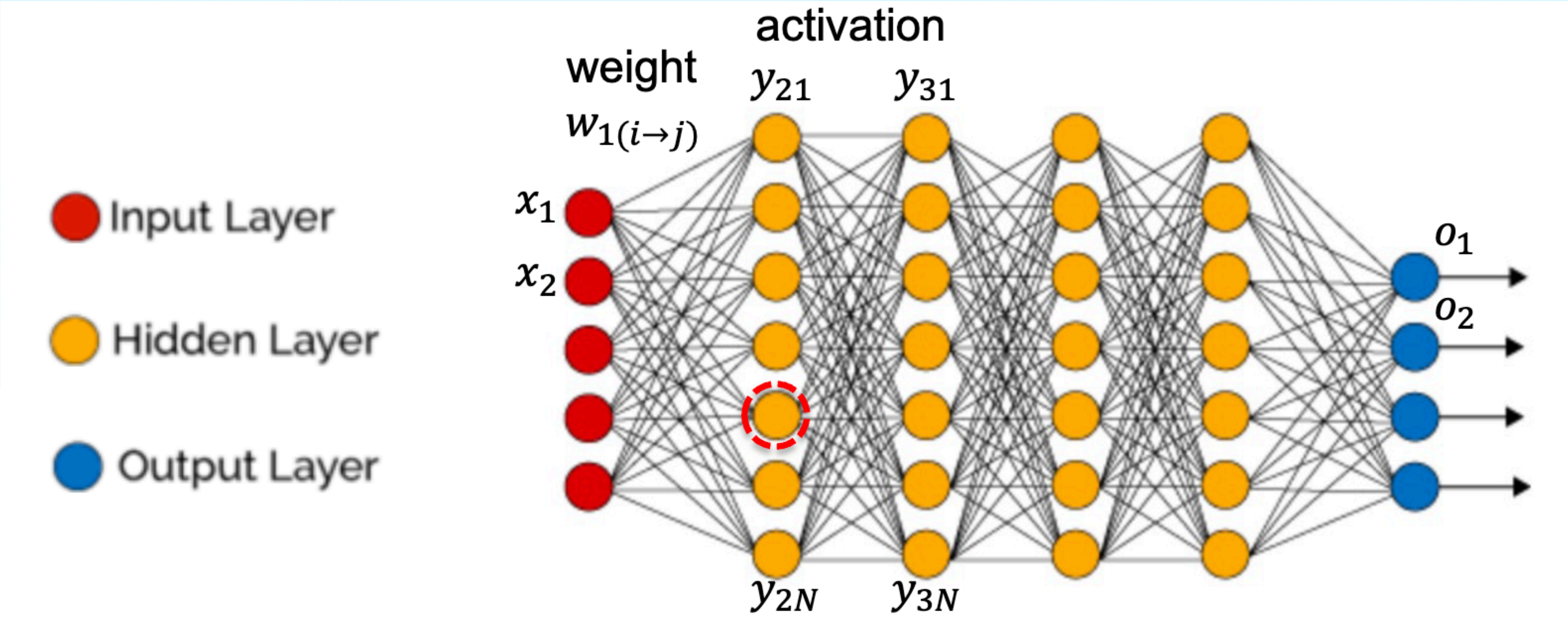
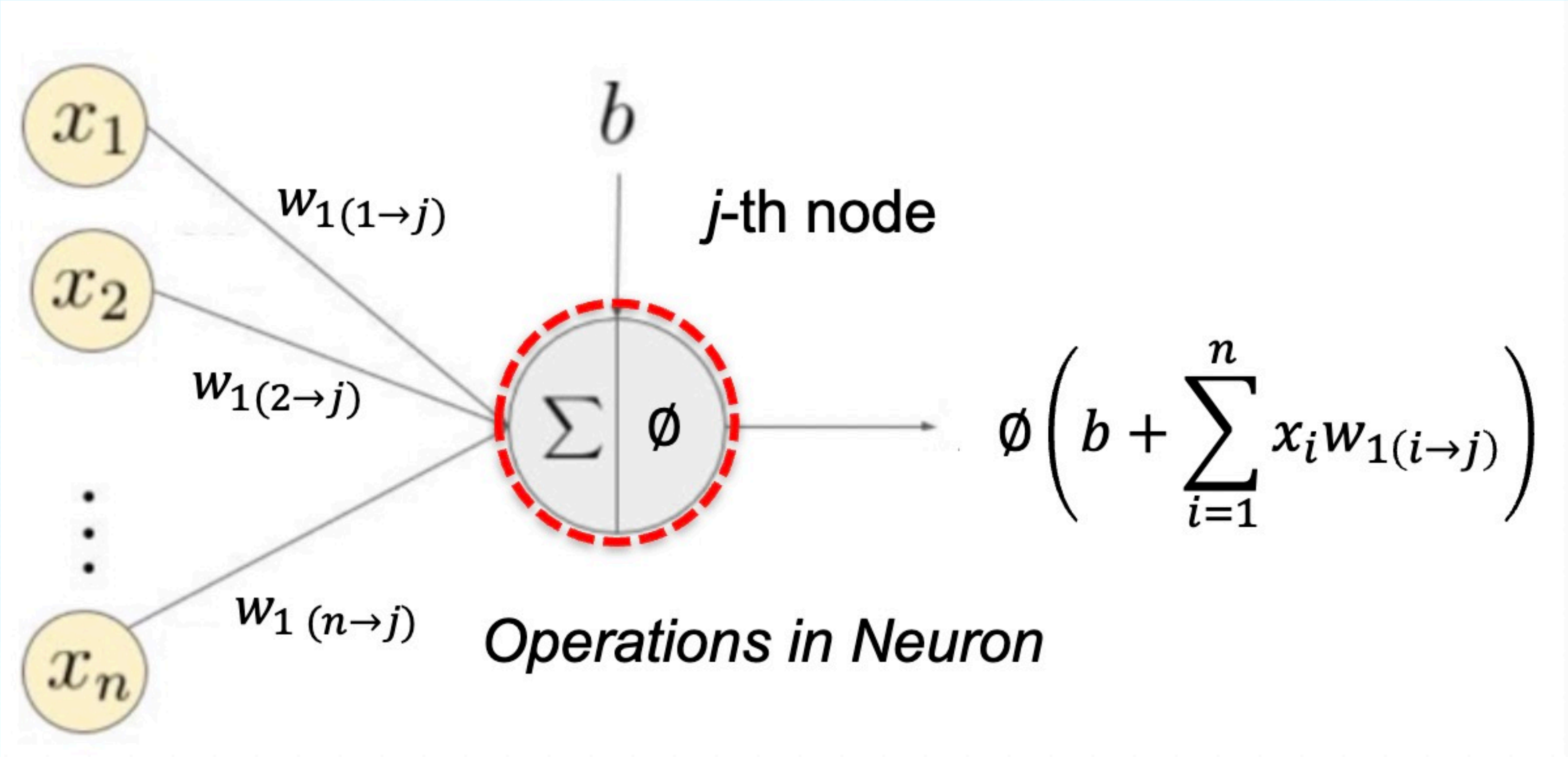
ECE 277 Project Presentation

GPU Accelerated Machine Learning with CUDA and Pybind

Dingye Chen A16164036

Deep Neural Network Multi layer Perceptron



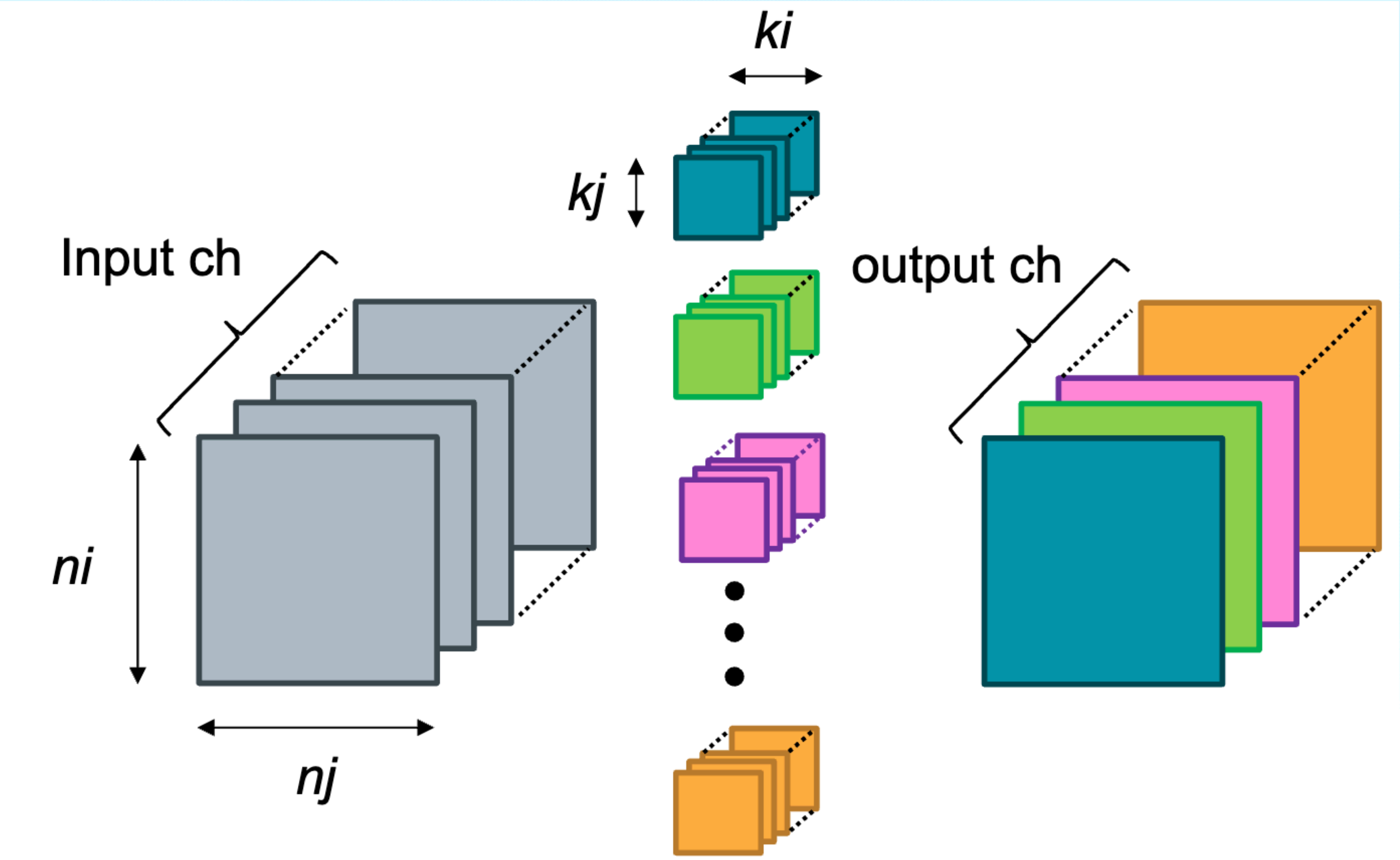


$$\begin{bmatrix} w_{1(1\to 1)} & \cdots & w_{1(5\to 1)} \\ \vdots & \ddots & \vdots \\ w_{1(1\to N)} & \cdots & w_{1(5\to N)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} y_{21} \\ y_{22} \\ y_{23} \\ \cdots \\ y_{2N} \end{bmatrix}$$

Matrix-vector multiplication
(bias omitted for simplicity)

Figures from https://becominghuman.ai/, https://www.learnopencv.com/, https://medium.com/

Convolution Layer



- Assumption: 3x3 kernel, 16x16 input feature map, 64 in / out channel.
- Activation size: 64x(32x32)
- Weight size: 64x64x(3x3)
- Weight is reused by nij times

Matrix multiplication

```
for kij = 0:8 (time, renew all the weights in registers)
   for out_ch = 0:63 (col #)
      for in_ch = 0:63 (row #)
         for nij = 0:255 (time for horizontal input)
              psum(out_ch, kij, nij)  += w(out_ch, in_ch, kij) * x(in_ch, nij)
```

Accumulation (SFU)

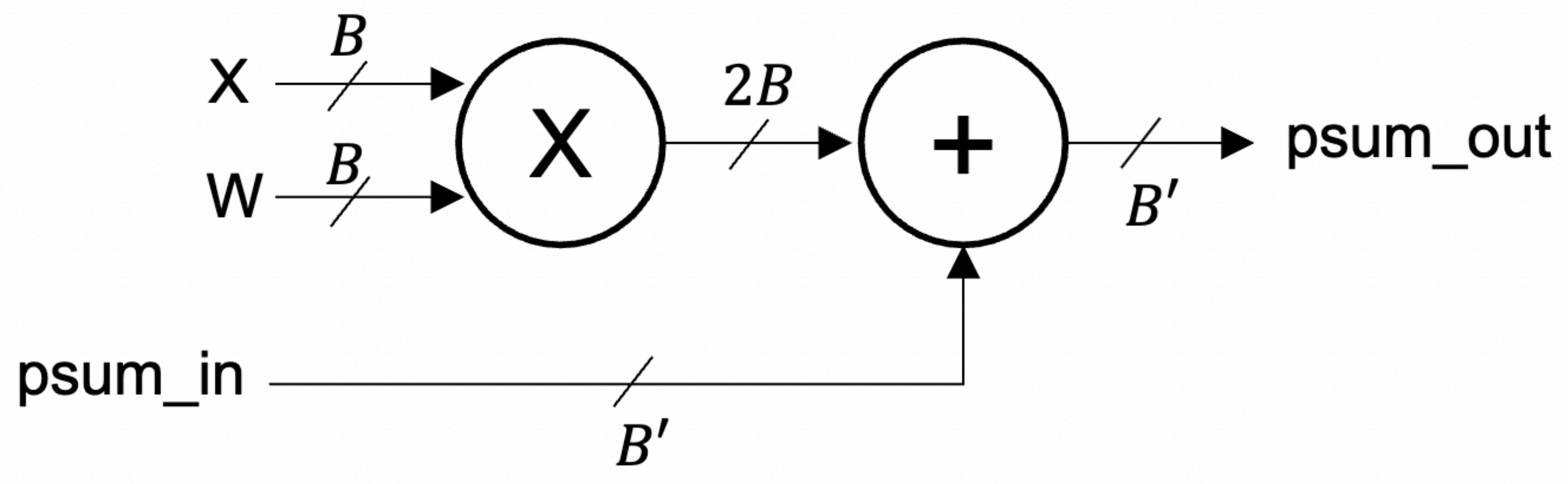
```
for nij = 0:255 (output index)
   for kij = 0:8 (time)
      output (out_ch, nij) += psum(out_ch, kij, nij')
```

- Note nij' = f(nij, kij) is shifted index of nij for conv.
- Matmult and acc can be processed simultaneously.

Key information

- GPU is good at parallel computing such as matrix addition and multiplication compare to CPU.
- Create Python library and bind with C++ using Pybind.
- Different convolution layer size and kernel size will be tested on CPU and GPU.
- MAC (Multiply and accumulation) function compare to separate addition and multiplication.
- #pragma unroll used to accelerate the process.

$$sum = \sum_{i=1}^{N} X_i W_i$$



Result

```
H:\Documents\Python2Cuda_example\Python2Cuda_example\py_src>py run_1tiny.py
nij = 4
input channel = 1
output channel = 1
kij = 1
Pass
Python Time: 0.009499999999995623 ms
GPU Time: 3.6980000000000346 ms
GPU Time(MAC): 3.0935000000000027 ms
```

```
H:\Documents\Python2Cuda_example\Python2Cuda_example\py_src>py run_2small.py
nij = 16
input channel = 4
output channel = 4
kij = 4
Pass
Python Time: 0.4761999999999822 ms
GPU Time: 4.990099999999997 ms
GPU Time(MAC): 3.7895000000000001 ms
```

```
H:\Documents\Python2Cuda_example\Python2Cuda_example\py_src>py run_3middle.py
nij = 64
input channel = 16
output channel = 16
kij = 4
Pass
Python Time: 24.559599999999985 ms
GPU Time: 5.782199999999996 ms
GPU Time(MAC): 3.9132000000000056 ms
```

Result

```
H:\Documents\Python2Cuda_example\Python2Cuda_example\py_src>py run_4big.py
nij = 256
input channel = 64
output channel = 64
kij = 9
Pass
Python Time: 3291.3327 ms
GPU Time: 10.460099999999972 ms
GPU Time(MAC): 5.849099999999829 ms
```

```
H:\Documents\Python2Cuda_example\Python2Cuda_example\py_src>py run_5huge.py
nij = 1024
input channel = 64
output channel = 64
kij = 9
Pass
Python Time: 12948.4131 ms
GPU Time: 11.343200000000664 ms
GPU Time(MAC): 7.578599999998659 ms
```

Conclusion

- While some edge cases (when the input matrix is very small) show that CPU can be faster than GPU, GPU has much better performance on huge matrix, which is quite normal in ML.
- MAC is always faster compare to separate multiplication and addition (only 65-75% time).
- Awake GPU do need time, around 70-90 millisecond. I put a test function before GPU calculation in order to awake GPU.