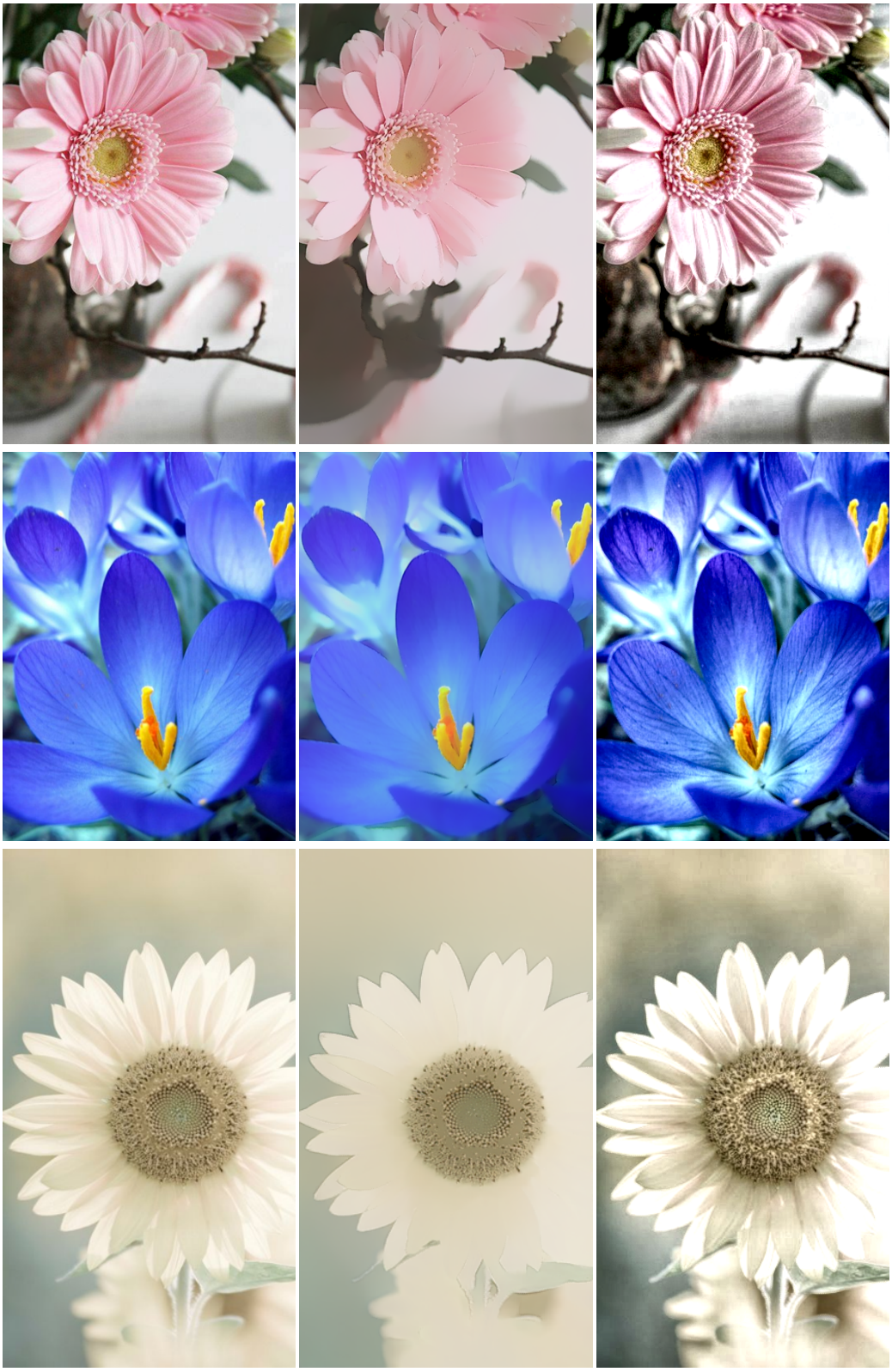
Figure 13: Detail enhancement results. Note all these results are generated without any parameter tweaking for this application.

Input Smooth image Detail Enhancement

Figure 14: Detail enhancement results. Note all these results are generated without any parameter tweaking for this application.

Input | Smooth image | Detail Enhancement

Figure 15: Detail enhancement results. Note all these results are generated without any parameter tweaking for this application.



Input Smooth image Detail Enhancement

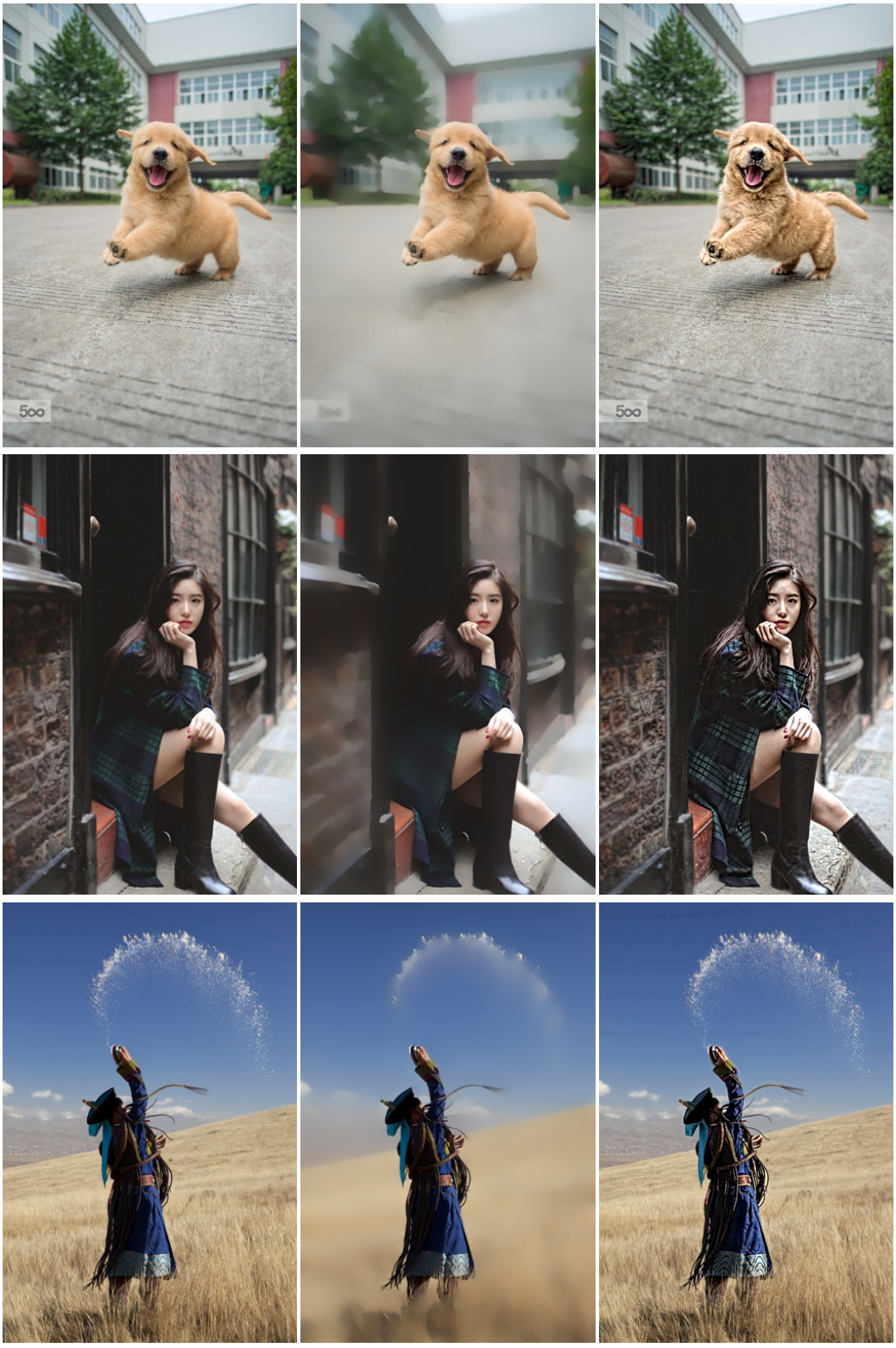
Figure 16: Detail enhancement results. Note all these results are generated without any parameter tweaking for this application.

Input | Smooth image | Detail Enhancement

Figure 17: Detail enhancement results. Note all these results are generated without any parameter tweaking for this application.

Figure 18: Texture removal results. Note all these results are generated without any parameter tweaking for this application.

Figure 19: Texture removal results. Note all these results are generated without any parameter tweaking for this application.

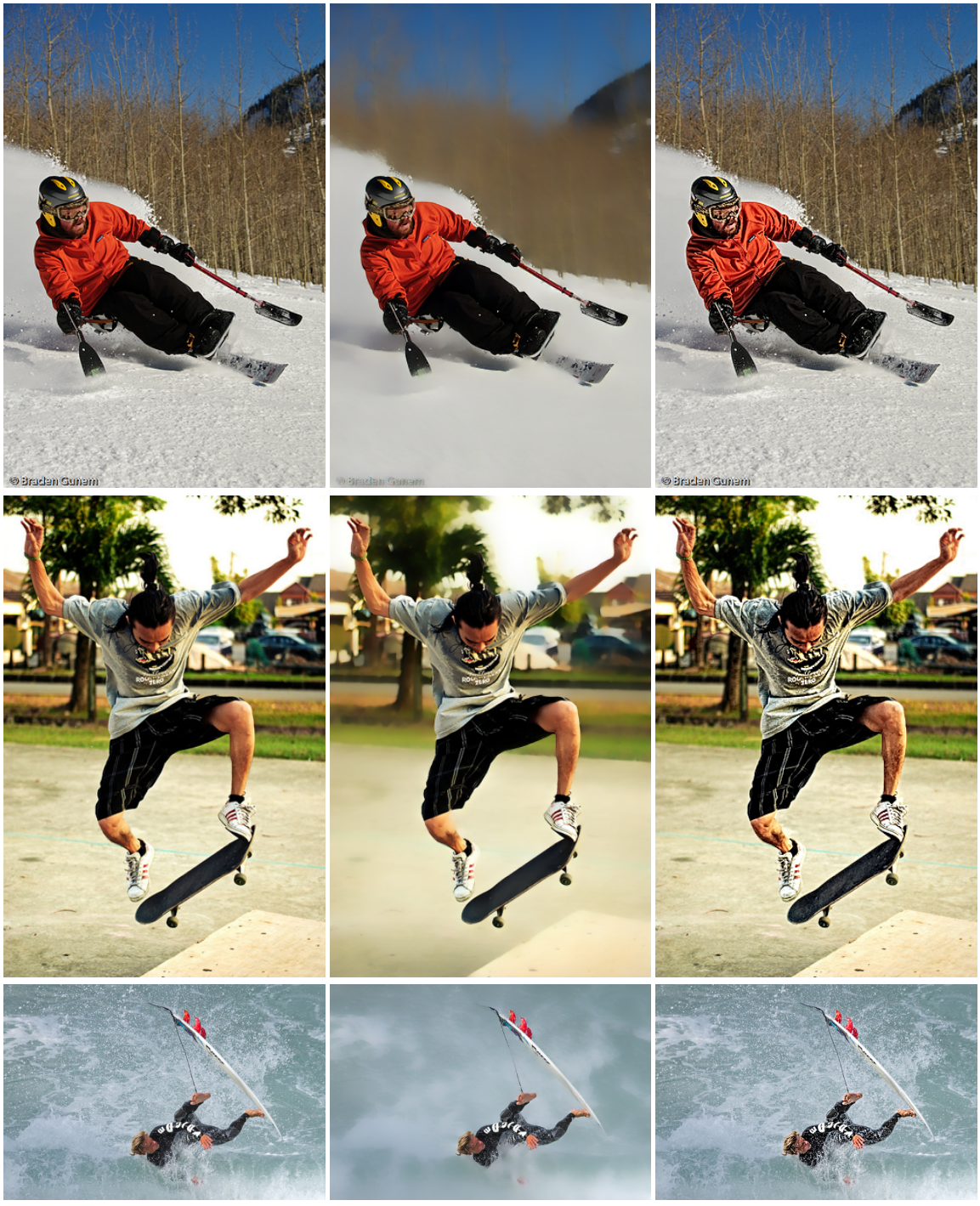


Input | Background smooth | Foreground Enhancement

Figure 20: Content-aware image manipulation results. Note all these results are generated without any parameter tweaking for this application.

Input | Background smooth | Foreground Enhancement

Figure 21: Content-aware image manipulation results. Note all these results are generated without any parameter tweaking for this application.



Input | Background smooth | Foreground Enhancement

Figure 22: Content-aware image manipulation results. Note all these results are generated without any parameter tweaking for this application.

Input Background smooth

Figure 23: Background smooth results. Note all these results are generated without any parameter tweaking for this application.

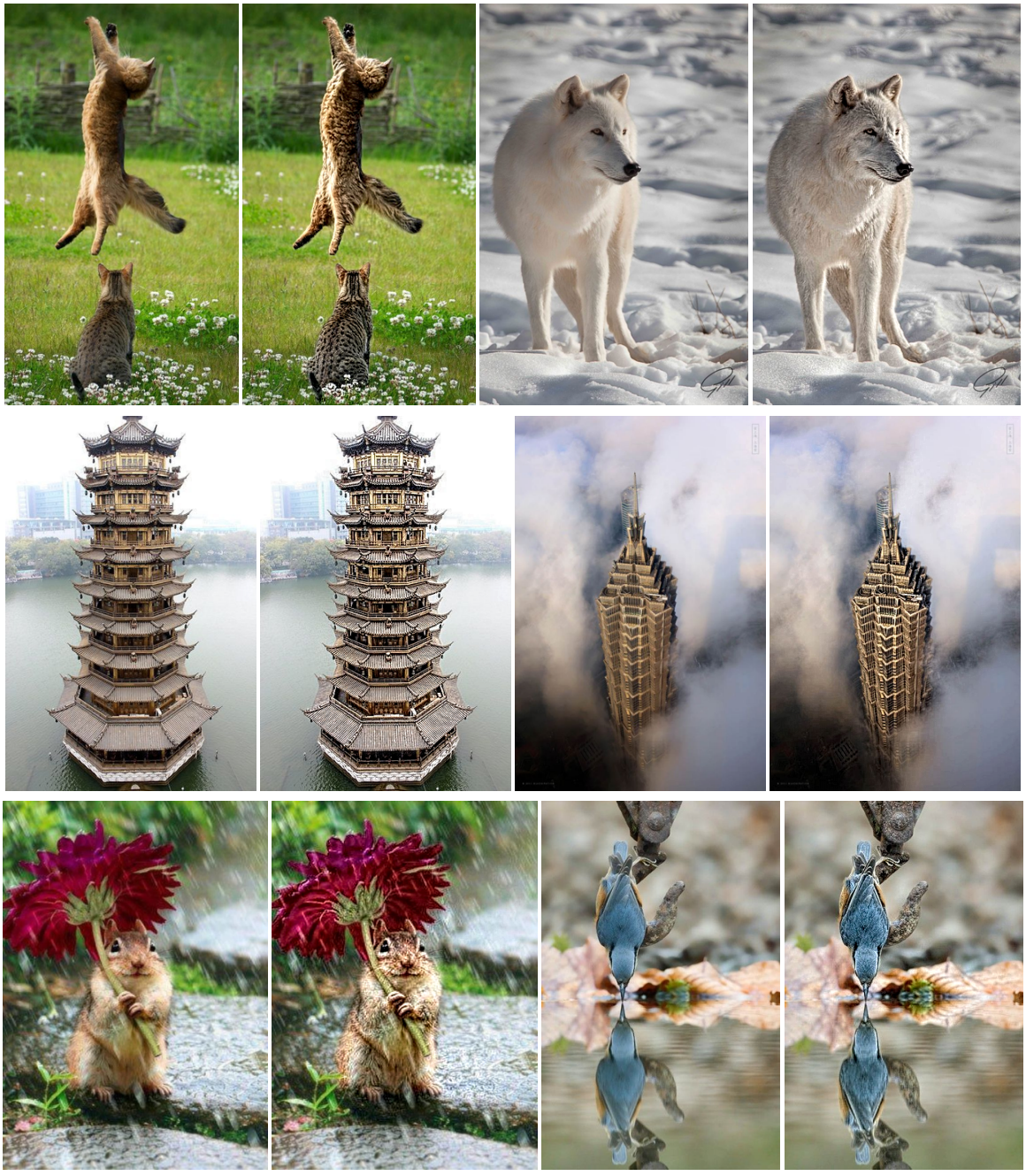


Figure 24: Foreground enhancement results. Note all these results are generated without any parameter tweaking for this application.

Figure 25: Foreground enhancement results. Note all these results are generated without any parameter tweaking for this application.