```python
#Applying sobel filter without cv2
import numpy as np
import cv2
import os


class SobelEdgeDetection():
    def __init__(self,filename):
        #constructor takes filename and assigns the orginal image and displays,
        #converts the original image to gray to apply sobel in only one channel
        self.__filename = filename
        self.__original_image = cv2.imread(self.__filename)
        self.__gray_image = cv2.cvtColor(self.__original_image,cv2.COLOR_BGR2GRAY)
        n_gray_image = np.float64(self.__gray_image)
        #padding with 1 as i am using 3*3 kernel, padding is done to work with border
        #elements
        self.__padded_image = np.pad(n_gray_image,((1,1),(1,1)),constant_values=0)
        self.__gradient_y_image = np.zeros_like(n_gray_image)
        self.__gradient_x_image = np.zeros_like(n_gray_image)
        self.__gradient_image = np.zeros_like(n_gray_image)


    #function that does the filter 2d operation in for loops
    def convol(self):
        for i in range(self.__gray_image.shape[0]):
            for j in range(self.__gray_image.shape[1]):
                #multiplying element wise and adding, by selecting a 3*3 from the padded image
                self.__gradient_y_image[i,j] = np.sum(self.__padded_image[i:i+3,j:j+3]*self.__filter_y)
                self.__gradient_x_image[i,j] = np.sum(self.__padded_image[i:i+3,j:j+3]*self.__filter_x)
                #final gradient is taken as sqrt of sum of squares of x and y
                self.__gradient_image[i,j] = np.sqrt(np.power(self.__gradient_x_image[i,j],2) + np.power(self.__gradient_y_image[i,j],2))

    def sobel(self,filter_x,filter_y):
        #assigns the value of filter to the member variables
        self.__filter_y = filter_y
        self.__filter_x = filter_x
        #calls the convolution function
        self.convol()
        #ALl the images are in float, taking their absolute and converting them into uint8
        self.__gradient_x_image = np.uint8(np.absolute(self.__gradient_x_image))
        self.__gradient_y_image = np.uint8(np.absolute(self.__gradient_y_image))
        self.__gradient_image = np.uint8(self.__gradient_image)

    def flip_image(self):
        #edges generated is white on black, making it black on white according to the question
        self.__flipped_image = np.uint8(np.where(self.__gradient_image > 127,0,255))

    def display_final(self):
        cv2.imshow("input image",self.__original_image)
        cv2.imshow("x",self.__gradient_y_image)
        cv2.imshow("y",self.__gradient_x_image)
        cv2.imshow("final_edge_image",self.__gradient_image)
        cv2.imshow("final_edge_image_flipped",self.__flipped_image)

    #finding the first name of the loaded image
```

```
    def __find_write_filename(self):
        first_name = os.path.basename(self.__filename)
        first_name = first_name.split('.',1)[0]
        return first_name

    def write_image(self):
        self.__write_filename = self.__find_write_filename() + "-sobel.png"
        if(cv2.imwrite(self.__write_filename,self.__flipped_image)):
            print("Successfully saved")
        else:
            print("Unsuccessfull")

if __name__ == "__main__":
    #defining the 3*3 kernel for sobel filetring in y and x respectively
    sobel_filter_x = np.asarray([[-1.,0,1.],[-2.,0,2.],[-1.,0,1.]])
    sobel_filter_y = np.asarray([[1.,2.,1.],[0.,0.,0.],[-1.,-2.,-1.]])
    filename = "/home/akshay/Downloads/CV/ps-3-q/ps3-images/professor.png"

    #creating the object
    edge_detection = SobelEdgeDetection(filename)
    #This operation takes some time while running
    edge_detection.sobel(sobel_filter_x,sobel_filter_y)
    #makes black edges on white background
    edge_detection.flip_image()
    edge_detection.display_final()
    edge_detection.write_image()

    cv2.waitKey(0)
```