Report (Final Lab)

1 Functionality

The parameters of the DNN are represented through these three variables in the code:

```
int layers_num = x;
int input_image_size = y;
int neurons_num[x] = {a, b, c, ...};
```

The layer number defines how many layers are included in the DNN.
The flatten image size is used to determine the input of the first layer. (In our case this is always 784)
The neurons number array defines the number of neurons in each layer.

Using these parameters we can easily switch between different DNNs. In my case this was between the test DNN (given to everyone) and the DNN for group 0.

Based on these parameters the necessary bias and weight values are asked for (over UART) and saved in corresponding structures. Processing of the images is thereby flexible for different DNN parameterization.

2 Baseline Performance of the System

Measurement were done for both, the test DNN and the Group 0 DNN.
The measured times include:

- overall responds time of the micro-server (time between the first byte received and the output printed out to UART)
- receiving the image (time between the first and last byte received)
- processing the image
- sending the result back

In order to start the timer first when the first byte of the image is sent this check was included:

```
while (!XUartPs_IsReceiveData(STDIN_BASEADDRESS)) {
    ; // wait unitl first byte of the image is sent before starting the timer
}
```

An average was taken of the times for identifying the numbers 0, 3 and 7. This presented it as useful because the time it took for the image to be sent from RealTerm to the Zybo z7 experienced some fluctuations. These were probably caused by other factors.

Also different compiler optimizations were used, these were:

- -NONE / -O0
- -O1
- -O2
- -O3

A table with the complete results of all the measurements taken can be found in the file `baseline-performance.xlsx`.

2.1 Overall respond time

The average overall responds time for no optimization was 813ms for the test DNN (873ms for the Group 0 DNN), fluctuating between a maximum of 816ms and minimum of 809ms (888ms and 810ms for Group 0 DNN). Most of the respond time can be attributed to sending the image, around 99.5%. Since this time does not change with compiler optimization the overall responds time was not significantly effected by this kind of optimization.

2.2 Time for receiving the image

As mentioned before the majority of the time was used to receive the image, resulting in an average of 841ms. Here the average was taken over all optimizations and DNNs, because the image size and reading time from the UART doest not change between the different setups.
With the size of one image we can also compute the bitrate at which the image is sent. The image is made out of 28x28 pixel, 784 pixel in total. For each pixel two bytes are used to store its brightness, resulting in 1568 bytes / 12544 bits. This is also matches the size of the files in the folder.
The bitrate can no be calculated by dividing the data size by the time it takes to transfer them. Therefore the average bitrate is 14.91 kbit/s. This is far slower than the expected/theoretical speed of 11520 bytes/s, given the baud rate is 115200 with 8 data bits, 1 stop bit and no parity.
A reason for this could not be directly be pinpointed but one possible reason might be a delay related to RealTerm, the software used to send the file to the board. Here it might be that reading the file and sending is done simultaneously and thereby slows down the sending process.
Later we will also compare the receive bitrate with the send bitrate in 2.3.

2.3 Time for processing the image

The processing time of the image was significantly reduced by compiler optimizations. Starting from 4.01ms (test DNN) and 3.87ms (group 0 DNN) and going down to 0.197ms (test DNN) and 0.195ms (Group 0 DNN) for -O3 optimization. A summary of the measurements can be found in the table below. It can be seen that having one layer less in the group 0 DNN reduces the time of computation by 0.14ms, but with higher optimization this slight gain is reduced to only 0.002ms.

DNN	Optimization	tics	time in ms
test	-NONE	1,335,087	4.01
test	-O1	81,876	0.246
test	-O2	69,025	0.207
test	-O3	65,694	0.197
group 0	-NONE	1,289,984	3.87
group 0	-O1	77,755	0.234
group 0	-O2	65,230	0.196
group 0	-O3	64,809	0.195

The table also shows that the step from no optimization to -O1 has the most impact on processing speedup of around 3.6ms.

As a next step the MACs and the overall GOPS/s are calculated.
In our DNN setup the multiply and accumulate operations in each layer can be calculated with this formula:

```
MACS_layer = input_neurons * output_neurons
```

with the total operations being two times the total MACs. For our two DNNs this results in the following calculations:

OPS - TEST DNN:

MACS_layer0 = 784 * 64 = 50176
MACS_layer1 = 64 * 32 = 2048
MACS_layer2 = 32 * 10 = 320
MACS_total = 50176 + 2048 + 320 = 52544
OPS = 2 * 52544 = <u>105088</u>

OPS GROUP0 DNN

MACS_layer0 = 784 x 64 = 50176
MACS_layer1 = 64 x 10 = 640
MACS_total = 50176 + 640 = 50816
OPS = 2 * 50816 = <u>101632</u>

With these numbers we can calculate the GOPS/s:

DNN	Optimization	GOPS/s in 1/s
test	-NONE	0.262
test	-O1	4.27
test	-O2	5.08
test	-O3	5.33
group 0	-NONE	0.263
group 0	-O1	4.35
group 0	-O2	5.19
group 0	-O3	5.22

2.3 Time for sending the responds

The responds consists of this string:

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

This responds contains 26 ASCII characters. For each ASCII character 8 bits are needed, coming to a total of 208 bits per reply. On average this took 6.275µs, resulting in a bitrate of 33.1 Mbit/s. Comparing this with the receive time, this is much higher and also exceeds the theoretical limit of 11520 bytes/s. A reason for this discrepancy could not be found.

2.4 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

The dynamically allocated memory can be found in the sections:

- .heap
- .stack

The heap section is used for dynamically allocated memory during runtime. This is where the parameters of the DNN are located in this solution for the lab. In order to hold all the data it was extended to a size of 0x20000. The heap starts at the (aligned) end of the `.bss` section and grows upwards towards the stack.

```
 . = ALIGN(16);
_heap = .;
HeapBase = .;
_heap_start = .;
. += _HEAP_SIZE;
_heap_end = .;
HeapLimit = .;
```

The stack section is used for local variables and function call management (eg. return addresses). It grows down from its starting point towards the heap. The end of the stack is the (aligned) end of the `.heap` section.

```
 . = ALIGN(16);
_stack_end = .;
. += _STACK_SIZE;
. = ALIGN(16);
_stack = .;
```

The stack size was left at 0x2000

The statically allocated memory can be found in the other sections like:

- .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.

2.5 Conclusion

In order to optimize the time for the image classification with the Zybo z7 the main focus should be in optimizing the transfer of the image. For small neural networks, as we deploy here, a optimization of the algorithm is secondary. Although it would be possible to improve the multiply and accumulate steps with NEON intrinsics, done similarly in the previous lab.

3 Optimization

Image Transfer Optimization

In order to save time when sending the the image, the files where converted from Q8.8 format to Q0.8 using the `remove_zeros.py` script. With the small addition of a new receive 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 goes down from 841ms to only 380ms with a bitrate of 16.52 kbit/s. Reason for the change of bitrate might be because of only three samples taken after the optimization.
The complete measurements can be found in the file `optimized-performance.xlsx`.

Bias/Weights Transfer Optimization

With this optimization the transfer speed of the bias and weights values was improved. The format of the values was changed form Q8.8 to Q1.7 using the `reduce_values_space.py` script. In the Zypo z7 code a new receive function for the values was introduced:

```
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.
Here also some measurement were taken to identify the time gain. These measurements were done on the values of the test DNN. As expected this optimization decreased the transfer time to about half the original time.

For example for the weights in layer one the time decreased from 2.23s to 1.21s. The bitrate varied between big data packets and small data packets. For bigger packets like weights 1 and weights 2 the bitrate was about the same as for the transfer of the picture ~16 kbit/s. But for the smaller packets the calculated bitrate was ~96 kbit/s. The second bitrate is more closer to the expected bitrate of a baud rate of 115200. The reason for this is probably that the smaller packets fit in the buffer of the 64-byte UART FIFO buffer. Thereby the transmission does not have to wait until this buffer is emptied again.
On outlier is the time for sending weights 0. The time calculated by the programme is much shorter that the real time. A measurement taken with a stopwatch gave a time of ~53s (not optimized weights 0). The reason for this discrepancy is probably because the int32 used for holding the internal tics overflowed during the transmission.

Processing with 8 bit values

A way to save memory space is to not only send the data in Q1.7 format but also save it as such. A new set of functions was introduced handling the processing of the image in Q1.7 format. This resulting in a lot of saturation errors and a DNN which could not correctly classify the images anymore.
Clearly it could be because of wrong implementation or also because of a wrong choice of the quantization factor `qf`.
The efforts done can be found in the file `main_files/vbyte_opt.c`.