Student: Michael Grossman

Project Due Date: 03/02/2022

Algorithm Steps for Computing Corner Preserving Averages given an array of 2d 5x5 masks named masks, a 2d array representing the framed input image named frameAry, and a 2d array for storing the averages at each pixed called outAry:

1. r ← 2
2. c ← 2
3. maskIndex ← 0
4. minAvg ← frameAry[r][c]
5. minDiff ← 9999
6. result ← convolution5x5( r, c, masks[masksIndex] ) / 9
7. diff ← abs( result – frameAry[r][c] )
8. if diff < minDiff:
9. minDiff ← diff
10. minAvg ← result
11. maskIndex++
12. repeat steps 6 to 11 while maskIndex < 8
13. c++
14. repeat steps 3 to 13 while c < numCols + 2
15. r++
16. repeat steps 3 to 15 while r < numRows + 2

Algorithmic Steps for Computing Image Reformatting for pretty printing with frame given an array to read from named ary, a min pixel value named newMin, a max pixel value named newMax, and an output file named output:

1. output ← output numRows, numCols, newMin, newMax
2. str ← to\_string( newMax )
3. width ← str.length()
4. r ← 2
5. c ← 2
6. output ← ary[r][c]
7. str ← to\_string( ary[r][c] )
8. ww ← str.length()
9. output ← “ “ //one blank space
10. ww++
11. repeat steps 9 to 10 while ww < width
12. c++
13. repeat steps 6 to 12 while c < numCols + 4
14. repeat steps 5 to 13 while r < numRows + 4

Source Code:

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

class imageProcessing{

    public:

    //variables

    int numRows, numCols, minVal, maxVal, thrVal;

    int \*\*frameAry, \*\*outAry, \*\*thrAry;

    int\*\*\* mask;

    //constructor + destructor

    imageProcessing(int\* vals, int thrv);

    ~imageProcessing();

    //functions

    void loadImage(ifstream& input);

    void mirrorFraming();

    void loadMask();

    int convolution5x5(int i, int j, int maskind);

    void cornerPreserveAvg();

    void threshold(int\*\* ary);

    void imgReformat(int\*\* inAry, int newMin, int newMax, ofstream& output);

};

int main(int argc, char\*\* argv){

    //get the input information

    string inputFilename = argv[1];

    string outputfile1 = argv[3];

    string outputFile2 = argv[4];

    //open the streams

    ifstream input(inputFilename);

    ofstream outFile1(outputfile1);

    ofstream outFile2(outputFile2);

    //get the threshold value

    int thresholdValue = atoi(argv[2]);

    //get the image header, #rows, #cols, #min, #max

    int imageSpecs[4];

    for(int i = 0; i < 4; ++i){

        input >> imageSpecs[i];

    }

    //create the image processing object, inits the arrays and frames

    imageProcessing imageprocessing(imageSpecs, thresholdValue);

    imageprocessing.loadImage(input);

    imageprocessing.mirrorFraming();

    //load all the masks from files named mask[i].txt

    //for 1 <= [i] <= 8

    imageprocessing.loadMask();

    //pretty print the input as is

    imageprocessing.imgReformat(imageprocessing.frameAry,

                                imageprocessing.minVal, imageprocessing.maxVal,

                                 outFile1);

    //threshold on the given value, store in thrAry, and pretty print that output

    imageprocessing.threshold(imageprocessing.frameAry);

    imageprocessing.imgReformat(imageprocessing.thrAry, 0, 1, outFile1);

    //take the 5x5 convolutions for every pixel and store it in outAry,

    //and pretty print it

    imageprocessing.cornerPreserveAvg();

    imageprocessing.imgReformat(imageprocessing.outAry, imageprocessing.minVal,

                                imageprocessing.maxVal, outFile1);

    //threshold outAry on the given value and pretty print it

    imageprocessing.threshold(imageprocessing.outAry);

    imageprocessing.imgReformat(imageprocessing.thrAry, 0, 1, outFile1);

    //output threshold array without frame to output2

    outFile2 << imageprocessing.numRows << " " << imageprocessing.numCols ;

    outFile2 << " " << 0 << " " << 1 << "\n";

    for(int i = 2; i < imageprocessing.numRows + 2; ++i){

        for(int j = 2; j < imageprocessing.numCols + 2; ++j){

            outFile2 << imageprocessing.thrAry[i][j] << " ";

        }

        outFile2 << "\n";

    }

    //close all streams

    input.close();

    outFile1.close();

    outFile2.close();

    return 0;

}

imageProcessing::imageProcessing(int\* vals, int thrv){

    numRows = vals[0];

    numCols = vals[1];

    minVal = vals[2];

    maxVal = vals[3];

    thrVal = thrv;

    int frameSizeRows = numRows + 4, frameSizeCols = numCols + 4;

    frameAry = new int\*[frameSizeRows];

    outAry = new int\*[frameSizeRows];

    thrAry = new int\*[frameSizeRows];

    for(int i = 0; i < frameSizeRows; ++i){

        frameAry[i] = new int[frameSizeCols]{0};

        outAry[i] = new int[frameSizeCols]{0};

        thrAry[i] = new int[frameSizeCols]{0};

    }

    mask = new int\*\*[8];

    for(int i = 0; i < 8; ++i){

        mask[i] = new int\*[5];

        for(int j = 0; j < 5; ++j){

            mask[i][j] = new int[5];

        }

    }

}

imageProcessing::~imageProcessing(){

    int frameSizeRows = numRows + 4;

    for(int i = 0; i < frameSizeRows; ++i){

        delete[] frameAry[i];

        delete[] outAry[i];

        delete[] thrAry[i];

    }

    delete[] frameAry;

    delete[] outAry;

    delete[] thrAry;

    for(int i = 0; i < 8; ++i){

        for(int j = 0; j < 5; ++j){

            delete[] mask[i][j];

        }

        delete[] mask[i];

    }

    delete[] mask;

}

void imageProcessing::loadImage(ifstream& input){

    int rows = numRows+2, cols = numCols + 2;

    for(int i = 2; i < rows; ++i){

        for(int j = 2; j < cols; ++j){

            input >> frameAry[i][j];

        }

    }

}

void imageProcessing::mirrorFraming(){

    int frameRows = numRows + 4, frameCols = numCols + 4;

    //mirror top then bottom

    for(int i = 0; i < 2; ++i){

        for(int j = 2; j < numCols+2; ++j){

            frameAry[i][j] = frameAry[3-i][j];

        }

    }

    for(int i = frameRows- 2; i < frameRows; ++i){

        for(int j = 2; j < numCols + 2; ++j){

            frameAry[i][j] = frameAry[2\*frameRows-5 - i][j];

        }

    }

    //mirror left then right

    for(int i = 2; i < frameRows-2; ++i){

        for(int j = 0; j < 2; ++j){

            frameAry[i][j] = frameAry[i][3-j];

        }

    }

    for(int i = 2; i < frameRows - 2; ++i){

        for(int j = frameCols-2; j < frameCols; ++j){

            frameAry[i][j] = frameAry[i][2\*frameCols-5-j];

        }

    }

    //mirror corners, reflected over appropriate corner

    frameAry[0][0] = frameAry[3][3];

    frameAry[1][1] = frameAry[2][2];

    frameAry[0][1] = frameAry[2][3];

    frameAry[1][0] = frameAry[3][2];

    frameAry[0][frameCols-2] = frameAry[2][frameCols-4];

    frameAry[0][frameCols-1] = frameAry[3][frameCols-4];

    frameAry[1][frameCols-2] = frameAry[2][frameCols-3];

    frameAry[1][frameCols-1] = frameAry[3][frameCols-3];

    frameAry[frameRows-2][0] = frameAry[frameRows-4][2];

    frameAry[frameRows-2][1] = frameAry[frameRows-3][2];

    frameAry[frameRows-1][0] = frameAry[frameRows-4][3];

    frameAry[frameRows-1][1] = frameAry[frameRows-3][3];

    frameAry[frameRows-2][frameCols-2] = frameAry[frameRows-3][frameCols-3];

    frameAry[frameRows-2][frameCols-1] = frameAry[frameRows-4][frameCols-3];

    frameAry[frameRows-1][frameCols-2] = frameAry[frameRows-3][frameCols-4];

    frameAry[frameRows-1][frameCols-1] = frameAry[frameRows-4][frameCols-4];

}

void imageProcessing::loadMask(){

    int rows, cols, mnv, mxv;

    for(int i = 1; i<= 8; ++i){

        ifstream maskInput("mask" + to\_string(i) + ".txt");

        maskInput >> rows;

        maskInput >> cols;

        maskInput >> mnv;

        maskInput >> mxv;

        for(int j = 0; j < rows; ++j){

            for(int k = 0; k < cols; ++k){

                maskInput >> mask[i-1][j][k];

            }

        }

    }

}

int imageProcessing::convolution5x5(int i, int j, int maskind){

    int retVal = 0;

    for(int rows = i-2; rows <= i + 2; ++rows){

        for(int cols = j-2; cols <= j+2; ++cols){

            retVal += mask[maskind][rows-i+2][cols-j+2]\*frameAry[rows][cols];

        }

    }

    return retVal;

}

void imageProcessing::cornerPreserveAvg(){

    int minAvg = 0, minDiff = 9999, result = 0, diff = 0;

    for(int r = 2; r < numRows+2; ++r){

        for(int c = 2; c < numCols+2; ++c){

            minAvg = frameAry[r][c], minDiff = 9999;

            for(int maskIndex = 0; maskIndex < 8; ++maskIndex){

                result = 1.0\*convolution5x5(r, c, maskIndex)/9;

                diff = abs(result - frameAry[r][c]);

                if(diff < minDiff){

                    minDiff = diff;

                    minAvg = result;

                }

            }

            outAry[r][c] = minAvg;

        }

    }

}

void imageProcessing::threshold(int\*\* ary){

    for(int i = 0; i < numRows+4; ++i){

        for(int j = 0; j < numCols + 4; ++j){

            thrAry[i][j] = ary[i][j] >= thrVal ? 1 : 0;

        }

    }

}

void imageProcessing::imgReformat(int\*\* inAry, int newMin, int newMax,

                                    ofstream& output){

    output << numRows << " ";

    output << numCols << " ";

    output << newMin << " ";

    output << newMax << "\n";

    string str = to\_string(newMax);

    int width = str.length();

    int r =0, c =0, ww = 0;

    for(int r = 0; r < numRows+4; ++r){

        for(int c = 0; c < numCols+4; ++c){

            output << inAry[r][c];

            str = to\_string(inAry[r][c]);

            output << " ";

            for(ww = str.length(); ww < width; ++ww){

                output << " ";

            }

        }

        output << "\n";

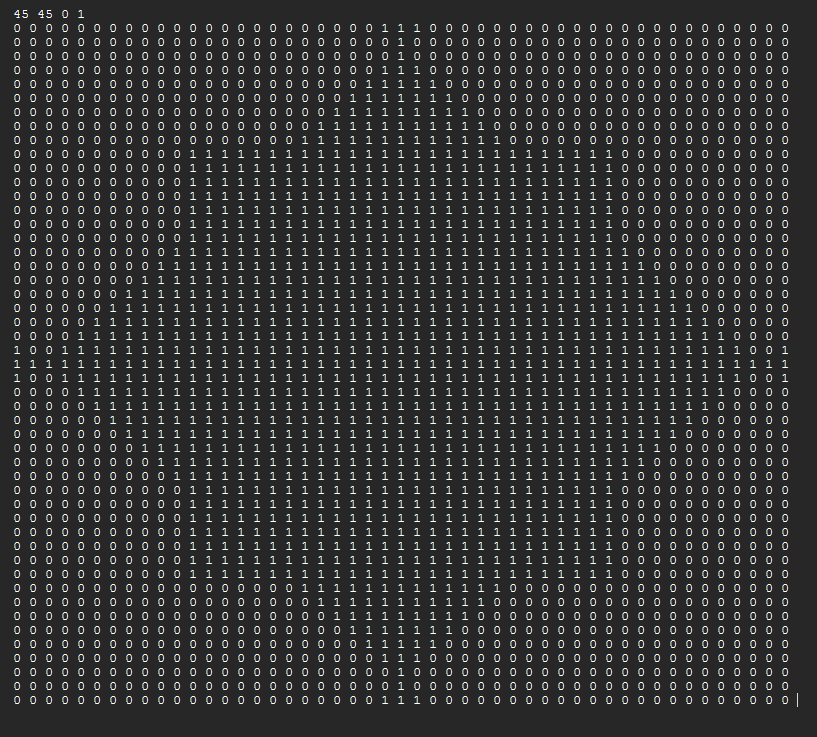
    }

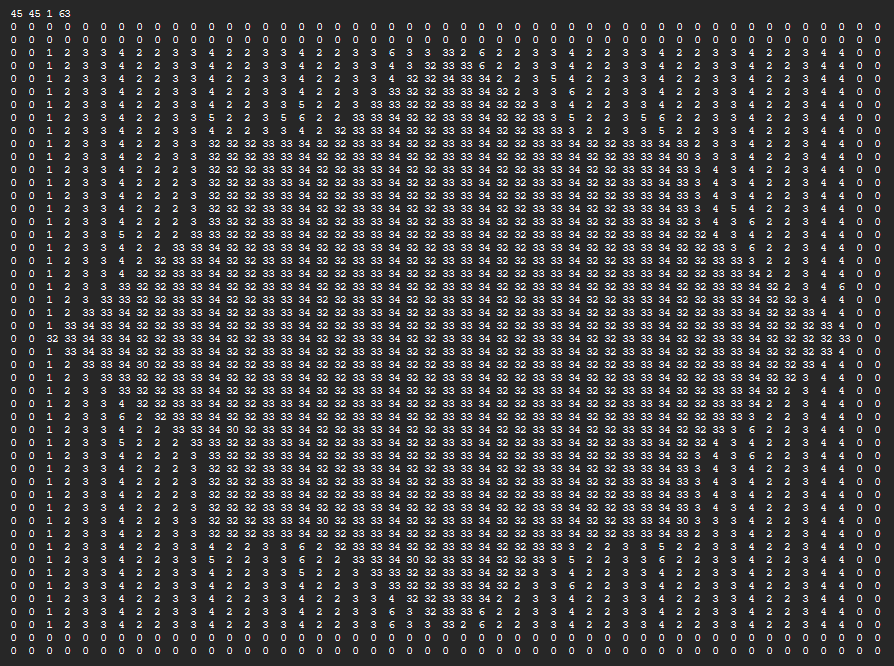
    output << "\n";

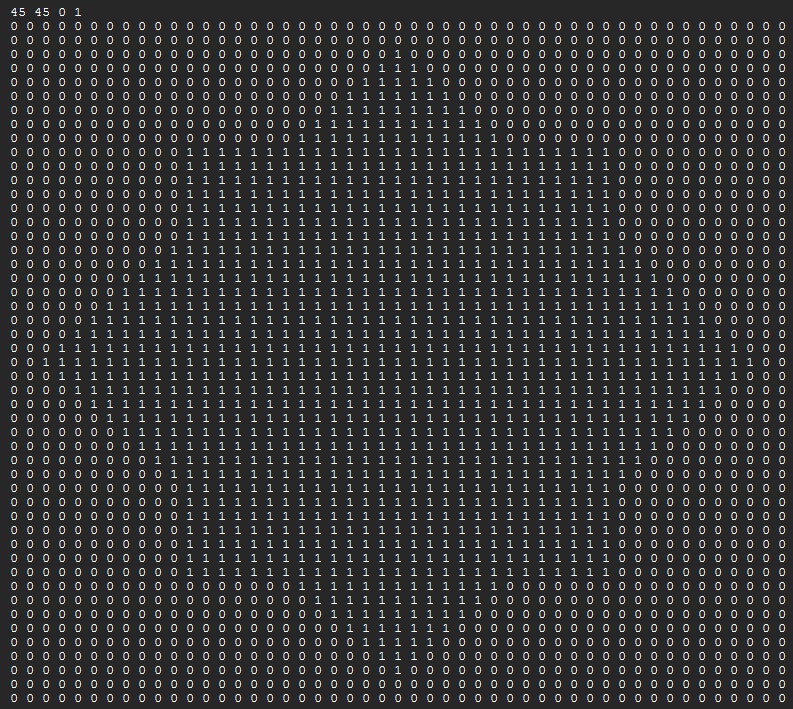
}

Output File 1:









Output File 2:

