Is ℓ_2 a Good Loss Function for Neural Networks for Image Processing?

Hang Zhao[1,2] Orazio Gallo[1] Iuri Frosio[1] Jan Kautz[1]
[1]NVIDIA Corp. [2]MIT Media Lab

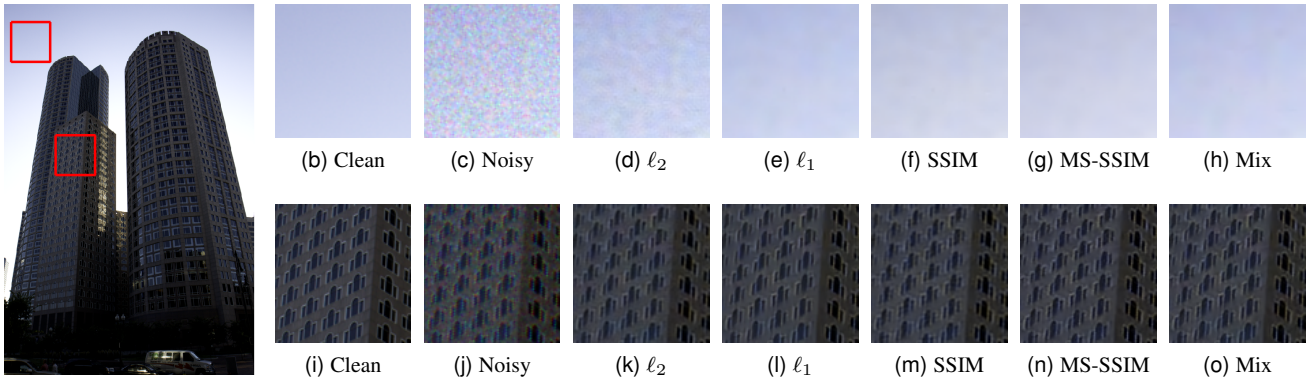


Figure 1: Joint demosaicking and denoising performed by a convolutional neural network, trained on different loss functions (best viewed in the electronic version by zooming in). The noisy patches are obtained by bilinear interpolation of the bayer data. ℓ_2, the default loss function for neural networks for image processing, produces splotchy artifacts in flat regions (d). MS-SSIM produces good results on both flat (g) and textured regions (n), but the color of the sky is more dull than the clean patch ((g) vs (b)). A combination of MS-SSIM and ℓ_1, here marked as "Mix," preserves structure, removes noise, and avoids biases in uniform regions.

Abstract

Neural networks are becoming central in several areas of computer vision and image processing. Different architectures have been proposed to solve specific problems. The impact of the loss layer of neural networks, however, has not received much attention by the research community: the default and most common choice is ℓ_2. This can be particularly limiting in the context of image processing, since ℓ_2 correlates poorly with perceived image quality.

In this paper we bring attention to alternative choices. We study the performance of several losses, including perceptually-motivated losses, and propose a novel, differentiable error function. We show that the quality of the results improves significantly with better loss functions, even for the same network architecture.

1. Introduction

For decades, neural networks have shown various degrees of success in several fields, ranging from robotics, to regression analysis, to pattern recognition. Despite the promising results they already produced in the 1980s on handwritten digit recognition [12], the popularity of neural networks in the field of computer vision has grown exponentially only recently, when deep learning boosted their performance in image recognition [11].

In the span of just a couple of years, neural networks have been employed for virtually any computer vision and image processing task known to the research community. Much research has focused on the definition of new architectures that are better suited to a specific problem [2, 22]. A large effort was also made to understand the inner mechanisms of neural networks, and what their intrinsic limitations are, for instance through the development of deconvolutional networks [24], or trying to fool networks with specific inputs [13]. Other advances were made on the techniques to improve the network's convergence [9].

The loss layer, despite being the effective driver of the network's learning, has attracted little attention from the research community: the choice of the cost function generally defaults to the squared ℓ_2 norm of the error. This is understandable, given the many desirable properties this norm possesses (Section 2.2). There is also a less well-funded, but just as relevant reason for the continued popularity of ℓ_2: standard neural networks packages, such as Caffe [10], only offer the implementation for this metric.

However, when the task at hand involves image quality, it is well known that ℓ_2 correlates poorly with image qual-

ity as perceived by a human observer [27]. This is because a number of assumptions implicitly made when using ℓ_2 are not satisfied. Arguably, the most important one is that ℓ_2 treats noise independently of the local characteristics of the image; on the contrary, the sensitivity of the Human Visual System (HVS) to noise depends on the local luminance, contrast and structure [19].

We focus on the use of neural networks for image processing tasks, and we study the effect of different metrics for the network's loss layer. Specifically, we compare ℓ_2 against four error metrics on representative image processing tasks: image super-resolution and joint denoising and demosaicking. First, we test whether a different local metric such as ℓ_1 can produce better results. We then evaluate the impact of perceptually-motivated metrics. We use two state-of-the-art metrics for image quality: the structural similarity index (SSIM [19]), and the multi-scale structural similarity index (MS-SSIM [21]). We choose these among the plethora of existing indexes, because they are established measures, and because they are differentiable—a requirement for the back-propagation stage. As expected, on the use cases we consider, the perceptual metrics outperform ℓ_2. However, and perhaps surprisingly, this is also true for ℓ_1, see Figure 1. Inspired by this observation, we propose a novel loss function and show its superior performance in terms of all the metrics we consider.

We offer several contributions. First we bring attention to the importance of the error metric used to train neural networks for image processing. We propose the use of three alternative error metrics (ℓ_1, SSIM, and MS-SSIM), and define a new one that combines the advantages of ℓ_1 and MS-SSIM. We analyze their performance and compare it with the commonly used ℓ_2. For each of the metrics we analyze, we implement a loss layer for Caffe, which we will make available to the research community.

2. Related work

In this paper, we target neural networks for image processing using the problems of super-resolution and joint demosaicking and denoising as test benches. Specifically, we show how established error measures can be adapted to work within the loss layer of a neural network, and how this can positively influence the results. Here we briefly review the existing literature on the subject of neural networks for images processing, and on the subject of measures of image quality.

2.1. Neural networks for image processing

Following the success of deep neural networks in several computer vision tasks [7, 11], neural networks have also received considerable attention in the context of image processing. Neural networks have been used for denoising [2], deblurring [22], demosaicking [17], and super-resolution [5] among others. To the best of our knowledge, however, the work on this subject has focused on tuning the architecture of the network for the specific application; the loss layer, which effectively drives the learning of the network to produce the desired output quality, is based on ℓ_2 for all of the approaches above.

We show that a better choice for the error measure has a strong impact on the quality of the results.

2.2. Evaluating image quality

The mean squared error, ℓ_2, is arguably the dominant error measure across very diverse fields, from regression problems, to pattern recognition, to signal and image processing. Among the main reasons for its popularity is the fact that it is convex and differentiable—very convenient properties for optimization problems. Other interesting properties range from the fact that ℓ_2 is preserved under orthogonal linear transformations, like the Fourier transform, and that it provides the maximum likelihood estimate in case of Gaussian, independent noise, to the fact that it is additive for independent noise sources. There is an even longer list of reasons for which we refer the reader to the work of Wang and Bovik [18].

These properties paved the way for ℓ_2's widespread adoption, which was further fueled by the fact that standard software packages tend to include tools to use ℓ_2, but not many other error functions. In the context of image processing, Caffe [10] actually offers *only* ℓ_2 as a loss layer[1], thus discouraging researchers to investigate the impact of other error measures.

However, it is widely accepted that ℓ_2, and consequently the Peak Signal-to-Noise Ratio, PSNR, do not correlate well with human's perception of image quality [27]: ℓ_2 simply does not capture the intricate characteristics of the human visual system (HVS). Thus, an image processing algorithm that optimizes ℓ_2, or the PSNR, does not generally maximize the quality of an image as perceived by a human observer.

There exists a rich literature of error measures, both reference-based and non reference-based, that attempt to address the limitations of the simple ℓ_2 error function. For our purposes, we focus on reference-based measures. A popular reference-based index is the structural similarity SSIM index [19]. SSIM evaluates images accounting for the fact that the HVS is sensitive to changes in local structure. Wang *et al*. [21] extend SSIM observing that the scale at which local structure should be analyzed is a function of factors such as image-to-observer distance. To account for these factors, they propose MS-SSIM, a multi-scale version of SSIM that weighs SSIM computed at different scales according to the sensitivity of the HVS. Exper-

[1] Caffe indeed offers other types of loss layers, but they are only useful for classification tasks.

imental results have shown the superiority of SSIM-based indexes over ℓ_2. As a consequence, SSIM has been widely employed as a metric to evaluate image processing algorithms. Moreover, given that it can be used as a differentiable cost function, SSIM has also been used in iterative algorithms designed for image compression [18], image reconstruction [1], denoising and super-resolution [15], and even downscaling [14]. To the best of our knowledge, however, SSIM-based indexes have never been adopted to drive the training of a neural network.

Recently, novel image quality indexes based on the properties of the HVS showed improved performance when compared to SSIM and MS-SSIM [27]. One of these is the Information Weigthed SSIM (IW-SSIM), a modification of MS-SSIM that also includes a weighting scheme proportional to the local image information [20]. Another is the Visual Information Fidelity (VIF), which is based on the amount of shared information between the reference and distorted image [16]. The Gradient Magnitude Similarity Deviation (GMSD) is characterized by simplified math and performance similar to that of SSIM [23]. Finally, the Feature Similarity Index (FSIM), leverages the perceptual importance of phase congruency, and measures the dissimilarity between two images based on local phase congruency and gradient magnitude [26]. FSIM has also been extended to FSIM$_c$, which can be used with color images. Despite the fact that they offer an improved accuracy in terms of image quality, the mathematical formulation of these indexes is generally more complex than SSIM and MS-SSIM, and possibly not differentiable, making their adoption for optimization procedures not immediate.

3. Loss layers for image processing

The loss layer of a neural network compares the output of the network with the ground truth. For image processing, these are generally the processed and reference patches. The process of minimizing the error on the training dataset allows to learn the network's parameters.

In our work, we investigate the impact of different loss function layers for image processing. Consider the case of a network that performs denoising and demosaicking jointly. The insets in Figure 1 show a zoom-in of different patches for the image in Figure 1(a) as processed by a network trained with different loss functions (see Section 4 for the network's description). A simple visual inspection is sufficient to appreciate the practical implications of the discussion on ℓ_2 (see Section 2).

Specifically, Figure 1(d) shows that in flat regions the network strongly attenuates the noise, but it produces visible splotchy artifacts. This is because ℓ_2 penalizes larger errors, but is more tolerant to small errors, regardless of the underlying structure in the image; the HVS, on the other hand, is extremely sensitive to luminance and color variations in texture-less regions. A few splotchy artifacts are still visible, though arguably less apparent, in textured regions, see Figure 1(k). The sharpness of edges, however, is well-preserved by ℓ_2, as blurring them would result in a large error. Note that these splotchy artifacts have been systematically observed before in the context of image processing with neural networks [2].

In this section we propose the use of different error functions. We provide a motivation for the different loss functions and we show how to compute their derivatives, which are necessary for the backpropagation step. We will also share our implementation of the different layers that can be readily used within Caffe.

Before diving in the specific loss layers, it is worth recalling that, given an error function $\mathcal{E}$, the loss for a patch P can be written as

$$L^{\mathcal{E}}(P) = \frac{1}{N} \sum_{p \in P} \mathcal{E}(p), \tag{1}$$

where N is the number of pixels p in the patch. The minimum satisfies the following system of p equations:

$$\nabla L^{\mathcal{E}}(P) = \vec{0} \Leftrightarrow \sum_{q \in P} \frac{\partial \mathcal{E}(p)}{\partial x(q)} = 0, \quad \forall p \in P. \tag{2}$$

To move towards the minimum we need to compute the derivatives in Equation 2.

3.1. The ℓ_1 error

As a first attempt to reduce the artifacts introduced by the ℓ_2 loss function, we want to train the exact same network using ℓ_1 instead of ℓ_2. The two losses weigh errors differently—ℓ_1 does not over-penalize larger errors—and, consequently, they may have different convergence properties.

Computing the ℓ_1 loss is straightforward:

$$L^{\ell_1}(P) = \frac{1}{N} \sum_{p \in P} |x(p) - y(p)|, \tag{3}$$

where p is the index of the pixel and P is the patch; $x(p)$ and $y(p)$ are the values of the pixels in the processed patch and the ground truth respectively. The derivatives for the backpropagation are also simple, since $\partial L^{\ell_1}(p)/\partial q = 0, \forall q \neq p$. Therefore, for each pixel p in the patch,

$$\partial L^{\ell_1}(P)/\partial p = \text{sign}\left(x(p) - y(p)\right) \tag{4}$$

Note that, although $L^{\ell_1}(P)$ is computed on the whole patch, the derivatives are back-propagated for each pixel in the patch. Somewhat unexpectedly, the network trained with ℓ_1 provides a significant improvement for several of the issues discussed above, see Figure 1(e) where the splotchy artifacts in the sky are removed.

3.2. SSIM

Although ℓ_1 shows improved performance over ℓ_2, if the goal is for the network to learn to produce visually pleasing images, it stands to reason that the error function should be perceptually motivated, as is the case with SSIM.

SSIM for pixel p is defined as

$$\begin{aligned} \text{SSIM}(p) &= \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \cdot \frac{2\sigma_{xy} + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} && (5) \\ &= l(p) \cdot cs(p) && (6) \end{aligned}$$

where we omitted the dependence of means and standard deviations on pixel p. Means and standard deviations are computed with a Gaussian filter with standard deviation σ_G:

$$\begin{aligned} \mu_x(p) &= G_{\sigma_G} * P_x && (7) \\ \sigma_x^2(p) &= G_{\sigma_G} * P_x^2 \ - \mu_x^2(p) && (8) \\ \sigma_{xy}(p) &= G_{\sigma_G} * (P_x \cdot P_y) \ - \mu_x(p)\mu_y(p), && (9) \end{aligned}$$

where P_x is a patch centered a pixel p, '$*$' denotes a convolution, and '$\cdot$' a point-wise multiplication. The values of $\mu_y(p)$ and $\sigma_y^2(p)$ can be computed similarly to Equations 7 and 8. The loss function for SSIM can be then written setting $\mathcal{E}\,(p) = 1 - \text{SSIM}\,(p)$:

$$L^{\text{SSIM}}(P) = \frac{1}{N} \sum_{p \in P} 1 - \text{SSIM}(p). \qquad (10)$$

Equations 5–9, highlight the fact that the computation of SSIM(p) requires looking at a neighborhood of pixel p as large as the support of G_{σ_G}. This means that $L^{\text{SSIM}}(P)$, as well as its derivatives, cannot be calculated in some boundary region of P. Note that this is not true for ℓ_1 or ℓ_2, which only need the value of the processed and reference patch at pixel p to be computed.

Luckily, however, the convolutional nature of the network allows us to write the loss as

$$L^{\text{SSIM}}(P) = 1 - \text{SSIM}(\tilde{p}), \qquad (11)$$

where $\tilde{p}$ is the center pixel of patch P. Again, this is because even though the network learns the weights maximizing SSIM for the central pixel, the learned kernel are then applied to all the pixels in the image. Note that the error can still be back-propagated to all the pixels within the support of G_{σ_G}, as they contribute to the computation of Equation 11.

Computing the derivatives is fairly straightforward; we report only the final results here and refer the reader to the additional material for the full derivation. Recall that we have to compute the derivatives at $\tilde{p}$ with respect to any other pixel q in patch P. We need to compute:

$$\begin{aligned} \frac{\partial L^{\text{SSIM}}(P)}{\partial x(q)} &= -\frac{\partial}{\partial x(q)} \text{SSIM}(\tilde{p}) \\ &= -\left(\frac{\partial l(\tilde{p})}{\partial x(q)} \cdot cs(\tilde{p}) + l(\tilde{p}) \cdot \frac{\partial cs(\tilde{p})}{\partial x(q)} \right), \qquad (12) \end{aligned}$$

where $l(\tilde{p})$ and $cs(\tilde{p})$ are the first and second term of SSIM (Equation 6) and their derivatives are

$$\frac{\partial l(\tilde{p})}{\partial x(q)} = 2 \cdot G_{\sigma_G}(q - \tilde{p}) \cdot \left(\frac{\mu_y - \mu_x \cdot l(\tilde{p})}{\mu_x^2 + \mu_y^2 + C_1} \right) \qquad (13)$$

and

$$\begin{aligned} \frac{\partial cs(\tilde{p})}{\partial x(q)} = \frac{2}{\sigma_x^2 + \sigma_y^2 + C_2} \cdot G_{\sigma_G}(q - \tilde{p}) \cdot \\ \cdot \left[\, (y(q) - \mu_y) - cs(\tilde{p}) \cdot (x(q) - \mu_x) \, \right], \qquad (14) \end{aligned}$$

where $G_{\sigma_G}(q - \tilde{p})$ is the Gaussian coefficient associated with pixel q.

3.3. MS-SSIM

The choice of σ_G has an impact on the quality of the processed results of a network that is trained with SSIM, as can be seen from the derivatives in the previous section. Specifically, for smaller values of σ_G the network loses the ability to preserve the local structure and the splotchy artifacts are reintroduced in flat regions, see Figure 2(e). For large values of σ_G, we observe that the network tends to preserve noise in the proximity of edges, Figure 2(c). See Section 5 for more details.

Rather than fine-tuning the σ_G, we propose to use the multi-scale version of SSIM, MS-SSIM. Given a dyadic pyramid of M levels, MS-SSIM is defined as

$$\text{MS-SSIM}(p) = l_M^{\alpha}(p) \cdot \prod_{j=1}^{M} cs_j^{\beta_j}(p) \qquad (15)$$

where l_j and cs_j are the terms we defined in Section 3.2 at scale j, and we set $\alpha = \beta_j = 1$, for $j = \{1, ..., M\}$ for convenience. Similarly to Equation 11, we can approximate the loss for patch P with the loss computed at its center pixel $\tilde{p}$:

$$L^{\text{MS-SSIM}}(P) = 1 - \text{MS-SSIM}(\tilde{p}). \qquad (16)$$

The considerations about computing the error only at $\tilde{p}$ are the same as the ones discussed in Section 3.2. Because we set all the exponents of Equation 15 to one, the derivatives

of the loss function based on MS-SSIM can be written as

$$\frac{\partial L^{\text{MS-SSIM}}(P)}{\partial x(q)} = \left(\frac{\partial l_M(\tilde{p})}{\partial x(q)} + l_M(\tilde{p}) \cdot \sum_{i=0}^{M} \frac{1}{cs_i(\tilde{p})} \frac{\partial cs_i(\tilde{p})}{\partial x(q)} \right) \cdot \prod_{j=1}^{M} cs_j(\tilde{p}), \tag{17}$$

where the derivatives of l_j and cs_j are the same as in Section 3.2. For the full derivation we refer the reader to the supplementary material.

Using $L^{\text{MS-SSIM}}$ to train the network, Equation 15 requires that we compute a pyramid of M levels of patch P, which is a computationally expensive operation given that it needs to be performed at each iteration. To avoid this, we approximate and replace the construction of the pyramid: instead of computing M levels of the pyramid, we propose to use M different values for σ_G, each one being half of the previous, on the full-resolution patch. Specifically, we use $\sigma_G^i = \{0.5, 1, 2, 4, 8\}$ and define $cs_i \triangleq G_{\sigma_G^i} \cdot cs_0(\tilde{p})$ and $\partial cs_i(\tilde{p})/\partial x(q) \triangleq G_{\sigma_G^i} \cdot \partial cs_0(\tilde{p})/\partial x(q)$, where the Gaussian filters are centered at pixel $\tilde{p}$, and "$\cdot$" is a point-wise multiplication. The terms depending on l_M can be defined in a similar way.

3.4. The best of both worlds: MS-SSIM + ℓ_1

By design, both MS-SSIM and SSIM are not particularly sensitive to uniform biases, in particular in regions where at least one of the color channels is strong (see Section 5). This can cause changes of brightness or shifts of colors, which typically become more dull. However, MS-SSIM preserves the contrast in high-frequency regions better than the other loss functions we experimented with. On the other hand, ℓ_1 preserves colors and luminance—an error is weighed equally regardless of the local structure—but does not produce quite the same contrast as MS-SSIM.

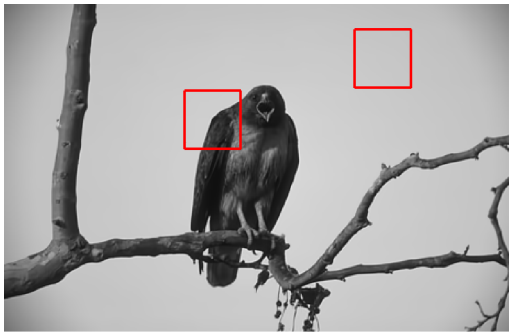
We propose to combine the characteristics of both error functions:

$$L^{\text{Mix}} = \alpha L^{\text{MS-SSIM}} + (1 - \alpha) \cdot G_{\sigma_G^M} \cdot L^{\ell_1}, \tag{18}$$

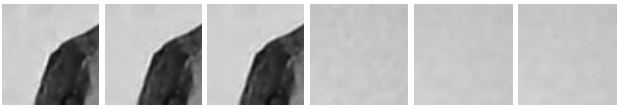
where we omitted the dependence on patch P for all loss functions, and we experimentally set $\alpha = 0.84$ in our experiments. The derivatives of Equation 18 are simply the weighed sum of the derivatives of its two terms, which we compute in the previous sections. Note that we add a point-wise multiplication between $G_{\sigma_G^M}$ and L^{ℓ_1}: this is because MS-SSIM propagates the error at pixel q based on its contribution to MS-SSIM of the central pixel $\tilde{p}$, as determined by the Gaussian weights, see Equations 13 and 14.

4. Results

For our analysis of the different loss functions we focus on joint demosaicking and denoising, a fundamental problem in image processing [8]. We define a fully convolutional neural network (CNN) that takes a 31×31×3 input. The first layer is a 64×9×9×3 convolutional layer, where the first term indicates the number of filters and the remaining terms indicate their dimensions. The second convolutional layer is 64×5×5×64, and the output layer is a 3×5×5×64 convolutional layer. We apply parametric rectified linear unit (PReLU) layers to the output of the inner convolutional layers, because of their superior performance compared to ReLU [7]. Similarly to the work of Dong *et al.* [5], the input to our network is an RGB image patch obtained by bilinearly up-sampling a 31×31 Bayer patch; in this sense the network is really doing joint denoising and super-resolution. We trained the network considering dif-



(a) Clean image

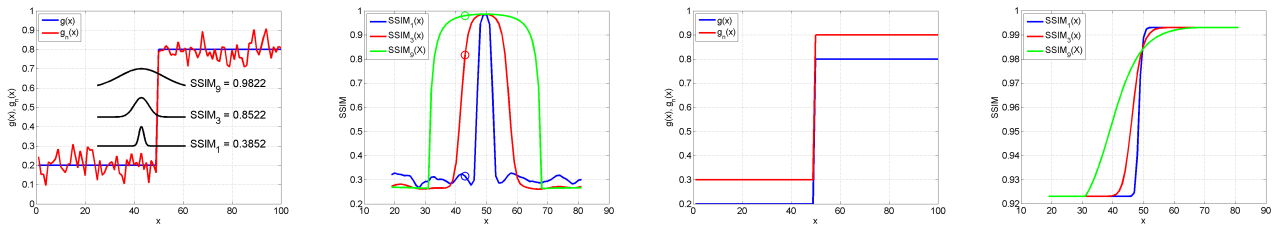


(b) SSIM_1 (c) SSIM_3 (d) SSIM_9 (e) SSIM_1 (f) SSIM_3 (g) SSIM_9

Figure 2: Comparison of the results of networks trained with SSIM with different sigmas (SSIM_k means $\sigma_G = k$). Insets (b)–(d), show an increasingly large halo of noise around the edge: smaller values of σ help at edges. However, in mostly flat regions, larger values of σ help reducing the splotchy artifacts (e)–(g). Best viewed by zooming in on the electronic copy.

(a) LR (b) ℓ_2 (c) ℓ_1 (d) Mix

Figure 3: Results for super-resolution. Notice the grating artifacts on the black stripes of the wing and around the face of the girl produced by ℓ_2.



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

Figure 4: Panel (a) shows a scanline of a 100×100 image of a step function, noise-free and corrupted by zero-mean Gaussian noise with $\sigma = 0.05$. The same panel shows the values of SSIM_{σ_G}, for a 37×37 patch centered in $x = 43$ and $\sigma_G = \{1, 3, 9\}$; the three corresponding weighting windows adopted by SSIM_{σ_G} are depicted in black. Panel (b) shows the values of SSIM_{σ_G}, for a 37×37 patch shifting along the x direction and $\sigma_G = \{1, 3, 9\}$. Panels (c) and (d) show the case of additive, uniform noise.

ferent cost functions (ℓ_1, ℓ_2, SSIM_5, SSSIM_9, MS-SSIM and MS-SSIM+ℓ_1)[2] on a training set of 700 RGB images taken from the MIT-Adobe FiveK Dataset [3]. To simulate a realistic image acquisition process, we corrupted each image with a mix of photon shot and zero-mean Gaussian noise, and introduced clipping due to the sensor zero level and saturation. We used the model proposed by Foi *et al.* for this task [6]. The average PSNR for the testing images after adding noise was 28.24dB, as reported in Table 1. Figure 1(c) shows a typical patch corrupted by noise. We used 40 images from the same dataset for testing (the network did not see this subset during training).

Beyond considering different cost functions for training, we also compare the output of our network with the images obtained by a state-of-the-art denoising method, BM3D, operating directly in the Bayer domain [4], followed by a standard demosaicking algorithm [25]. Since BM3D is designed to deal with Gaussian noise, rather than the more realistic noise model we use, we apply a Variance Stabilizing Transform [6] to the image data in Bayer domain before applying BM3D, and its inverse after denoising.

Figure 1 and 5 show several results and comparisons between the different networks. Note the splotchy artifacts for the ℓ_2 network on flat regions, and the noise around the edges for the SSIM_5 and SSIM_9 networks. The network trained with MS-SSIM addresses these problems but tends to render the colors more dull, see Section 5. The network trained with MS-SSIM+ℓ_1 generates the best results (see Figures 1 and 5).

We also perform a quantitative analysis of the results. We evaluate several image quality metrics on the output of the CNNs trained with the different cost functions and with BM3D [4]. The image quality indexes, range from the traditional ℓ_2 metric and PSNR, to the most refined, perceptually inspired metrics, like FSIM [26]. The average values of these metrics on the testing dataset are reported in Table 1. When the network is trained using ℓ_1 as a cost function, instead of the traditional ℓ_2, the average quality of the output images is superior for all the quality metrics considered. It is quite remarkable to notice that, even when the ℓ_2 or PSNR metrics are used to evaluate image quality, the network trained with the ℓ_1 loss outperforms the one trained with the ℓ_2 loss. We offer an explanation of this in Section 5. On the other hand, we note that the network trained with SSIM performs either at par or slightly worse than the one trained with ℓ_1, both on traditional metrics and on perceptually-inspired losses. The network trained on MS-SSIM performs better than the one based on SSIM, but still does not clearly outperforms ℓ_1. This is due to the color shifts mentioned above and discussed in Section 5. However, the network that combines MS-SSIM and ℓ_1 achieves the best results on all of the image quality metrics we consider.

We also verify the outcome of our analysis on the network for super-resolution proposed by Dong *et al.* [5]. We make a few minor but important changes to their approach. First we use PReLU, instead of ReLU, layers. Second we use bilinear instead of bicubic interpolation for initialization. The latter introduces high-frequency artifacts that hurt the learning process. Finally we train directly on the RGB data. We made these changes for all the loss functions we test, including ℓ_2. Figure 3 shows some sample results. An analysis of Table 3 brings similar considerations as for the case of joint denoising and demosaicking.

5. Discussion

In our experiments, we observe that a CNN trained with ℓ_1 provides better results than one trained with ℓ_2 on the image quality metric we measured, see Tables 1 and 3. However, one would expect that a CNN trained with ℓ_2 should achieve better results in terms of ℓ_2 error over one trained with a different error function. Table 2 reports the value of different cost functions measured on the *training* set, for the different networks, after convergence. Note that the network trained with ℓ_1 achieves a better ℓ_2 score than the network trained with ℓ_2 itself. We speculate that this unexpected result may be related to the smoothness and the

[2] SSIM_k means SSIM computed with $\sigma_G = k$.

(a) Clean (b) Noisy (c) BM3D (d) ℓ_2 (e) ℓ_1 (f) SSIM (g) MS-SSIM (h) MS-SSIM+ℓ_1

Figure 5: Results for denosing+demosaicking for different approaches. The noisy patches are obtained by simple bilinear interpolation. Note the splotchy artifacts ℓ_2 produces in flat regions. Also note the change in colors for SSIM-based losses. The proposed metric, MS-SSIM+ℓ_1, addresses the former issues.

			Training cost function					
Image quality metric	Noisy	$BM3D$	ℓ_2	ℓ_1	SSIM_5	SSIM_9	MS-SSIM	Mix
$1000 \cdot \ell_2$	1.65	0.45	0.56	0.43	0.58	0.61	0.55	**0.41**
PSNR	28.24	34.05	33.18	34.42	33.15	32.98	33.29	**34.61**
$1000 \cdot \ell_1$	27.36	14.14	15.90	13.47	15.90	16.33	15.99	**13.19**
SSIM	0.8075	0.9479	0.9346	0.9535	0.9500	0.9495	0.9536	**0.9564**
MS-SSIM	0.8965	0.9719	0.9636	0.9745	0.9721	0.9718	0.9741	**0.9757**
IW-SSIM	0.8673	0.9597	0.9473	0.9619	0.9587	0.9582	0.9617	**0.9636**
GMSD	0.1229	0.0441	0.0490	0.0434	0.0452	0.0467	0.0437	**0.0401**
FSIM	0.9439	0.9744	0.9716	0.9775	0.9764	0.9759	0.9782	**0.9795**
FSIM_c	0.9381	0.9737	0.9706	0.9767	0.9752	0.9746	0.9769	**0.9788**

Table 1: Average value of different image quality metrics for the dataset considered here, and networks trained on the demosaicking + denoising problem and different cost functions. For SSIM, MS-SSIM, IW-SSIM, GMSD and FSIM the value reported here has been obtained as an average of the three color channels. Best results are shown in bold.

	Training cost function					
Cost function @ convergence	ℓ_2	ℓ_1	SSIM_5	SSIM_9	MS-SSIM	Mix
$1000{\cdot}\ell_2$	0.1888	0.1457	0.1990	0.2056	0.1856	**0.1399**
$1000{\cdot}\ell_1$	10.6578	9.1549	10.7562	10.9749	10.6671	**8.9309**
1 - SSIM_5	0.0893	0.0636	0.0688	0.0688	0.0622	**0.0600**
1 - SSIM_9	0.0872	0.0618	0.0669	0.0670	0.0605	**0.0584**
1 - MS-SSIM	0.3571	0.2684	0.2921	0.2917	0.2640	**0.2582**
Mix	0.0787	0.0648	0.0743	0.0754	0.0718	**0.0629**

Table 2: Cost function values at convergence, for different networks with the same architecture, trained using different cost functions on the joint demosaicking and denoising problem. The minimum value of each cost function at convergence is reported in bold.

	Bilinear	ℓ_2	ℓ_1	MS-SSIM	Mix
$1000 \cdot \ell_2$	2.5697	1.2407	1.1062	1.3223	**1.0990**
PSNR	27.1634	30.6588	31.2629	30.1128	**31.3426**
$1000 \cdot \ell_1$	28.7764	20.4730	19.0643	22.3968	**18.8983**
SSIM	0.8632	0.9274	0.9322	0.9290	**0.9334**
MS-SSIM	0.9603	0.9816	0.9826	0.9817	**0.9829**
IW-SSIM	0.9532	0.9868	0.9879	0.9866	**0.9881**
GMSD	0.0714	0.0298	0.0259	0.0316	**0.0255**
FSIM	0.9070	0.9600	0.9671	0.9601	**0.9680**
FSIM_c	0.9064	0.9596	0.9667	0.9597	**0.9677**

Table 3: Average value of different image quality metrics for the dataset considered here, and networks trained on the super-resolution problem and different cost functions. For SSIM, MS-SSIM, IW-SSIM, GMSD and FSIM the value reported here has been obtained as an average of the three color channels. Best results are shown in bold.

local convexity properties of each measure: ℓ_2 may have many more local minima that prevent convergence towards a better local minimum. ℓ_1 may be smoother and thus more likely to get to a better local minimum, for both ℓ_1 and ℓ_2—the "good" minima of the two should be related, after all. A similar consideration can be made about the mix of MS-SSIM and ℓ_1: by looking for solutions that are perceptually plausible, this metric may be removing some of the local minima that exist in the space of all the other loss functions, thus allowing it to converge to a better local minimum.

Table 1 also reveals that SSIM and MS-SSIM do not perform as well as ℓ_1. To investigate the phenomenon we trained several SSIM networks with different σ_G's and found that smaller values of σ_G produce better results at edges, but worse results in flat regions, while the opposite is true for larger values, see Figure 2. This can be understood by looking at Figure 4(a) and 4(b): close to an edge, a larger σ_G is more tolerant to the same amount of noise because it detects the presence of an edge, which is known to have a masking effect for the HVS. Thanks to its multi-scale nature, MS-SSIM solves this issue; however, much like SSIM, it is not particularly sensitive to a uniform bias on a flat region, in particular for bright regions. This is shown for SSIM in figure 4(c) and 4(d), and is due to the fact that the l term in SSIM measures the error in terms of a contrast, thus effectively reducing the importance of an error when the background is bright.

6. Conclusions

In this paper we focus on an aspect of neural networks for image processing that is usually underestimated: the loss layer. We propose several alternatives to ℓ_2, which is the *de facto* standard, and we also define a novel loss. We use the problems of super-resolution and joint denoising and demosaicking for our tests; the network trained with the proposed loss outperforms other networks in terms of both traditional and perceptually-motivated metrics. Because the networks we use are fully convolutional, they are extremely efficient, as they do not require an aggregation step. Nevertheless, thanks to the loss we propose, our joint denoising and demosaicking network outperforms CFA-BM3D, a variant of BM3D tuned for denoising in Bayer domain, which is the state-of-the-art denoising algorithm. We will make the implementation of the layers described in this paper available to the research community.

References

[1] D. Brunet, E. R. Vrscay, and Z. Wang. Structural similarity-based approximation of signals and images using orthogonal bases. In *International Conference on Image Analysis and Recognition*, pages 11–22, 2010. 3

[2] H. Burger, C. Schuler, and S. Harmeling. Image denoising: Can plain neural networks compete with BM3D? In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2392–2399, June 2012. 1, 2, 3

[3] V. Bychkovsky, S. Paris, E. Chan, and F. Durand. Learning photographic global tonal adjustment with a database of input / output image pairs. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2011. 6

[4] A. Danielyan, M. Vehvilainen, A. Foi, V. Katkovnik, and K. Egiazarian. Cross-color BM3D filtering of noisy raw data. In *Intern. Workshop on Local and Non-Local Approximation in Image Processing*, pages 125–129, 2009. 6

[5] C. Dong, C. Loy, K. He, and X. Tang. Learning a deep convolutional network for image super-resolution. In *European Conference on Computer Vision*, pages 184–199, 2014. 2, 5, 6

[6] A. Foi. Clipped noisy images: Heteroskedastic modeling and practical denoising. *Signal Processing*, 89(12):2609–2629, 2009. 6

[7] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, arXiv:1502.01852, 2015. 2, 5

[8] F. Heide, M. Steinberger, Y.-T. Tsai, M. Rouf, D. Pajak, D. Reddy, O. Gallo, J. Liu, W. Heidrich, K. Egiazarian, J. Kautz, and K. Pulli. FlexISP: A flexible camera image processing framework. *ACM Trans. Graph.*, 33(6):231:1–231:13, 2014. 5

[9] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, arXiv:1207.0580, 2012. 1

[10] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv:1408.5093*, 2014. 1, 2

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105, 2012. 1, 2

[12] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989. 1

[13] A. Nguyen, J. Yosinski, and J. Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015. 1

[14] A. C. Öztireli and M. Gross. Perceptually based downscaling of images. *ACM Trans. Graph.*, 34(4):77:1–77:10, 2015. 3

[15] A. Rehman, M. Rostami, Z. Wang, D. Brunet, and E. Vrscay. Ssim-inspired image restoration using sparse representation. *EURASIP Journal on Advances in Signal Processing*, 2012(1), 2012. 3

[16] H. Sheikh and A. Bovik. Image information and visual quality. *IEEE Transactions on Image Processing*, 15(2):430–444, 2006. 3

[17] Y.-Q. Wang. A multilayer neural network for image demosaicking. In *IEEE International Conference on Image Processing*, pages 1852–1856, 2014. 2

[18] Z. Wang and A. Bovik. Mean squared error: Love it or leave it? A new look at signal fidelity measures. *IEEE Signal Processing Magazine*, 26(1):98–117, 2009. 2, 3

[19] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004. 2

[20] Z. Wang and Q. Li. Information content weighting for perceptual image quality assessment. *IEEE Transactions on Image Processing*, 20(5):1185–1198, 2011. 3

[21] Z. Wang, E. P. Simoncelli, and A. C. Bovik. Multiscale structural similarity for image quality assessment. In *Asilomar Conference on Signals, Systems and Computers*, volume 2, pages 1398–1402, 2003. 2

[22] L. Xu, J. S. Ren, C. Liu, and J. Jia. Deep convolutional neural network for image deconvolution. In *NIPS*, pages 1790–1798, 2014. 1, 2

[23] W. Xue, L. Zhang, X. Mou, and A. Bovik. Gradient magnitude similarity deviation: A highly efficient perceptual image quality index. *IEEE Transactions on Image Processing*, 23(2):684–695, 2014. 3

[24] M. Zeiler, D. Krishnan, G. Taylor, and R. Fergus. Deconvolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2528–2535, 2010. 1

[25] D. Zhang and X. Wu. Color demosaicking via directional linear minimum mean square-error estimation. *IEEE Trans. on Image Processing*, 14(12):2167–2178, 2005. 6

[26] L. Zhang, D. Zhang, X. Mou, and D. Zhang. FSIM: A feature similarity index for image quality assessment. *IEEE Transactions on Image Processing*, 20(8):2378–2386, 2011. 3, 6

[27] L. Zhang, L. Zhang, X. Mou, and D. Zhang. A comprehensive evaluation of full reference image quality assessment algorithms. In *IEEE International Conference on Image Processin*, pages 1477–1480, 2012. 2, 3