©2016 Society for Imaging Science and Technology
DOI: 10.2352/ISSN.2470-1173.2016.15.IPAS-013

Intelligent Image Filtering using Multilayer Neural Network with Multi-Valued Neurons

Igor Aizenberg; Texas A&M University-Texarkana;Texarkana, TX, USA

Abstract

Image filtering, regardless of whether it is denoising (low pass filtering) or edge detection (high pass filtering) can be considered as a machine learning problem. In fact, filtering is a process of approximation of a desirable result. A filter is considered good, if it approaches this ideal result better than other filters. But any machine learning problem should be considered from exactly the same standpoint. In this paper, we suggest to use a multilayer neural network with multi-valued neurons (MLMVN) as an intelligent image filter. After MLMVN is trained using a number of n x n patches from different images to obtain a clean patch from a noisy one, it can be used as an intelligent filter. It is shown that this filter is robust, since it performs well on different images, which did not participate in the learning process. It terms of PSNR, this approach shows results comparable with other filters commonly recognized as good. A specific advantage of the presented approach is its ability to preserve small details carefully.

Introduction

There are numerous different filters and many of them have become classical. The most of filters should be considered as "engineered" algorithms where their designers know what they want to achieve and use well determined certain techniques to achieve their goals. Perhaps, BM3D filter [1] is the best example of such well-engineered filters (it is also commonly recognized as one of the most efficient nonlinear filters ever).

We would like to consider here the filtering problem from a different standpoint. Instead of designing a filter we will let a machine learning tool to learn what a filtering process is, and then perform this process accordingly. Image filtering, regardless of whether it is denoising (low pass filtering) or edge detection (high pass filtering) can certainly be considered as a machine learning problem. In fact, filtering is a process of approximation of a desirable result. A filter is considered good, if it approaches this ideal desirable result better than other filters. But any machine learning problem should be considered from exactly the same standpoint. Any machine learning tool learns a problem in terms of better approximation of a desirable result. In such a case, on the one hand, a machine learning tool remains a "black box" because the researcher does not know what exactly the tool has extracted from the learned data. But on the other hand, any machine learning tool performs as a sophisticated nonlinear interpolator/approximator. Learning from some representative data, it can then be used to interpolate (approximate) or predict other similar data, which did not participate in the learning process.

The use of machine learning in image filtering is not new. In this paper, we will employ the idea first suggested in [2] and then developed and presented in detail in [3], [4]. This idea is to filter an image using a neural network, applying filtering to all pixels of an *n* x *n* patch simultaneously and averaging then the results for overlapping parts of patches. In [2]-[4], it was proposed to use a classical multilayer feedforward neural network (the multilayer perceptron, or simply MLP) with 3-4 hidden layers as an intelligent filtering tool. Basically, the idea of filtering by patches was originally proposed in [1] and employed in BM3D filter. In [2]-[4], this idea was considered from the different standpoint. While in BM3D filter statistically similar patches are blocked in a group and then this group is filtered simultaneously, in the neural intelligent approach developed in [2]-[4], it was suggested to learn using a neural network how to create a clean patch from a noisy one using the large number of patches taken from many different images. Then a trained network is expected to filter patches from other images, which were not presented in a learning set.

In this paper, we will employ the same idea of intelligent image filtering using a neural network, which processes overlapping patches after it was trained using a representative learning set containing patches taken from different images. Hence our first task is to create a learning set from a number of *n* x *n* patches randomly taken from different images. Noisy patches will be used as inputs (therefore a network will have n^2 inputs) and the corresponding clean patches will be used as desired outputs (therefore the network will have n^2 outputs). Then we need to train a neural network. After a network is trained, it should be used for filtering an image through dividing it into *n* x *n* overlapping patches, filtering all pixels in each patch simultaneously and averaging then the results for overlapping parts of patches. We will have to find the optimal *n* (size of a patch), optimal topology of a neural network, optimal size of a learning set, optimal number of images used to create the learning set, and optimal offset for overlapping patches to ensure the best filtering results. Basically, we have to consider the same tasks, which were solved for MLP in [2]-[4]. However, we will use here a different type of a neural network. We suggest doing intelligent image filtering using a multilayer neural network with multi-valued neurons (MLMVN) [5]. This is also feedforward neural network with a standard feedforward topology (where the outputs of all neurons from the preceding layer are connected to the corresponding inputs of all neurons from the current layer). However this network consists of multi-valued neurons with complex-valued weights [6]. The latter determines important advantages of MLMVN over many other machine learning techniques. Being more functional, learning faster and generalizing better, MLMVN significantly outperforms MLP in many applications [5]-[7]. Hence it is very attractive to use MLMVN as an intelligent filter hoping that it should be able to outperform MLP when solving this problem.

As it will be shown below, MLMVN meets these expectations. The network just with a single hidden layer, after it is trained using a representative set of patches taken from different images, is able to filter an image, which has not participated in the learning process. This can be done without any significant smoothing of small details. Thus MLMVN demonstrates its robustness and shows results comparable with other filters commonly recognized as good.

©2016 Society for Imaging Science and Technology
DOI: 10.2352/ISSN.2470-1173.2016.15.IPAS-013

Multilayer Neural Network with Multivalued Neurons (MLMVN)

MLMVN consists of multi-valued neurons (MVN) with complex-valued weights, and this is its main distinction from a classical feedforward neural network. Complex-valued neural networks are as natural as the real-valued ones. Using complex-valued inputs/outputs, weights and activation functions, it is possible to increase the functionality of a single neuron and a neural network, to improve their performance, and to reduce the training time [6].

MVN, which is the first historically known complex-valued neuron, implements a mapping between *n* inputs and a single output. While MVN's inputs and output are complex numbers located on the unit circle, its weights are arbitrary complex numbers. An input/output mapping of a continuous MVN is described by a function of *n* variables $f(x_1,...,x_n)$, $f: O^n \to O$, where O is a set of points located on the unit circle. An input/output mapping of a discrete MVN is also described by a function of *n* variables $f(x_1,...,x_n)$, $f: E_k^n \to O$, where $E_k = \left\{\varepsilon_k^0, \varepsilon_k, \varepsilon_k^2, ..., \varepsilon_k^{k-1}\right\}$, $\varepsilon_k = e^{i2\pi/k}$, and i is an imaginary unit. When an MVN output is also discrete, the last function is transformed to $f: E_k^n \to E_k$, while if MVN inputs are continuous and its output is discrete, this function becomes $f: O^n \to E_k$. Such a function can be represented using *n*+1 complex-valued weights as follows [6]:

$$f(x_1,...,x_n) = P(w_0 + w_1x_1 + ... + w_nx_n), \quad (1)$$

where $x_1,...,x_n$ ($x_j \in E_k, j = 1,...,n$) are neuron inputs and $w_0, w_1, ..., w_n$ are the weights. *P* is the activation function, which is for the continuous and discrete MVN, respectively:

$$P(z) = e^{iArg(z)} = z/|z| \quad (2)$$

$$P(z) = \varepsilon_k^j = e^{i2\pi j/k}, \text{ if } 2\pi j/k \le \arg z < 2\pi(j+1)/k, \quad (3)$$

where $z = w_0 + w_1x_1 + ... + w_nx_n$ is the weighted sum, Arg *z* is the main value of the argument of the complex number *z*.

The MVN learning is based on the error-correction learning rule [6]:

$$W_{r+1} = W_r + \frac{C_r}{(n+1)|z_r|}(D-Y)\bar{X}, \quad (4)$$

where $\bar{X}$ is the vector of neuron inputs complex-conjugated, *n* is the number of neuron inputs, *D* is the desired output of the neuron, $Y = P(z)$ is the actual output of the neuron, *r* is the number of the learning step, W_r is the current weighting vector, W_{r+1} is the following weighting vector, C_r is a learning rate (it is complex-valued in general, but in all simulations known so far $C_r = 1$ was used. So in all simulations, which we have done in this work, we used $C_r = 1$ either), and $|z_r|$ is the absolute value of the weighted sum obtained on the *r*th learning step. A factor $1/|z_r|$ should be used when correcting the weights of hidden neurons in a neural network, which for the exact errors are not known. But it should not be used for output neurons in a network, which for the exact errors are known.

The use of MVN as a basic neuron in an MLMVN was suggested in [5]. MLMVN is a feedforward neural network (like MLP), but its significant distinctions and advantages are determined by using MVN as its basic neuron. The most important advantages of MLMVN are its higher functionality, better generalization capability and simplicity of learning when compared to MLP. MLMVN learning is derivative-free. Its backpropagation learning algorithm [5]-[7] is based on the same error-correction learning rule (4) as the one for a single MVN.

In this paper, we will employ the batch learning technique for MLMVN recently suggested in [8]. This is a batch linear least squares (LLS) learning algorithm, which drastically (two orders of magnitude) reduces the number of learning iterations and learning time keeping the same generalization accuracy as a regular MLVN learning algorithm. Batch learning for a neural network is a learning technique where, in each iteration, the weights are adjusted for all learning samples simultaneously. This means that the network errors shall be calculated first for all learning samples, and then the adjustment factors should be found based on these errors. In [9], it was suggested to use such a procedure to train MLMVN with a single output neuron. A batch procedure was applied only to the output neuron, while all hidden neurons were trained in the regular way. Then in [8] this procedure was modified and adapted to MLMVN with multiple output neurons.

Let us consider an MLMVN with n inputs, a single hidden layer containing H neurons, and multiple neurons in the output layer. Let us have a learning set containing M learning samples. The network errors should be calculated as

$$\delta_{j2}^* = D_{j2} - Y_{j2}; j = 1,...,N_2. \quad (5)$$

where D_{j2} is the desired output of the *j*th neuron from the 2nd (output) layer; Y_{j2} is the actual output of the same neuron, and N_2 is the number of output neurons. Then these errors they should be backpropagated as follows

$$\delta_{j2} = \frac{1}{t_2}\delta_{j2}^*; j = 1,...,N_2, \quad (6)$$

where *j*2 specifies the *j*th neuron of the 2nd (output) layer; $t_2 = N_1 + 1$, i.e. the number of all neurons in the preceding layer (the first hidden layer where the error (6) will be then backpropagated to) incremented by 1.

Then the errors of the first hidden layer neurons are

$$\delta_{i1} = \frac{1}{n+1}\sum_{j=1}^{N_2}\delta_{j2}(w_i^{j,2})^{-1}, \quad (7)$$

©2016 Society for Imaging Science and Technology
DOI: 10.2352/ISSN.2470-1173.2016.15.IPAS-013

where $i1$ specifies the jth neuron of the 1st (hidden) layer, $w_i^{j,2}$ is the ith weight of the jth neuron from the 2nd (output) layer, and n is the number of network inputs.

The learning process shall continue until the root mean square error of the network (averaged over all output neurons) drops below a pre-determined desired tolerance threshold. Hence the squared error of the network for the rth learning sample is

$$\Delta_r = \frac{1}{N_2}\sum_{j=1}^{N_2}\left(\gamma_{jr}\right)^2; r = 1,...,M. \tag{8}$$

where M is the number of learning samples and $\gamma_r = \left(\alpha_{j_r} - \alpha_r\right)\bmod k; \alpha_{j_r}, \alpha_r \in \{0,1,...,k-1\}$ (k is taken from (3)) is a local error for the rth learning sample taken from the jth output neuron. Then the MLMVN learning process (under RMSE criterion) continues until RMSE drops below a pre-determined desired tolerance threshold λ:

$$RMSE = \sqrt{\frac{1}{M}\sum_{r=1}^{M}\Delta_r} \le \lambda. \tag{9}$$

To adjust simultaneously the weights of the h th hidden neuron ($h = 1,...,N_1$), we need to add the weight adjustment factors $\Delta w_0^{[h]}...\Delta w_n^{[h]}$ to its weights. For the h th hidden neuron and for the j th learning sample must satisfy [8]

$$\Delta w_0^{[h]} + \Delta w_1^{[h]} x_1^j + ... + \Delta w_n^{[h]} x_n^j = \delta_j^{[h]} \tag{10}$$

where $x_1^j,...,x_n^j$ are the network inputs for the j th learning sample. Actually, (10) can be considered as a system of M linear algebraic equations in $n+1$ unknowns (adjustment terms) $\Delta w_0^{[h]},...,\Delta w_n^{[h]}$ over the field of complex numbers. These unknowns are the $\Delta w_0, \Delta w_1,...,\Delta w_n$
In matrix-vector notation we can rewrite (10) as

$$\begin{bmatrix} 1 & x_{1,1} & \cdots & x_{1,n} \\ \vdots & \vdots & & \vdots \\ 1 & x_{M,1} & \cdots & x_{M,n} \end{bmatrix} \begin{bmatrix} \Delta w_0^{[h]} \\ \vdots \\ \Delta w_n^{[h]} \end{bmatrix} = \begin{bmatrix} \delta_1^{[h]} \\ \vdots \\ \delta_M^{[h]} \end{bmatrix} \tag{11}$$

or simply

$$\mathbf{X}\mathbf{a}^{[h]} = \boldsymbol{\delta}^{[h]} \tag{11a}$$

In principle, if there exists an exact unique solution $\mathbf{a}^{[h]}$ to (11a), we can adjust the weights $w_0^{[h]}, w_1^{[h]}..., w_n^{[h]}$ in a manner that reduces the neuron's error for all M learning samples simultaneously. Then the adjusted weights $w_0^{h,1}, w_1^{h,1}..., w_n^{h,1}$ are given by

$$\tilde{\mathbf{w}}^{[h]} = \mathbf{w}^{[h]} + \mathbf{a}^{[h]} \tag{12}$$

where $\mathbf{w}^{[h]} = \left[w_0^{h,1}...w_n^{h,1}\right]^{\mathrm{T}}$ and $\tilde{\mathbf{w}}^{[h]} = \left[\tilde{w}_0^{h,1}...\tilde{w}_n^{h,1}\right]^{\mathrm{T}}$.

System (11a) is typically overdetermined, meaning $M \gg n+1$. In this case, we can find a unique least squares solution $\tilde{\mathbf{a}}^{[h]}$ that satisfies

$$\tilde{\mathbf{a}}^{[h]} = \arg\min_{\mathbf{a}^{[h]}} \left\|\mathbf{X}\mathbf{a}^{[h]} - \boldsymbol{\delta}^{[h]}\right\|^2$$

$$\tilde{\mathbf{a}}^{[h]} = \mathbf{X}^{\dagger}\boldsymbol{\delta}^{[h]} \tag{13}$$

where $\mathbf{X}^{\dagger} = (\mathbf{X}^{\mathrm{H}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{H}}$ is the pseudo-inverse of $\mathbf{X}$ and $(\ )^{\mathrm{H}}$ denotes the conjugate-transpose (transjugation) operation. For computer implementation purposes, $\mathbf{X}^{\dagger}$ can be efficiently computed using the QR decomposition or the singular value decomposition (SVD) of $\mathbf{X}$. The adjusted weights of the hidden neuron are then given by

$$\tilde{\mathbf{w}}^{[h]} = \mathbf{w}^{[h]} + \tilde{\mathbf{a}}^{[h]} \tag{14}$$

Eq. (13) and (14) are of course valid for any hidden neuron $[h], h = 1,...,N_1$, and therefore provide a general LLS-based learning rule for hidden neurons.

Once the weights of all hidden neurons are adjusted, and their outputs are updated, we can apply the same procedure to adjust the weights of output neurons. It is important to take into account that before adjusting output neurons' weights, we must update the network errors according to (5) and the output neurons' errors according to (6). It is straightforward to see that for the output neurons (11a), (13), (14) are transformed to the following equations, respectively:

$$\begin{bmatrix} 1 & x_{1,1}^{[o]} & \cdots & x_{1,H}^{[o]} \\ \vdots & \vdots & & \vdots \\ 1 & x_{M,1}^{[o]} & \cdots & x_{M,H}^{[o]} \end{bmatrix} \begin{bmatrix} \Delta w_0^{[o]} \\ \vdots \\ \Delta w_H^{[o]} \end{bmatrix} = \begin{bmatrix} \delta_1^{[o]} \\ \vdots \\ \delta_M^{[o]} \end{bmatrix}$$

$$\mathbf{X}^{[o]}\mathbf{a}^{[o]} = \boldsymbol{\delta}^{[o]} \tag{15}$$

$$\tilde{\mathbf{a}}^{[o]} = \left(\mathbf{X}^{[o]}\right)^{\dagger}\boldsymbol{\delta}^{[o]} \tag{16}$$

$$\tilde{\mathbf{w}}^{[o]} = \mathbf{w}^{[o]} + \tilde{\mathbf{a}}^{[o]} \tag{17}$$

In (15)-(17), $[o], o = 1,...,N_2$ is the index of an output neuron, $\mathbf{X}^{[o]}$ is the matrix of output neuron inputs, or equivalently the matrix of hidden neuron outputs.

Namely, a particular entry $x_{j,h}^{[o]}$ is given by

$$x_{j,h}^{[o]} = y_j^{[h]} = P\left(\begin{bmatrix}1 & x_{j,1} & \cdots & x_{j,n}\end{bmatrix}\tilde{\mathbf{w}}^{[h]}\right) \tag{18}$$

where $y_j^{[h]}$ is the output of the h th hidden neuron for the j th learning sample, and $P(\ )$ is the MVN activation function (1). Also, consistent with previously used notation, $\mathbf{w}^{[o]} = \left[w_0^{o,2}...w_H^{o,2}\right]^{\mathrm{T}}$ is the vector of original output neuron weights, and $\tilde{\mathbf{w}}^{[o]} = \left[\tilde{w}_0^{o,2}...\tilde{w}_H^{o,2}\right]^{\mathrm{T}}$ is the vector of adjusted output neuron weights.

After the adjustment of output neuron weights, a learning iteration is completed. This learning procedure is iteratively repeated until the RMSE criterion (9) has been satisfied.

This algorithm is utilized in Matlab. The corresponding Matlab functions along with the data used for simulations in this paper can be found at http://www.freewebs.com/igora/ under "Download Software Simulators and Data".

Approach to Filtering using MLMVN

As it was mentioned above, our general approach to filtering is basically similar to the one used in [2]-[4], but additionally to the use of MLMVN instead of MLP we found that a simpler network topology can be employed as well as there are some distinctions in the implementation.

Thus the approach to filtering is as follows. A basic idea is to filter not isolated pixels based on the information taken from their local neighborhood, but to filter all pixels in an *n* x *n* patch simultaneously. An image to be filtered shall be divided into *n* x *n* overlapping patches. The smaller is the offset for creating patches, the better final result should be expected.

Each *n* x *n* patch shall be filtered separately using a neural network with n^2 inputs and n^2 neurons in the output layer. This means that all pixels in the patch shall be processed simultaneously based on the information only from the same patch. Each input of the network represents the intensity value in the corresponding pixel of the input patch. Each output neuron creates the output intensity value in the corresponding pixel of the output patch.

In [2]-[4], MLP with 4 hidden layers, each containing 512 neurons was found the most efficient for solving the filtering problem. However, we found experimentally that MLMVN with just a single, but relatively big hidden layer learns much faster than a network with multiple hidden layers, not losing the generalization capability. The use of just a single hidden layer makes it also possible to employ a batch linear least squares (LLS) learning algorithm [8], which speeds up the learning process drastically, also allowing for the extension of a learning set. Hence, we employed $n^2 \to N_1 \to n^2$ network topology (here the first $n^2 = n \times n$ is the number of network inputs, N_1 is the number of hidden neurons, and the second $n^2 = n \times n$ is the number of network output neurons and network outputs, accordingly).

To train MLMVN, a learning set containing the pixel intensities from the noisy patches randomly selected from different images as inputs and the pixel intensities from the patches of the corresponding clean images as desired outputs was created.

For our experiments at this stage of this work we used additive non-correlated Gaussian noise artificially generated for each image with $\sigma_{noise} = 0.2\sigma$ where σ is the standard deviation of the corresponding clean image.

We employed MVN with discrete inputs and continuous output in the hidden layer and MVN with continuous inputs and discrete output in the output layer.

Since MVN and MLMVN, accordingly, have complex-valued inputs and produce complex-valued outputs located on the unit circle, a pretty standard approach was used to transform integer input intensities into complex-valued inputs of the network and complex-valued outputs of the network to the integer output intensities. To avoid closeness of the "white" and "black" parts of the intensity range $I = \{0,1,...,255\}$ on the unit circle, we used $k = 288$ in (3). This value was chosen experimentally. While, for example, for $k = 264$ and $k = 384$ a learning process goes much slower, for $k = 288$ it goes much faster. This experimental finding confirms considerations made in [10] where it was shown that a multiple-valued discrete input/output mapping, being non implementable using a neuron or a neural network in *l*-valued logic, can be implementable in *m*-valued logic where $l \neq m$. Thus if the intensity $j \in I$, then the corresponding network input located on the unit circle is $e^{i2\pi j/288}$. Since the output neurons employ discrete activation function (3), then the output of each output neuron is determined by (3) and the corresponding output intensity equals j taken from (3) accordingly. Taking into account that $k = 288$ in (3) and to ensure that $j \in I$, the following additional procedure was used to adjust the resulting intensities, if necessary

$$255 < j < 272 \to j = 255,$$
$$272 \le j < 288 \to j = 0.$$

After MLMVN is trained, it can then be used for performing actual filtering. To filter an image, the overlapped patches with a given offset shall be taken from an image, starting from its top-left corner down to its bottom-right corner. Each patch shall be filtered separately using MLMVN. Then, the final output intensity $g(x,y)$ in the pixel with the coordinates (x,y) shall be found by averaging the resulting intensities $g_i(x,y), i = 1,...,S_{xy}$ in this pixel from all S_{xy} overlapping patches where this pixel appears

$$g(x,y) = \frac{\sum_{i=1}^{S_{xy}} g_i(x,y)}{S_{xy}}.$$

Experimental Results

To test the approach presented above, a number of experimental tests were performed.

The first question, which we had to answer, was: what size of a patch should be optimal? After testing different sizes of patches, we found that the best noise filtering results are obtained for the patch size 15x15. The smaller patches yield the larger ones, while starting from the 17x17 size the results practically do not show any improvement, while both learning and filtering take significantly longer time.

Then we also found that the smaller is the offset used for creating patches, the better final result is obtained. Thus the best offset is 1. So the more patches participate in the formation of output intensity through pixel-wise averaging, the closer to its expected ideal value this intensity is. This actually should be expected from the known fact that averaging a large number of different noisy realizations of the same image we may better approach a corresponding clean image. Hence best offset for overlapping patches is 1.

We made a number of experiments to find how many hidden neurons should be used in MLMVN to get better filtering results. On the one hand, it was discovered that the more hidden neurons are used, the easier it is to train MLMVN with a smaller RMSE. But on the other hand, a reasonably smaller network, even trained with a larger RMSE may give better filtering results. This finding can be explained from the following considerations. The more neurons are used in a single hidden layer, the larger is dimensionality of a space where we are trying to solve our problem. This means that the easier is to train MLMVN with a

©2016 Society for Imaging Science and Technology
DOI: 10.2352/ISSN.2470-1173.2016.15.IPAS-013

lower error. However, the larger is a space determined by a neural network, the smaller part of it is used for fitting there all learning samples. This in fact means that a network can be trained with a smaller error. A problem is that if a working space determining by a network is too big, this negatively affects this network's generalization capability. When a network "sees" an unknown sample, it may "put" it too far from its desirable location in that big space. In fact, when a working space is too large, there is always a chance that any test sample can be put too far from its desirable location. In our case this may negatively affect filtering PSNR/RMSE. To take care of this issue for larger networks, it is necessary to use larger learning sets accordingly. However, the latter may extend the learning process significantly, leading to more learning iterations and time needed to complete the learning process. We used MLMVN with 1024, 1536 and 2048 hidden neurons in our experiments. The results are shown and compared below.

Another important experimental finding is that the more images participate in the formation of a learning set, the better final filtering results are obtained.

Let us now demonstrate our experimental results. As it follows from our previous considerations, since the best results were obtained for 15x15 patches, we will show the results only for these patch size.

As for the images used in our experiments, there are pretty classical images used worldwide and widely available, like "Lena", "Airplane F16", "Cameraman", "Boat", "Bridge", "Man", "Mandrill" along with other more than 100 images downloaded from [11] and 300 more images (mostly different landscapes and urban views from the author's personal collection).

Our first experiment is interesting only from the standpoint of its "extremal" nature. We used a single image ("Lena") to create a learning set containing 6000 15x15 patches, whose starting coordinates were randomly selected. The learning process for MLMVN containing 2048 hidden neurons took 43 iterations to converge with RMSE=4.0 and 87 iterations to converge with RMSE 3.0. Then the "Airplane F16" image (Figure 1), which was not presented in the learning set, was filtered after Gaussian noise with the standard deviation $\sigma_{noise} = 0.2\sigma$ was added to it (Fig. 2). It is interesting that the network, being trained using the data taken just from just a single pair of noisy/clean images is able to filter other noisy images removing noise completely (Figure 3). However, a disadvantage of this approach is "narrowing" of histogram in a resulting image. This narrowing means lowering of the standard deviation (and variance accordingly), especially in the areas of high jump of intensities (what was "almost black" or "almost white" becomes "dark gray" or "light gray" if there is an intensity jump in the corresponding area). While noise can be removed, this lowers PSNR for a filtered image due this narrowing of its histogram.

It is important to mention that the same side effect (narrowing of a histogram) we observe in all our experiments. However, the more images are used to create a learning set and the bigger is the learning set (the more patches are used for learning), the less the resulting image is affected by this issue. Table 1 summarizes our simulation results. In all testing experiments we employed ten images, which were never used in our learning sets.

Table 1: Simulation Results: Summary of Learning and Filtering using MLMVN

# of images used in a learning set	# of learning samples (patches used for learning)	# of hidden neurons in MLMVN	# of learning iterations	Learning RMSE	Testing Standard Deviation (average over 10 images)	Testing PSNR (average over 10 images)	Min PSNR/ Max PSNR (over 10 images)
200	12,000	2048	284	5.0	8.18	30.44	25.90/ 35.10
200	12,000	1536	271	5.5	7.88	30.69	26.50/ 35.02
200	12,000	1024	165	6.3	7.57	30.94	27.24/ 34.90
400	12,000	1536	302	5.8	7.74	30.82	26.82/ 34.95
400	12,000	1024	155	6.7	7.44	31.07	27.59/ 34.95
400	16,000	1024	153	7.05	7.30	31.21	27.90/ 34.96

©2016 Society for Imaging Science and Technology
DOI: 10.2352/ISSN.2470-1173.2016.15.IPAS-013

These simulations lead to some conclusions. It is important to mention that MLMVN with 512 and 4096 hidden neurons were also tested, but they yield significantly to MLMVNs with 1024, 1536 and 2048 hidden neurons in terms of the filtering quality. Because of that the results for the networks with 512 and 4096 hidden neurons are not included in Table 1.

The experimental results show that the more representative is a learning set, the better filtering results are obtained. In fact, all networks (regardless of the number of hidden neurons used), which were trained using the learning sets created from 400 images, outperform the networks, which were trained using the learning set created from 200 images.

The network containing 1024 hidden neurons, which was trained using the learning set containing 16,000 samples taken from 400 images outperforms the same network trained using the learning set containing 12,000 samples taken from the same 400 images.

The larger is MLMVN (the more hidden neurons it employs), the smaller is RMSE with which it can be trained. However, at the same time, the smaller is MLMVN (the fewer hidden neurons it contains), the lower standard deviation from the original clean image (and the higher PSNR, accordingly) can be obtained even with a larger training RMSE when filtering images containing more small details. The latter does not mean that larger MLMVN is always less efficient for filtering. But as it was mentioned above, a network has more "degrees of freedom" in a larger space determining by a bigger network. Evidently, for images containing more small details this can be resulted in a larger deviation of the result from the expected one in this larger dimensional space. A smaller network, while yields a little bit to a larger one when filtering images with fewer small details, outperforms it when filtering images with more small details. Again, this property stably holds although any smaller network yields any larger network in its ability to reach a smaller learning PSNR value.

The smaller is MLMVN, the more images were used to create a learning set, and the more learning samples are contained in the learning set, the smaller is deviation in the results (this is clearly seen from the last column of Table 1) because all these conditions lead to better filtering of images containing more small details.

Let us now consider more examples.

Figure 4 presents the filtering result for the "Airplane F16" noisy image shown in Figure 2. This image was obtained using MLMVN with 1024 hidden neurons trained with the learning set containing 16,000 learning samples (patches), which were taken from 400 images, with the learning RMSE=7.05. Even visually this image looks much better than the one from Figure 3 (because its histogram is almost not narrowed). But in terms of PSNR=34.96 the quality of this image is excellent. BM3D filter with the noise standard deviation set to $\sigma_{noise} = 15.0$ applied to the same noisy image shown in Figure 2 is resulted in about the same PSNR=34.52 (see Figure 5). It is important to mention that BM3D filter and MLMVN affect an image differently. While after BM3D filtering no noise leftovers are visible at all, MLMVN removes noise, but some of its "washed" leftovers are visible at the homogenous areas (one may take a look at the big cloud and compare the one in Figure 4 and Figure 5). BM3D filter preserves sharp edges more carefully than MLMVN, however MLMVN better preserves small details whose edges are not necessary sharp (one may compare characters and digits on the one hand and details of the mountain on the other hand in Figures 4 and 5, respectively).

Exactly the same conclusion we can make analyzing and comparing to each other images from Figures 8 and 9. Figure 8 presents the filtering result for the "Sailfish" noisy image shown in Figure 7 (this noisy image was obtained from the original "Sailfish" image shown in Figure 6 by adding Gaussian noise with the standard deviation $\sigma_{noise} = 0.2\sigma$) using MLMVN. Figure 9 presents the filtering result for the same image from Figure 7, but using BM3D filter with the noise standard deviation set to $\sigma_{noise} = 15.0$. We again observe comparable results, while MLMVN very slightly outperforms BM3D filter (PSNR 33.53 and 33.24, respectively). Again noise is completely removed and sharp edges are better preserved after BM3D filtering. Again, while some "washed" noise leftovers are observed after filtering using MLMVN, small details are better preserved.

Comparing other test images, we are coming to the same conclusions. While BM3D filter removes noise completely and better preserves sharp edges, it may "wash" small details. While MLMVN may leave some "washed" noise leftovers and smooth sharp edges, it better preserves the smallest details.

Let us consider another example. The "Train Station" test image is shown in Figure 10. Its noisy version with Gaussian noise with the standard deviation $\sigma_{noise} = 0.2\sigma$ added is shown in Figure 11. Figure 12 presents the filtering result for the "Train Station" noisy image shown in Figure 11 and filtered using MLMVN. Figure 13 presents the filtering result for the same noisy image filtered using BM3D filter. The "Train Station" image contains a number of different textures, sharp edges and small details. If we take a look at the small detailed walkway texture, we clearly see that it is completely gone after BM3D filtering, while it is pretty well preserved after filtering by MLMVN. While if we take a look at pretty homogenous textures of the train cars under windows or the rough above the walkway, we have to conclude that their homogeneity is restored by BM3D filtering, however they contain many "washed" noise leftovers after MLMVN filtering. Since this image contains more different pretty homogenous areas and sharp edges than small detailed textures, BM3D filter shows better PSNR than MLMVN (33.34 and 31.51, respectively).

The comparative results for all 10 test images are summarized in Table 2. The best result for MLMVN 225-1024-225 trained with RMSE=7.05 was taken.

Table 2: Simulation Results: Comparison of Filtering using MLMVN and BM3D filter

Test Images		1	2	3	4	5	6	7	8	9	10	Average
MLMVN	PSNR	31.51	29.78	28.43	30.28	28.49	27.90	33.53	32.29	34.92	34.96	31.21
	St. Div.	6.77	8.27	9.66	7.80	9.60	10.26	5.37	6.19	4.57	4.55	7.30
BM3D	PSNR	33.34	30.35	29.97	30.37	29.88	29.34	33.24	32.71	36.43	34.52	32.01
	St. Div.	5.48	7.74	8.09	7.72	8.17	8.69	5.55	5.60	3.84	4.79	6.60

DOI: 10.2352/ISSN.2470-1173.2016.15.IPAS-013

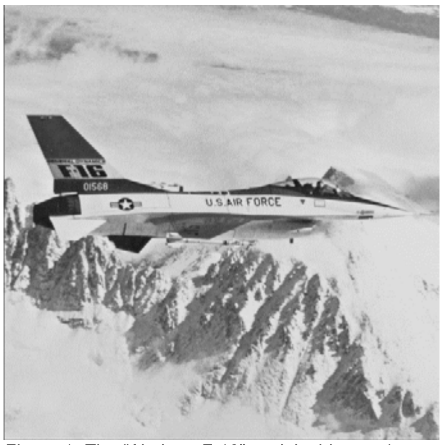
Figure 1. The “Airplane F-16” - original image (was not used in the learning sets)

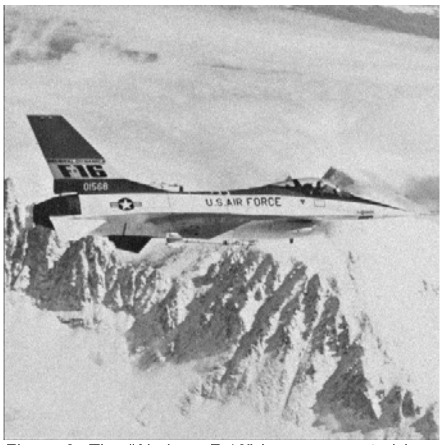
Figure 2. The “Airplane F-16” image corrupted by additive Gaussian noise with the standard deviation $\sigma_{noise} = 0.2\sigma$

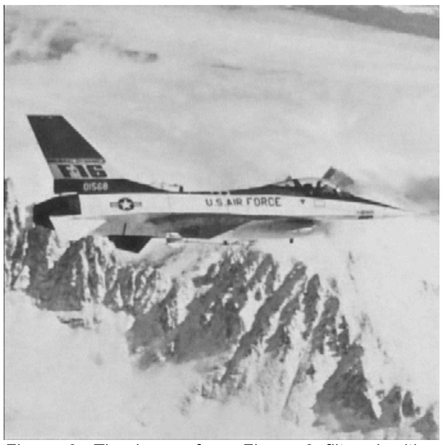
Figure 3. The image from Figure 2 filtered with MLMVN after using a learning set created from the 6000 patches taken from a single image. Learning RMSE= 4.0; PSNR=32.35. Image histogram is narrowed.

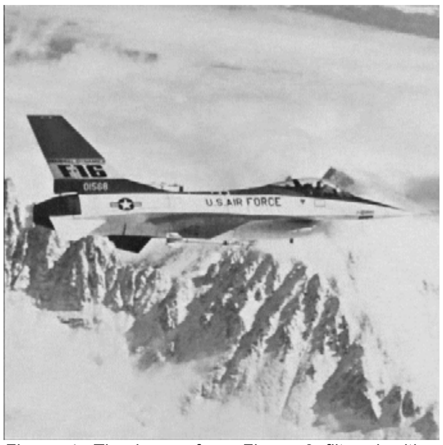
Figure 4. The image from Figure 2 filtered with MLMVN containing 1024 hidden neurons after using a learning set created from the 16000 patches taken from 400 images. Learning RMSE=7.05; PSNR=34.96. Small details are mostly preserved.

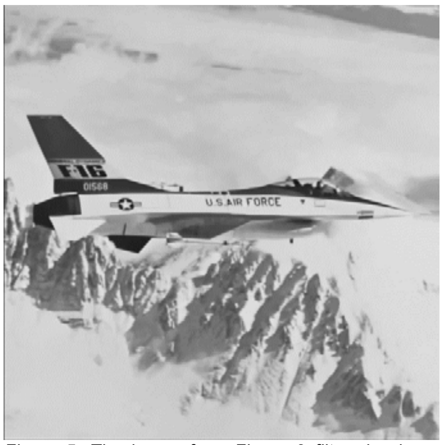
Figure 5. The image from Figure 2 filtered using BM3D filter with the $\sigma_{noise} = 15.0$ *; PSNR=34.52. Some small details are gone after filtering.*

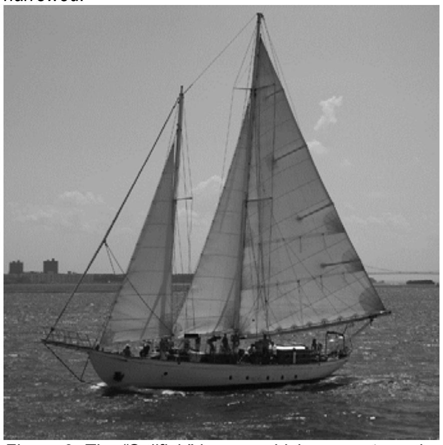
Figure 6. The “Sailfish” image, which was not used in the learning sets (the original image)

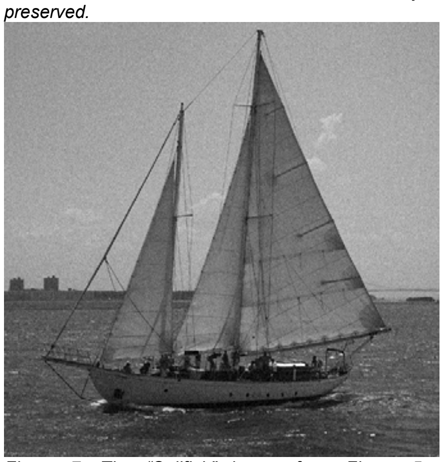
Figure 7. The “Sailfish” image from Figure 5 corrupted by additive Gaussian noise with the standard deviation $\sigma_{noise} = 0.2\sigma$

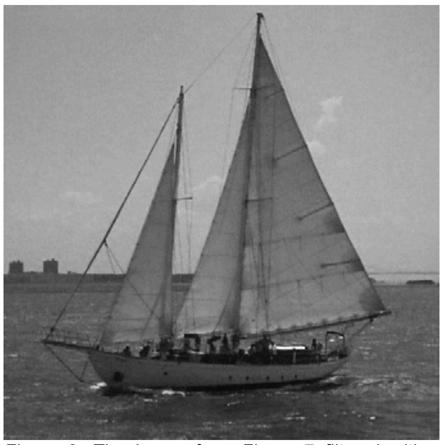
Figure 8. The image from Figure 7 filtered with MLMVN containing 1024 hidden neurons after using a learning set created from the 16000 patches taken from 400 images. Learning RMSE=7.05; PSNR=33.53. Small details are mostly preserved.

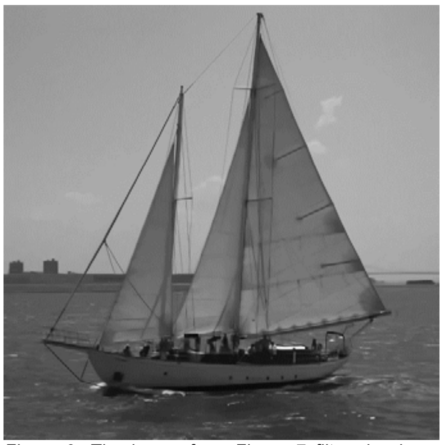
Figure 9. The image from Figure 7 filtered using BM3D filter with the $\sigma_{noise} = 15.0$ *; PSNR=33.24. Some small details are gone after filtering.*

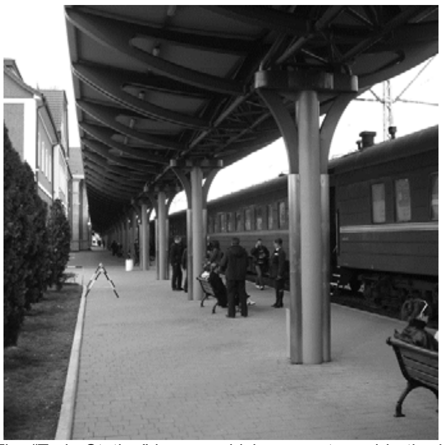
Figure 10. The "Train Station" image, which was not used in the learning sets (the original image)

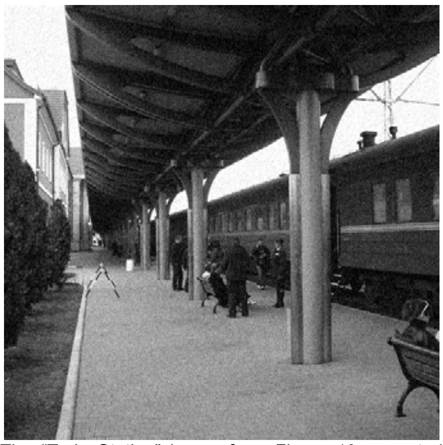
Figure 11. The "Train Station" image from Figure 10 corrupted by additive Gaussian noise with the standard deviation $\sigma_{noise} = 0.2\sigma$

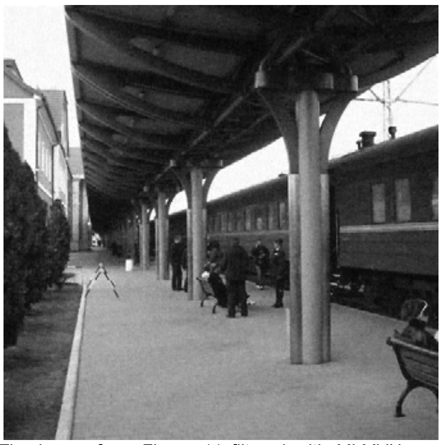
Figure 12. The image from Figure 11 filtered with MLMVN containing 1024 hidden neurons after using a learning set created from the 16000 patches taken from 400 images. Learning RMSE=7.05; PSNR=31.51. The walkway texture is mostly preserved, but "washed" noise leftovers remain in a number of homogenous areas.

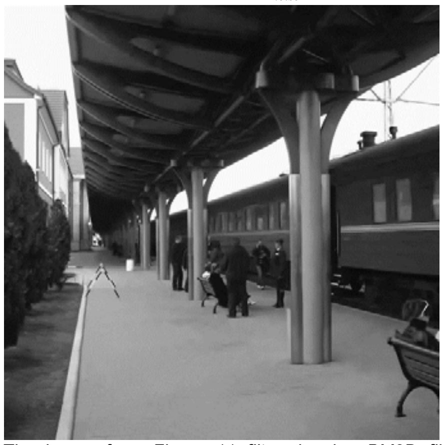
Figure 13. The image from Figure 11 filtered using BM3D filter with the $\sigma_{noise} = 15.0$ *; PSNR=33.34. The walkway texture is gone, but noise is also completely gone.*

As we see from Table 1, MLMVN slightly outperforms BM3D filter for 2 images out of 10; slightly yields, but shows comparable results on 4 images out of 10 and yields to BM3D on 4 images out of 10. Based on the statistics, which we got from this experiment, we have to conclude that as for this moment MLMVN slightly yields to BM3D filter, but nevertheless shows a comparable result. MLMVN better preserves small details and highly detailed textures, however, some "washed" noise leftovers remain in a resulting image. At the same time, BM3D filter better preserves sharp edges and removes noise completely.

About the same results were reported in [2]-[4] for MLP (when compared to BM3D filter), however, MLMVN used in this paper contains 4 times less hidden neurons than MLP employed in [2]-[4]. Moreover, all hidden neurons in MLMVN used in this paper are located in a single hidden layer, while 4-layer network, which requires significantly more operations and time for its training, was employed in [2]-[4].

It is important to mention that previously MVN was used for image filtering as a part of a cellular neural network [12] and also MLMVN was used as a high pass filter for edge detection [13]. But basically, in both those earlier works a pretty standard approach to spatial domain filtering based on the local neighborhood processing, which creates an output for a single pixel, was implemented. Particularly, the results reported in [12] were just comparable with order statistic filters, but they were not better. Definitely, MLMVN filtering presented in this paper gives significantly better results.

Conclusions and Furture Work

It was shown that MLMVN can successfully be used as an intelligent filter. It outperforms MLP because it makes possible to use a simpler network with just a single hidden layer and to get the learning algorithm converged much faster employing a batch LLS-based learning algorithm.

Since the more images are used to create a learning set, the better filtering results are obtained, a very clear direction for the further work is to use learning sets based on more images and improve the results in this way.

It should also be interesting to discover whether all estimations of the resulting intensities from overlapping patches should be used to obtain the resulting intensities or for example, those

©2016 Society for Imaging Science and Technology
DOI: 10.2352/ISSN.2470-1173.2016.15.IPAS-013

located closer to the borders of the patch should be ignored (or some weights should be assigned to different estimations depending how far from the center of a corresponding patch they are located).

Another interesting direction for the further work is to apply the same approach for edge detection, especially for edge detection on noisy images (to suppress noise and detect clean edges).

It should also be considered whether MLMVN can be used in about the same way for restoration of blurred images.

Another attractive continuation of this work can be study of how to use MLMVN for filtering in the frequency domain. In fact, a complex-valued neural network can be potentially a great tool to process complex-valued data in the Fourier domain.

Acknowledgement

The author greatly appreciates Prof. Karen Egiazarian's suggestion to try to use MLMVN for this intelligent approach to image filtering.

Image processing with BM3D filer was done using the Matlab implementation available online from the web page devoted to BM3D filter by its authors [14].

References

[1] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, "Image denoising by sparse 3-D transform-domain collaborative filtering". IEEE Transactions on Image Processing, vol. 16, no 8, pp. 2080–2095, 2007.

[2] H. C. Burger, C. J. Schuler, and S. Harmeling, "Image denoising: Can plain Neural Networks compete with BM3D?", in Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR-2012), pp. 4321-4328, 2012.

[3] H. C. Burger, C. J. Schuler, and S. Harmeling, "Image denoising with multi-layer perceptrons, part 1: comparison with existing algorithms and with bounds", available online at http://arxiv.org/pdf/1211.1544.pdf, 2012.

[4] H. C. Burger, C. J. Schuler, and S. Harmeling, "Image denoising with multi-layer perceptrons, part 2: training trade-offs and analysis of their mechanisms, available online at http://arxiv.org/pdf/1211.1552.pdf, 2012.

[5] I. Aizenberg and C. Moraga, "Multilayer Feedforward Neural Network Based on Multi-Valued Neurons (MLMVN) and a Backpropagation Learning Algorithm", Soft Computing, vol. 11, no 2, pp. 169-183, 2007.

[6] I. Aizenberg, "Complex-valued neural networks with multi-valued neurons, Springer", Hidelberg, 2011.

[7] I. Aizenberg, D. Paliy, J. Zurada, and J. Astola, "Blur Identification by Multilayer Neural Network based on Multi-Valued Neurons", IEEE Transactions on Neural Networks, vol. 19, no 5, pp. 883-898, 2008.

[8] E. Aizenberg and I. Aizenberg, "Batch LLS-based Learning Algorithm for MLMVN with Soft Margins", Proceedings of the 2014 IEEE Symposium Series of Computational Intelligence (SSCI-2014), pp. 48-55, 2014.

[9] I. Aizenberg, A. Luchetta, and S. Manetti, "A modified Learning Algorithms for the Multilayer Neural Network with Multi-Valued Neurons based on the Complex QR Decomposition", Soft Computing, vol. 16, pp. 563-575, 2012.

[10] I. Aizenberg, "A Periodic Activation Function and a Modified Learning Algorithm for a Multi-Valued Neuron", IEEE Transactions on Neural Networks, vol. 21, no 12, pp. 1939-1949, 2010.

[11] Test Images. University of Granada Image Data Base. Available Online http://decsai.ugr.es/cvg/dbimagenes/

[12] I. Aizenberg and C. Butakoff, "Image Processing Using Cellular Neural Networks Based on Multi-Valued and Universal Binary Neurons", Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology, vol. 32, pp. 169-188, 2002.

[13] I. Aizenberg, S. Alexander, J. Jackson, T. Neal, J. Wilson, and K. Kendrick, "Intelligent edge enhancement using multilayer neural network based on multi-valued neurons", SPIE Proceedings, vol. 7870 (Proc. of SPIE Congress "Electronic Imaging 2011"), March 2011, pp. 7870004-1 – 7870004-11.

[14] Image and video denoising by sparse 3D transform-domain collaborative filtering. Block-matching and 3D filtering (BM3D) algorithm and its extensions. Tampere University of Technology. Online http://www.cs.tut.fi/~foi/GCF-BM3D/index.html#ref_software

Author Biography

Igor Aizenberg received his MS in mathematics from Uzhhorod National University (Ukraine) (1982) and his PhD in computer science from Dorodnitsyn Computing Center of the Russian Academy of Science (1986). Then he worked in academia and industry in Russia, Ukraine, Belgium, Israel, Finland, Germany, and USA. Since 2006 he is with Texas A&M University-Texarkana (Texarkana TX, USA) where he is currently Professor of Computer Science and Director of the Computational Intelligence Lab. His main research areas are complex-valued neural networks and image filtering.