Image Deblurring and Noise Reduction in Python
TJHSST Senior Research Project
Computer Systems Lab 2009-2010

Vincent DeVito

June 16, 2010

Abstract

In the world of photography and machine vision, blurry images can spell disaster. They can ruin an otherwise perfect photo or make it impossible for a computer to recognize the image or certain components of it for processing. The best way to counter this without taking another, clearer picture is to utilize deconvolution techniques to remove as much blur as possible. That is the design of this project. My plan was to first design a program that takes an image, blurs it using a known blur kernel, then deblurs it to reproduce the original image. Then I created a program to take the noisy deblurred image and smooth it using noise reduction. I used Python as my programming language and the .pgm uncompressed image format. My success was be measured simply by how much the output (deblurred) image matches the input (original) image.

Keywords: deblurring, deconvolution, image processing, noise reduction

1 Introduction and Background

The goal of this project was to create a program that can take an image input that has been blurred and to employ image deblurring techniques to restore the image and create a sharp, more recognizable output image with as few blur artifacts and noise as possible.

1.1 Previous Research

1.1.1 Deblurring

I found a paper regarding image deblurring and noise suppression called "Image Deblurring with Blurred/Noisy Image Pairs" by Lu Yuan, et al. that I utilized in helping me understand the techniques and algorithms that go into reducing the noise of and deblurring an image. In their research they used a blurry image with proper intensity and poor sharpness and paired it with an identical picture with good sharpness but poor intensity and riddled with noise to create a sharp, correct intensity output with few or no artifacts left in the output image.

Another paper[1] I read discusses an algorithm that the group of researchers discovered that allows for a mostly accurate estimation of the blur kernel, or the function through which the pixel values of the image are blurred. Their algorithm takes four inputs: the blurry image, a section of the image that has a good sample of blurring (in case the image is not uniformly blurred), if the blur is estimated to be more horizontal or more vertical, and the estimated size of the blue kernel. Given these inputs, their algorithm can sufficiently estimate the blur kernel such that the image, which was captured using poor technique with a standard, off-the-shelf camera, is satisfactorily deblurred with few artifacts after deconvolution. Any artifacts that are left can generally be removed by an experienced photo editor.

1.1.2 Noise Reduction

Noise reduction is the last step in my deblurring process. I've learned in previous computer science courses that Gaussian smoothing (replacing every pixel with a weighted average of its surrounding pixels) can reduce noise by averaging it with surrounding color values. The problem exists, though, that it smooths the image and therefore softens its edges, reducing the desired effect of the deblurring process. To learn how to reduce noise and maintain edge sharpness, I read a paper by German scientist Dr. Holger Adelmann

[1] Source: Fergus

entitled "An edge-sensitive noise reduction algorithm for image processing." In his paper, Adelmann discusses using a 5x5 neighborhood to determine in which of four directions the edge that pixel is located on is oriented. Then, using that information, Gaussian smoothing is applied using a kernel that matches the direction of the edge, instead of the normal 3x3 square kernel. The 5x5 analyzing kernels and 3x3 Gaussian smoothing kernels can be found below in Figure 1. By using edge sensitive kernels, edge sharpness is maintained while still reducing noise.

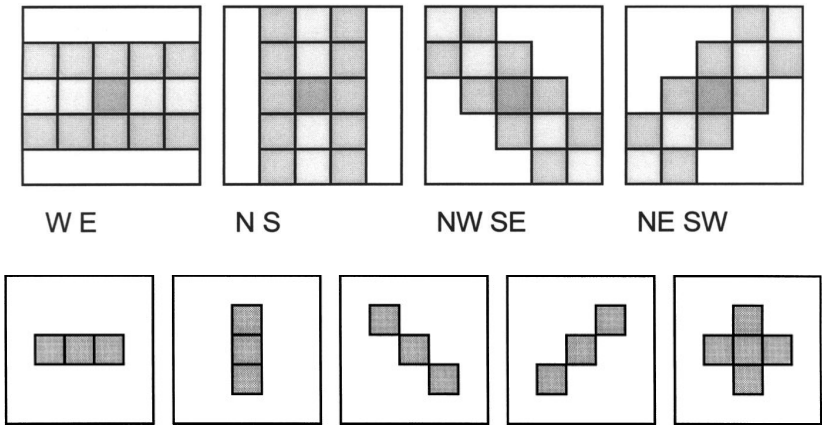


Figure 1. The 5x5 neighborhoods (above) used to detect the edge orientation about that pixel and the 3x3 kernels (below) that correspond with those orientations. The fifth kernel (flat pattern) is used when no edge is detected.

1.1.3 Other Research

Through my own work I have accrued a detailed understanding of basic and intermediate image processing techniques and algorithms from various online worksheets and lessons at http://homepages.inf.ed.ac.uk/rbf/HIPR2/wksheets.htm. I plan to use these techniques to help me code and understand the more complex concepts behind image deblurring and the intermediate steps involved. For example, I have extensively used the section referring to the Fourier Transform, located here: http://homepages.inf.ed.ac.uk/rbf/HIPR2/fourier.htm.

2 Development

2.1 Project Design

I used the programming language Python to write the code for this

project. I decided to use Python because of its simplicity and adaptability. As for the images, I used the uncompressed, grayscale .pgm image format. This allowed me to confirm the accuracy of the outputs because of the uncompressed nature of the .pgm format, which means that the image information doesn't need to be altered before being saved. Also, it is much easier to code using the .pgm format since it can be read in as and saved as a string without using any packages or software, also making it more reliable.

Also, for the purposes of this project, I chose to focus on motion blur convolution and deconvolution. This doesn't affect the image to be deblurred, but rather the design of the kernel used to blur and deblur the image. I found that my chosen method of image division deconvolution doesn't work correctly on other forms of blur, such as Gaussian blur and potentially out-of-focus camera blur.

The first step in this project was to artificially blur an input image using a known and given blur kernel. This is accomplished by converting both images to the frequency domain, using the Fast Fourier Transform (FFT), point multiplying the two images, then converting them back to the spatial domain using the Inverse Fast Fourier Transform (IFFT). This is known as convolution.

The next step was the deconvolution algorithm that, when given an image and its known blur kernel, could deblur the input image. This is fairly straightforward and involves the reverse of the aforementioned convolution algorithm. This is done by instead point dividing the blurred image by the blur kernel in the frequency domain.

The final step was to take the noisy image that was acquired through the deconvolution process and apply a noise reduction algorithm. This noise reduction algorithm attempted to remove as much of the noise from the deblurred image as possible, while still maintaining sharpness and clarity.

2.2 Testing

My project's success was measured by its ability to take an artificially blurred image and return it to its original, sharp quality. I tested my project's adaptability and thoroughness by running a series of tests that entailed attempting to deblur images of different contrast and content with varying magnitudes and directions of blur. This tested my program's ability to repair images regardless of image content, or magnitude or direction of blur distortion, although there will obviously be an upper limit to the amount of blur that can plausibly be

removed. An example of what would be deemed a successful run is illustrated below:

Figure 2. The image on the left is an example of a blurry image input, with a particularly blurry section highlighted. The image on the right is the same image, with the blurriness drastically reduced due to deconvolution.[2]

2.3 Theory

2.3.1 Fourier Transform

The Fourier Transform is heavily involved with image convolution and deconvolution because it allows for greater speed and simpler code. The Fourier Transform converts values in an array from the spatial domain to the frequency domain using a sum of complex numbers, as given by the equation:

$$F(x) = \sum_{n=0}^{N-1} f(n) e^{-j2\pi(x\frac{n}{N})}$$

The 2-Dimensional Discrete Fourier Transform (DFT) does this using a matrix or 2D array of values and uses a nested sum:

$$F(x,y) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f(m,n) e^{-j2\pi(x\frac{m}{M}+y\frac{n}{N})}$$

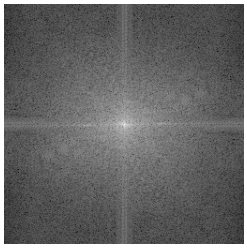
Since the 2-Dimensional Discrete Fourier Transform uses a nested sum, it can be separated to create two 1-Dimensional Fourier Transforms in a row, first in one direction (vertically or horizontally), then in the other direction.

$$P(k,b) = \frac{1}{N}\sum_{a=0}^{N-1} f(a,b)\ e^{-\iota 2\pi\frac{ka}{N}}$$

$$F(k,l) = \frac{1}{N}\sum_{b=0}^{N-1} P(k,b)\ e^{-\iota 2\pi\frac{lb}{N}}$$

This is known as the Fast Fourier Transform (FFT) and runs significantly faster than the DFT, since the DFT has a runtime of $O(n^2)$ and the FFT has a runtime of $O(nlog_2 n)$. The following is an example of a picture being converted from the spatial domain to the frequency domain via the Fourier Transform.

[2]Pictures from Source 3

 is then transformed to 

The reason the FFT is so important to image convolution and deconvolution is that it takes long iterative algorithms and turns them into simple point arithmetic. For example, image convolution becomes as simple as taking the Fourier Transform of the image and the blur kernel (known as the Point Spread Function (PSF) after transformation), transforming them to the frequency domain and point multiplying the two images. Then the two images can be converted back to the spatial domain by the Inverse Fourier Transform, given by

$$f(m,n) = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} F(x,y)e^{j2\pi(x\frac{m}{M}+y\frac{n}{N})}$$

and the result will be a blurry (convoluted) image. To reverse this process and deconvolute the image, assuming the blur kernel is known, is as simple as point dividing the transformed image by the PSF, instead of multiplying.

The IDFT can also be separated and turned into the Inverse Fast Fourier Transform (IFFT). When using the IDFT or IFFT, though, the values need to be the full complex numbers from the Fourier Transform. This means that the IFFT cannot be performed on an image that is transformed to display the magnitude or the phase of the Fourier Transform. This is demonstrated below in Figure 3.

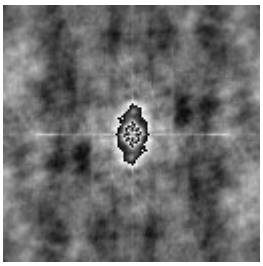




Figure 3. This shows the original image, the result of the IFFT using only the magnitude of the Fourier Transform output, and the result of the IFFT using only the phase of the Fourier Transform output.[3]

[3]All images from Source: Young

2.4 Noise Reduction Algorithm

As I mentioned in my previous research on noise reduction algorithms (section 1.1.2), I learned previously that Gaussian smoothing can be employed to reduce the noise in an image. However, this softens the image (as seen below in Figure 4) and therefore is not a desirable approach.

Figure 4. Using Gaussian smoothing produces a cleaner image, but also an undesirably soft/blurry output.

To start, I took the basic concept of Gaussian smoothing (weighted average of the 3x3 neighborhood for each pixel) and adapted it to better handle noise. For each pixel, I took the average of all the pixels in the 3x3 neighborhood. Then, I determined which of the 9 pixels in the neighborhood had values that fell within 15 of the average and took the average of those to use as the new pixel value. This works by establishing a rough range for what that pixel value should be by analyzing the surround pixels in conjunction with its own value. However, since noise is usually an extreme value (high or low), any noise pixels within that 3x3 neighborhood won't fall within the range, and therefore will be excluded from the final average used to calculate the new pixel value. In the event that none of the neighboring pixels fall within the 30 point range, the value of the pixel will remain unchanged. This reduces the noise much better than a simple weighted average of the 3x3 neighborhood.

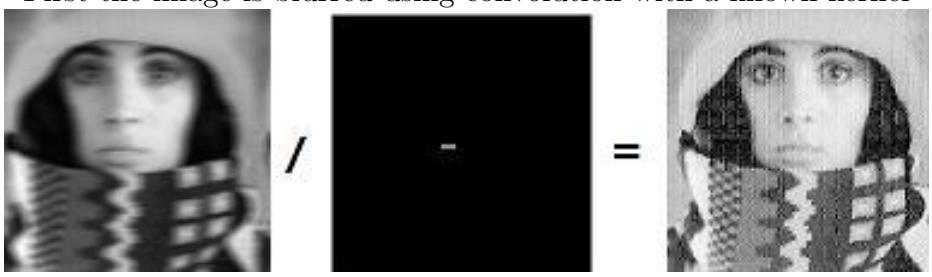
The problem still exists, however, that the final output image is less sharp than the original, noisy image. To correct this problem, I added an unsharp sharpening filter to my algorithm. The unsharp filter works by subtracting the smoothed version of an image from the original image, thus producing an image with only the sharp edges from the original picture. The edge values are then scaled (between .1 and .7; .6 in my algo-

rithm) and added back to the original image, sharpening the edges of the picture. By supplementing my dynamic Gaussian smoothing algorithm with an unsharp filter, I designed a successful noise reduction algorithm that clears up a lot of noise without losing very much sharpness in the image.

3 Results

I completed my project as much as I had originally expected. My convolution program works correctly and can convolute any[4] image using any blur kernel. My deblurring program can also deblur that image, as long as the aforementioned blur kernel is an acceptable motion blur kernel, as mentioned in my project development, section 2.1. A successful run is outlined below:

First the image is blurred using convolution with a known kernel



Then it is deblurred using deconvolution with that known kernel.

I have also developed a successful noise reduction algorithm of my own. To test its ability to reduce noise, I compared its output to the outputs of the basic Gaussian smoothing algorithm as well as the edge-sensitive noise reduction algorithm I mentioned previously in section 1.1.2. My results were surprising, as my algorithm seemed to surpass the abilities of the other two algorithms. This wasn't surprising for the Gaussian

[4]For best results, I used a square image of size 128x128

smoothing algorithm, but I was impressed when my results were compared to those of the published edge-sensitive algorithm. The results are outlined below:

The noisy, deblurred image.



Figure 5. The outputs from the three tested noise reduction algorithms (left to right): Gaussian smoothing, the published edge-sensitive noise reduction algorithm, and my noise reduction algorithm.

As is evident in Figure 5, the Gaussian smoothing algorithm clears up some of the noise, but softens the edges. The edge-sensitive noise reduction algorithm takes care of this, but at the loss of not reducing as much noise. My algorithm combines the two and produces a clearer, smooth output that also has sharp edges.

3.1 Limitations

Even though I have deemed my project successful within the goals I originally set, there are obviously limitations. First and foremost, my deconvolution algorithm only works with some kernels that fall within a certain type of blur kernel. Anything outside that category and my program will just return unreadable noise. Another limitation is simply that my program needs the known

blur kernel to deblur the image; it doesn't have the ability to estimate the kernel. This is a severe limitation as it can only be used to deconvolute artificially blurred images where the kernel is known, it has no real world applicability yet.

Another limitation of my project is the vast difference in color/intensity between the original image and the final, corrected image. In theory and ideally these two images will be identical, however, it can be seen from Figure 6 that they are very dissimilar in intensity.

Figure 6. The deblurred and noise reduced image (left) is much brighter than the original image (right).

This also affects the contrast of the image. I speculate that this happens because of the multiple transforms the image undergoes throughout the process (one logarithmic transform for every convolution and deconvolution). This could potentially pose a problem in machine vision applications, though I expect only because of the reduced contrast, not the change in color/intensity, which could possibly be adjusted using another, linear contrast stretching transform.

3.2 Scope

The scope of this project is rather narrow, but important. It pertains only to blurry images, and in this case only those blurred using a motion blur kernel, but the concept is a rather large problem in the worlds of image processing, photography, and machine vision. In photography, blurry images are undesirable because they lack sharpness or clarity and in machine vision, blurriness can make an image indecipherable by the computer or render certain processes ineffective, such as edge detection. These results can help ex-

plore the worlds of image deconvolution and noise reduction, within the larger world of image correction.

4 Conclusions

I managed to transform and inversely transform an image using the Fourier Transform with complete success, as well as successfully blur an image using convolution. I also was able to deconvolute that blurry image with some consistency, as long as a motion blur kernel was used. My noise reduction algorithm worked successfully and arguably surpassed the abilities of the published edge-sensitive noise reduction algorithm I found in my research.

4.1 Future Work

There is a lot of room for future work on this project since and in this field in general. The next area of research would be into blur kernel and point spread function types so that the deconvolution process can be made more adaptive and generalized, since my program is very restrictive and specific. The largest area for future research for this project is the one that is most applicable to the real world and also the subject of much study in the computer science community. This is the topic of blind deconvolution, which estimates the blur kernel from an image in which it is not known (e.g. a naturally blurred image acquired through poor image capture) and then deconvolutes the image based on this estimate. Research is ongoing on trying to find the most efficient and adaptive method of estimating the blur kernel. Lastly, it would be beneficial to look into color or contrast correction algorithms in an attempt to try and match the output images color intensity and contrast with the original image so that the output is as accurate as possible.

References

[1] Adelmann, H. G. (1999, March). An edge-sensitive noise reduction algorithm for image processing. *Computers in Biology and Medicine, 29*(2), 137-145.

[2] Brayer, J. M. (n.d.). Introduction to the Fourier transform. In *Topics in human and computer vision.* Retrieved from University of New Mexico Department of Computer Science website: http://www.cs.unm.edu/ brayer/vision/fourier.html

[3] Convolution and deconvolution using the FFT. (1992). In *Numerical recipes in C: The art of scientific computing* (pp. 538-545). Cambridge, Massachusetts: Cambridge University Press.

[4] Fergus, R., Singh, B., Hertzmann, A., Roweis, S. T., & Freeman, W. T. (2006, July). Removing camera shake from a single photograph. *ACM Transactions on Graphics, 25*(3), 787-794. doi:10.1145/1141911.1141956

[5] Fisher, R., Perkins, S., Walker, A., & Wolfart, E. (2000, October). *Hypermedia Image Processing Resource (HIPR2)* [Image processing learning resource]. Retrieved from http://homepages.inf.ed.ac.uk/rbf/HIPR2/wksheets.htm

[6] Smith, S. W. (1997). A closer look at image convolution. In *The scientist and engineer's guide to digital signal processing* (pp. 418-422). Retrieved from http://www.dspguide.com/

[7] Young, I. T., Gerbrands, J. J., van Vliet, L. J. (n.d.). Properties of Fourier transforms. In *Image processing fundamentals*. Retrieved from http://www.ph.tn.tudelft.nl/Courses/FIP/noframes/fip-Properti-2.html

[8] Yuan, L., Sun, J., Quan, L., & Shum, H.-Y. (2007, July). Image deblurring with blurred/noisy image pairs. *ACM Transactions on Graphics, 26*(3). doi:10.1145/1276377.1276379