UNIVERSITÄT TÜBINGEN
Prof. Dr.-Ing. Hendrik P.A. Lensch
Lehrstuhl für Computergrafik
Lukas Ruppert (lukas.ruppert@uni-tuebingen.de)
Mehran Hosseinzadeh (mehran.hosseinzadeh@uni-tuebingen.de)
Deborah Kornwolf (deborah.kornwolf@student.uni-tuebingen.de)
Philipp Langsteiner (philipp.langsteiner@student.uni-tuebingen.de)

February 28, 2025

Massively Parallel Computing
Assignment 5

Instructions

Download the framework `exercise05.zip` from the ILIAS course web page.

Present your results to the exercise instructors to get a grading on this exercise sheet.

5.1 Matrix Multiplication (50 P)

- Start with the source code in the `MatrixMul` folder.
- Implement two square matrix multiplications:
  - `multiplyMatrixGpu1(...)` should calculate one element per thread by reading the input data trivially from global memory.
  - `multiplyMatrixGpu2(...)` should also calculate one element per thread, but blocks should cooperate to read common input data to shared memory first.
- *Bonus:* Reduce the runtime of your fully working `multiplyMatrixGpu2` to below 2 ms.

5.2 MNIST with Center Surround Convolution in PyTorch (50 P)

Figure 1: MNIST Dataset

The MNIST-dataset consists of small images of hand written numbers, some of which are shown in Figure 1. In this exercise we will use `pytorch` to train a deep neural network on classifying the numbers shown in the images. We will use the architecture shown in Figure 2. However, instead of using the 2D Convolutions from `pytorch` we will write an extension for a *Center Surround Convolution* layer.

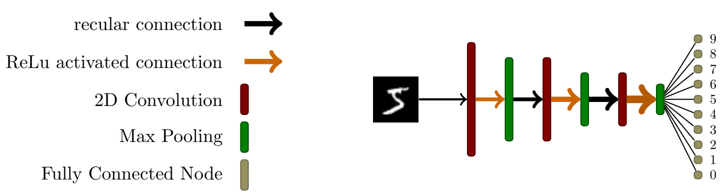


Figure 2: Network Architecture

One element of the output of a center surround convolution is computed by summing up the same position in the input tensor, weighted with the center weight and its direct neighbourhood weighted with the surround weight. Finally, a output channel specific bias is added.

These are the variables used for the mathematical formulation:

- I is the input tensor of shape $N \times C_i \times H \times W$.
- O is the output tensor of shape $N \times C_o \times H - 2 \times W - 2$.
- L is the final loss of your network, the value you are optimizing for and that we would like to calculate derivatives for.
- N is the batch count.
- C_i is the number of channels in the input Tensor.
- C_o is the number of channels in the output Tensor.
- H is the *height* of the input Tensor.
- W is the *width* of the input Tensor.
- b is the batch number.
- c_o is the output channel.
- y_o is the output position on the y - axis.
- x_o is the output position on the x - axis.
- w_c is the weight tensor for the centers of shape $C_i \times C_o$.
- w_s is the weight tensor for the surrounding of shape $C_i \times C_o$.
- w_b is the weight tensor for the bias of shape C_o. (Yes it is a vector)

The forward pass can be written as

$$O[b, c_o, y_o, x_o] = \left(\sum_{c_i=0}^{C_i-1} I[b, c_i, y_o+1, x_o+1] \cdot w_c[c_i, c_o] + w_s[c_i, c_o] \cdot S(I, b, c_i, y_o+1, x_o+1) \right) + w_b[c_o] \tag{1}$$

Where S represents the sum over the $3x3$ spatial neighbourhood of the given position:

$$S(I, b, c_i, Y, X) = \sum_{y=Y-1}^{Y+1} \sum_{x=X-1}^{X+1} \begin{cases} I[b, c_i, y, x] & \text{if } \neg((y = Y) \wedge (x = X)) \\ 0 & \text{else} \end{cases} \tag{2}$$

See Figure 3a for a visualization of this computation. Note that the result is smaller than the input.

The derivatives for the backward pass are given by

$$\frac{\partial L}{\partial w_b[c_o]} = \sum_{b=0}^{N-1} \sum_{y_o=0}^{H-3} \sum_{x_o=0}^{W-3} \frac{\partial L}{\partial O[b, c_o, y_o, x_o]} \tag{3}$$

$$\frac{\partial L}{\partial w_c[c_i, c_o]} = \sum_{b=0}^{N-1} \sum_{y_o=0}^{H-3} \sum_{x_o=0}^{W-3} \frac{\partial L}{\partial O[b, c_o, y_o, x_o]} \cdot I[b, c_i, y_o+1, x_o+1] \tag{4}$$

$$\frac{\partial L}{\partial w_s[c_i, c_o]} = \sum_{b=0}^{N-1} \sum_{y_o=0}^{H-3} \sum_{x_o=0}^{W-3} \frac{\partial L}{\partial O[b, c_o, y_o, x_o]} \cdot S(I, b, c_i, y_o+1, x_o+1) \tag{5}$$

The derivative w.r.t. the input tensor is very similar to the forward pass except for the border cases shown in Figure 3c. By padding $\frac{\partial L}{\partial O}$ with two zero borders the equation for the derivatives becomes:

$$\frac{\partial L}{\partial I[b, c_i, y_i, x_i]} = \sum_{c_o=0}^{C_o-1} \left(\frac{\widetilde{\partial L}}{\partial O[b, c_o, y_i+1, x_i+1]} w_c[c_i, c_o] + w_s[c_i, c_o] \cdot S\left(\frac{\widetilde{\partial L}}{\partial O}, b, c_o, y_i+1, x_i+1 \right) \right) \tag{6}$$

Where that $\frac{\widetilde{\partial L}}{\partial O}$ is the padded version of $\frac{\partial L}{\partial O}$ and has the shape $N \times C_o \times H + 1 \times W + 1$. The code for padding is already given in the template.

a) If you run this task on your own machine, first install the required dependencies in a terminal.

```
# only if you run this on your own machine
pip3 install --user --upgrade numpy matplotlib torch torchvision tqdm tensorboard
```

On our machines, everything should already be setup for you.

b) The given framework contains a trainable implementation of the network architecture shown in Figure 2. The Network is defined in `networks.py`. Make sure you can train it by calling:

```
python3 train_network.py --model conv
```

c) Implement the forward pass of the Center Surround Convolution.

- Complete the `center_surround_convolution_forward_kernel` that computes the forward pass from Equation 1.
- The function for memory allocation and launching is already given in: `center_surrond_convolution_forward`.

d) Implement the backward pass of the Center Surround Convolution.

- Complete the CUDA kernels for the partial derivatives $\frac{\partial L}{\partial I}$, $\frac{\partial L}{\partial w_c}$, $\frac{\partial L}{\partial w_s}$ and $\frac{\partial L}{\partial w_b}$ by implementing Equations 3, 4, 5 and 6.
- Here, the function for launching and allocation is also already provided in: `center_surround_convolution_backward`.

e) A third model is already setup and the remaining setup for your new Center Surround Convolution is provided in `center_surround_convolution.py`. This loads your model dynamically. Train the network by calling:

```
python3 train_network.py --model csc
```

f) Optimize your Kernels to get more speed. (optional)

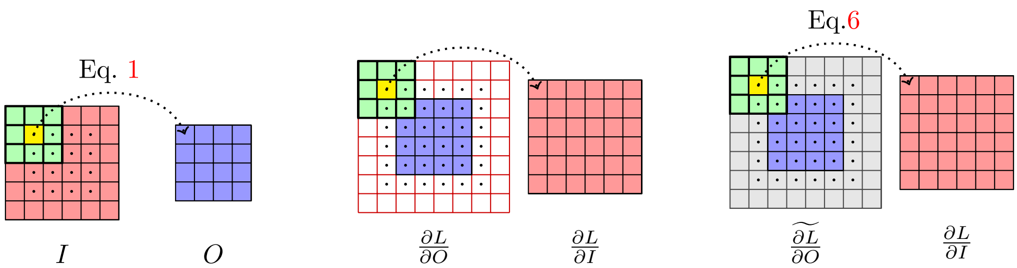
Hints

- See Figure 3 for a visualization of the forward and backward pass where $B = C_i = C_o = 1$, and $H = W = 6$.
- You can test your implementations against the provided unit tests:

```
python3 test_center_surround_convolution.py
```

- The `train_network.py` script creates a training log, which can be viewed in `tensorboard`.

```
tensorboard --logdir /tmp/mnist_$USER
```



(a) Forward Pass

(b) Backward Pass (required border handling)

(c) Backward Pass with padding

Figure 3: Memory access for forward and backward pass. Valid pixel of I or $\frac{\partial L}{\partial I}$. Valid pixel of O or $\frac{\partial L}{\partial O}$. Center part of the center surround kernel. Surrounding part of the center surround kernel. Invalid coordinate. padded border. Position on which the kernel is placed to compute one of the output positions.