CA1 – Two-Dimensional Convolution

Nik Jensen
ECE 5630

This assignment was to give practice in coding to perform digital signal processing and to help solidify my understanding of two-dimensional convolution. The assignment was broken up into 4 different topics of convolution, involving filtering with 2D filters, lowpass filters, a Sobel filter, and using correlation.

Part 1 – Convolving with an arbitrary 2D filter

For this portion of the assignment, I was tasked to write a program that will read in the provided image.pgm, filter with an arbitrary-sized 2D filter, and then write out to an output file of type .pgm. Taking note of the code in the appendix, this was done by parsing the image, convolving with the created filter, and outputting to a new file. Below is the arbitrary-sized 2D filter.

$$Filter = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \cdot \frac{1}{16}$$

Results are shown in Figure 1, where the left image is the original parsed image and the right is the resulting filtered image. An added glow effect was also added to show padding around image better. It can be seen that the image to the right is less saturated and even slightly blurred.

Figure 1 Convolution with an arbitrary-sized 2D filter. Left is original, right is the resulted image.

Part 2 – Convolving with a provided lowpass filter

The next portion of the assignment asked to convolve the same parsed image.pgm with a lowpass filter provided. Below is the provided lowpass filter.

$$\mathbf{H}_1 = \frac{1}{81}\begin{bmatrix} 1 & 2 & 3 & 2 & 1 \\ 2 & 4 & 6 & 4 & 2 \\ 3 & 6 & 9 & 6 & 3 \\ 2 & 4 & 6 & 4 & 2 \\ 1 & 2 & 3 & 2 & 1 \end{bmatrix}$$

Given that part 2 of the assignment was so similar to part 1, the results came easily as not much was changed. Results are shown in Figure 2 below. Something to take note of is how the image was blurred slightly. This was due to convolving an image with a 2D lowpass filter. By adjusting the scaling factor, this can alter the blurring effect as well.

Figure 2 Lowpass filtering blur effect. Left is the original image, right is the resulting image.

Part 3 – Sobel Edge Detection

Next involved the convolution of two filters. This can be seen in the code provided in the appendix. This filtering involved the use of the Sobel filter. How it is implemented is by convolving the image with a horizontal and vertical Sobel filter, which in turn adds a comparatively edge, or gradient, to each resulting filtered image. The two filtered images are then combined by calculating the gradient magnitude, which is then converted to its correct numerical type and written to a new filtered .pgm file. Below are the two provided Sobel filters.

$$\mathbf{S}_1 = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \qquad \mathbf{S}_2 = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

As a result of the mention process, the filtered image is going to represent an edge detected version of the original image. In Figure 3 below, we can see the results.

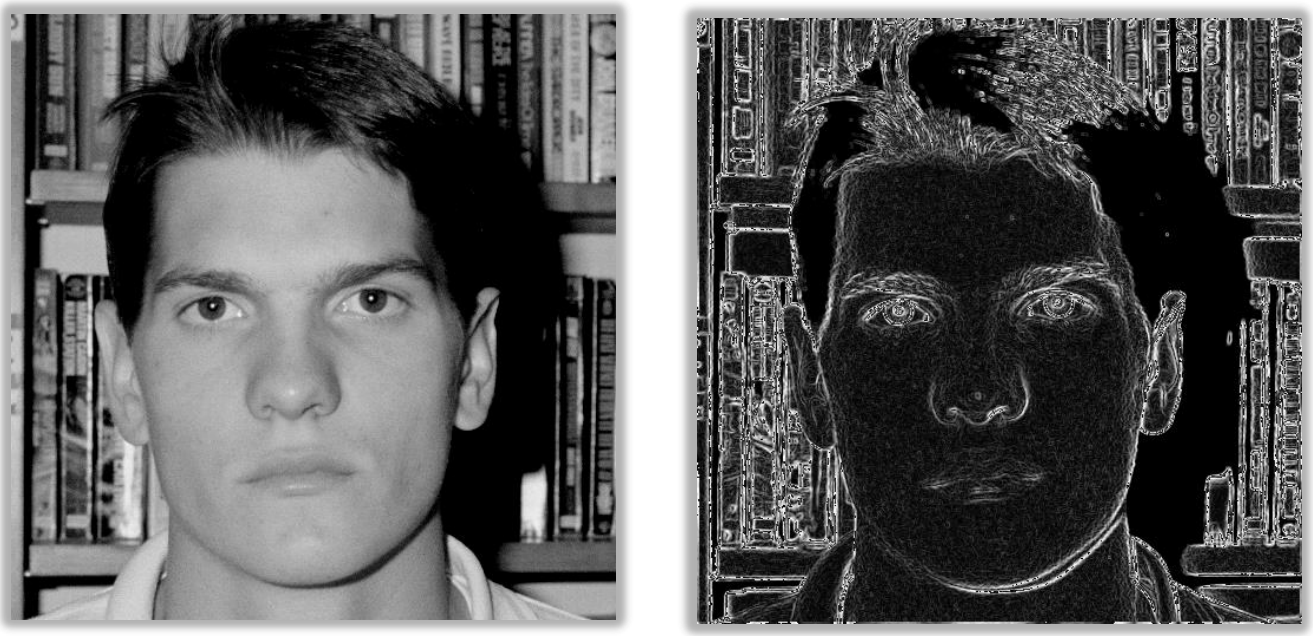

Figure 3 Sobel edge detection results. Left is the original, right is the filtered image with edge detection.

Part 4 – Correlation Filtering

While definitely the most difficult part of the assignment, it was also the most interesting. The process can be viewed in the code provided in the appendix. This part of the assignment involved parsing in an impulse input file to be filtered with the input image.pgm file. The process started by squaring the values of each pixel in the image, then convolving with the parsed impulse file, filter_final.pgm. Next was to create another filter of the same size as the impulse file and convolve that filter with the image. The two results were then divided by each other and truncated to finally output the resulting file. In Figure 4 below, the end results of the part 4 process are shown. While, not entirely sure if correct, this image reflects the application of filtering by using correlation to find features in an image. By viewing Figure 4, it is apparent that features from the image are represented.

Figure 4 Correlation filtering results. Left is the original, right is the filtered image using correlation filtering to find apparent features.

Conclusion

This computer assignment, grew to be a difficult, but strong teacher in how to manipulate and process images using convolution. Some challenges faced involved problems such as reading the file, manipulating the image values with correct parameter type, and convolution with dynamically stored arrays. A major problem faced was reading the image. The provided file type is new to me, and can be quite tedious on trying to parse the file. The next problem involved using correct parameter types. When the image is parsed into the program, it is read in an unsigned char* format. This would then need to be changed to type float in order to manipulate and convolve the image values for output. Lastly, dynamically stored values for the impulse response, input file, and output file can be confusing with nested loops and complex storage of data. While difficult, this assignment did teach a good amount of programming skills and give me a better understanding of digital signal processing in two dimensions.

Appendix (code)

Code for part 1, 2, 3

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
//#include "image2.txt"
//*********************************************************************************//
//  Nik Jensen
//  Digital Signal Image Processing
//  A02250195
//*********************************************************************************//
//pgm file format (MxN matrix = row X col)
//P5 - file type
//Width (n)
//Height (m)
//Max num val in image
//cycles from top row down to bottom row (of size height) while each row is of size width
//*********************************************************************************//

typedef struct{
    char *in_type;
    int in_width;
    int in_height;
    int in_max;
} file_header;

void main(int argc, char** argv) {
    FILE* infile;
    FILE* outfile;
    unsigned char buffer[1096];
    if (NULL == (infile = fopen(argv[1], "rb"))) { //error check and open file
        printf("error: Cannot open image.pgm file for input.\n");
        return;
    }
    if (NULL == (outfile = fopen(argv[2], "wb"))) { //error check and open file
        printf("error: Cannot open output file for writing.\n");
        return;
    }
    char line[1096];

    float lpf[3][3] = {
                    {1,2,1},
                    {2,4,2},
                    {1,2,1}};//5x5 (part 1 - low pass filter divided by 81)
    //H block
    float H[5][5] = {{1, 2, 3, 2, 1},
            {2, 4, 6, 4, 2},
            {3, 6, 9, 6, 3},
            {2, 4, 6, 4, 2},
            {1, 2, 3, 2, 1}};
    float S1[3][3] = {{1, 0, -1},
                        {2, 0, -2},
                        {1, 0, -1}};
    float S2[3][3] = {{-1, -2, -1},
                        {0, 0, 0},
                        {1, 2, 1}};
    for (int i = 0; i < 5; i++)
        for(int j = 0; j < 5; j++){
            H[i][j] = H[i][j] * (0.012345679); //  1/81
        }
    float h[5][5];
    float lpf_1[3][3];
    float s1[3][3];
    float s2[3][3];
    //part 1
    // for(int i = 0; i < 3; i++){
    //     for(int j = 0; j<3; j++){
    //         lpf[i][j] = lpf[i][j] * (0.03125); //for part 1
    //     }
    // }
    // rotate matrix
    for(int i = 0; i < 3; i++){
        for(int j = 0; j < 3; j++){
            // int temp = lpf[i][j];
            // lpf[i][j] = lpf[3-i-1][3-j-1];
            // lpf[3-i-1][3-j-1] = temp;

            // int temp = H[i][j];
            // H[i][j] = H[5-i-1][5-j-1];
            // H[5-i-1][5-j-1] = temp;
```

```
            int temp = S1[i][j];
            S1[i][j] = S1[3-i-1][3-j-1];
            S1[3-i-1][3-j-1] = temp;
            int temp1 = S2[i][j];
            S2[i][j] = S2[3-i-1][3-j-1];
            S2[3-i-1][3-j-1] = temp1;
        }
    }
    //fetch image header
    file_header h0, hout;
    for (int i = 0; i <= 21; i++) {
        if (fgets(line, sizeof(line), infile)) { //fetch one line at a time for header
            if (line[0] == '#') { //if a comment exists in the header
                continue;
            }
            if (i == 0) { //file type
                printf("in here\n");
                h0.in_type = "P5";
                continue;
            }
            if (i == 20) {
                for (int j = 0; j < 2; j++) {
                    char* delim = strtok(line, " ");
                    if (j == 0) {
                        h0.in_width = atoi(delim);
                        delim = strtok(NULL, " ");
                    }
                    else if (j == 1) {
                        h0.in_height = atoi(delim);
                        delim = strtok(NULL, " ");
                    }
                }
                continue;
            }
            if (i == 21) {
                h0.in_max = atoi(line);
                continue;
            }
        }
    }
    printf("\tfile type = %s\n", h0.in_type);
    printf("\theight = %d\n", h0.in_height);
    printf("\twidth = %d\n", h0.in_width);
    printf("\tMax num val in image = %d\n", h0.in_max);
    int Rh = 3;//160; //length of impulse response
    int Ch = 3;//165;
    int Rx = h0.in_width; // Width of input image (n)
    int Cx = h0.in_height; //Height of input image (m)
    int Ry = Rx + Rh - 1; //length of convolution result
    int Cy = Cx + Ch - 1;
    int Rz = Rx + 2*(Rh-1); //length with zero padding
    int Cz = Cx + 2*(Ch-1); //height with zero padding
    printf("Rh = %d, Rx = %d, Ry = %d, Rz = %d\n", Rh, Rx, Ry, Rz);

    unsigned char* x = (unsigned char*)calloc(sizeof(unsigned char), Rz*Cz);
    float* x1 = (float*)calloc(sizeof(float), Rz*Cz);
    float* x_sq = (float*)calloc(sizeof(float), Rz*Cz);
    float* y = (float*)calloc(sizeof(float), Ry*Cy);
    float* y_4 = (float*)calloc(sizeof(float), Ry*Cy);
    float* y1 = (float*)calloc(sizeof(float), Ry*Cy);
    float* y2 = (float*)calloc(sizeof(float), Ry*Cy);
    unsigned char* y0 = (unsigned char*)calloc(sizeof(unsigned char), Ry*Cy);
    unsigned char* y1c = (unsigned char*)calloc(sizeof(unsigned char), Ry*Cy);
    unsigned char* y2c = (unsigned char*)calloc(sizeof(unsigned char), Ry*Cy);
    // int a = Rz*Ch + Rh+1; //start Ch rows down to include zero padding
    int a = Ch-1;
    while (!feof(infile)){
        //read in each line with each pixel of size unsigned char (8 bytes)
        fread(x + (a*Cz + Ch-1), sizeof(unsigned char), Rx, infile);
        a++;
    }
    for(int k = 0; k < Rz*Cz; k++) //convert x to float for convolution
        x1[k] = (float)x[k];


    printf("done reading\n");
    float tmp0,tmp1;
    int i;
```

```
    printf("Start Convolution\n");
    //for part 1, 2, 3
    for (int m = 0; m < Ry; m++){ //go to next row
        for (int n = 0; n < Cy; n++){ //go through columns of one row
            //convolution with impulse
            for (tmp0 = 0.0,tmp1 = 0.0,i=0; i<Rh; i++){ //convolves for one point of n inside row m
                for (int j = 0; j<Ch; j++){
                    // tmp0 += h[i][j] * x1[(m+i)*Cz + (n+j)]; //for convolution
                    // tmp0 += lpf[i][j] * x1[(m+i)*Cz + (n+j)]; //for part 1
                    tmp0 += S1[i][j] * x1[(m+i)*Cz + (n+j)]; //for part 3
                    tmp1 += S2[i][j] * x1[(m+i)*Cz + (n+j)];
                }
            }
            y1[m*Cy+n] = tmp0;
            y2[m*Cy+n] = tmp1;
            // y[m*Cy+n] = tmp0;
        }
    }
    // for(int k = 0; k < Ry*Cy; k++){
    //     y1c[k] = (unsigned char)y1[k];
    //     y2c[k] = (unsigned char)y2[k];
    // }
    for(int m = 0; m < Ry; m++){
        for(int n = 0; n < Cy; n++){
            y[m*Cy+n] = sqrt( (y1[m*Cy+n]*y1[m*Cy+n]) + (y2[m*Cy+n]*y2[m*Cy+n]) );
        }
    }
    for(int k = 0; k < Ry*Cy; k++){//convert output to unsigned char
        y0[k] = (unsigned char)y[k];
    }

    for(int i = 0; i < Ry; i++){ //check if values are higher or lower than bounds provided by file
        for(int j = 0; j < Cy; j++){
            if(y0[i*Cy+j] > h0.in_max){
                y0[i*Cy + j] = h0.in_max;
            }
            else if(y0[i*Cy + j] < 0){
                y0[i*Cy + j] = 0;
            }
        }
    }
    printf("finish and write\n");
    // fprintf(outfile, "%s\n%d\n%d\n%d\n", h0.in_type, Rz, Cz, h0.in_max);
    // fwrite(x_sq, sizeof(unsigned char), Rz*Cz, outfile);
    fprintf(outfile, "%s\n%d\n%d\n%d\n", h0.in_type, Ry, Cy, h0.in_max);
    fwrite(y0, sizeof(unsigned char), Ry*Cy, outfile);
    fclose(infile);
    fclose(outfile);
    printf("done\n");
}
```

Code for part 4

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
//#include "image2.txt"
//*********************************************************************************************//
//  Nik Jensen
//  Digital Signal Image Processing
//  A02250195
//*********************************************************************************************//
//pgm file format (MxN matrix = row X col)
//P5 - file type
//Width (n)
//Height (m)
//Max num val in image
//cycles from top row down to bottom row (of size height) while each row is of size width
//*********************************************************************************************//
typedef struct{
    char *in_type;
    int in_width;
    int in_height;
    int in_max;
} file_header;

void main(int argc, char** argv) {
    FILE* infile;
    FILE* outfile;
    FILE* impfile;
    unsigned char buffer[1096];
    if (NULL == (infile = fopen(argv[1], "rb"))) { //error check and open file
        printf("error: Cannot open image.pgm file for input.\n");
        return;
    }
    if (NULL == (outfile = fopen(argv[2], "wb"))) { //error check and open file
        printf("error: Cannot open output file for writing.\n");
        return;
    }
    if (NULL == (impfile = fopen(argv[3], "rb"))) { //error check and open file
        printf("error: Cannot open impulse file for reading.\n");
        return;
    }
    char line[1096];
    char line1[1096];
    //fetch image header
    file_header h0, h_1;
    for (int i = 0; i <= 21; i++) {
        if (fgets(line, sizeof(line), infile)) { //fetch one line at a time for header
            if (line[0] == '#') { //if a comment exists in the header
                continue;
            }
            if (i == 0) { //file type
                printf("in here\n");
                h0.in_type = "P5";
                continue;
            }
            if (i == 20) {
                for (int j = 0; j < 2; j++) {
                    char* delim = strtok(line, " ");
                    if (j == 0) {
                        h0.in_width = atoi(delim);
                        delim = strtok(NULL, " ");
                    }
                    else if (j == 1) {
                        h0.in_height = atoi(delim);
                        delim = strtok(NULL, " ");
                    }
                }
                continue;
            }
```

```
            if (i == 21) {
                h0.in_max = atoi(line);
                continue;
            }
        }
    }
    for (int i = 0; i <= 21; i++) {
        if (fgets(line1, sizeof(line1), impfile)) { //fetch one line at a time for header
            if (line1[0] == '#') { //if a comment exists in the header
                continue;
            }
            if (i == 0) { //file type
                printf("in here\n");
                h_1.in_type = "P5";
                continue;
            }
            if (i == 20) {
                for (int j = 0; j < 2; j++) {
                    char* delim = strtok(line1, " ");
                    if (j == 0) {
                        h_1.in_width = atoi(delim);
                        delim = strtok(NULL, " ");
                    }
                    else if (j == 1) {
                        h_1.in_height = atoi(delim);
                        delim = strtok(NULL, " ");
                    }
                }
                continue;
            }
            if (i == 21) {
                h_1.in_max = atoi(line1);
                continue;
            }
        }
    }
    printf("\tfile type = %s\n", h_1.in_type);
    printf("\theight = %d\n", h_1.in_height);
    printf("\twidth = %d\n", h_1.in_width);
    printf("\tMax num val in image = %d\n", h_1.in_max);
    int Rh = 160; //length of impulse response
    int Ch = 165;
    int Rx = h0.in_width; // Width of input image (n)
    int Cx = h0.in_height; //Height of input image (m)
    int Ry = Rx + Rh - 1; //length of convolution result
    int Cy = Cx + Ch - 1;
    int Rz = Rx + 2*(Rh-1); //length with zero padding
    int Cz = Cx + 2*(Ch-1); //height with zero padding
    printf("Rh = %d, Rx = %d, Ry = %d, Rz = %d\n", Rh, Rx, Ry, Rz);

    unsigned char* x = (unsigned char*)calloc(sizeof(unsigned char), Rz*Cz);
    unsigned char* h = (unsigned char*)calloc(sizeof(unsigned char), Rh*Ch);
    float h1[165][160];
    // float h2[160][160];
    float* h2 = (float*)calloc(sizeof(float), Rh*Ch);
    float* x1 = (float*)calloc(sizeof(float), Rz*Cz);
    float* x_sq = (float*)calloc(sizeof(float), Rz*Cz);
    float* y = (float*)calloc(sizeof(float), Ry*Cy);
    float* y1 = (float*)calloc(sizeof(float), Ry*Cy);
    float* y2 = (float*)calloc(sizeof(float), Ry*Cy);
    unsigned char* y0 = (unsigned char*)calloc(sizeof(unsigned char), Ry*Cy);
    // int a = Rz*Ch + Rh+1; //start Ch rows down to include zero padding
    int a = Ch-1;
    while (!feof(infile)){
        //read in each line with each pixel of size unsigned char (8 bytes)
        fread(x + (a*Cz + Ch-1), sizeof(unsigned char), Rx, infile); //input file
        a++;
    }
    while (!feof(impfile)){
```

```
        //read in each line with each pixel of size unsigned char (8 bytes)
        fread(h, sizeof(unsigned char), Rh, impfile); //impulse
    }
    for(int k = 0; k < Rz*Cz; k++){ //convert x to float for convolution and square for part 4
        x1[k] = (float)(x[k]*x[k]);//have to square each pixel that is read in. Possibly read in, square,
then convert to float
    }
    printf("h[10] = %u\n",h[10]);
    for(int m = 0; m < Rh; m++){
        for(int n = 0; n < Ch; n++){
            h1[m][n] = 1.0;
        }
    }
    for(int k = 0; k < Rh*Ch; k++){
        h2[k] = (float)h[k];
    }

    printf("h2[10] = %f\n",h2[10]);
    printf("done reading\n");
    float tmp0,tmp1;
    int i;
    printf("Start Convolution\n");
    //for part 1, 2, 3
    for (int m = 0; m < Ry; m++){ //go to next row
        printf("m = %d\n", m);
        for (int n = 0; n < Cy; n++){ //go through columns of one row
        printf("n = %d\n", n);
            //convolution with impulse
            for (tmp0 = 0.0,tmp1 = 0.0,i=0; i<Rh; i++){ //convolves for one point of n inside row m
                for (int j = 0; j<Ch; j++){
                    // tmp0 += h[i][j] * x1[(m+i)*Cz + (n+j)]; //for convolution
                    tmp0 += h1[i][j] * x1[(m+i)*Cz + (n+j)];
                    tmp1 += h2[m*Ch + n] * x1[(m+i)*Cz + (n+j)];
                }
            }
            y1[m*Cy+n] = tmp0;
            y2[m*Cy+n] = tmp1;
            y[m*Cy+n] = ((y2[m*Cy+n])/(y2[m*Cy+n]));
        }
    }
    printf("finished convolution\n");
    for(int k = 0; k < Ry*Cy; k++){//convert output to unsigned char
        y0[k] = (unsigned char)y[k];
    }

    for(int i = 0; i < Ry; i++){ //check if values are higher or lower than bounds provided by file
        for(int j = 0; j < Cy; j++){
            if(y0[i*Cy+j] > h0.in_max){
                y0[i*Cy + j] = h0.in_max;
            }
            else if(y0[i*Cy + j] < 0){
                y0[i*Cy + j] = 0;
            }
        }
    }
    printf("finish and write\n");
    // fprintf(outfile, "%s\n%d\n%d\n%d\n", h0.in_type, Rz, Cz, h0.in_max);
    // fwrite(x_sq, sizeof(unsigned char), Rz*Cz, outfile);
    fprintf(outfile, "%s\n%d\n%d\n%d\n", h0.in_type, Ry, Cy, h_1.in_max);
    fwrite(y0, sizeof(unsigned char), Ry*Cy, outfile);
    fclose(infile);
    fclose(outfile);
    fclose(impfile);
    printf("done\n");
}
```