ECE 6310- INTRODUCTION TO COMPUTER VISION

LAB 1: CONVOLUTION, SEPARABLE FILTERS, SLIDING WINDOWS

Submitted To:

Dr. Adam Hoover

Ashit Mohanty

C13582787

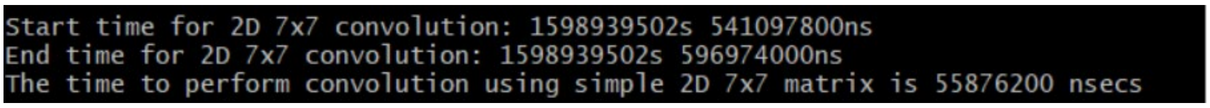
amohant@g.clemson.edu

Date of submission: 09-01-2020

PART 1: Convolution using 2D 7x7 mean filter

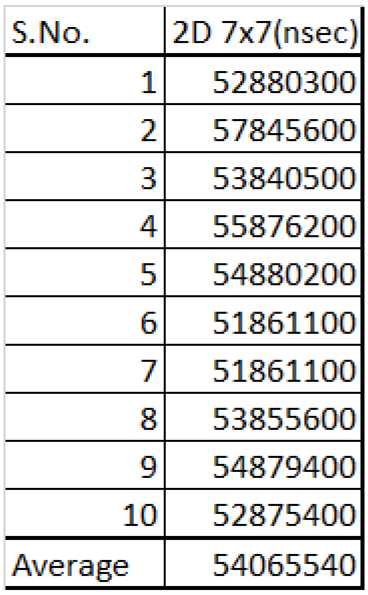
The first part of the lab assignment was to perform a simple 2D convolution using a 7x7 mean filter. The image that was used to perform this operation was the bridge.ppm file. The output of this file was an image stored as smoothed.ppm. The program was written in C and run on Windows. As per the instructions of the lab, any pixel for which the convolution was to extend outside the image should have an output of zero.

The program also had a provision that generated the time for the convolution process to take place.



```
Start time for 2D 7x7 convolution: 1598939502s 541097800ns
End time for 2D 7x7 convolution: 1598939502s 596974000ns
The time to perform convolution using simple 2D 7x7 matrix is 55876200 nsecs
```

This operation was performed over 10 consecutive runs to generate the average time for the 2D 7x7 mean filter to perform convolution.



S.No.	2D 7x7(nsec)
1	52880300
2	57845600
3	53840500
4	55876200
5	54880200
6	51861100
7	51861100
8	53855600
9	54879400
10	52875400
Average	54065540

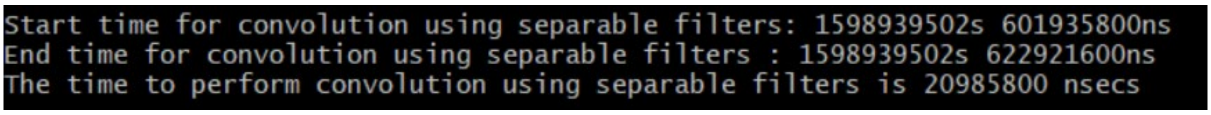
As seen on the image on the top, the average of consecutive 10 readings were taken, and the average time to run the 2D 7x7 mean filter and perform convolution was 54,065,540 nsec or 54.1 milliseconds.

PART 2: Convolution using separable filters

The second part of the lab assignment was to perform 2 1D convolutions using separable filters. Instead of using 1 2D 7x7 filter, this part used a 1x7 filter followed by a 7x1 filter. The image that was used to perform this operation was the bridge.ppm file. The output of this file was an image stored as smoothed_sep.ppm. The program was written in C and run on Windows.

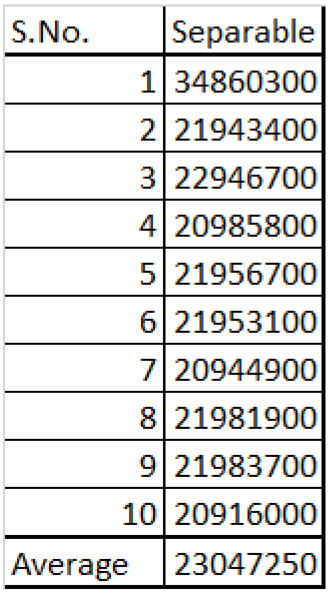
For the 1x7 filter, as per the instructions of the lab, any pixel for which the convolution was to extend outside the image should have an output of zero. The output from this 1x7 filter was then stored in an intermediate float array, which was used as the input for the next filter, 7x1. The output of this filter was an image stored as smoothed_sep.ppm.

The program also had a provision that generated the time for the convolution process to take place.



```
Start time for convolution using separable filters: 1598939502s 601935800ns
End time for convolution using separable filters : 1598939502s 622921600ns
The time to perform convolution using separable filters is 20985800 nsecs
```

This operation was performed over 10 consecutive runs to generate the average time for the separable mean filters to perform the convolution.



S.No.	Separable
1	34860300
2	21943400
3	22946700
4	20985800
5	21956700
6	21953100
7	20944900
8	21981900
9	21983700
10	20916000
Average	23047250

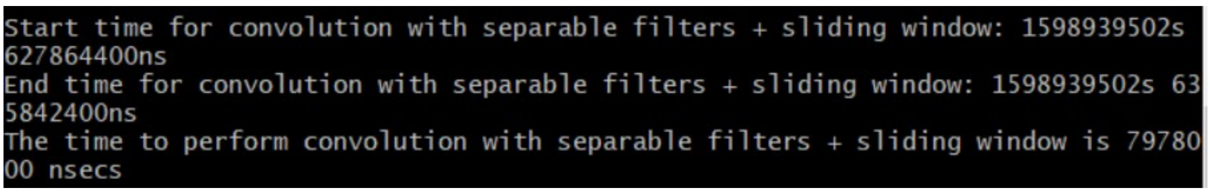
As seen on the image on the top, the average of consecutive 10 readings were taken, and the average time to run the separable mean filters (1x7 and 7x1) to perform the convolution was 23,047,250 nsec or 23.05 milliseconds.

PART 3: Convolution using separable filters and sliding window

The third part of the lab assignment was to perform convolution using separable filters and sliding window. The sliding window is bound to reduce the computation time because when traversing along the same row or same column via a 1x7 filter or 7x1 filter respectively, there is no need to sum up all the pixels. Rather just remove the first pixel in the previous iteration and add the last pixel of this iteration. This step saves up a lot of the processing time and makes the convolution process faster. The image that was used to perform this operation was the bridge.ppm file. The output of this file was an image stored as smoothed_sep_sw.ppm. The program was written in C and run on Windows.

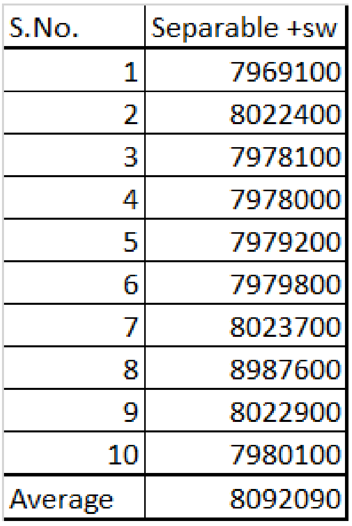
For the 1x7 filter, as per the instructions of the lab, any pixel for which the convolution was to extend outside the image should have an output of zero. The output from this 1x7 filter was then stored in an intermediate float array, which was used as the input for the next filter, 7x1.The output of this filter was an image stored as smoothed_sep_sw.ppm.

The program also had a provision that generated the time for the convolution process to take place.



```
Start time for convolution with separable filters + sliding window: 1598939502s 
627864400ns
End time for convolution with separable filters + sliding window: 1598939502s 63
5842400ns
The time to perform convolution with separable filters + sliding window is 79780
00 nsecs
```

This operation was performed over 10 consecutive runs to generate the average time for the separable mean filters and sliding window to perform the convolution.

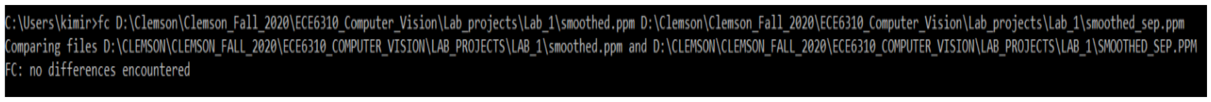


S.No.	Separable +sw
1	7969100
2	8022400
3	7978100
4	7978000
5	7979200
6	7979800
7	8023700
8	8987600
9	8022900
10	7980100
Average	8092090

As seen on the image on the top, the average of consecutive 10 readings were taken, and the average time to run the separable mean filters (1x7 and 7x1) and sliding window to perform the convolution was 8,092,090 nsec or 8.1 milliseconds.

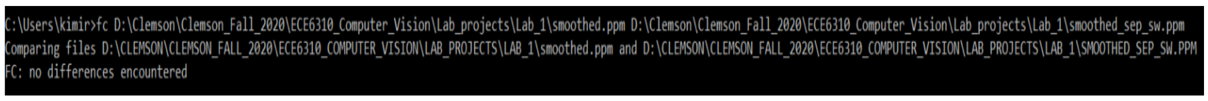
COMPARISON OF THE THREE METHODS

The output of the three methods which are the image files, smoothed.ppm, smoothed_sep.ppm and smoothed_sep_sw.ppm respectively, were compared on Windows using the file compare function. The syntax of this function is to write fc [file 1] [file 2]. In the image attached underneath the file along with the locations is entered for comparison.



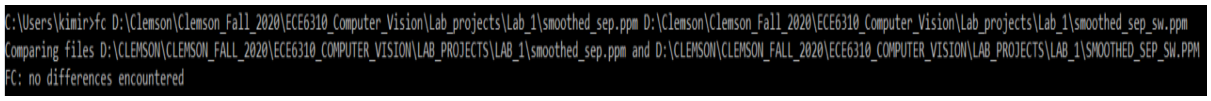
```
C:\Users\kimir>fc D:\Clemson\Clemson_Fall_2020\ECE6310_Computer_Vision\Lab_projects\Lab_1\smoothed.ppm D:\Clemson\Clemson_Fall_2020\ECE6310_Computer_Vision\Lab_projects\Lab_1\smoothed_sep.ppm
Comparing files D:\CLEMSON\CLEMSON_FALL_2020\ECE6310_COMPUTER_VISION\LAB_PROJECTS\LAB_1\smoothed.ppm and D:\CLEMSON\CLEMSON_FALL_2020\ECE6310_COMPUTER_VISION\LAB_PROJECTS\LAB_1\SMOOTHED_SEP.PPM
FC: no differences encountered
```

As seen in the image above, in the first file compare program command, smoothed.ppm (output of 2D convolution) and smoothed_sep.ppm (output of separate filter) was compared. The program returns the output no differences encountered.



```
C:\Users\kimir>fc D:\Clemson\Clemson_Fall_2020\ECE6310_Computer_Vision\Lab_projects\Lab_1\smoothed.ppm D:\Clemson\Clemson_Fall_2020\ECE6310_Computer_Vision\Lab_projects\Lab_1\smoothed_sep_sw.ppm
Comparing files D:\CLEMSON\CLEMSON_FALL_2020\ECE6310_COMPUTER_VISION\LAB_PROJECTS\LAB_1\smoothed.ppm and D:\CLEMSON\CLEMSON_FALL_2020\ECE6310_COMPUTER_VISION\LAB_PROJECTS\LAB_1\SMOOTHED_SEP_SW.PPM
FC: no differences encountered
```

As seen in the image above, in the second file compare program command, smoothed.ppm (output of 2D convolution) and smoothed_sep_sw.ppm (output of separate filter and sliding window) was compared. The program returns the output no differences encountered.



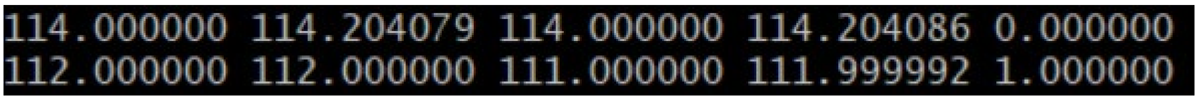
```
C:\Users\kimir>fc D:\Clemson\Clemson_Fall_2020\ECE6310_Computer_Vision\Lab_projects\Lab_1\smoothed_sep.ppm D:\Clemson\Clemson_Fall_2020\ECE6310_Computer_Vision\Lab_projects\Lab_1\smoothed_sep_sw.ppm
Comparing files D:\CLEMSON\CLEMSON_FALL_2020\ECE6310_COMPUTER_VISION\LAB_PROJECTS\LAB_1\smoothed_sep.ppm and D:\CLEMSON\CLEMSON_FALL_2020\ECE6310_COMPUTER_VISION\LAB_PROJECTS\LAB_1\SMOOTHED_SEP_SW.PPM
FC: no differences encountered
```

As seen in the image above, in the third file compare program command, smoothed_sep.ppm (output of separate filter) and smoothed_sep_sw.ppm (output of separate filter and sliding window) was compared. The program returns the output no differences encountered.

ISSUES FACED WHILE COMPARISON OF OUTPUTS

Initially there was a discrepancy in the images that were being generated by the three different methods. The output of the 2D 7x7 mean filter was different to the outputs of the separate filter convolution and the separate filter + sliding window convolution. This was because if the rounding off issue when the mean was being calculated for every pixel. I decided to get into the bottom of this as to why was this happening.

Let’s take this case as an example. The screenshot is part of the debugging process that I did to get to the bottom of the situation.



```
114.000000 114.204079 114.000000 114.204086 0.000000
112.000000 112.000000 111.000000 111.999992 1.000000
```

- The first column is the value being written into the image character array.
- The second column in the above image shows the actual value of the pixel upon taking the mean i.e. dividing the sum by 49, i.e. using a 2D 7x7 mean filter.
- The third column in the image is the value being written into the character array that would generate the image for separate filters and sliding windows.
- The fourth column in the above image shows the actual value being calculated upon taking the mean using separate filters and separate filters and sliding window.

As seen when the values are being stored in the variable ‘sum’ which was of type ‘int’ the numbers were being rounded of to the lowest integer. In the first row that was 114 in both the cases, so no issues.

However, in certain cases, like in the second column, the value for the 2D filter is 112.0 but the value for the separate filters in 111.99. So, on storing it in the variable ‘sum’ of type ‘int’, the values were being stored as 112 and 111. To eliminate this discrepancy, I decided to round of the answer of the mean and store the rounded of integer value into the character array. This way 112.0 was being round of to 112 and 111.999 was being round of 112. And this way the output of all the images was the same.