Published in Image Processing On Line on 2019–01–06.
Submitted on 2018–08–02, accepted on 2018–12–10.
ISSN 2105–1232 © 2019 IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials, software, datasets and online demo at
https://doi.org/10.5201/ipol.2019.231

An Analysis and Implementation of the FFDNet Image Denoising Method

Matias Tassano[1,2], Julie Delon[1], Thomas Veit[2]

[1] MAP5, Université Paris Descartes, France
({matias.tassano, julie.delon}@parisdescartes.fr)
[2] GoPro
(tveit@gopro.com)

Communicated by Gabriele Facciolo *Demo edited by* Gabriele Facciolo

Abstract

FFDNet is a recent image denoising method based on a convolutional neural network architecture. In contrast to other existing neural network denoisers, FFDNet exhibits several desirable properties such as faster execution time and smaller memory footprint, and the ability to handle a wide range of noise levels with a single network model. The combination between its denoising performance and lower computational load makes this algorithm attractive for practical denoising applications. In this paper we propose an open-source implementation of the method based on PyTorch, a popular machine learning library for Python. Code for the training of the network is also provided. We also discuss the characteristics of the architecture of this algorithm and we compare it to other similar methods.

Source Code

The Python implementation of the FFDNet image denoising algorithm has been peer-reviewed and accepted by IPOL. The source code, its documentation, and the online demo are available from the web page of this article[1]. Compilation and usage instructions are included in the `README.txt` file of the archive.

Keywords: denoising; residual learning; neural networks; CNN

[1]https://doi.org/10.5201/ipol.2019.231

1 Introduction

1.1 Related Work

Recently, new image denoising methods based on deep learning techniques have drawn considerable attention due to their outstanding performance. In particular, discriminative learning methods exhibit relative fast inference times and very good denoising performance. Schmidt and Roth proposed in [22] the cascade of shrinkage fields (CSF) method that unifies the random field-based model and half-quadratic optimization into a single learning framework. Based on this method, Chen and Pock proposed in [4] a trainable nonlinear reaction diffusion (TNRD) model. This model can be expressed as a feed-forward deep network by concatenating a fixed number of gradient descent inference steps. Methods such as these two attain denoising performances comparable to those of well-known algorithms such as BM3D. Both CSF and TNRD have shown promising results to bridge the gap between denoising quality and computational efficiency. However, their performance is restricted to specific forms of prior. On top of that, many hand-tuned parameters are involved in the training process. In [2], and later in [28], the multi-layer perceptron (MLP) was successfully applied for image denoising. Nevertheless, a significant drawback of all these algorithms is that a specific model must be trained for each noise level.

Another popular approach involves the use of convolutional neural networks (CNN), e.g. RBDN [21], DnCNN [33], and FFDNet [34]. Their performance compares favorably to other state-of-the-art algorithms, both quantitatively and visually. These methods are composed of a succession of convolutional layers with nonlinear activation functions in between them. This type of architecture has been applied to the problem of joint denoising and demosaicing of RGB and raw images by Gharbi et al. in [7], while [23, 3] approach the same problem for low-light conditions. Contrary to other deep learning denoising methods, one of the remarkable features that these CNN based methods present is the ability of denoising several levels of noise with only one trained model.

Proposed by Zhang et al. in [33], DnCNN is an end-to-end trainable deep CNN for image denoising. This method is able to denoise different noise levels (e.g. with standard deviation $\sigma \in [0, 55]$) with only one trained model. One of its main features is that it implements residual learning [8], i.e. it estimates the noise existent in the input image rather than the denoised image. In a following paper [34], Zhang et al. proposed FFDNet, which builds upon the work done for DnCNN. The main difference of FFDNet with respect to DnCNN is the inclusion of preprocessing and postprocessing layers before and after the same nonlinear mapping of DnCNN. The preprocessing layer reorganizes the pixels of the input image into a quarter-resolution multi-channel image. Most of the computations are performed at this lower scale, thus reducing the global complexity of the algorithm. Overall, FFDNet is about three times faster than DnCNN and more memory-friendly. Also, an extra channel containing a noise map is concatenated to the input at the preprocessing layer. For spatially invariant AWGN with noise level σ, the noise map is uniform with all its elements equal to σ. As for the role of the noise map as input parameter, it can be conceived as a control of the trade-off between noise reduction and detail preservation. Lastly, the postprocessing layer reshapes the output of the nonlinear mapping back into the original input resolution. These characteristics make FFDNet an appealing method, even for consumer applications, as it achieves an interesting balance between denoising performance and complexity.

1.2 Definition of the Problem

The noise model in digital camera raw data is composed of two mutually independent parts, a Poissonian signal-dependent component and a Gaussian signal-independent part [6]. The former is mainly due to the photon-counting process, while the latter accounts for the signal-independent errors such as electric and thermal noise. A question quickly arises: if we have that the main source of noise in images follows a Poisson distribution, why is the additive Gaussian noise model the one which is used in most publications about denoising? The use of this type of model is justified as one can always apply a variance-stabilizing transformation to transform a random variable with a Poisson distribution into one with an approximately standard Gaussian distribution. In this context, the removal of the Poisson noise can be achieved in three steps. First, the noise variance is stabilized by applying one of these transformations, producing a signal which can be considered homoscedastic. Secondly, a conventional denoising method is applied to the transformed signal. Finally, an inverse transformation is applied to the denoised signal. The most commonly used example of such transforms is the Anscombe transform [1]. As for the inverse variance-stabilizing transformation, care must be taken to choose an optimal form which minimizes bias. In [16], an optimal unbiased inverse of the Anscombe transformation is introduced, while [15] proposes a closed-form approximation of this inverse. The authors show that the use of this exact unbiased inverse within the three-step process described above yields results which are competitive with some of the best Poisson image denoising methods. A second reason for using an additive Gaussian noise model is simplicity: it is a reasonable assumption and it makes the problem of denoising easier to approach.

In this paper we will only focus on the problem of denoising images contaminated with additive Gaussian noise. FFDNet is designed to work with RGB or grayscale images—and not other types of data such as raw images. Let $\mathbf{I}$ be a noiseless image, while $\tilde{\mathbf{I}}$ is its noisy version corrupted by a Gaussian white noise $\mathbf{N}$, then

$$\tilde{\mathbf{I}} = \mathbf{I} + \mathbf{N} . \tag{1}$$

Observe that most experiments focus on the case of white additive Gaussian noise (AWGN), but this algorithm can be potentially applied to other types of noise such as spatially varying noise.

The rest of the paper is organized as follows. In Section 2 we introduce the FFDNet architecture and training details. The following sections are devoted to studying the key attributes of FFDNet. In Section 3 we study the impact of residual learning and in Section 4 the impact of upscaling layers. Section 5 analyzes the importance of the orthogonalization regularizer, while Section 6 presents a study regarding the choice of the loss. Section 7 contains an experimental comparison with different state of the art methods, and Section 8 concludes the paper.

2 FFDNet Architecture

A standard feed-forward network architecture is used to implement FFDNet, as shown in Figure 1. The network is composed of D convolutional layers, which share the same structure. Each of these has W outputs, and the spatial size of their kernels is $K \times K$.

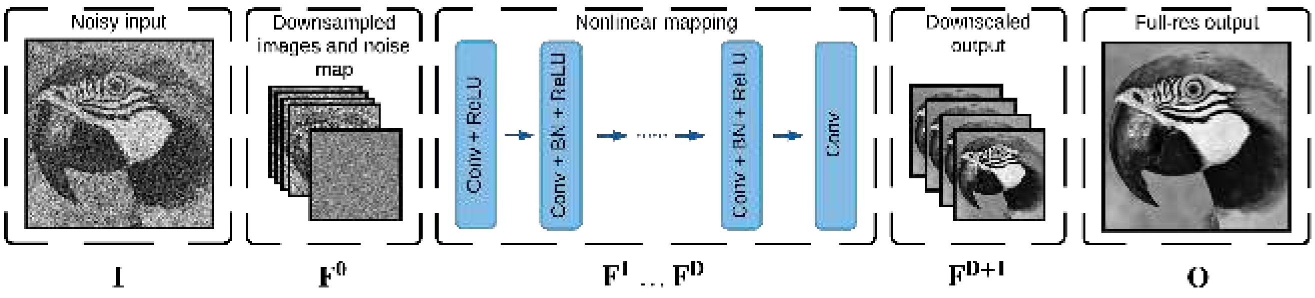


Figure 1: Architecture of FFDNet.

Preprocessing layer The network first reorganizes the pixels of an $n_{ch} \times h \times w$ input image $\mathbf{I}$ into a lower resolution image of size $4n_{ch} \times h/2 \times w/2$. Layer $\mathbf{F^0}$ extracts 2×2 patches and reorganizes their pixels in the different channels of the output image according to

$$F^0[\,c,\,x,\,y\,] = I\left[\,\left\lfloor \frac{c}{4} \right\rfloor,\, 2x + (c \bmod 2),\, 2y + \left\lfloor \frac{c}{2} \right\rfloor\,\right]\,, \tag{2}$$

where $0 \leq c < 4n_{ch}$, $0 \leq x < h$, $0 \leq y < w$. The majority of the processing will be performed at this reduced scale. An extra channel of the same resolution composed of an estimate of the noise map $\mathbf{M}$ is added to the input. This noise map controls the trade-off between denoising and detail preservation. For spatially invariant Gaussian noise with standard deviation σ, the noise map is uniform with all its elements equal to σ. Figure 2 shows a diagram of this layer.

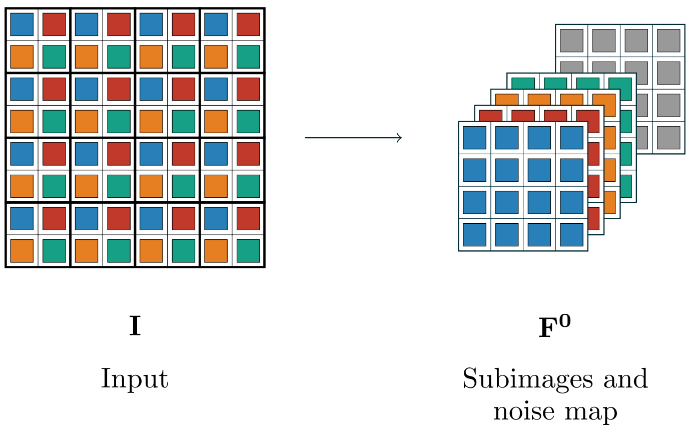


Figure 2: Diagram of the downscaling layer.

Nonlinear mapping A nonlinear mapping composed of D convolutional layers (layers $\mathbf{F^1} \ldots \mathbf{F^D}$) comes after the preprocessing layer $\mathbf{F^0}$ (see Figure 1). Each of these layers consists of W convolutional filters of spatial size $K \times K$. The outputs of layers $\mathbf{F^1}$ to $\mathbf{F^{D-1}}$ are followed by point-wise ReLU [11] activation functions $ReLU(\cdot) = \max(\cdot, 0)$. At training time, batch normalization layers (BN [9]) are placed between the convolutional and ReLU layers in $\mathbf{F^2}$ to $\mathbf{F^{D-1}}$. At evaluation time, the batch normalization layers are removed, and replaced by an affine layer that applies the learned normalization. Then, the c-*th* channel of the d-*th* layer, $\mathbf{F^d_c}$ can be written as

$$\mathbf{F^d_c} = ReLU\left(\sum_{c'=0}^{D-1} \mathbf{w^d_{cc'}} * \mathbf{F^{d-1}_{c'}}\right) \quad \text{for } c \in \{0 \ldots W-1\}\,, \tag{3}$$

where $\mathbf{w^d_{cc'}}$ is a two-dimensional convolution kernel of size $K \times K$ (the c-*th* three-dimensional filter of layer $\mathbf{F^d}$ is the collection of the W two-dimensional filters $\mathbf{w^d_{cc'}}$). Summarizing the characteristics of the D layers of the nonlinear mapping, we have

- Layer $\mathbf{F^1}$: Conv+ReLU. W filters of size $(4n_{ch} + 1) \times K \times K$ generate W feature maps. A point-wise ReLU activation function is used as nonlinearity.
- Layers $\mathbf{F^2} \ldots \mathbf{F^{D-1}}$: Conv+BN+ReLU. W filters of size $W \times K \times K$ are used. At training time, batch normalization layers are placed between the convolutional and ReLU layers.
- Layer $\mathbf{F^D}$: Conv. $4n_{ch}$ filters of size $W \times K \times K$ are used in this layer.

The input of each convolutional layer is zero-padded by $\frac{K-1}{2}$ so that the spatial size does not decrease with depth. The stride is set to one in all cases.

Postprocessing layer Finally, layer $\mathbf{F^{D+1}}$ upsamples the low-resolution output of $\mathbf{F^{D}}$ back into the original resolution. That is, it repacks its input of dimension $4n_{ch} \times h/2 \times w/2$ into an image of size $n_{ch} \times h \times w$ by reversing Equation (2), as shown in Figure 3. The total number of layers is equal

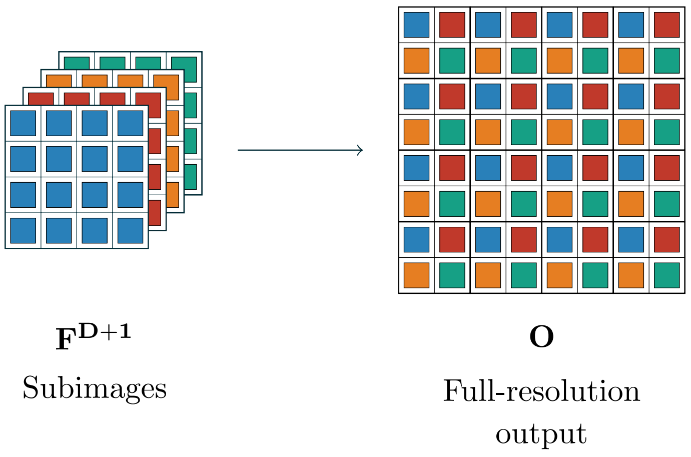


Figure 3: Diagram of the upscaling layer.

to $D+2$, where D is the number of convolutional layers. The spatial size of the convolutional kernels K is equal to 3. The depth D is set to 15 for grayscale denoising, and 12 for color denoising. As for the number of feature maps W, it is set to 64 and 96 for grayscale and color denoising, respectively. These settings represent a good compromise between complexity and denoising performance. Note that the network designed for color denoising is shallower ($D_{RGB} < D_{gray}$) but wider ($W_{RGB} > W_{gray}$) than the grayscale denoising network. This increased width contributes to a better color consistency in the results as the correlations between the color channels are better treated by the network. As will be discussed in Section 7, FFDNet features remarkable chroma noise handling. In any case, the global capacity of the color denoising network is larger than the grayscale network, as the number of learnable parameters of the former is 8.5×10^5 versus 5.6×10^5 of the latter.

2.1 Comparison to DnCNN

DnCNN and FFDNet share several attributes. In particular, the architectures of their nonlinear mappings are comparable—generally speaking, they are a collection of Conv+BN+ReLU layers with filters of spatial size 3×3. Nevertheless, FFDNet implements additional techniques which render this algorithm faster, more efficient, and more versatile than its predecessor. In terms of the design of the architecture, the FFDNet network is wider but shallower than DnCNN, as shown in Table 1. However, FFDNet features a larger receptive field. Contrary to DnCNN, FFDNet does not implement residual learning, i.e. it estimates the latent image instead of the input noise. Regarding the training details of each model, the amount of patches which compose the training set goes from 384000 for DnCNN to 1024000 for FFDNet. The number of epochs is also increased. These parameters contribute to an increase of performance in FFDNet, notably when dealing with strong noise as well as handling chroma noise.

Other salient characteristics of FFDNet are:

- Regularization by orthogonalization of the convolutional filters.
- Use of downscaling and upscaling layers before and after the nonlinear mapping.
- Introduction of a noise map as an input channel.

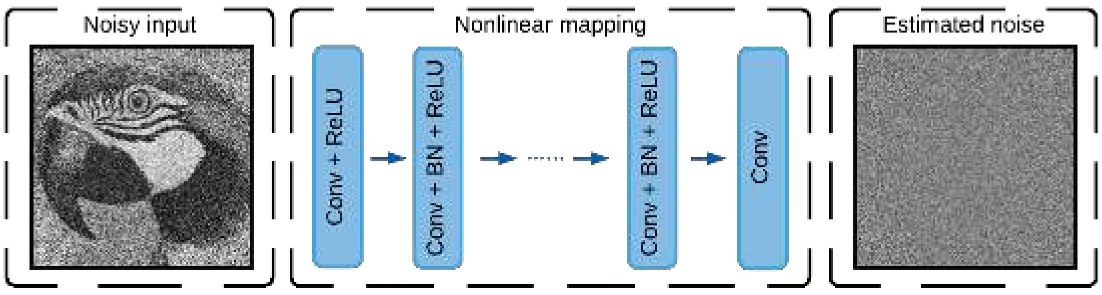


Figure 4: Architecture of DnCNN. Note that there are no down- and upscaling layers. The characteristics of the nonlinear mapping are similar to those of FFDNet.

	DnCNN	FFDNet gray	FFDNet RGB
Depth (conv layers)	20	15	12
Width (feature maps per layer)	64	64	96
Receptive field	41×41	62×62	50×50
Learnable parameters	6.7×10^5	5.6×10^5	8.5×10^5

Table 1: Comparison of characteristics of FFDNet and DnCNN.

See Figure 4 for a DnCNN diagram and Figure 1 for a FFDNet diagram.

In this paper, we are interested in exploring these differences more in detail to identify their benefits. The techniques mentioned above will be discussed in the following sections.

2.2 Training Details

The training dataset is composed of pairs of input-output patches $\left\{\left((\tilde{\mathbf{I}}_j, \mathbf{M}_j),\, \mathbf{I}_j\right)\right\}_{j=0}^{m}$ which are generated by adding AWG of $\sigma \in [0,\, 75]$ to the clean patches $\mathbf{I}_j$ and building the corresponding noise map $\mathbf{M}_j$ (which is in this case constant with all its elements equal to σ). A total of $m = 128 \times 8000$ patches are extracted from the Waterloo Exploration Database [14], where the mini-batch size is 128. The patch size is 64×64 and 50×50 for grayscale and color images, respectively. Patches are randomly cropped from randomly sampled images of the training dataset. Data is augmented five times by introducing rescaling by different scale factors and random flips[2]. In the cases in which residual learning is used, the network outputs an estimation of the input noise

$$\mathcal{F}(\,\tilde{\mathbf{I}}\,) = \hat{\mathbf{N}}\,. \tag{4}$$

Then, the denoised image is computed by subtracting the output noise to the noisy input

$$\hat{\mathbf{I}} = \tilde{\mathbf{I}} - \mathcal{F}(\,\tilde{\mathbf{I}}\,)\,. \tag{5}$$

In this case, the loss function is the following

$$\mathcal{L}_{res}(\theta) = \frac{1}{2m}\sum_{j=1}^{m}\left\|\mathcal{F}(\,(\tilde{\mathbf{I}}_j, \mathbf{M}_j);\, \theta\,) - \mathbf{N}_j\right\|^2, \tag{6}$$

[2]As a side note, a model with equivalent performance was trained by extracting overlapping patches from the images of the database. This way only requires about one third of the amount of images in the Waterloo database to extract the necessary amount of patches.

where θ is the collection of all learnable parameters.

On the other hand, models without residual learning estimate the denoised image directly, i.e.

$$\mathcal{F}(\tilde{\mathbf{I}}) = \hat{\mathbf{I}}\,, \tag{7}$$

resulting in the following loss function

$$\mathcal{L}_{no-res}(\theta) = \frac{1}{2m}\sum_{j=1}^{m}\left\|\mathcal{F}(\,(\tilde{\mathbf{I}}_{\mathbf{j}}, \mathbf{M}_{\mathbf{j}});\,\theta\,) - \mathbf{I}_j\right\|^2 . \tag{8}$$

In all cases, the ADAM algorithm [10] is applied to minimize the loss function, with all its hyper-parameters set to their default values. The number of epochs is set to 80. The scheduling of the learning rate is also common to all cases. It starts at 1e−3 for the first 50 epochs, then changes to 1e−4 for the following 10 epochs, and finally switches to 1e−6 for the remaining of the training. This scheduling strategy is in the same spirit as the one proposed by the authors of FFDNet. It is implemented in the code provided with the use of the *milestone* parameters. During training, after the number of epochs surpasses the first milestone, the learning rate is divided by 10, and is divided again by 100 after the second milestone is surpassed. The actual values of the these parameters have been chosen heuristically after observing the loss graphs of several trainings. In other words, a learning rate step decay is used in conjunction with ADAM. The mix of learning rate decay and adaptive rate methods has also been applied to other deep learning projects [29, 25, 26, 30], usually with positive results.

Two datasets are used to evaluate the grayscale and color FFDNet denoisers. The BSD68 database and Set12 databases are utilized for grayscale denoising, while the CBSD68 and the Kodak24 are used for color denoising. The BSD68 is composed of 68 images of size 481×324 from the testing set (testset for short) of the BSD database [17], while CBSD68 corresponds to the color version of this image set. The Set12 dataset is a collection of standard test images of sizes either 256×256 or 512×512. Finally, the Kodak24 (Kodak Lossless True Color Image Suite[3]) is composed of 24 768×512 color images. As for validation during training, all the images from the Kodak24 suite are used as the validation dataset for color denoising, while the Set12 database is used for grayscale denoising. The open-source implementation of the FFDNet method provided with this paper is based on PyTorch [18], a popular machine learning library for Python. Testing and training code is available from the IPOL website.

Table 2 shows a comparison of $PSNR$ on the two color datasets obtained with the code provided by the authors and the code proposed in this paper. It can be observed that both versions perform very similarly.

3 Residual Learning

Residual networks were introduced by He et al. in [8] and have become since then state-of-the-art in several CNN related fields. These networks feature "residual" or "skip" connections which copy features from shallower layers directly to deeper ones by skipping intermediary layers. Skip connections fast-forward the identity mapping through the network, thus allowing the later to learn a residual instead of the absolute mapping. Propagating the identity through several nonlinear layers is harder than through the skip connection. The use of skip connections thus eases the training process. Residual architectures have also been successfully applied in the fields of image denoising and restoration [7, 20, 23]. In particular, DnCNN implements residual learning as it estimates the

[3]http://r0k.us/graphics/kodak/.

	CBSD68	Ours	Zhang et al.	Kodak24	Ours	Zhang et al.
$\sigma = 15$		33.76	33.80		34.53	34.55
$\sigma = 25$		31.18	31.18		32.12	32.11
$\sigma = 35$		29.58	29.57		30.59	30.56
$\sigma = 45$		28.45	28.42		29.49	29.45
$\sigma = 55$		27.58	27.55		28.63	28.58
$\sigma = 70$		26.57	26.52		27.61	27.55

Table 2: Comparison of $PSNR$ obtained with our FFDNet implementation and the implementation provided by the author. Both implementations perform similarly.

input noise instead of the latent image. Its authors argue that residual learning facilitates training and improves the performance of the network, especially when used in combination with batch normalization [33].

Despite the apparent advantages of residual learning in the case of DnCNN, FFDNet does not apply this technique. In this section, we would like to investigate further into the possible benefits of applying residual learning to plain CNN denoisers such as FFDNet. To this end, two different FFDNet models were trained: one implementing residual learning and one without residual connections. See Section 2.2 for more details about the training. Figure 5 shows the validation $PSNR$ during the training of both models. The validation set is the Kodak24 image suite. The noise added to the images was of standard deviation $\sigma = 25$. The final $PSNR$ values are 32.16dB for FFDNet model with residual learning and 32.11dB for the model without residual learning. Although the performance of both models is close, the $PSNR$ curve for the residual model appears to be somehow smoother and is always slightly above the curve of the model without residual connection in the last 20 epochs of fine-tunning. Table 3 displays the average $PSNR$ on two different color testsets. In all cases, there are small to moderate differences in favor of the model with residual learning. Notably, this differences are larger for smaller values of σ. Although improvements in performance with residual learning are modest, it is clear that the use of this technique in plain CNN denoisers such as FFDNet is advantageous, as there is virtually no increase in the overall complexity of the algorithm. Therefore, the rest of the models for this paper implement residual learning.

4 Examining the Downscaling and Upscaling Layers

As discussed in Section 2, FFDNet first reorganizes the pixels of the $n_{ch} \times h \times w$ input image $\mathbf{I}$ into a lower resolution image of size $4n_{ch} \times h/2 \times w/2$. An extra channel of the same resolution composed of an estimate of the noise map $\mathbf{M}$ is added to the input. The main advantage of this technique is that processing the subimages instead of the full resolution input leads to sensible reductions in running times and memory requirements, without sacrificing denoising performance [7, 34].

Another advantage is that downscaling the image effectively doubles the receptive field. The receptive field of a CNN with a similar nonlinear mapping and without downsampling is $D(K-1)+1$. It would take twice as many convolutional layers ($D' = 2D$) to attain the same size of receptive field in this case. Such increase of depth would have a huge impact on the final complexity of the algorithm. FFDNet features a receptive field of 62×62 for the grayscale denoising network and 50×50 for the color denoising network, which is comparable to receptive fields of other state-of-the-art algorithms.

A similar technique was previously proposed by Gharbi et al. in [7]. Their algorithm rearranges the four color channels of the Bayer input mosaic in a lower-resolution multi-channel image. An estimate

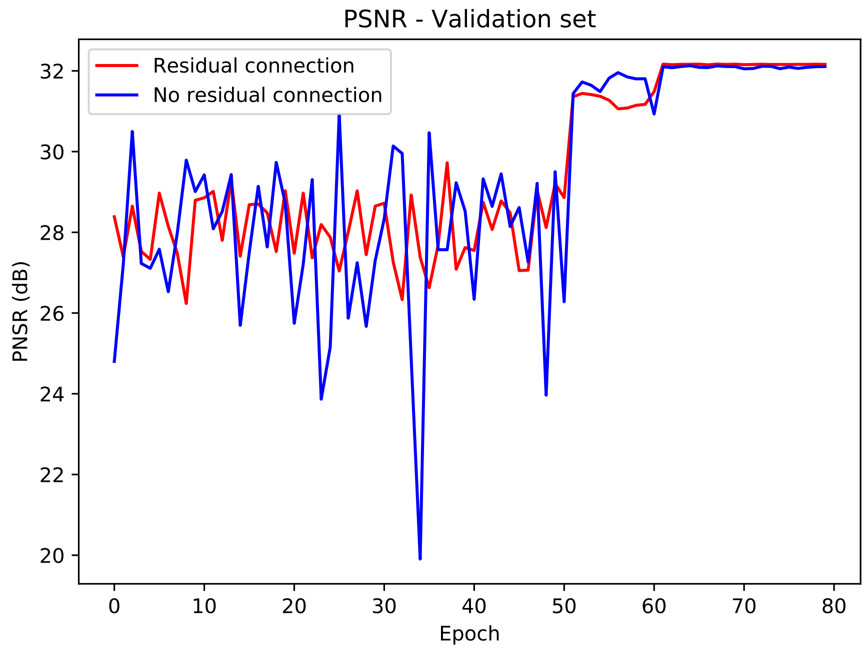


Figure 5: *Validation PSNR during training with and without residual learning.* The validation set is the Kodak24 image suite. The noise added to the images was of standard deviation $\sigma = 25$. The final $PSNR$ values of these two representative trainings are 32.16dB for FFDNet model with residual learning and 32.11dB for the model without residual learning.

CBSD68 testset	With residual connection	Without residual connection
$\sigma = 15$	**33.76**	33.53
$\sigma = 25$	**31.18**	31.05
$\sigma = 35$	**29.58**	29.50
$\sigma = 45$	**28.45**	28.39
$\sigma = 55$	**27.58**	27.53
$\sigma = 70$	**26.57**	26.53
Kodak24 testset	**With residual connection**	**Without residual connection**
$\sigma = 15$	**34.53**	34.26
$\sigma = 25$	**32.12**	31.98
$\sigma = 35$	**30.59**	30.50
$\sigma = 45$	**29.49**	29.42
$\sigma = 55$	**28.63**	28.57
$\sigma = 70$	**27.61**	27.58

Table 3: Comparison of $PSNR$ obtained with FFDNet models with and without residual connections. Values shown are the average of three different trainings.

of the noise level is also concatenated to the input as an additional channel. In their case, subscaling the input image appears as a natural step, as it is equivalent to repacking the Bayer mosaic in four individual color channels. On the contrary, the reason to apply this downsampling technique in the case of RGB or grayscale images does not appear as straightforward. The advantages of working with the downsampled subimages instead of the full resolution input image are clear. However, what does not appear obvious at first glance is how the pixels are handled in this process in the first layers of the network. To better understand this procedure, the rest of this section will further explore it.

As can be seen in Figure 6, reorganizing the input into a quarter-resolution image with four times as many channels and applying a 3×3 convolution (layers $\mathbf{F^0}$ and $\mathbf{F^1}$) is similar to convolving the input image with 6×6 kernels and stride equal to two. Let us call this new convolutional layer $\mathbf{F}^*$, and let us consider all non-overlapping 2×2 patches of the grayscale input image $\mathbf{I}$. The first channel of the output of $\mathbf{F^0}$ is composed of all the top-left pixels of these patches. The 3×3 kernels of $\mathbf{F^1}$ will only be centered on these pixels. Thus, one out of two pixels will have to be skipped in $\mathbf{F}^*$.

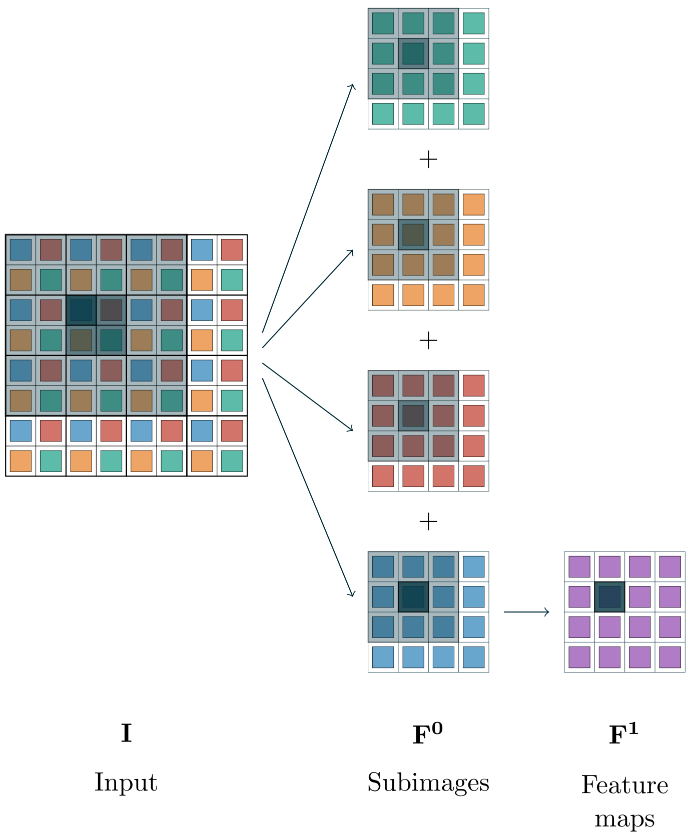


Figure 6: *Downscaling process in FFDNet.* The pixel in gray in the output feature map in $\mathbf{F^1}$ is the result of convolving the pixels in light gray in $\mathbf{F^0}$. These pixels covered by the 3×3 kernels in $\mathbf{F^0}$ span over a 6×6 window in the input image $\mathbf{I}$.

Figure 7 displays the evolution of validation $PSNR$ during two representative trainings of a regular FFDNet model and an equivalent model without the downscaling layer. In the latter, layers $\mathbf{F^0}$ and $\mathbf{F^1}$ were replaced by the equivalent convolutional layer $\mathbf{F}^*$ with kernels of size 6×6 and stride 2. A full resolution three-channel noise map is concatenated to the input at this layer. The validation set is the Kodak24 image suite. The noise added to the images was of standard deviation $\sigma = 25$. The final $PSNR$ values are 32.11dB for the FFDNet model with the downscaling layer and 32.10dB for the equivalent model without the downscaling layer. Table 4 displays the $PSNR$ of the denoised Kodak24 and CBSD68 testsets. Values shown are the average of three different trainings. The differences between results of both models are negligible. It can thus be concluded that both techniques lead to equivalent models. There is however a difference in running times: the model with larger convolutional kernels takes about 40% longer to run.

As for the upscaling process, FFDNet implements a repacking of the pixels, as introduced by Shi et al. in [24]—note that a similar technique was also proposed by Gharbi et al. in [7]. An alternative way to perform the upscaling of the sub-images in $\mathbf{F^{D+1}}$ would be to use dilated convolutions [31]. However, dilated convolutions are less effective than sub-pixel convolutions and tend to create artifacts such as "gridding" [33, 32].

In conclusion, the combination of the described downscaling and upscaling methods implemented by FFDNet is a clever way of doubling the receptive field, that sensibly reduces runtimes and memory requirements while maintaining the denoising performance. In addition, this strategy does not suffer from artifacts like other alternatives such as dilated convolutions.

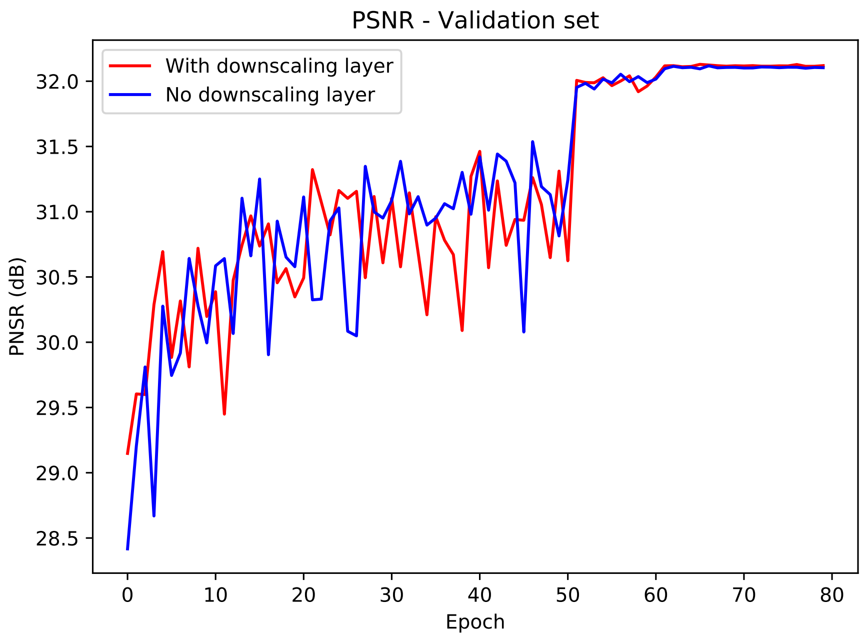


Figure 7: *Validation PSNR during training with and without downscaling layers.* The validation set is the Kodak24 image suite. The noise added to the images was of standard deviation $\sigma = 25$. The final $PSNR$ values of these two representative trainings are 32.11dB for the FFDNet model with the downscaling layer and 32.10dB for the equivalent model without the downscaling layer. Orthogonalization was disabled in both cases.

5 Examining the Noise Map and Orthogonalization

In a similar fashion as done by Burger et al. [2] and Gharbi et al. [7], FFDNet incorporates the estimate of the noise level σ as additional information into the convolutional architecture. As shown in Figure 2, the noise level is spatially replicated to match the input dimensions of layer $\mathbf{F^0}$ and is concatenated as an extra channel. The noise map $\mathbf{M}$ can be composed of multiple channels to represent the noise existent in the R, G, and B channels of color images. In principle, $\mathbf{M}$ can be non-uniform to represent spatially variant noises. The inclusion of the noise map $\mathbf{M}$ as input facilitates the handling of a variety of noise levels with a single CNN model.

The noise map $\mathbf{M}$ can also be seen as a control of the trade-off between noise reduction and detail preservation. In effect, if one wanted to further denoise an image, the values of $\mathbf{M}$ could be increased to augment the noise reduction at the expense of smoothing the result and loosing detail. In order to guarantee the role of $\mathbf{M}$ as a control of said trade-off, the authors of FFDNet propose to

CBSD68 testset	Without downscaling layer	With downscaling layer
$\sigma = 15$	33.71	33.72
$\sigma = 25$	31.12	31.14
$\sigma = 35$	29.54	29.56
$\sigma = 45$	28.41	28.43
$\sigma = 55$	27.54	27.56
$\sigma = 70$	26.53	26.56
Kodak24 testset	**Without downscaling layer**	**With downscaling layer**
$\sigma = 15$	34.53	34.49
$\sigma = 25$	32.12	32.09
$\sigma = 35$	30.59	30.56
$\sigma = 45$	29.49	29.45
$\sigma = 55$	28.63	28.59
$\sigma = 70$	27.61	27.59

Table 4: Comparison of $PSNR$ obtained with FFDNet models with and without downscaling layers. Values shown are the average of three different trainings. No orthogonalization was applied.

regularize the convolution filters by orthogonalization during training. This regularization method is explained in the following. Let us suppose a given convolutional layer is composed of C_{out} kernels of size $C_{in} \times K \times K$. These filters are first reshaped into a two-dimensional matrix $\mathbf{W_{ker}}$ of size $(K \times K \times C_{in}) \times C_{out}$. A singular value decomposition (SVD) is then performed on the matrix $\mathbf{W_{ker}}$, i.e. $(\mathbf{U}, \mathbf{S}, \mathbf{V}) = SVD(\mathbf{W_{ker}})$, where $\mathbf{W_{ker}} = \mathbf{U}\,\mathbf{S}\,\mathbf{V^T}$, $\mathbf{S}$ is a diagonal matrix of singular values, and $\mathbf{U}$ and $\mathbf{V}$ are two orthogonal matrices. The orthogonality between the kernels of the layer is enforced by setting the singular values of $\mathbf{S}$ to 1. At training time, this orthogonalization procedure is performed only every T iterations. Once the learning rate becomes small ($\textit{learning rate} \leq 1 \times 10^{-6}$) the regularization process is abandoned. Algorithm 1 summarizes the procedure.

Algorithm 1: Orthogonal Regularization

```
input  : $C_{out}$ kernels of size $C_{in} \times K \times K$ of a given convolutional layer
output: new orthogonal kernels

while $\textit{learning rate} > 1 \times 10^{-6}$ do
    if $\textit{iteration}$ mod $T = 0$ then                Perform every $T$ iterations.
        Build a matrix $\mathbf{W_{ker}}$ by placing the kernels as columns of this matrix
        $(\mathbf{U}, \mathbf{S}, \mathbf{V}) = SVD(\mathbf{W_{ker}})$
        Set singular values of $\mathbf{S}$ to 1
        $\mathbf{W'_{ker}} = \mathbf{U}\,\mathbf{S}\,\mathbf{V^T}$
        Replace the old kernels with the columns of the new matrix $\mathbf{W'_{ker}}$
```

Figure 8 shows the validation $PSNR$ during two representative trainings with and without regularization by orthogonalization of the kernels of each layer. Here, the same FFDNet model was trained twice: once by applying the procedure described in Algorithm 1 as regularization method, and once without regularization at all. The validation set is the Kodak24 image suite. The noise added to the images was of standard deviation $\sigma = 25$. The final values of $PSNR$ are 32.16dB for the training with regularization and 32.12dB for the training without regularization. It can be observed that this type of regularization slows down the convergence rate and the evolution of the model performance, as the validation $PSNR$ of the regularized model stays well under that of the model with no regularization for most of the training duration. The learning rate is set to 1×10^{-6} at epoch 60 and regularization is not used anymore. It is at this point that the performance of the regularized model recovers to the level of performance of the model with no regularization. This point indicates that while orthogonal initialization may be beneficial, maintaining the orthogonality constraint during all the training procedure can be detrimental. These observations are in line with the findings of [27].

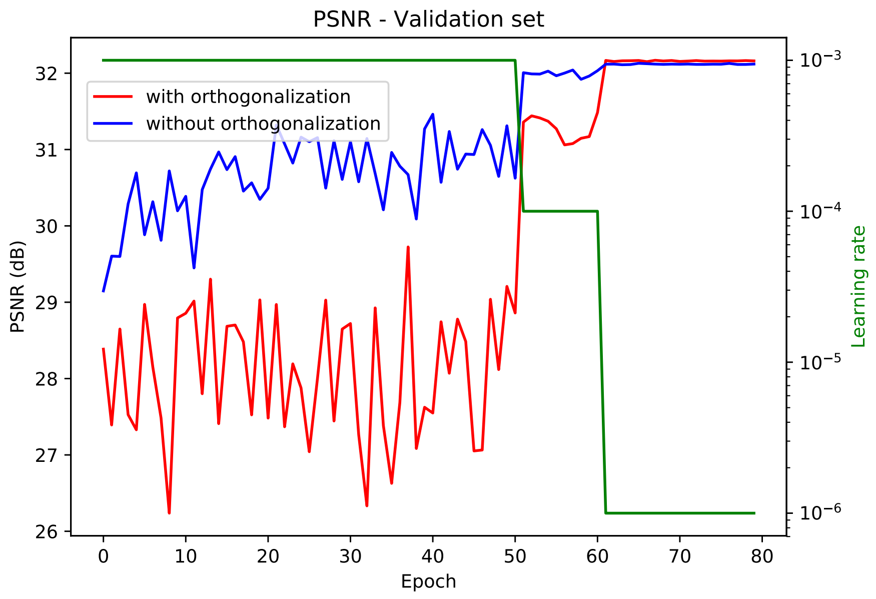


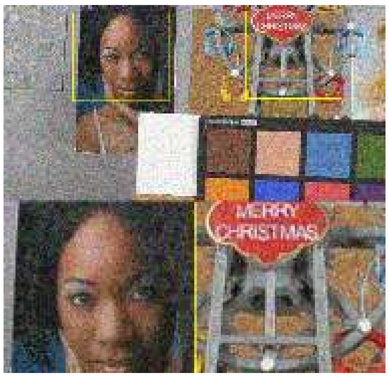
Figure 8: *Validation PSNR during two representative trainings with and without regularization by orthogonalization of the kernels of each layer.* The validation set is the Kodak24 image suite. The noise added to the images was of standard deviation $\sigma = 25$. Regularization is abandoned when the learning rate becomes 1×10^{-6}. The final values of $PSNR$ are 32.16dB for the training with regularization and 32.12dB for the training without regularization.

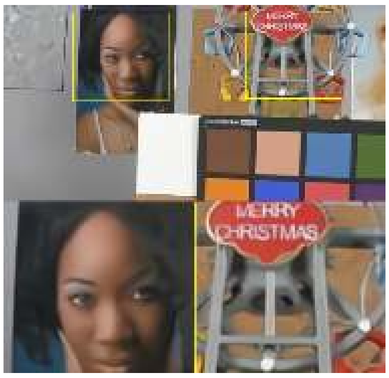
The regularization by orthogonalization of the convolutional filters has been proposed by the authors of FFDNet as a means of avoiding visual artifacts in the event of mismatch between the noise level $\sigma_{\mathbf{M}}$ indicated in the noise map and the actual noise in the input σ, particularly in the case in which the noise level in the noise map is higher than the latter ($\sigma_{\mathbf{M}} > \sigma$). However, it is clear than in such case the output of the network would be overly-smooth, as can be observed in Figure 9. Visual artifacts are not likely to appear in these conditions. The regularized model performs marginally better than the one without regularization as it recovers slightly more structures. Table 5 compares the PSNR of the denoised Kodak24 and CBSD68 testsets of the models trained with and without regularization.

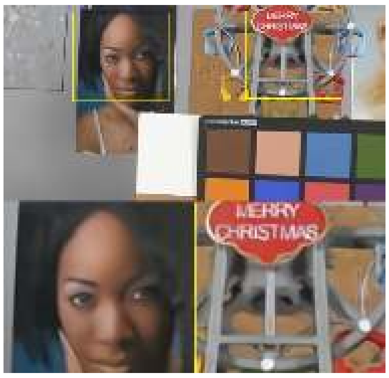
6 Training with the L_2 Error Function Versus the L_1 Error Function

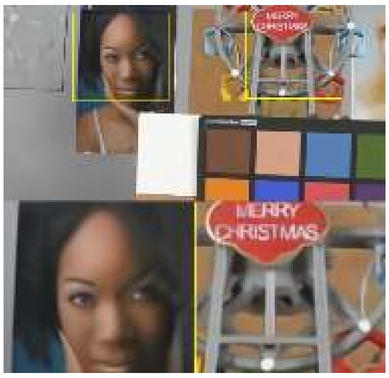
Recently, a number of related image restoration methods have proposed the use of different loss functions to train their networks [3, 35, 23]. In the same spirit, we compare in this section the results

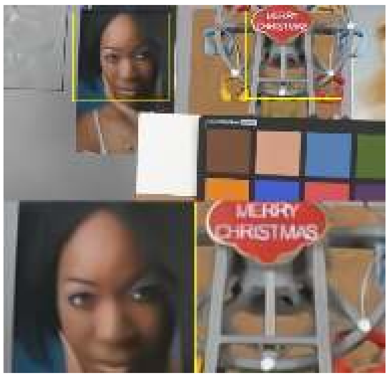
(a) Clean image

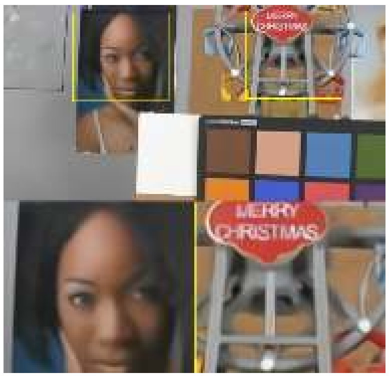

(b) Noisy $\sigma = 25$


(c) $\sigma_{\mathrm{M}} = 30$, $PSNR = 34.79$dB


(d) $\sigma_{\mathrm{M}} = 30$, $PSNR = 34.73$dB


(e) $\sigma_{\mathrm{M}} = 35$, $PSNR = 34.07$dB


(f) $\sigma_{\mathrm{M}} = 35$, $PSNR = 34.02$dB


(g) $\sigma_{\mathrm{M}} = 40$, $PSNR = 33.44$dB

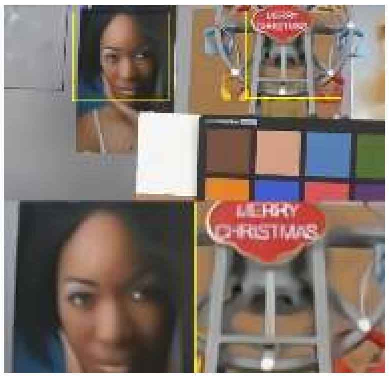

(h) $\sigma_{\mathrm{M}} = 40$, $PSNR = 33.36$dB

Figure 9: *Comparison of models with and without orthogonalization in case of noise mismatch.* Left: regularized model. Right: model with no regularization. The noise level σ_{M} indicated in the noise map mismatches the actual noise in the input σ. The regularized model performs marginally better than the one without regularization as it recovers slightly more details and structures.

CBSD68 testset	With orthogonalization	Without orthogonalization
$\sigma = 15$	**33.75**	33.71
$\sigma = 25$	**31.17**	31.14
$\sigma = 35$	**29.57**	29.55
$\sigma = 45$	**28.45**	28.42
$\sigma = 55$	**27.58**	27.56
$\sigma = 70$	**26.57**	26.55
Kodak24 testset	**With orthogonalization**	**Without orthogonalization**
$\sigma = 15$	**34.52**	34.49
$\sigma = 25$	**32.13**	32.08
$\sigma = 35$	**30.59**	30.55
$\sigma = 45$	**29.49**	29.44
$\sigma = 55$	**28.63**	28.60
$\sigma = 70$	**27.60**	27.58

Table 5: Comparison of $PSNR$ obtained with FFDNet models trained with and without regularization by orthogonalization of the convolution kernels. Values shown are mean values among three different trainings.

of FFDNet when trained with the L_2 error versus the L_1 error in the loss function. In the latter, the error in the loss function shown in Equation (6) is replaced by the L_1 error.

Figure 10 displays the evolution of the $PSNR$ of the validation imageset during training. The validation set is the Kodak24 image suite. It can be observed that the $PSNR$ evolves similarly in both cases, with a slight advantage to the L_2 loss function (32.09 versus 32.16dB). Table 6 compares the PSNR of the denoised Kodak24 and CBSD68 testsets.

Figure 11 presents a visual comparison of results of both cases. The image on the top was contaminated with AWGN of $\sigma = 40$ ($PSNR = 16.09$dB). The middle column shows the results of the model trained with the L_2 error ($PSNR = 32.08$dB), while the column on the right displays the results of the model trained with the L_1 error ($PSNR = 32.06$dB). The L_1 error renders flat areas smoother, as less low-frequency noise can be observed in this case. Generally speaking, this fact accounts for results which are more visually appealing. On the other hand, detail information is better preserved with the L_2 error, while dappled artifacts can be observed in flat areas. This is due to increased L_2 penalization on large errors and relative tolerance to small errors, regardless the structure of the image.

	CBSD68	L_2 error	L_1 error	Kodak24	L_2 error	L_1 error
$\sigma = 15$		**33.76**	33.72		**34.53**	34.46
$\sigma = 25$		**31.18**	31.11		**32.12**	32.05
$\sigma = 35$		**29.58**	29.51		**30.59**	30.51
$\sigma = 45$		**28.45**	28.38		**29.49**	29.41
$\sigma = 55$		**27.58**	27.52		**28.63**	28.55
$\sigma = 70$		**26.57**	26.51		**27.61**	27.54

Table 6: Comparison of $PSNR$ obtained with FFDNet models trained with the L_2 and the L_1 error.

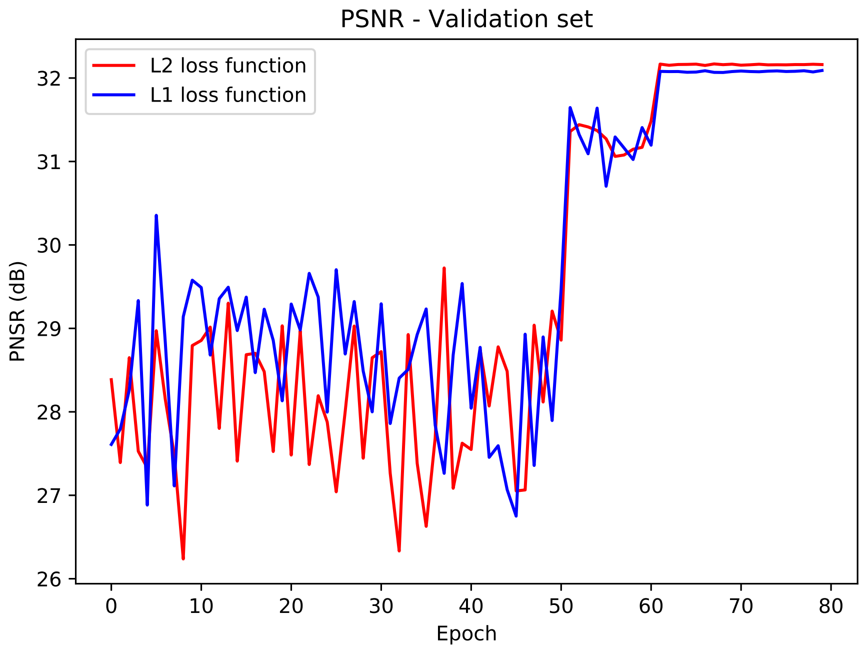


Figure 10: *Validation $PSNR$ for FFDNet trained with the L_2 error and the L_1 error in the loss function.* The validation set is the Kodak24 image suite. The noise added to the images was of standard deviation $\sigma = 25$. The final values of $PSNR$ of these two representative trainings are 32.09dB for the L_1 error and 32.16dB for the L_2 error.

7 Benchmarking

This section presents the results of FFDNet on noisy images corrupted with AWGN. We compare FFDNet with other state-of-the-art methods. First, three non-local patch-based methods were selected: BM3D [5, 12], Non-local dual image denoising (NLDD [19]), and Non-local Bayes (NLB [13]). The respective code of each of these algorithms was downloaded from IPOL. Additionally, we tested DnCNN [33], and also TNRD [4] on the grayscale testsets. The code of these algorithms was downloaded from the website of their respective authors[4]. When running DnCNN, its "blind" model was used—that is, one model trained for values of $\sigma \in [0, 55]$. As for TNRD, this method must be run with one specific model for each value of σ. The rest of the algorithms were run with default parameter values.

Table 7 shows average $PSNRs$ on four testsets: two color testsets, CBSD68 and Kodak24, and two grayscale testsets, BSD68 and Set12. It can be seen that DnCNN and FFDNet surpass all the other methods, usually by a large margin. Overall, FFDNet averages a margin of 0.51dB over NLDD, 0.69dB over NLB, and 0.61dB over BM3D. DnCNN is a close competitor of FFDNet, and even slightly surpasses the latter in some cases ($\sigma \leq 25$ in CBSD68 and $\sigma = 15$ in BSD68). FFDNet has a larger receptive field than DnCNN, which favors the removal of higher values of noise.

Figures 12 and 13 present two examples for visual comparison of color denoising results of the aforementioned methods. It can be observed that BM3D shows artifacts related to wavelet filters, i.e. ringing, while NLB presents artifacts associated with non-local patch-based methods, such as staircasing artifacts on smooth regions. Even if NLDD is also a non-local method, its second denoising stage based on dual-domain denoising is able to reduce the artifacts associated with non-local methods. In general, these methods tend to smooth details and textures, especially for strong values of noise. In comparison, FFDNet is able to preserve and recover more details, while keeping flat areas

[4]TNRD: `http://www.escience.cn/people/chenyunjin/RelatedCodes.html`. DnCNN: `https://github.com/cszn/DnCNN` .

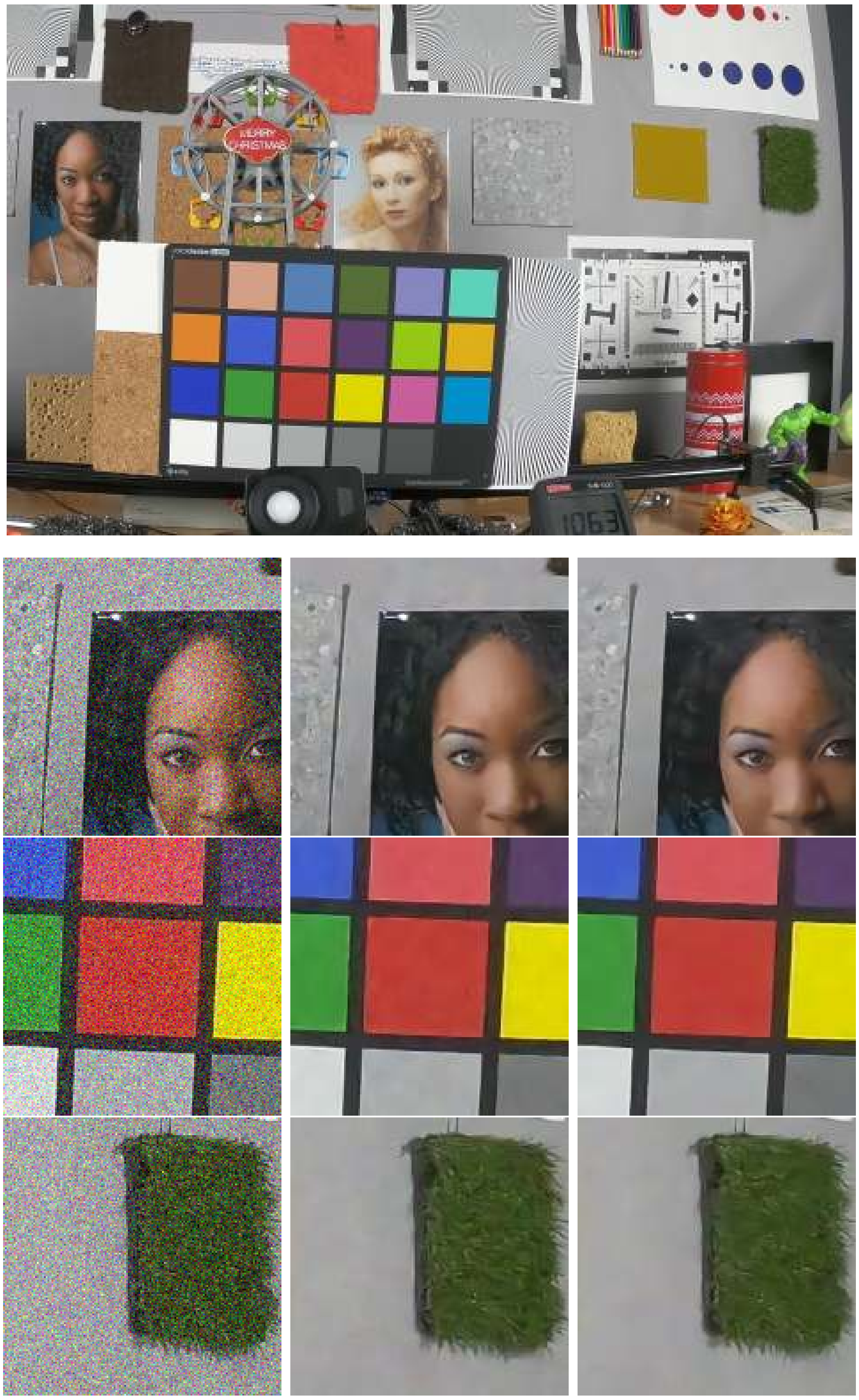


Figure 11: *Comparison of results with L2 and L1 loss functions.* Left to right: noisy image ($\sigma = 40$, $PSNR = 16.09\text{dB}$), result of a model trained with the L2 loss ($PSNR = 32.08\text{dB}$), result of a model trained with the L1 loss ($PSNR = 32.06\text{dB}$). The image on the top is the original image. Flat areas appear smoother for the L1 loss. There is however a loss of detail information with respect to the L2 loss result.

smooth. The strong color consistency of FFDNet results can be observed, particularly in Figure 12, where chroma noise remains in the results of methods such as BM3D or NLB. Figure 14 shows a grayscale denoising example containing self-similar structures. It can be observed that TNRD creates artifacts in flat areas. In this example, both DnCNN and FFDNet are surpassed by the three non-local patch-based methods: repetitive structures as the ones found in this picture are better handled by methods which exploit the non-local similarity prior. On the other hand, CNN based methods handle non repetitive textures very well. Generally speaking, FFDNet has arguably the most appealing perceptual quality among the presented methods.

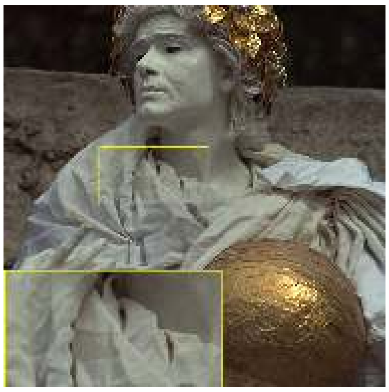
CBSD68 testset	NLDD	NLB	BM3D	DnCNN	FFDNet
$\sigma = 15$	33.16	33.14	33.45	**33.89**	33.76
$\sigma = 25$	30.72	30.55	30.65	**31.23**	31.18
$\sigma = 35$	29.08	28.77	28.95	**29.58**	**29.58**
$\sigma = 45$	27.91	27.46	27.70	28.40	**28.45**
$\sigma = 55$	27.06	26.95	26.66	27.47	**27.58**
$\sigma = 70$	25.86	25.79	25.86	-	**26.57**

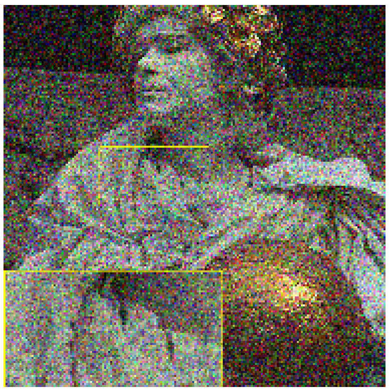
Kodak24 testset	NLDD	NLB	BM3D	DnCNN	FFDNet
$\sigma = 15$	33.93	33.84	34.19	34.48	**34.53**
$\sigma = 25$	31.65	31.41	31.59	32.03	**32.12**
$\sigma = 35$	30.04	29.64	29.98	30.46	**30.59**
$\sigma = 45$	28.87	28.35	28.76	29.32	**29.49**
$\sigma = 55$	28.02	27.89	27.78	28.39	**28.63**
$\sigma = 70$	26.86	26.76	26.91	-	**27.61**

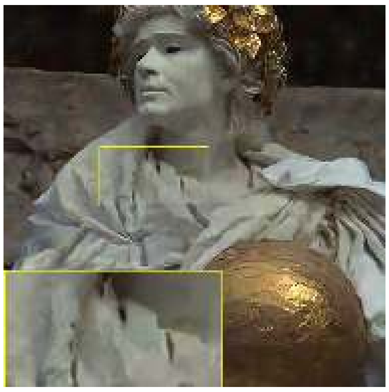
BSD68	NLDD	NLB	BM3D	TNRD	DnCNN	FFDNet
$\sigma = 15$	31.16	31.15	31.09	31.41	**31.61**	31.60
$\sigma = 25$	28.73	28.69	28.61	28.91	29.16	**29.19**
$\sigma = 35$	27.28	27.19	27.11	-	27.68	**27.73**
$\sigma = 45$	26.20	26.11	26.09	-	26.65	**26.72**
$\sigma = 55$	25.39	25.28	25.33	-	25.82	**25.95**
$\sigma = 70$	24.49	24.32	24.45	-	-	**25.06**

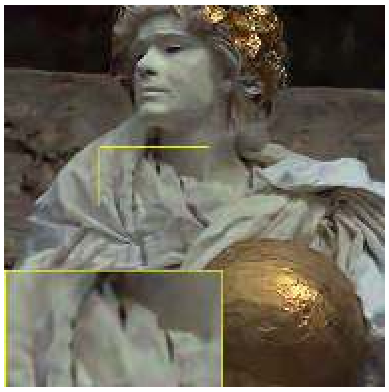
Set12	NLDD	NLB	BM3D	TNRD	DnCNN	FFDNet
$\sigma = 15$	32.31	32.25	32.29	32.49	32.68	**32.73**
$\sigma = 25$	29.99	29.90	29.90	30.03	30.36	**30.46**
$\sigma = 35$	28.49	28.30	28.31	-	28.83	**28.94**
$\sigma = 45$	27.35	27.12	27.13	-	27.69	**27.83**
$\sigma = 55$	26.41	26.14	26.24	-	26.71	**26.93**
$\sigma = 70$	25.20	24.90	25.13	-	-	**25.82**

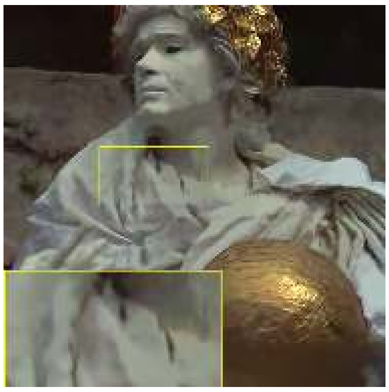
Table 7: *Comparison of $PSNR$(dB) for different state-of-the-art denoising algorithms.* CBSD68 and Kodak24 (top) are color testsets, while BSD68 and Set12 (bottom) are grayscale testsets.

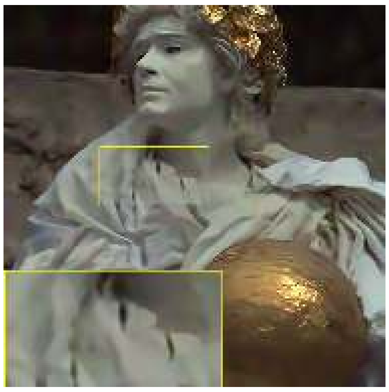

(a) Clean image


(b) Noisy image, $\sigma_{noise} = 55$


(c) DnCNN, PSNR 28.82dB


(d) BM3D, PSNR 28.21dB


(e) NLB, PSNR 28.26dB


(f) NLDD, PSNR 28.56dB

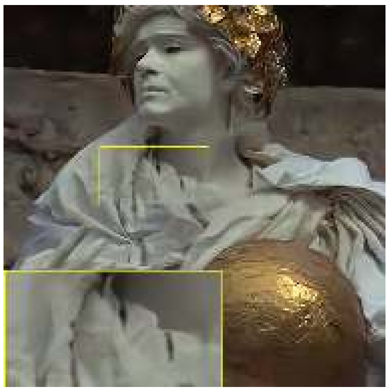
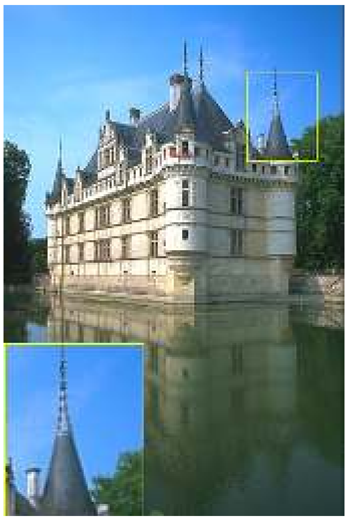
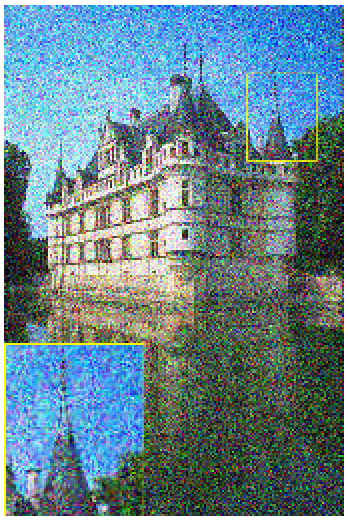
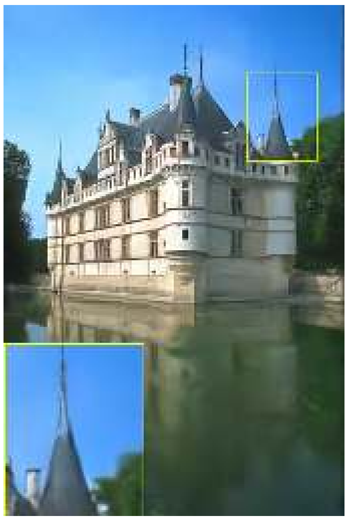
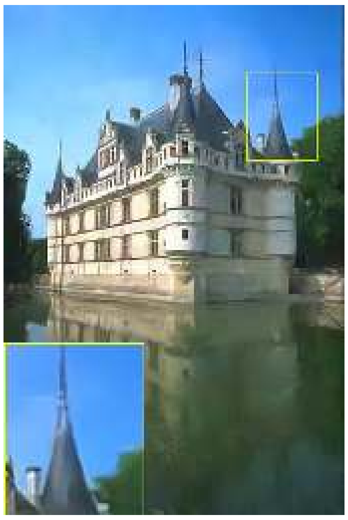

(g) FFDNet, PSNR 29.08dB

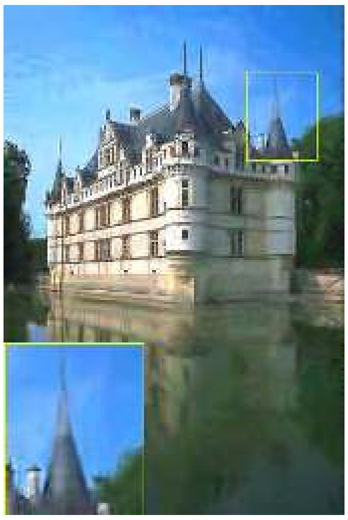
Figure 12: *Comparison of color denoising results.*

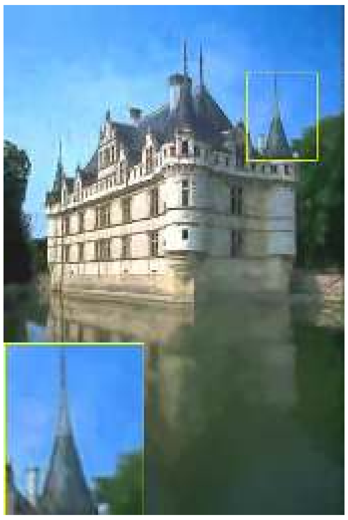

(a) Clean image


(b) Noisy image, $\sigma_{noise} = 55$


(c) FFDNet, PSNR 28.52dB


(d) DnCNN, PSNR 28.32dB


(e) BM3D, PSNR 27.31dB


(f) NLB, PSNR 27.79dB

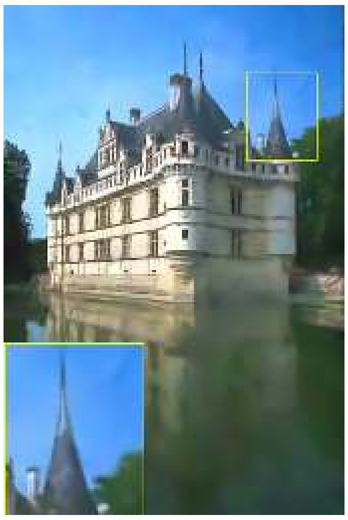
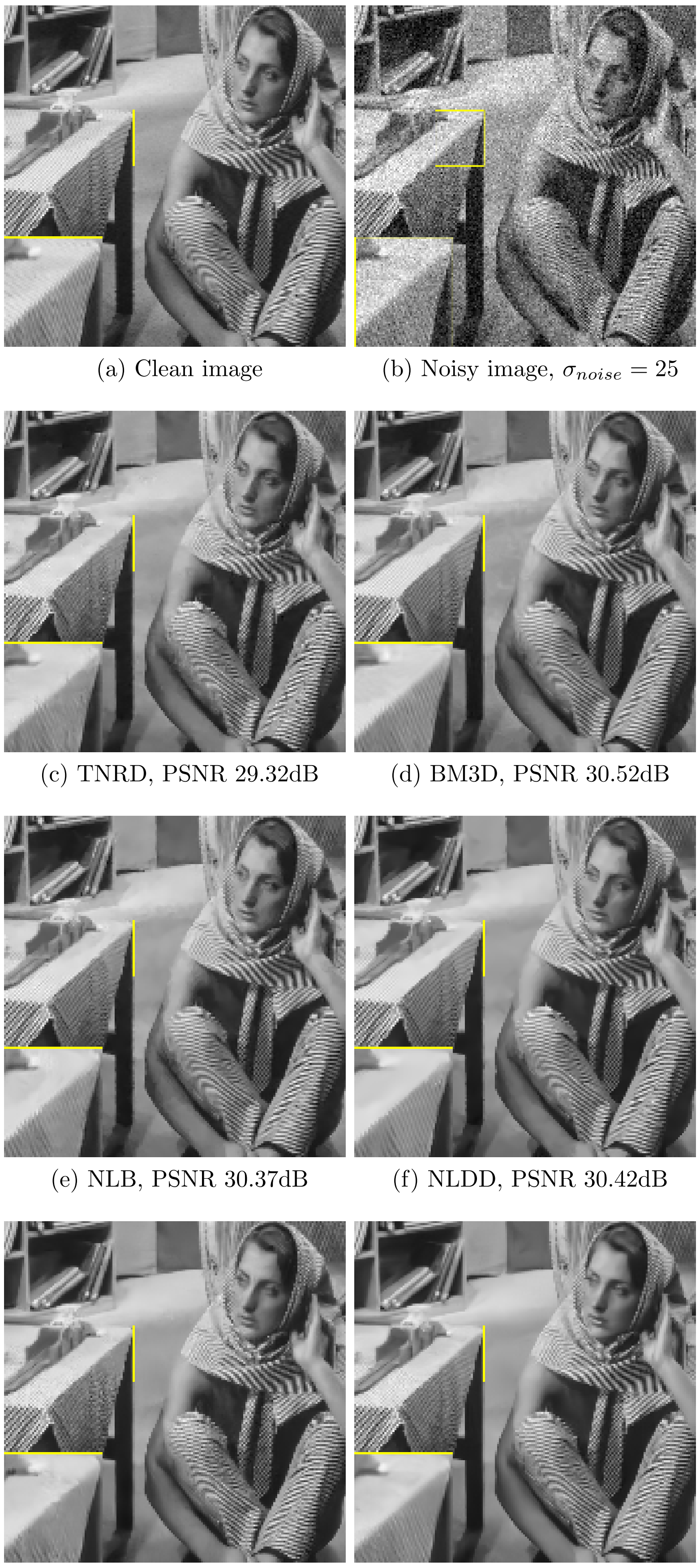

(g) NLDD, PSNR 27.92dB

Figure 13: *Comparison of color denoising results.*



(a) Clean image

(b) Noisy image, $\sigma_{noise} = 25$

(c) TNRD, PSNR 29.32dB

(d) BM3D, PSNR 30.52dB

(e) NLB, PSNR 30.37dB

(f) NLDD, PSNR 30.42dB

(g) DnCNN, PSNR 29.69dB

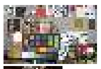
(h) FFDNet, PSNR 30.05dB

Figure 14: *Comparison of grayscale denoising results.*

8 Conclusions

In this paper, we analyzed FFDNet, a CNN denoising algorithm. An open-source implementation and online demo of this algorithm are available on IPOL. Its design characteristics, along with its differences with its predecessor DnCNN, were discussed. Compared to the latter, FFDNet is faster, more effective, and more versatile. These improvements are attained thanks to the use of diverse techniques, which were discussed in detail. The most salient of such techniques is the denoising performed on the downscaled sub-images, which yields a remarkable reduction of running times and memory footprint without sacrificing performance. This fact also accounts for doubling the receptive field without the need of increasing the depth of the network. The proposed downscaling layer is equivalent to a convolutional layer with filters of double spatial size and stride equal to 2. Lastly, we observed that the orthogonal regularization during training leads to a modest increase in performance.

Image Credits

 GoPro

Kodak Image Suite

Standard test image

BSD dataset

References

[1] F.J. Anscombe, *The transformation of Poisson, binomial and negative-binomial data*, Biometrika, 35 (1948), pp. 246–254. https://dx.doi.org/10.2307/2332343.

[2] H.C. Burger, C.J. Schuler, and S. Harmeling, *Image denoising: Can plain neural networks compete with BM3D?*, Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), (2012), pp. 2392–2399. https://dx.doi.org/10.1109/CVPR.2012.6247952.

[3] C. Chen, Q. Chen, J. Xu, and V. Koltun, *Learning to See in the Dark*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018. http://arxiv.org/abs/1805.01934.

[4] Y. Chen and T. Pock, *Trainable Nonlinear Reaction Diffusion: A Flexible Framework for Fast and Effective Image Restoration*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 39 (2017), pp. 1256–1272. https://dx.doi.org/10.1109/TPAMI.2016.2596743.

[5] K. Dabov, A. Foi, and V. Katkovnik, *Image denoising by sparse 3D transformation-domain collaborative filtering*, IEEE Transactions on Image Processing, 16 (2007), pp. 1–16. https://dx.doi.org/10.1109/TIP.2007.901238.

[6] A. Foi, M. Trimeche, V. Katkovnik, and K. Egiazarian, *Practical Poissonian-Gaussian noise modeling and fitting for single-image raw-data*, IEEE Transactions on Image Processing, 17 (2008), pp. 1737–1754. https://dx.doi.org/10.1109/TIP.2008.2001399.

[7] M. Gharbi, G. Chaurasia, S. Paris, and F. Durand, *Deep joint demosaicking and denoising*, ACM Transactions on Graphics, 35 (2016), pp. 1–12. https://dx.doi.org/10.1145/2980179.2982399.

[8] K. He, X. Zhang, S. Ren, and J. Sun, *Deep Residual Learning for Image Recognition*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770–778. https://dx.doi.org/10.1109/CVPR.2016.90.

[9] S. Ioffe and C. Szegedy, *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, in Proceedings of the 32nd International Conference on Machine Learning, JMLR.org, 2015, pp. 448–456. http://arxiv.org/abs/1502.03167.

[10] D.P. Kingma and J.L. Ba, *ADAM: a Method for Stochastic Optimization*, Proceedings of the International Conference on Learning Representations 2015, (2015), pp. 1–15. https://arxiv.org/abs/1412.6980.

[11] A. Krizhevsky, I. Sutskever, and G.E. Hinton, *ImageNet Classification with Deep Convolutional Neural Networks*, Advances In Neural Information Processing Systems, (2012), pp. 1–9. http://dl.acm.org/citation.cfm?id=2999134.2999257.

[12] M. Lebrun, *An Analysis and Implementation of the BM3D Image Denoising Method*, Image Processing On Line, 2 (2012), pp. 175–213. https://dx.doi.org/10.5201/ipol.2012.l-bm3d.

[13] M. Lebrun, A. Buades, and J-M. Morel, *Implementation of the Non-Local Bayes (NL-Bayes) Image Denoising Algorithm*, Imae Processing On Line, 3 (2013), pp. 1–42. https://dx.doi.org/10.5201/ipol.2013.16.

[14] K. Ma, Z. Duanmu, Q. Wu, Z. Wang, H. Yong, H. Li, and L. Zhang, *Waterloo Exploration Database: New Challenges for Image Quality Assessment Models*, IEEE Transactions on Image Processing, 26 (2017), pp. 1004–1016. https://dx.doi.org/10.1109/TIP.2016.2631888.

[15] M. Makitalo and A. Foi, *A closed-form approximation of the exact unbiased inverse of the Anscombe variance-stabilizing transformation*, IEEE Transactions on Image Processing, 20 (2011), pp. 2697–2698. https://dx.doi.org/10.1109/TIP.2011.2121085.

[16] ——, *Optimal inversion of the Anscombe transformation in low-count Poisson image denoising*, IEEE Transactions on Image Processing, 20 (2011), pp. 99–109. https://dx.doi.org/10.1109/TIP.2012.2202675.

[17] D. Martin, C. Fowlkes, D. Tal, and J. Malik, *A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics*, Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2 (2001), pp. 416–423. https://dx.doi.org/10.1109/ICCV.2001.937655.

[18] A. Paszke, G. Chanan, Z. Lin, S. Gross, E. Yang, L. Antiga, and Z. Devito, *Automatic differentiation in PyTorch*, Advances in Neural Information Processing Systems 30, (2017), pp. 1–4. https://openreview.net/forum?id=BJJsrmfCZ.

[19] N. Pierazzo, M. Lebrun, M. E. Rais, J. M. Morel, and G. Facciolo, *Non-local dual image denoising*, Proceedings of the IEEE International Conference on Image Processing (ICIP), (2014), pp. 813–817. https://dx.doi.org/10.1109/ICIP.2014.7025163.

[20] T. Remez, O. Litany, R. Giryes, and A.M. Bronstein, *Deep Class Aware Denoising*, in Proceedings of the International Conference on Sampling Theory and Applications (SampTA), IEEE, 2017. https://dx.doi.org/10.1109/SAMPTA.2017.8024474.

[21] V. Santhanam, V.I. Morariu, and L.S. Davis, *Generalized Deep Image to Image Regression*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016. https://dx.doi.org/10.1109/CVPR.2017.573.

[22] U. Schmidt and S. Roth, *Shrinkage fields for effective image restoration*, Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), (2014), pp. 2774–2781. https://dx.doi.org/10.1109/CVPR.2014.349.

[23] E. Schwartz, R. Giryes, and A.M. Bronstein, *DeepISP: Learning End-to-End Image Processing Pipeline*, (2018). http://arxiv.org/abs/1801.06724.

[24] W. Shi, J. Caballero, F. Huszar, J. Totz, A.P. Aitken, R. Bishop, D. Rueckert, and Z. Wang, *Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 1874–1883. https://dx.doi.org/10.1109/CVPR.2016.207.

[25] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, *Rethinking the Inception Architecture for Computer Vision*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), (2015), pp. 2818–2826. http://doi.org/10.1109/CVPR.2016.308.

[26] Dustin Tran, Alp Kucukelbir, Adji B. Dieng, Maja Rudolph, Dawen Liang, and David M. Blei, *Edward: A library for probabilistic modeling, inference, and criticism*, (2016).

[27] E. Vorontsov, C. Trabelsi, S. Kadoury, and C. Pal, *On orthogonality and learning recurrent networks with long term dependencies*, (2017). http://arxiv.org/abs/1702.00071.

[28] Y-Q. Wang, *Small neural networks can denoise image textures well : a useful complement to BM3D Image denoising neural networks*, Image Processing On Line, 6 (2016), pp. 1–7. https://doi.org/10.5201/ipol.2016.150.

[29] A.C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht, *The marginal value of adaptive gradient methods in machine learning*, in Advances in Neural Information Processing Systems, 2017, pp. 4148–4158. https://arxiv.org/abs/1705.08292.

[30] Y. Wu, M. Schuster, Z. Chen, Q.V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Ł. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*, (2016).

[31] F. Yu and V. Koltun, *Multi-Scale Context Aggregation by Dilated Convolutions*, in Proceedings of the International Conference on Learning Representations, 2016. http://arxiv.org/abs/1511.07122.

[32] F. Yu, V. Koltun, and T. Funkhouser, *Dilated residual networks*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), vol. 2017-Janua, 2017, pp. 636–644. https://dx.doi.org/10.1109/CVPR.2017.75.

[33] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, *Beyond a Gaussian denoiser: Residual learning of deep CNN for image denoising*, IEEE Transactions on Image Processing, 26 (2017), pp. 3142–3155. https://dx.doi.org/10.1109/TIP.2017.2662206.

[34] K. Zhang, W. Zuo, and L. Zhang, *FFDNet: Toward a Fast and Flexible Solution for CNN based Image Denoising*, IEEE Transactions on Image Processing, 27 (2018), pp. 4608–4622. http://doi.org/10.1109/TIP.2018.2839891.

[35] H. Zhao, O. Gallo, I. Frosio, and J. Kautz, *Loss Functions for Image Restoration with Neural Networks*, IEEE Transactions on Computational Imaging, 3 (2017), pp. 47–57. https://doi.org/10.1109/TCI.2016.2644865.