Image Deblurring Techniques
TJHSST Senior Research Project
Computer Systems Lab 2009-2010

Vincent DeVito

April 8, 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 is to first design a program that takes an image, blurs it using a known blur kernel, then deblurs it to reproduce the original image. After that I will attempt to create a program to determine the blur kernel of a naturally blurred image. I will use Python and a package called Python Imaging Library that will allow me to utilize multiple image formats. My success will 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 is to create a program that can take an image input that has been blurred (first artificially, and later hopefully by poor image capture) 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

So far I have found a paper regarding image deblurring and noise suppression called "Image Deblurring with Blurred/Noisy Image Pairs" by Lu Yuan, et al. that I plan to utilize 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 have 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.

1.2 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/

[1] Source 2

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 will use the uncompressed, grayscale .pgm image format. This will allow me to confirm the accuracy of the outputs because of the uncompressed nature of the .pgm, 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 and saved as a string without using any packages or software.

The first step in this project is 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 is the deconvolution algorithm that, when given an image and its known blur kernel, can 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. After this step, I can attempt to add a noise reduction filter to remove any excess noise in the image and further sharpen and clarify the image.

The final step would be to design a program that can estimate the blur kernel. This program would first be tested on blurry images with known blur kernels to make sure that the estimated blur kernel is similar to the actual blur kernel used. Then, after deemed acceptable, this program would estimate the blur kernel of a naturally blurred image with an unknown blur kernel, which can then be used to deblurring that image, hopefully to an acceptable level. This last step is a large one and currently under a lot of research since blind deconvolution, as it is known, is quite difficult. Currently, perfecting this step is considered the "holy grail" of image deblurring and likely unobtainable with my limited knowledge of the subject.

2.2 Testing

My project's success will be measured by its ability to take an artificially blurred image and return it to its

original, sharp quality. I will test my projects adaptability and thoroughness by running a series of tests that will entail attempting to deblur images of different contrast with varying magnitudes and types of blur. This will test my programs ability to repair images regardless of image content, or magnitude or type of blur distortion, although there will obviously be an upper limit to the amount of blur that can plausibly be removed, as well as this algorithm not working very well with all types of blur. An example of a successful run is illustrated below:

Figure 1. This is an example of a blurry image input, with a particularly blurry section highlighted.

Figure 2. This is the same section from Figure 1, but with the blur 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

[2]This and the above are from Source 2

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:

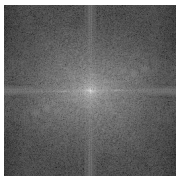
$$F(x,y) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f(m,n) e^{-j2\pi(x\frac{m}{M}+y\frac{n}{N})}$$

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.

 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.

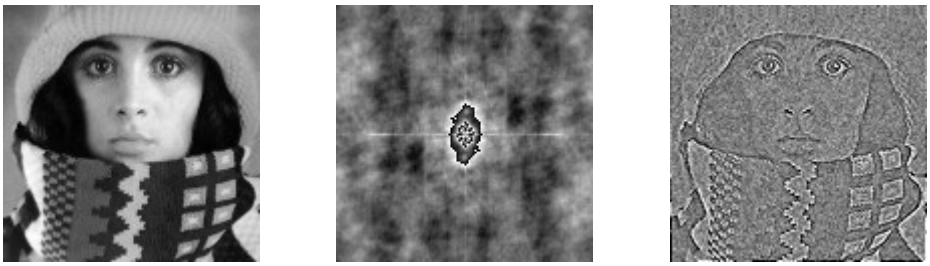
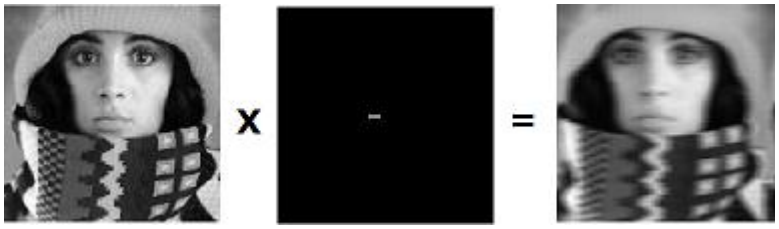


Figure 3. This shows the original image, the result of the IFFT using only the magnitude of the Fourier Transform output, and the result of the IFFT using only the phase of the Fourier Transform output.[3]

3 Results

Thus far I have confirmed my FFT and IFFT are working correctly by being to produce approximately the same output image as the input image after being converted to and from the frequency domain. Also, my convolution program is working correctly and can convolute any[4] image using any blur kernel.



3.1 Errors

My program has not been entirely debugged and has some running errors. My current roadblock is that my deconvolution algorithm doesn't work for most blur kernels, even of only one specific type. For example, take the blur kernel in the above example. For that kernel I can achieve a roughly clear deconvoluted output as

[3] All images from Source 5

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

seen below in Figure 3. This output has some noise but is easily recognizable and not blurred. However, using the same kernel, with the white line shifted over one or two pixels, the convolution algorithm produces a similar blurry image, but the deconvolution algorithm produces extremely high values that are very tightly grouped, with one outlying low value. This presents an issue, since, when logarithmically or linearly transformed, the values are all still high and tightly grouped, producing an all white image, with the exception of the one outlying black pixel. I have not yet determined what is causing this strange output or why it changes suddenly with minor variations in the blur kernel, but I intend to focus on this problem. I believe the issue might be in the type of kernel I am using and how that plays a role in deconvolution. I speculate that it is possible that by changing the blur kernel slightly, I may have altered what type of kernel it is and therefore made the deconvolution method functional, though this seems farfetched.

Figure 4. A noisy output of the deconvolution algorithm using the blur kernel in the above example.

3.2 Scope

The scope of this project is rather narrow, but important. It pertains only to blurry images, but this 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. Blind image deconvolution is also a very large area of research, specifically on the subject of estimating unknown blur kernels to increase adaptability for deblurring programs to deblur any image.

4 Conclusions

I have managed to transform and inversely transform an image using the Fourier Transform with complete success, as well as successfully blurring an image using convolution. As of yet, I continue to work on the deconvolution algorithm to attain complete success, instead of the current partial success.

4.1 Future Work

There is a lot of room for future work on this project since, at this point, I have yet to refine the deconvolution algorithm. The next and obvious step is to finish work on the deconvolution algorithm so that it works correctly and reliably, or at least predictably. The next and final area of 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 area of blind deconvolution, which estimates the blur kernel from an image in which it is not known and then deconvolutes the image based on this estimate. Research is ongoing in trying to find the most efficient and adaptive method of estimating the blur kernel.

References

[1] 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

[2] 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

[3] 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

[4] 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/

[5] 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

[6] 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