Local Entropy Based Image Reconstruction

Carsten Croonenbroeck

European University Viadrina Frankfurt (Oder)
Department of Business Administration and Economics
Discussion Paper No. 312
January 2012
ISSN 1860 0921

Local Entropy Based Image Reconstruction

Carsten Croonenbroeck, European University Viadrina Frankfurt (Oder)
Chair of Economics, in particular Macroeconomics
Postfach 1786
15207 Frankfurt (Oder)
GERMANY

January 29, 2012

Abstract

This article presents a local entropy based image reconstruction algorithm that performs quite well in cases where there is distortion in an image. If the "wanted" image information is still available but distributed over two or more distinct images, the algorithm can collect the required information from the set of the images given. Instead of inferring pixel data information from the remainder of one single image, the algorithm provides a decision rule on what information from which one of the set of given images to actually use in order to create a new, (ideally) distortion-free image.

1 Introduction

In real world applications there are several occasions in which there are two or more versions of a specific image available, all of them very similar to each other. For example, those pictures may be images taken from the same scene using a tripod mounted camera. Willing to capture a landscape for example, these images all show the same background (the landscape), but unwanted distortion in the foreground (e.g. pedestrians passing by). The image artist needs to digitally remove those pedestrians from the image by reconstructing the background information from other images. This process can hardly be automatized, the images need to be "montaged" manually.[1]

If the actual information is available but distributed throughout two or more images, local entropy based image reconstruction algorithms have proven to be capable to automatically resolve the image that contains the actually "wanted" information. Those parts can then be combined into a new image that contains all the wanted information. After all, that algorithm can do what the montager does manually, but in an automatized way.

There are several areas of application for such an algorithm. Apart from the scenario described above, one application comes up when there are (physical) images that need to be digitized using a scanner. Typically, scanned images exhibit distortion due to dust particles all over the image. As soon as the same image is scanned twice (i.e. the second scan is applied after physically moving the image to a degree, causing the dust particles to likely be redistributed) or more times, the algorithm can gather a new, dust-free version of the wanted image information.

That latter scenario has in fact been the motivation for developing an actual

[1] However, there are methods called "Content Aware Fill" available in modern image processing software, but these methods do not reconstruct information. They just infer pixel colors from the surrounding area and thus, are limited to low-entropy texture imagery like plain blue skies or grassy floor for example.

local entropy based image reconstruction software. This article presents how the algorithm works and what its advantages and limitations are. The remainder of the text is structured as follows: Section two presents an overview of existing entropy based image processing algorithms as well as the basic idea of entropy. Section three introduces the algorithm's implementation, while section four describes the software's performance on several applications. Section five concludes.

2 Local Entropy Processing

Using entropy based methods for image processing has been a well established approach in computer science for quite some time. Skilling and Bryan (1984) used an entropy based approach on astronomic data. In more recent research, German et al. (2005) used entropy analysis for high dynamic range image processing. Yan et al. (2003) used entropy calculations for grey-scale image clarification required in the field of medical imagery.
Basic tools for entropy based image processing are implemented in a variety of software applications. For example, Gonzalez et al. (2003) describe these tools' implementation in MATLAB.

In dissociation from the term "Entropy" as it is used in physics (thermodynamics), in the field of information science the term is sometimes entitled as "Shannon-Entropy", because it was first introduced by Shannon (1948). Entropy, in that usage of the term, is defined as the expected information content of an event. If the probability for an event x_i is $p_i(x_i)$, then the entropy can be calculated as $H = -\sum_{i=1}^{l} \log(p_i(x_i))$ or, in slightly different notation,

$$H = \sum_{i=1}^{l} \log(p_i(x_i)^{-1}). \tag{1}$$

In usual applications, the logarithm is taken to the basis of two due to easy handling of binary information content. For observed data, probabilities can be interpreted as relative frequencies of an event. For example, the entropy of two given vectors $X = (A, B, C, D, E)$ and $Y = (A, B, A, A, C)$ is calculated as $H_X = \log_2(5) + \log_2(5) + \log_2(5) + \log_2(5) + \log_2(5) = 11.6096$ and $H_Y = \log_2\left(\frac{5}{3}\right) + \log_2(5) + \log_2\left(\frac{5}{3}\right) + \log_2\left(\frac{5}{3}\right) + \log_2(5) = 6.8548$. Obviously the information content of vector X is larger than that of vector Y.

3 Simple Entropy Comparison as a Decision Rule

When processing image data (pixels), local entropy of a single pixel can be calculated in the way section 2 describes. This is the framework:
Given a pixel (its color, to be precise) at position $x = i, y = j$ inside the bitmap, a data vector needs to be defined. The vector contains the color information of the nearby pixels. For example, to calculate the horizontal local entropy of the pixel $P_{i,j}$, the data vector D will be defined as $D = (c(P_{i-1,j}), c(P_{i,j}), c(P_{i+1,j}))$, where $c(P_{x,y})$ is the color of the pixel at position x, y. To calculate the vertical local entropy, D will be defined as $D = (c(P_{i,j-1}), c(P_{i,j}), c(P_{i,j+1}))$. For the surrounding entropy, both approaches need to be combined. In addition, the positions $(x = i - 1, y = j - 1), (x = i + 1, y = j - 1), (x = i - 1, y = j + 1)$ and $(x = i + 1, y = j + 1)$ can be taken into account as well. Furthermore, an even wider measure can be defined by not only taking one neighboring pixel, respectively, into account but using two, three or more neighboring pixels in each direction.
For the above described approach of one pixel per direction, the pixel under investigation is in the center of the surrounding eight pixels, so the data vector has the length of nine. If the number of pixels taken into account per direction is k, then the number of elements in the data vector D can be calculated using

$$l = (2k + 1)^2, \tag{2}$$

where l symbolizes the length of the vector D. If the number of pixels per direction (k) is two, then $l = 25$ and in the case that $k = 3$ it follows that $l = 49$.
Having defined the surrounding data vector of a pixel, the entropy of the vector D can be calculated easily using equation (1). This concludes the framework – It returns a scalar as a measure for the surrounding entropy of a pixel. The process described can then be run for all pixels in the bitmap.[2]

The most simple case is that there are just two images. Unless explicitly noted, in the following text it is assumed that there are just two images without loss of generality. The images are denoted as I_1 and I_2, while the new image the algorithm generates is denoted as I_z.
According to the framework introduced above, the algorithm needs to run

[2]The nonexisting surrounding pixels for pixels near the borders of the bitmap are adaptively dismissed: The respective data vector's length changes accordingly.

through all pixels of the bitmaps and for all given images in the set simultaneously. For each pixel in the bitmaps, as a first step the software checks if the pixels currently investigated differ between both images ($P_{x,y}(I_1) \neq P_{x,y}(I_2)$). If not, the algorithm does not need to compute the entropy and instantly writes the pixel information into image I_z: $P_{x,y}(I_1) \rightarrow P_{x,y}(I_z)$.
If the pixels in fact differ, the algorithm generates the local data vectors D for both images (D_1 and D_2) and calculates their entropies using equation (1): H_1 and H_2. Then, the decision rule is quite easy. Pick the pixel that returned the larger local entropy and write it into the resulting image: $P_{x,y}(I_m|H_m \geq H_n) \rightarrow P_{x,y}(I_z)$, or, for a set of q images,

$$P_{x,y}(I_m|H_m = \max\{H_1, \ldots, H_q\}) \rightarrow P_{x,y}(I_z)\,. \tag{3}$$

Using this algorithm the image I_z is being built up successively and should contain the distortion-free version of the image in the end.

4 Real World Application

As a praxis test, several distortion filters have been applied to two versions of the same image. In image 1 there are these filters:

1. Mosaic,
2. text,
3. solid color,
4. random shapes,

while in image 2 there are

5. solid border,
6. blur and
7. overbright blur.

[Insert Figure 1 about here]
[Insert Figure 2 about here]

Filters 1 to 5 are clearly low entropy filters and are expected to be handled quite well. Filters 6 and 7 tend to increase local entropy to some degree at certain locally bounded areas. These filters are expected to require comparably large lengths of the data vectors, i.e. the number of neighboring pixels per direction (k) needs to be large to capture the larger local entropy in a wider area around the investigated pixel.

As a measure for quantitative performance evaluation, MSE (Mean Squared Error) and, derived from that, PSNR (Peak Signal to Noise Ratio) have been chosen. If M is the set of pixels that actually differ between the two problem images I_1 and I_2 and m is the number of elements of M, then

$$MSE = \frac{1}{m} \sum_{P_\mu \in M} \left(P_\mu \left(I_z \right) - P_\mu \left(I_R \right) \right)^2 , \tag{4}$$

where $P_\mu \left(I_z \right)$ is the μ^{th} problem pixel in the resulting (processed) image I_z and $P_\mu \left(I_R \right)$ is the corresponding pixel in the ex ante distortion free reference image. Accordingly,

$$PSNR = 20 \cdot \log_{10} \left(\frac{\iota}{\sqrt{MSE}} \right) \tag{5}$$

is the Peak Signal to Noise Ratio, where ι denotes the maximum signal intensity: For RGB images (Red, Green, Blue), $\iota = 255$ per color channel. The unit, in which PSNR is measured, is Bel or Decibel, dB.

Figure 3 presents the result for $k = 7$. Filters 1 to 5 are entirely removed, while there are still minor remains of the blur filter (6) and severe remains of the overbright blur filter (7). For this result, $MSE = 1,036.9624$ and $PSNR = 17.97$ dB.

Increasing the number of neighboring pixels k to 25, the remains of the overbright filter almost entirely disappear. Figure 4 shows the results. There are still some remains of the blur filter (6). Also, parts of the random shapes (filter 4) are not treated correctly using this comparably large k. MSE did decrease to 80.4489 and $PSNR$ increased to 29.08 dB.

[Insert Figure 3 about here]
[Insert Figure 4 about here]

Simulating the above mentioned real world application (dust particles over scanned images), two more images have been prepared. Figure 5 shows dark random dust particles while Figure 6 shows white particles. Figure 7 presents

the result after running the algorithm using $k = 1$. All particles have been removed, the image is entirely reconstructed. As the dust particles are added at random, the run has been repeated 20 times. Each run reconstructed the image entirely, all 20 results were binary-identical. Therefore, for each run, $MSE = 0$ and, by definition, $PSNR = \infty$ dB.

[Insert Figure 5 about here]
[Insert Figure 6 about here]
[Insert Figure 7 about here]

5 Conclusion

Local entropy data as a basis for image reconstruction has proven to be a useful tool for given real world applications. Using minor enhancements, the method can be applied to even more applications, for example watermark removal (by using a local *minimum* entropy criterion) or for the treatment of compression artifacts.

It is crucial that the user is able to apply several runs using differently large local entropy areas (k) in order to find the best trade-off of image reconstruction performance for different kinds of distortion. One problem of this could be the large computation time impact of large values of k: If, as for the example in section 4, $k = 25$ it follows that $l = 2,601$. Given an image of the dimension 1,600 x 1,200, this results in almost 5 billions of required computations. However, the algorithm is perfectly qualified for massive parallelization: Instead of processing one pixel after the other, arbitrarily many pixels can be processed simultaneously as there is no interdependence between local entropy of pixels, even if pixels are close by. Hence, the algorithm can be run on multiple CPU cores simultaneously or on the GPGPU chips of modern video cards.[3]

[3]GPGPU = General Purpose computation on Graphics Processing Unit.

References

A. German, M.R. Jenkin, and Y. Lesperance (2005), 'Entropy-based Image Merging', Technical report, CRV '05 Proceedings of the 2nd Canadian conference on Computer and Robot Vision.

R.C. Gonzalez, R.E. Woods, and S.L. Eddins (2003), *Digital Image Processing Using MATLAB*, Prentice Hall, New Jersey.

C.E. Shannon (1948), 'A Mathematical Theory of Communication', *Bell System Technical Journal* **27**, pp. 379–423.

J Skilling and R. K. Bryan (1984), 'Maximum Entropy Image Reconstruction - General Algorithm', *Monthly Notices of the Royal Astronomical Society* **211**, no. 1, p. 111ff.

C. Yan, N. Sang, and T. Zhang (2003), 'Local Entropy-Based Transition Region Extraction and Thresholding', *Pattern Recognition Letters* **24**, p. 2935ff.

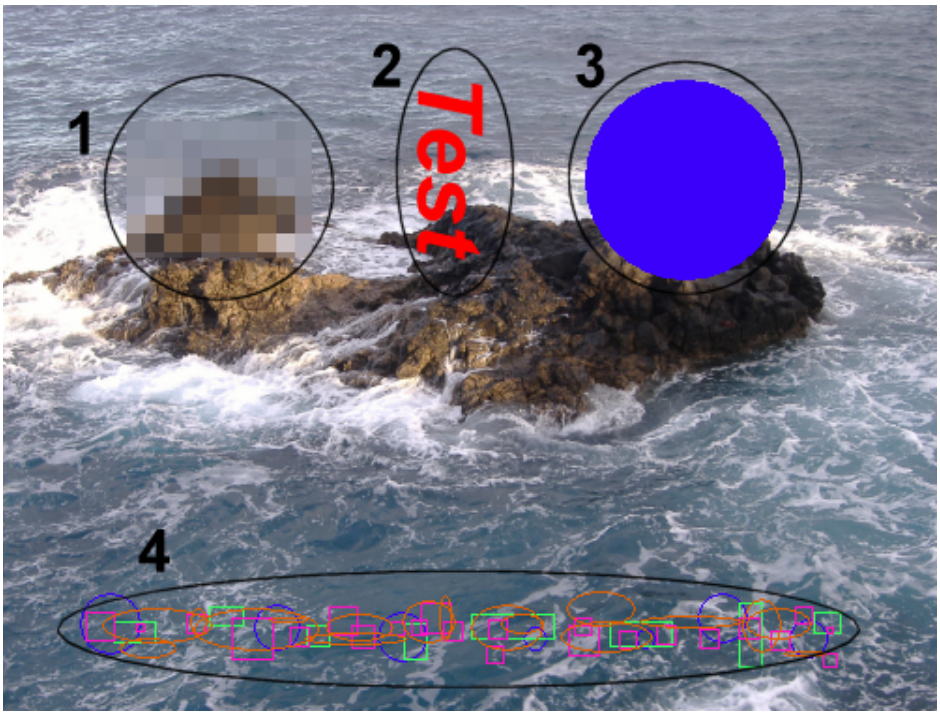

Figure 1: Example image A, I_1.

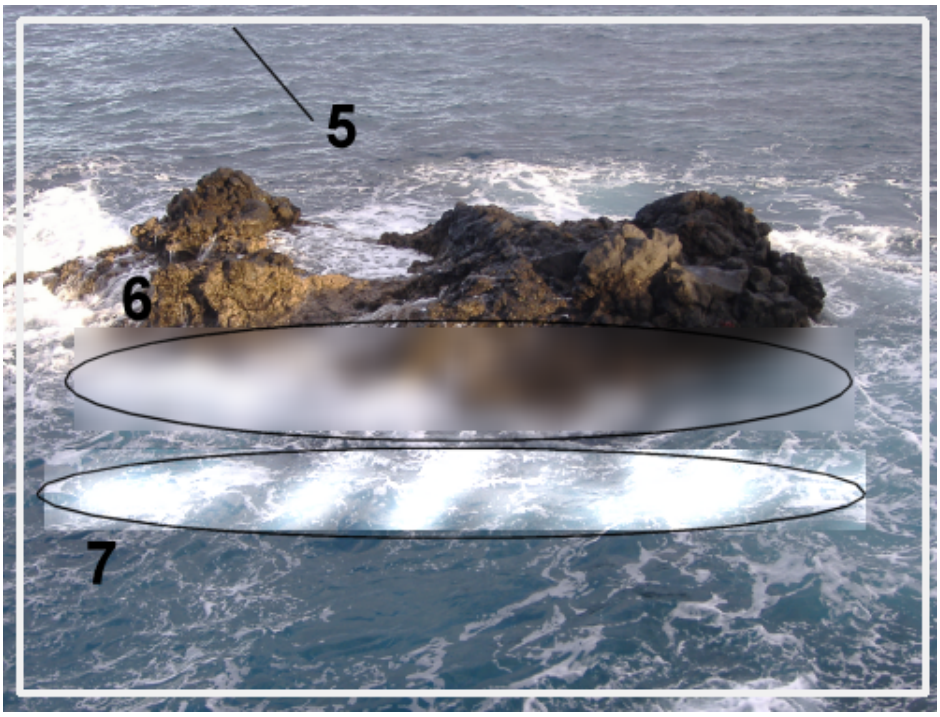

Figure 2: Example image A, I_2.

Figure 3: Example image A, $I_z, k = 7$.

Figure 4: Example image A, $I_z, k = 25$.

Figure 5: Example image B, I_1.

Figure 6: Example image B, I_2.

Figure 7: Example image B, $I_z, k = 1$.