Image denoising: Can plain Neural Networks compete with BM3D?

Harold C. Burger, Christian J. Schuler, and Stefan Harmeling
Max Planck Institute for Intelligent Systems, Tübingen, Germany
http://people.tuebingen.mpg.de/burger/neural_denoising/

Abstract

Image denoising can be described as the problem of mapping from a noisy image to a noise-free image. The best currently available denoising methods approximate this mapping with cleverly engineered algorithms. In this work we attempt to learn this mapping directly with a plain multi layer perceptron (MLP) applied to image patches. While this has been done before, we will show that by training on large image databases we are able to compete with the current state-of-the-art image denoising methods. Furthermore, our approach is easily adapted to less extensively studied types of noise (by merely exchanging the training data), for which we achieve excellent results as well.

1. Introduction

An image denoising procedure takes a noisy image as input and outputs an image where the noise has been reduced. Numerous and diverse approaches exists: Some selectively smooth parts of a noisy image [25, 26]. Other methods rely on the careful shrinkage of wavelet coefficients [24, 18]. A conceptually similar approach is to denoise image patches by trying to approximate noisy patches using a sparse linear combination of elements of a learned dictionary [1, 4]. Learning a dictionary is sometimes accomplished through learning on a noise-free dataset. Other methods also learn a global image prior on a noise-free dataset, for instance [20, 27, 9]. More recent approaches exploit the “non-local” statistics of images: Different patches in the same image are often similar in appearance [3, 13, 2]. This last class of algorithms and in particular BM3D [3] represent the current state-of-the-art in natural image denoising.

While BM3D is a well-engineered algorithm, could we also *automatically learn* an image denoising procedure purely from training examples consisting of pairs of noisy and noise-free patches? This paper will show that it is indeed possible to achieve state-of-the-art denoising performance with a plain multi layer perceptron (MLP) that maps noisy patches onto noise-free ones.

This is possible because the following factors are combined:

- The capacity of the MLP is chosen large enough, i.e. it consists of enough hidden layers with sufficiently many hidden units.
- The patch size is chosen large enough, i.e. a patch contains enough information to recover a noise-free version. This is in agreement with previous findings [12].
- The chosen training set is large enough. Training examples are generated on the fly by corrupting noise-free patches with noise.

Training high capacity MLPs with large training sets is feasible using modern Graphics Processing Units (GPUs).

Contributions: We present a patch-based denoising algorithm that is *learned* on a large dataset with a plain neural network. Results on additive white Gaussian (AWG) noise are competitive with the current state of the art. The approach is equally valid for other types of noise that have not been as extensively studied as AWG noise.

2. Related work

Neural networks have already been used to denoise images [9]. The networks commonly used are of a special type, known as *convolutional neural networks* (CNNs) [10], which have been shown to be effective for various tasks such as hand-written digit and traffic sign recognition [23]. CNNs exhibit a structure (local receptive fields) specifically designed for image data. This allows for a reduction of the number of parameters compared to plain multi layer perceptrons while still providing good results. This is useful when the amount of training data is small. On the other hand, multi layer perceptrons are potentially more powerful than CNNs: MLPs can be thought of as universal function approximators [8], whereas CNNs restrict the class of possible learned functions.

A different kind of neural network with a special architecture (i.e. containing a *sparsifying logistic*) is used in [19] to denoise image patches. A small training set is used. Results are reported for strong levels of noise. It has also been attempted to denoise images by applying multi layer perceptrons on wavelet coefficients [28]. The use of wavelet bases can be seen as an attempt to incorporate prior knowledge about images.

Differences to this work: Most methods we have described make assumptions about natural images. Instead we do not explicitly impose such assumptions, but rather propose a pure *learning* approach.

3. Multi layer perceptrons (MLPs)

A *multi layer perceptron* (MLP) is a nonlinear function that maps vector-valued input via several hidden layers to vector-valued output. For instance, an MLP with two hidden layers can be written as,

$$f(x) = b_3 + W_3 \tanh(b_2 + W_2 \tanh(b_1 + W_1 x)). \quad (1)$$

The weight matrices W_1, W_2, W_3 and vector-valued biases b_1, b_2, b_3 parameterize the MLP, the function $\tanh$ operates component-wise. The *architecture* of an MLP is defined by the number of hidden layers and by the layer sizes. For instance, a (256,2000,1000,10)-MLP has two hidden layers. The input layer is 256-dimensional, i.e. $x \in \Re^{256}$. The vector $v_1 = \tanh(b_1 + W_1 x)$ of the first hidden layer is 2000-dimensional, the vector $v_2 = \tanh(b_2 + W_2 v_1)$ of the second hidden layer is 1000-dimensional, and the vector $f(x)$ of the output layer is 10-dimensional. Commonly, an MLP is also called *feed-forward neural network*.

3.1. Training MLPs for image denoising

The idea is to learn an MLP that maps noisy image patches onto clean image patches where the noise is reduced or even removed. The parameters of the MLP are estimated by training on pairs of noisy and clean image patches using *stochastic gradient descent* [11].

More precisely, we randomly pick a clean patch y from an image dataset and generate a corresponding noisy patch x by corrupting y with noise, for instance with additive white Gaussian (AWG) noise. The MLP parameters are then updated by the *backpropagation* algorithm [21] minimizing the quadratic error between the mapped noisy patch $f(x)$ and the clean patch y, i.e. minimizing $(f(x) - y)^2$.

To make backpropagation more efficient, we apply various common neural network tricks [11]:

1. Data normalization: The pixel values are transformed to have approximately mean zero and variance close to one. More precisely, assuming pixel values between 0 and 1, we subtract 0.5 and divide by 0.2.

2. Weight initialization: The weights are sampled from a normal distribution with mean 0 and standard deviation $\sigma = \sqrt{N}$, where N is the number of input units of the corresponding layer. Combined with the first trick, this ensures that both the linear and the non-linear parts of the sigmoid function are reached.

3. Learning rate division: In each layer, we divide the learning rate by N, the number of input units of that layer. This allows us to change the number of hidden units without modifying the learning rate.

The basic learning rate was set to 0.1 for all experiments. No regularization was applied on the weights.

3.2. Applying MLPs for image denoising

To denoise images, we decompose a given noisy image into overlapping patches and denoise each patch x separately. The denoised image is obtained by placing the denoised patches $f(x)$ at the locations of their noisy counterparts, then averaging on the overlapping regions. We found that we could improve results slightly by weighting the denoised patches with a Gaussian window. Also, instead of using all possible overlapping patches (stride size 1, i.e. patch offset 1), we found that results were almost equally good by using every third sliding-window patch (stride size 3), while decreasing computation time by a factor of 9. Using a stride size of 3, we were able to denoise images of size 512×512 pixels in approximately one minute on a modern CPU, which is slower than BM3D [3], but much faster than KSVD [1].

3.3. Implementation

The computationally most intensive operations in an MLP are the matrix-vector multiplications. *Graphics Processing Units* (GPUs) are better suited for these operations than *Central Processing Units* (CPUs). For this reason we implemented our MLP on a GPU. We used nVidia's C2050 GPU and achieved a speed-up factor of more than one order of magnitude compared to an implementation on a quad-core CPU. This speed-up is a crucial factor, allowing us to run much larger-scale experiments.

4. Experimental setup

We performed all our experiments on gray-scale images. These were obtained from color images with MATLAB's `rbg2gray` function. Since it is unlikely that two noise samples are identical, the amount of training data is effectively infinite, no matter which dataset is used. However, the number of uncorrupted patches is restricted by the size of the dataset.

Training data: For our experiments, we define two training sets:

Small training set: The Berkeley segmentation dataset [15], containing 200 images, and

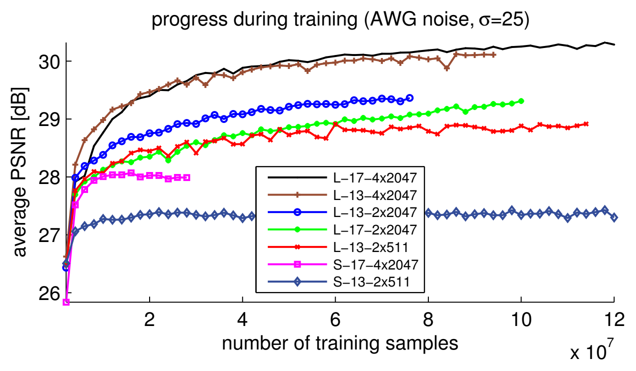

Figure 1. Improving average PSNR on the images "Barbara" and "Lena" while training.

Large training set: The union of the LabelMe dataset [22] (containing approximately $150,000$ images) and the Berkeley segmentation dataset.

Some images in the LabelMe dataset appeared a little noisy or a little blurry, so we downscaled the images in that dataset by a factor of 2 using MATLAB's `imresize` function with default parameters.

Test data: We define three different test sets to evaluate our approach:

Standard test images: This set of 11 images contains standard images, such as "Lena" and "Barbara", that have been used to evaluate other denoising algorithms [3],

Pascal VOC 2007: We randomly selected 500 images from the Pascal VOC 2007 test set [5], and

McGill: We randomly selected 500 images from the McGill dataset [17].

5. Results

We first study how denoising performance depends on the MLP architecture and the number of training examples. Then we compare against BM3D and other existing algorithms, and finally we show how MLPs perform on other types of noise.

5.1. More training data and more capacity is better

We train networks with different architectures and patch sizes. We write for instance L–17–4x2047 for a network that is trained on the large training set with patch size 17×17 and 4 hidden layers of size 2047; similarly S–13–2x511 for a network that is trained on the small training set with patch size 13×13 and 2 hidden layers of size 511. Other architectures are denoted in the legend of Figure 1. All these MLPs are trained on image patches that have been corrupted with Gaussian noise with $\sigma = 25$.

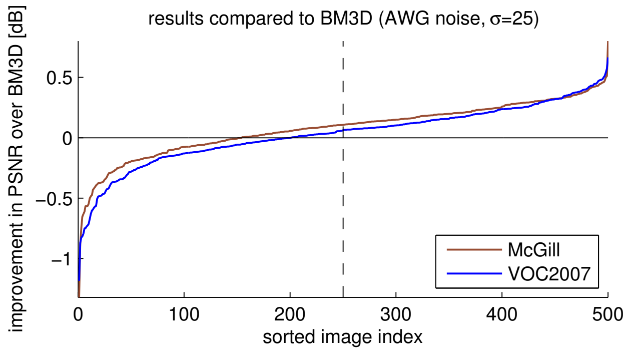

Figure 3. Performance profile of our method on two datasets of 500 test images compared to BM3D.

To monitor the performance of the network we test the different networks after every two million training examples on two test images "Barbara" and "Lena" that have been corrupted with Gaussian noise with standard deviation $\sigma = 25$. Figure 1 shows the improving PSNR of the networks on the two test images.

Observations: Many training examples are needed to achieve good results. Progress is steady for the first 40 million training samples. After that, the PSNR on the test images still improves, albeit more slowly. Overfitting never seems to be an issue. Better results can be achieved with patches of size 17×17 compared to patches of size 13×13. Also, more complex networks lead to better results. Switching from the small training set (Berkeley) to the large training set (LabeleMe + Berkeley) improves the results enormously. We note that most attempts to learn image statistics using a training dataset have used only the Berkeley segmentation dataset [20, 9, 19].

5.2. Can MLPs compete with BM3D?

In the previous section, the MLP L–17–4x2047 with four hidden layers of size 2047 and a patch size of 17×17 trained on the large training set achieved the best results. We trained this MLP on a total of 362 million training samples, requiring approximately one month of computation time on a GPU. In the following, we compare its results achieved on the test data with other denoising methods, including BM3D [3].

Pascal VOC 2007, McGill: Figure 3 compares our method with BM3D on PASCAL VOC 2007 and McGill. To reduce computation time during denoising, we used a patch offset (stride size) of 3. On average, our results are equally good on the PASCAL VOC 2007 (better by 0.03dB) and on the McGill dataset (better by 0.08dB).

More precisely, our MLP outperforms BM3D on exactly 300 of the 500 images of the PASCAL VOC 2007 images, see Figure 3. Similarly, our MLP is better on 347 of the

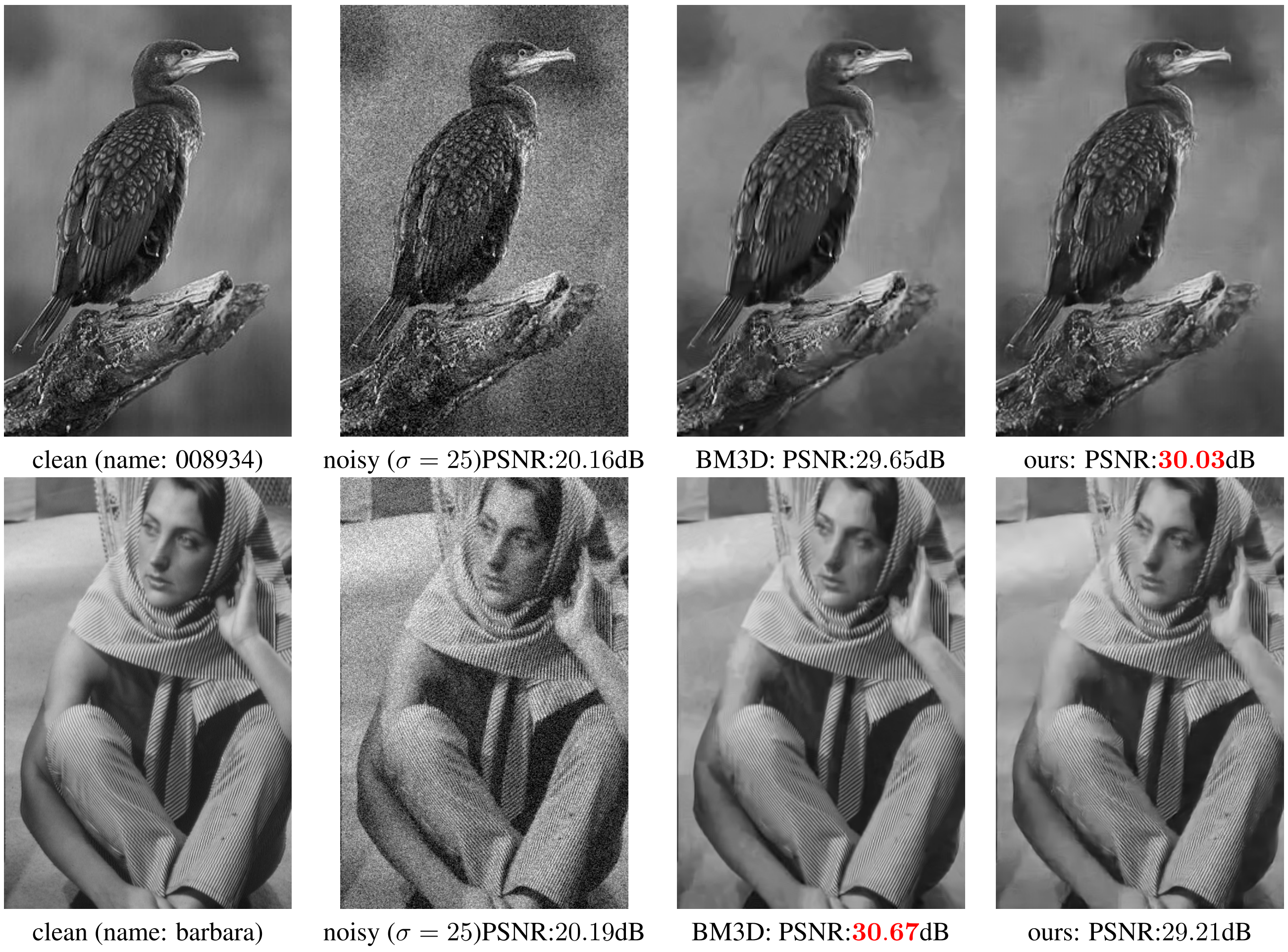

Figure 2. Our results compared to BM3D. Our method outperforms BM3D on some images (top row). On other images however, BM3D achieves much better results than our approach. The images on which BM3D is much better than our approach usually contain some kind of regular structure, such as the stripes on Barbara's pants (bottom row).

image	GSM [18]	KSVD [1]	BM3D [3]	us
Barbara	27.83dB	*29.49dB*	**30.67dB**	29.21dB
Boat	29.29dB	29.24dB	*29.86dB*	**29.89dB**
C.man	28.64dB	28.64dB	**29.40dB**	*29.32dB*
Couple	28.94dB	28.87dB	*29.68dB*	**29.70dB**
F.print	27.13dB	27.24dB	**27.72dB**	*27.50dB*
Hill	29.26dB	29.20dB	*29.81dB*	**29.82dB**
House	31.60dB	32.08dB	**32.92dB**	*32.50dB*
Lena	31.25dB	31.30dB	*32.04dB*	**32.12dB**
Man	29.16dB	29.08dB	*29.58dB*	**29.81dB**
Montage	30.73dB	30.91dB	**32.24dB**	*31.85dB*
Peppers	29.49dB	29.69dB	*30.18dB*	**30.25dB**

Table 1. PSNRs (in dB) on standard test images, $\sigma = 25$.

500 images of the McGill images. Our best improvement over BM3D is 0.81dB on image "pippin0120"; BM3D is better by 1.32dB on image "merry mexico0152", both in the McGill dataset.

Standard test images: We also compare our MLP (with a stride size of 1) to BM3D on the set of standard test images, see Table 1. For BM3D, we report the average results for 105 different noisy instances of the same test image. Due to longer running times, we used only 17 different noisy instances for our approach. We outperform BM3D on 6 of the 11 test images. BM3D has a clear advantage on images with regular structures, such as the pants of Barbara. We do outperform KSVD [1] on every image except Barbara. KSVD is a dictionary-based denoising algorithm that learns a dictionary that is adapted to the noisy image at hand. Images with a lot of repeating structure are ideal for both BM3D and KSVD. We see that the neural network is able to compete with BM3D.

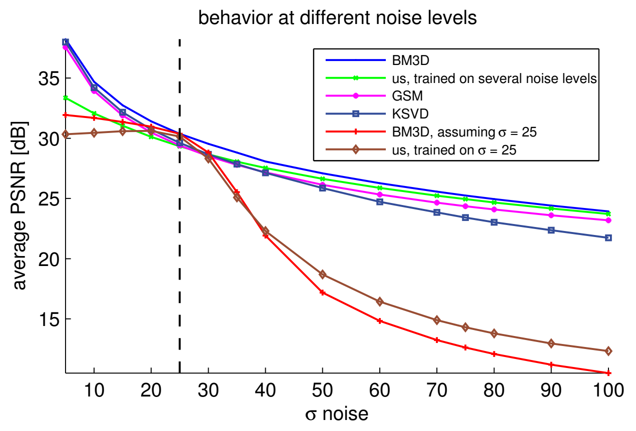

Figure 4. Comparison on images with various noise levels: the MLP trained for $\sigma = 25$ is competitive for $\sigma = 25$. The MLP trained on several noise levels is also competitive on higher noise levels.

5.3. Robustness at other noise levels

The MLP from the previous section was trained solely on image patches that were corrupted with AWG noise with $\sigma = 25$. Is it able to handle other noise levels (σ smaller or larger than 25) as well? To answer this question, we applied it to the 11 standard test images that were corrupted with AWG noise with different values of σ. Figure 4 shows a comparison against results achieved by GSM, KSVD and BM3D. We see that for $\sigma = 25$ our MLP (brown line) is competitive, but deteriorates for other noise levels. While our MLP does not know that the level of the noise has changed, the other methods were provided with that information. To study this effect we also run BM3D for the different noise levels but fixing its input parameter to $\sigma = 25$ (red curve). We see a similar behavior to our method. Our MLP generalizes even slightly better to higher noise levels (brown above red).

5.4. MLPs trained on several noise levels

To overcome the limitations of an MLP trained on examples from a single noise level, we attempted to train a network on image patches corrupted by noise with different noise levels. We used the same architecture as our network trained on $\sigma = 25$. The amount of noise of a given training patch (i.e. the value of σ) was given as additional input to the network. This was done in two ways: One additional input unit provided the value of σ directly; 15 additional input units worked as switches with all units set to -1 except for the one unit coding the corresponding value of σ. Training proceeded as previously and σ was chosen randomly in steps of 5 between 0 and 105. We tested this network on 11 standard test images for different values of σ, see green line in Figure 4.

Even though we outperform BM3D on none of the noise levels, we do perform better than both GSM and KSVD at high noise levels. At low noise levels ($\sigma = 5$) our denoising results are worse than the noisy input. We draw the following conclusions: Denoising at several noise levels is more difficult than denoising at a single noise level. Hence, a network with more capacity (i.e. parameters) should be used. The fact that the network performs better at high noise levels is presumable due to the fact that noisier patches provide stronger gradients. The higher noise levels therefore dominate the training procedure. A potential solution might be to adapt the learning rate to the value of σ.

5.5. Learning to remove arbitrary noise types

Virtually all denoising algorithms assume the noise to be AWG. However, images are not always corrupted by AWG noise. Noise is not necessarily additive, white, Gaussian and signal independent. For instance in some situations, the imaging process is corrupted by Poisson noise (such as photon shot noise). Denoising algorithms which assume AWG noise might be applied to such images using some image transform [14]. Similarly, Rice-distributed noise, which occurs in magnetic resonance imaging, can be handled [6].

In most cases however, it is more difficult or even impossible to find Gaussianizing transforms. In such cases, a possible solution is to create a denoising algorithm specifically designed for that noise type. MLPs allow us to effectively learn a denoising algorithm for a given noise type, provided that noise can be simulated. In the following , we present results on three noise types that are different from AWG noise.We make no effort to adapt our architecture or procedure in general to the specific noise type but rather use the architecture that yielded the best results for AWG noise (four hidden layers of size 2047 and patches of size 17×17).

Stripe noise: It is often assumed that image data contains structure, whereas the noise is uncorrelated and therefore unstructured. In cases where the noise also exhibits structure, this assumption is violated and denoising results become poor. We here show an example where the noise is additive and Gaussian, but where 8 horizontally adjacent noise values have the same value.

Since there is no canonical denoising algorithm for this noise, we choose BM3D as the competitor. An MLP trained on 58 million training examples outperformed BM3D for this type of noise, see left column of Figure 5.

Salt and pepper noise: When the noise is additive Gaussian, the noisy image value is still correlated to the original image value. With salt and pepper noise, noisy values are not correlated with the original image data. Each pixel has a probability p of being corrupted. A corrupted pixel has probability 0.5 of being set to 0; otherwise, it is set to high-

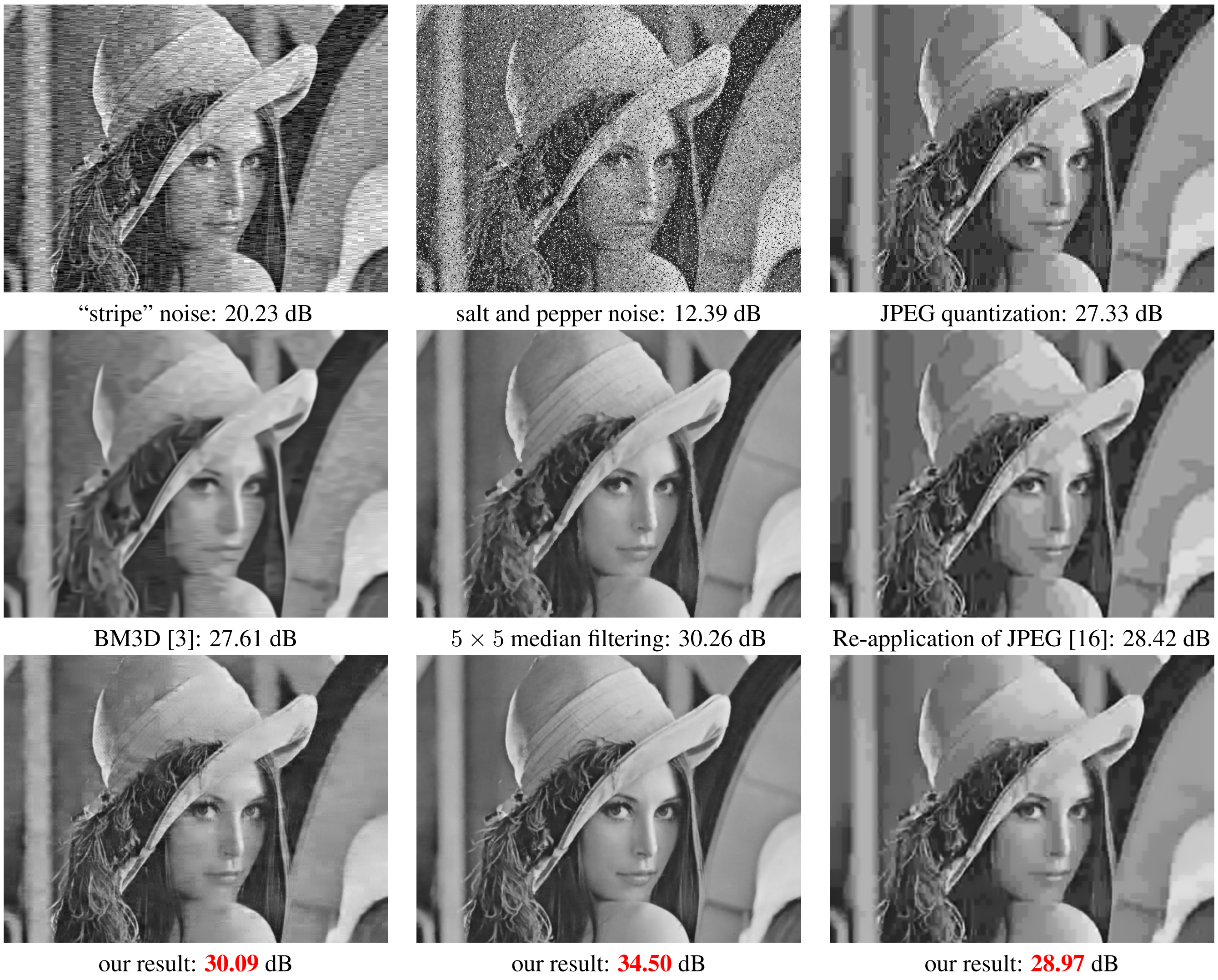
"stripe" noise: 20.23 dB
salt and pepper noise: 12.39 dB
JPEG quantization: 27.33 dB
BM3D [3]: 27.61 dB
5 × 5 median filtering: 30.26 dB
Re-application of JPEG [16]: 28.42 dB
our result: **30.09** dB
our result: **34.50** dB
our result: **28.97** dB

Figure 5. Comparison of our method to others on different kinds of noise. The comparison with BM3D is unfair.

est possible value (255 for 8-bit images). We show results with $p = 0.2$.

A common algorithm for removing salt and pepper noise is median filtering. We achieved the best results with a filter size of 5×5 and symmetrically extended image boundaries. We also experimented with BM3D (by varying the value of σ) and achieved a PSNR of 25.55dB. An MLP trained on 50 million training examples outperforms both methods, see middle column of Figure 5.

JPEG quantization artifacts: Such artifacts occur due to the JPEG image compression algorithm. The quantization process removes information, therefore introducing noise. Characteristics of JPEG noise are blocky images and loss of edge clarity. This kind of noise is not random, but rather completely determined by the input image. In our experiments we use JPEG's quality setting $Q = 5$, creating visible artifacts.

A common method to enhance JPEG-compressed images is to shift the images, re-apply JPEG compression, shift back and average [16] which we choose for comparison. BM3D achieves similar results on this task after parameter tweaking. An MLP trained on 12 million training examples with that noise, outperforms both methods, see right column of Figure 5.

6. Discussion

The learned weights applied to the input layer and the weights that calculate the output layer can be visualized as patches, see Figures. 7 and 6.

The latter patches in Figure 6 form a dictionary of the denoised patch since they are linearly combined with the scalars in the last hidden layers as weighting coefficients.

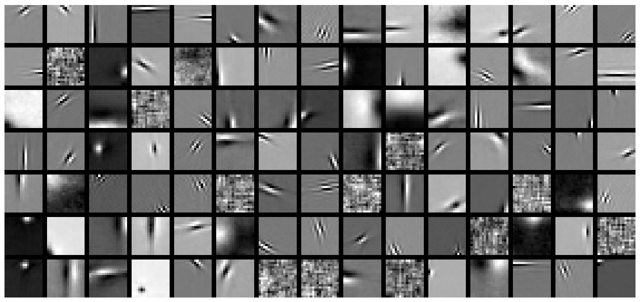

Figure 6. Random selection of weights in the output layer. Each patch represents the weights from one hidden neuron to the output pixels.

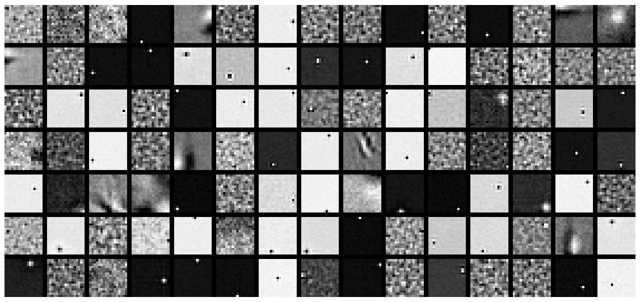

Figure 7. Random selection of weights in the input layer. Each patch represents the weights from the input pixels to one hidden neuron.

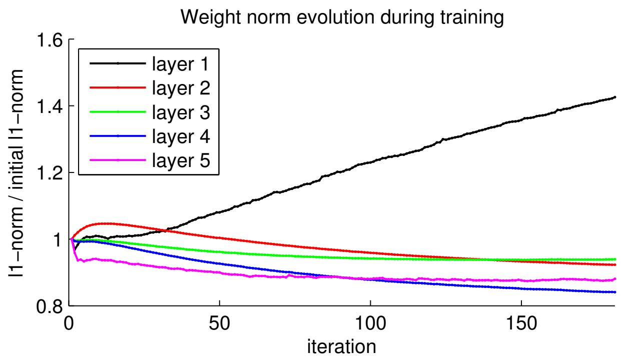

Figure 8. The ℓ_1-norm of the weights of some layers decreases during training (without any explicit regularization).

They can be categorized coarsely into four categories: 1) patches resembling Gabor filters, 2) blobs, 3) larger scale structures, and 4) "noisy" patches. The Gabor filters occur at different scales, shifts and orientations. Similar dictionaries have also been learned by other denoising approaches. It should be noted that MLPs are not shift-invariant, which explains why some patches are shifted versions of each other.

The weights connecting the noisy input pixels to one hidden neuron in the first hidden layer can also be represented as an image patch, see Figure 7. The patches can be interpreted as filters, with the activity of the hidden neuron connected to a patch corresponding to the filter's response to the input. These patches can be classified into three main categories: 1) patches that focus on just a small number of pixels, 2) patches focusing on larger regions and resembling Gabor filters, and 3) patches that look like random noise. These filters are able to extract useful features from noisy input data, but are more difficult to interpret than the output layer patches.

It is also interesting to observe the evolution of the ℓ_1-norm of the weights in the different layers during training, see Figure 8. One might be tempted to think of the evolution of the weights as following a random walk. In that case, the ℓ_1-norm should increase over time. However, we observe that in all layers but the first, the ℓ_1-norm decreases over time (after a short initial period where it increases). This happens in the absence of any explicit regularization on the weights and is an indication that such regularization is not necessary.

MLPs vs. Support Vector Regression: We use MLPs to solve a regression problem to learn a denoising method. An equally valid approach would have been to use a kernel approach such as *support vector regression* (SVR). For practical (rather than fundamental) reasons we preferred MLPs over SVR: (i) MLPs are easy to implement on a GPU since they are based on matrix-vector-multiplications. (ii) MLPs can easily be trained on very large datasets using stochastic gradient descent. However, we make no claim regarding the quality of results potentially achievable with SVR: It is entirely possible that SVR would yield still better results than our MLPs.

Is deep learning necessary? Training MLPs with many hidden layers can lead to problems such as vanishing gradients and over-fitting. To avoid these problems, new training procedures called *deep learning* that start with an unsupervised learning phase have been proposed [7]. Such an approach makes the most sense when labeled data is scarce but unlabeled data is plentiful and when the networks are too deep to be trained effectively with back-propagation. In our case, labeled data is plentiful and the networks contain no more than four hidden layers. We found back-propagation to work well and therefore concluded that deep learning techniques are not necessary, though it is possible that still better results are achievable with an unsupervised pre-training technique.

7. Conclusion

Neural networks can achieve state-of-art image denoising performance. For this, it is important that (i) the capacity of the network is large enough, (ii) the patch size is large enough, and (iii) the training set is large enough. These requirements can be fulfilled by implementing MLPs on GPUs that are ideally suited for the computations necessary to train and apply neural networks. Without the use of GPUs our computations could have easily taken a year of running time.

However, our most competitive MLP is tailored to a single level of noise and does not generalize well to other noise levels compared to other denoising methods. This is a serious limitation which we already tried to overcome with an MLP trained on several noise levels. However, the latter does not yet achieve the same performance for $\sigma = 25$ as the specialized MLP. Nonetheless, we believe that this will also be possible with a network with even higher capacity and sufficient training time.

References

[1] M. Aharon, M. Elad, and A. Bruckstein. K-svd: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on Signal Processing*, 54(11):4311–4322, 2006.

[2] A. Buades, C. Coll, and J. Morel. A review of image denoising algorithms, with a new one. *Multiscale Modeling and Simulation*, 4(2):490–530, 2005.

[3] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian. Image denoising by sparse 3-D transform-domain collaborative filtering. *IEEE Transactions on Image Processing*, 16(8):2080–2095, 2007.

[4] M. Elad and M. Aharon. Image denoising via sparse and redundant representations over learned dictionaries. *IEEE Transactions on Image Processing*, 15(12):3736–3745, 2006.

[5] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[6] A. Foi. Noise estimation and removal in mr imaging: The variance-stabilization approach. In *2011 IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, pages 1809–1814, 2011.

[7] G. Hinton, S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[8] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[9] V. Jain and H. Seung. Natural image denoising with convolutional networks. *Advances in Neural Information Processing Systems (NIPS)*, 21:769–776, 2008.

[10] Y. LeCun, L. Bottou, Y. Bengio, and H. P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[11] Y. LeCun, L. Bottou, G. Orr, and K. Müller. Efficient backprop. *Neural networks: Tricks of the trade*, pages 546–546, 1998.

[12] A. Levin and B. Nadler. Natural Image Denoising: Optimality and Inherent Bounds. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011.

[13] J. Mairal, F. Bach, J. Ponce, G. Sapiro, and A. Zisserman. Non-local sparse models for image restoration. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2272–2279, 2010.

[14] M. Mäkitalo and A. Foi. Optimal inversion of the anscombe transformation in low-count poisson image denoising. *IEEE Transactions on Image Processing*, 20:99–109, 2011.

[15] 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. In *Proc. 8th International Conference on Computer Vision (ICCV)*, volume 2, pages 416–423, July 2001.

[16] A. Nosratinia. Enhancement of jpeg-compressed images by re-application of jpeg. *The Journal of VLSI Signal Processing*, 27(1):69–79, 2001.

[17] A. Olmos et al. A biologically inspired algorithm for the recovery of shading and reflectance images. *Perception*, 33(12):1463, 2004.

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

[19] M. Ranzato, Y.-L. Boureau, S. Chopra, and Y. LeCun. A unified energy-based framework for unsupervised learning. In *Proc. Conference on AI and Statistics (AI-Stats)*, 2007.

[20] S. Roth and M. Black. Fields of experts. *International Journal of Computer Vision*, 82(2):205–229, 2009.

[21] D. Rumelhart, G. Hinton, and R. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[22] B. Russell, A. Torralba, K. Murphy, and W. Freeman. Labelme: a database and web-based tool for image annotation. *International Journal of Computer Vision*, 2007.

[23] P. Sermanet and Y. LeCun. Traffic Sign Recognition with Multi-Scale Convolutional Networks. In *Proceedings of International Joint Conference on Neural Networks (IJCNN)*, 2011.

[24] E. Simoncelli and E. Adelson. Noise removal via Bayesian wavelet coring. In *Proceedings of the International Conference on Image Processing*, pages 379–382, 1996.

[25] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Proceedings of the Sixth International Conference on Computer Vision*, pages 839–846, 1998.

[26] J. Weickert. *Anisotropic diffusion in image processing*. ECMI Series, Teubner-Verlag, Stuttgart, Germany, 1998.

[27] Y. Weiss and W. Freeman. What makes a good model of natural images? In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, 2007.

[28] S. Zhang and E. Salari. Image denoising using a neural network based non-linear filter in wavelet domain. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 2, pages ii–989, 2005.