CSC420 Assignment 1

Ruijie Sun

Student number 1003326046

Utorid sunruij2

Q1

(a) If it is not separable, it costs O(m^2n^2) since we need to do m^2 times operation to each pixel.

(b) if it is separable, it costs O($2mn^2$) = O(mn^2) we only do the 1-d horizontal convolution and 1-d vertical convolution FOR each pixel in instead.

Q2

Canny edge detection steps are following:

1. **Removing noise**, in this step, we apply horizontal and vertical direction derivative of Gaussian into the image to remove the influence by the noise.
2. **Finding magnitude and orientation of gradient,** in this step we will compute the magnitude and orientation on each pixel, so that we can determine the strength of each pixel and its direction.
3. **Non-max suppression,** in this step, for each pixel, we check if it is local maximum along gradient direction. If it is the local maximum, it will remain. So that we can locate the edge more accurately by removing the non max pixel.
4. **Hysteresis Thresholding,** in this step, we “link” the edges by setting two threshold, high threshold and low threshold. If one pixel’s magnitude is larger than the high threshold, it is marked as strong edge pixel. If it is between high threshold and low threshold and it is neighbor of a strong edge pixel, it is marked as weak edge pixel. Otherwise it is suppressed.

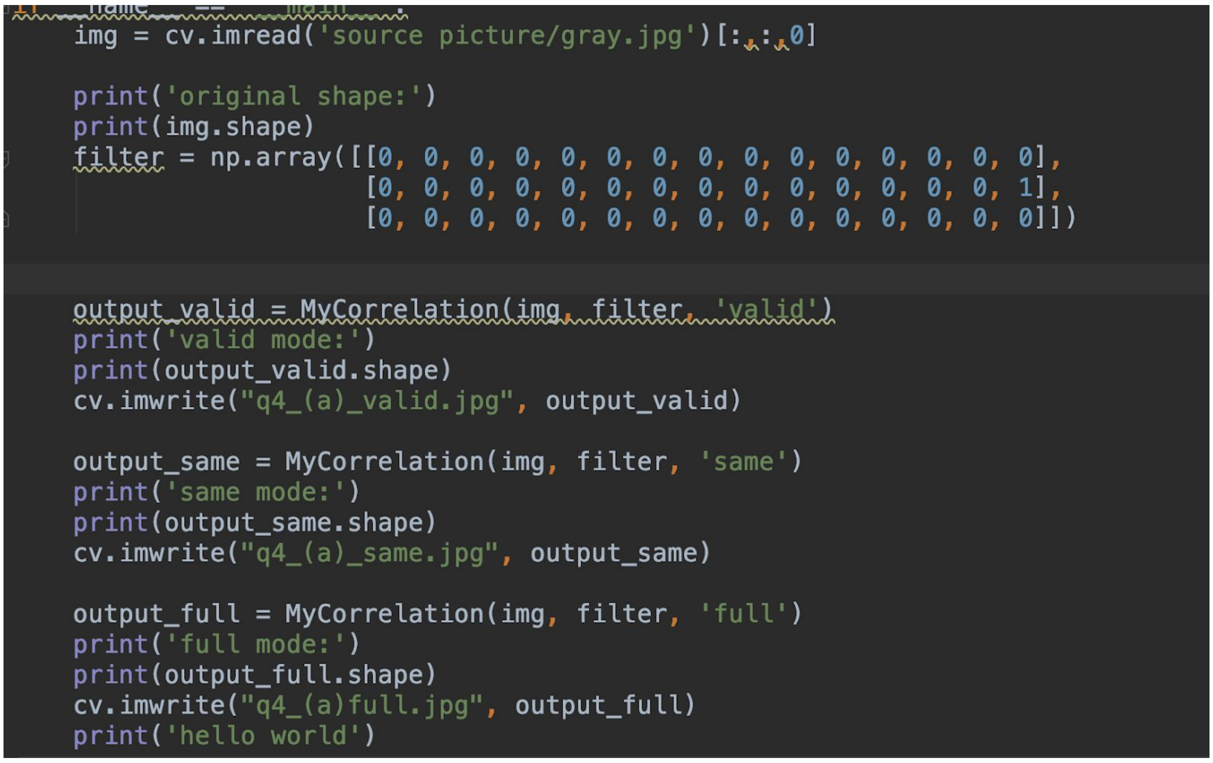
Q3

The Laplacian of Gaussian approach find edges by detecting zero-crossing.
The zero-crossing is the place where second derivative changes sign, indicating that in this point, the first derivative is **local maximum or local minimum**. In other words, **the pixel magnitude changes significantly** which tells that this pixel is the edge pixel.

Q4

(a)

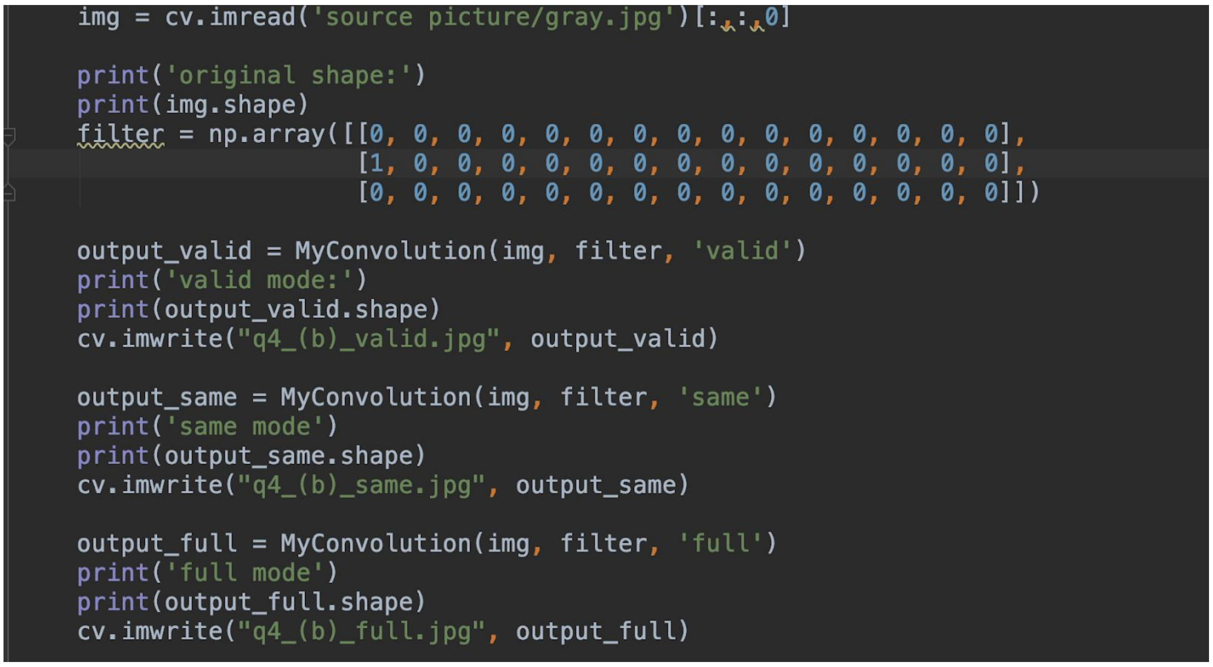
shift test



```
if __name__ == '__main__':
    img = cv.imread('source picture/gray.jpg')[:,:,0]

    print('original shape:')
    print(img.shape)
    filter = np.array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
                       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])


    output_valid = MyCorrelation(img, filter, 'valid')
    print('valid mode:')
    print(output_valid.shape)
    cv.imwrite("q4_(a)_valid.jpg", output_valid)

    output_same = MyCorrelation(img, filter, 'same')
    print('same mode:')
    print(output_same.shape)
    cv.imwrite("q4_(a)_same.jpg", output_same)

    output_full = MyCorrelation(img, filter, 'full')
    print('full mode:')
    print(output_full.shape)
    cv.imwrite("q4_(a)full.jpg", output_full)
    print('hello world')
```

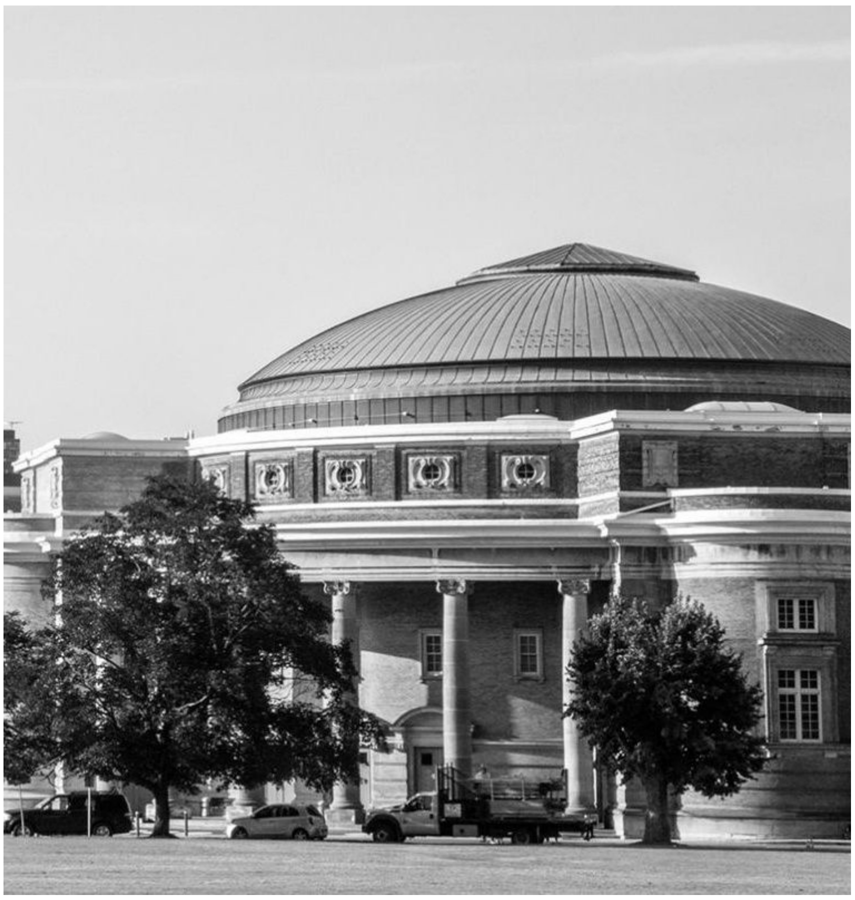
image shifts as expected

(b) shift test



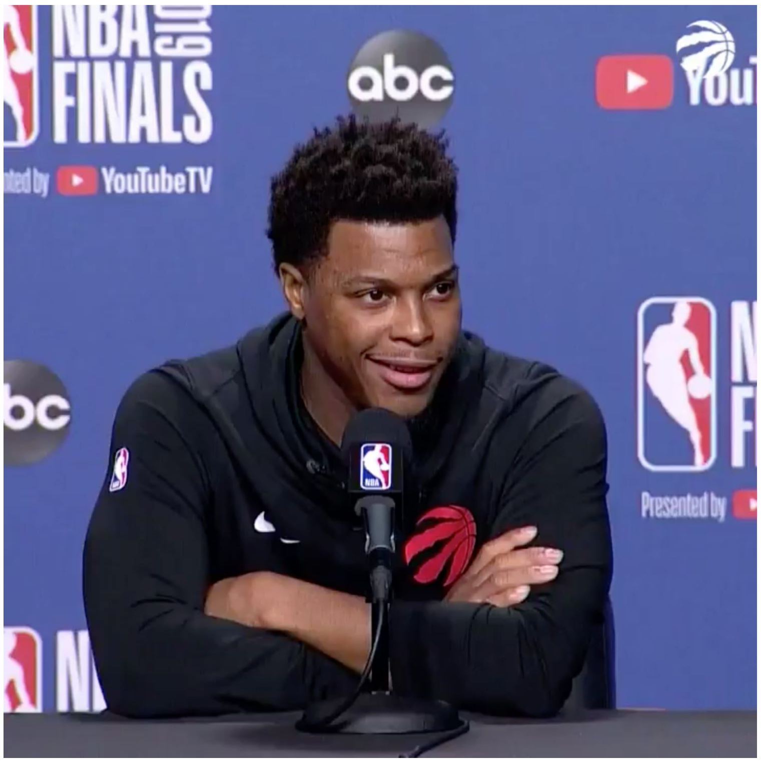
```
img = cv.imread('source picture/gray.jpg')[:,:,0]

print('original shape:')
print(img.shape)
filter = np.array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                   [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                   [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])

output_valid = MyConvolution(img, filter, 'valid')
print('valid mode:')
print(output_valid.shape)
cv.imwrite("q4_(b)_valid.jpg", output_valid)

output_same = MyConvolution(img, filter, 'same')
print('same mode')
print(output_same.shape)
cv.imwrite("q4_(b)_same.jpg", output_same)

output_full = MyConvolution(img, filter, 'full')
print('full mode')
print(output_full.shape)
cv.imwrite("q4_(b)_full.jpg", output_full)
```

image shifts as expected

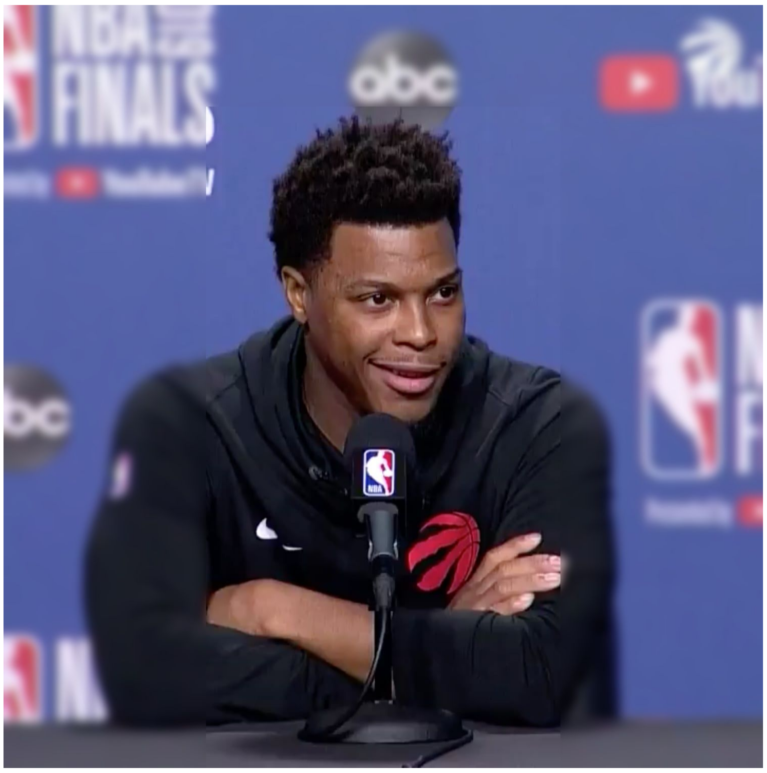


(c) The filter I choose is the **size 51x51 Gaussian filter (an approximation of a 2d Gaussian function with sigma 10)**. The Gaussian filter enables the **nearest neighboring pixels** to have the **most influence** on the output to make output smooth. And when we choose a relatively large filter size and sigma, the **smooth effect will become blurred**.

Original



Output



Q5
(a) The convolution computation requires K^2 operation for each pixel. Instead of this, this operation can be replaced by performing a **1D horizontal convolution** followed by a **1D vertical convolution**, costing 2k operations for each pixel. A filter is **separable** if there is only **one singular value is non-zero in its singular value decomposition.**

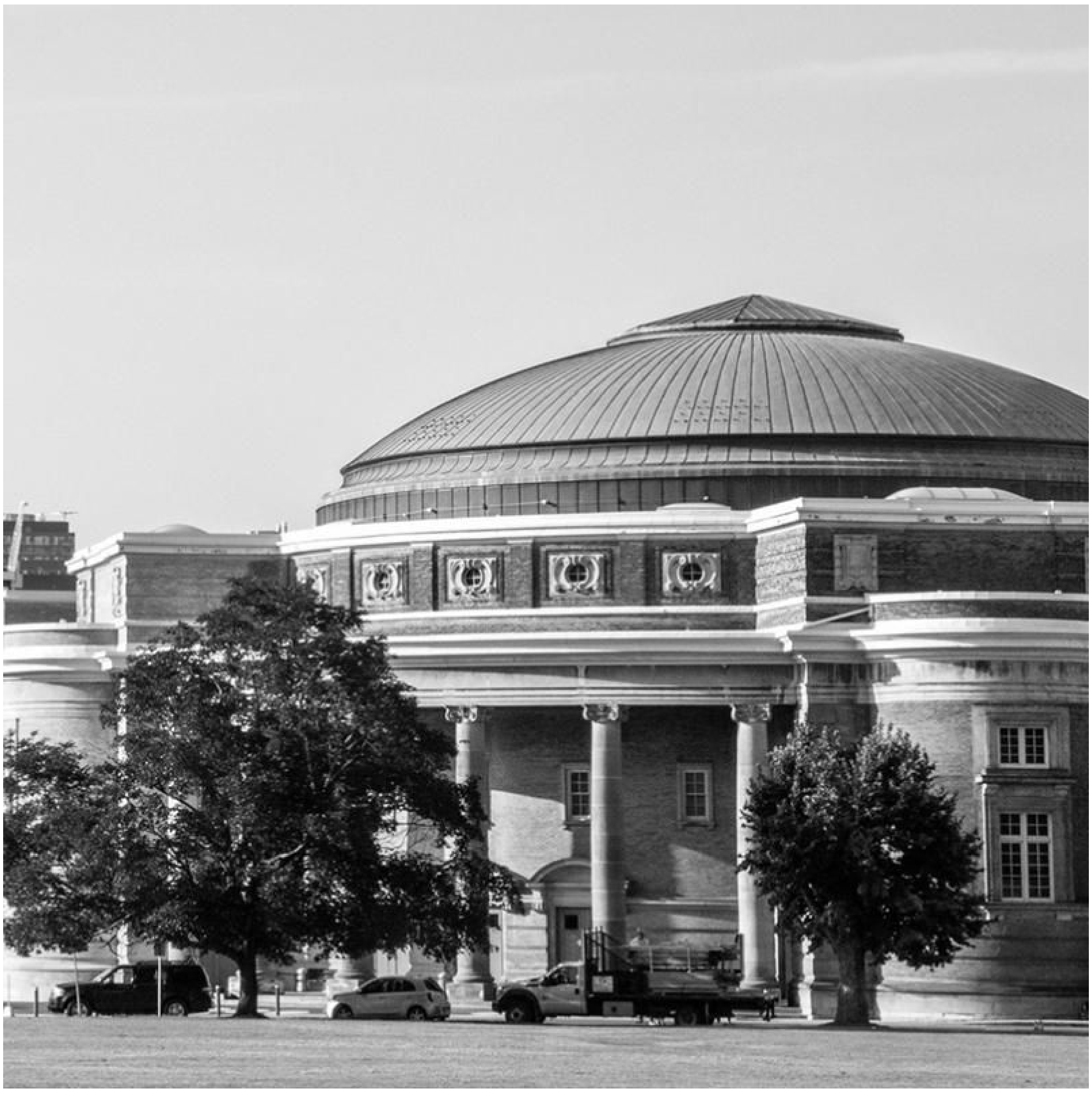
For the following two separable filters, we get two pair of 1d filter

① $$\begin{bmatrix} 16 & 0 & 16 \\ 0 & 0 & 0 \\ 16 & 0 & 16 \end{bmatrix}$$

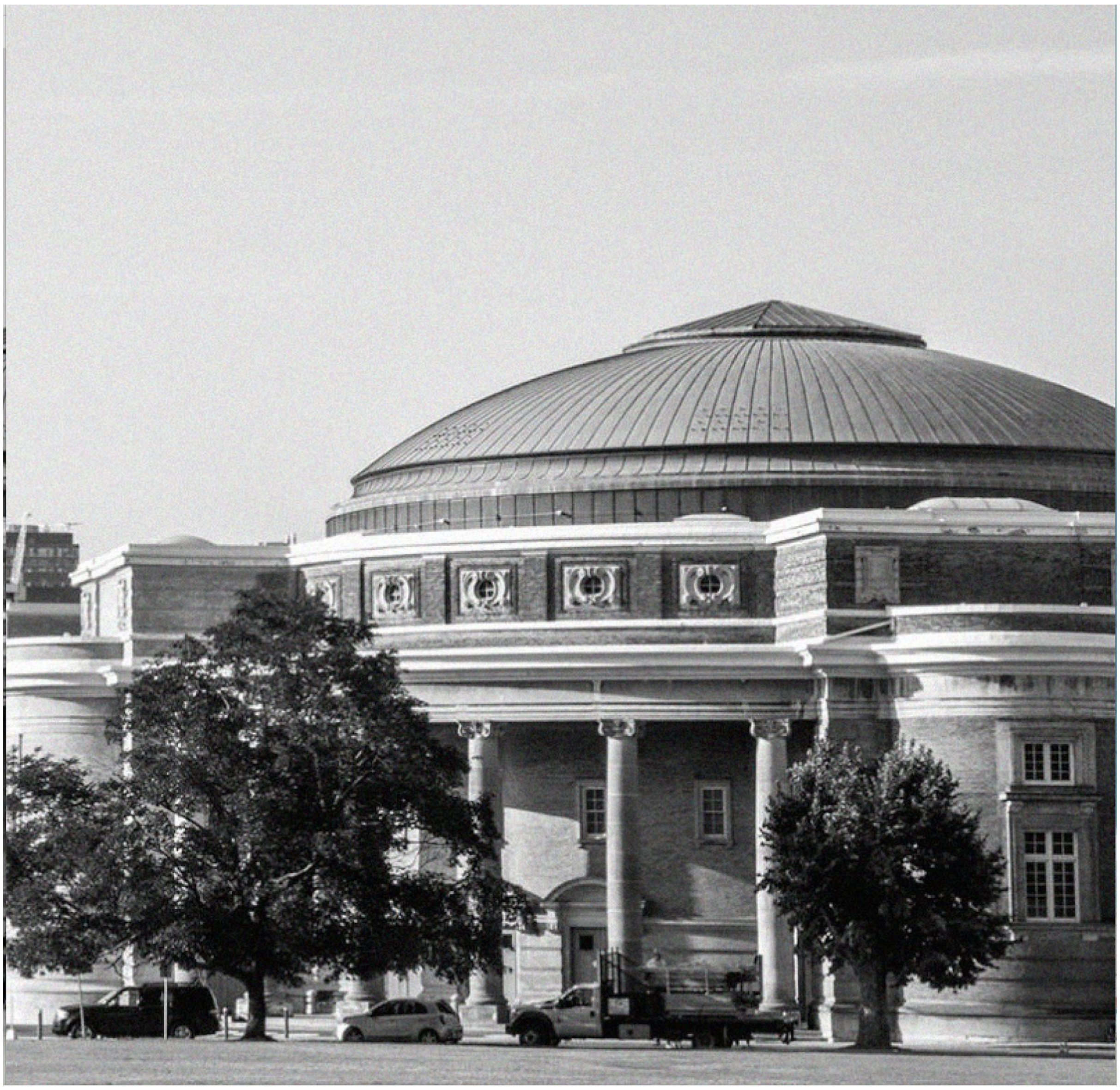
② $$\begin{bmatrix} 25 & 0 & 25 \\ 0 & 0 & 0 \\ 25 & 0 & 25 \end{bmatrix}$$

For No.1, vertical filter [-4, 0, -4], horizontal filter [-4, 0, -4]
For No.2, vertical filter [-5, 0, -5], horizontal filter [-5, 0, -5]

Q6
Original



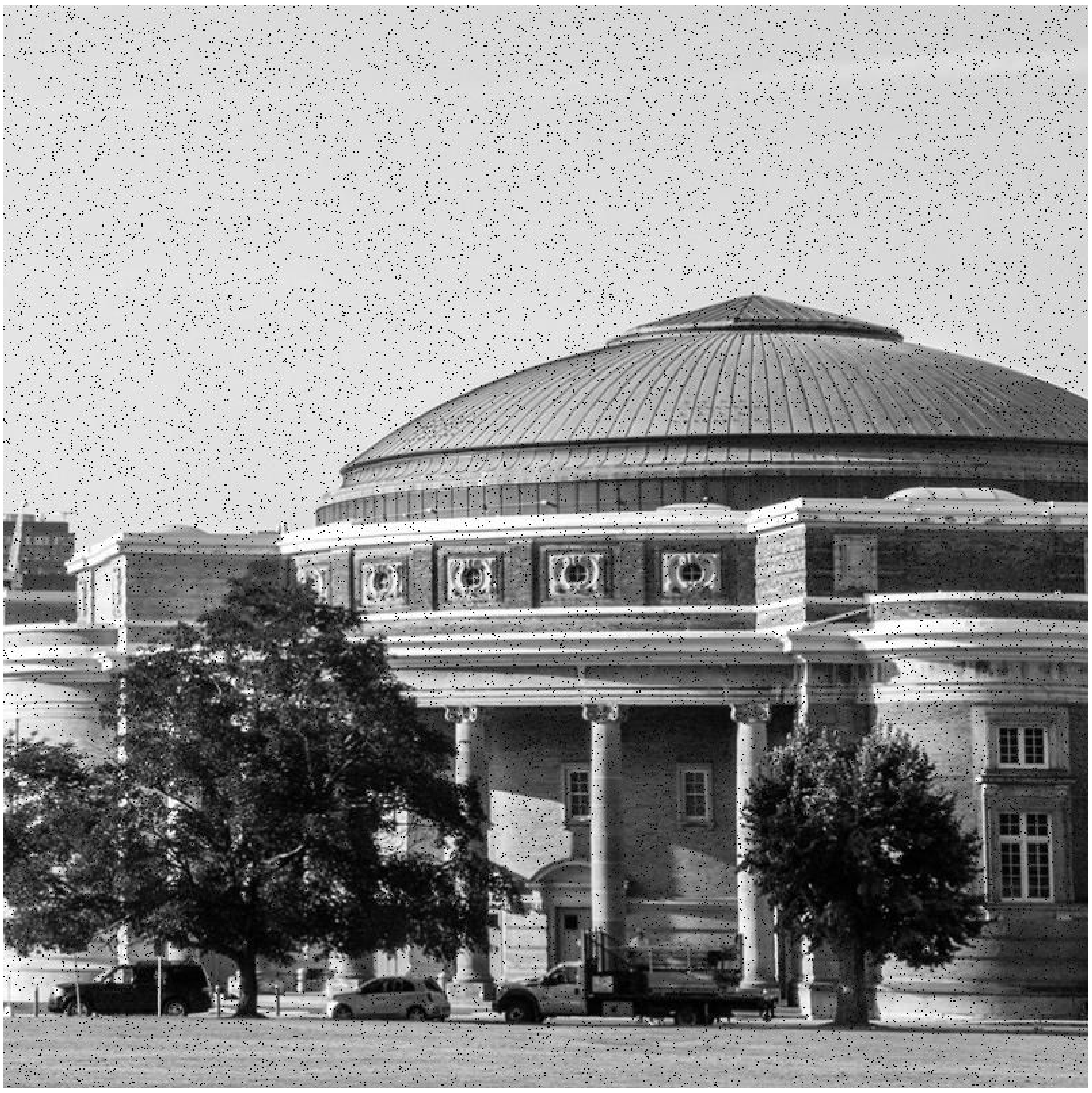
(a)
image with noise



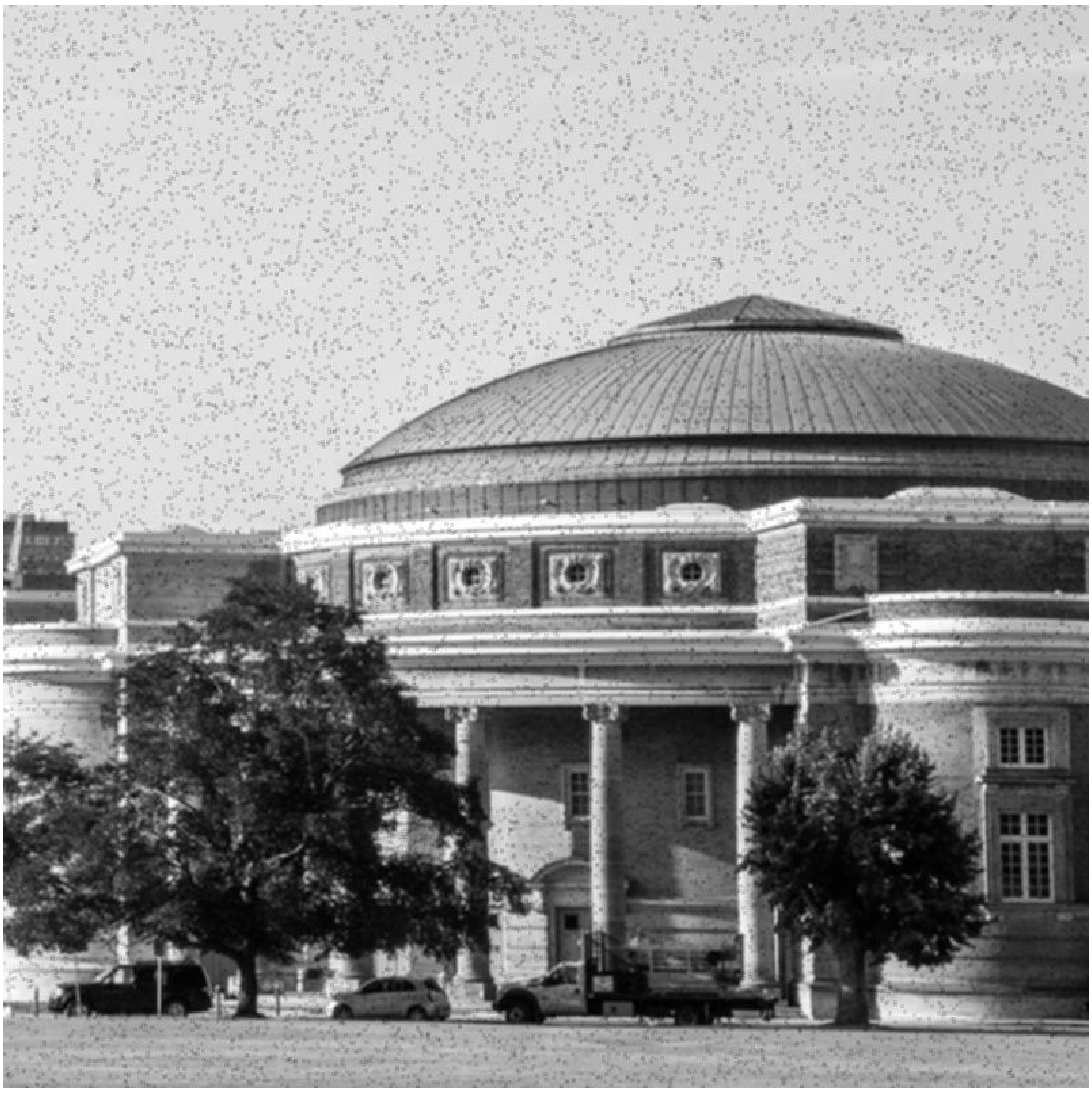
(b)
The linear filter is a **3x3 moving average filter**. The reason is that we believe the neighboring pixels are similar and the noise to be independent from pixel to pixel so that the noise will cancel each other when we apply the moving average filter.

After noise removal:

(c) image with salt-and-pepper noise



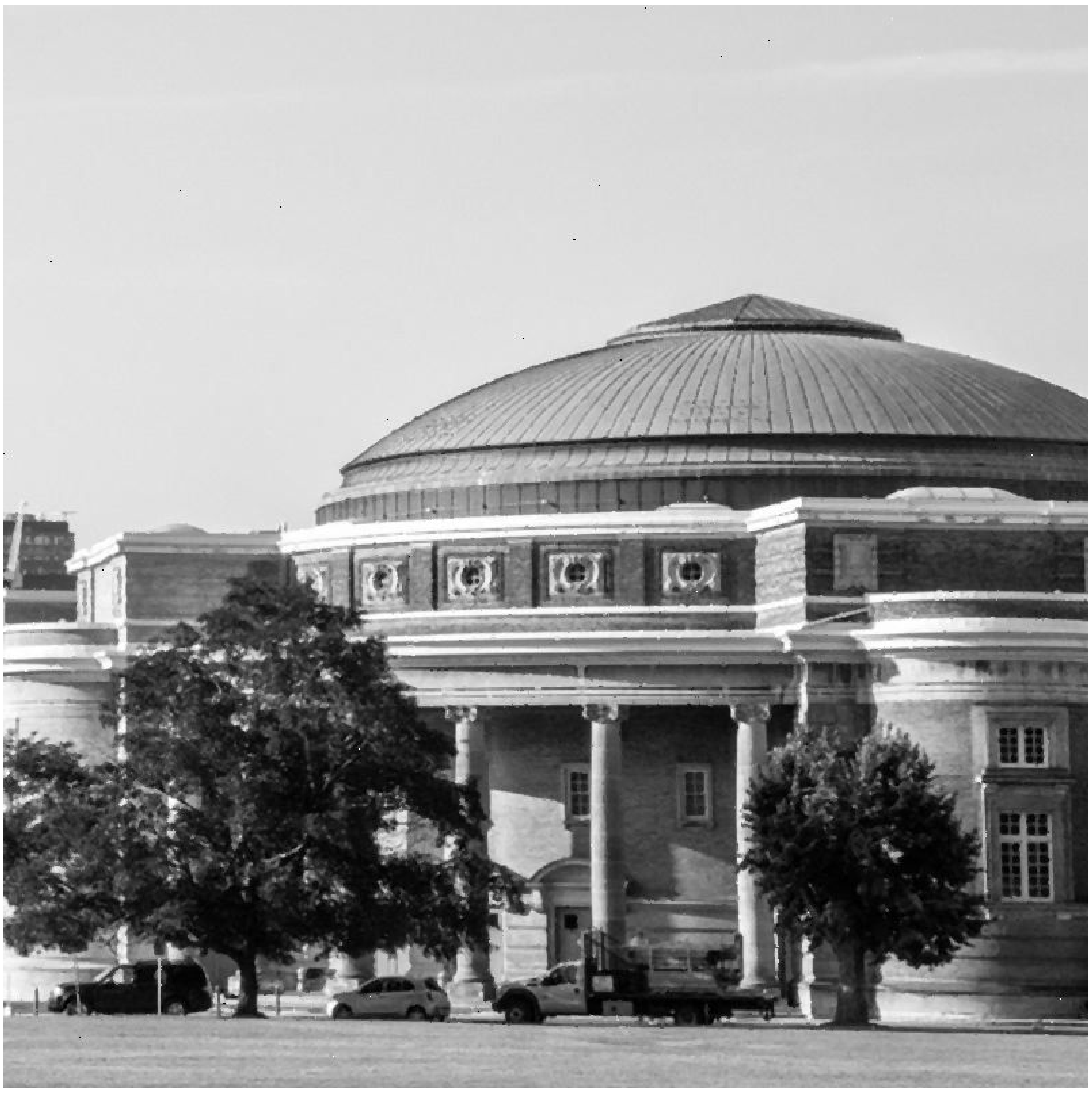
(d)
image after using (b) solution



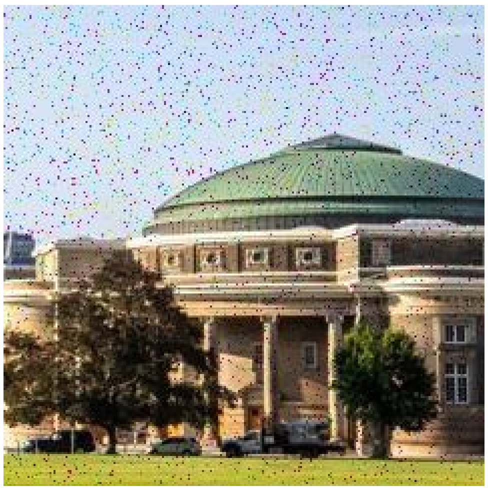
It doesn’t work since the simple moving average method can not deal with the case where new large noise and small noise are randomly distributed in the image.

In this step, I choose the median filter and it works. The reason behind it is that the extremely large or small value is an outlier to its neighbors so that it will work if we replace the extreme value with its neighbors median.

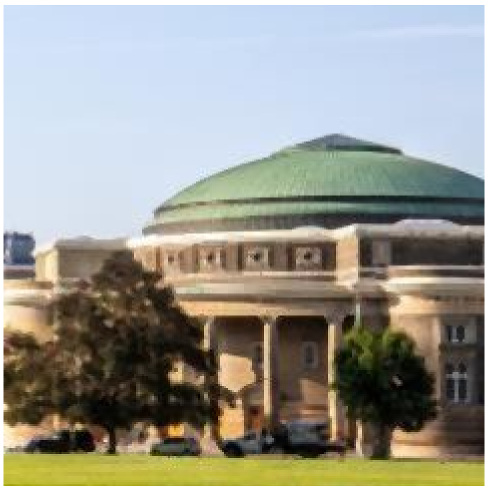
The result is the following:



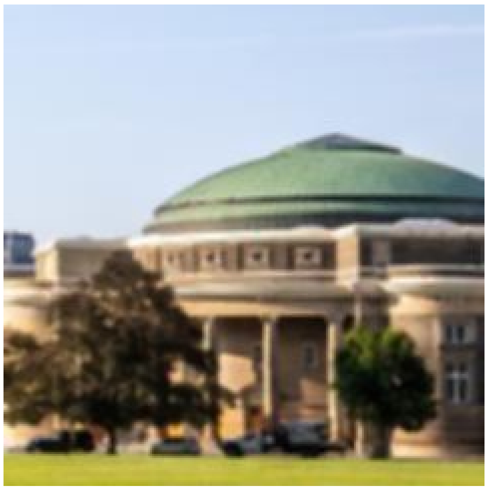
(e)
salt-and-pepper color image:



by applying the method in (d), we get the image with flaws, because if there are a lot of noise values, then a huge value would be selected as the median value, which is a problem that most filters will have to face.



Then I apply 3x3 moving average filter(from b) to the above picture and I get:



The reason why it works is that we “remove” the remaining noise by averaging the pixel value and its neighbors’ value.’