Student: Michael Grossman

Project Due Date: 3/17/2022

Algorithm Steps for ComputeGauss given a frameArray, maskArray, gaussArray, and a weight:

1. newMin←9999 ; newMax ←0
2. i←2
3. j←2
4. gaussArray[i,j]←(int)(convolution5x5(i,j,frameArray,maskArray) / weight)
5. if newMin > gaussArray[i,j]
6. newMin ← gaussArray[i,j]
7. end-if
8. if newMax < gaussArray[i,j]
9. newMax ← gaussArray[i,j]
10. end-if
11. j++
12. repeat 3 to 10 while j < numCols + 2
13. i++
14. repeate 3 to 12 while i < numRows + 2

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

class imageProcessing{

    public:

        int numRows, numCols, minVal, maxVal;

        int newMin, newMax;

        int maskRows, maskCols, maskMin, maskMax;

        int thr;

        int \*\*frameArray, \*\*gaussArray, \*\*thrArray, \*\*maskArray;

        int weight;

        imageProcessing(int\* imH, int\* mskH, int t);

        ~imageProcessing();

        void loadImage(ifstream& in, int\*\* ary);

        void mirrorFraming();

        int loadMask(ifstream& in);

        int convolutions5x5(int i, int j, int\*\* fArray, int\*\* mArray);

        void computeGaus(int\*\* fAarray, int\*\* mArray, int\*\* gArray, int w);

        void threshold(int\*\* ary, int\*\* tAry, int t);

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

        void prettyPrint(int\*\* ary, ofstream& out);

};

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

    string inputFileName = argv[1], maskFileName = argv[2], outputFileName = argv[4];

    int threshold = atoi(argv[3]);

    ifstream input(inputFileName), mask(maskFileName);

    ofstream output(outputFileName);

    int inputHeader[4], maskHeader[4];

    for(int i = 0; i < 4; ++i){

        input >> inputHeader[i];

        mask >> maskHeader[i];

    }

    imageProcessing imageprocessing(inputHeader, maskHeader, threshold);

    imageprocessing.loadImage(input, imageprocessing.frameArray);

    imageprocessing.weight = imageprocessing.loadMask(mask);

    imageprocessing.imageReformat(imageprocessing.frameArray, imageprocessing.minVal, imageprocessing.maxVal, output);

    imageprocessing.computeGaus(imageprocessing.frameArray, imageprocessing.maskArray, imageprocessing.gaussArray, imageprocessing.weight);

    imageprocessing.imageReformat(imageprocessing.gaussArray, imageprocessing.newMin, imageprocessing.newMax, output);

    imageprocessing.threshold(imageprocessing.gaussArray, imageprocessing.thrArray, imageprocessing.thr);

    imageprocessing.imageReformat(imageprocessing.thrArray, imageprocessing.newMin, imageprocessing.newMax, output);

    imageprocessing.prettyPrint(imageprocessing.thrArray, output);

    input.close();

    output.close();

    mask.close();

}

imageProcessing::imageProcessing(int\* imH, int\* mskH, int t){

    numRows = imH[0];

    numCols = imH[1];

    minVal = imH[2];

    maxVal = imH[3];

    maskRows = mskH[0];

    maskCols = mskH[1];

    maskMin = mskH[2];

    maskMax = mskH[3];

    thr = t;

    frameArray = new int\*[numRows + 4];

    gaussArray = new int\*[numRows + 4];

    thrArray = new int\*[numRows + 4];

    maskArray = new int\*[maskRows];

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

        frameArray[i] = new int[numCols + 4]{0};

        gaussArray[i] = new int[numCols + 4]{0};

        thrArray[i] = new int[numCols + 4]{0};

    }

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

        maskArray[i] = new int[maskCols]{0};

    }

}

imageProcessing::~imageProcessing(){

    int frameSizeRows = numRows + 4;

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

        delete[] frameArray[i];

        delete[] gaussArray[i];

        delete[] thrArray[i];

    }

    delete[] frameArray;

    delete[] gaussArray;

    delete[] thrArray;

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

        delete[] maskArray[i];

    }

    delete[] maskArray;

}

void imageProcessing::loadImage(ifstream& in, int\*\* ary){

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

    for(int i = 2; i < rows; ++i){

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

            in >> frameArray[i][j];

        }

    }

    mirrorFraming();

}

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){

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

        }

    }

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

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

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

        }

    }

    //mirror left then right

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

        for(int j = 0; j < 2; ++j){

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

        }

    }

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

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

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

        }

    }

    //mirror corners, reflected over appropriate corner

    frameArray[0][0] = frameArray[3][3];

    frameArray[1][1] = frameArray[2][2];

    frameArray[0][1] = frameArray[2][3];

    frameArray[1][0] = frameArray[3][2];

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

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

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

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

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

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

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

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

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

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

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

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

}

int imageProcessing::loadMask(ifstream& in){

    int w = 0;

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

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

            in >> maskArray[i][j];

            w += maskArray[i][j];

        }

    }

    return w;

}

int imageProcessing::convolutions5x5(int i, int j, int\*\* fArray, int\*\* mArray){

    int retVal = 0;

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

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

            retVal += mArray[rows-i+2][cols-j+2]\*fArray[rows][cols];

        }

    }

    return retVal;

}

void imageProcessing::computeGaus(int\*\* fAarray, int\*\* mArray, int\*\* gArray, int w){

    newMin = 99999;

    newMax = 0;

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

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

            gArray[i][j] = (int) (convolutions5x5(i, j, fAarray, mArray) / w);

            newMin = newMin > gArray[i][j] ? gArray[i][j] : newMin;

            newMax = newMax < gArray[i][j] ? gArray[i][j] : newMax;

        }

    }

}

void imageProcessing::threshold(int\*\* ary, int\*\* tAry, int t){

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

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

            tAry[i][j] = ary[i][j] >= t ? 1 : 0;

        }

    }

}

void imageProcessing::imageReformat(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\n";

}

void imageProcessing::prettyPrint(int\*\* ary, ofstream& out){

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

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

            if(ary[i][j] > 0){

                out << "1 ";

            }

            else{

                out << ". ";

            }

        }

        out << "\n";

    }

    out << "\n\n";

}







