機器視覺 HW4 資工三 109590041 范遠皓

環境

Visual Studio 2019、C++ 、openCV2.4.13.6

image Padding

在 Filter 之前先根據 mask 大小填充圖片使 Filter 後圖片保持原本大小。使用的填充方式為複製邊界像素，將填充的像素設成離原圖片最接近的像素點。三種 Filter 都使用同一種 Padding。

```
24      // padding 填充圖片
25      virtual Mat PadImage(const Mat& image, int paddingSize)
26      {
27          Mat padded = Mat(image.rows + paddingSize * 2, image.cols + paddingSize * 2, image.type());
28
29          // 複製邊界像素值
30          for (int i = 0; i < padded.rows; i++) {
31              for (int j = 0; j < padded.cols; j++) {
32                  int ii = i - paddingSize;
33                  int jj = j - paddingSize;
34
35                  if (ii < 0)
36                      ii = 0;
37                  else if (ii >= image.rows)
38                      ii = image.rows - 1;
39
40                  if (jj < 0)
41                      jj = 0;
42                  else if (jj >= image.cols)
43                      jj = image.cols - 1;
44
45                  padded.at<Vec3b>(i, j) = image.at<Vec3b>(ii, jj);
46              }
47          }
48          return padded;
49      }
50  };
```

Mean Filter

將 mask 內所有像素加起來後再根據 mask 大小進行平均。

```
class MeanFilter : public Filter
{
public:
    MeanFilter() {

    }

    Mat FilterImage(const Mat& sourceImage) override {
        Mat resultImage = Mat(sourceImage.size(), CV_8UC3);
        Mat paddedImage = this->PadImage(sourceImage, this->_mask / 2);
        for (int i = 0; i < resultImage.rows; i++)
            for (int j = 0; j < resultImage.cols; j++)
            {
                // 相加後平均
                int value = 0;
                for (int x = 0; x < this->_mask; x++)
                    for (int y = 0; y < this->_mask; y++)
                        value += paddedImage.at<Vec3b>(i + x, j + y)[0];
                value /= this->_mask * this->_mask;
                resultImage.at<Vec3b>(i, j) = Vec3b(value, value, value);
            }
        return resultImage;
    }
};
```

Median Filter

排列 mask 內所有像素選中間值當作結果。

```
78  class MedianFilter : public Filter
79  {
80  public:
81      MedianFilter() {
82
83      }
84
85      Mat FilterImage(const Mat& sourceImage) override {
86          Mat resultImage = Mat(sourceImage.size(), CV_8UC3);
87          Mat paddedImage = this->PadImage(sourceImage, this->_mask / 2);
88          for (int i = 0; i < resultImage.rows; i++)
89              for (int j = 0; j < resultImage.cols; j++)
90              {
91                  // 取中間值
92                  int value = 0;
93                  vector<int> temp;
94                  for (int x = 0; x < this->_mask; x++)
95                      for (int y = 0; y < this->_mask; y++)
96                          temp.push_back(paddedImage.at<Vec3b>(i + x, j + y)[0]);
97                  std::sort(temp.begin(), temp.end());
98                  value = temp[(this->_mask * this->_mask) / 2];
99                  resultImage.at<Vec3b>(i, j) = Vec3b(value, value, value);
100             }
101         return resultImage;
102     }
103 };
```

Gaussian Filter

根據公式 $G(x,y)=e^{-(x^2+y^2)}$ 建立 Gaussian Kernel 並正規化。

sigma 值設定根據 mask 的大小計算 。

3x3->0.8

5x5->1.1

7x7->1.4

```
109     // 定義 Kernel 型別
110     typedef vector<vector<double>> Kernel;
```

```
133 private:
134     // 創建 Gaussian Kernel
135     Kernel CreateGaussianKernel(int kernelSize, double sigma = -1)
136     {
137         // 未給出 sigma 自動計算
138         if (sigma <= 0)
139             sigma = 0.3 * ((kernelSize - 1.0) * 0.5 - 1.0) + 0.8;
140 
141         const int CENTER = kernelSize / 2;
142         Kernel kernel(kernelSize, vector<double>(kernelSize, 0.0));
143 
144         // 計算 Gaussian Kernel 中每個元素的值
145         double sum = 0.0;
146         for (int i = 0; i < kernelSize; i++)
147         {
148             for (int j = 0; j < kernelSize; j++)
149             {
150                 double x = double(i) - double(CENTER);
151                 double y = double(j) - double(CENTER);
152                 kernel[i][j] = exp(-(x * x + y * y) / (2.0 * sigma * sigma));
153                 sum += kernel[i][j];
154             }
155         }
156 
157         // 正規化
158         for (int i = 0; i < kernelSize; i++)
159             for (int j = 0; j < kernelSize; j++)
160                 kernel[i][j] /= sum;
161 
162         return kernel;
163     }
```

將 mask 內所有像素乘上對應 Gaussian Kernel 的係數並相加計算出像素值。

```
106  class GaussianFilter : public Filter
107  {
108  public:
109      // 定義 Kernel 型別
110      typedef vector<vector<double>> Kernel;
111
112      GaussianFilter() {
113
114      }
115
116      Mat FilterImage(const Mat& sourceImage) override {
117          Mat resultImage = Mat(sourceImage.size(), CV_8UC3);
118          Mat paddedImage = this->PadImage(sourceImage, this->_mask / 2);
119          Kernel kernel = CreateGaussianKernel(this->_mask);
120          for (int i = 0; i < resultImage.rows; i++)
121              for (int j = 0; j < resultImage.cols; j++)
122              {
123                  // Gaussian
124                  double value = 0;
125                  for (int x = 0; x < this->_mask; x++)
126                      for (int y = 0; y < this->_mask; y++)
127                          value += paddedImage.at<Vec3b>(i + x, j + y)[0] * kernel[x][y];
128                  resultImage.at<Vec3b>(i, j) = Vec3b(value, value, value);
129              }
130          return resultImage;
131      }
```

結果圖片

House256_noise

Mean 3x3

Mean 7x7

Median 3x3

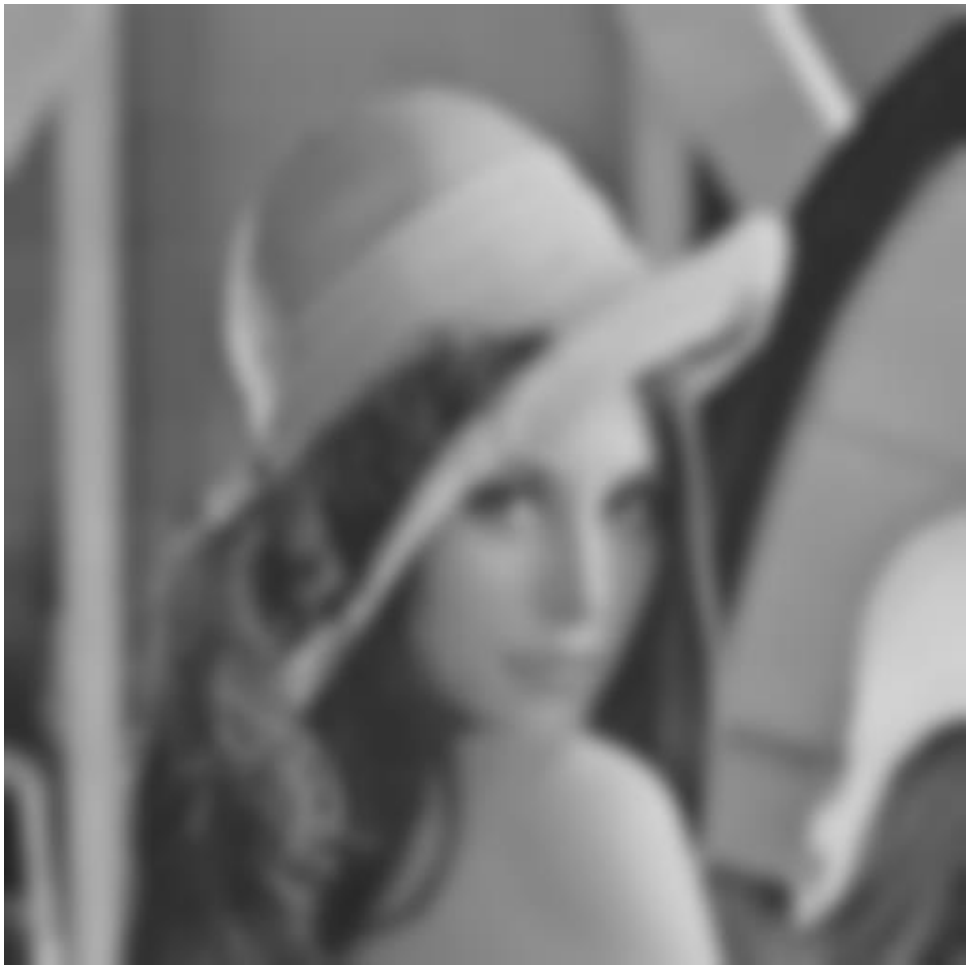
Median 7x7

Gaussian 5x5

Lena_gray

Mean 3x3

Mean 7x7



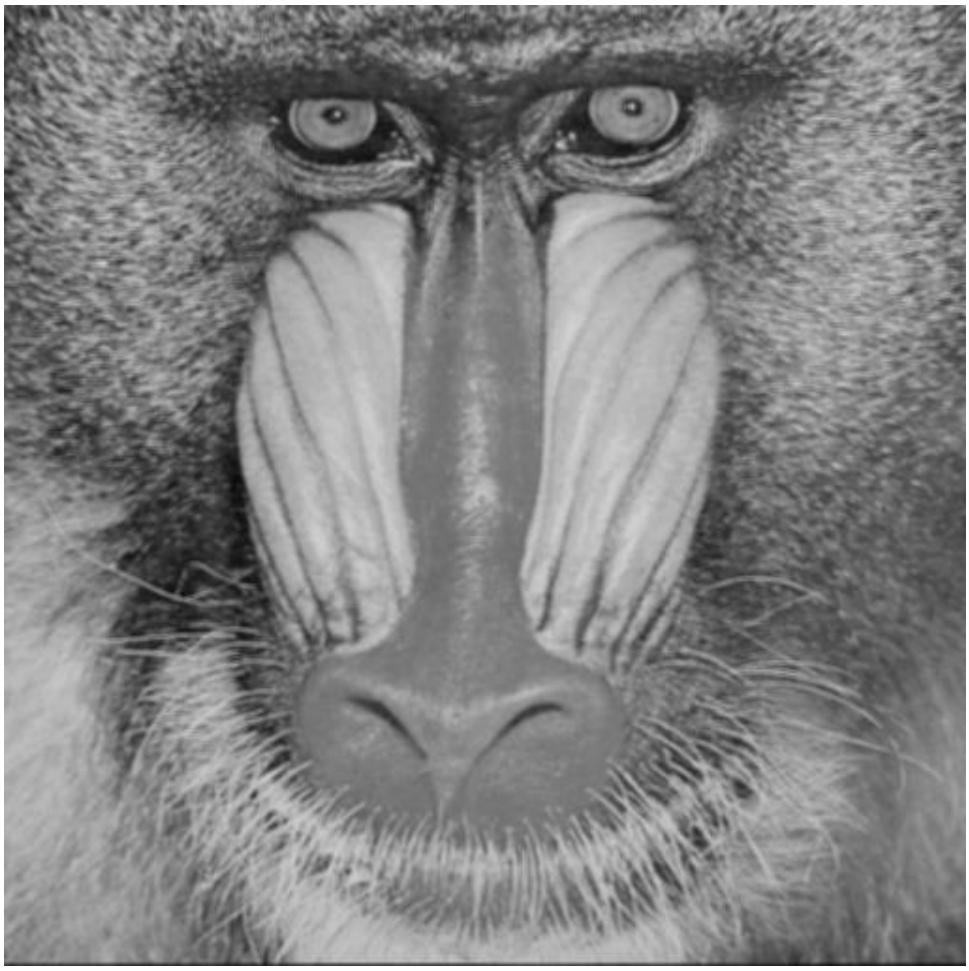
Median 3x3

Median 7x7

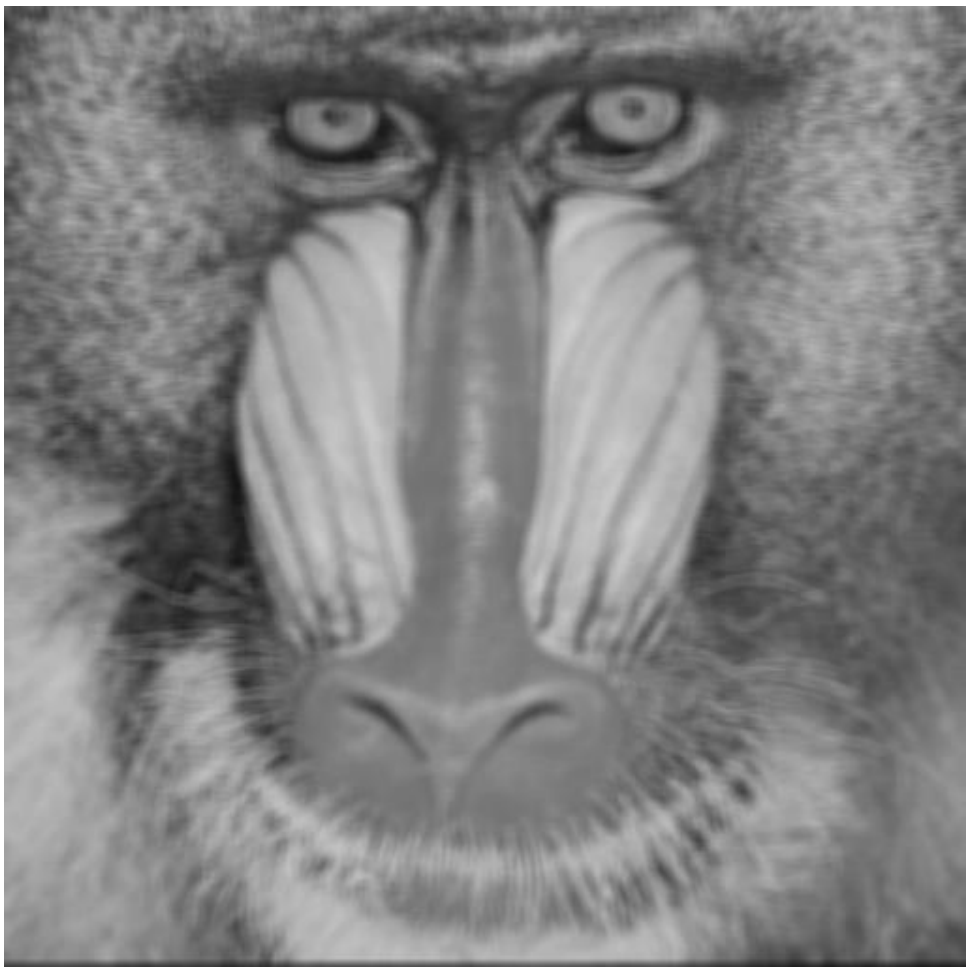
Gaussian 5x5

Mandrill_gray

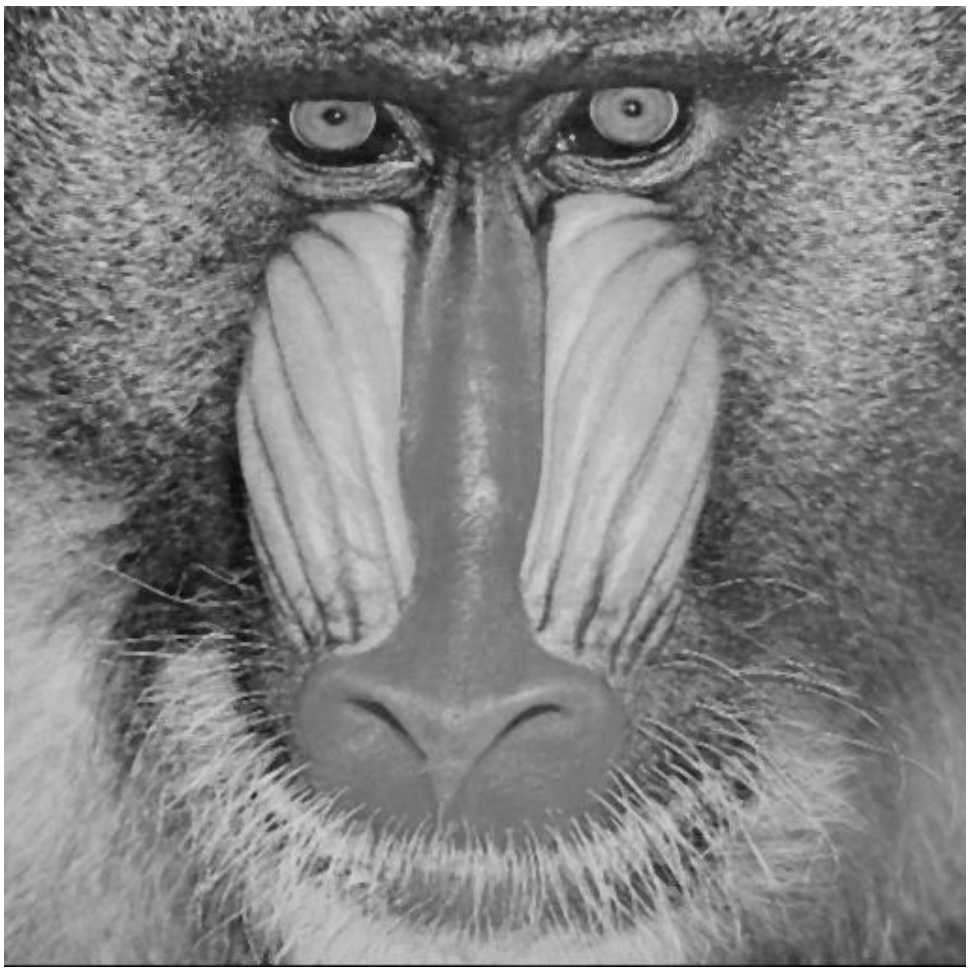
Mean 3x3

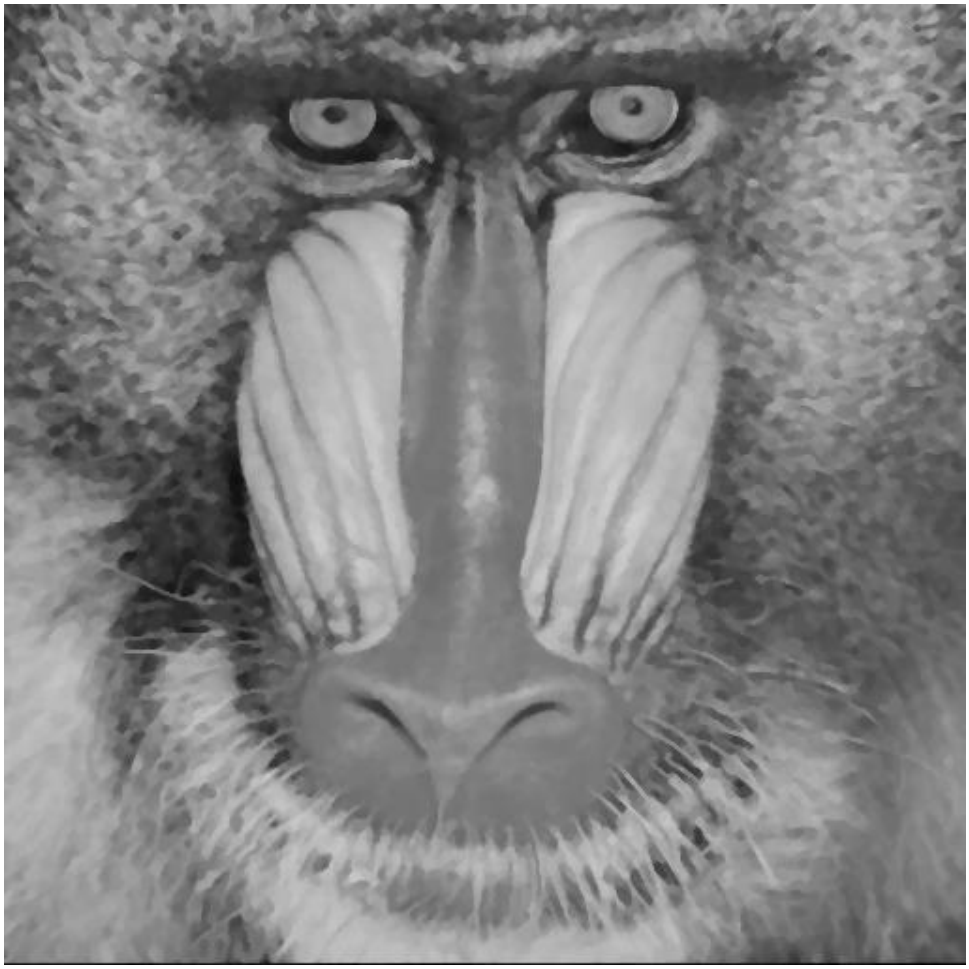


Mean 7x7

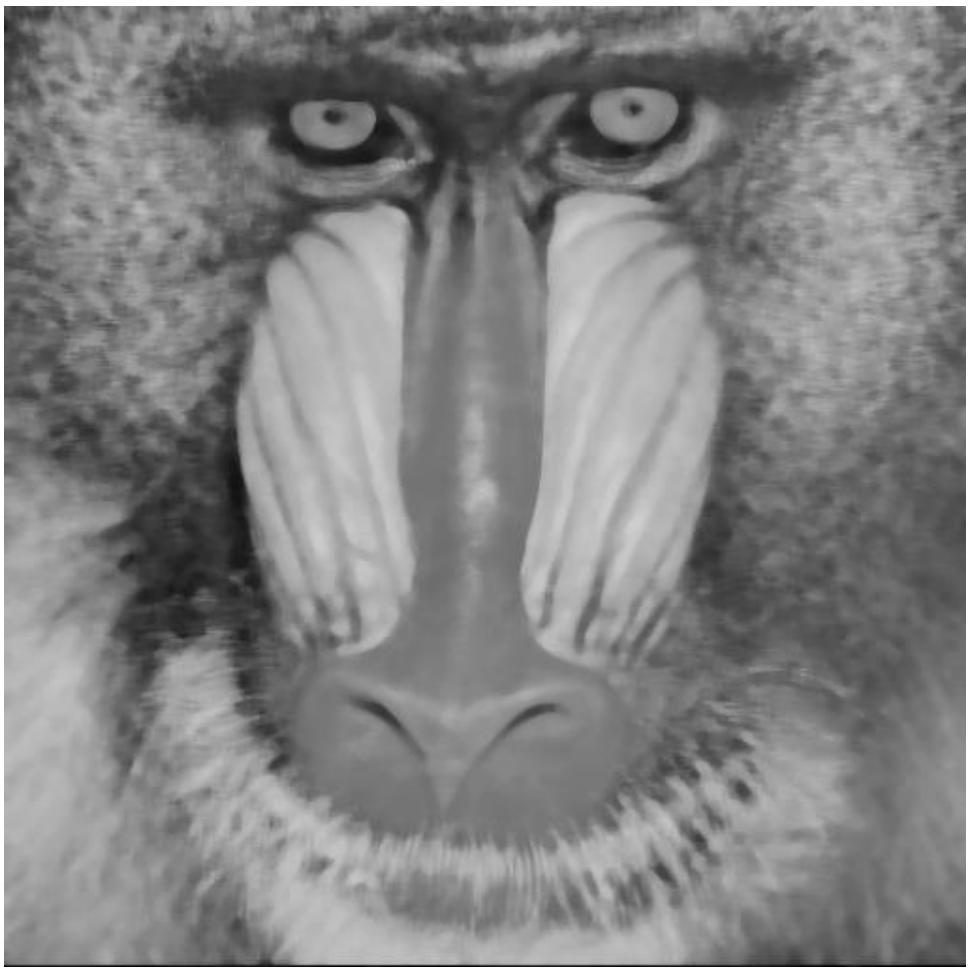


Median 3x3

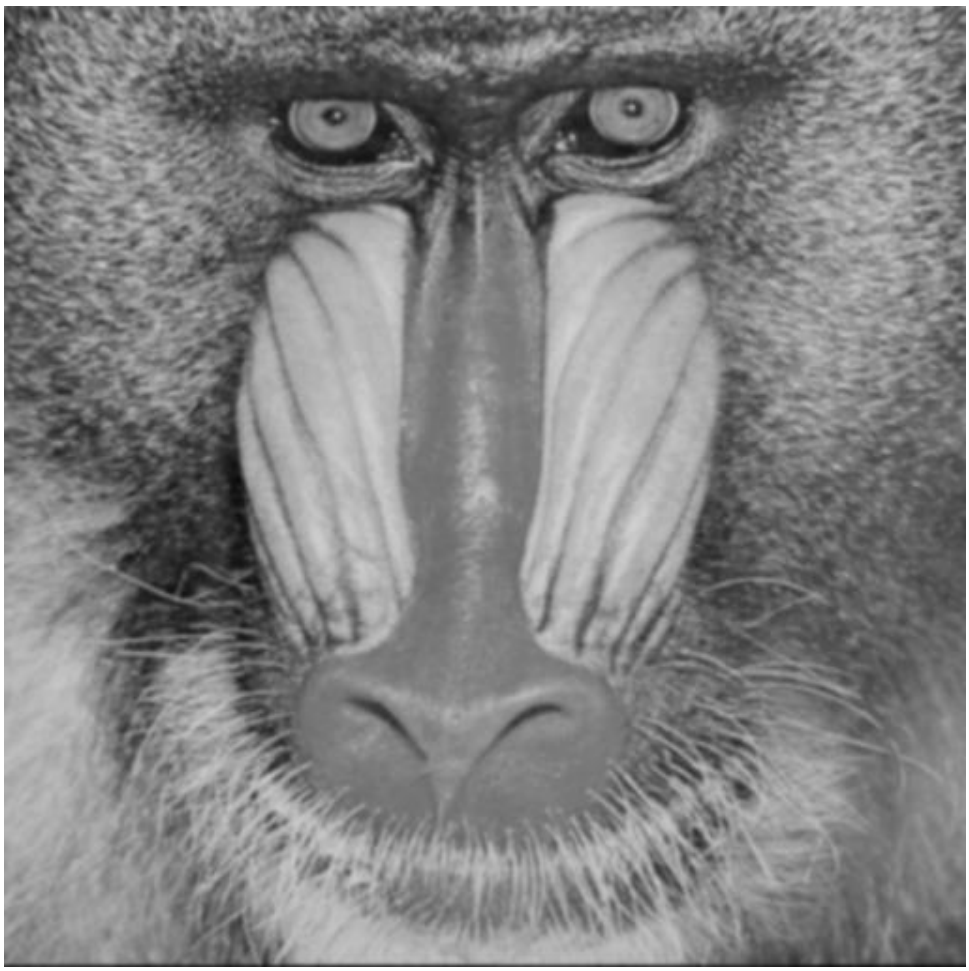




Median 7x7

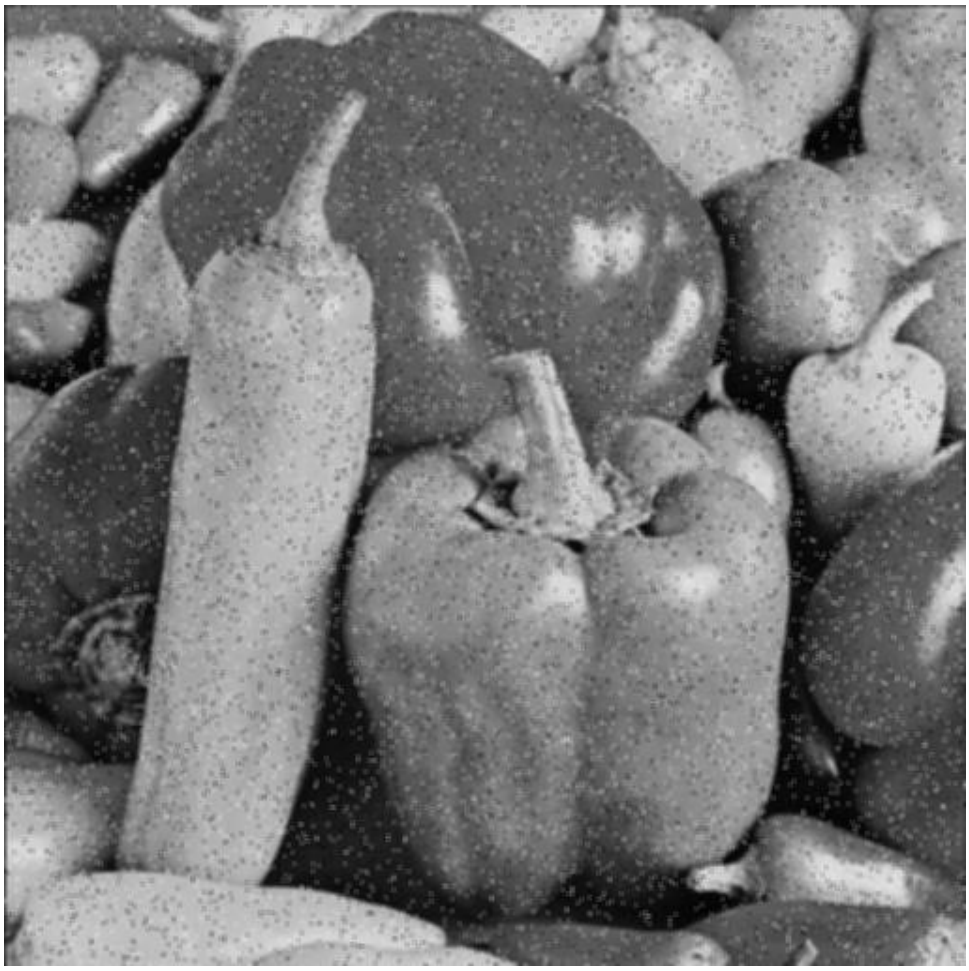


Gaussian 5x5



Peppers_noise

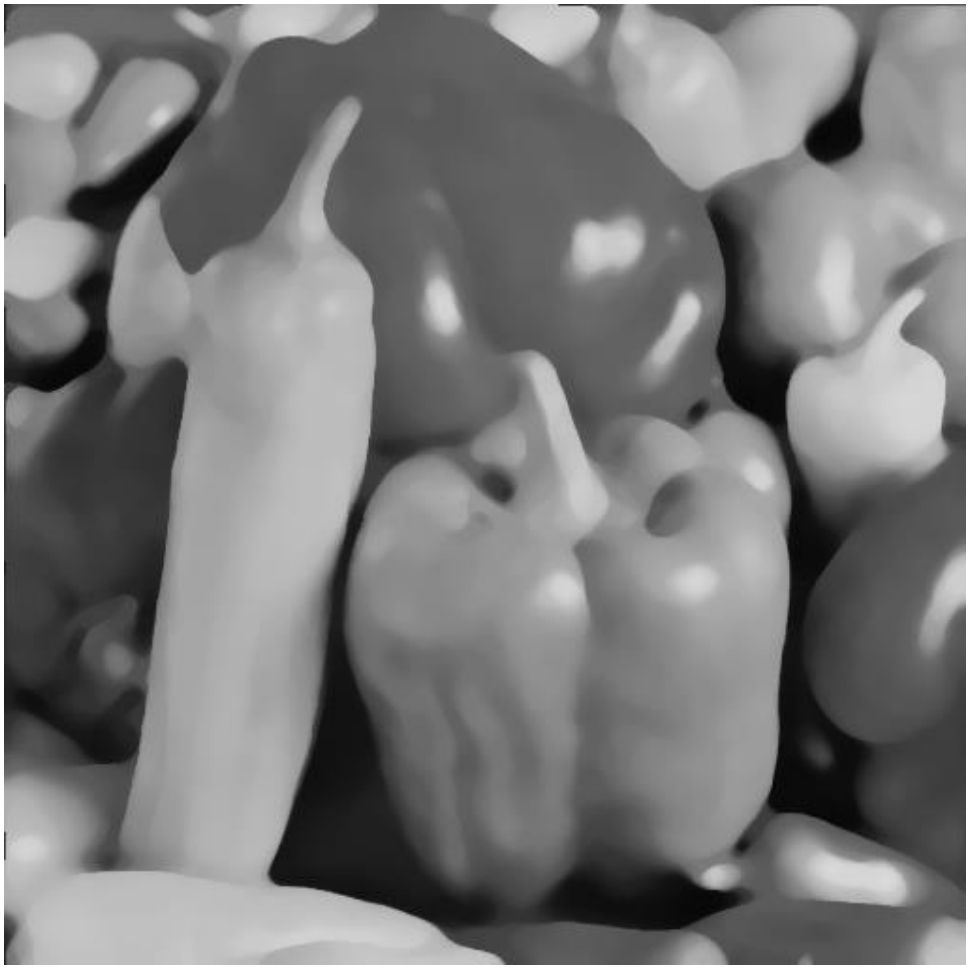
Mean 3x3



Mean 7x7

Median 3x3

Median 7x7



Gaussian 5x5