DLR: Deep Learning Routines

- high-level synthesizable C/C++ routines for deep learning inference network -

2021

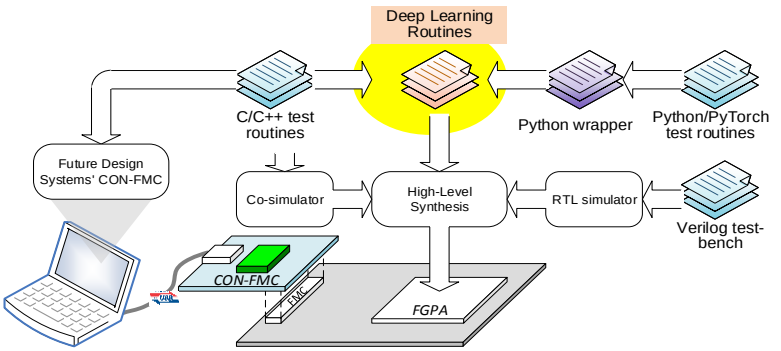
Ando Ki, Ph.D.
adki@future-ds.com

Table of contents

- DLR: Deep Learning Routine
- How to prepare DLR library
- DLR: Deep Learning Routines
- Convolution routine
- ReLU Activation routine
- LeakyReLU Activation routine
- Max pooling routine
- Linear routine
- Batch normal routine

DLR: Deep Learning Routine

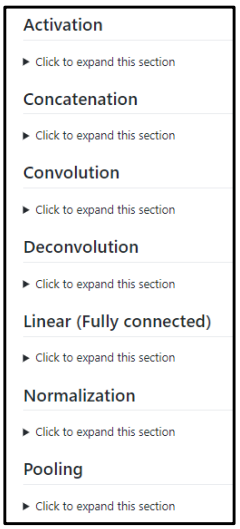
- 'DLR' as a part of DPU (Deep Learning Processing Unit) is a collection of high-level synthesizable C/C++ routines for deep learning inference network.
  - https://github.com/github-fds/Deep_Learning_Routines



How to prepare DLR library

- Get a clone
  - $ git clone https://github.com/github-fds/Deep_Learning_Routines.git
- Go to 'src' directory
  - $ cd Deep_Learning_Routines/v1.3/src
- Compile and install
  - $ make
  - $ make install
- See 'include' and 'lib' directories
  - $ cd ..
  - $ ls include lib

DLR: Deep Learning Routines

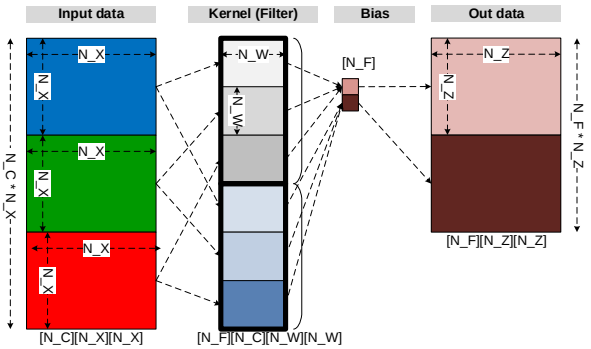


Convolution routine (1/2)

```
template<class TYPE=float>
void Convolution2d
(           TYPE     *out_data
  , const TYPE     *in_data
  , const TYPE     *kernel
  , const TYPE     *bias
  , const uint16_t  out_size
  , const uint16_t  in_size
  , const uint8_t   kernel_size
  , const uint16_t  bias_size
  , const uint16_t  in_channel
  , const uint16_t  out_channel
  , const uint8_t   stride
  , const uint8_t   padding=0
  #if !defined(__SYNTHESIS__)
  , const int       rigor=0
  , const int       verbose=0
  #endif
);
```

- 2 Dimensional convolution for specific data type
- output
  - out_data: pointer to output buffer in ‘out_channel x out_size x out_size’.
- intputs
  - in_data: pointer to input buffer in ‘in_channel x in_size x in_size’
  - kernel: pointer to kernel buffer in ‘out_channel x in_channel x kernel_size x kernel_size’
  - stride: stride for kernel
  - padding: padding for kernel
- debugging arguments when ‘__SYNTHESIS_’ is not defined

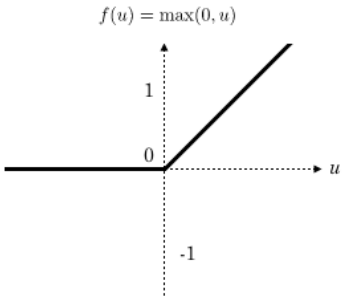
Convolution routine (2/2)



- Example
  - input channel: 2
  - output channel: 2
  - number of kernels: (output channel) x (input channel)

ReLU Activation routine

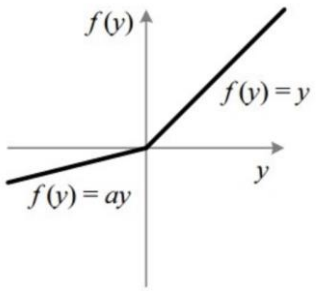
```
template<class TYPE =float>
void ActivationReLu
(
          TYPE      *out_data
    , const TYPE    *in_data
    , const uint32_t  size
    , const uint16_t  channel
    #if !defined(__SYNTHESIS__)
    , const int       rigor=0
    , const int       verbose=0
    #endif
);
```



- Rectified Linear Unit
  - Non-linear activation function
- output
  - out_data: pointer to output buffer in 'channel x size x size' elements
- intputs
  - in_data: pointer to input buffer in 'channel x size x size'
- debugging arguments when '__SYNTHESIS_' is not defined

LeakyReLU Activation routine

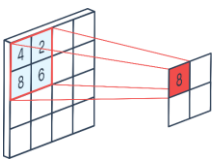
```
template<class TYPE =float>
void ActivationLeakyReLu
(           TYPE     *out_data
  , const TYPE     *in_data
  , const uint32_t  size
  , const uint16_t  channel
  , const uint32_t  negative_slope=0x3DCCCCCD
  #if !defined(__SYNTHESIS__)
  , const int       rigor=0
  , const int       verbose=0
  #endif
);
```

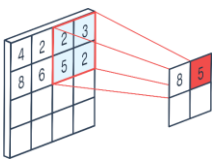


- Leaky Rectified Linear Unit
  - Non-linear activation function
- output
  - out_data: pointer to output buffer in 'channel x size x size' elements
- intputs
  - in_data: pointer to input buffer in 'channel x size x size'
  - negative_slope: slope for negative input (0.01 by default)
    - Note that it uses 32-bit bit-pattern for floating-point value (IEEE 754 single-precision)
      - E.g., 0x3DCCCCCD means 0.01
- debugging arguments when '__SYNTHESIS_' is not defined

Max pooling routine

```
template< class TYPE=float
          , const int ReLu=0
          , const int LeakyReLu=0
          , const uint32_t negative_slope=0x3DCCCCCD >
void Pooling2dMax
(           TYPE     *out_data
  , const TYPE     *in_data
  , const uint16_t  out_size
  , const uint16_t  in_size
  , const uint8_t   kernel_size
  , const uint16_t  channel
  , const uint8_t   stride
  , const uint8_t   padding=0
  , const int       ceil_mode=0
  #if !defined(__SYNTHESIS__)
  , const int       rigor=0
  , const int       verbose=0
  #endif
);
```



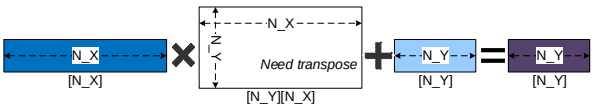


- 2 Dimensional max pooling
  - Down sampling
- output
  - out_data: pointer to output buffer in 'channel x out_size x out_size'.
- intputs
  - in_data: pointer to input buffer in 'channel x in_size x in_size'
  - kernel_size: pooling dimension
  - stride: stride for kernel
  - padding: padding for kernel
  - ceiling_mode: use ceil() instead of floor() to calculate output size
- debugging arguments when '__SYNTHESIS_' is not defined

Linear routine

```
template< class TYPE=float
        , int ReLu=0
        , const int LeakyReLu=0
        , const uint32_t negative_slope=0x3DCCCCCD // 0.01
        >
void Linear1d
(           TYPE    *out_data // out_size
    , const TYPE    *in_data  // in_size
    , const TYPE    *weight   // out_size x in_size
    , const TYPE    *bias     // out_size
    , const uint16_t  out_size
    , const uint16_t  in_size
    , const uint16_t  bias_size
    #if !defined(__SYNTHESIS__)
    , const int       rigor=0     // check rigorously when 1
    , const int       verbose=0   // verbose level
    #endif
);
```

$$y = xW^T + b$$



- Linear transformation
- output
  - out_data: pointer to output buffer in ‘out_size’.
- intputs
  - in_data: pointer to input buffer in ‘in_size’
  - weights: pointer to contiguous buffer containing weights in ‘out_size x in_size’
  - bias: point to contiguous buffer containing biases
  - out_size:
  - in_size:
  - bias_size:
- debugging arguments when ‘__SYNTHESIS_’ is not defined

Batch normal routine

```
template< class TYPE=float
        , int LeakyReLu=0
        , int negative_slope1000=100>
void Norm2dBatch
(           TYPE    *out_data
    , const TYPE    *in_data
    , const TYPE    *running_mean
    , const TYPE    *running_var
    , const TYPE    *scale
    , const TYPE    *bias
    , const uint32_t  in_size
    , const uint16_t  scale_size
    , const uint16_t  bias_size
    , const uint16_t  in_channel
    , const float     epsilon=1E-5
    #if !defined(__SYNTHESIS__)
    , const int       rigor=0
    , const int       verbose=0
    #endif
);
```

$$y = \frac{x - E[x]}{\sqrt{Var[x] + \epsilon}} * \gamma + \beta$$

where ‘y’ is outut, ‘x’ is input, ‘E[x]’ is mean, ‘Var[x]’ is variance, ‘gamma’ is scaling factor, ‘beta’ is shift factor (bias), ‘epsilon’ is value for numerical stability.

- Batch normalization for each channel
- output
  - out_data: pointer to output buffer in ‘in_channel x out_size x out_size’.
- intputs
  - in_data: pointer to input buffer in ‘in_channel x in_size x in_size’
  - running_mean: means of each channel in ‘in_channel’
  - running_var: variance of each channel in ‘in_channel’
  - scale (gamma): pointer to buffer of scale factor in ‘in_channel’
  - bias (beta): pointer to buffer of shift factor of each channel in ‘in_channel’
  - epsilon: numeric stability (1.0E-5 by default)
- debugging arguments when ‘__SYNTHESIS_’ is not defined