SPATIAL FILTERING

PATRICK KEVORKIAN

TOPICS

- Introduction
- Brief Code review
- Max Filter
- Min Filter
- Median Filter
- Laplacian Filter
- Robinson
- Sobel
- High Boost Laplacian
- Prewitts

SPATIAL FILTERS

- Grid imposed on area of pixel values around origin pixel
- Calculation
- Grid moves to next pixel in image array
- Calculations in examples done in 2 ways
- Min, Max, and Median
- Convolution (element wise)

CODE REVIEW

PYTHON

- Aquire image, convert to greyscale (using Skimage)
- Show original image
- Define Grids as arrays
- No padding used
- Iterate through image array (Col wise) without going to very edge
- Apply filter to grid array values
- Place new value in new image array
- Display new image

CODE REVIEW

GRIDS

```
def threeXthreeGrid(r,c,arr):
    threeXthreeEle =  [ arr[r-1][c-1],arr[r-1][c],arr[r-1][c+1],arr[r][c-1],arr[r][c],arr[r][c+1],arr[r+1]
[c-1],arr[r+1][c],arr[r+1][c+1] ]
    return threeXthreeEle

def twoXtwoGrid(r,c,arr):
    twoXtwoEle = [arr[r][c],arr[r][c+1],arr[r+1][c],arr[r+1][c+1]]
    return twoXtwoEle
```

CODE REVIEW

ITERATING IMAGE ARRAY

```
#Begin loop: to iterate grid through image array
#This is done col wise moving over then down
#No padding used, filters dont hit very edge of array
#-----------------------------------------------

while c <= (len(img[:,1]-2)):

    c+=1
    limitCol = len(img[:,1])-2
    limitRow = len(img[1,:])-2

#-----------------------------------------------
    #Comment and uncomment for grid size

    #gridArr = twoXtwoGrid(r,c,img)
    gridArr = threeXthreeGrid(r,c,img)

#-----------------------------------------------
    #Comment and uncomment to use desired filter

    #newPixVal = maxFilter(gridArr)
    #newPixVal = medianSmooth(gridArr)
    #newPixVal = minFilter(gridArr)
    #newPixVal = Lap(gridArr,img)
    #newPixVal = robinson(gridArr,img)
    #newPixVal = SobelFilter(gridArr,img)
    #newPixVal = HigBoostLap(gridArr,img,A)
    newPixVal = prewittsFilter(gridArr,img)
#-----------------------------------------------

    newImage[r][c] = newPixVal

    if c == limitCol & r ==limitRow:
        break
    if c == limitCol :
        r+=1
        c=0

#end loop
```

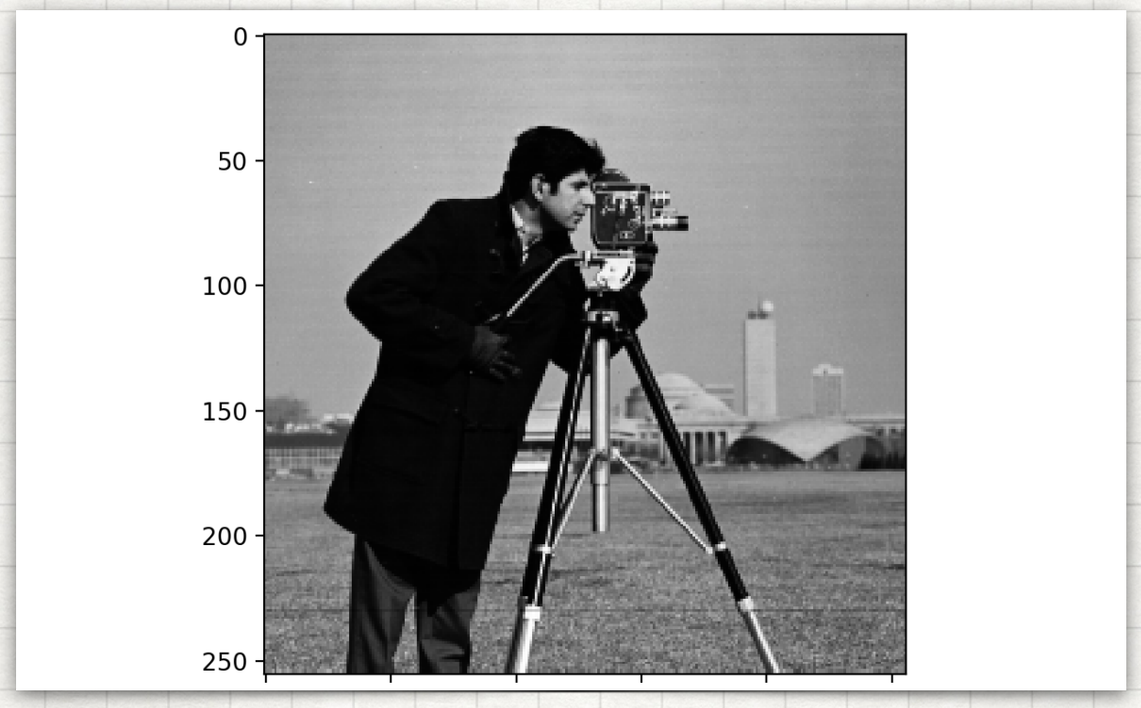
MAX FILTER

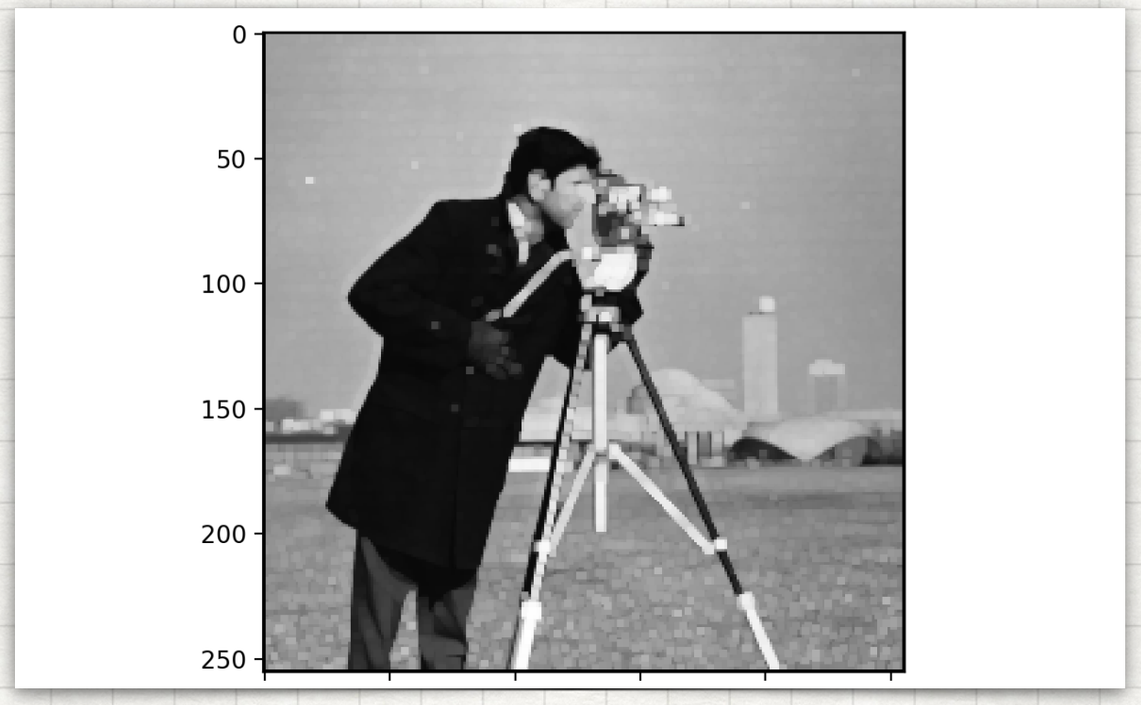
- Super imposed over elements around origin pixel
- Elements are then sorted (least to greatest)
- New pixel value is the largest value
- Value placed into new array at same position as origin pixel
- Grid moves on to next pixel

MAX FILTER

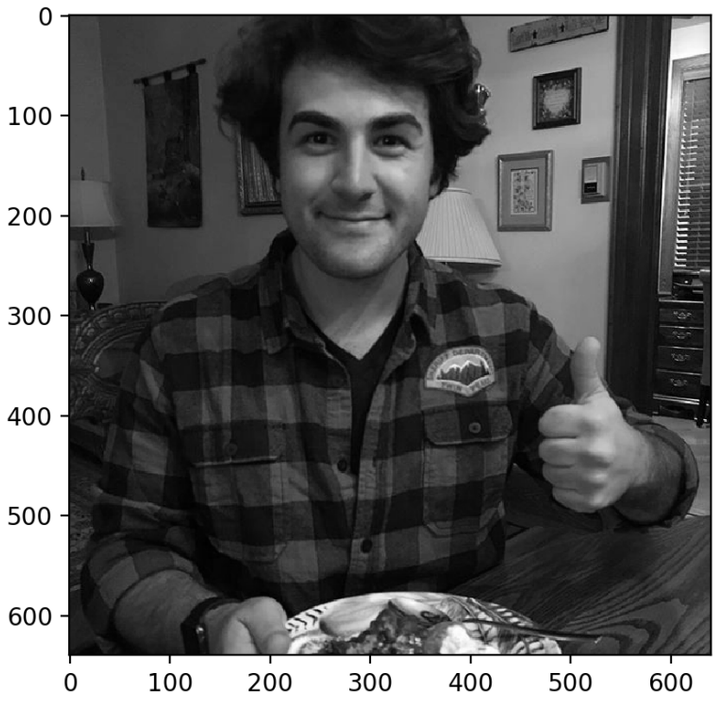
FUNCTION

```
#-------------------------------------

def maxFilter(arr):
    sortedAry = sorted(arr)
    maxEle = len(arr)-1
    return sortedAry[maxEle]

#-------------------------------------
```

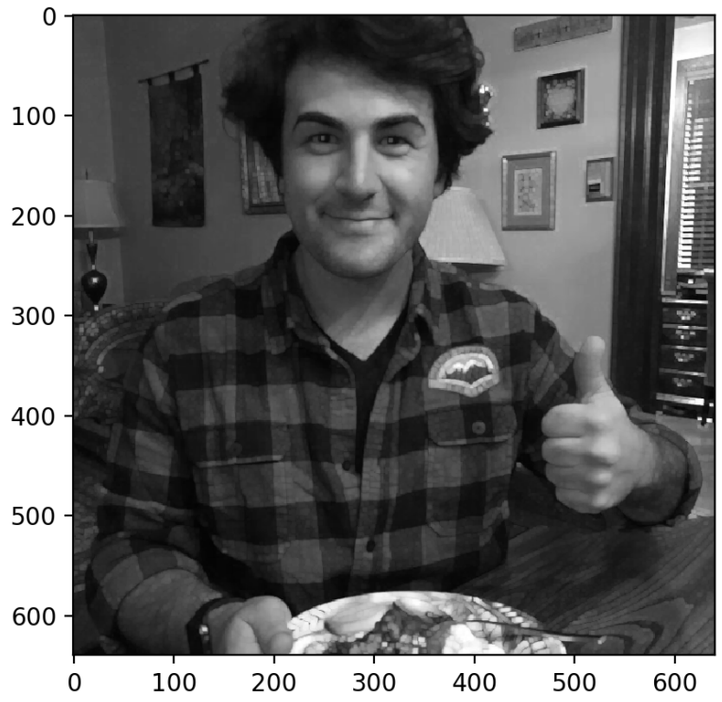




Original Image



New Image



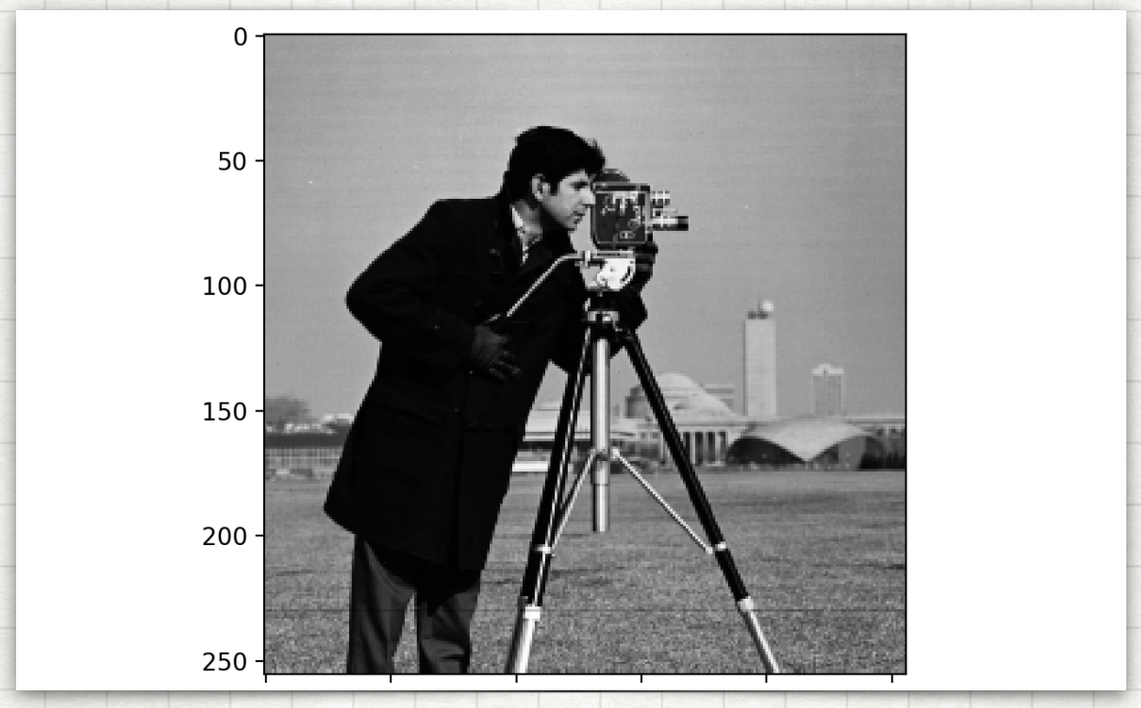
MIN FILTER

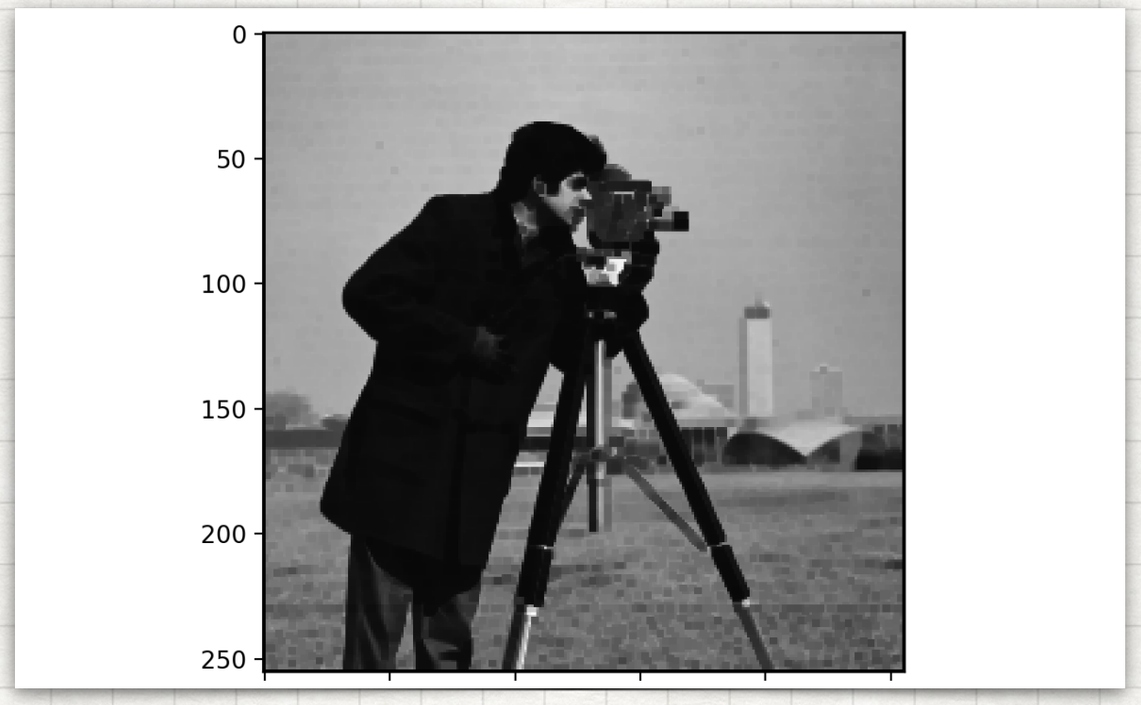
- Super imposed over elements around origin pixel
- Elements are then sorted (least to greatest)
- New pixel value is the smallest value
- Value placed into new array at same position as origin pixel
- Grid moves on to next pixel

MIN FILTER

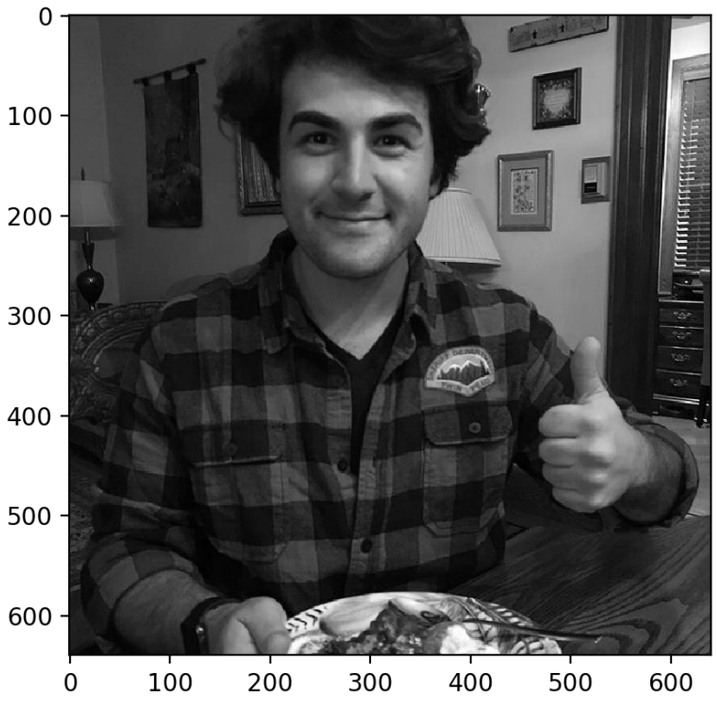
FUNCTION

```
#-------------------------------

def minFilter(arr):
    sortedAry = sorted(arr)
    return sortedAry[0]

#-------------------------------
```

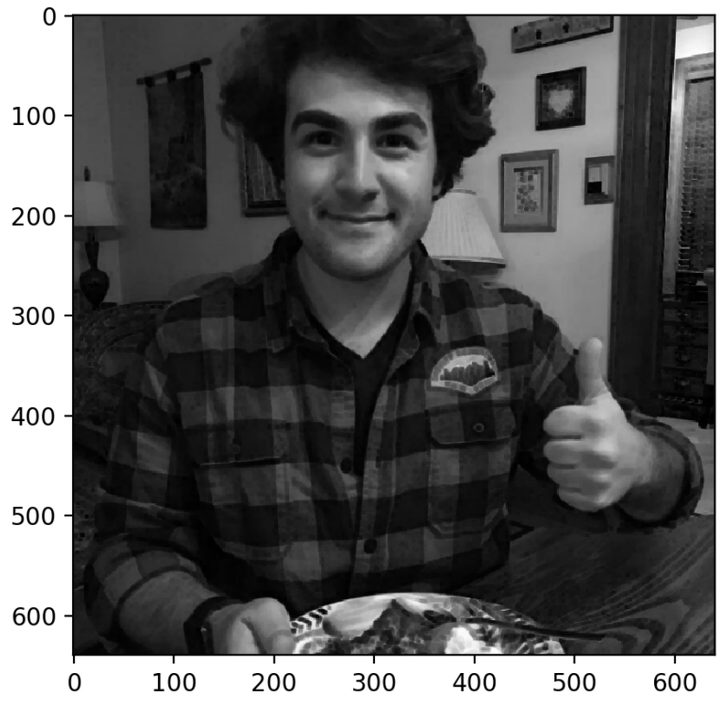




Original Image



New Image



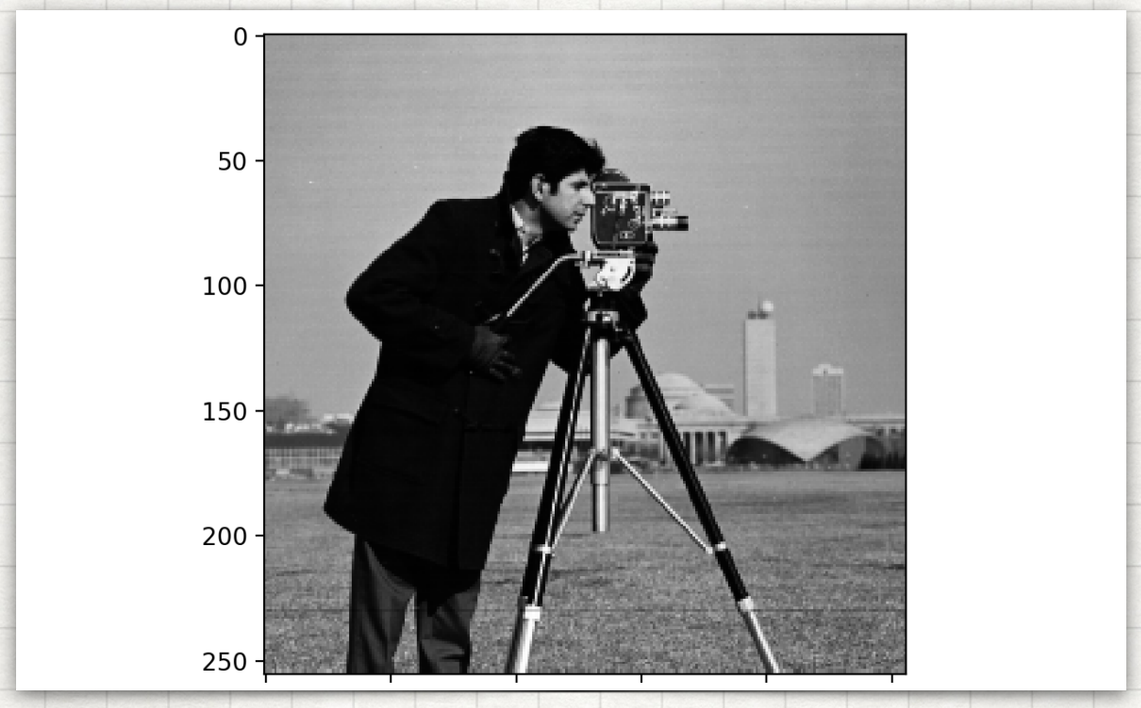
MEDIAN FILTER

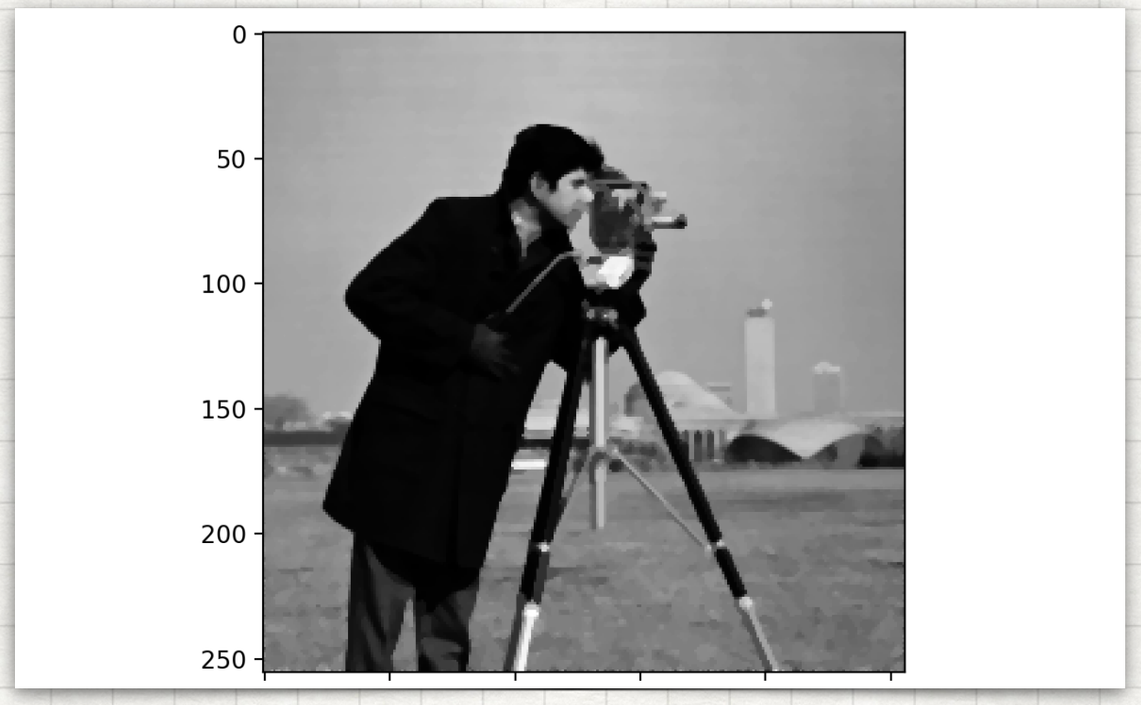
- Super imposed over elements around origin pixel
- Elements are then sorted (least to greatest)
- New pixel value is the median value of array elements
- In 3X3 the median value is (9/2)+1
- Value placed into new array at same position as origin pixel
- Grid moves on to next pixel

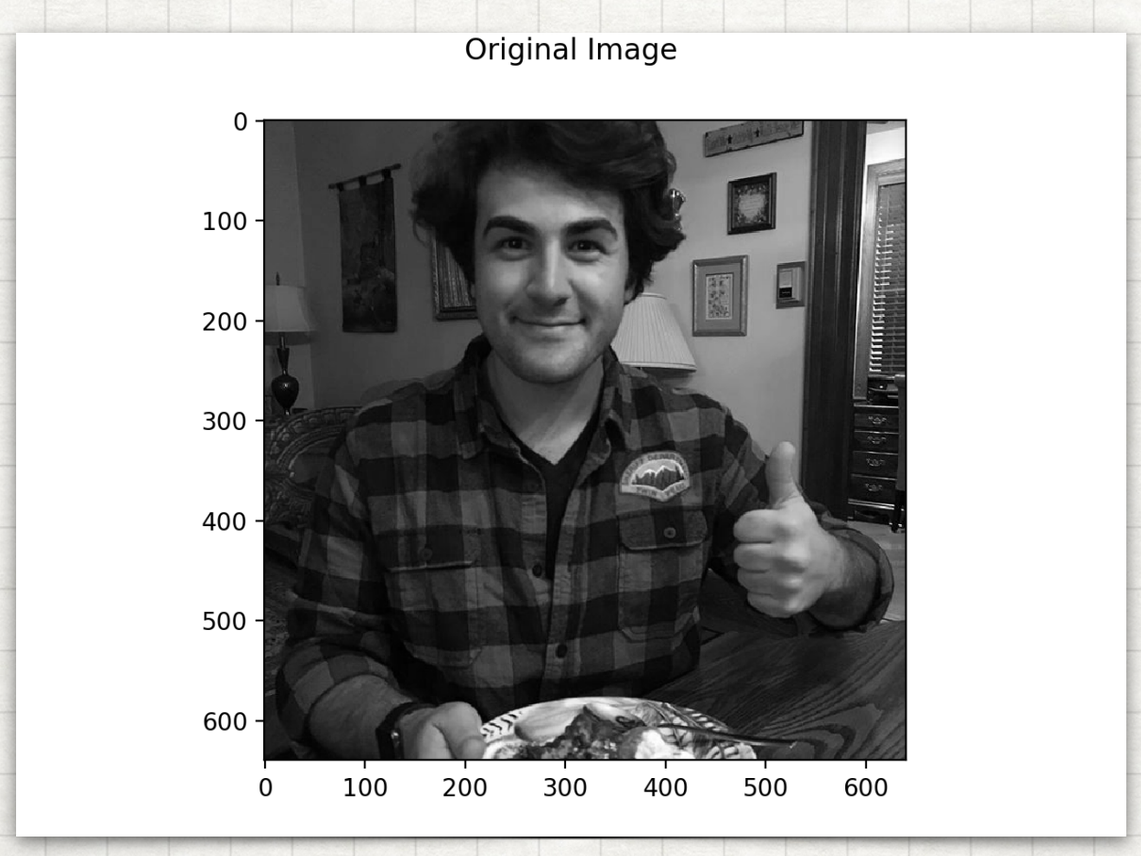
MEDIAN FILTER

FUNCTION

```
#-----------------------------------------------------------

def medianSmooth(arr):
    sortedAry = sorted(arr)
    medianEle = math.floor((len(arr)/2)+1)-1
    return sortedAry[medianEle]

#-----------------------------------------------------------
```


0
50
100
150
200
250


0
50
100
150
200
250


Original Image
0
100
200
300
400
500
600
0
100
200
300
400
500
600


0
100
200
300
400
500
600
0
100
200
300
400
500
600

LAPLACIAN FILTER

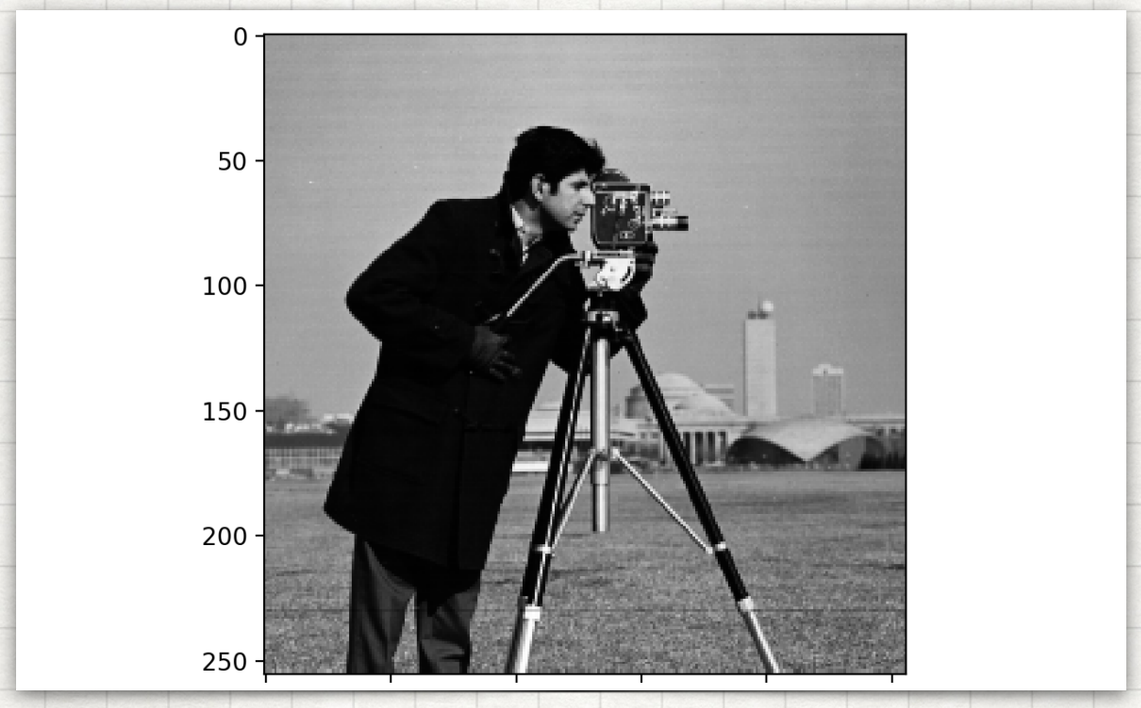
- the standard Lap Filter
- Supper imposed
- Elements of the filter and elements of pixels imposed are multiplied
- Then added
- Moves to next pixel

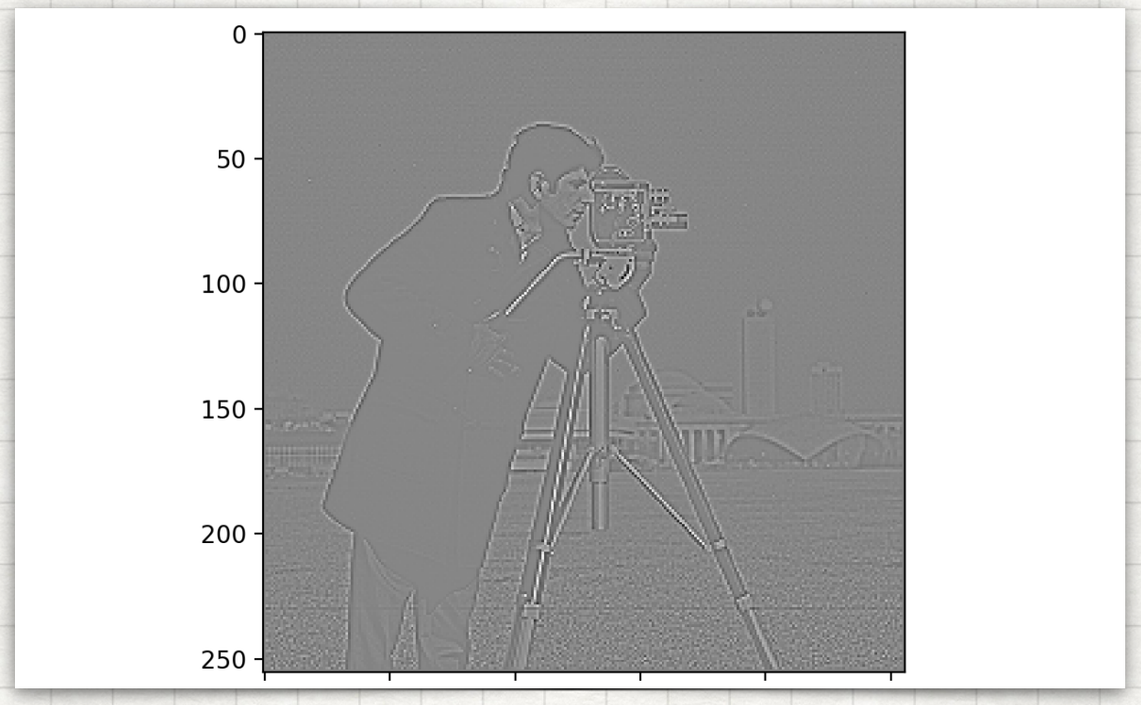
0	-1	0
-1	4	-1
0	-1	0

LAP FILTER

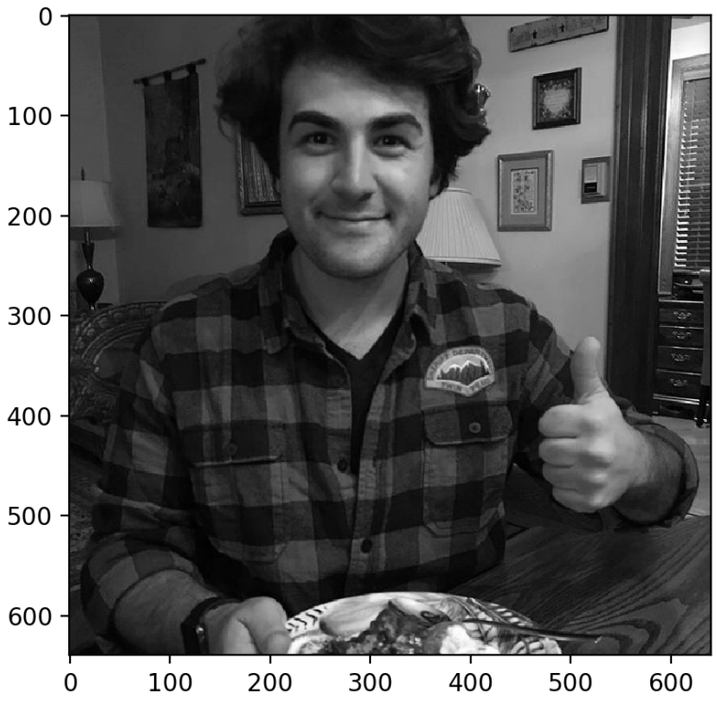
FUNCTION

```
#-------------------------------------------------------------------------

def Lap(arr,img):
    lapMask = [0,-1,0,-1,4,-1,0,-1,0]
    maskMult = [a*b for a,b in zip(arr,lapMask)]
    sumLap = sum(maskMult)
    return sumLap

#-------------------------------------------------------------------------
```

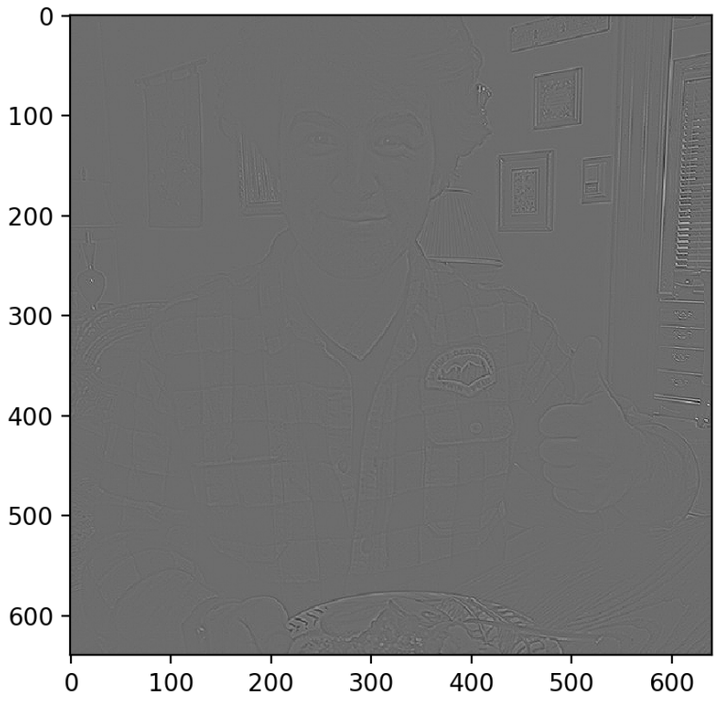




Original Image



New Image



ROBINSON

- 2X2 Grid
- 2 kernels
- Origin is filt[0][0]
- Like before the filter is applied by element wise multiplication and added together
- This is done with both filters and the absolute value of the each is added together.

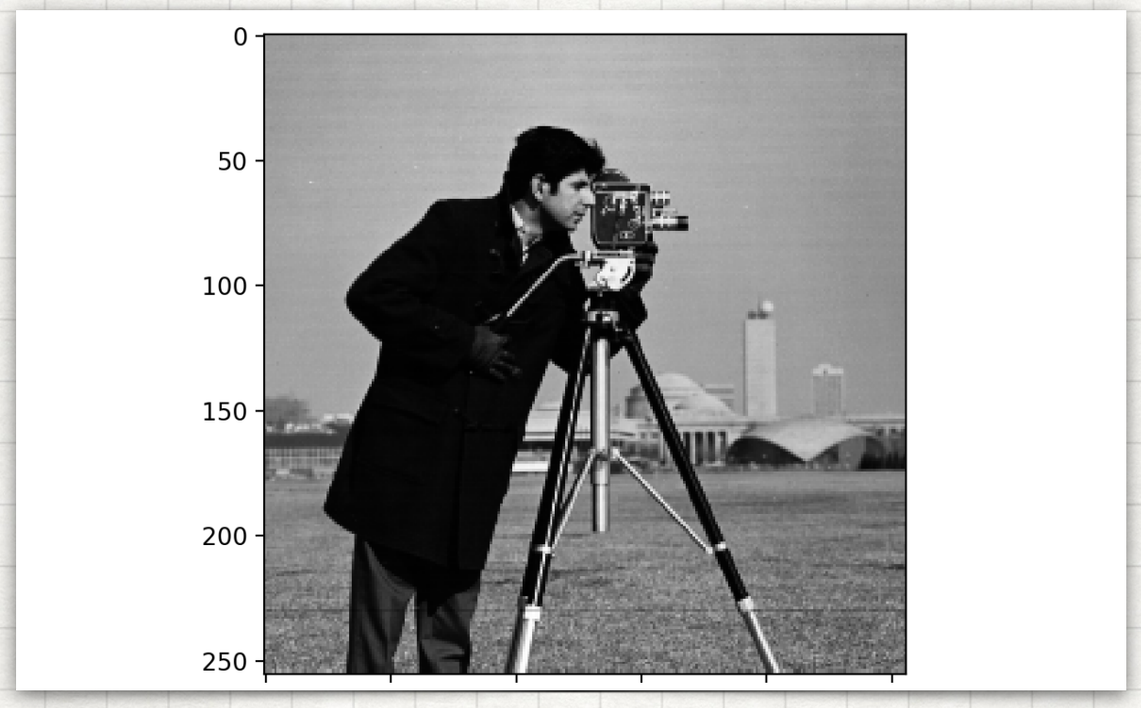
1	0
0	-1

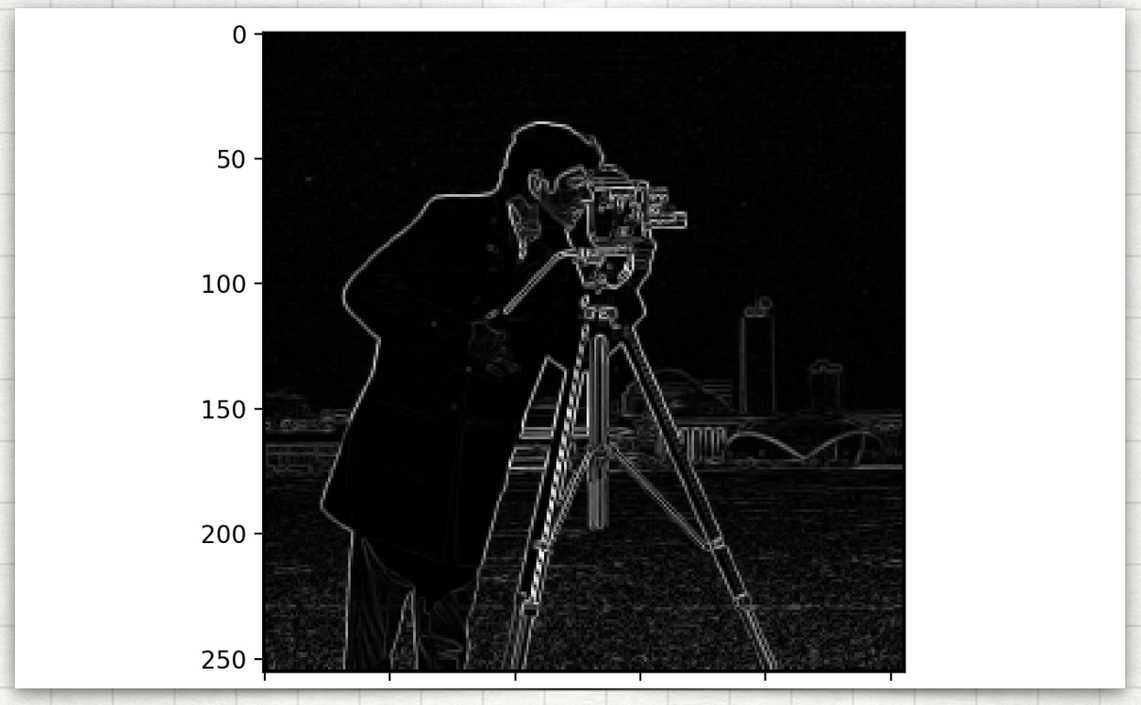
0	1
-1	0

ROBINSON

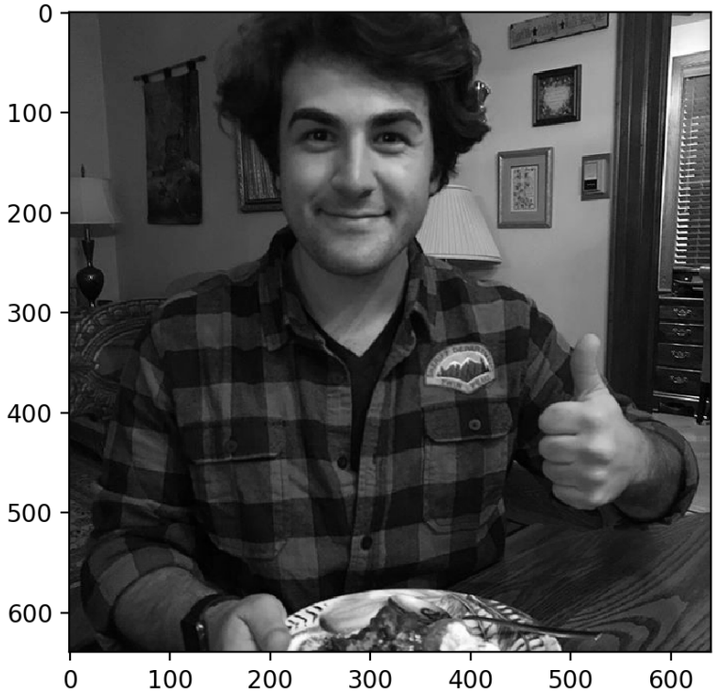
FUNCTION

```
#-------------------------------------------------------------------

def robinson(arr,img):
    rob1 = [1,0,0,-1]
    rob1Mult = [a*b for a,b in zip(arr,rob1)]
    sumRob1 = sum(rob1Mult)

    rob2 = [0,1,-1,0]
    rob2Mult = [a*b for a,b in zip(arr, rob2)]
    sumRob2 = sum(rob2Mult)

    return abs(sumRob1) + abs(sumRob2)

#-------------------------------------------------------------------
```

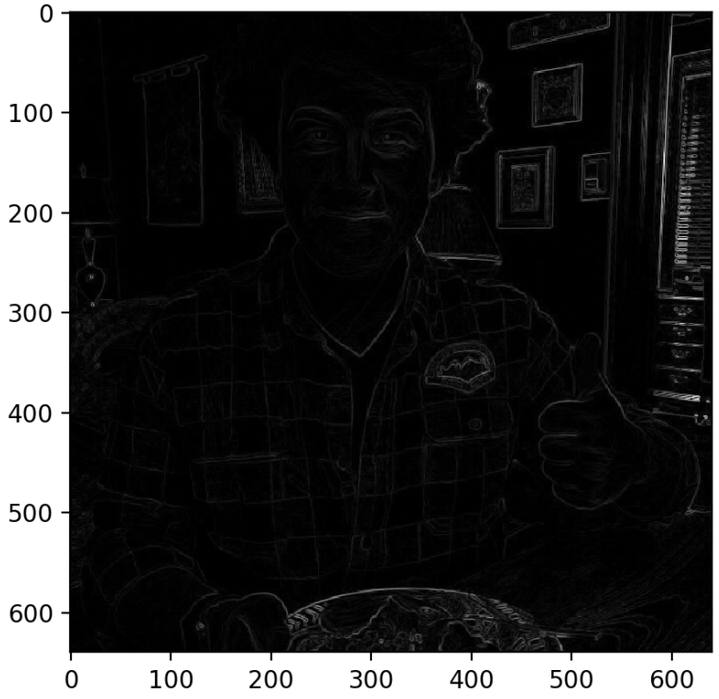




Original Image



New Image



SOBEL FILTER

- Isotropic 3X3 image gradient operator
- Two kernels
- Horizontal and vertical directions
- Element wise Mult
- Addition of results
- Absolute value taken

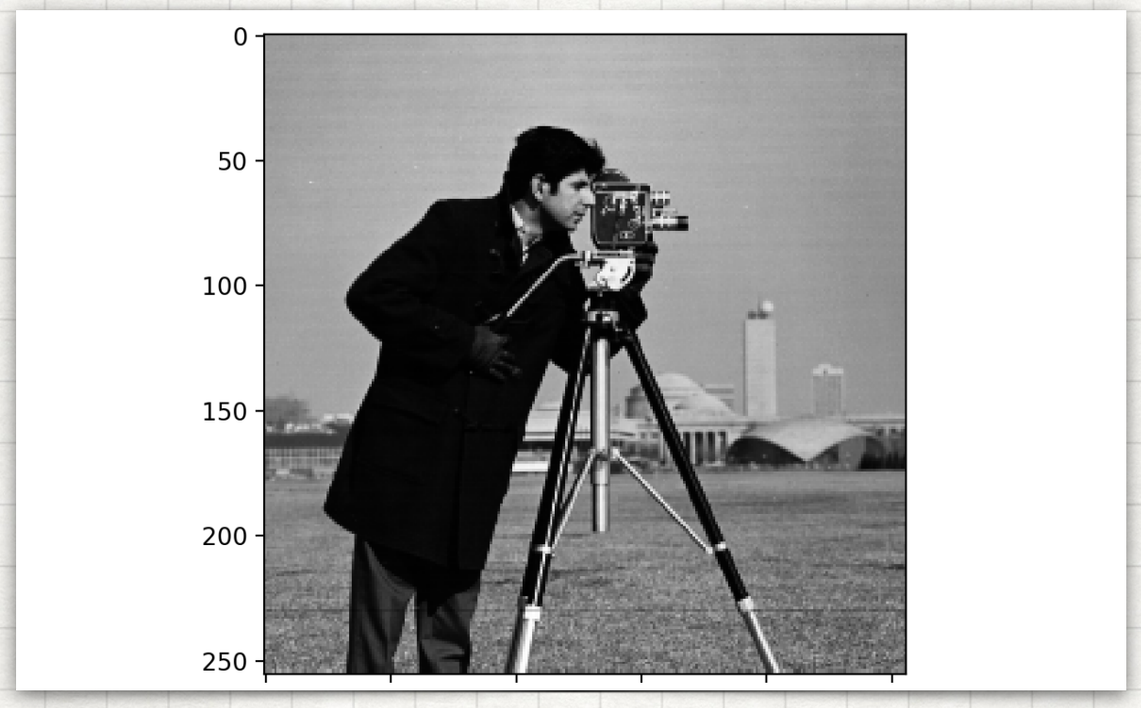
1	0	-1
2	0	-2
1	0	-1

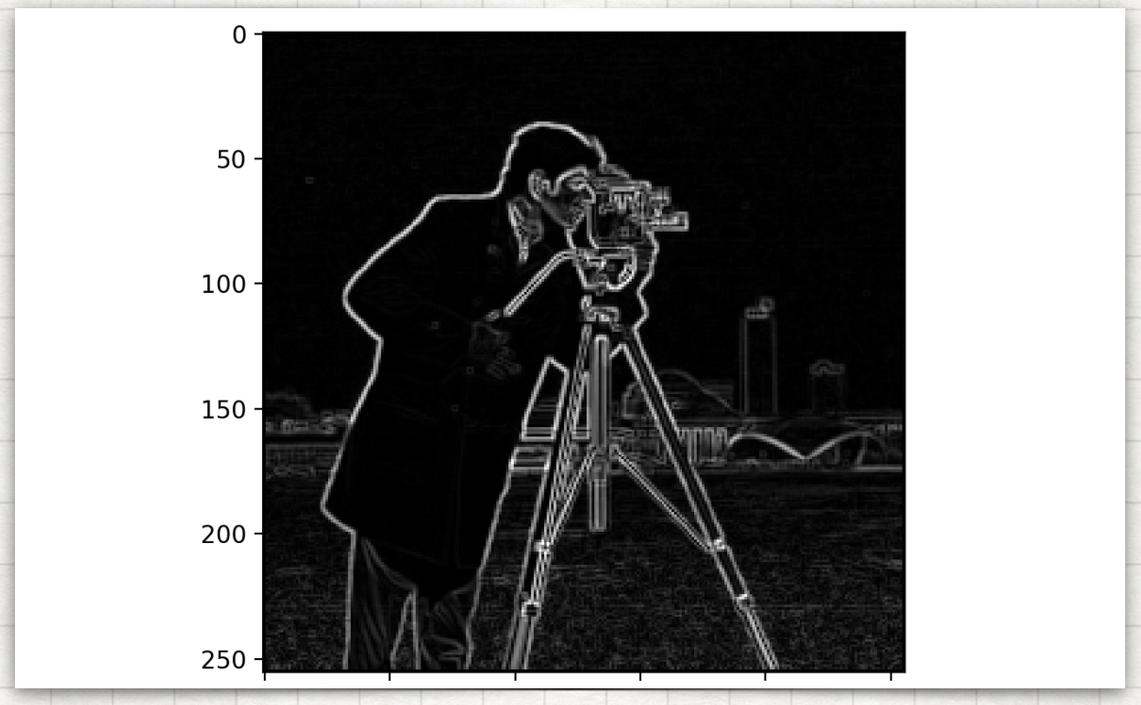
1	2	1
0	0	0
-1	-2	-1

SOBEL FILTER

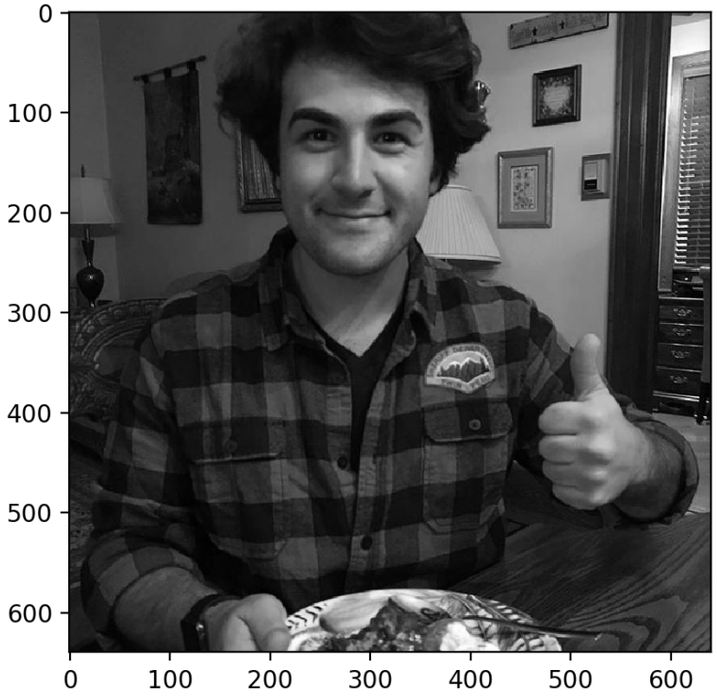
FUNCTION

```
#-----------------------------------------------------

def SobelFilter(arr,img):
    sobel1 = [1,0,-1,2,0,-2,1,0,-1]
    sobel1Mult = [a*b for a,b in zip(arr,sobel1)]
    sumSobel1 = sum(sobel1Mult)

    sobel2 = [1,2,1,0,0,0,-1,-2,-1]
    sobel2Mult = [a*b for a,b in zip(arr,sobel2)]
    sumSobel2 = sum(sobel2Mult)

    return abs(sumSobel1) + abs(sumSobel2)
#-----------------------------------------------------
```

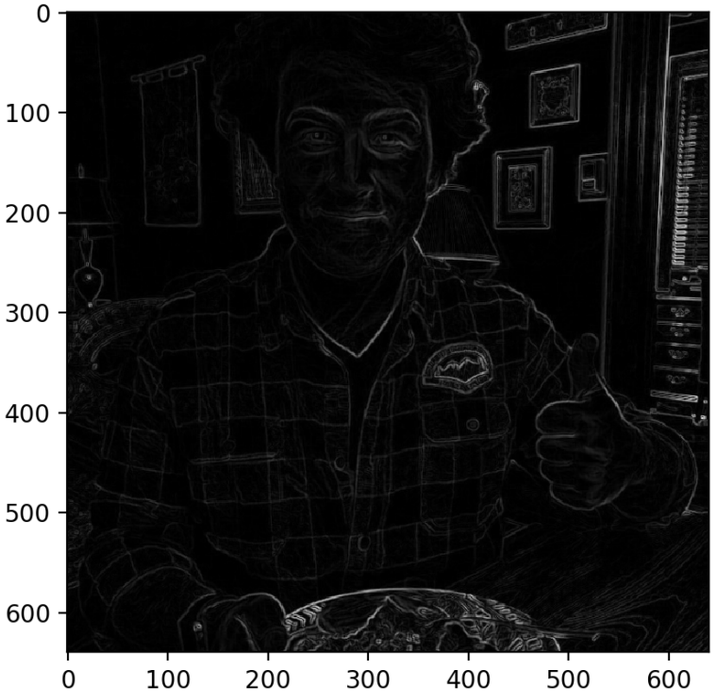




Original Image



New Image



HIGH BOOST LAP

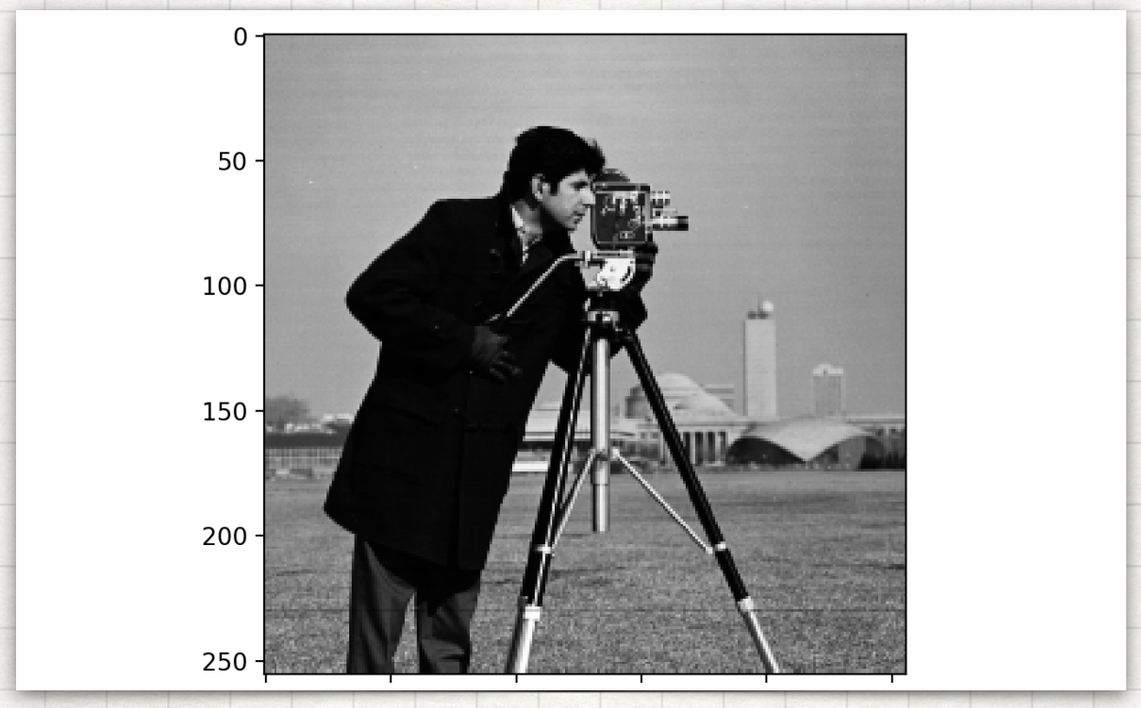
- Same element wise operations
- Important to emphasize high frequency components
- But also not eliminating low frequency components
- A is some constant (In example its 5)
- Low are suppressed, high are enhanced

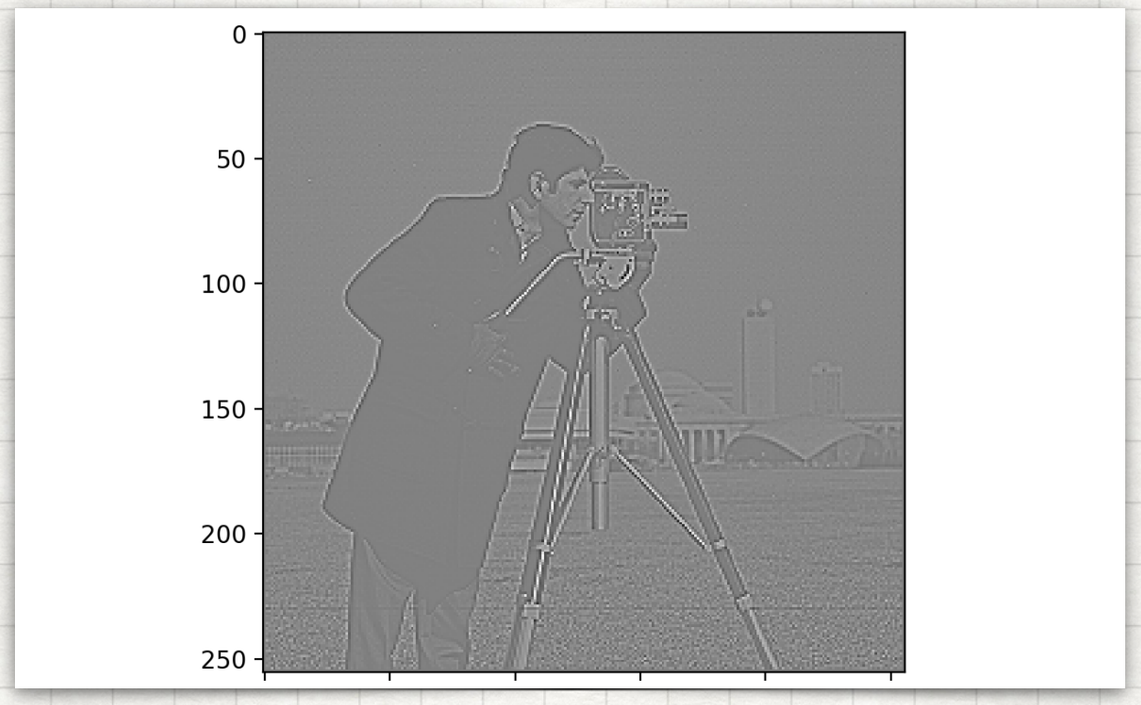
0	-A	0
-A	4A+1	-A
0	-A	0

HIGH BOOST LAP

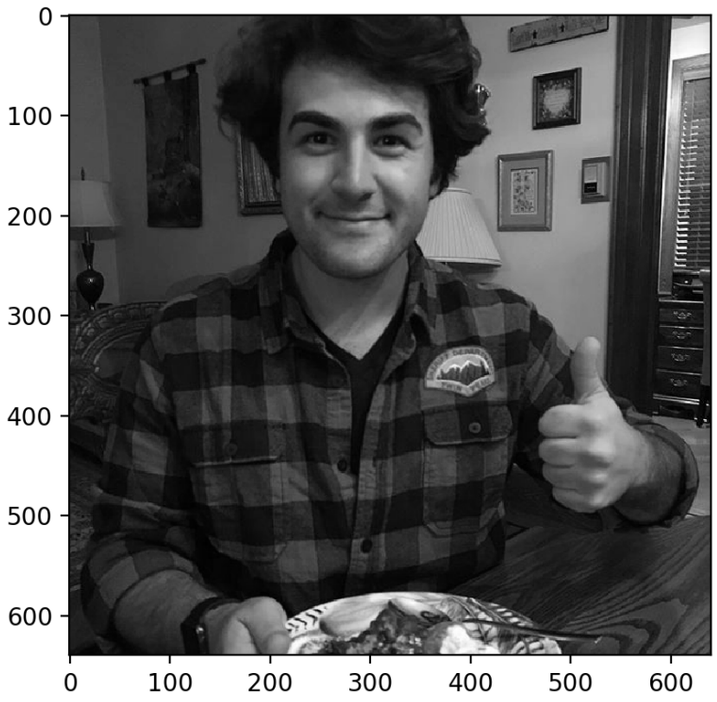
FUNCTION

```
#-------------------------------------------------------

def HigBoostLap(arr,img, A):
    HighBoostLapMask = [0,-A,0,-A,4*A+1,-A,0,-A,0]
    lapMaskMult = [a*b for a,b in zip(arr,HighBoostLapMask)]
    sumHighBoostLap = sum(lapMaskMult)
    return sumHighBoostLap

#-------------------------------------------------------
```

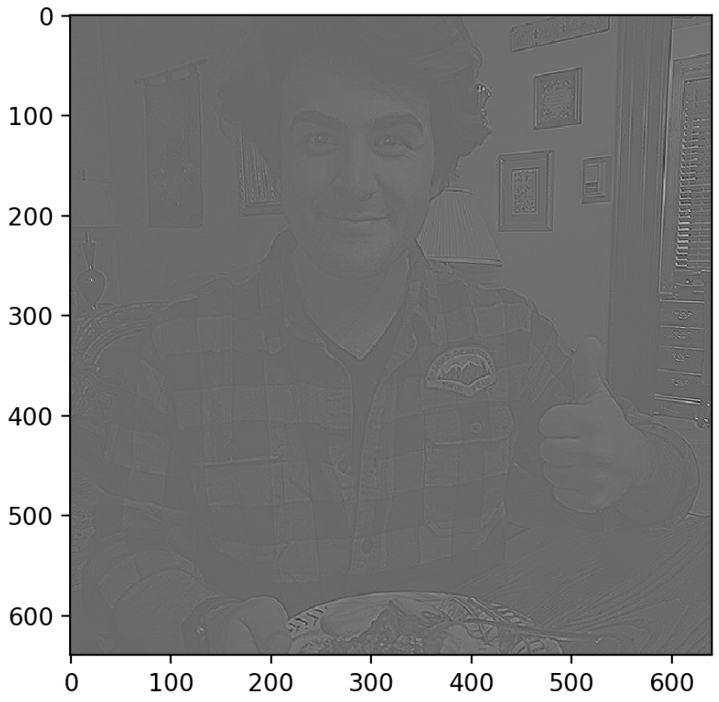




Original Image



New Image



PREWITTS FILTER

- Element wise operation
- Horizontal and vertical
- Similar operations and procedures as the previous edge detection filters

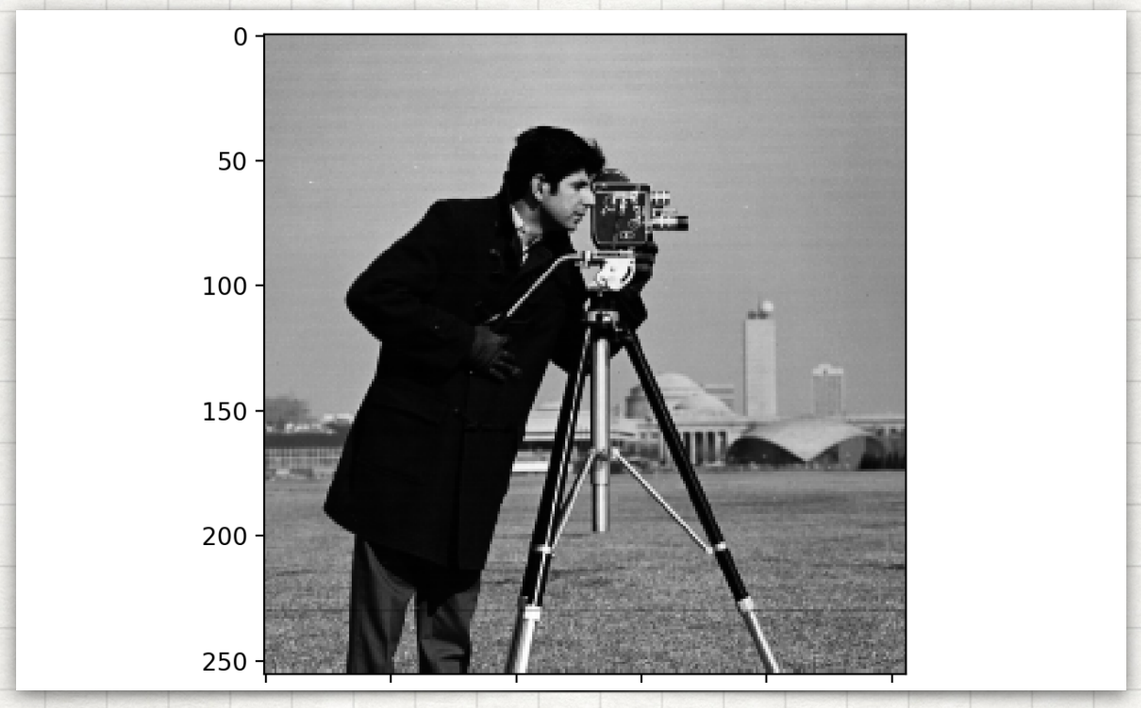
-1	-1	-1
0	0	0
1	1	1

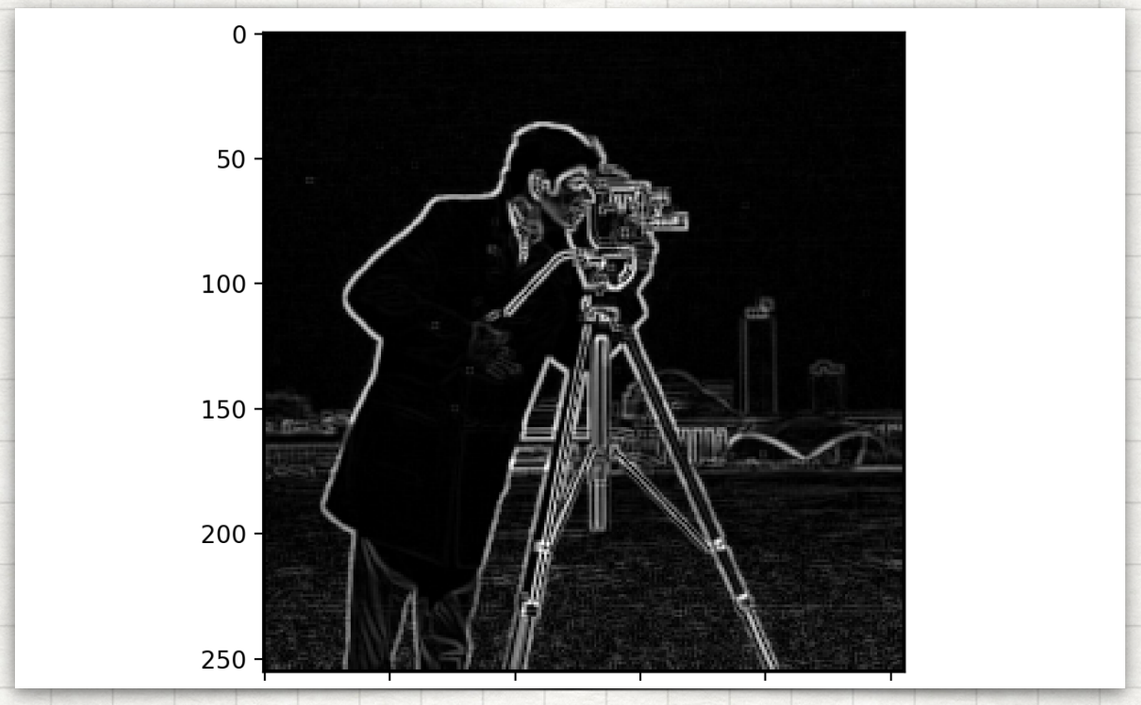
-1	0	1
-1	0	1
-1	0	1

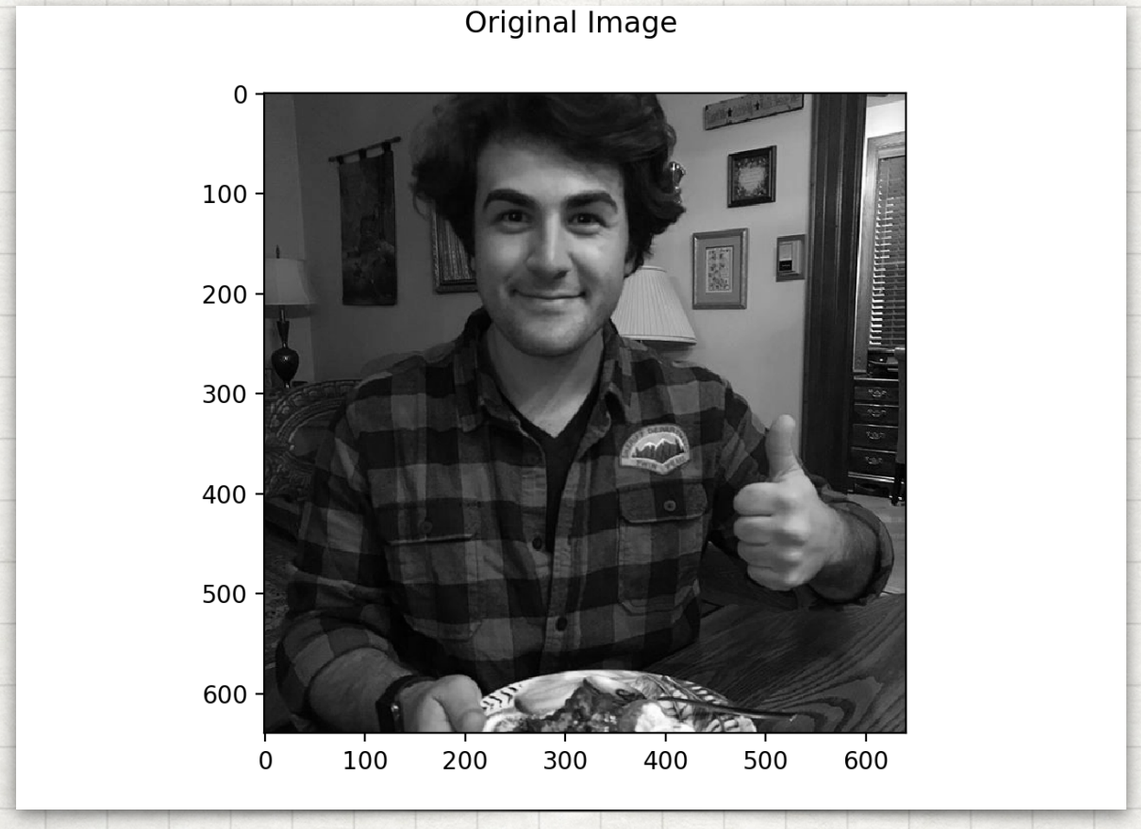
PREWITTS

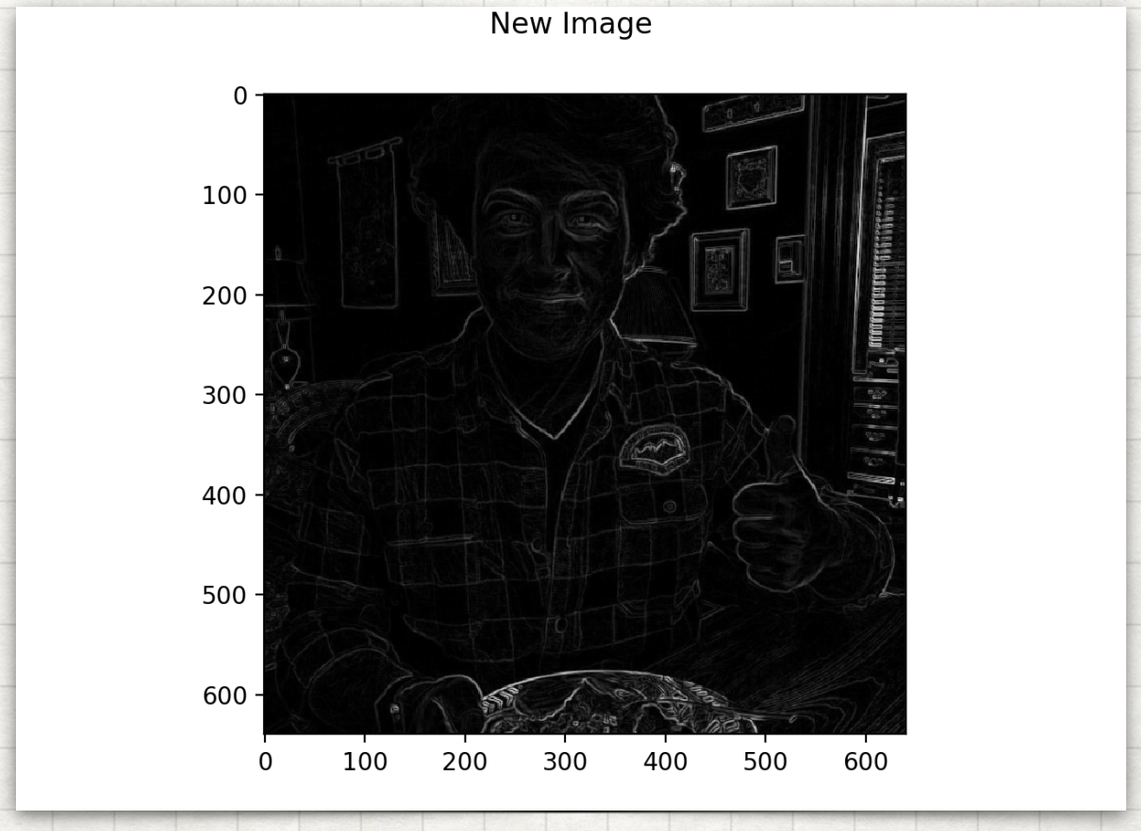
FUNCTION

```
#-------------------------------------------------

def prewittsFilter(arr, img):
    prewitt1 = [-1,-1,-1,0,0,0,1,1,1]
    prewitt1Mult = [a*b for a,b in zip(arr,prewitt1)]
    sumPrewitt1 = sum(prewitt1Mult)

    prewitt2 = [-1,0,1,-1,0,1,-1,0,1]
    prewitt2Mult = [a*b for a,b in zip(arr,prewitt2)]
    sumPrewitt2 = sum(prewitt2Mult)

    return abs(sumPrewitt1) + abs(sumPrewitt2)
```


Original Image
0
50
100
150
200
250


New Image
0
50
100
150
200
250


Original Image
0
100
200
300
400
500
600
0
100
200
300
400
500
600


New Image
0
100
200
300
400
500
600
0
100
200
300
400
500
600

COMMENTS?

THANK YOU