Introduction to Computer Vision
Filters and Timing

Mayukh Sattiraju
msattir@clemson.edu

ECE 631
Lab 1 Report

September 6, 2017

1 Naive Mean Filter

The trivial version of the Mean filter is implemented by nesting two for-loops for implementing the convolution. The nested loop loops over the entire 7x7 filter.

Below is a code snippet of the filter implementation.

Listing 1: Naive Filter

```
      /* smooth image, skipping the border points */
for (r=3; r<ROWS-3; r++)
  for (c=3; c<COLS-3; c++)
    {
    sum=0;
    for (r2=-3; r2<=3; r2++)
      for (c2=-3; c2<=3; c2++)
        sum+=image[(r+r2)*COLS+(c+c2)];
    smoothed[r*COLS+c]=sum/49;
    }
```

The time taken for this code to run: 67,314 micro seconds.

2 Separated Kernel

The 7x7 filter is separated into two filters a column and a row filter. Then each is convolved with the input image separately.

The time gained is expected to be significant as the operations performed at each input pixel is reduced from 49 to 14.

Below is a code snippet of the filter implementation.

Listing 2: Separated Kernel

```
        /* smooth image, skipping the border points */
for (c=0; c<COLS; c++)
   for (r=3; r<ROWS-3; r++)
    {
    sum=0;
    for (r2=-3; r2<=3; r2++)
        sum+=image[((r+r2)*COLS)+c];
    smoothed1[r*COLS+c]=sum;
    }
```

```
for (r=0; r<ROWS; r++)
  for (c=3; c<COLS-3; c++)
    {
    sum1=0;
      for (c2=-3; c2<=3; c2++)
        sum1+=smoothed1[(r*COLS)+(c+c2)];
    smoothed[r*COLS+c]=sum1/49;
    }
```

The time recorded for this implementation is: 22,820 micro seconds.

3 Sliding Window and a Separated Kernel

This part implements a sliding window to maintain sums across the filter.

The improvement in computation is that instead of 14 operations per input pixel, this implementation uses 4 operations per input pixel.

Here's a code snippet of this implementation:

Listing 3: Sliding Window and Separated Kernel

```
        /* smooth image, skipping the border points */

first_time=0;
for (c=0; c<COLS; c++){
  for (r=3; r<ROWS-3; r++)
    {

    if(first_time==0){
        first_time=1;
        sum=0;
        for (r2=-3; r2<=3; r2++)
                sum+=image[((r+r2)*COLS)+c];
        smoothed1[r*COLS+c]=sum;
        subtractor=image[(r-3)*COLS+c];
        }
    else{
        sum+=image[((r+3)*COLS)+c];
        sum-=image[((r-4)*COLS)+c];
        smoothed1[r*COLS+c]=sum;
        }
    }
  first_time=0;
}
```

```
first_time=0;
for (r=0; r<ROWS; r++){
  for (c=3; c<COLS-3; c++)
    {
    if (first_time==0){
        first_time=1;
        sum1=0;
      for (c2=-3; c2<=3; c2++)
        sum1+=smoothed1[(r*COLS)+(c+c2)];
    smoothed[r*COLS+c]= (sum1/49);
    }
    else{
        sum1+=smoothed1[((r)*COLS)+(c+3)];
        sum1-=smoothed1[((r)*COLS)+(c-4)];
        smoothed[r*COLS+c]=sum1/49;

    }
  }
 first_time=0;
}
```

The time recorded for this implementation is : 4,515 micro seconds

4 Results

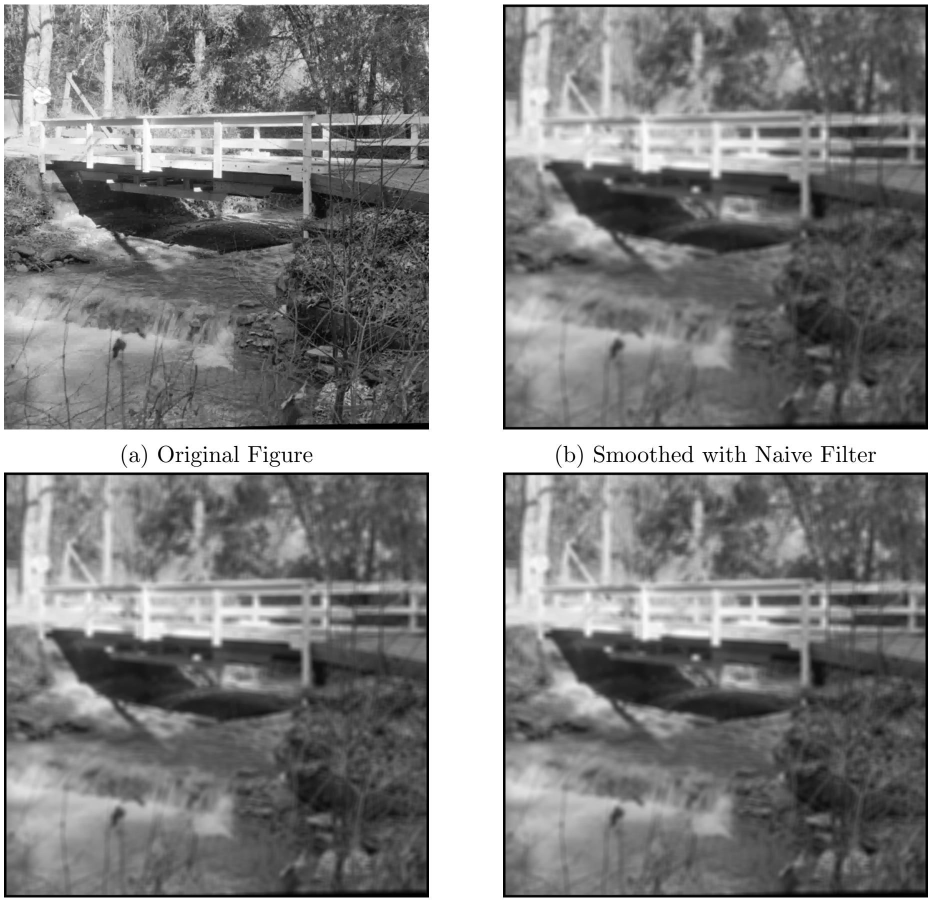
The processing time for each implementation is shown below:

Filter	Time (micro sec)
Naive	67,314
Separated Filter	22,820
Sliding Window and Separated	4,515

Table 1: Processing Times

We can see that as expected, the processing time for the separable filter is lesser than the naive implementation. The time for the Sliding window and Separated filter is the least.

Here is the original image and the images filtered with the above three methods. It was also confirmed using diff that the three output images were identical.



(a) Original Figure

(b) Smoothed with Naive Filter

(c) Smoothed with Separated Filter

(d) Smoothed with Sliding Window

Figure 1: Plots of Filtered Images