Homework requirement:

1. Read a RGB image and write a function to convert the image to **grayscale image**.
2. Write a convolution operation with **edge detection *kernel*** and **activation function** (ReLU: rectified linear unit)
3. Write a pooling operation with using **Max pooling, 2x2 filter, and stride 2**
4. Write a **binarization operation** (threshold = 128). (>=128) set 255 (<128) set 0

Development environment:

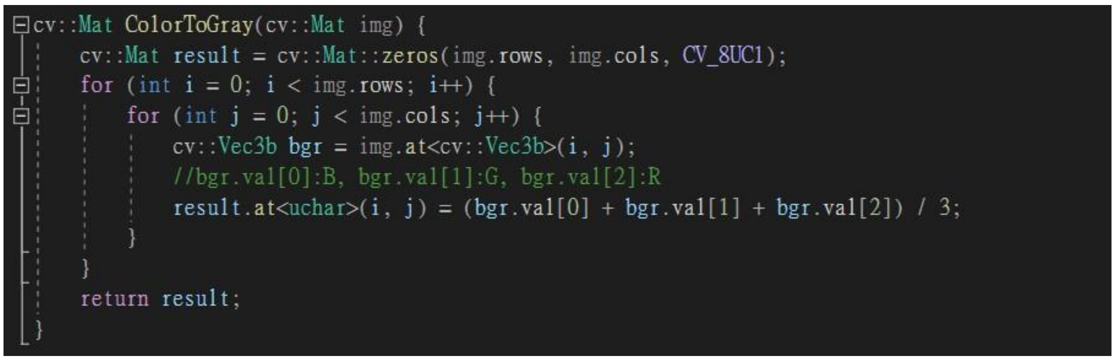
Windows 10

Visual Studio 2022 community

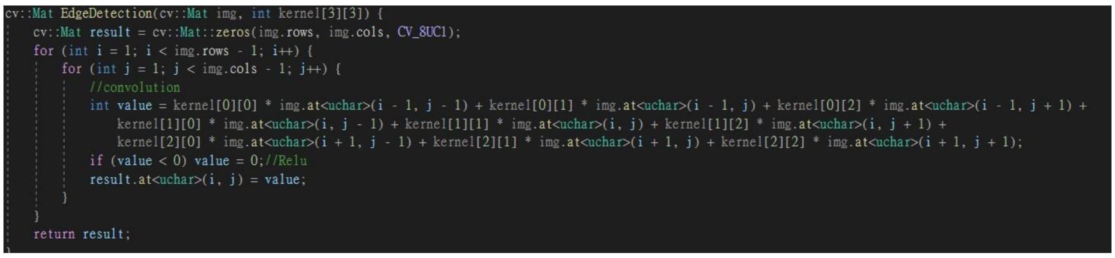
OpenCV 4.6.0

Implementation:

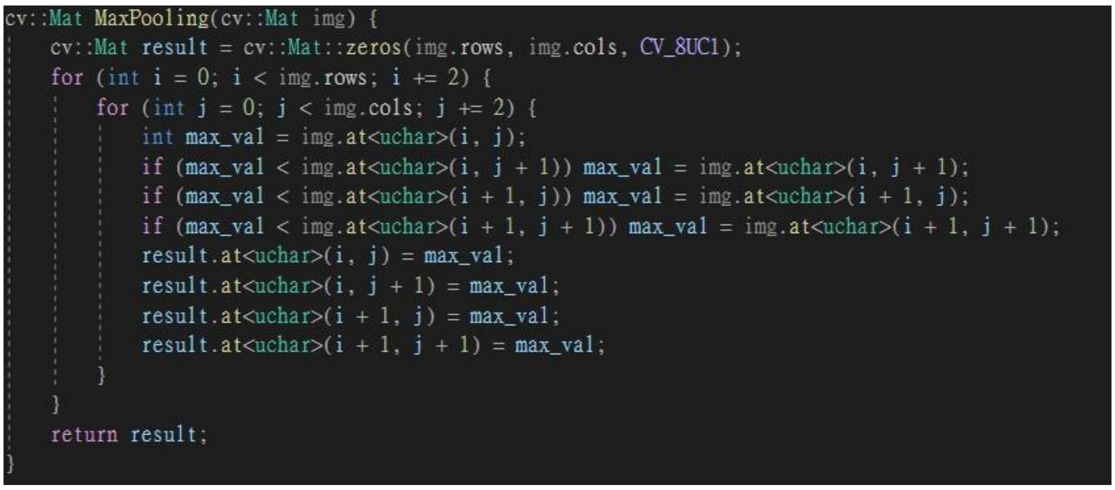
0. Load the image and check row number and column number of the image. If row number or column number is odd, pad the image with zeros.
1. Iterate through each pixel and calculate the average value of three channels. Store the average value in result1.
2. Iterate from pixel(1, 1) to pixel(rows-1, cols-1). Calculate element-wise multiplication with an edge detection kernel. If the output is less than zero set the output as zero and then store value in result2. Since the multiplication cannot be done at the edge of the image, the edge value of result2 is 0.
3. Iterate through the image with the increment of 2. Compare all value in the nearby 2x2 area. Find the max value and store in the corresponding area in result3.
4. Iterate through the image and read the value of each pixel. If the pixel value is less than 128, output 0. If the pixel value is greater than or equall to 128, output 255. Store the output in the corresponding pixel in result4.



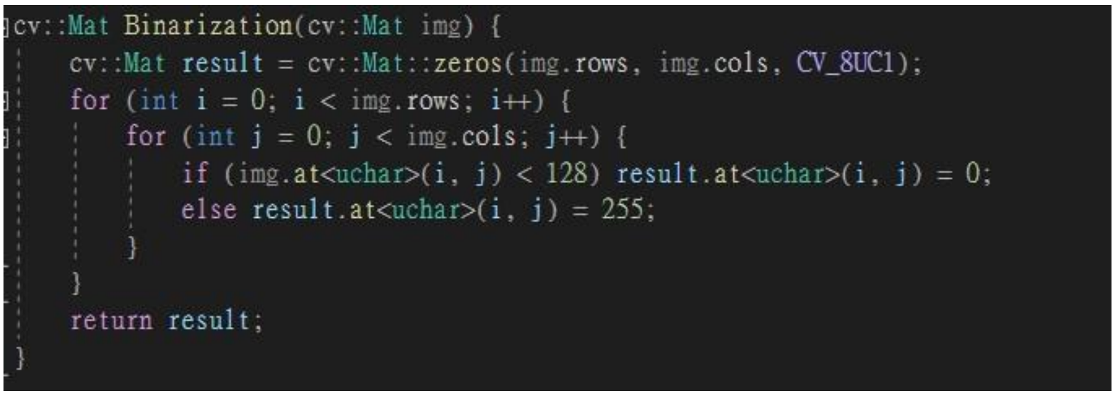
```
cv::Mat ColorToGray(cv::Mat img) {
    cv::Mat result = cv::Mat::zeros(img.rows, img.cols, CV_8UC1);
    for (int i = 0; i < img.rows; i++) {
        for (int j = 0; j < img.cols; j++) {
            cv::Vec3b bgr = img.at<cv::Vec3b>(i, j);
            //bgr.val[0]:B, bgr.val[1]:G, bgr.val[2]:R
            result.at<uchar>(i, j) = (bgr.val[0] + bgr.val[1] + bgr.val[2]) / 3;
        }
    }
    return result;
}
```



```cpp
cv::Mat EdgeDetection(cv::Mat img, int kernel[3][3]) {
    cv::Mat result = cv::Mat::zeros(img.rows, img.cols, CV_8UC1);
    for (int i = 1; i < img.rows - 1; i++) {
        for (int j = 1; j < img.cols - 1; j++) {
            //convolution
            int value = kernel[0][0] * img.at<uchar>(i - 1, j - 1) + kernel[0][1] * img.at<uchar>(i - 1, j) + kernel[0][2] * img.at<uchar>(i - 1, j + 1) +
                kernel[1][0] * img.at<uchar>(i, j - 1) + kernel[1][1] * img.at<uchar>(i, j) + kernel[1][2] * img.at<uchar>(i, j + 1) +
                kernel[2][0] * img.at<uchar>(i + 1, j - 1) + kernel[2][1] * img.at<uchar>(i + 1, j) + kernel[2][2] * img.at<uchar>(i + 1, j + 1);
            if (value < 0) value = 0;//Relu
            result.at<uchar>(i, j) = value;
        }
    }
    return result;
}
```

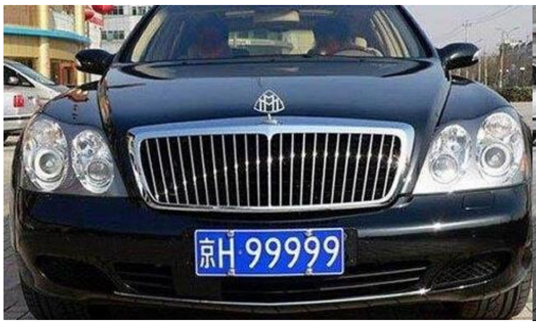


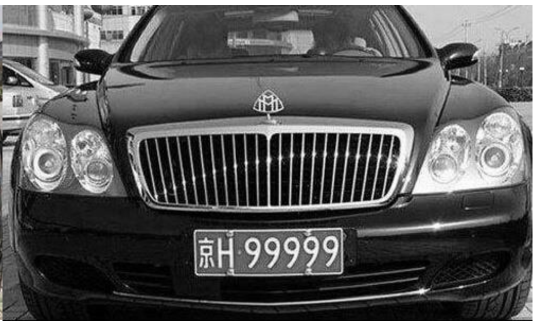
```cpp
cv::Mat MaxPooling(cv::Mat img) {
    cv::Mat result = cv::Mat::zeros(img.rows, img.cols, CV_8UC1);
    for (int i = 0; i < img.rows; i += 2) {
        for (int j = 0; j < img.cols; j += 2) {
            int max_val = img.at<uchar>(i, j);
            if (max_val < img.at<uchar>(i, j + 1)) max_val = img.at<uchar>(i, j + 1);
            if (max_val < img.at<uchar>(i + 1, j)) max_val = img.at<uchar>(i + 1, j);
            if (max_val < img.at<uchar>(i + 1, j + 1)) max_val = img.at<uchar>(i + 1, j + 1);
            result.at<uchar>(i, j) = max_val;
            result.at<uchar>(i, j + 1) = max_val;
            result.at<uchar>(i + 1, j) = max_val;
            result.at<uchar>(i + 1, j + 1) = max_val;
        }
    }
    return result;
}
```

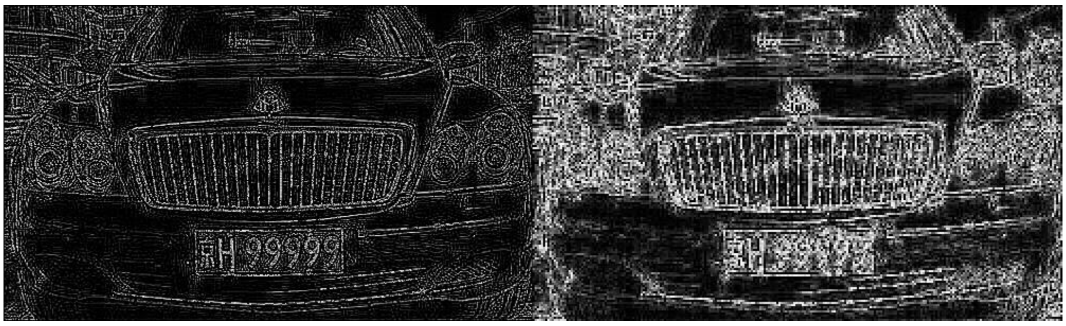


```cpp
cv::Mat Binarization(cv::Mat img) {
    cv::Mat result = cv::Mat::zeros(img.rows, img.cols, CV_8UC1);
    for (int i = 0; i < img.rows; i++) {
        for (int j = 0; j < img.cols; j++) {
            if (img.at<uchar>(i, j) < 128) result.at<uchar>(i, j) = 0;
            else result.at<uchar>(i, j) = 255;
        }
    }
    return result;
}
```

Results:






H 99999