Computer vision Homework8

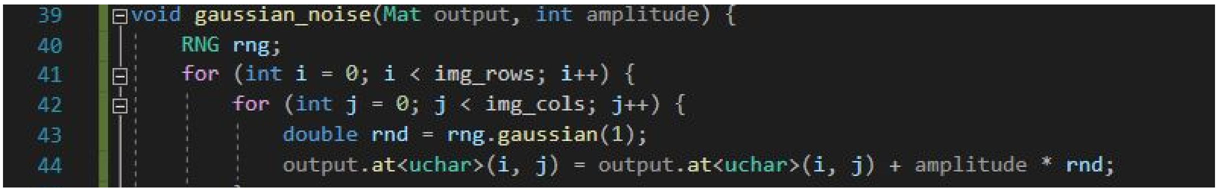
B06902091 資工四 羅寶瑩

Noise Removal

Description

(a) Generate noisy images with gaussian noise(amplitude of 10 and 30)

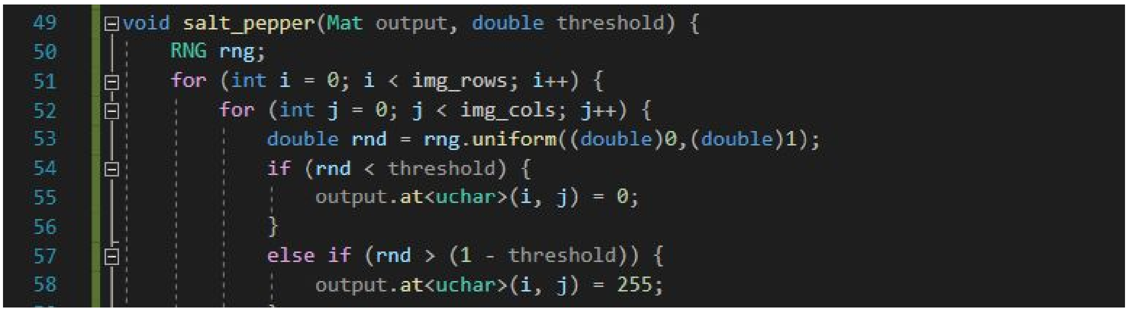
Refer to the function from the lecture slide.



```
void gaussian_noise(Mat output, int amplitude) {
    RNG rng;
    for (int i = 0; i < img_rows; i++) {
        for (int j = 0; j < img_cols; j++) {
            double rnd = rng.gaussian(1);
            output.at<uchar>(i, j) = output.at<uchar>(i, j) + amplitude * rnd;
```

(b) Generate noisy images with salt-and-pepper noise(probability 0.1 and 0.05)

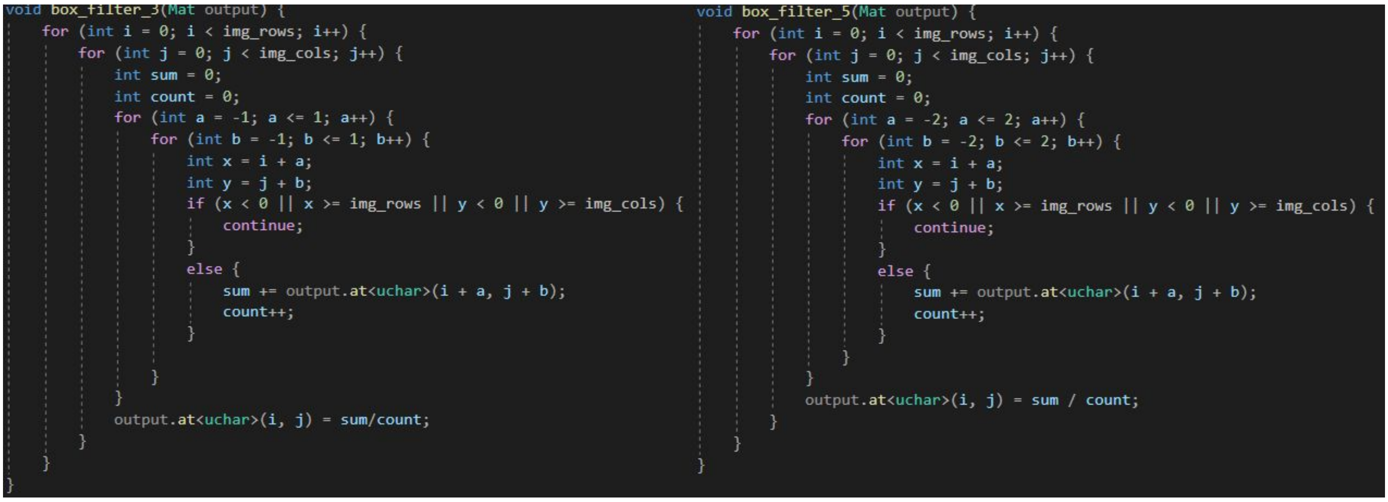
Refer to the function from the lecture slide.



```
void salt_pepper(Mat output, double threshold) {
    RNG rng;
    for (int i = 0; i < img_rows; i++) {
        for (int j = 0; j < img_cols; j++) {
            double rnd = rng.uniform((double)0,(double)1);
            if (rnd < threshold) {
                output.at<uchar>(i, j) = 0;
            }
            else if (rnd > (1 - threshold)) {
                output.at<uchar>(i, j) = 255;
```

(c) Use the 3x3, 5x5 box filter on images generated by (a)(b)

Calculate the average value of the pixels in a 3x3/5x5 box filter and apply it to the current pixel.

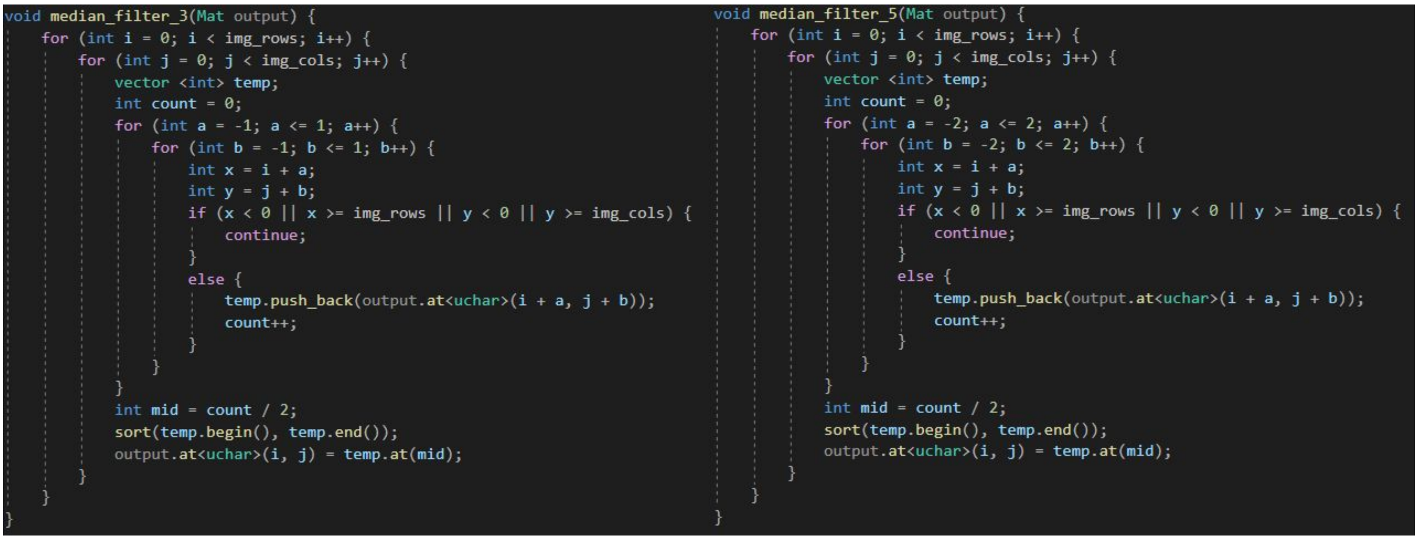


```
void box_filter_3(Mat output) {
    for (int i = 0; i < img_rows; i++) {
        for (int j = 0; j < img_cols; j++) {
            int sum = 0;
            int count = 0;
            for (int a = -1; a <= 1; a++) {
                for (int b = -1; b <= 1; b++) {
                    int x = i + a;
                    int y = j + b;
                    if (x < 0 || x >= img_rows || y < 0 || y >= img_cols) {
                        continue;
                    }
                    else {
                        sum += output.at<uchar>(i + a, j + b);
                        count++;
                    }

                }
            }
            output.at<uchar>(i, j) = sum/count;
        }
    }
}
```

```
void box_filter_5(Mat output) {
    for (int i = 0; i < img_rows; i++) {
        for (int j = 0; j < img_cols; j++) {
            int sum = 0;
            int count = 0;
            for (int a = -2; a <= 2; a++) {
                for (int b = -2; b <= 2; b++) {
                    int x = i + a;
                    int y = j + b;
                    if (x < 0 || x >= img_rows || y < 0 || y >= img_cols) {
                        continue;
                    }
                    else {
                        sum += output.at<uchar>(i + a, j + b);
                        count++;
                    }
                }
            }
            output.at<uchar>(i, j) = sum / count;
        }
    }
}
```

(d) Use 3x3, 5x5 median filter on images generated by (a)(b)

Record the value of every pixel in a 3x3/5x5 box filter and sort it to find the median to be the new value of the current pixel.



```
void median_filter_3(Mat output) {
    for (int i = 0; i < img_rows; i++) {
        for (int j = 0; j < img_cols; j++) {
            vector <int> temp;
            int count = 0;
            for (int a = -1; a <= 1; a++) {
                for (int b = -1; b <= 1; b++) {
                    int x = i + a;
                    int y = j + b;
                    if (x < 0 || x >= img_rows || y < 0 || y >= img_cols) {
                        continue;
                    }
                    else {
                        temp.push_back(output.at<uchar>(i + a, j + b));
                        count++;
                    }
                }
            }
            int mid = count / 2;
            sort(temp.begin(), temp.end());
            output.at<uchar>(i, j) = temp.at(mid);
        }
    }
}
```

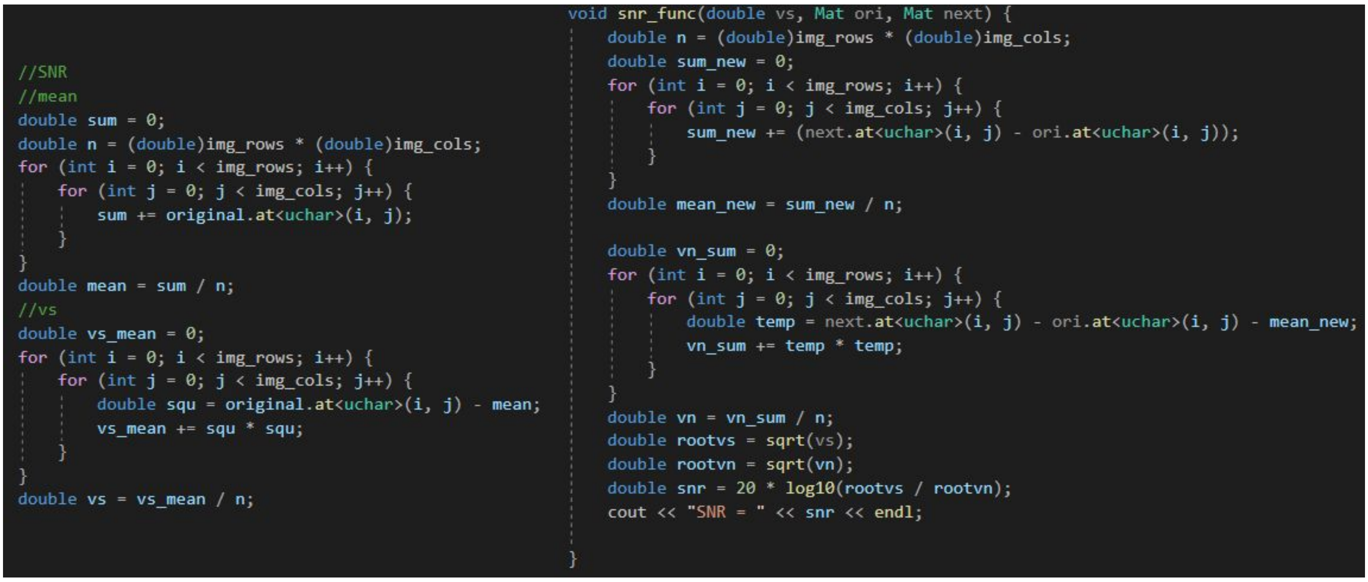
```
void median_filter_5(Mat output) {
    for (int i = 0; i < img_rows; i++) {
        for (int j = 0; j < img_cols; j++) {
            vector <int> temp;
            int count = 0;
            for (int a = -2; a <= 2; a++) {
                for (int b = -2; b <= 2; b++) {
                    int x = i + a;
                    int y = j + b;
                    if (x < 0 || x >= img_rows || y < 0 || y >= img_cols) {
                        continue;
                    }
                    else {
                        temp.push_back(output.at<uchar>(i + a, j + b));
                        count++;
                    }
                }
            }
            int mid = count / 2;
            sort(temp.begin(), temp.end());
            output.at<uchar>(i, j) = temp.at(mid);
        }
    }
}
```

(e) Use both opening-then-closing and closing-then opening filter (using the octogonal 3-5-5-5-3 kernel, value = 0) on images generated by (a)(b)

Use the dilation and erosion function implemented in homework 5 to archive this task.
(Please refer to the source code **[161~205, 308~337]**)

(f) Calculate the signal-to-ratio (SNR) for each instance(4 noisy images and 24 processed images)

Refer to the function from the course website.



```
//SNR
//mean
double sum = 0;
double n = (double)img_rows * (double)img_cols;
for (int i = 0; i < img_rows; i++) {
    for (int j = 0; j < img_cols; j++) {
        sum += original.at<uchar>(i, j);
    }
}
double mean = sum / n;
//vs
double vs_mean = 0;
for (int i = 0; i < img_rows; i++) {
    for (int j = 0; j < img_cols; j++) {
        double squ = original.at<uchar>(i, j) - mean;
        vs_mean += squ * squ;
    }
}
double vs = vs_mean / n;
```

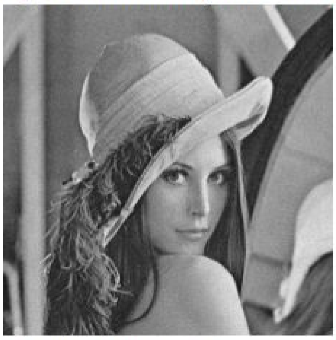
```
void snr_func(double vs, Mat ori, Mat next) {
    double n = (double)img_rows * (double)img_cols;
    double sum_new = 0;
    for (int i = 0; i < img_rows; i++) {
        for (int j = 0; j < img_cols; j++) {
            sum_new += (next.at<uchar>(i, j) - ori.at<uchar>(i, j));
        }
    }
    double mean_new = sum_new / n;

    double vn_sum = 0;
    for (int i = 0; i < img_rows; i++) {
        for (int j = 0; j < img_cols; j++) {
            double temp = next.at<uchar>(i, j) - ori.at<uchar>(i, j) - mean_new;
            vn_sum += temp * temp;
        }
    }
    double vn = vn_sum / n;
    double rootvs = sqrt(vs);
    double rootvn = sqrt(vn);
    double snr = 20 * log10(rootvs / rootvn);
    cout << "SNR = " << snr << endl;
}
```

Reference:

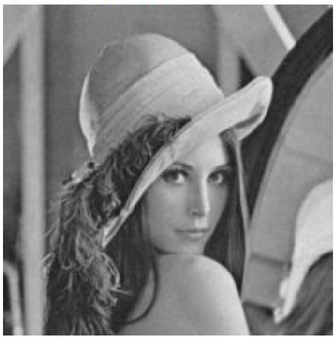
1. *RNG Usage - https://blog.csdn.net/qq_33485434/article/details/78980587*
2. *lecture slide p.126 - p.138*
3. *SNR function - http://cv2.csie.ntu.edu.tw/CV/_material/snr.pdf*

Result (For clearer image, please refer to the image output of the program)

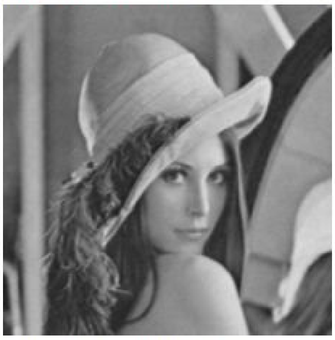
1. gaussian noise, amplitude = 10 SNR = 13.6018



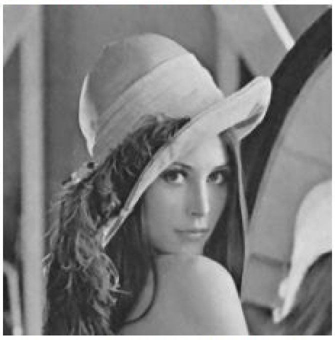
Box_3x3, SNR = 17.2905



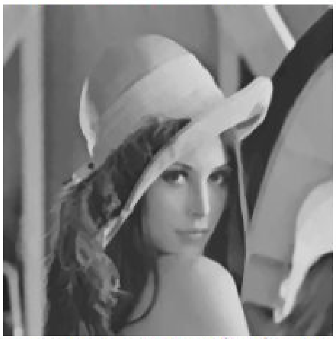
Box_5x5, SNR = 14.1521



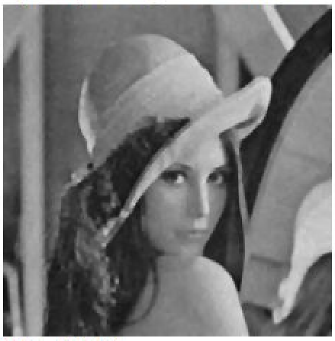
Median_3x3, SNR = 17.0723



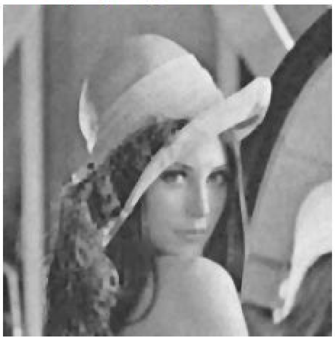
Median_5x5, SNR = 13.5204



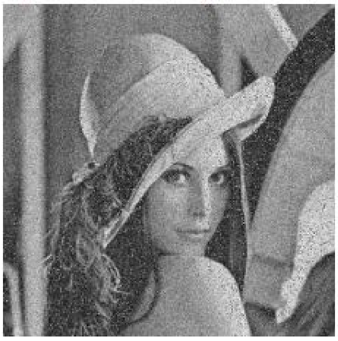
Opening-closing, SNR = 12.6453



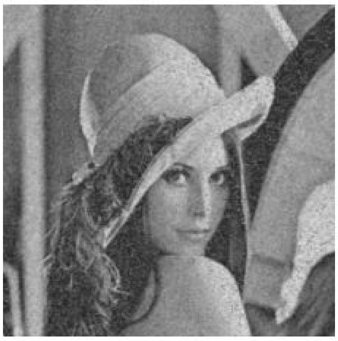
Closing-opening, SNR = 12.9733



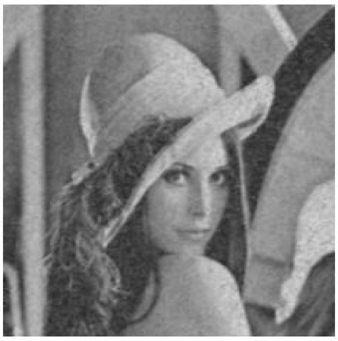
2. gaussian noise, amplitude =30, SNR = 2.19428



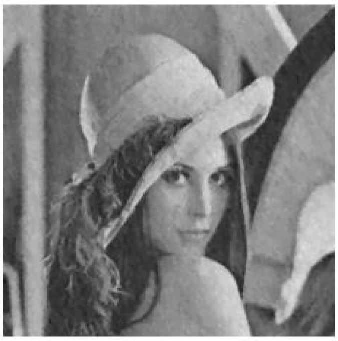
Box_3x3, SNR = 10.0534



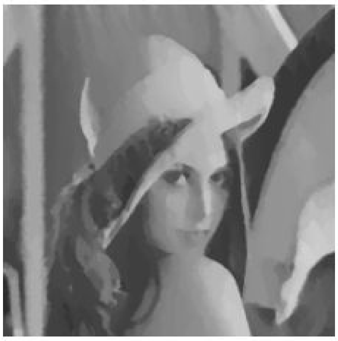
Box_5x5, SNR = 10.6605



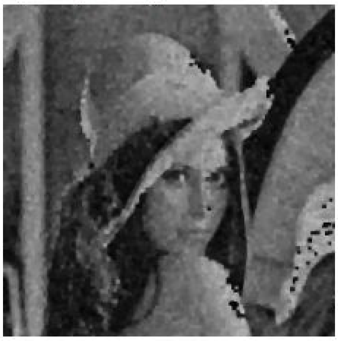
Median_3x3, SNR = 12.4197



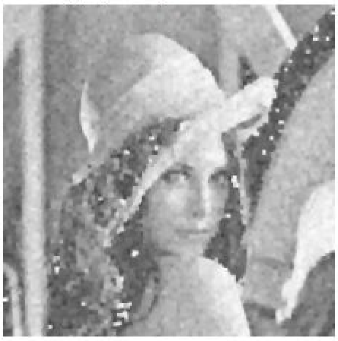
Median_5x5, SNR = 9.5826



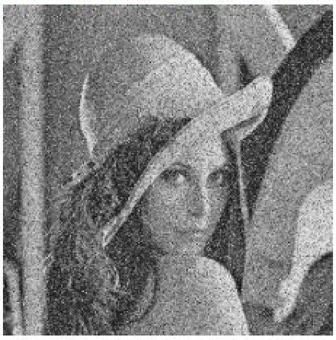
Opening-closing, SNR = 7.12942



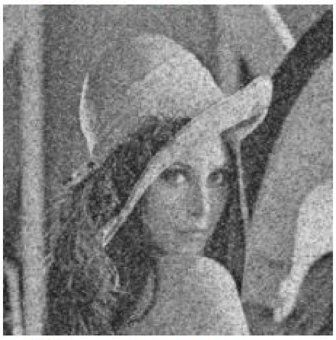
Closing-opening, SNR = 6.71576



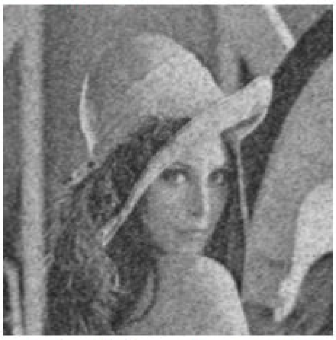
3. salt-and-pepper noise, probability =0.1, SNR = -2.10496



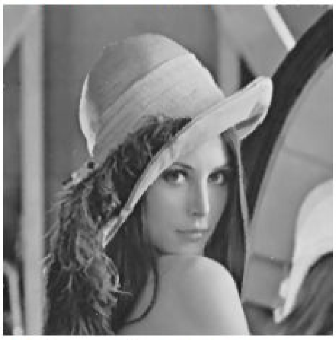
Box_3x3, SNR = 6.65619



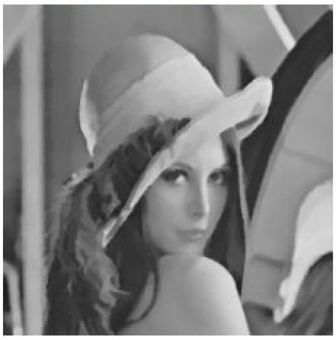
Box_5x5, SNR = 8.5079



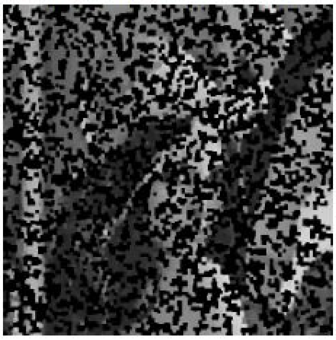
Median_3x3, SNR = 16.5337



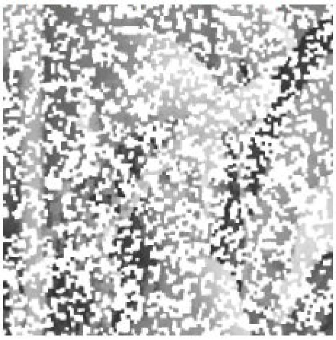
Median_5x5, SNR = 12.9768



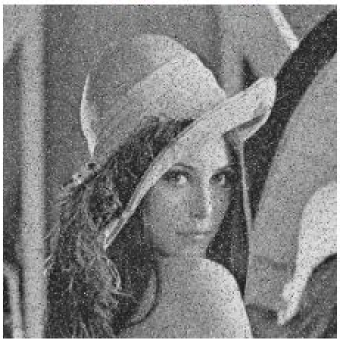
Opening-closing, SNR = -2.84523



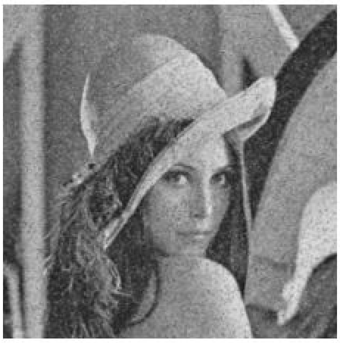
Closing-opening, SNR = -3.13575



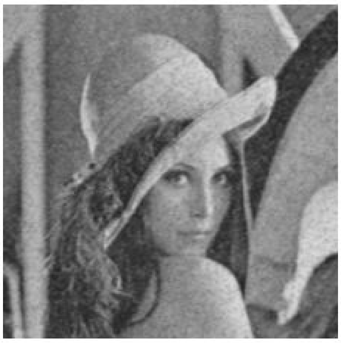
4. salt-and-pepper noise, probability =0.05, SNR = 0.871399



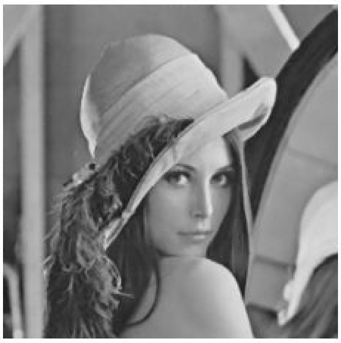
Box_3x3, SNR = 9.64565



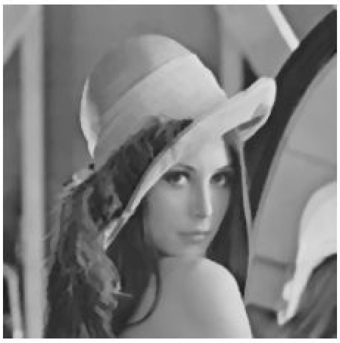
Box_5x5, SNR = 10.8934



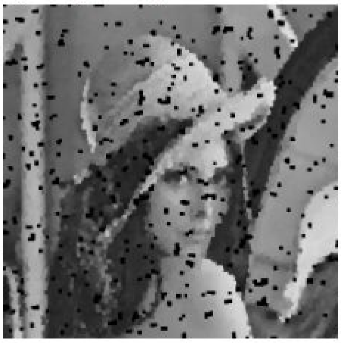
Median_3x3, SNR = 18.0151



Median_5x5, SNR = 13.9951



Opening-closing, SNR = 2.97874



Closing-opening, SNR = 2.44513