Report: Final Lab

Leon Mehl

January 22, 2025

1 Functionality

In the code, the parameters of the DNN are represented by the following three variables:

- `int layers_num = x;` // Number of layers in the DNN
- `int input_image_size = y;` // Size of the flattened input image
- `int neurons_num[x] = {a, b, c, ...};` // Number of neurons in each layer

The variable `layers_num` specifies the total number of layers in the DNN. The `input_image_size` determines the input size for the first layer (set to 784 in this case). The array `neurons_num` contains the number of neurons in each layer of the network.

With these parameters, it is possible to quickly adjust the code to work with different DNN configurations. In my implementation, I switched between the test DNN (common to all) and the DNN for group 0. The bias and weight values required for each configuration are requested via UART and stored in appropriate data structures. This approach makes the image processing adaptable to any DNN setup.

2 Baseline Performance of the System

Measurements were conducted for both the test DNN and the Group 0 DNN. The recorded times include the following:

- The total response time of the micro-server (the time between receiving the first byte and printing the result to UART).
- The time taken to receive the image (measured from the arrival of the first byte to the last byte).
- The time required to process the image.
- The time taken to send the result back.

To ensure accurate timing measurements, the following check was implemented to start the timer as soon as the first byte of the image was sent:

```
while (!XUartPs_IsReceiveData(STDIN_BASEADDRESS)) {
    ; // wait until the first byte of the image is received
}
```

To mitigate fluctuations in the image transmission time from RealTerm to the Zybo Z7, an average was calculated using measurements from identifying the digits 0, 3, and 7. These fluctuations were likely caused by external factors affecting the transmission process. Due to the large size of the table, a detailed comparison can be found in the **performance.xlsx** file available in the GitHub repository.

2.1 Time for Receiving the Image

As previously mentioned, the majority of the total time was spent receiving the image, with an average time of 841 ms. This average was calculated across all optimizations and DNN configurations, as the image size and the time required to read from the UART remain consistent across different setups.

Given the size of a single image, we can calculate the bitrate at which it was transmitted. Each image consists of 28x28 pixels, totaling 784 pixels. For each pixel, two bytes are used to store its brightness, resulting in a total of 1568 bytes or 12,544 bits. This matches the file sizes in the folder.

The bitrate was calculated by dividing the data size by the time required for transmission. The resulting average bitrate is 14.91 kbit/s. This is significantly slower than the expected or theoretical rate of 11.52 kbytes/s (or 92.16 kbit/s), assuming a baud rate of 115,200 with 8 data bits, 1 stop bit, and no parity.

The cause of this discrepancy could not be definitively identified. However, one potential explanation might be a delay introduced by RealTerm, the software used to send the file to the board. It is possible that the simultaneous reading of the file and sending of data could slow down the overall transmission process.

2.2 Overall Response Time

The average overall response time without optimization was 813 ms for the test DNN, compared to 873 ms for the Group 0 DNN. The response times fluctuated between a maximum of 816 ms and a minimum of 809 ms for the test DNN, while for the Group 0 DNN, the maximum was 888 ms and the minimum was 810 ms.

The majority of the response time, approximately 99.5%, can be attributed to the time taken to send the image. Since this transmission time remains unaffected by compiler optimizations, the overall response time was not significantly impacted by such optimizations.

2.3 Time for Sending the Response

The response sent back consists of the following string:

```
xil_printf("Image shows the number %d\r\n", result);
```

This string contains 26 ASCII characters. Since each ASCII character is represented by 8 bits, the total size of the response is 208 bits.

On average, transmitting this response took approximately 6.275 µs, corresponding to a bitrate of 33.1 Mbit/s.

2.4 Time for Processing the Image

The processing time for the image was significantly reduced by compiler optimizations. Starting from 4.01ms for the test DNN and 3.87ms for the Group 0 DNN, the time was reduced to 0.197ms (test DNN) and 0.195ms (Group 0 DNN) when using `-O3` optimization. A summary of the measurements is shown in the table below.

It can also be observed that the reduced complexity of the Group 0 DNN (having one layer less than the test DNN) decreases the computation time by 0.14ms under no optimization, while this difference is reduced to only 0.002ms under higher optimization levels.

DNN	**Optimization**	**Tics**	**Time (ms)**
test DNN	NONE	1,335,087	4.01
test DNN	`-O1`	81,876	0.246
test DNN	`-O2`	69,025	0.207
test DNN	`-O3`	65,694	0.197
Group 0 DNN	NONE	1,289,984	3.87
Group 0 DNN	`-O1`	77,755	0.234
Group 0 DNN	`-O2`	65,230	0.196
Group 0 DNN	`-O3`	64,809	0.195

Table 1: Summary of processing times for different optimization levels.

The table clearly shows that the step from no optimization to `-O1` had the most significant impact on processing speed, reducing the computation time by approximately 3.6ms, so the other tests i compared None with `-O1`.

3 OPS and GOPS/s Calculations for Group 0 DNN

For the Group 0 DNN, the Multiply-Accumulate Operations (MACs) and Operations per Second (OPS) were calculated as follows:

3.1 MACs Calculation

- **MACs for Layer 0:** $784 \times 64 = 50,176$
- **MACs for Layer 1:** $64 \times 10 = 640$
- **Total MACs:** $50,176 + 640 = 50,816$

3.2 OPS Calculation

Each MAC consists of two operations: a multiplication and an addition. Therefore, the total OPS can be calculated as:

$$\text{OPS} = 2 \times \text{Total MACs} = 2 \times 50,816 = 101,632 \tag{1}$$

3.3 GOPS/s Calculation

Using the calculated OPS and the measured execution times, the Giga Operations per Second (GOPS/s) were determined. The formula used is:

$$\text{GOPS/s} = \frac{\text{OPS}}{\text{Time in seconds}} \tag{2}$$

The results for different optimization levels are summarized in the table below:

Table 2: GOPS/s Results for Test DNN and Group 0 DNN

DNN	**Optimization Level**	**GOPS/s (1/s)**
Test DNN	None	0.262
Test DNN	-O1	4.27
Test DNN	-O2	5.08
Test DNN	-O3	5.33
Group 0 DNN	None	0.263
Group 0 DNN	-O1	4.35
Group 0 DNN	-O2	5.19
Group 0 DNN	-O3	5.22

3.4 Memory Footprint

The linker script defines four memory regions:

Region	**Base Address**	**Size**
`ps7_ddr_0`	0x100000	0x3FF00000
`ps7_qspi_linear_0`	0xFC000000	0x1000000
`ps7_ram_0`	0x0	0x30000
`ps7_ram_1`	0xFFFF0000	0xFE00

Table 3: Defined Memory Regions in the Linker Script

The dynamically allocated memory in this project is managed through the following sections:

- `.heap`
- `.stack`

The `.heap` section is utilized for dynamically allocated memory during runtime. In this lab setup, it holds the parameters for the DNN. To accommodate all the required data, the heap size was increased to `0x20000`. The heap starts at the aligned end of the `.bss` section and grows upwards towards the stack. The heap configuration is defined as follows:

The `.stack` section, on the other hand, is responsible for local variables and managing function calls (e.g., storing return addresses). It grows downwards from its starting point towards the heap. The stack's end is aligned with the end of the `.heap` section. The stack size was kept at `0x2000`.

The statically allocated memory can be found in the following sections:

- `.text` : Executable code
- `.rodata` : Read-only data (constants)
- `.data` : Initialized global and static variables
- `.bss` : Uninitialized global and static variables

All of these sections are based in the `ps7_ddr_0` region.

4 Optimization

4.1 Image Transfer Optimization

In order to save time during the image transfer, the files were converted from Q8.8 format to Q0.8 format using the `remove_zeros.py` script. With the addition of a new receive function:

Listing 1: Optimized Image Transfer Function

```
DATA readPixelfromUART_opt(){
    unsigned char in;
    DATA out;
    in = XUartPs_RecvByte(STDIN_BASEADDRESS);
    out = (DATA) in;
    return out;
}
```

The image can now be sent in half the time. The time for receiving the image decreased from 841ms to only 380ms, with a bitrate of 16.52 kbit/s. This change in bitrate might be attributed to the limited number of samples taken after optimization.

The complete measurements are documented in the file `optimized-performance.xlsx`.

4.2 Bias/Weights Transfer Optimization

To improve the transfer speed of bias and weight values, their format was changed from Q8.8 to Q1.7 using the `reduce_values_space.py` script. A new receive function was introduced in the Zybo Z7 code:

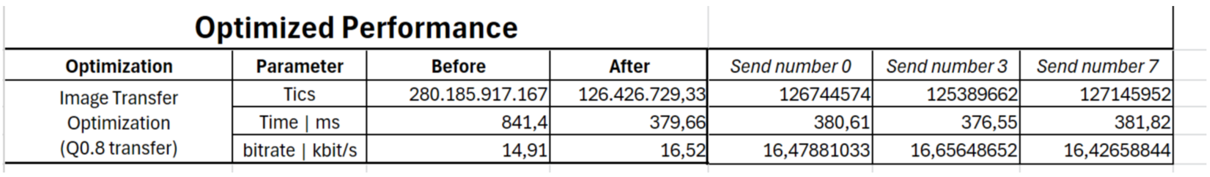
Listing 2: Optimized Bias/Weights Transfer Function

```
DATA readQ1_7ValuesFromUart(){
    unsigned char in;
    DATA sign, out;
    in = XUartPs_RecvByte(STDIN_BASEADDRESS);
    if (in & 0x80){
        sign = 0xFF00;
    } else {
        sign = 0x0000;
    }
    out = (DATA) (in << 1) | sign;
    return out;
}
```

This function converts the received data back to Q8.8 format. Measurements were conducted to evaluate the time improvement. For instance, the transfer time for weights in layer one decreased from 2.23 seconds to 1.21 seconds.

The bitrate for larger packets, such as `weights 1` and `weights 2`, was approximately 16 kbit/s, similar to the image transfer. However, for smaller packets, the bitrate was around 96 kbit/s. This closer approximation to the theoretical bitrate of 115200 baud is likely due to the smaller packets fitting within the 64-byte UART FIFO buffer, eliminating delays caused by buffer refills.

One outlier was the time for sending `weights 0`. The time calculated by the program was significantly shorter than the actual time. A manual measurement using a stopwatch recorded a time of approximately 53 seconds for non-optimized `weights 0`. This discrepancy is likely due to an overflow in the `int32` variable used to store internal ticks during transmission.



Optimized Performance

Optimization	**Parameter**	**Before**	**After**	*Send number 0*	*Send number 3*	*Send number 7*
Image Transfer Optimization (Q0.8 transfer)	Tics	280.185.917.167	126.426.729,33	126744574	125389662	127145952
	Time | ms	841,4	379,66	380,61	376,55	381,82
	bitrate | kbit/s	14,91	16,52	16,47881033	16,65648652	16,42658844

Figure 1: Optimized performance