Name: Saroj Kumar Dash
Course: ECE-6310
Course Name: Intro to Computer Vision

# Lab#1 – Convolution, separable filters, sliding windows

Problem Statement:

To implement 3 version of 7x7 mean filters.
Version 1 :- with basic 2d convolution
Version 2 :- with separable filters(1x7 and 7x1)
Version 3 :- with separable and sliding window

Solution Version 1:-- with basic 2d convolution

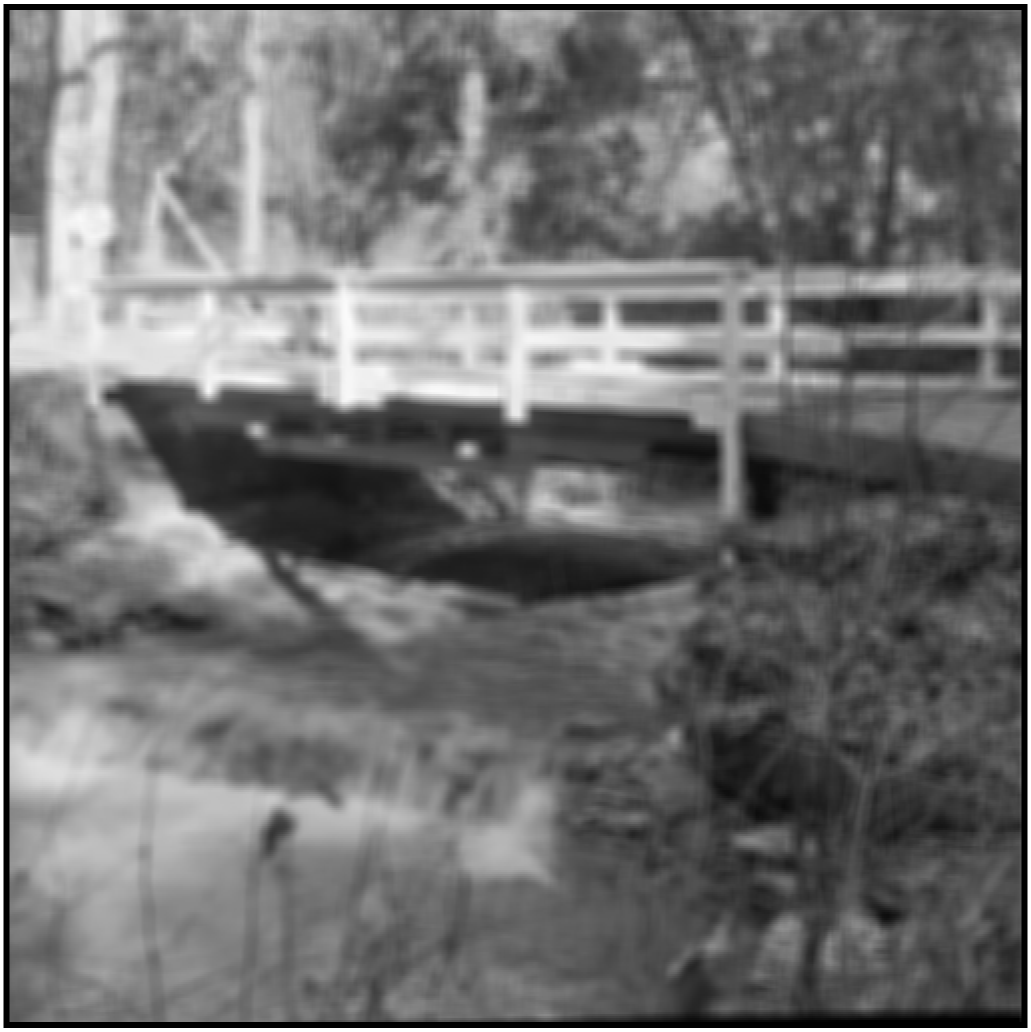
I made two separate function to separate the kernel multiplication.Image is a class that I have created to manage the data structure of the image rows, cols and pixels which reduces the number of arguments to pass to the methods.

Code:-

```
//Separated kernel multiplication operation
int kernel2dConv(image &iImg,int iCol,int iRow){

  int r,c,sum=0;
  int outPixel = 0;
  int ker[7][7] = {    //Now using a mean filter but later we can use any filter here
                    {1,1,1,1,1,1,1},
                    {1,1,1,1,1,1,1},
                    {1,1,1,1,1,1,1},
                    {1,1,1,1,1,1,1},
                    {1,1,1,1,1,1,1},
                    {1,1,1,1,1,1,1},
                    {1,1,1,1,1,1,1}};
  int kSize = 7;
  unsigned char *pInPixels = iImg.getPixels();

  for( r=-(kSize/2); r<=(kSize/2); r++)
    for(c=-(kSize/2); c<=(kSize/2); c++)
        sum += (pInPixels[(iRow+r)*iImg.getCols() + (iCol+c)]*ker[c+(kSize/2)][r+(kSize/2)]);

  outPixel = sum/(kSize*kSize);
  return outPixel;
}

//do the main convolution operation here whoever uses this method should delete the returned pointer
//this method is clocked in main()
unsigned char *convolveSimple(image &iImg){
   unsigned char *opPixels = NULL;
   int r,c;
   int sum;
   opPixels = new unsigned char[iImg.getRows() * iImg.getCols()];
   //Do convolve operation here
  for(r=3; r< (iImg.getRows()-3); r++)
    for(c=3 ; c<(iImg.getCols()-3); c++)
      opPixels[r*iImg.getCols()+c] = (unsigned char)kernel2dConv(iImg,c,r);

   return opPixels;
}
```

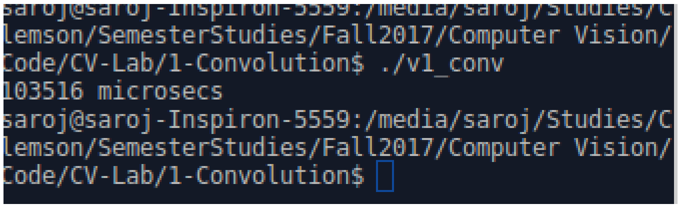
Output:-

Generates a smoothed output image named, **out.ppm**, which is a smoothed version of input image bridge.ppm. Which looks as below:-



Time Clocked is :- **103516 microsecs**

I have taken observation by measuring the time taken by the function  convolveSimple().



```
saroj@saroj-Inspiron-5559:/media/saroj/Studies/C
lemson/SemesterStudies/Fall2017/Computer Vision/
Code/CV-Lab/1-Convolution$ ./v1_conv
103516 microsecs
saroj@saroj-Inspiron-5559:/media/saroj/Studies/C
lemson/SemesterStudies/Fall2017/Computer Vision/
Code/CV-Lab/1-Convolution$ 
```

## Solution Version 2 :- with separable filters(1x7 and 7x1)

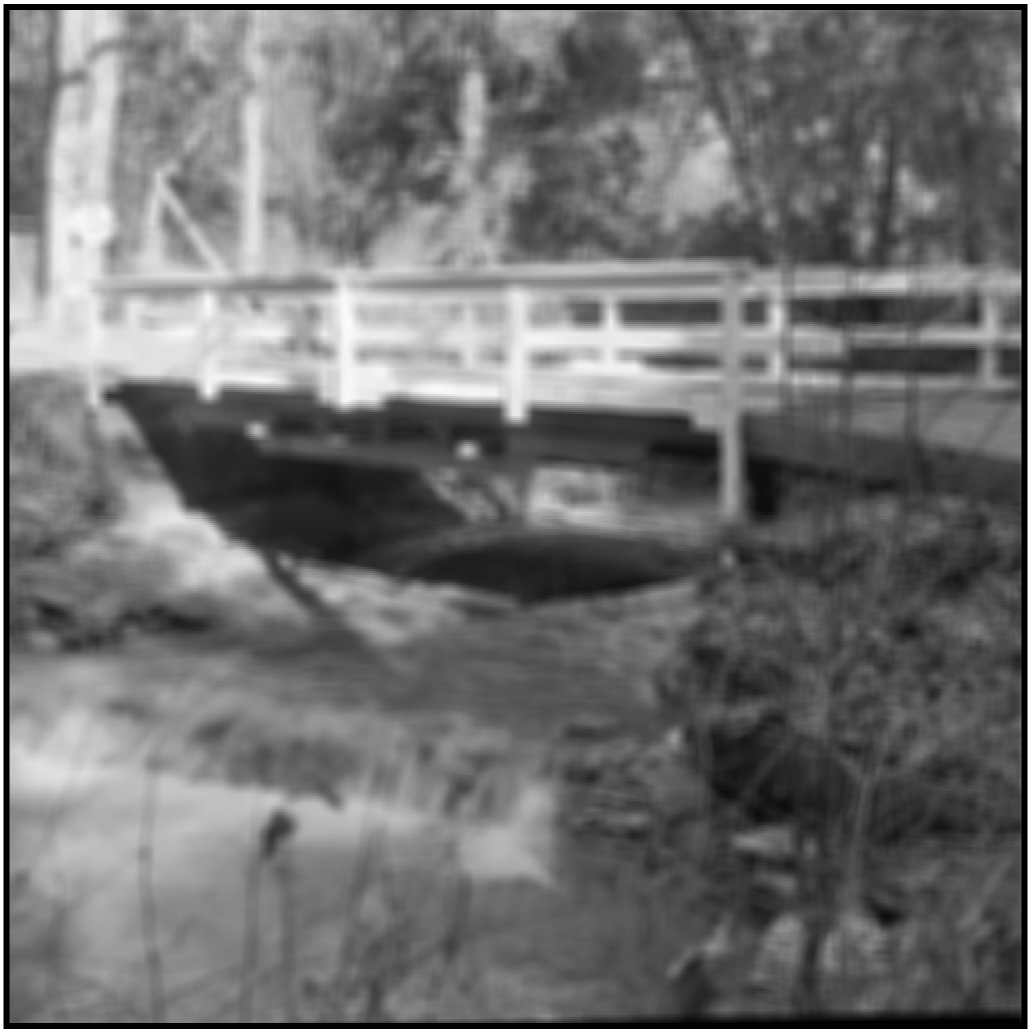
## Code:

```
//Convolves a 1x7 kernel horizontally //@return:  outPixSum : sum of kernel operation for the mean filter
int kernelSepHor(image &iImg,int iCol,int iRow){

        int r,c,kSize = 7;
        int outPixSum = 0;
        int kerR[7] = {1,1,1,1,1,1,1};
        int sum =0,temp =0;
        unsigned char *pInPixels = iImg.getPixels();

        // horizontal convolution
        for( c=-(kSize/2); c<=(kSize/2); c++)
                sum += pInPixels[(iRow)*iImg.getCols()+(iCol+c)]*kerR[c+(kSize/2)];

        outPixSum = sum;
        //std::cout<<outPixSum<<"   "<<iCol<<"   "<<iRow<<std::endl;
        return outPixSum;
}

//Convolves a 7x1 kernel vertically
//@return: outPixSum : sum of kernel operation for the mean filter
int kernelSepVer(unsigned int *&pInPixels,int iCol,int iRow,int iW,int iH){

        int r,c,kSize = 7;
        int outPixSum = 0;
        int kerR[7] = {1,1,1,1,1,1,1};
        int sum =0,temp =0;

        // vertical convolution
        for( r=-(kSize/2); r<=(kSize/2); r++)
                sum += pInPixels[((iRow+r)*iW)+(iCol)]*kerR[r+(kSize/2)];

        outPixSum = sum;
        //std::cout<<outPix<<"   "<<iCol<<"   "<<iRow<<std::endl;
        return outPixSum;
}

//do the main convolution operation here whoever uses this method should release the returned pointer.
//this function is time clocked
unsigned char *convolveImage(image &iImg){
   unsigned char *opPixels = NULL;
   unsigned int *tmpPixels = NULL;
   int r,c,kSize = 7;
   int sum;
   //allocate memory for the out Pixel pointer
   opPixels = new unsigned char[iImg.getRows() * iImg.getCols()];
   tmpPixels = new unsigned int[iImg.getRows() * iImg.getCols()];

   //Do horizontal convolution, output stored in a temporary array
   for(r=0; r<iImg.getRows(); r++)
     for(c=3 ; c<(iImg.getCols()-3); c++)
        tmpPixels[r*iImg.getCols()+c] = kernelSepHor(iImg,c,r);
```

```
    //Do vertical convolution
    for(c=3; c<(iImg.getRows()-3); c++)
      for(r=3 ; r<(iImg.getCols()-3); r++)
           opPixels[r*iImg.getCols()+c] =
                     (unsigned char)(kernelSepVer(tmpPixels,c,r,iImg.getCols(),iImg.getRows())/(kSize*kSize));
    delete(tmpPixels);
    return opPixels;
}
```

**Output:-**

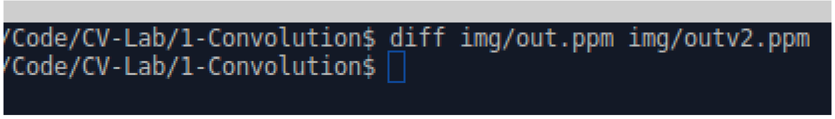
Generates a smoothed output image named, **outv2.ppm**, which is a smoothed version of input image **bridge.ppm**. Which looks as below:-



**Output Validation:-**

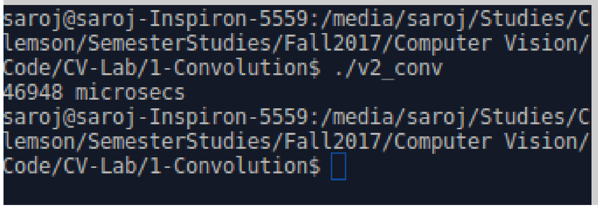
Linux version of diff on **out.ppm** and **outv2.ppm** proves both the files are equal.

*My machine result:-*



```
/Code/CV-Lab/1-Convolution$ diff img/out.ppm img/outv2.ppm
/Code/CV-Lab/1-Convolution$ 
```

**Time Clocked:-** **46948 microsecs**



```
saroj@saroj-Inspiron-5559:/media/saroj/Studies/C
lemson/SemesterStudies/Fall2017/Computer Vision/
Code/CV-Lab/1-Convolution$ ./v2_conv
46948 microsecs
saroj@saroj-Inspiron-5559:/media/saroj/Studies/C
lemson/SemesterStudies/Fall2017/Computer Vision/
Code/CV-Lab/1-Convolution$ 
```

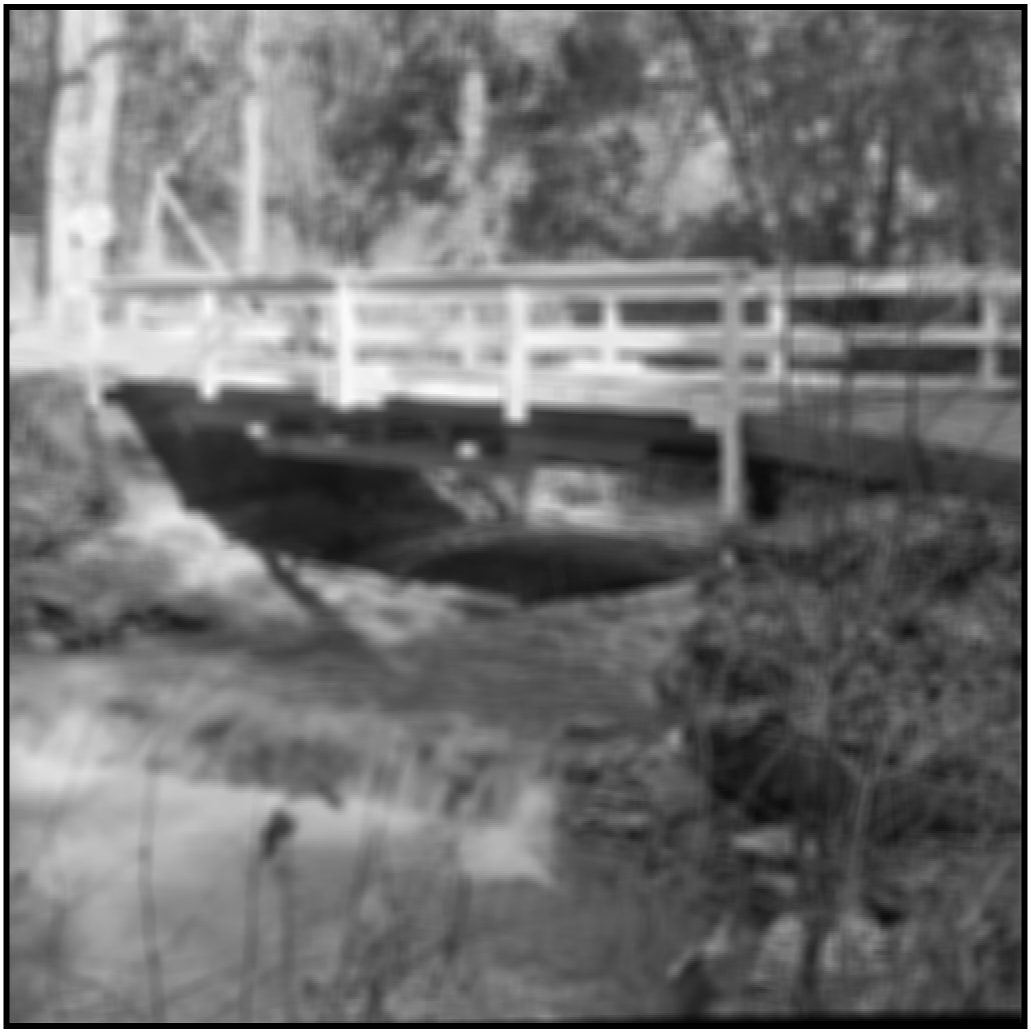
## Solution Version 3 :- with separable and sliding window

Code:

```
//Kernel sliding window algorithm //a 1x7 filter is slided horizontally in the image
int kernelSepSlideWinhor(image &iImg,int iCol,int iRow){

        int outPixSum=0;
        static int prev1stColSum = 0,prevSum = 0;
        int sum = 0;
        unsigned char *pInPixels = iImg.getPixels();

        //kernel separable details
        int ker[7] = {1,1,1,1,1,1,1};
        int kSize = 7;

        //first pixel of the row, iCol = 3
        if( (kSize/2) == iCol){
          //Compute the full kernel and save the prev1stColSum and the prevSum
          for( int c=-(kSize/2); c<=(kSize/2); c++)
          {
                //we increment horizontally
                sum += (pInPixels[(iRow)*iImg.getCols() + (iCol+c)]*ker[c+(kSize/2)]);
                if((c+(kSize/2) == 0)) //saving the first convolved value
                        prev1stColSum = sum;
                }
          //save the prevSum
          prevSum = sum;
        }//not a first pixel of the row, iCol!=3
        else{
          int lastColSum=0,nb1stColSum=0;
          //Compute the last col sum, lastColSum
          lastColSum = pInPixels[iRow*(iImg.getCols()) + (iCol+(kSize/2))];
          nb1stColSum = pInPixels[iRow*(iImg.getCols()) + (iCol-(kSize/2))];
          //compute sum, sum = prevSum - prev1stColSum + lastColSum & save values for next iteration
          sum = prevSum - prev1stColSum + lastColSum;
          prevSum = sum;
          prev1stColSum = nb1stColSum;
        }
        outPixSum = sum;
        return outPixSum;
}

//kernel separable filter applied vertically //a 7x1 filter is slided vertically in the image
int kernelSepSlideWinVer(unsigned int *&pInPixels,int iCol,int iRow,int imgWidth,int imgHeight){
        int outPixSum=0;
        static int prev1stRowSum,prevSum;
        int sum = 0;
        //unsigned char *pInPixels = iImg.getPixels();

        //kernel separable details
        int ker[7] = {1,1,1,1,1,1,1};
        int kSize = 7;

        //first pixel of the row, iCol = 3
        if( (kSize/2) == iRow){
                //Compute the full kernel and save the prev1stRowSum and the prevSum
                for( int r=-(kSize/2); r<=(kSize/2); r++)
                {
                  sum += (pInPixels[(iRow+r)*imgWidth + (iCol)]*ker[r+(kSize/2)]);
                  if((r+(kSize/2)) == 0)
                    prev1stRowSum = sum; //which is the first convolved value
                }
                //save the prevSum
                prevSum = sum;
        }//not a first pixel of the row, iCol!=3
        else{
                int lastRowSum=0,nb1stRowSum=0;
                //Compute the last col sum, lastRowSum
                lastRowSum = pInPixels[((iRow+(kSize/2))*(imgWidth)) + (iCol)];
                nb1stRowSum = pInPixels[((iRow-(kSize/2))*(imgWidth)) + (iCol)];

                //compute sum, sum = prevSum - prev1stRowSum + lastRowSum
                sum = prevSum - prev1stRowSum + lastRowSum;
                prevSum = sum;
                prev1stRowSum = nb1stRowSum;
        }

        outPixSum = sum;
        return outPix;
}

//do the main convolution operation here we will clock this function
//whoever uses this method should release the returned pointer
unsigned char *convolveImage(image &iImg){

  unsigned char *opPixels = NULL;
  unsigned int *tmpPixels = NULL;
  int kSize = 7;
  int r,c;
  int sum;
  //allocate memory for the out Pixel pointer
  opPixels = new unsigned char[iImg.getRows() * iImg.getCols()];
  tmpPixels = new unsigned int[iImg.getRows() * iImg.getCols()];

  //Do convolve operation horizontally
  for(r=0; r<(iImg.getRows()); r++)  //rows will be the same here
    for(c=3 ; c<(iImg.getCols()-3); c++)
        tmpPixels[r*iImg.getCols()+c] = kernelSepSlideWinhor(iImg,c,r);

  //Do convolve operation vertically
  for(c=3; c<(iImg.getRows()-3); c++)  //rows will be the same here
   for(r=3 ; r<(iImg.getCols()-3); r++)
        opPixels[r*iImg.getCols()+c] = (unsigned char)(kernelSepSlideWinVer(tmpPixels,c,r,
                                                        iImg.getCols(),iImg.getRows())/(kSize*kSize));

  delete(tmpPixels);
  return opPixels;

}
```

**Output for version 3:-**

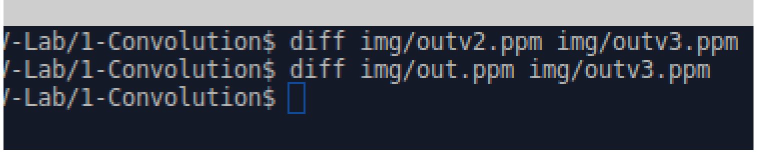
Generates a smoothed output image named, **outv3.ppm**, which is a smoothed version of input image **bridge.ppm**. Which looks as below:-



**Output Validation:-**

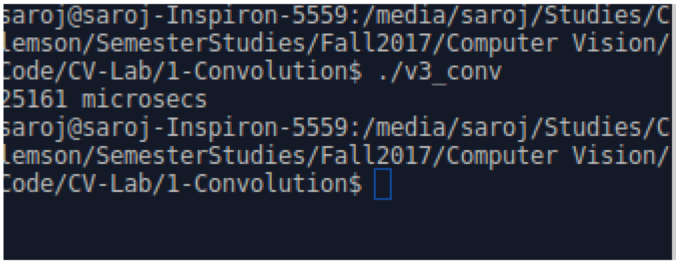
Linux version of diff on **out.ppm** and **outv2.ppm** proves both the files are equal. Diff on **out.ppm** and **outv3.ppm** says both are equal. Diff on **outv2.ppm** and **outv3.ppm** is also successful.

*My machine result:-*



```
/-Lab/1-Convolution$ diff img/outv2.ppm img/outv3.ppm
/-Lab/1-Convolution$ diff img/out.ppm img/outv3.ppm
/-Lab/1-Convolution$ 
```

**Time Clocked:-** **25161 microsecs**



```
saroj@saroj-Inspiron-5559:/media/saroj/Studies/C
lemson/SemesterStudies/Fall2017/Computer Vision/
Code/CV-Lab/1-Convolution$ ./v3_conv
25161 microsecs
saroj@saroj-Inspiron-5559:/media/saroj/Studies/C
lemson/SemesterStudies/Fall2017/Computer Vision/
Code/CV-Lab/1-Convolution$ 
```

**Conclusion:-**

| | Time Taken(µs) | %time taken vs V1 |
|---|---|---|
| 2dConvV1(basic 2d Convolve) | **103516** | 0 |
| 2d ConvV2(kernel Separable) | **46948** | 54% |
| 2d ConvV3<br>(separable + sliding Window) | **25161** | 75% |

Thus we can observe that the version is 75% more efficient i.e. takes .25 times of the basic 2d Convolution algorithm. Therefore a combination of separable and sliding window operation is the most efficient way for the convolution operation on the image.