Homework-01

Xiaoyu Xiang (xiang43@purdue.edu)

Two input images are shown below in Fig1(1280x720) and 2(1920x1080):

Figure 1: 1280 x 720

Figure 2: 1920 x 1080

Platform and Packages used:

- Python 2.7, PyTorch 0.4.1
- PIL, matplotlib, time, numpy

Python File Details

- Run the script: python main.py
- conv.py defines the class Conv2D (in_channel, o_channel, kernel_size, stride, mode), the input should be integers
- to calculate specific image: [int, 3D FloatTensor] Conv2D.forward(input_image), the input image can be numpy ndarray or 3D FloatTensor
- the output 3D FloatTensor of Conv2D.forward is already normalized to 0-255, which can be easily saved as grayscale image

C File Details

- Run the script: main.c
- Here we define a function: long long c_conv(int in_channel, int o_channel, int kernel_size, int stride, uint8_t *input_image)
- In main.c, the image is randomly generated (1280 x 720 x 3)
- Kernel of size 3 x 3 is created randomly from [-1, 0, 1]
- Both the operation number and consumed time(s) are printed for each i

Part A

For task 1, each image generates an output image, as shown in Fig.3 and 4.

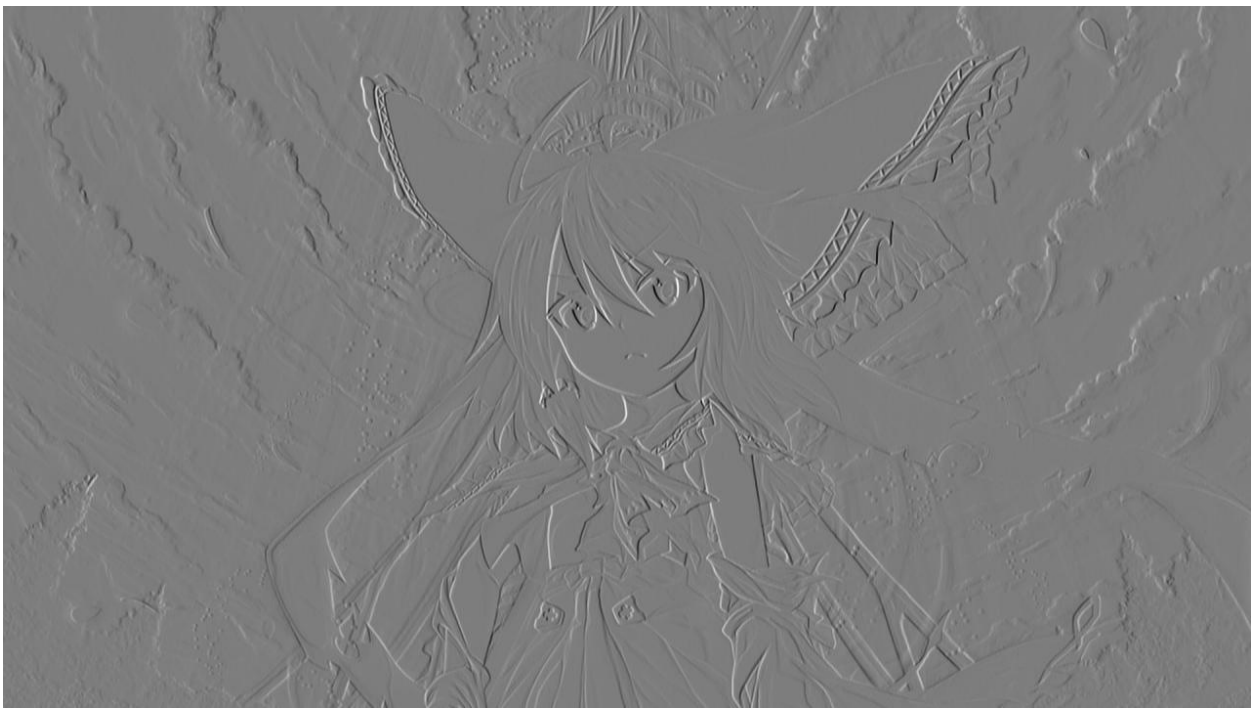

Figure 3: task-1-image-1

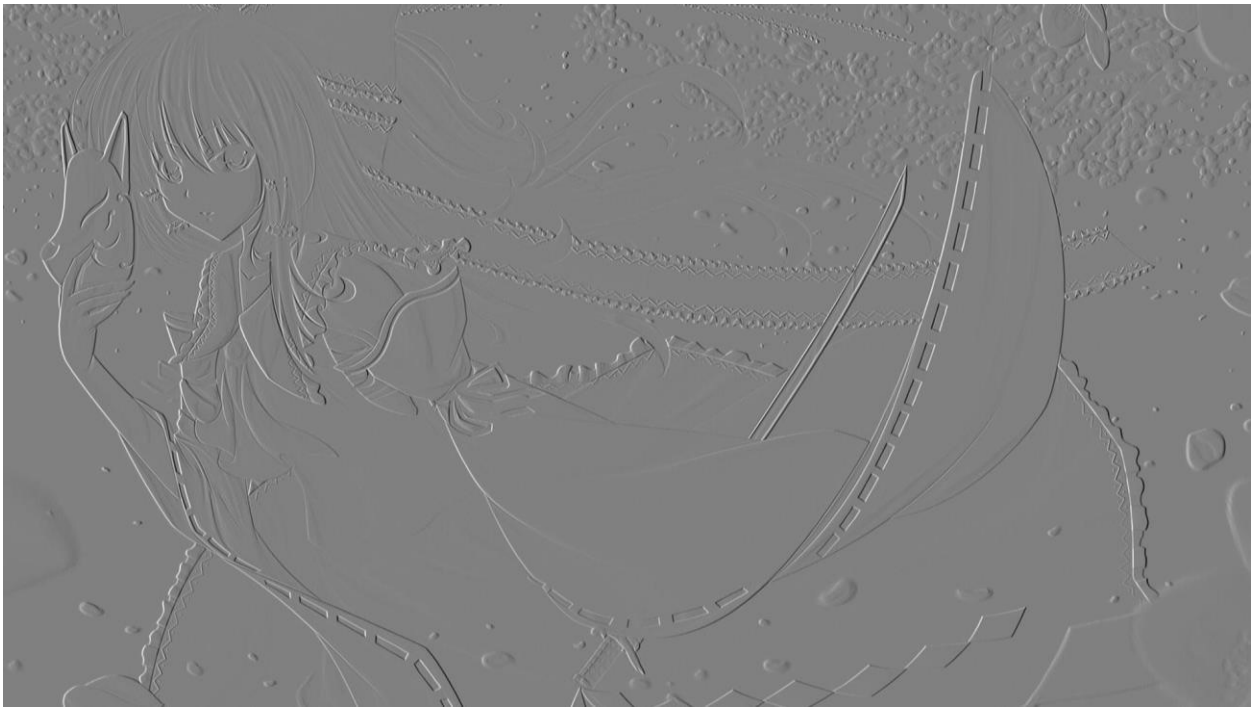

Figure 4: task-1-image-2

For task 2, each image generates 2 output images, as shown in Fig.5 and 6(1st input image), 7 and 8(2nd input image).

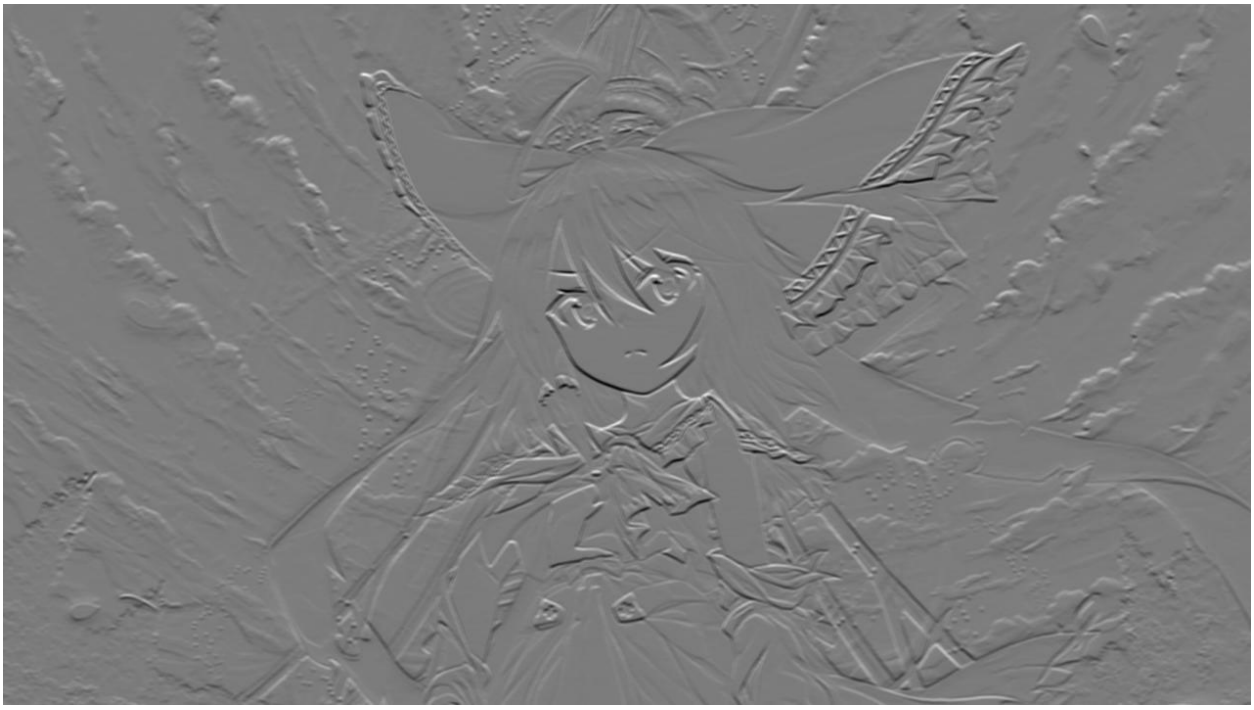

Figure 5: task-2-image-1-o_channel-1

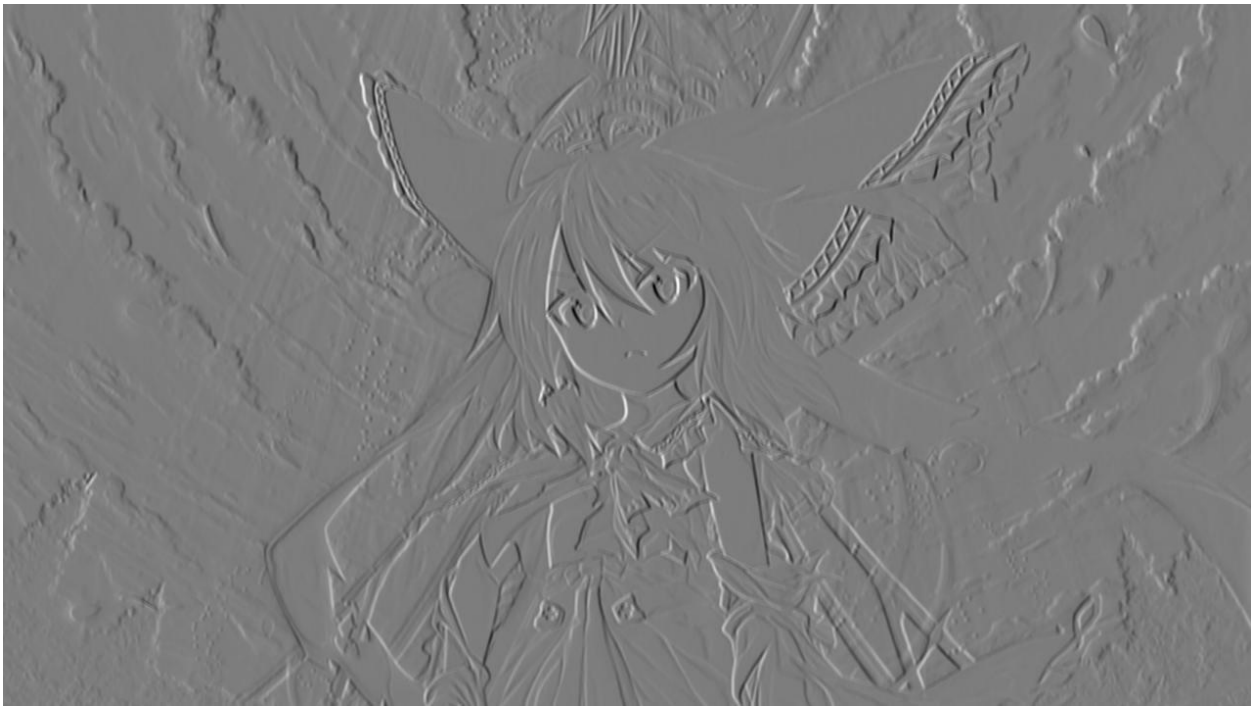

Figure 6: task-2-image-1-o_channel-2

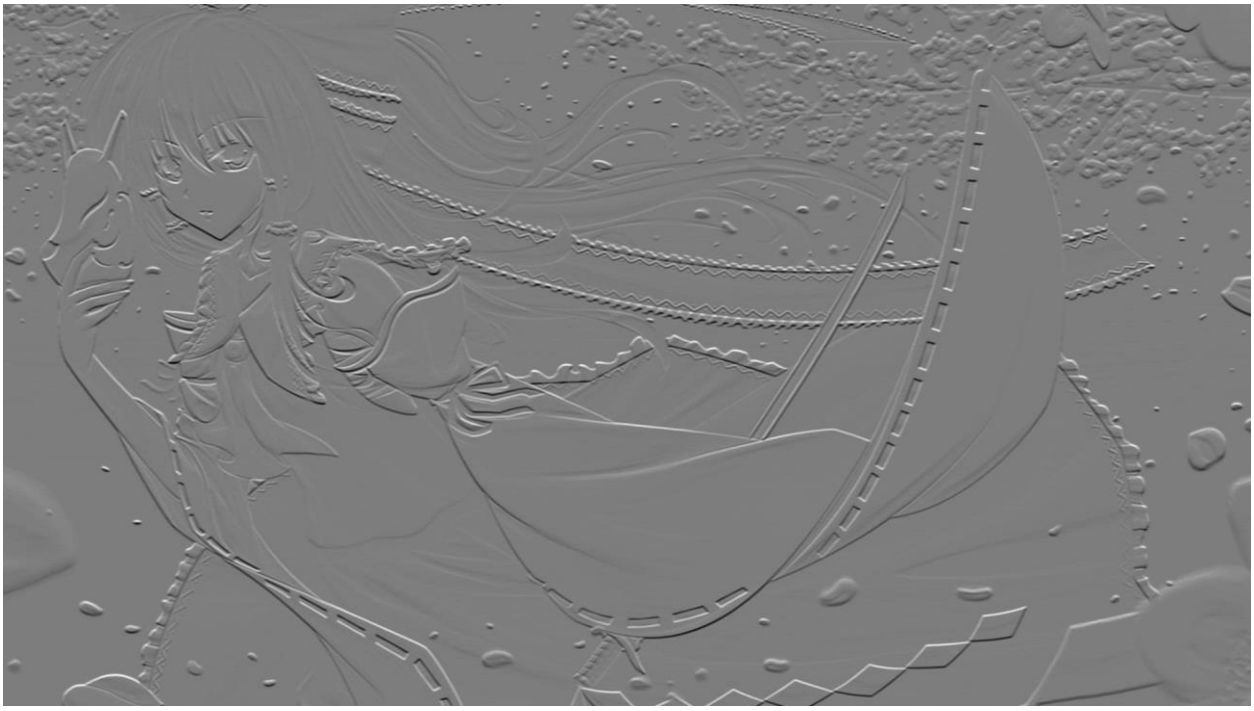

Figure 7: task-2-image-2-o_channel-1

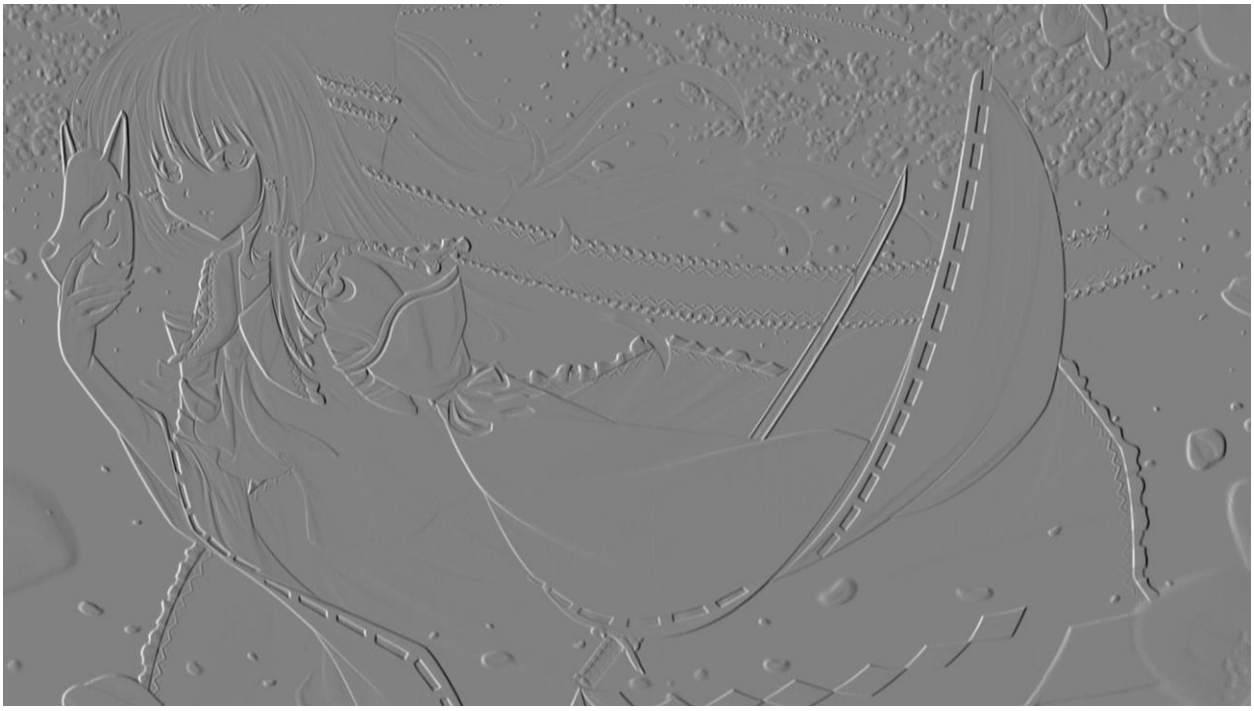

Figure 8: task-2-image-2-o_channel-2

For task 3, each image generates 3 output images, as shown in Fig.9, 10 and 11(1st input image), 12, 13 and 14(2nd input image).

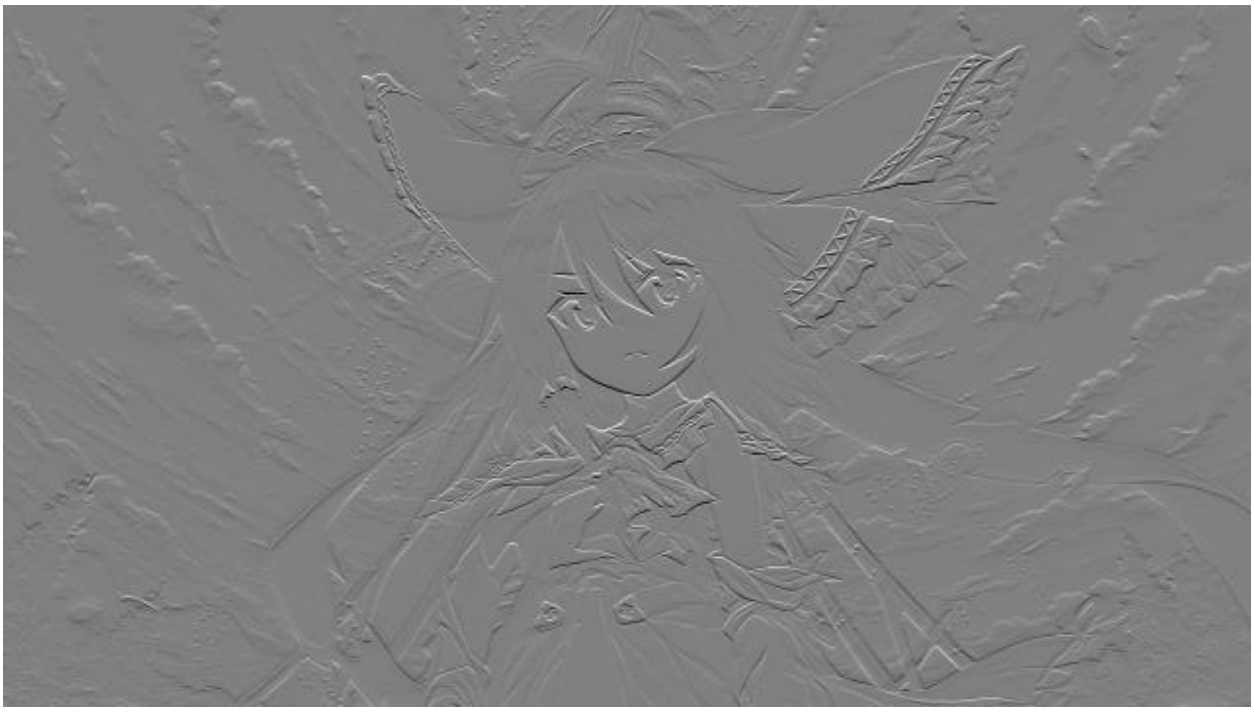

Figure 9: task-3-image-1-o_channel-1

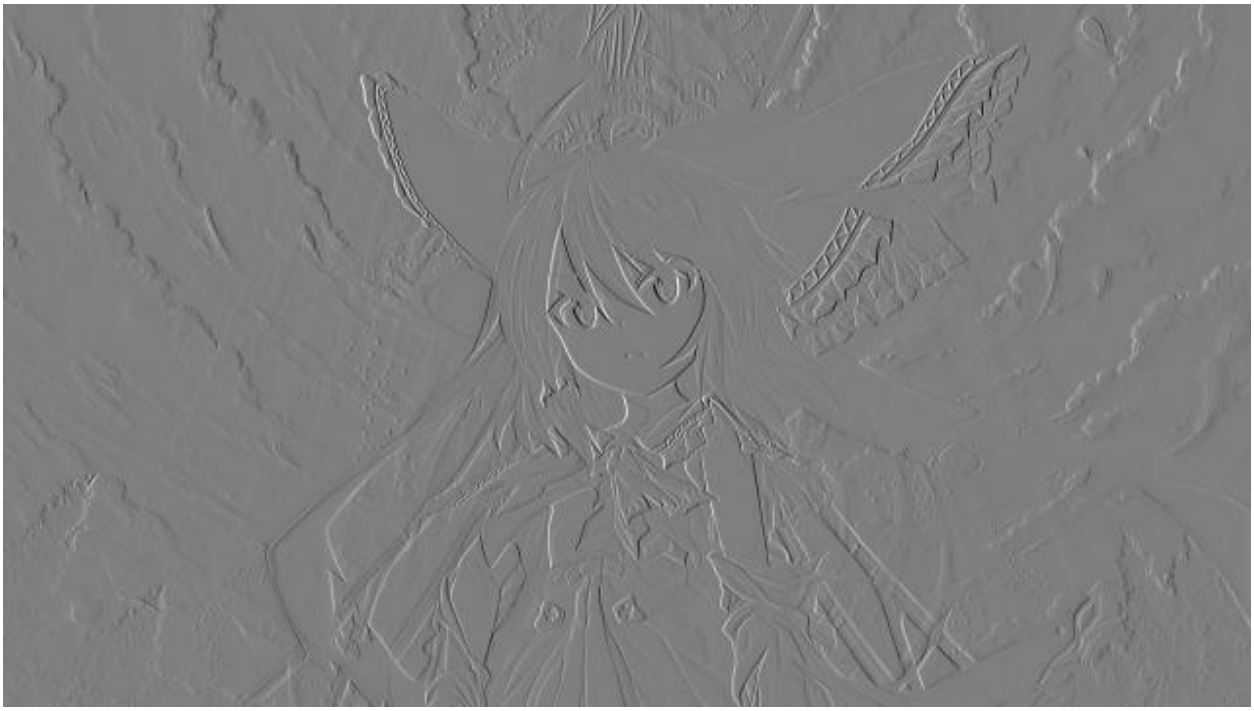

Figure 10: task-3-image-1-o_channel-2

Figure 11: task-3-image-1-o_channel-3

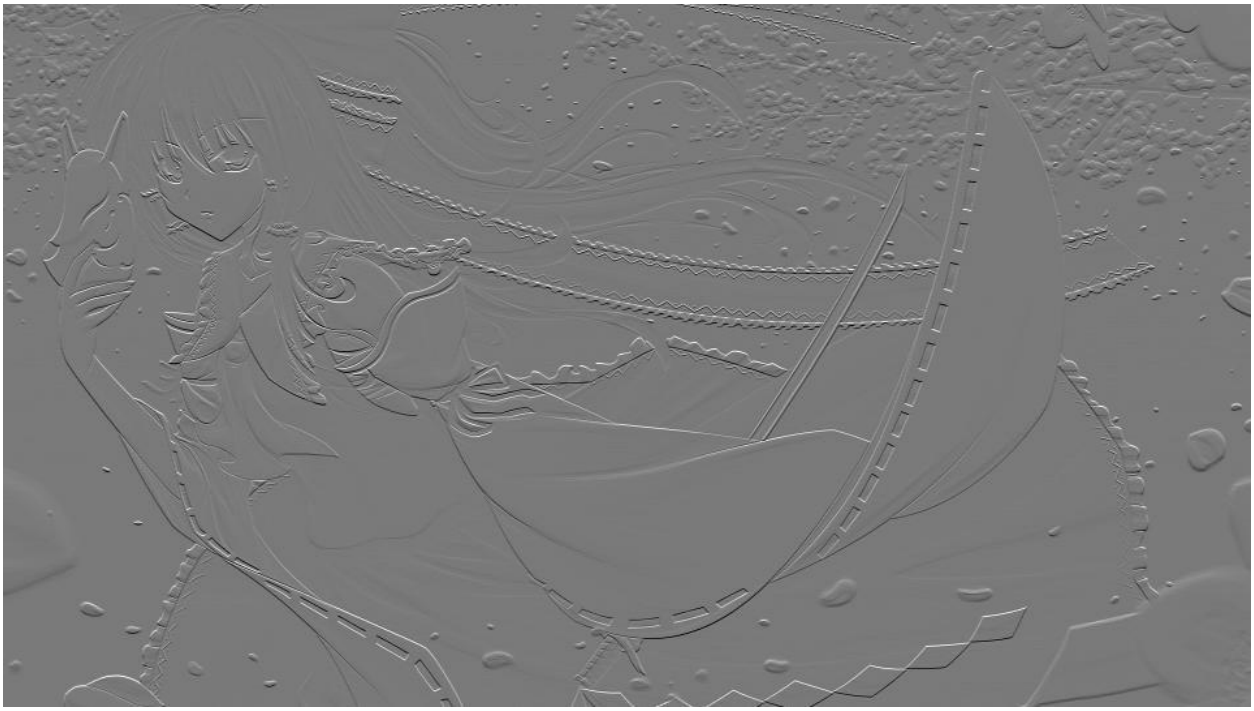

Figure 12: task-3-image-2-o_channel-1

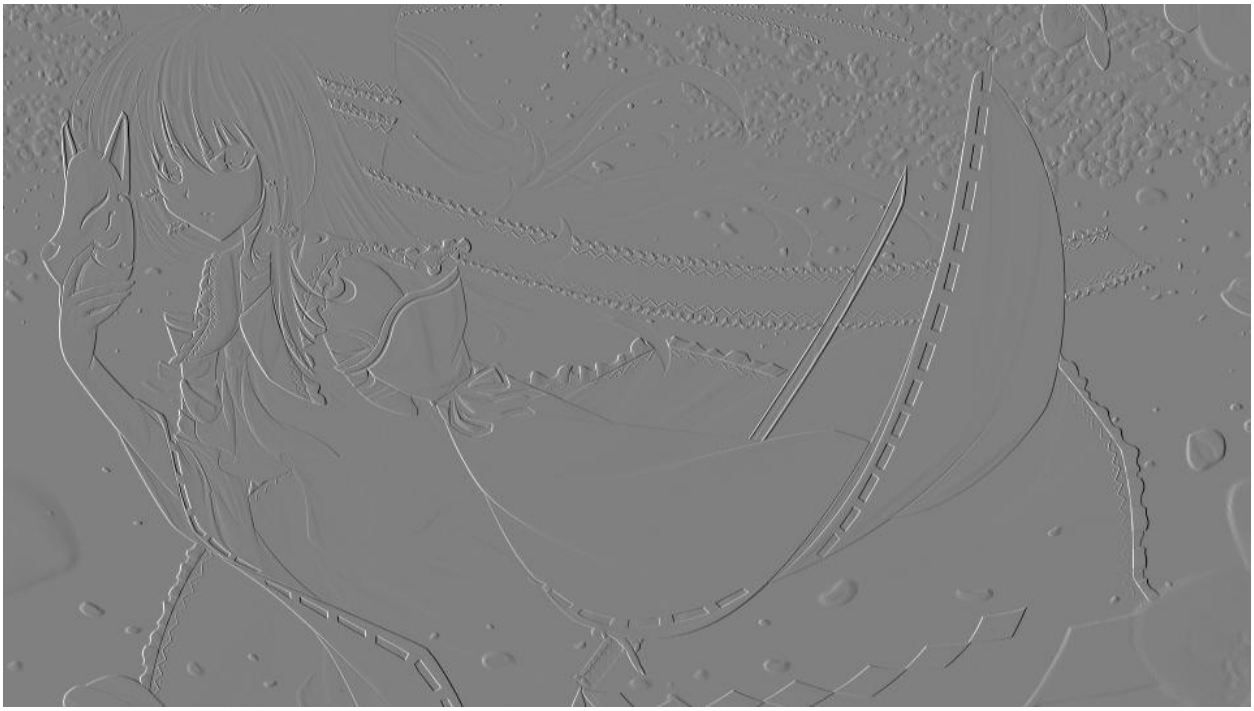

Figure 13: task-3-image-2-o_channel-2

Figure 14: task-3-image-2-o_channel-3

Part B

The time taken increases with the o_channel's increasement as shown in Table.1. The performance plot is shown in Fig.15.

Table 1

i	0	1	2	3	4	5	6	7	8	9	10
o_channel	1	2	4	8	16	32	64	128	256	512	1024
Time(s)	131	278	557	923	1647	3277	6700	13123	28703	57214	122043

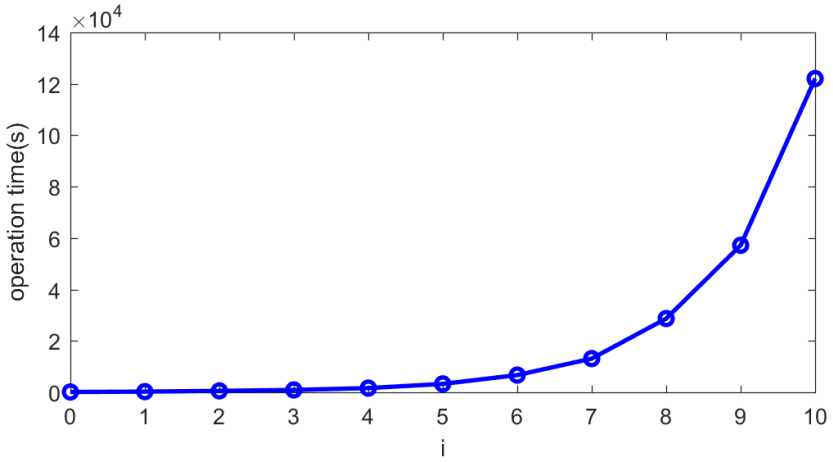

Figure 15: time vs 2^i o_channel

Part C

The number of operations increases with kernel size as shown in Table.2, whose plot is shown in Fig. 16. We can see the operation times has a close-to-linear relationship with kernel size.

Table 2

Kernel_size	3	5	7	9	11
Operation number	64232280	162623648	303818424	487247232	712343000

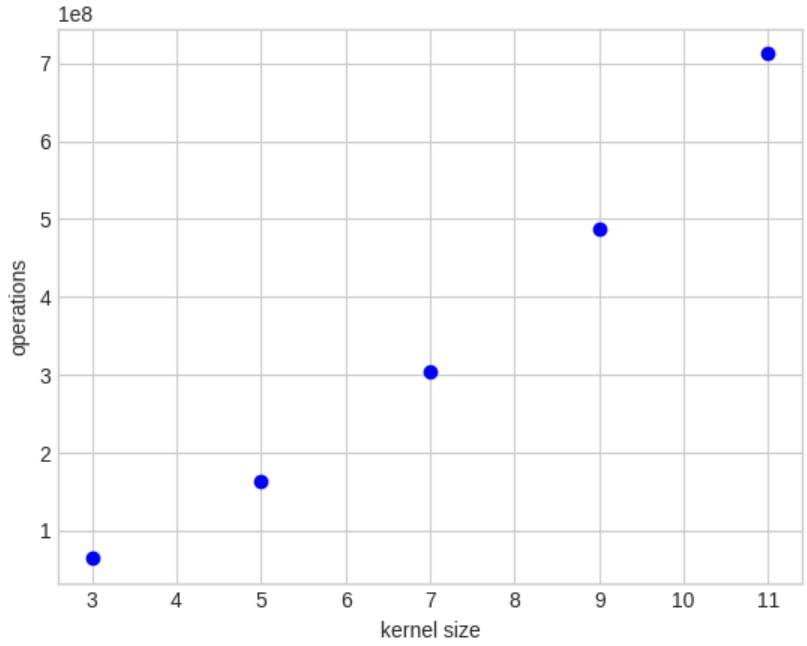

Figure 16: operation number vs kernel size

Part D

See the c code in main.c. The performance time is less than python program. For each i, the consumed time can be viewed in Table.3. The plot is shown in Fig.17.

Table 3

i	0	1	2	3	4	5	6	7	8	9	10
o_channel	1	2	4	8	16	32	64	128	256	512	1024
Time(s)	0.051	0.072	0.150	0.238	0.426	0.792	0.969	1.935	3.847	7.705	16.159

Besides, The number of operations of c is calculated and plot in Fig.18.

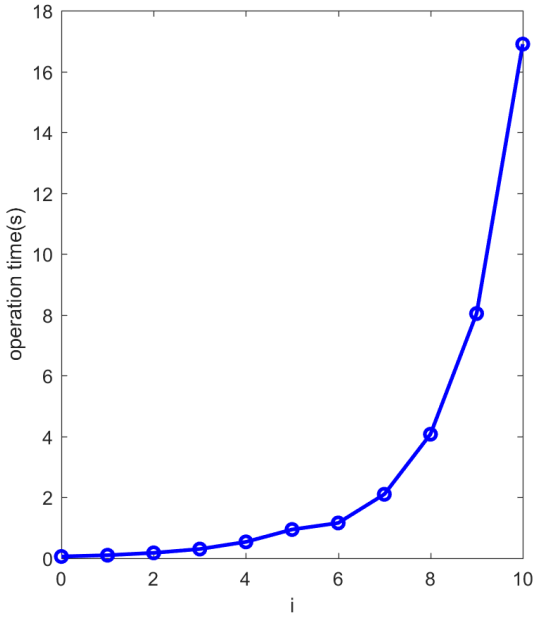

Figure 17

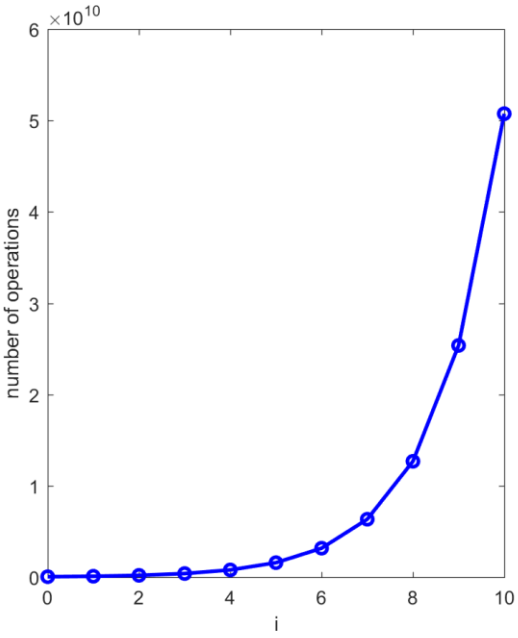

Figure 18