Homework 1 report

Heng Zhang

0029109755

# Introduction

The two source images used in this homework are Figure 1 (1280\* 720) and Figure 2 (1960 \* 1080). They are “img0.jpg” and “img1.jpg” in the submission folder.

Figure 1280 \* 720 Figure 2 1960 \* 1080

There are three source code files in the submission folder: conv.py, main.py, and main.c. The python files are used in part A – C while the c file is used in part D.

## PartA

In total, there are 12 output plots where 6 plots originate from img0.jpg and the other 6 from img1.jpg by applying 6 different configurations to each plots.

### Config 1 (Task 1 kernel 1)

Figure 3 and 4 are generated by applying kernel 1 in task 1. Kernel 1 is a horizontal edge detector. Therefore, the horizontal edges are highlighted in these figures. The output images size in both height and width are decreased by 2 because the kernel has length of 3 and the stride is only 1 by the formula: (original\_size – kernel length) / stride + 1

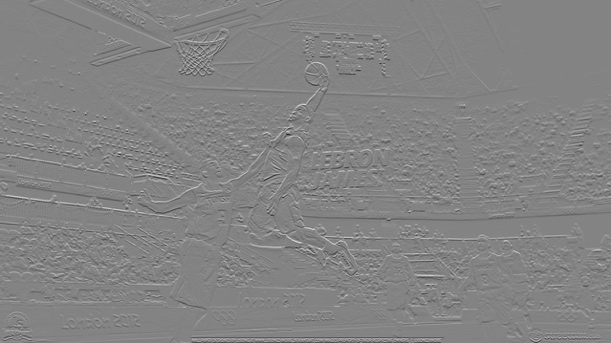
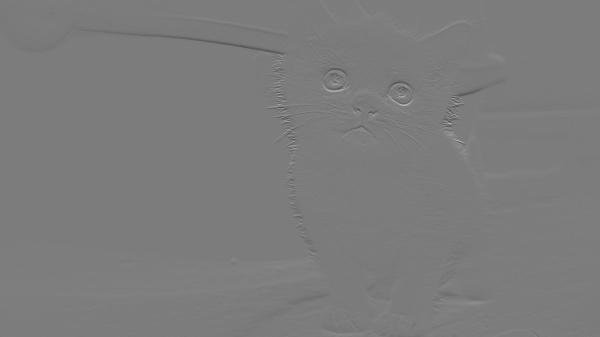
 

Figure 3 img0 by Task 1 kernel 1 Figure img1 by Task 1 kernel 1

### Config 2 (Task 2 kernel 4)

The effect in Config 2 compared with Config 1 is the size of the kernel. Kernel 4 is 5 by 5 which gives stronger horizontal edge detecting effect. The Figure 5 and 6 compared with Figure 3 and 4 have thicker horizontal edges

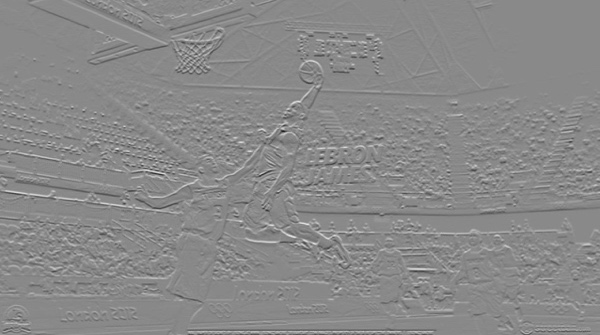
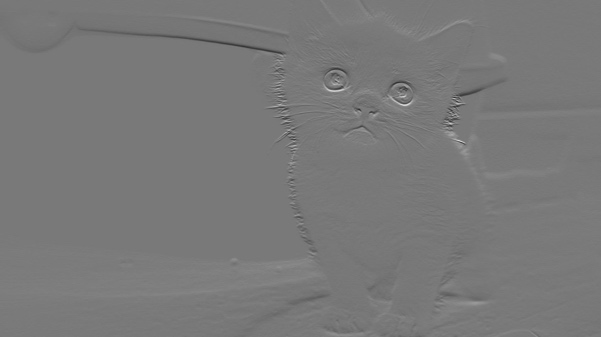
 

Figure 5 img0 by Task 2 kernel 4 Figure img1 by Task 2 kernel 4

### Config 3 (Task 2 kernel 5)

Kernel 5 is a vertical edge detector with size of 5 by 5. Therefore in Figure 7 and 8, we see thick vertical edges are detected.

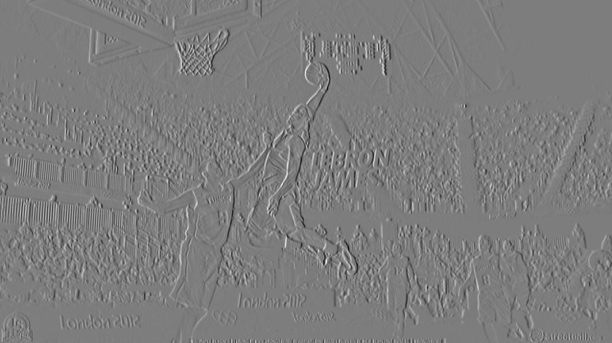
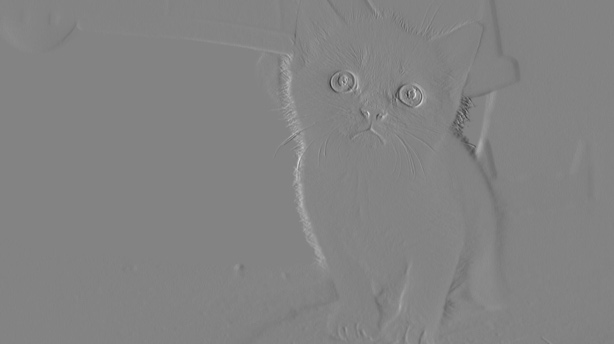
 

Figure 7 img0 by Task 2 kernel 5 Figure img1 by Task 2 kernel 5

### Config 4 (Task 3 kernel 1 stride 2)

Compared with Config 1, the change is the larger stride. By (original\_size – kernel length) / stride + 1, the output figure will have smaller height and weight by roughly half of the the plots in Config 1.

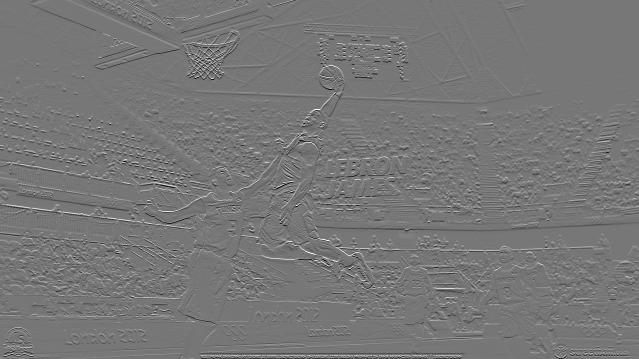
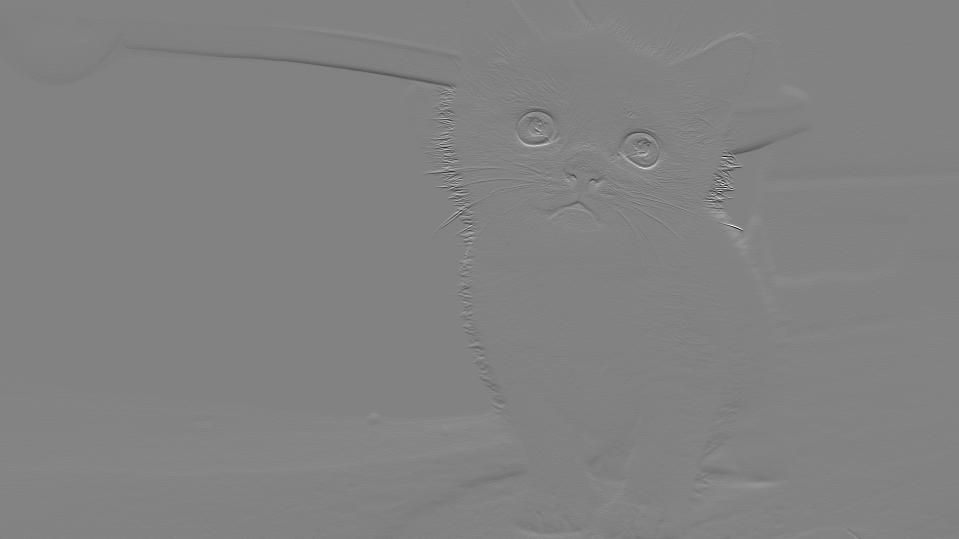
 

Figure img0 by Task 3 kernel 1 Figure img1 by Task 3 kernel 1

### Config 5 (Task 3 kernel 2 stride 2)

Kernel 2 is a vertical edge detector. We see the Figures 11 and 12 are showing clear vertical edges.

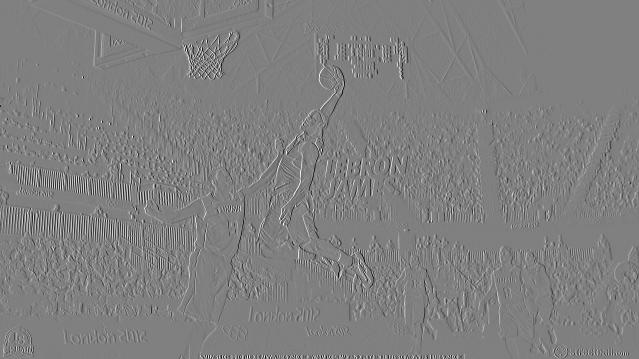
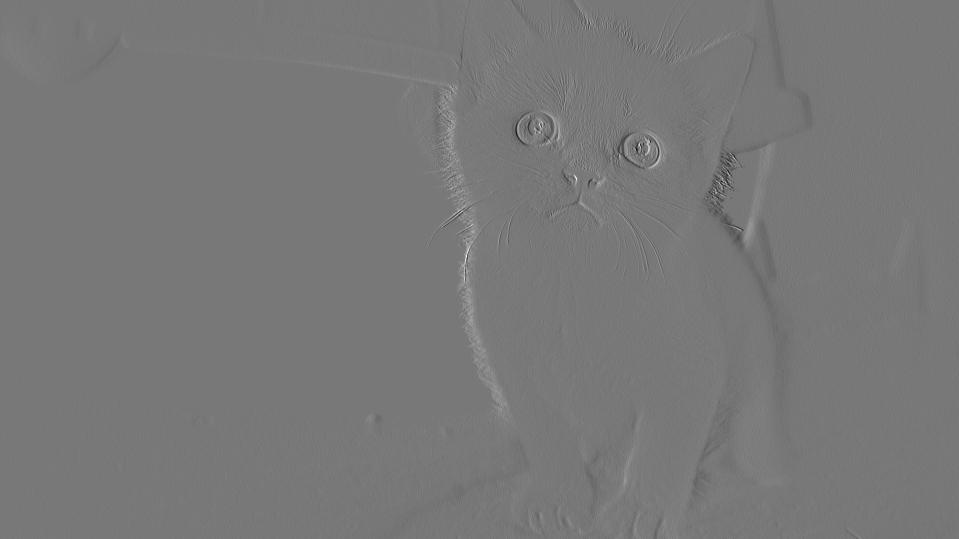
 

Figure img0 by Task 3 kernel 2 Figure img1 by Task 3 kernel 2

### Config 6 (Task 3 kernel 3 stride 2)

Kernel 3 is an average detector because it has all 1s in all elements which means the convolution will be just an average of all three RGB channels. The resulting plots are Figure 13 and 14.

Figure img0 by Task 3 kernel 3 Figure img1 by Task 3 kernel 3

## Part B

This part is comparing the computation efficiency with respect to different number of output channels for both img0 and img1. The number of output channels is calculated by 2i where i is the x axis of Figure 15 and 16. The computing time is in seconds. Clearly the computation time is exponentially increasing with i.

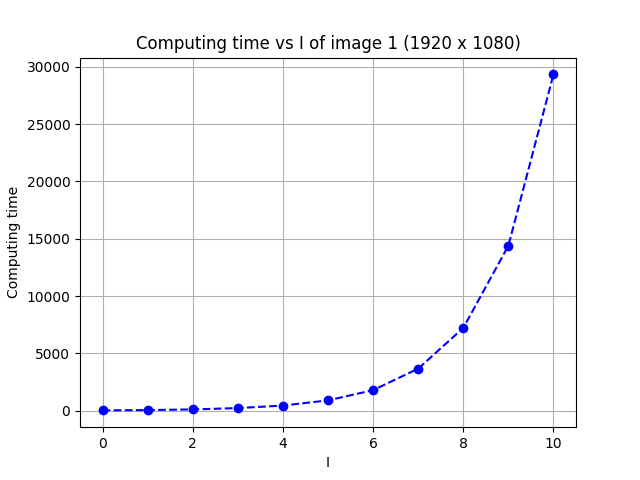
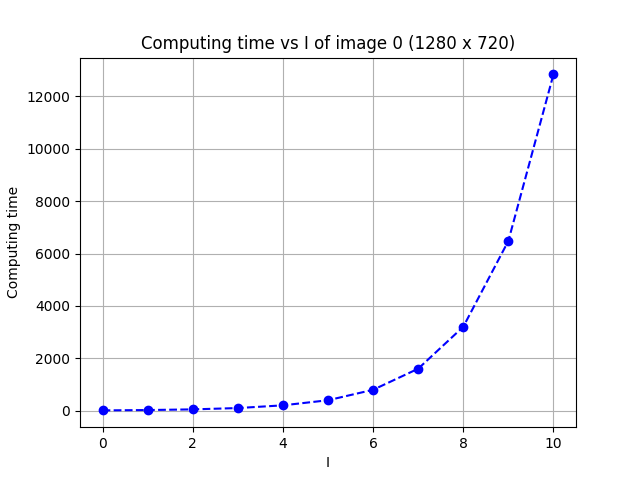


Figure 15 img0 computation efficiency Figure 16 img1 computation efficiency

## Part C

This part compares the number of operations with respect to the kernel size. It is clear that the relation is similar to quadratic shape, which is the in correspondence to the total size of kernel because the kernel is 2D and the total elements is quadratic to the length.

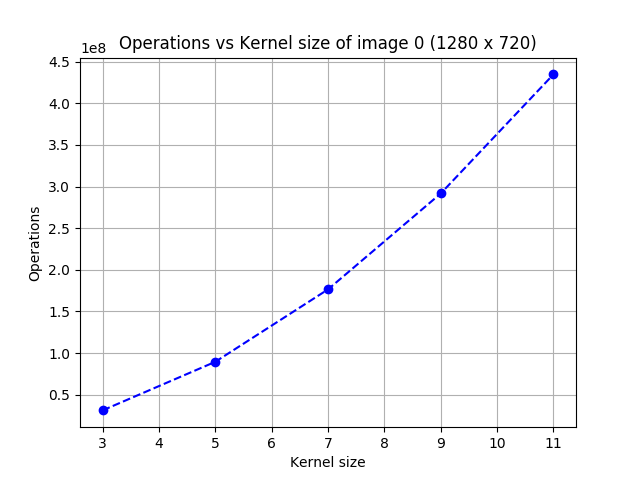
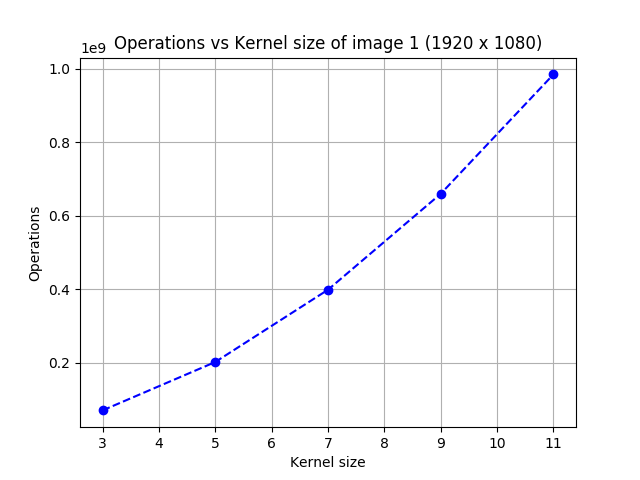
 

Figure 17 img0 operations with kernel size Figure 18 img1 operations with kernel size

## Part D

This part is the replication of part B but in C programming language. The running time is still quadratic in terms of the size of output channels but the absolute running time in C compared with Python is much smaller so the C implementation is more efficient.

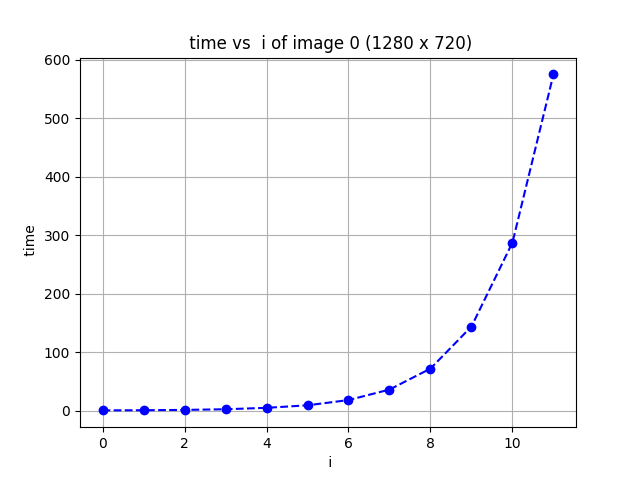
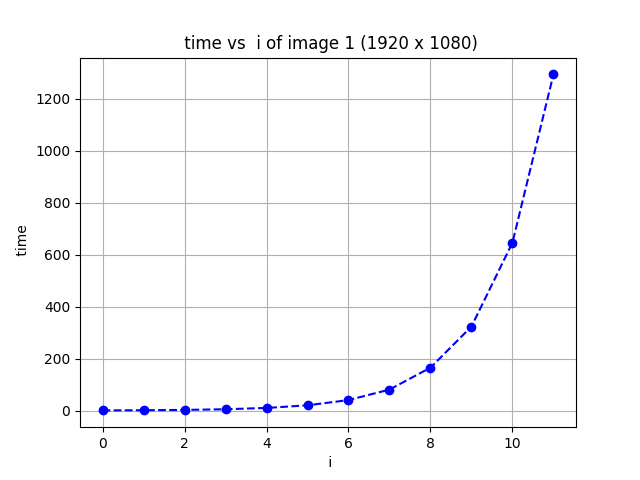
 

Figure 19 img0 computation efficiency in c Figure 20 img1 computation efficiency in c

# Appendix

1. For replicating results for part ABC:

Run as:

python3 main.py [partA, partB, partC]

1. For replicating results for part D:

Compile it with:

gcc main.c -lm -o main

run as

./main

1. For generating all the plots:

python3 plot.py [partA\_result.csv, partB\_result.csv, partC\_result.csv, partC\_result.csv]

conv.py

1. **import** torch

4. **class** Conv2D:
5. **def** \_\_init\_\_(self, in\_channel, o\_channel, kernel\_size, stride, mode='known'):
6. self.in\_channel = in\_channel
7. self.o\_channel = o\_channel
8. self.kernel\_size = kernel\_size
9. self.stride = stride
10. self.mode = mode
11. # predefined kernels
12. self.k1 = torch.FloatTensor([[-1, -1, -1], [0, 0, 0], [1, 1, 1]])
13. self.k2 = torch.FloatTensor([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]])
15. self.k3 = torch.FloatTensor([[1, 1, 1], [1, 1, 1], [1, 1, 1]])
16. self.k4 = torch.FloatTensor([[-1, -1, -1, -1, -1], [-1, -1, -1, -1, -1], [0, 0, 0, 0, 0], [1, 1, 1, 1, 1], [1, 1, 1, 1, 1]])
17. self.k5 = torch.FloatTensor([[-1, -1, 0, 1, 1], [-1, -1, 0, 1, 1],[-1, -1, 0, 1, 1],[-1, -1, 0, 1, 1],[-1, -1, 0, 1, 1]])
19. **def** forward(self, img):
20. # img is a 3D FloatTensor
21. # output is a tuple of (# of operations, 3D FloatTensor impage)
22. # the 3D FloatTensor image is a greyscale image with 0s on the third domain.
24. channels, height, width = img.size()
25. #print("img height:", height, "img width", width, "img stride", self.stride)
26. out\_height = int((( height - self.kernel\_size ) / self.stride + 1 ))
27. out\_width = int((( width - self.kernel\_size ) / self.stride + 1 ))
28. out\_img = torch.zeros(self.o\_channel, out\_height, out\_width)
29. #print("out\_height:", out\_height, "out\_width", out\_width)
30. mul\_cnt = 0
31. add\_cnt = 0
32. **if** self.mode == 'known':
33. # task 1
34. **if** self.o\_channel == 1:
35. kernel = torch.stack([self.k1 **for** i **in** range(self.in\_channel)])
36. kernels = [kernel]
37. **elif** self.o\_channel == 2:
38. # task 2
39. kernel1 = torch.stack([self.k4 **for** i **in** range(self.in\_channel)])
40. kernel2 = torch.stack([self.k5 **for** i **in** range(self.in\_channel)])
41. kernels = [kernel1, kernel2]
43. **elif** self.o\_channel == 3:
44. # task 3
45. kernel1 = torch.stack([self.k1 **for** i **in** range(self.in\_channel)])
46. kernel2 = torch.stack([self.k2 **for** i **in** range(self.in\_channel)])
47. kernel3 = torch.stack([self.k3 **for** i **in** range(self.in\_channel)])
48. kernels = [kernel1, kernel2, kernel3]
50. **for** ind **in** range(self.o\_channel):
51. **for** row **in** range(out\_height):
52. **for** col **in** range(out\_width):
53. temp\_tensor = torch.mul(kernels[ind], img[:, row \* self.stride : row \* self.stride + self.kernel\_size, col \* self.stride : col \* self.stride + self.kernel\_size])
54. out\_img[ind, row, col] = temp\_tensor.sum()
55. mul\_cnt += 1
56. add\_cnt += 1
57. # change the output image to 3D float tensor
58. ops = mul\_cnt \* self.kernel\_size \*\*2 + add\_cnt \* ( self.kernel\_size \*\*2 - 1)
59. **return** ops, out\_img
60. **elif** self.mode == 'rand':
61. # for part B
62. kernels = []
63. **for** i **in** range(self.o\_channel):
64. k = torch.stack([torch.rand(self.kernel\_size, self.kernel\_size) **for** i **in** range(self.in\_channel)])
65. kernels.append(k)
67. **for** ind **in** range(self.o\_channel):
68. **for** row **in** range(out\_height):
69. **for** col **in** range(out\_width):
70. temp\_tensor = torch.mul(kernels[ind], img[:, row : row + self.kernel\_size, col : col + self.kernel\_size])
71. out\_img[ind, row, col] = temp\_tensor.sum()
72. mul\_cnt += 1
73. add\_cnt += 1
74. # change the output image to 3D float tensor
75. ops = mul\_cnt \* self.kernel\_size \*\*2 + add\_cnt \* ( self.kernel\_size \*\*2 - 1)
76. **return** ops, out\_img
78. **else**:
79. **print**("unknown mode: " + self.mode)
80. exit(1)

main.py

1. **from** conv **import** Conv2D
2. **from** PIL **import** Image
3. **from** torchvision.transforms **import** Compose, ToTensor
4. **from** torchvision.utils **import** save\_image
5. **import** torch
6. **from** sys **import** argv
7. **from** time **import** time
8. **import** csv
9. **import** numpy as np

12. transform = Compose([ToTensor()])
13. # transform a image to tensor (channel, height, width)
14. img1 = Image.open("img1.jpg")
15. img0 = Image.open('img0.jpg')
16. img\_tensors = [transform(img0), transform(img1)]
17. **if** len(argv) != 2:
18. **print**("Usage: python main.py part[A, B, C]")
19. exit(1)
21. tasks = [[3,1,3,1], [3,2,5,1], [3,3,3,2]]
22. **if** argv[1] == 'partA':
23. # Part A
24. **for** tsk\_id **in** range(len(tasks)):
25. task = tasks[tsk\_id]
26. **print**('Part A Task ' + str(tsk\_id + 1))
28. **for** img\_id **in** range(len(img\_tensors)):
30. **print**("Image ", img\_id, "size: ", img\_tensors[img\_id].size())
31. conv = Conv2D(task[0],task[1],task[2],task[3],)
32. ops, output\_img = conv.forward(img\_tensors[img\_id])
33. **print**('Total operation', ops, ', output tensor size:', output\_img.size())
34. num\_channels = output\_img.size()[0]
35. **if** num\_channels == 1:
36. # task 1
37. file\_name = "image" + str(img\_id) + "/plt\_" + str(img\_id) + "\_partA\_task" + str(tsk\_id + 1) + "\_k1.jpg"
38. **print**("Save to", file\_name, "\n")
39. save\_image(output\_img, file\_name, normalize=True)
41. **elif** num\_channels == 2:
42. # task 2
43. **for** i **in** range(num\_channels):
44. file\_name = "image" + str(img\_id) + "/plt\_" + str(img\_id) + "\_partA\_task" + str(tsk\_id + 1) + "\_k" + str(i + 4) +".jpg"
45. **print**("Save to", file\_name, "\n")
46. save\_image(output\_img[i, :, :], file\_name, normalize=True)
47. **else**:
48. # task 3 with 3 o channels
49. **for** i **in** range(num\_channels):
50. file\_name = "image" + str(img\_id) + "/plt\_" + str(img\_id) + "\_partA\_task" + str(tsk\_id + 1) + "\_k" + str(i + 1) +".jpg"
51. **print**("Save to", file\_name, "\n")
52. save\_image(output\_img[i, :, :], file\_name, normalize=True)
53. **elif** argv[1] == 'partB':
54. **print**("Part B")
55. task = tasks[0]
57. with open('partB\_result.csv', 'w') as out:
58. csv\_out = csv.writer(out)
59. csv\_out.writerow(['image', 'i', 'computing time'])
60. **for** img\_id **in** range(len(img\_tensors)):
61. **print**("Image " + str(img\_id))
62. **print**("Image, i, Time")
63. **for** i **in** range(11):
64. o\_c = 2\*\*i
65. s = time()
66. conv = Conv2D(task[0], o\_c, task[2], task[3], 'rand')
67. ops, output\_img = conv.forward(img\_tensors[img\_id])
68. e = time()
69. row = (img\_id, i, e-s)
70. **print**(row)
71. csv\_out.writerow(row)
73. **elif** argv[1] == 'partC':
74. **print**("part C")
75. task = tasks[1]
77. with open('partC\_result.csv', 'w') as out:
78. csv\_out = csv.writer(out)
79. csv\_out.writerow(['image', 'kernel size', 'operations'])
80. **for** img\_id **in** range(len(img\_tensors)):
81. **print**("Image " + str(img\_id))
82. **print**("Image, Kernel Size, Operations")
83. **for** ker\_size **in** range(3, 12, 2):
84. conv = Conv2D(task[0], task[1], ker\_size, task[3], 'rand')
85. ops, output\_img = conv.forward(img\_tensors[img\_id])
86. row = (img\_id, ker\_size, ops)
87. **print**(row)
88. csv\_out.writerow(row)
90. **else**:
91. **print**("Wrong argument", argv[1])
92. **print**("Abort!")

main.c

1. #include <stdlib.h>
2. #include <stdio.h>
3. #include <time.h>
4. #include <math.h>
6. void c\_conv(int in\_channel, int o\_channel, int kernel\_size, int stride, double\*\*\* img, int img\_height, int img\_width);
8. void init\_plot(double\*\*\* img, int height, int width, int in\_channel);

11. int main(int argc, char\*\* argv){
12. int in\_channel = 3;
13. int kernel\_size = 3;
14. int stride = 1;
16. // Create Image matrix
17. double \*\*\* img0 = (double \*\*\*) malloc(in\_channel \* sizeof(double\*\*));
18. double \*\*\* img1 = (double \*\*\*) malloc(in\_channel \* sizeof(double\*\*));
19. init\_plot(img0, 720, 1280, 3);
20. init\_plot(img1, 1080, 1920, 3);
21. char\* filename = "partD\_result.csv";
23. FILE \*fp;
24. fp = fopen(filename, "w+");
25. fprintf(fp, "Image, i, time\n");
27. **for** (int i = 0; i < 12; i++){
28. int o\_c = (int)pow(2, i);
29. clock\_t s = clock();
30. c\_conv(in\_channel, o\_c, kernel\_size, stride, img0, 720, 1280);
31. clock\_t e = clock();
32. double elapse =  (double) (e - s) / CLOCKS\_PER\_SEC;
33. fprintf(fp, "0,%d,%f\n", i, elapse);
34. printf("img0: 0,%d,%f\n", i, elapse);
36. // next **for** image 1
37. s = clock();
38. c\_conv(in\_channel, o\_c, kernel\_size, stride, img1, 1080, 1920);
39. e = clock();
40. elapse = (double) (e - s) / CLOCKS\_PER\_SEC;
41. fprintf(fp, "1,%d,%f\n", i, elapse);
42. printf("img1: 0,%d,%f\n", i, elapse);
43. }
44. fclose(fp);
45. **return** 0;
46. }
48. void c\_conv(int in\_channel, int o\_channel, int kernel\_size, int stride, double\*\*\* img, int img\_height, int img\_width){
49. //printf("morning\n");
50. double \*\*\*kernel = (double\*\*\*) malloc(in\_channel \* sizeof(double\*\*));
51. **for** (int i = 0; i < in\_channel; i++)
52. kernel[i] = (double\*\*) malloc(kernel\_size \* sizeof(double\*));
53. **for** (int i = 0; i < kernel\_size; i++){
54. **for** (int j = 0; j < kernel\_size; j ++)
55. kernel[i][j] = (double\*) malloc(kernel\_size\* sizeof(double));
56. }
57. **for** (int i = 0; i < in\_channel; i ++){
58. **for** (int j = 0; j < kernel\_size; j++){
59. **for** (int k = 0; k < kernel\_size; k++)
60. kernel[i][j][k] = (double) (rand() % 10);
61. }
62. }
63. int out\_height = (img\_height - kernel\_size) /stride + 1;
64. int out\_width = (img\_width - kernel\_size) / stride + 1;
65. //    printf("in\_channel = %d, o\_channel = %d, kernel\_size = %d,  stride = %d\n", in\_channel, o\_channel, kernel\_size, stride);
66. //printf("out\_height = %d, out\_width = %d\n", out\_height, out\_width);
67. **for** (int ker = 0; ker < o\_channel; ker ++){
68. **for** (int i = 0; i < in\_channel; i++){
69. **for** (int j = 0; j < out\_height; j++){
70. **for** (int k = 0; k < out\_width; k++){
71. double temp = 0;
72. **for** (int p = 0; p < in\_channel; p++){
73. **for** (int m = 0; m < kernel\_size; m++){
74. **for** (int n = 0; n < kernel\_size; n++){
75. temp += kernel[p][m][n] \* img[p][j + m][k + n];
76. //**if** (img\_height == 1080 && j >= out\_height - 360)
77. //printf("i = %d, j = %d, k = %d, p = %d, m = %d, n = %d\n", i, j, k, p, m, n);
78. }
79. }
80. }
82. }
83. }
84. }
85. }
86. };
88. void init\_plot(double \*\*\*img, int height, int width, int in\_channel){
89. **for** (int i = 0; i < in\_channel; i ++)
90. {
91. img[i] = (double\*\*) malloc(height \* sizeof(double\*));
92. }
94. **for** (int i = 0; i < in\_channel; i ++){
95. **for** (int j = 0; j < height; j ++)
96. img[i][j] = (double\*) malloc(width \* sizeof(double));
97. }
99. **for** (int i = 0; i < in\_channel; i++){
100. **for** (int j = 0; j < height; j ++){
101. **for** (int k = 0; k < width; k++)
102. img[i][j][k] = (i + k + j) % 255;
103. }
105. }
107. };