HW7 – Adaptive Median Filtering

This assignment was used to teach how to implement the Adaptive Median Filtering algorithm. The program loops through odd filter sizes from 3x3 to 21x21 and calculates the adaptive median filter value for each pixel by calling a two-function algorithm that had to be implemented from scratch.

Ryan Orf

11/23/2021 CS4650: Digital Image Processing

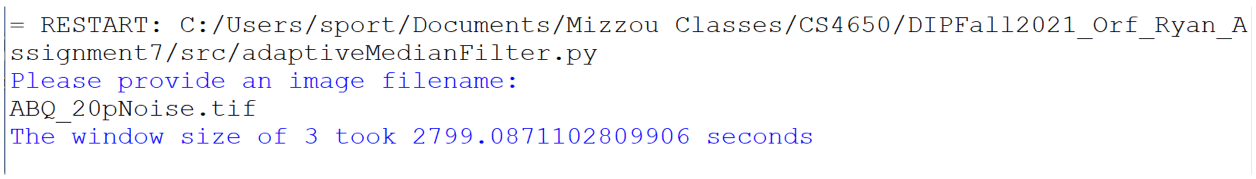
For this assignment, students were challenged to implement the Adaptive Median Filter algorithm from scratch. By accepting a Grayscale image as input, the program would begin by padding the edges with zeroes to prevent out-of-bounds errors before proceeding to the main calculation. Using a for loop that would iterate based on the minimum and maximum filter sizes, for this assignment 3 and 21, the program would call the Adaptive Median Filter functions (starting at StageA) for each pixel value. If the median was an impulse, StageB was called. Z_(xy) was returned if the central pixel was not an impulse, or Z_(median) was returned if it was. Back in StageA, if the median is an impulse, the window size would be increased. If the maximum window size, S_(max), had been surpassed, Z_(median) would be returned. If not, StageA would get recursively called. Once this has happened at every pixel, the corrected image is output to the user, and the time it took for the filter to be completely applied to the image is printed to the user.

This assignment was extremely straightforward in terms of implementation. For statistical results, I have some notable runtimes which will be included in the end. However, first I will explain noteworthy parts of the code, instead of providing a line-by-line examination since this assignment involved less code implementation. First and foremost, I used the same image filename prompt as I have previously. This will ensure a Grayscale image is provided as input (or convert an RGB/BGR image to Grayscale if necessary). I have the variable “s” set to 3 and “sMax” set to 21, since these are the filter sizes we will be working with. Next, I calculate a variable for the padding which is determined by the floor of sMax divided by 2. This is because zero padding is what I implemented in order to prevent errors, and if you start at the most extreme case (0,0), you will need exactly the number of rows and columns I calculated for the padding (the floor of half the max) in order to calculate the Adaptive Mean Filters without error.

Next, I loop through the filter sizes I will be using. I include the variables of "img_w_padding" (the padded image) and img_filtered (the output filtered image) inside this loop so that I am able to reuse the variables and do not need to re-initialize them. First, I pad the padded image with zeroes and replace all of the values corresponding to the original image with their proper pixel values. Note that I must cast these values as a "unit8", or else it will default to "float64", providing the user with the incorrect output. Then, I pad the filtered image with zeroes as well, once again casting the result as a "uint8". I also note the start time now, before the first iteration of the loop through the pixels in the image runs. Since I extended the image with zeroes, I do not start and end at the typical image pixel values, hence the difference in this code compared to prior implementations. Now, the Adaptive Median Filter algorithm is simply called at each pixel. Once this has completed, the time is noted, the start time is subtracted from it, and this runtime result is output to the user, along with the newly cleared image. Each filtered image is displayed for 20 seconds.

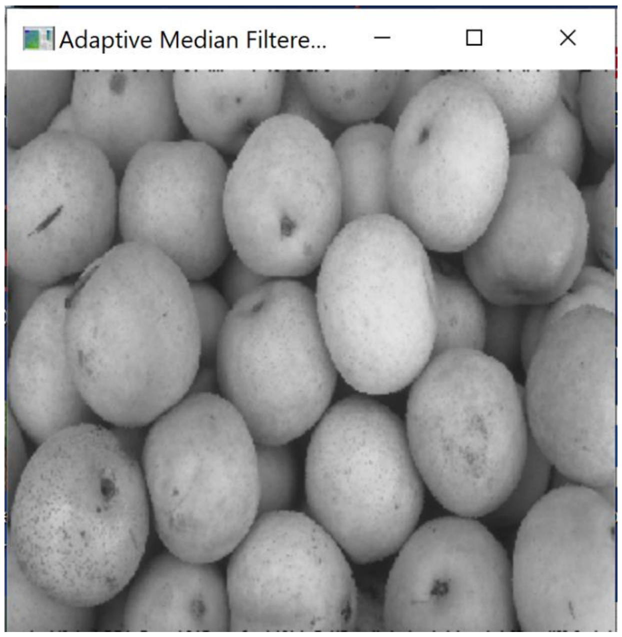
I felt that this was a helpful assignment to implement. Much of the program we should have already known how to implement, or it was provided. On top of that, most of the Adaptive Median Filter algorithm we had to implement was provided in notes. I believe the hardest part of this assignment was some of the logic, such as calculating index values for working with zero padding and remembering to cast initialized variables as "uint8". As was foretold and expected, this program takes an immensely long time to run, especially for the noisy pictures of ABQ provided. Simply one pass over the image with a filter took my machine 46.65 minutes. Thankfully, the Pears images took less than or around 30 seconds per filter size. If I were to speed up the implementation of the Adaptive Median Filter, I would conceive a couple of ideas that may help in this department. As discussed in class, instead of using a sorting algorithm that is calculated at each pixel, we could use the histogram of the image and implement a bucket sort algorithm. Another possibility could be working with all pixel values in the filter at once instead of calculating values for each individual pixel. This could be done by comparing each value in the filter to the median and replacing pixel values greater than the average of the filter. In theory, this too would greatly decrease the runtime of the program. Although I have chosen not to implement these for extra credit, I am curious in what this would look like. However, now I can show the results of my work, so please find below sample of images of the program running, which includes the runtimes for each filter size.

Runtime for the ABQ_20pNoise Image with a Window Size of 3:

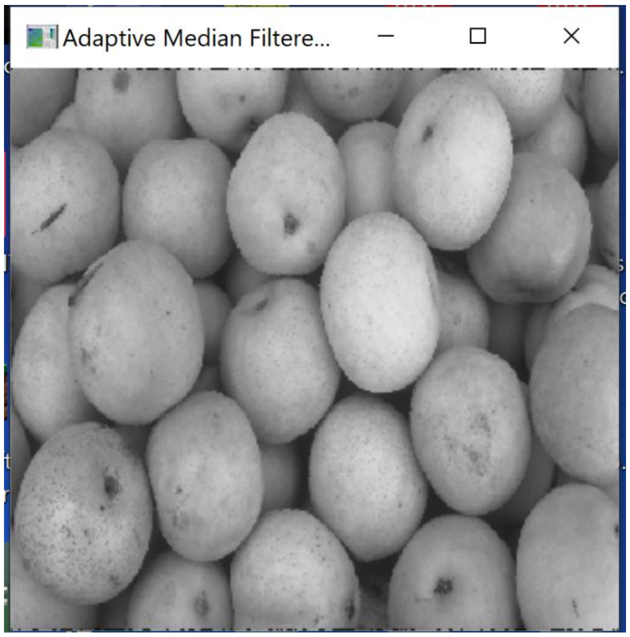


```
= RESTART: C:/Users/sport/Documents/Mizzou Classes/CS4650/DIPFall2021_Orf_Ryan_A
ssignment7/src/adaptiveMedianFilter.py
Please provide an image filename:
ABQ_20pNoise.tif
The window size of 3 took 2799.0871102809906 seconds
```

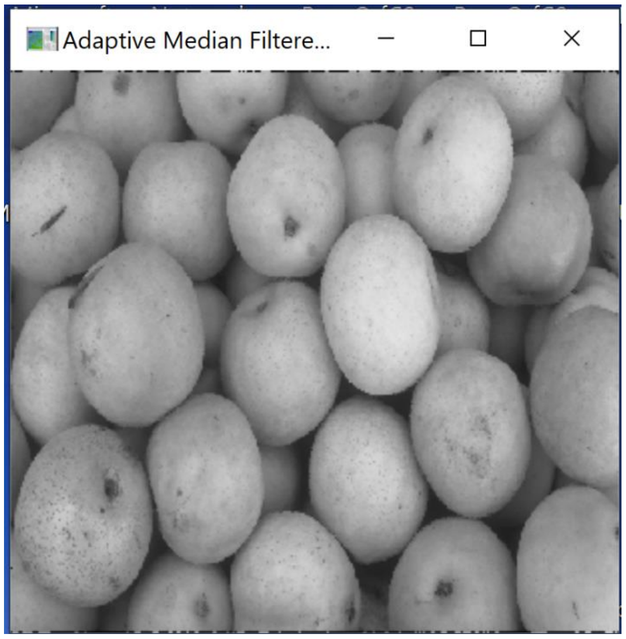
Pears_60pNoise Image with a Window Size of 3:



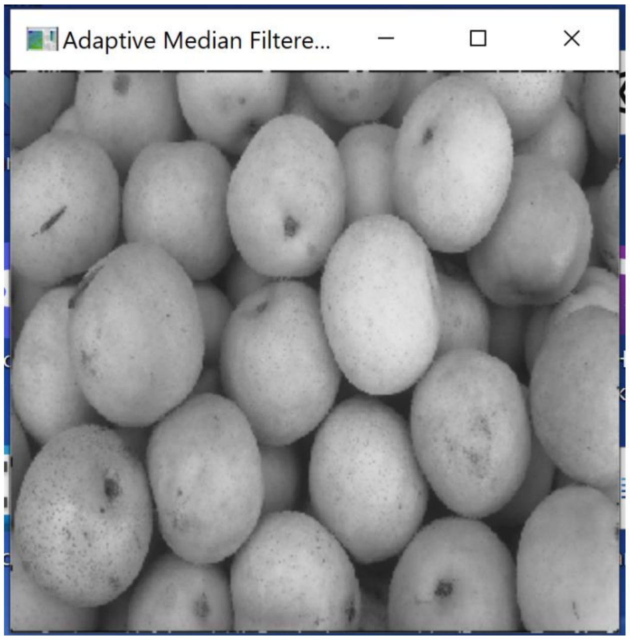
Pears_60pNoise Image with a Window Size of 5:



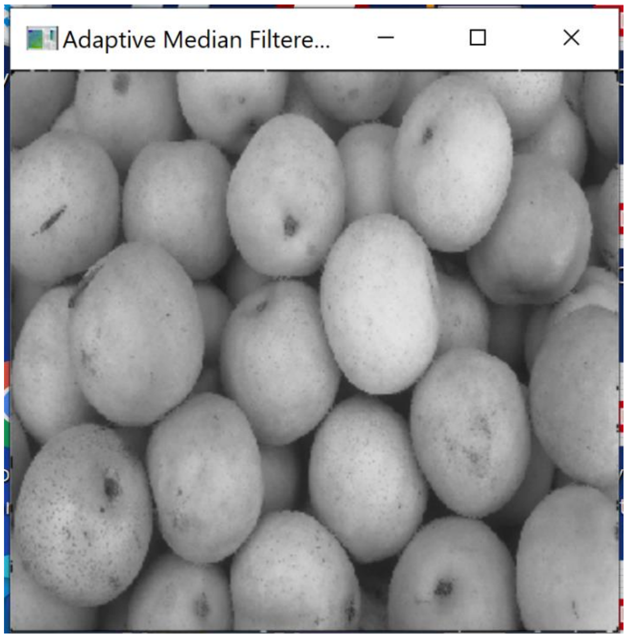
Pears_60pNoise Image with a Window Size of 7:



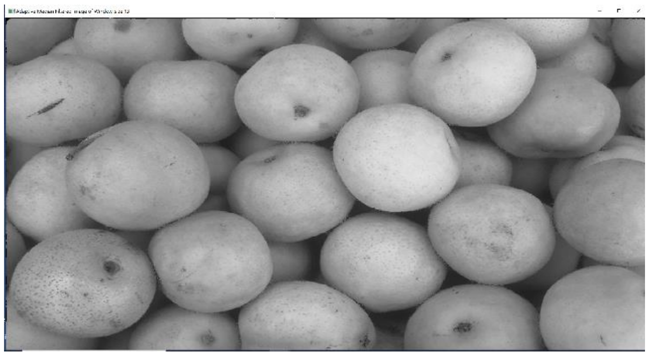
Pears_60pNoise Image with a Window Size of 9:



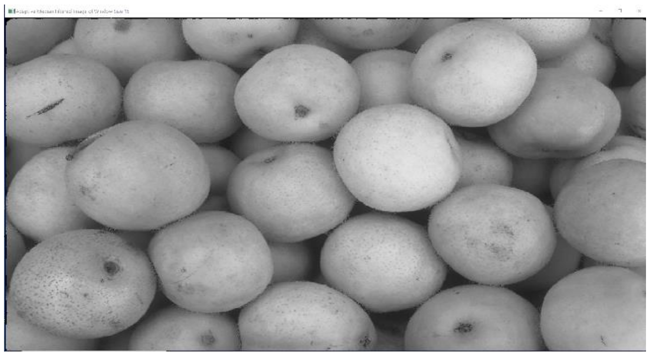
Pears_60pNoise Image with a Window Size of 11:



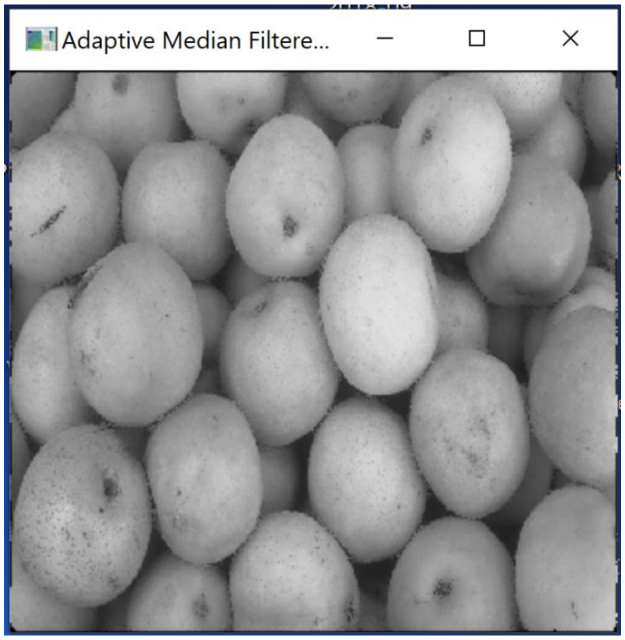
Pears_60pNoise Image with a Window Size of 13 (**Note: For the window sizes of 13 and 15, the cv.WINDOW_NORMAL window size defaulted to full screen, so the sample images for window sizes 13 and 15 look slightly different in this documentation):



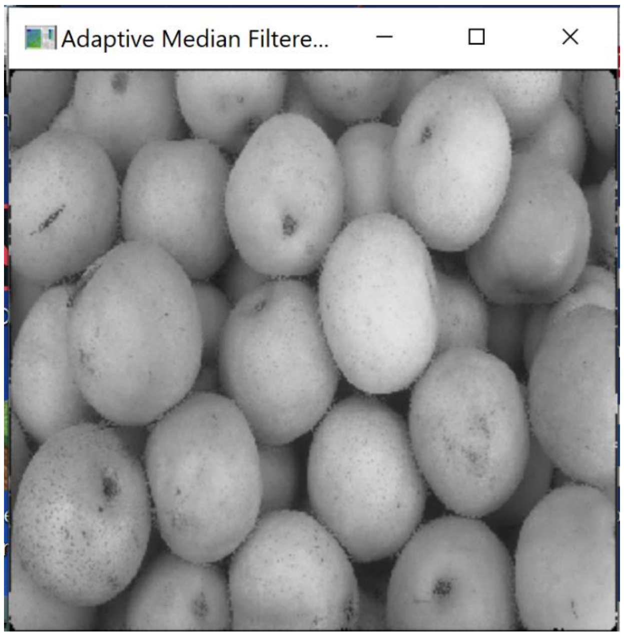
Pears_60pNoise Image with a Window Size of 15:



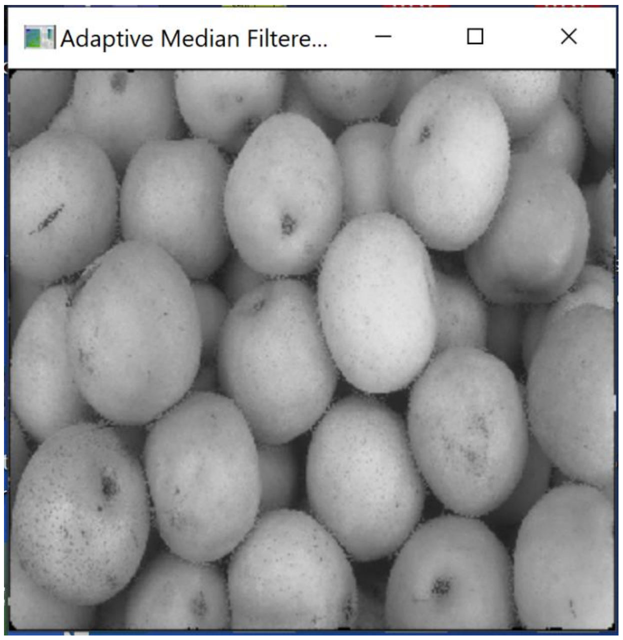
Pears_60pNoise Image with a Window Size of 17:



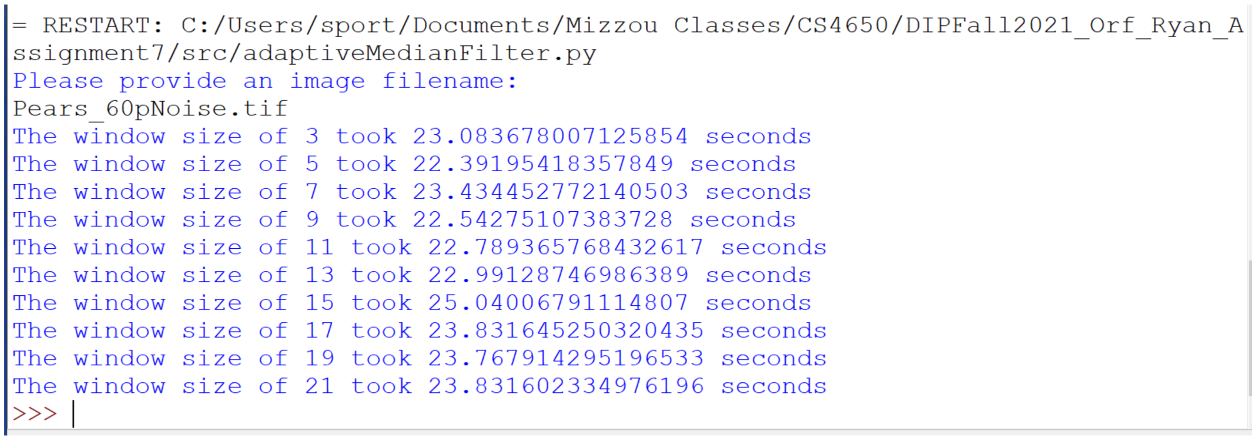
Pears_60pNoise Image with a Window Size of 19:



Pears_60pNoise Image with a Window Size of 21:



Runtime for the Pears_60pNoise Image with All Window Sizes:



```
= RESTART: C:/Users/sport/Documents/Mizzou Classes/CS4650/DIPFall2021_Orf_Ryan_A
ssignment7/src/adaptiveMedianFilter.py
Please provide an image filename:
Pears_60pNoise.tif
The window size of 3 took 23.083678007125854 seconds
The window size of 5 took 22.39195418357849 seconds
The window size of 7 took 23.434452772140503 seconds
The window size of 9 took 22.54275107383728 seconds
The window size of 11 took 22.789365768432617 seconds
The window size of 13 took 22.99128746986389 seconds
The window size of 15 took 25.04006791114807 seconds
The window size of 17 took 23.831645250320435 seconds
The window size of 19 took 23.767914295196533 seconds
The window size of 21 took 23.831602334976196 seconds
>>> 
```