Inpainting by Sensitive Mean Filtering

Leo Büttiker Urs Fässler Nicolas Rüegg
Group moebs, Department of Computer Science, ETH Zurich, Switzerland

***Abstract*—A task often required in image processing is inpainting – the reconstruction of missing pixels where the location of the missing pixels is known. While many different methods for inpainting exist, we present a particularly intuitive method based on *Mean Filtering*. Our method called *Inpainting by Sensitive Mean Filtering* is extremly fast and achieves qualitatively good results. To support this claim we present results of our method compared to *Inpainting with Overcomplete Dictionaries* and an algorithm based on partial differential equations.**

I. Introduction

Images often contain missing values for certain pixels or pixel patches. Examples are scratches or random noise in an image. Using an inpainting method, one tries to approximate values for these pixels. In *Inpainting with Overcomplete Dictionaries*, overcomplete Haar wavelet or DCT dictionaries are often used to reconstruct the missing pixels. These methods both proved to be quite slow even for relatively small images. Another algorithm for inpainting is *Inpainting NaNs* – an algorithm based on partial differential equations.

In this paper, we present a simpler and faster method for inpainting of small parts of images. In section II, we describe our *Inpainting by Sensitive Mean Filtering* method. It is an intuitive and simple method in which missing pixel values are approximated by using existing neighboring pixels.

Experiments discussed in III have shown that considering random noise, our method achieves appealing results over a broad range of images.

In section IV we discuss the advantages and disadvantages in terms of computation time and error of our method in comparison to the methods *Inpainting with Overcomplete Dictionaries* and *Inpainting NaNs*.

Finally, section V gives an outlook on future work and section VI summarizes our results.

II. Sensitive Mean Filtering

The *Sensitive Mean Filter* is an extension of the well known *mean filter* as described in [1]. This algorithm sets the value of every pixel to the mean of itself and its surrounding pixels. This is based on the idea that values of spatial close pixels usually just vary moderately.

Unlike the classical *mean filter*, our method uses a bitmask which indicates the missing pixels. By exploiting this additional information, we can build the mean just over given pixels and can also guarantee not to corrupt correct pixels.

Our method can formally be described as follows: Let $f_{m,n}$ be a $[0,1]$ pixel value of a $M \times N$ gray scale image and let l be the given $M \times N$ mask indicating the missing pixels, where

$$l_{m,n} = \begin{cases} 1 & \text{if pixel (m,n) is set} \\ 0 & \text{if pixel (m,n) missing} \end{cases} \tag{1}$$

For a $(2K+1)\times(2L+1)$ kernel, we calculate the value of a pixel $f'_{i,j}$ in the approximation of the image f' as follows:

$$f'_{i,j} = \frac{\sum_{m=i-K}^{i+K} \sum_{n=j-L}^{j+L} f_{m,n}}{\sum_{m=i-K}^{i+K} \sum_{n=j-L}^{j+L} l_{m,n}} \tag{2}$$

We assume missing pixels in f to be set to 0. Therefore, formula 2 calculates the mean over the given correct pixels.

We calculate the value for $f'_{i,j}$ on a 3×3 kernel as long as the denominator is not equal to zero. I.e., a pixel is surrounded by at least one given pixel. If this approach fails, we take a 5×5 kernel. If this also fails, we set the missing pixel to the overall mean of the image.

As a post processing step, smoothing is performed. For this we use the Linear Noise Cleaning method as described in [2].

In a first step, we set the given correct pixels from f in the approximated image f' and apply a discrete convolution on it with the impulse response array H_1. It defines a low-pass noise cleaning filter.

$$H_1 = \frac{1}{100}\begin{bmatrix} 1 & 8 & 1 \\ 8 & 64 & 8 \\ 1 & 8 & 1 \end{bmatrix} \tag{3}$$

In a second step, we set the given correct pixels from f again and use a different filter H_2. This kernel places emphasis on the directly neighboring pixels.

$$H_2 = \frac{1}{8}\begin{bmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{4}$$

As a final step, we set the given correct pixels from f for the last time to get the reconstructed image.

Sensitive Mean Filtering can be implemented computationally efficient as a set of matrix operations.

III. Results

From a subjective point of view, our method achieves visually excellent results for missing patches of less than 5×5 pixels. This is true for randomly as well as non-randomly missing pixels as Figure 1 and Figure 2 show.

We tested our method on twelve different 512×512 gray scale images, each with ten different masks of missing pixels. Each mask consisted of 60% randomly missing pixels. The set of images encompasses the categories *Natural Pictures*, *Comics* and *Textures* with four images in each one. The category *Natural Pictures* includes photographs, *Comics* contains comic images with hard transitions of intensity and *Textures* consists of photographs of small repeating structures such as a brick wall or grass.

To quantify the error of our method, we used the Mean Square Error (MSE) over every single reconstructed image:

$$MSE = \frac{1}{M\,N} \sum_{i=1}^{M} \sum_{j=1}^{N} [f_{i,j} - f'_{i,j}]^2 \qquad (5)$$

The visually appealing result in Figures 1 and 2 is confirmed by the expectation value over all MSEs which is 0.0054. The standard deviation is 0.0076.

The experiments also showed that our method does not depend on the mask for randomly generated masks.

Figure 3 shows that our method achieves significantly smaller errors on the category *Natural Pictures* than on the category *Comics* and *Textures*, respectively. The expectation value of the error on *Natural Pictures* is 0.0011. For *Comics* it is at 0.0093 and for *Textures* at 0.0057. We assume this to be due to the harder transitions of intensity in *Comics* and fine-grained structures in *Textures*.

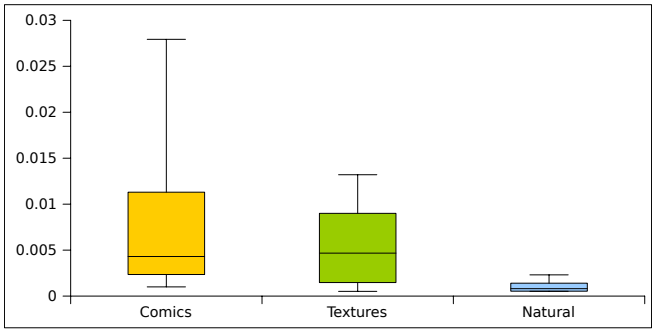


Figure 3. Error (MSE) depending on the image category. Generated on the test set of the twelve images and ten randomly generated masks of 60% missing pixels.

On 512×512 images, our method implemented in *Matlab* running on an Intel Core i5 CPU M 520 at 2.40GHz with 4GB memory needs approximately 0.26 seconds. It does not depend on the rate of of missing pixels as Figure 7 shows.

IV. Discussion

In this section we look at our method's advantages and disadvantages and compare it to *Inpainting with Overcomplete Dictionaries* with a Haar wavelet dictionary and a DCT dictionary, as well as to *Inpainting NaNs*.

Inpainting NaNs is a boundary value solver. It is described by its author as follows[3]: "The basic idea is to formulate a partial differential equation (PDE) that is assumed to apply in the domain of the artifact to be inpainted. The perimeter of the hole supplies boundary values for the PDE. Then the PDE is approximated using finite difference methods (the array elements are assumed to be equally spaced in each dimension) and then a large (and very sparse) linear system of equations is solved for the NaN elements in the array." In our context, the NaN elements are the missing pixels.

Inpainting with Overcomplete Dictionaries is implemented as described in [4]. The parameters for the Haar wavelet and DCT dictionary are chosen as in [5]. I.e., the image is split into non-overlapping patches of size 8×8 and the individual patches are coded into vectors $x \in \mathbb{R}^{64}$. With the vectorized pixel patches as columns, the entire image is coded in a matrix $X \in \mathbb{R}^{64 \times (d/8)^2}$, where d is the side length of the quadratic image in pixels. For DCT as well as Haar wavelets, the dictionary $U \in \mathbb{R}^{64 \times 441}$ contains 441 linearly dependent atoms. Finally, using a *Matching Pursuit* algorithm, matrix X is approximated by atoms from the overcomplete dictionary U.[1]

An experiment over the test set of the twelve images and ten randomly generated masks of 60% missing pixels confirmed the competitiveness of our method. The resulting error is shown in Figure 4. As Figure 5 shows, *Inpainting by Sensitive Mean Filtering* also performs good for lower and higher rates of missing pixels than 60%.

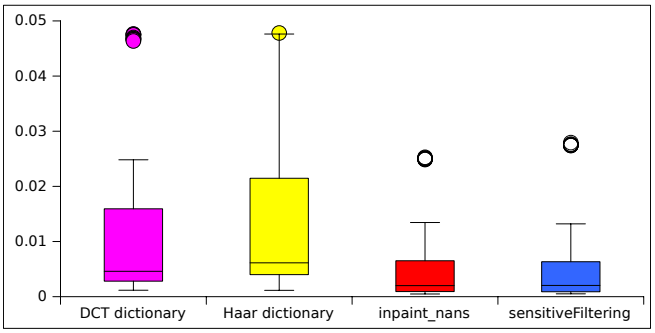
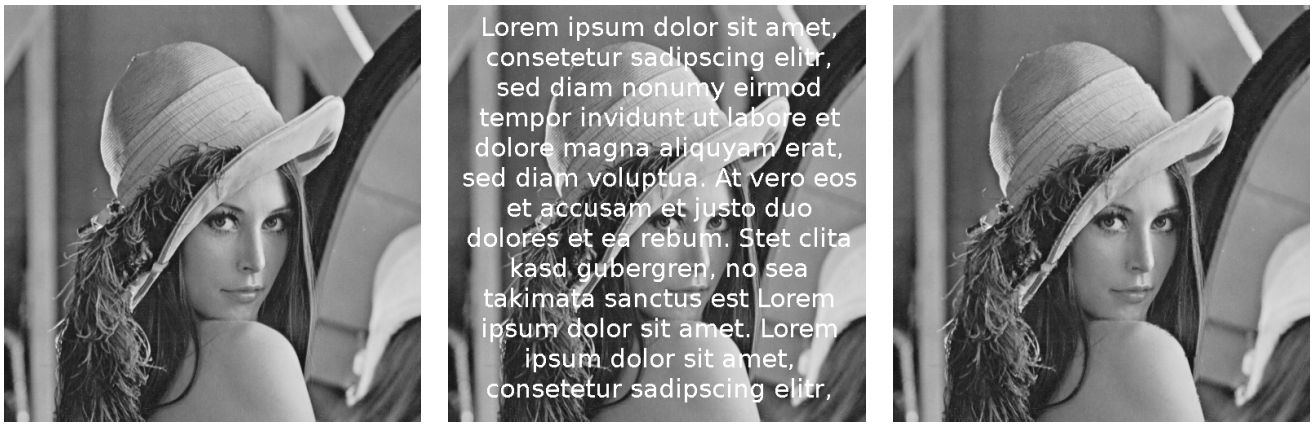


Figure 4. Error (MSE) depending on the inpainting method. Generated on the test set of twelve images and ten randomly generated masks of 60% missing pixels.

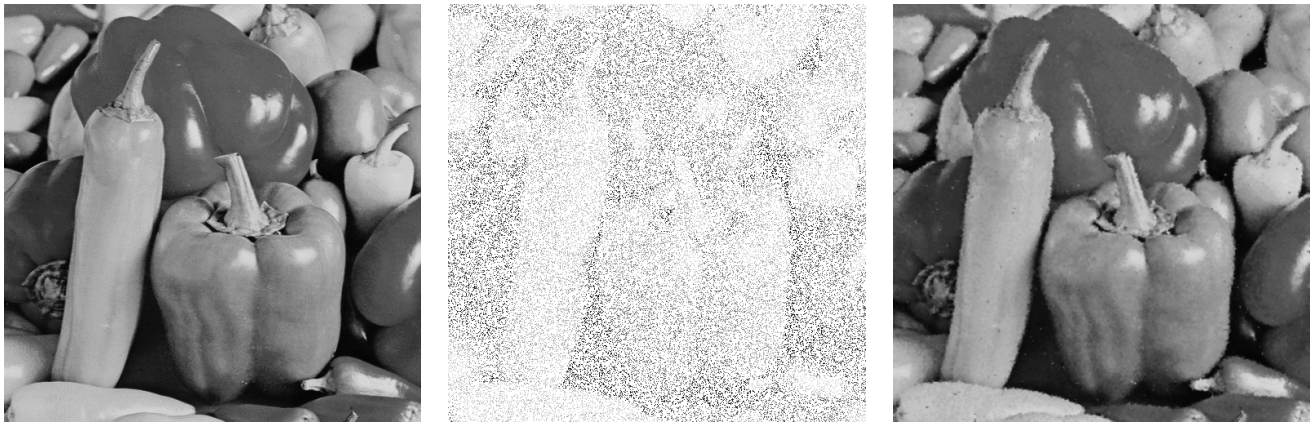
But despite of the fact that *Inpainting by Sensitive Mean Filtering* achieves competitive results regarding the quality

[1]The algorithm to generate the overcomplete Haar wavelet dictionary was taken from [6]. The algorithm to generate the overcomplete DCT dictionary was taken from [7], [8].



(a) Original (b) With missing pixels (c) Reconstructed with Sensitive Mean Filtering

Figure 1. The *Inpainting by Sensitive Mean Filtering* Method achieves visually appealing results on relatively small non-randomly missing pixels.



(a) Original (b) With 80% missing pixels (c) Reconstructed with Sensitive Mean Filtering

Figure 2. The *Inpainting by Sensitive Mean Filtering* Method achieves visually appealing results on relatively small patches of randomly missing pixels.

of the result, its biggest advantage is its speed. On the test set of twelve different images and ten different masks of randomly missing pixels (60% missing), our method is significantly faster than the other methods as is shown in Figure 6.

Despite all its advantages, one has to consider that our method is not well suited for missing pixel patches of more than 5×5 pixels as experiments with larger patches of missing pixels have shown. This is due to our method's dependence on spatial close pixels and the relatively small choice of the kernel, which for the implementation described in this paper is of size 5×5 maximally.

V. Future Work

To overcome the dependence on relatively small patches of missing pixels, we also developed a method called *Inpainting by Clustering*. It is based on the idea that within an image, there exist many similar areas. In this method, an image is divided into (possibly overlapping) pixel patches of equal size which then are vectorized. Subsequently, these vectors are grouped into clusters using *k-means*. The actual inpainting is done by setting the missing pixels of a vector to the corresponding pixel values of the assigned cluster's centroid.

A set of non-scientifically conducted experiments lets us assume that this method also achieves good results regarding the error, but can by no means compete with the presented *Inpainting by Sensitive Mean Filtering* in terms of speed. Nevertheless, for applications with large patches of missing pixels where computing power is not an issue, further investigation of the *Inpainting by Clustering* method might be worthy.

Considering the presented method *Inpainting by Sensitive Mean Filtering*, it could be interesting to investigate in what extent an iterative version could improve the results for missing pixel patches of large scale. Furthermore it should be investigated if border detection could help to improve the algorithm on images with hard transitions, e.g. our *Comics*

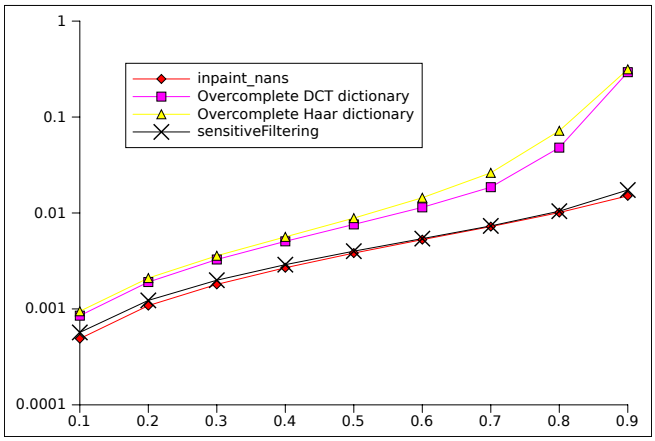


Figure 5. Error (MSE) depending on the rate of missing pixels. Generated on the test set of twelve images and ten randomly generated masks of missing pixels for every rate of missing pixels.

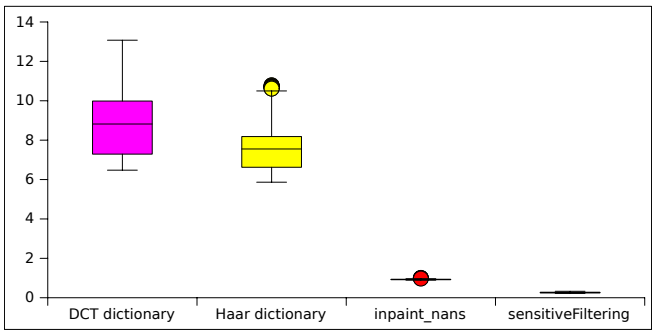


Figure 6. Time in seconds depending on the inpainting method. Generated on the test set of twelve images and ten randomly generated masks of 60% missing pixels. The means are 9.1 (DCT dictionary), 7.7 (Haar dictionary), 0.93 (inpaint_nans) and 0.26 (sensitiveFiltering) seconds.

test set. Especially for masks with large missing patches, an algorithm with an adaptive kernel size might also be worth to consider.

VI. Summary

The experiments for small patches of missing pixels (less than 5×5) have shown, that our method *Inpainting by Sensitive Mean Filtering* is superior to the methods *Inpainting with Overcomplete Dictionaries* and *Inpainting NaNs* in terms of speed, with no losses in quality. Due to the relatively fast computation time, our method is in particular an eligible candidate for applications with high demands on speed or little computing power.

Our method also shows that for small areas of missing pixels, a simple algorithm can achieve very good results.

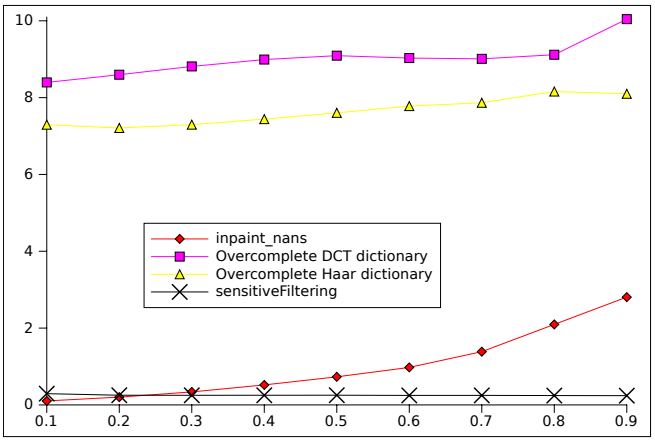


Figure 7. Time in seconds depending on rate of missing pixels. Generated on the test set of twelve images and ten randomly generated masks of missing pixels for every rate of missing pixels.

References

[1] M. McDonnell, "Box-filtering techniques," *Computer Graphics and Image Processing*, vol. 17, no. 1, pp. 65 – 70, 1981. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0146664X81800093

[2] W. K. Pratt, *Digital image processing*. Wiley, 1991.

[3] J. D'Errico, "inpaint_nans," 2006. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=4551&objectType=file

[4] J. M. Buhman and E. Lomakina, "Computational intelligence lab - series 11, may 11th, 2011 (sparse coding)," 2011. [Online]. Available: http://cil.inf.ethz.ch/material/exercise/series11.pdf

[5] A. B. M. Aharon, M. Elad, "The k-svd: An algorithm for designing of overcomplete dictionaries for sparse representation," vol. 54, no. 11, pp. 4311–4322, 2006.

[6] S. Christen, "Dictionary learning for super-resolution," 2010. [Online]. Available: http://people.ee.ethz.ch/~simonch/lect/SA_DL/report_SA_Dl_for_SR_print.pdf

[7] R. Rubinstein, M. Zibulevsky, and M. Elad, "Learning sparse dictionaries for sparse signal approximation," 2009. [Online]. Available: http://www.cs.technion.ac.il/~ronrubin/Publications/sparsedict-rep.pdf

[8] R. Rubinstein, "Ksvd-box v13." [Online]. Available: http://www.cs.technion.ac.il/~ronrubin/software.html