Wavelet Filters for Image Segmentation

For my summer research, Edward Boyda and I investigated wavelet filters for image segmentation. Image segmentation is the task of dividing a digital image into thematically coherent regions. Previously, the filters used in image segmentation have implemented sine waves, and only recently have wavelets been more of an influence. We created wavelet filters and compared them with sine wave filters, and our results were slightly better. We intend to continue our research and refine our wavelet filters, so as to have significantly better results rather than marginally better.

Human beings can segment visual imagery with ease. I can look at a satellite image of San Francisco and think, “There’s an urban part with lots of man-made things, there’s water, and there’s some forestry up north.” In doing this, I have easily segmented the image. However, despite its conceptual simplicity and the ease with which human beings perform the task, image segmentation remains one of the more stubborn problems in machine vision. That is, a computer is hard-pressed to sort out a similar image with ease and accuracy. Whereas a human being can look at a picture and immediately think, “Trees! Grassy hills! Roads and developments!” a computer might think, “Green of high intensity with rough texture. Green of low intensity with smooth texture. Grayish, reddish color with fragmented texture.” All of this is to say that human beings see an image for what it is using the semantic categories we individually develop to give meaning to it. These semantic categories that human beings use so effectively are
still unavailable to machines, possibly suggesting that machines will need real artificial intelligence to see as well as humans.

Image segmentation is extremely important in contemporary machine vision applications. It often serves as a precursor to object-detection, recognition, and reconstruction tasks, with specific applications in medical imaging and satellite-based remote sensing, to name just two. Due to its applications, the importance of extremely accurate image segmentation is massive, and it can be improved. When browsing through segmented images online, it’s easy to pick out “mistakes”. In an original image of forested area and urban area, some forestry makes its way into the urban area in the segmentation. That is to say, if half the segmented image should be colored black for urban area and the other half gray for forestry, some gray makes it into what should be completely black. Depending on the problem, a well-segmented image may be anywhere from 75-99% accurate. This leaves room for improvement, needed improvement when it comes to important applications.

Edward Boyda’s pre-summer research image segmentation program sends images through a series of computer-programmed filters that help identify texture and patterns. These filters, to make a long story short, are masks of dilated and rotated sine waves passed over the image to highlight periodic structures.
They do not effectively identify sharp, large-scale patterns. So we explored new wavelet-based filters. Wavelets are particularly well suited to identification of sharp features such as the edges and discontinuities observable in, for example, a cluster of bricks. So, using these new wavelet-based filters, we aimed to create a program that was both extremely flexible and extremely accurate. In this context, flexible means able to segment a wide variety of images with no prior knowledge. For example, it’s relatively easy and straightforward to create a successful program that knows its only type of input image will be green forestry. In this case, the program can be tailored to specialize in this type of image because the properties of all possible input images will be pretty similar (they will probably all be mostly green, for example). But we aimed to create a program that had no prior knowledge of the input image and could segment well no matter what sort of input (could be forestry, fire, water, bricks, a car, a house, a mermaid, etc.). Further, we obviously want our segmentations to be as accurate as possible. The problem is that there is a trade-off between program flexibility and accuracy, and this is what made our research unique. Generally speaking, as the variety of possible input images goes down, accuracy goes up, and vice versa, because programs can be specialized or not. But we
wanted to create a program that could accurately segment just about any image we threw at it, and do it well at that. That is to say, we wanted to overcome the inverse link between accuracy and flexibility as best we could.

However, the ultimate goal of this research was, and still is, to use our program for continental-scale segmentation of Landsat images as part of a project to monitor the changes in land cover and land use accompanying climate change. For example, the program might be used for the monitoring of deforestation, desertification, or glacial melting. Overall, our goal was and still is to create a computer program that will improve image segmentation and eventually serve to benefit the environment.

The obvious and necessary first step for this research project was learning the mathematics of wavelets. We started by reviewing Fourier analysis and the Fourier transform to give me a good analogy for what wavelets do and how they are implemented. Plus, Edward Boyda’s pre-summer research program uses Fourier methods. Once Fourier analysis was mastered, we had the exact framework for wavelet analysis; we just needed to substitute wavelet concepts for the Fourier ones. That is to say, the wavelet transform and the Fourier transform are related methodically, and only the specifics are different. To simplify, it might help to think of the equation $2x=y$. $X$ is what changes (analogous in this example to the wavelet transform versus the Fourier transform), and depending on which $x$ is used, $y$ will respond accordingly. But no matter the $x$ used, the equation (framework) and method of implementing $x$ is always the same.

\begin{align*}
    c_n &= \int_{-T/2}^{T/2} f(x) e^{-2\pi i (n/T) x} \, dx, \\
    WT_\psi \{x\}(a, b) &= \langle x, \psi_{a,b} \rangle = \int_R x(t) \psi_{a,b}(t) \, dt
\end{align*}
A Fourier transform (top left) and a wavelet transform (top right), the only difference being the sine wave (bottom left) and the wavelet (bottom right) that get used in their respective transforms.

\[ e^{-2\pi i (n/T)x_i} \]

\[ \psi_{m,n}(t) = a^{-m/2} \psi(a^{-m}t - nb). \]

So what is a wavelet transform? It might be thought of as a variation of the Fourier transform. For image analysis, which was our project, the Fourier transform basically passes sine waves over an image, and depending on how closely the image’s signal resembled the sine wave, there would be a large or small output number. Hypothetically, if the output number of one image was 10 and another was 2, the image that gave a 10 Fourier response had a signal that more closely resembled the sine wave that passed over it. Wavelet transforms can be thought of in the same way, except the waves passed over images are not limited to sine waves. They can be jagged and rough, smooth, asymmetric or not, and are not really very limited in their shape. Thus, wavelet transforms can be thought of as being like Fourier transforms that use differently shaped waves. The wavelet we chose to use most had orthogonal properties and also happened to be quite jagged and had a discontinuous slope. It looks more like a jagged mountain face than smooth, rolling hills. We had other options that resembled sine waves, but our goal was initially to test if wavelet transforms could better segment images. Thus, it wouldn’t have made much sense to use something similar to sine wave/Fourier analysis if we wanted to extract what was different and hopefully better about wavelets. Not only that, we were hoping to identify patterns and sharp features, which a smooth, continuous wavelet would not have been very effective for. Finally, Edward Boyda’s pre-summer research program already used rotated and dilated sine waves, which ended up being extremely similar to a smooth wavelet.
Wavelets...we used the top left one. It can be seen that some wavelets look more similar to the dilated and rotated sine waves above, and we tried to deviate from those.

Once we had learned the mathematical theory behind wavelets and had decided which type to use (sharp and jagged), we needed to put our knowledge into practice. And this is where wavelets differ most from Fourier analysis in their practical implementation. While in Edward’s pre-summer research program, and in Fourier signal analysis in general, the code can basically mirror the prescriptions and equations one would find in a textbook on the mathematical theory of Fourier analysis, the practical implementation of wavelet analysis varies completely with the theory we learned about in our initial education on wavelets. That is, to practically implement the Fourier transform, one just codes its exact equation. But for the practical implementation of wavelet transforms, one codes something that is completely different, and in fact, has nothing explicitly to do with wavelets. This practical implementation of wavelet transforms is called Multiresolution analysis (MRA). Basically, there are two ways of getting at wavelets: the mathematical theory and Multiresolution analysis. In mathematical theory, we work our way up logically to wavelets. It’s sort of “on purpose”. In Multiresolution analysis, we work our way down. We have Multiresolution analysis and then can prove that this method indeed implements wavelet transforms. It’s sort of “on accident”. In other words, if we were to
use a wavelet transform like Edward used a Fourier transform in his code, we would have been directly and explicitly using wavelet analysis. But we used Multiresolution analysis, which has nothing explicitly to do with wavelets. Only implicitly can it be proven that this method implements wavelet transforms. Thus, there is nothing in my code that has anything to do with a wavelet transform, but it is utilizing one implicitly.

So, using Python as our programming language, we went about implementing Multiresolution analysis, not wavelet transforms, and this is one of the great things about wavelet filters. Not only is Multiresolution analysis much simpler and easier to code than the mathematical equation for a wavelet transform, but it also allows us to use any wavelet we want by simply changing a few numbers. This is as opposed to coding an actual Fourier or wavelet transform, where we would have to change the entire framework of the code to use a new wave. Basically, we have a set of “h” values (like .845 or .224) that correspond to different wavelets. There can be really any number of “h” values, but practically speaking, there are usually four to ten “h” values for each wavelet. We mainly used a wavelet with four “h” values. Once we have our “h” values, we match them to individual pixels. So our wavelet “mask”, as I’ll call it now, is matched up to four pixels. Then, it matrix multiplies each “h” value with each pixel’s intensity value, adds all these products, and puts the value in a new Python array. Then it moves over two pixels and does the same thing, until it reaches the end of the digital image, at which point it moves down a row of pixels and does the same thing. In our program, it operates both horizontally and vertically. After it has done the entire image, it does the same thing on the new image (the new array that the matrix products were being stored in), that is, the next resolution. Hence, Multiresolution Analysis. And it does this for as many
resolutions as we so choose. Generally, we didn’t do or need more than four resolutions, and really any more after that caused our computers to crash. It’s just too much information at that point. All of this is to say that our wavelet filter is incredibly simple. We simply multiply “h” values with pixel intensity values. And if we want to use a new wavelet, we simply change the “h” values. It’s as easy as changing .845 to .224 as many times as there are “h” values for a given wavelet. Generally this is not more than ten. This method of implementing a wavelet transform is much easier and more efficient than explicitly using complex mathematical equations to do it.

This is not to say that our program was that simple or easy to create. The difficulties were in the little things. For one, storing the matrix products in a new array and then utilizing the new array as the next resolution for the program to operate on was oftentimes tricky. There were various things to keep account of that made for a complicated process, such as where exactly to store the values. Further, what if the pixel length of our digital image was not a multiple of four, the number of “h” values our wavelet had? If this was the case, we oftentimes might come to the end of a pixel row and have our mask hanging off the end of the picture, so to speak. It may only be operating on one, two, or three values, which would not produce a proper signal analysis on that part of the picture. This is called the “edge effect”. The solution, though not perfect, is to have the program automatically add as many values to the end of a pixel row as there are “h” values without a pixel partner to match up with. These imaginary pixels were given values according their nearest neighboring pixels, because implementing those values would make more sense to use than the pixels at the beginning of the row. That is, if there actually were more pixels in the row, they would likely be more similar to the pixel
values at the end of the row than the front. Using these values keeps the continuity of the
filtering for that row. Another difficulty was upsampling. Each successive resolution, due
to the mask moving over two pixels with each new matrix multiplication and only storing
one value in the new array along with operating in both the horizontal and vertical
directions, was a fourth as big as the previous resolution. That means that if we do three
resolution analyses, as was our norm, we would end up with an image one sixteenth as
big as the original. And of course, this would be useless to segment with. So we would
need to upsample, that is, make the output image as big as the input was. This was tricky
in many ways due to not only the theory of wavelet image analysis but also due to
programming. We ended up assigning each pixel value sixteen pixels in our “zoomed-in”
image. We may change this method as our research continues. Further, another issue was
program running speed. Our initial program would take a few minutes or more to run on
three resolutions, but we tweaked it enough so that it runs in a few seconds. There were
certain techniques and changes we implemented to do this, one of which was doing away
with as many Python “loops” as we could. Another difficulty was adapting Edward’s
segmentation program to the new filter program. The filter program that we made this
summer doesn’t actually do any segmenting. The segmentation is another complicated
process entirely. There are many processes (unrelated to wavelets) in the segmentation
program, such as simulated annealing and color filtering. The wavelet filter filters for
texture and patterns, but there are also color-filtering schemes that produce filtered
images to be used for segmentation. To put it probably too simply, the texture and pattern
filter outputs a few images along with the color filters, and these images are then
“superimposed” on one another, and then segmented as a whole. We also output
segmented imagery with only color features and only texture features, and these were not very well segmented. The only way to get good segmentations is by utilizing both texture and color features. But all of this (the actual segmentation program) was Edward’s domain, not mine. I focused on the wavelet filters while he focused on the segmentation program. This being said, I know he had to tweak his segmentation program along with my wavelet program so as to allow them to collaborate.

![Color features in red, green, and blue (top row) along with a couple wavelet-filtered images (bottom row) to be “superimposed” in order to segment.](image)

Overall and oddly enough, the actual bare bones of wavelet filtering were the absolute easiest part of the project, thanks to Multiresolution analysis. Simply do a little multiplication. The hardest parts were always the technical difficulties that inevitably arise when putting mathematical theory into practice. The final and most annoying difficulties of the research were the subtle, small, and minute programming quirks that either were programmed slightly wrong or demanded a better knowledge of the subtleties
of Python. Of this is included debugging. I would say that half my time spent programming was spent trying to figure out how to implement the smallest techniques into Python, or figuring out the subtlest of errors in my code. To give examples, oftentimes errors that demanded my attention for hours on end included tweaks as small as changing a zero to a one or a less than sign to a less than or equal to sign. A lot of times the issues were a bit bigger than this, but not much, so my point still stands. It oftentimes felt like trying to find a needle in a haystack, though probably a bit easier, but still the same principle. Perhaps most of the times these issues arose because modifications in the code would affect things that were not intended to be changed and not obvious to know would be changed. This is to say that a lot of times I would be looking for the needle in the wrong haystack, that is, searching for the code errors in the wrong places. How frustrating, especially when I would finish the logic and mathematics of the code in an hour and then think I was just about done, only to spend two more hours searching for some subtle defect. But that’s the life of a programmer...

Ultimately, we ended up with a program that filtered wavelets in one dimension at a time, linearly. That is, it filtered in the horizontal direction and then the vertical, or vice versa, as opposed to something that filtered radially outwards or in two dimensions at a time (this is the next step in our continuing research). Before we finally hooked it up with the segmentation program, however, we scrutinized our filtered output images, which revealed to us a few mistakes in our program. We quickly corrected these and connected the filter to the segmentation program and ran some segmentations. Our results were interesting, but not all that profound. That is, they were generally marginally better than Edward’s pre-summer research filters, but not that much. Sometimes even it was hard to
tell which was better because segmented images would be comparatively better in some aspects but not in others, which made comparing them like comparing apples to oranges. But sometimes the wavelet segmentations were quite a bit better. On average though, it would probably be a fair assessment to say that our wavelet filters were slightly better, but nothing to write home about. Our best segmentation actually ended up being a fusion of Edward’s pre-summer research sine wave filter and the wavelet filter. That is, the “superposition” of all the color and texture features included color-filtered images in red, green, and blue, along with a wavelet-filtered image and a sine wave-filtered image. This produced the most accurate segmentation to date of the bricks, our main test image.
All in all, we did succeed though maybe not as well as we might have hoped. That being said, while we were hoping to profoundly succeed, we didn’t necessarily expect to. Though I stated earlier that our goal was to create a program that dramatically increased accuracy and flexibility in image segmentation, perhaps a less lofty and more down to earth goal was to actually see if wavelet filters were an improvement at all to the previous standards, such as sine wave filtering. After all, this was an experiment that had unknown potential outcomes to begin with. That is, our research was not devised in such a way that if we worked really hard and efficiently, we were guaranteed dramatic success. It was devised in such a way that we may work hard and efficiently but still not see awesome success, in which case the experiment as a whole would still be successful because we would know how wavelet filters compare to other filtering schemes, not to mention the experience I, personally, gained in programming, mathematics, and scientific research as
a whole. Obviously, I probably benefited from that experience a bit more than Professor Boyda because he’s done research before whereas I haven’t. That is, as much as I was learning about image segmentation and Python and wavelet mathematics, I was also learning about the scientific research processes in general. I’ve always learned from a textbook that always has a known direction and conclusion, so to speak. This was the first time I had ever learned something that isn’t really all that well-known, where I couldn’t just turn to a textbook for a guaranteed answer if I needed help. We carved our own path more so than I’ve ever done before, which is obviously the point of scientific research in the first place: to understand and utilize that which we don’t already understand and don’t already utilize.

Referring back to our moderate success concerning our segmentation improvement but our good success in that we now know wavelet filters may not be such a great improvement to anything else people already utilize in machine vision, Edward more or less suspected this even before we begun our research. That being said, a new perspective on our research goals was to confirm or deny his suspicions. In a way, our research was to check something else off the list and cover all of our bases concerning image segmentation techniques. As Professor Boyda has said to me throughout the research, he thinks it quite probable that machine vision has plateau’d, at least for now. He thinks that computer vision may have gone as far as it will go without artificial intelligence being able to learn and develop the semantic categories humans utilize so well to interpret visual input. In my philosophizing on the matter, it seems to me that when I look at an image, I am, in a matter of microseconds, interpreting a lot of data. Same with a computer. The difference is that I have semantic categories in my mind that
each has a checklist. For example, one semantic category may be bricks. Bricks have an associated checklist that allows me to know something is a set of bricks. Items on this checklist may be the color red, the fact that individual bricks are about half a foot by a third of a foot big, they oftentimes make up walls, a brick wall is the sum total of a lot of bricks laid on one another, etc. And all in a matter of microseconds, I interpret my visual data, and if it crosses enough things off the brick checklist, I know I’m looking at bricks.

The problem for computers is that they don’t have these semantic categories each with their own checklist. So they can gather a bunch of data and not really have anything to do with it. I believe this is why Professor Boyda believes computer vision has plateau’d until artificial intelligence develops to the point that it can develop semantic faculties like humans. Further, this also applies to what I previously asserted about it being easy to segment an image if the program knows what type of input image there is going to be.

This prior knowledge, metaphorically speaking, endows a program with semantic faculties. This is because one or two or three semantic faculties can be endowed in a program relatively easily. Semantic checklists, like I wrote of previously, can be programmed easily if there are only a few. For example, if we know the input image will be only of forestry and city, we could easily just create a program that says if the pixel intensity value is within a green range, assign it a black segmentation value, and if it’s more in the range of the colors of an urban environment, assign it a gray segmentation value. This would probably produce a near-perfect segmentation, but it wouldn’t be a very flexible program. If it received an input image of glaciers and ocean, it would be lost. All of this is to say that the problem arises when tens and hundreds and thousands of semantic categories are needed to produce a flexible segmentation program. It’s easy
enough to program for two semantic categories like trees and urban areas, but it’s not realistic to manually program for all the semantic categories that a truly flexible program would require.

All of this being the case, Professor Boyda and I have seen enough success with the wavelet filter to warrant further research. The past couple paragraphs have made it sound like our program has been a waste in terms of being on track to improve image segmentation, but this is not necessarily the case. Perhaps wavelet filters will improve image segmentation dramatically, despite the logic above, and thus the issues are inherent in our implementation of wavelet filtering (our program), not in wavelet filtering itself. Because of this possibility, we are continuing our research during this semester. The core emphasis of our continued research will be on constructing what’s known as a quincunx wavelet filter, and the result of segmentations using this filter will probably be enough to conclusively state whether wavelet filtering significantly improves image segmentation or not. The difference between this wavelet filter and the one we previously used lies in the “mask” of “h” values. Previously, the mask has been one-dimensional and has separately gone about filtering in the horizontal and vertical directions. This makes it a “separable” filter. Being separable is thought to suppress a wavelet filter’s capabilities, however. The quincunx filter, by contrast, is a non-separable filter, and works in the horizontal and vertical directions simultaneously. The shape and number of dimensions (two) of the quincunx “h” mask manifests this property. Hopefully, it will take better advantage of a wavelet’s edge-detecting capabilities while getting better region-filling features. We have begun research on the quincunx filter and have gotten to the point where we are on the brink of implementing it in code. After creating a quincunx-filtering
program, we will probably perform a trial, error, and tweak method before drawing any definitive conclusions on the segmentations it produces. This will be a similar process to what we used during the summer for our first wavelet filter. After this, we will run segmentations on a variety of images to test its general flexibility and accuracy. We will then hopefully be able to create a program that quantitatively measures accuracy of segmentation. This will be challenging because a segmented image that correctly assigned half of its pixels to a given category may be far less accurate than a segmentation of the same image that also assigned half of its pixels to a given category for reasons that I understood once but now seem unable to recollect. Anyways, we will then use this program to compare our segmentations to the leading standards of unsupervised image segmentation, available at an online database called Jseg, to see where we stand.

As ever, if profoundly successful, we hope to use our program on satellite imagery to monitor changes such as deforestation, glacial melt, and desertification.