Homework requirement:

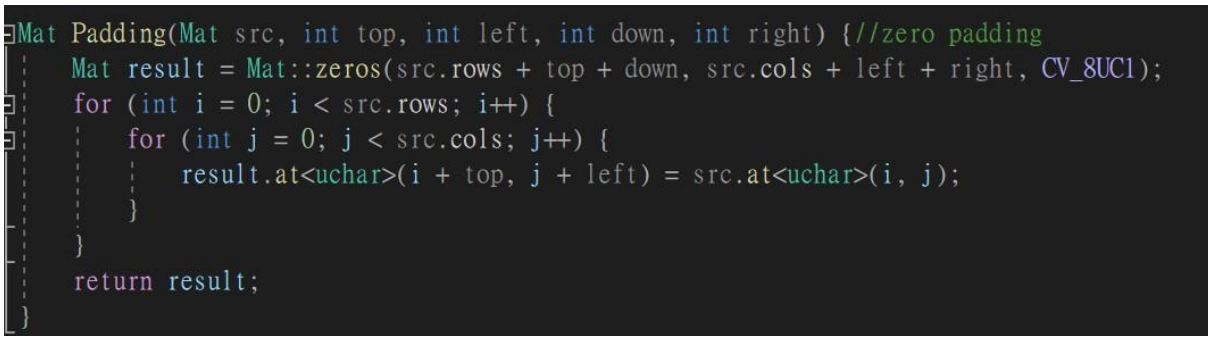
1. Output image with mean filter
2. Output image with median filter
3. Output 3 image histogram
4. Compare 3 histogram and explain the difference

Development environment:

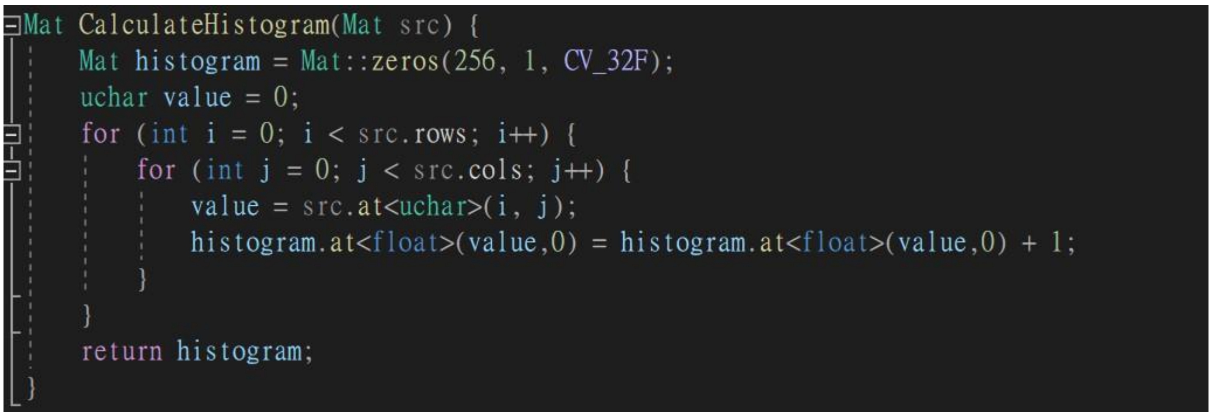
Windows 10
Visual Studio 2022 community
OpenCV 4.6.0

Implementation:

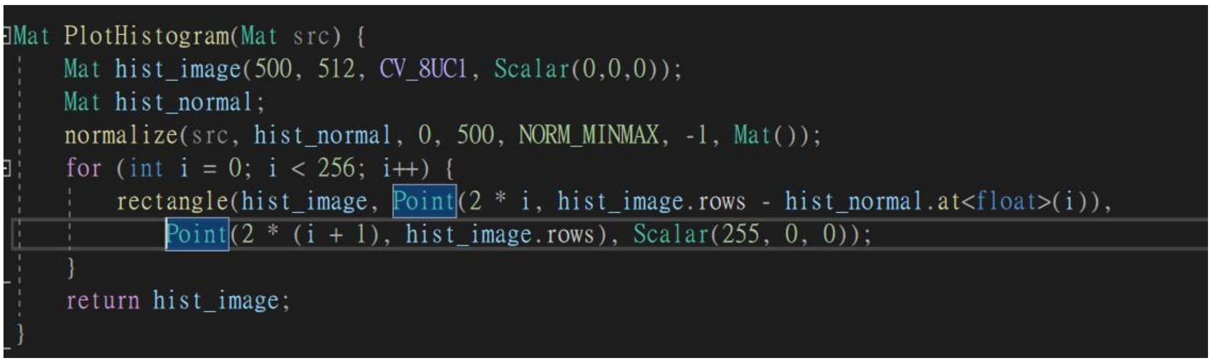
1. Load image and use CalculateHistogram() count the number of pixel for each value(0~255). Use these numbers construct a graph x axis is intensity and y axis is the number of pixels.
2. Pad the source image with zero padding and use function MeanFilter() to process the padded image with a 3x3 kernel. Output a histogram of the processed image.
3. Pad the source image and use function MedianFilter() to process the padded image with a 3x3 kernel. Output a histogram of the processed image.



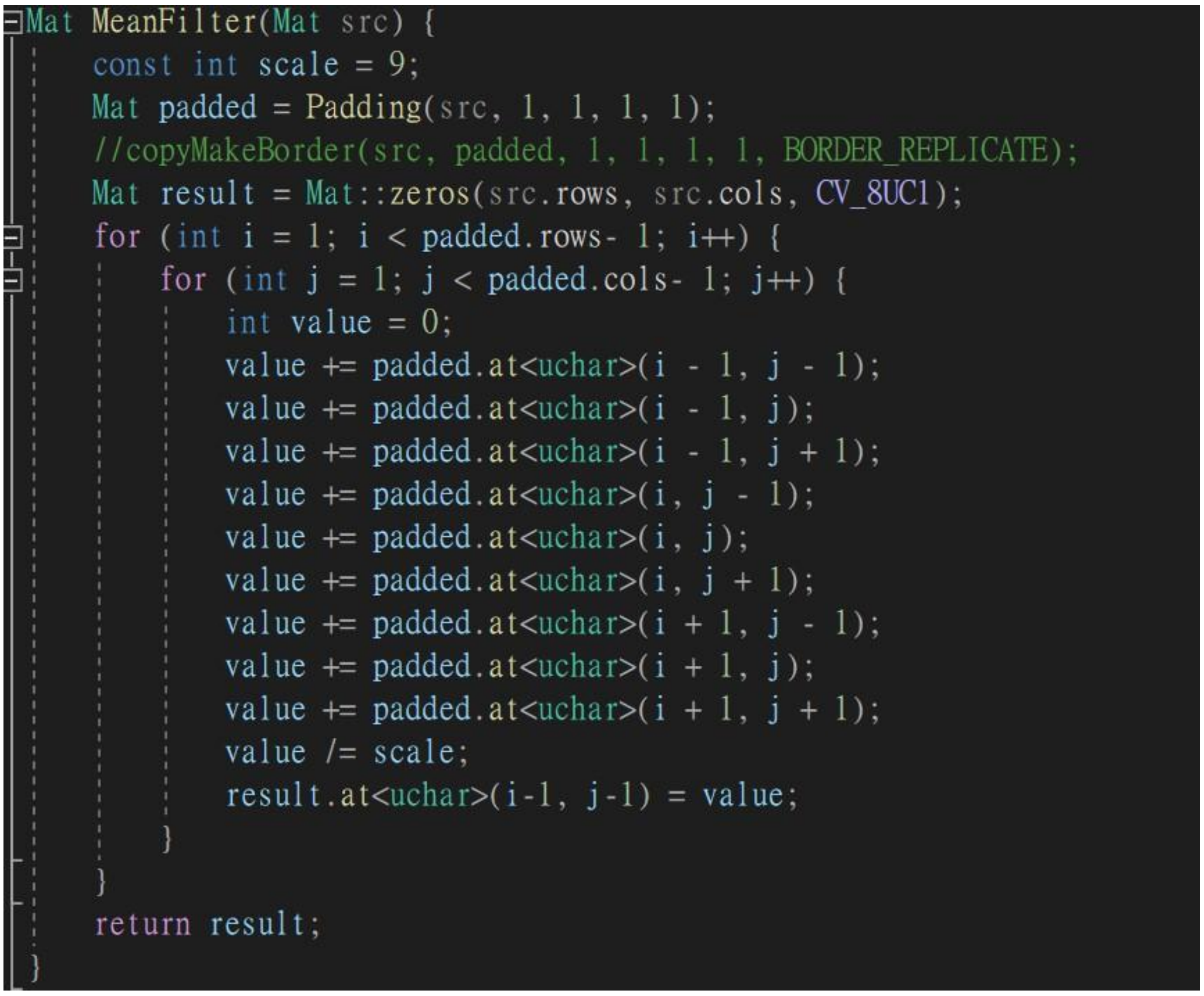
```
Mat Padding(Mat src, int top, int left, int down, int right) {//zero padding
    Mat result = Mat::zeros(src.rows + top + down, src.cols + left + right, CV_8UC1);
    for (int i = 0; i < src.rows; i++) {
        for (int j = 0; j < src.cols; j++) {
            result.at<uchar>(i + top, j + left) = src.at<uchar>(i, j);
        }
    }
    return result;
}
```



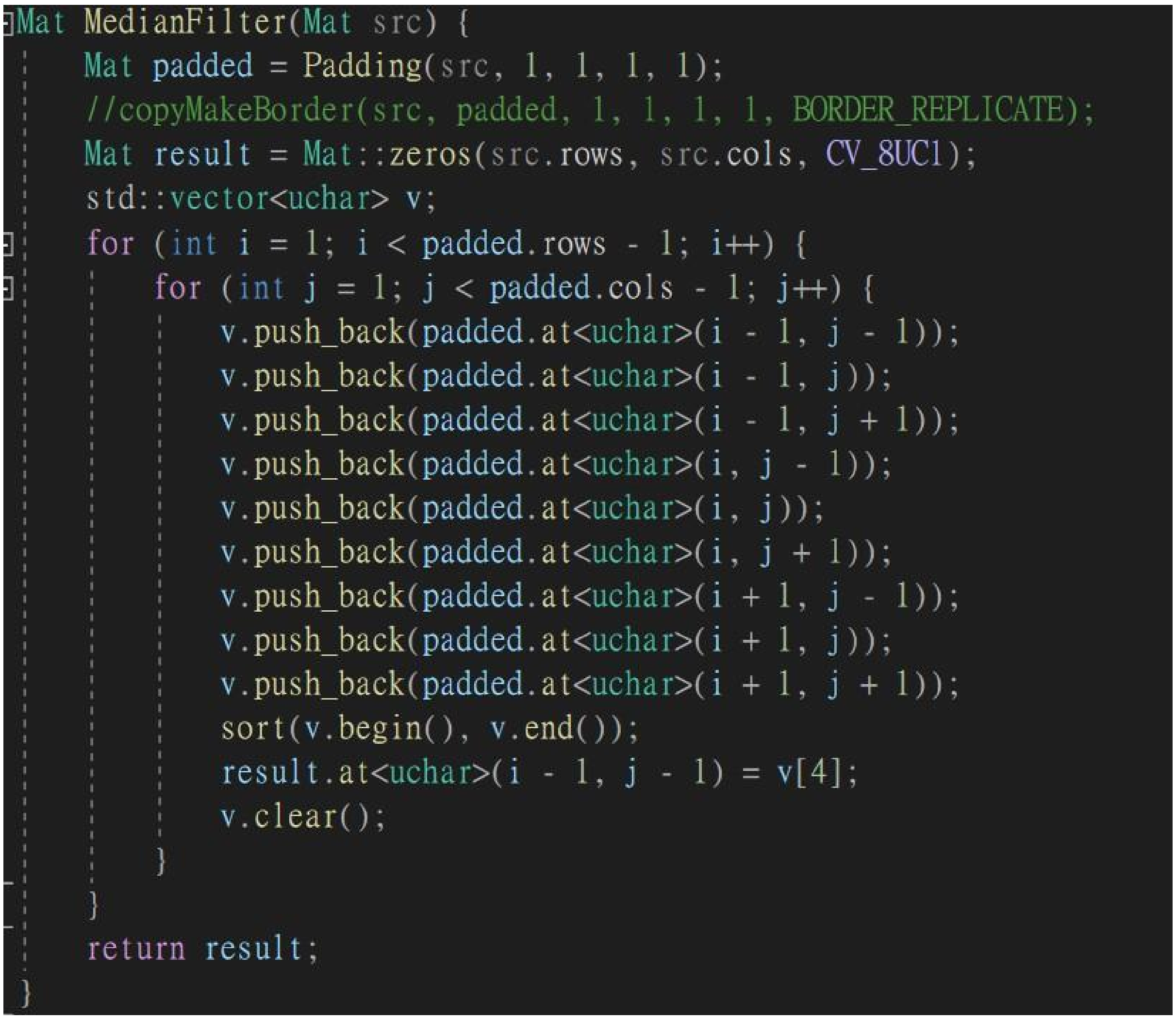
```
Mat CalculateHistogram(Mat src) {
    Mat histogram = Mat::zeros(256, 1, CV_32F);
    uchar value = 0;
    for (int i = 0; i < src.rows; i++) {
        for (int j = 0; j < src.cols; j++) {
            value = src.at<uchar>(i, j);
            histogram.at<float>(value,0) = histogram.at<float>(value,0) + 1;
        }
    }
    return histogram;
}
```



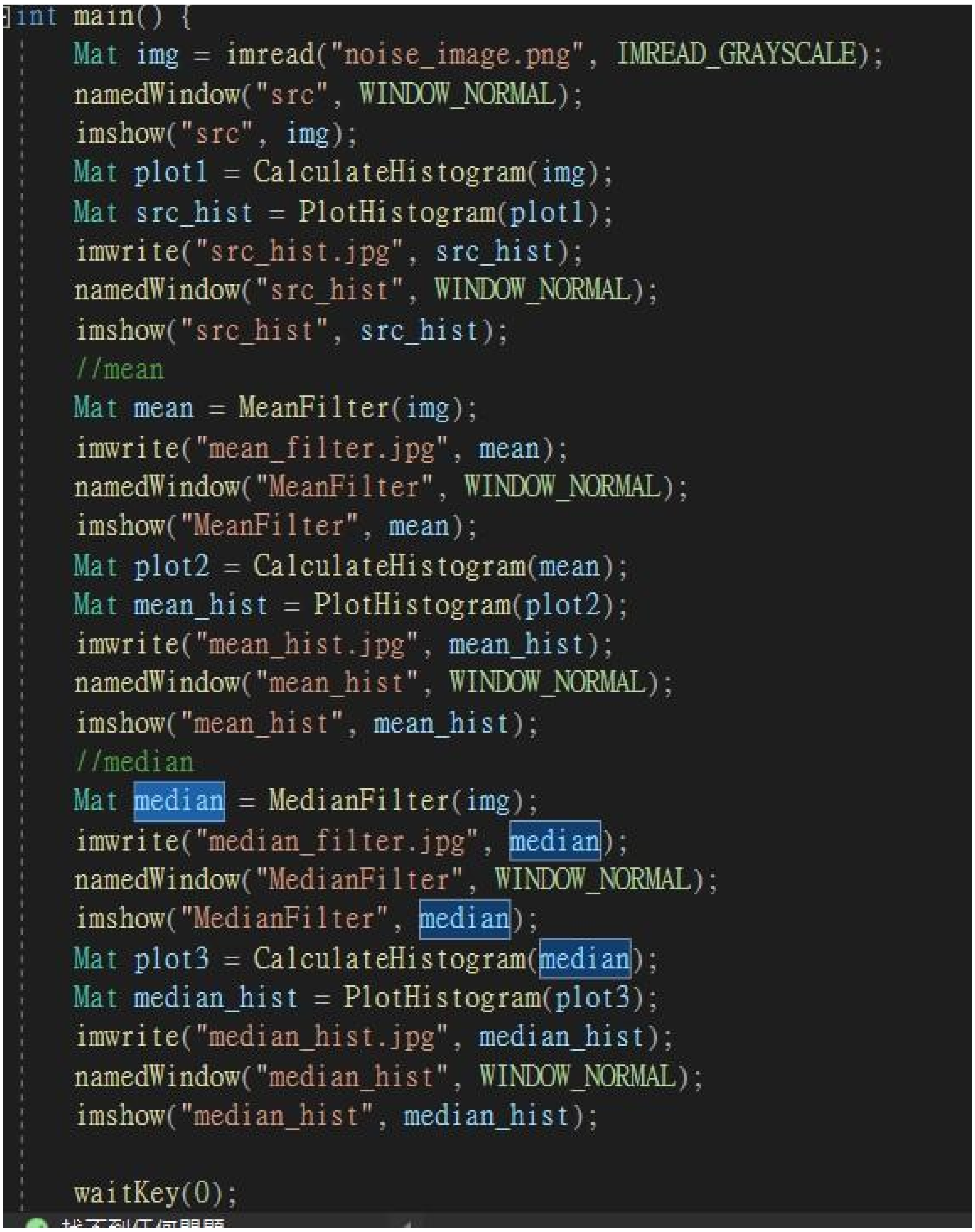
```
Mat PlotHistogram(Mat src) {
    Mat hist_image(500, 512, CV_8UC1, Scalar(0,0,0));
    Mat hist_normal;
    normalize(src, hist_normal, 0, 500, NORM_MINMAX, -1, Mat());
    for (int i = 0; i < 256; i++) {
        rectangle(hist_image, Point(2 * i, hist_image.rows - hist_normal.at<float>(i)),
            Point(2 * (i + 1), hist_image.rows), Scalar(255, 0, 0));
    }
    return hist_image;
}
```



```
Mat MeanFilter(Mat src) {
    const int scale = 9;
    Mat padded = Padding(src, 1, 1, 1, 1);
    //copyMakeBorder(src, padded, 1, 1, 1, 1, BORDER_REPLICATE);
    Mat result = Mat::zeros(src.rows, src.cols, CV_8UC1);
    for (int i = 1; i < padded.rows- 1; i++) {
        for (int j = 1; j < padded.cols- 1; j++) {
            int value = 0;
            value += padded.at<uchar>(i - 1, j - 1);
            value += padded.at<uchar>(i - 1, j);
            value += padded.at<uchar>(i - 1, j + 1);
            value += padded.at<uchar>(i, j - 1);
            value += padded.at<uchar>(i, j);
            value += padded.at<uchar>(i, j + 1);
            value += padded.at<uchar>(i + 1, j - 1);
            value += padded.at<uchar>(i + 1, j);
            value += padded.at<uchar>(i + 1, j + 1);
            value /= scale;
            result.at<uchar>(i-1, j-1) = value;
        }
    }
    return result;
}
```



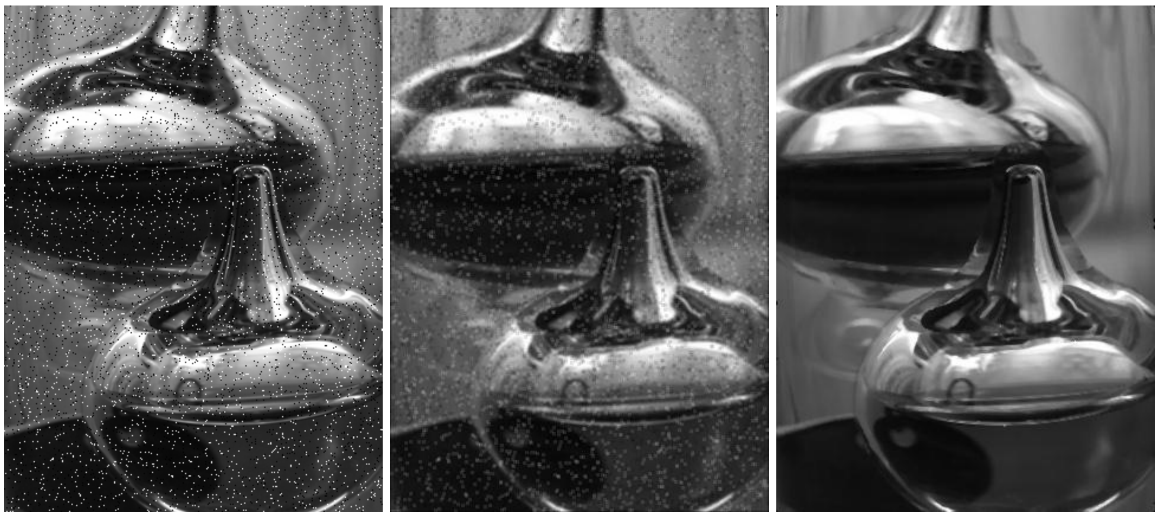
```cpp
Mat MedianFilter(Mat src) {
    Mat padded = Padding(src, 1, 1, 1, 1);
    //copyMakeBorder(src, padded, 1, 1, 1, 1, BORDER_REPLICATE);
    Mat result = Mat::zeros(src.rows, src.cols, CV_8UC1);
    std::vector<uchar> v;
    for (int i = 1; i < padded.rows - 1; i++) {
        for (int j = 1; j < padded.cols - 1; j++) {
            v.push_back(padded.at<uchar>(i - 1, j - 1));
            v.push_back(padded.at<uchar>(i - 1, j));
            v.push_back(padded.at<uchar>(i - 1, j + 1));
            v.push_back(padded.at<uchar>(i, j - 1));
            v.push_back(padded.at<uchar>(i, j));
            v.push_back(padded.at<uchar>(i, j + 1));
            v.push_back(padded.at<uchar>(i + 1, j - 1));
            v.push_back(padded.at<uchar>(i + 1, j));
            v.push_back(padded.at<uchar>(i + 1, j + 1));
            sort(v.begin(), v.end());
            result.at<uchar>(i - 1, j - 1) = v[4];
            v.clear();
        }
    }
    return result;
}
```

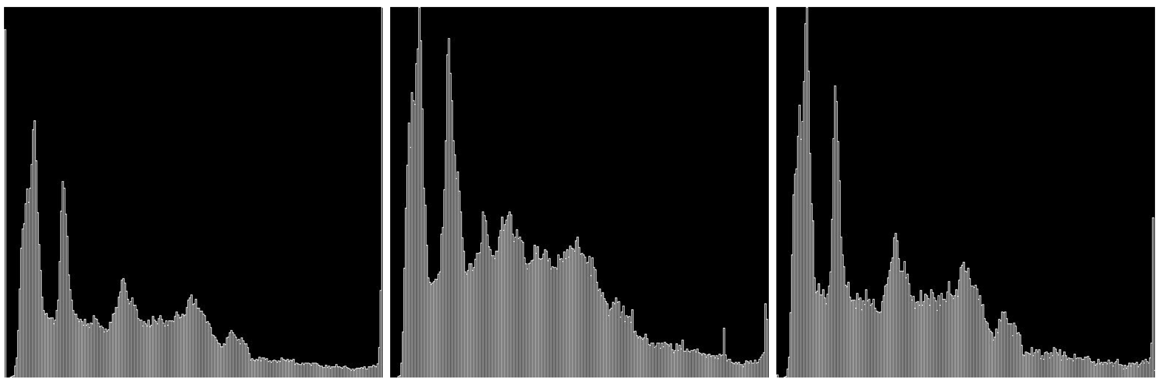


```
int main() {
    Mat img = imread("noise_image.png", IMREAD_GRAYSCALE);
    namedWindow("src", WINDOW_NORMAL);
    imshow("src", img);
    Mat plot1 = CalculateHistogram(img);
    Mat src_hist = PlotHistogram(plot1);
    imwrite("src_hist.jpg", src_hist);
    namedWindow("src_hist", WINDOW_NORMAL);
    imshow("src_hist", src_hist);
    //mean
    Mat mean = MeanFilter(img);
    imwrite("mean_filter.jpg", mean);
    namedWindow("MeanFilter", WINDOW_NORMAL);
    imshow("MeanFilter", mean);
    Mat plot2 = CalculateHistogram(mean);
    Mat mean_hist = PlotHistogram(plot2);
    imwrite("mean_hist.jpg", mean_hist);
    namedWindow("mean_hist", WINDOW_NORMAL);
    imshow("mean_hist", mean_hist);
    //median
    Mat median = MedianFilter(img);
    imwrite("median_filter.jpg", median);
    namedWindow("MedianFilter", WINDOW_NORMAL);
    imshow("MedianFilter", median);
    Mat plot3 = CalculateHistogram(median);
    Mat median_hist = PlotHistogram(plot3);
    imwrite("median_hist.jpg", median_hist);
    namedWindow("median_hist", WINDOW_NORMAL);
    imshow("median_hist", median_hist);

    waitKey(0);
```

Result(source, mean, median):





As pictures above, we can see that median filter works better than mean filter regarding to salt and pepper noise. The mean filter calculate the mean value of nearby 3x3 area, so we can see the shape of the histogram greatly change. The median filter choose the median value in the 3x3 area, so the shape of the histogram is similar to the source image.