FPGA Acceleration of Convolutional Neural Networks

Shiwen Tang (st3510), Yufei Qian (yq2352), YuFei Jin (yj2725), Gehui Liang(gl2800)， Yi Wang(yw3903)

1. Introduction

As a class of artificial neural networks (ANNs), convolutional neural networks (CNNs) are one of the most used algorithms for visual image analysis. Face segmentation is a bio-metric community research that has received more and more attention in the last two decades with their application in different fields. Based on the interest of the usage of CNN to solve image segmentation problems, facial segmentation has been chosen as the task for this project. However, the traditional general processing unit such as CPU or GPU both cannot provide ideal running rate for CNN. Thus, in this project, FPGA will be chosen as the processor. As a no instruction and no shared memory architecture deice, FPGA is a ideal accelerator for the CNN-based network. To be more specific, the CNN-based network will be designed, pre-trained and accelerated on a FPGA device. Then it will be used for face segmentation of the image captured by a camera and shown the segmentation results on the monitor.

2. Block Diagram

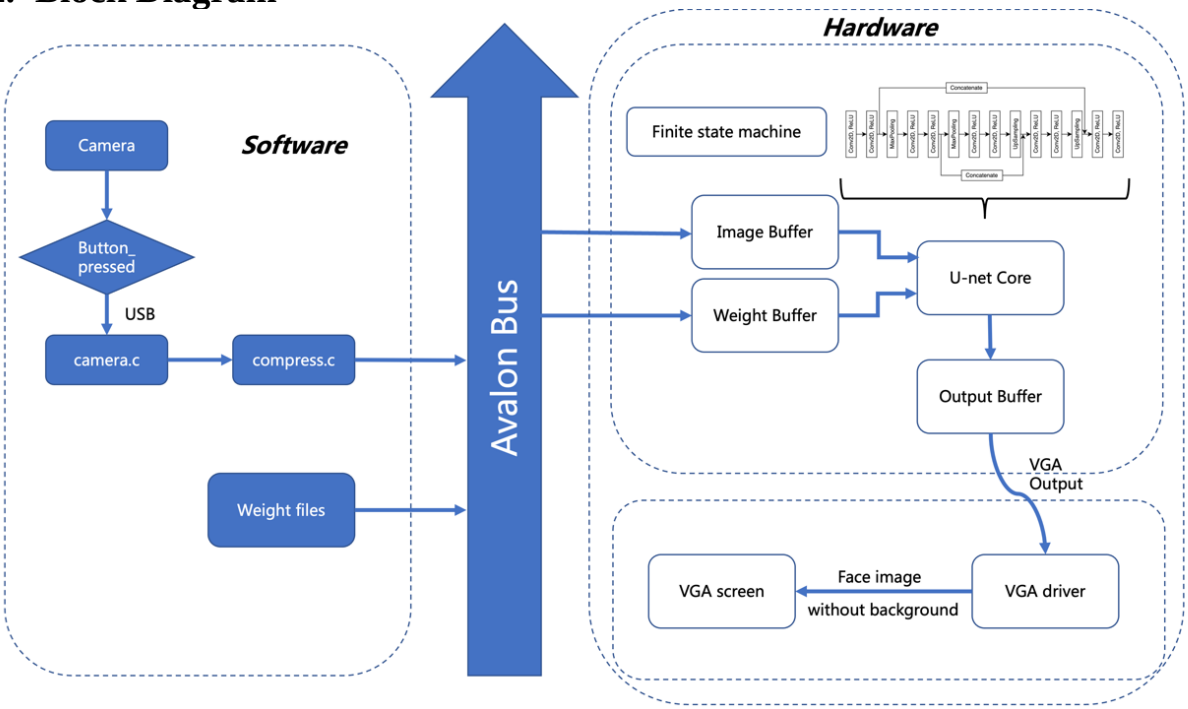


Figure 1 Block Diagram

Our block diagram, as shown in Figure 1, is divided into two parts: software and hardware. The software part runs on the Linux kernel in HPS, and the input data is also stored on the SD card. In the software part, we use a button to control a USB camera to capture an image, compress it into a 64*48 8-bit grayscale image, and then send it to the hardware side of the FPGA through the Avalon Bus. The hardware side mainly includes an input buffer, a controller, and a U-net kernel and all these blocks are controlled by a Finite state machine. The result of each layer will be stored in the B-RAM and transmitted as input to the next layer. The U-net will generate a

mask of the face, and the image within the mask will be output to the display via VGA, as shown in Figure 2 and Figure 3.

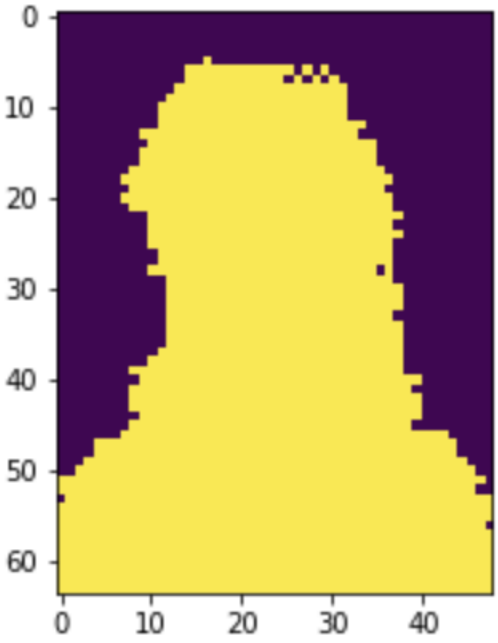


Figure 2 The Mask of Face

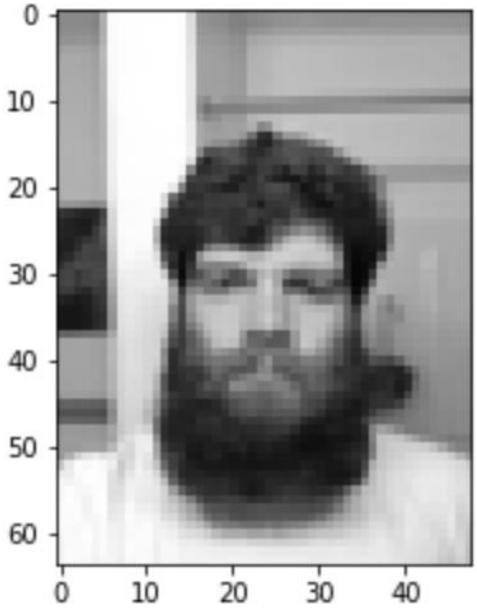
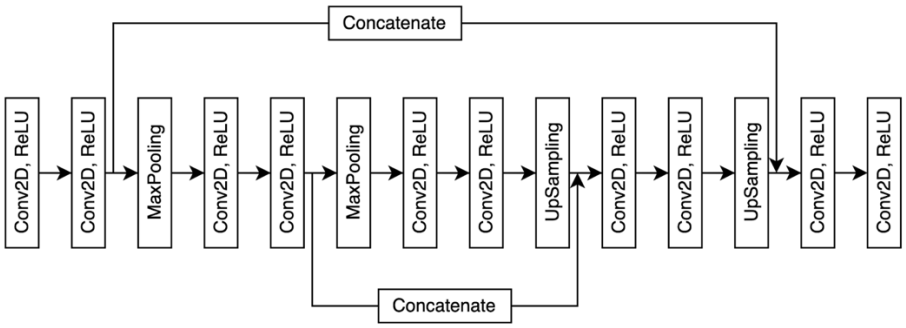


Figure 3 The Input Photo

3. Algorithm

The simple U-Net is modified from U-Net, consisting of 10 convolutional blocks, 2 upsampling layers, 2 donwsampling layers, and 2 concatenate layers. The convolutional blocks and downsampling layers construct the contracting path, which reduces spatial information but increases feature information. The upsampling layers and convolutional blocks construct the expansive path, which combines feature and spatial information by using a series of up-convolutions and merging them with high-resolution features from the contracting path.



3.1 Convolutional Block

algorithm: Conv2d
input*: an image with size (Cin, H, W); kernel with size (Cout, Cin, 3, 3) for convolution; the number of channels Cout*
output*: an image with size (Cout, H, W)*

```
1   def Conv2d(image, kernel, Cout):
2       output = a zero array with size (Cout, H, W)
3       image = image with 0 padding for all edges  //size is (Cin, (H+2), (W+2))
4       for a=0; a<Cout; a++ do
5           for i=0; i<H; i++ do
6               for j=0; j<W; j++ do
7                   for b=0; b<Cin; b++ do
8                       for x=0; x<3; x++ do
9                           for y=0; y<3; y++ do
10                              output[a][i][j] += kernel[a][b][x][y]*image[b][i+x][j+y]
11              output[a][i][j] = ReLU(output[a][i][j])
12      return output
```

3.2 UpSampling Layer

algorithm: UpSampling

input*: an image with size(C, H, W); kernel with size (C, C, 2, 2) for convolution*
output*: an image with size (C, 2H, 2W)*

```
1   def UpSampling(image, kernel):
2       output = a zero array with size (C, 2H, 2W)
3       for a=0; a<C; a++ do
4           for i=0; i<H; i=i+2 do  //stride is (2, 2)
5               for j=0; j<W; j=j+2 do
6                   for b=0; b<C; b++ do
7                       for x=0; x<2; x++ do
8                           for y=0; y<2; y++ do
9                               output[a][i+x][j+y] += image[a][i][j]*kernel[a][b][x][y]
10      return output
```

3.3 DownSampling Layer

algorithm: DownSampling

input*: an image with size (C, 2H, 2W);*
output*: an image with size (C, H, W)*

```
1   def DownSampling(image):
2       output = a zero array with size (C, H, W)
3       for a=0; a<C; a++ do
4           for i=0; i<2H; i=i+2 do
5               for j=0; j<2W; j=j+2 do
6                   output[i/2][j/2] = max(image[i][j], image[i+1][j], image[i][j+1],
                    image[i+1][j+1])
7       return output
```

3.4 Concatenate Layer

algorithm: Concatenate

***input**: image1 and image2 with size (C, H, W)*
***output**: an image with size (2C, H, W)*

```
def Concatenate(image1, image2):
    for a=0; a<C; a++ do
        image1.append(image2[a])
    return image1
```

4. Resource Budget

The input is a grayscale 64*64 image with 6bits. So, the size of the input should be

$$64 \times 48 \times 8 = 24576$$

According to the Algorithm part, we can get a table about the estimation for memory utilization as:

Layer	Data (Bits)	Weights (Bits)	Memory Needed (Bits)
Input	64*48*8	0	24576
Conv2d	96*128*8	8*3*3	98376
Conv2d	96*128*8	8*3*3	98376
downsample	48*64*8	0	24576
Conv2d	48*64*16	16*3*3	49296
Conv2d	48*64*16	16*3*3	49296
downsample	24*32*16	0	12288
Conv2d	24*32*32	32*3*3	24864
Conv2d	24*32*32	32*3*3	24864
upsample	48*64*16	0	49152
concatenate	48*64*32	0	98304
Conv2d	48*64*16	16*3*3	49296
Conv2d	48*64*16	16*3*3	49296
Upsample	96*128*8	0	98304
concatenate	96*128*16	0	196608
Conv2d	96*128*8	8*3*3	98376
Conv2d	96*128*1	1*3*3	12297
		Total:	1058145

As the embedded memory of the FPGA is 4450kb, the estimation of memory utilization for our project is shown as above which is 1059kb. It is totally capable in this project.

5. Hardware-Software Interface

Compared to complex software designs (algorithms), the software/hardware interface is much simpler. An Avalon bus is used to transfer control and data signals through the hardware interfaces. We need only two 16-bit input registers to store image data and parameter data. After being processed into the ideal type (size and gray level), the data will be serially sent to these two registers and then uploaded to FPGA's memory. In our design, the output from the accelerator will be stored in two 16-bit output registers and will be sent back to the software after each batch. An 8-bit register is utilized to control the accelerator, each bit of the control register is described below（the last two bits are reserved for further design）:

1. clk: clock signal

2. rst_n: global reset signal, ’1’ to rest and initialize
3. in_enable: enable signal, ’1’ to start the accelerator
4. in_image_data: input signal，’1’ to tell the accelerator to load image data into memory
5. in_parameter_data: input signal, ‘1’ to tell the accelerator to load parameter data into memory
6. out_ready: ‘1’ means the output is valid, mark the end of the processing

Clk	Ret_n	In_enable	In_image_data	In_parameter_data	Out_ready		

6. Reference

[1] Ronneberger, Olaf & Fischer, Philipp & Brox, Thomas. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. LNCS. 9351. 234-241. 10.1007/978-3-319-24574-4_28.