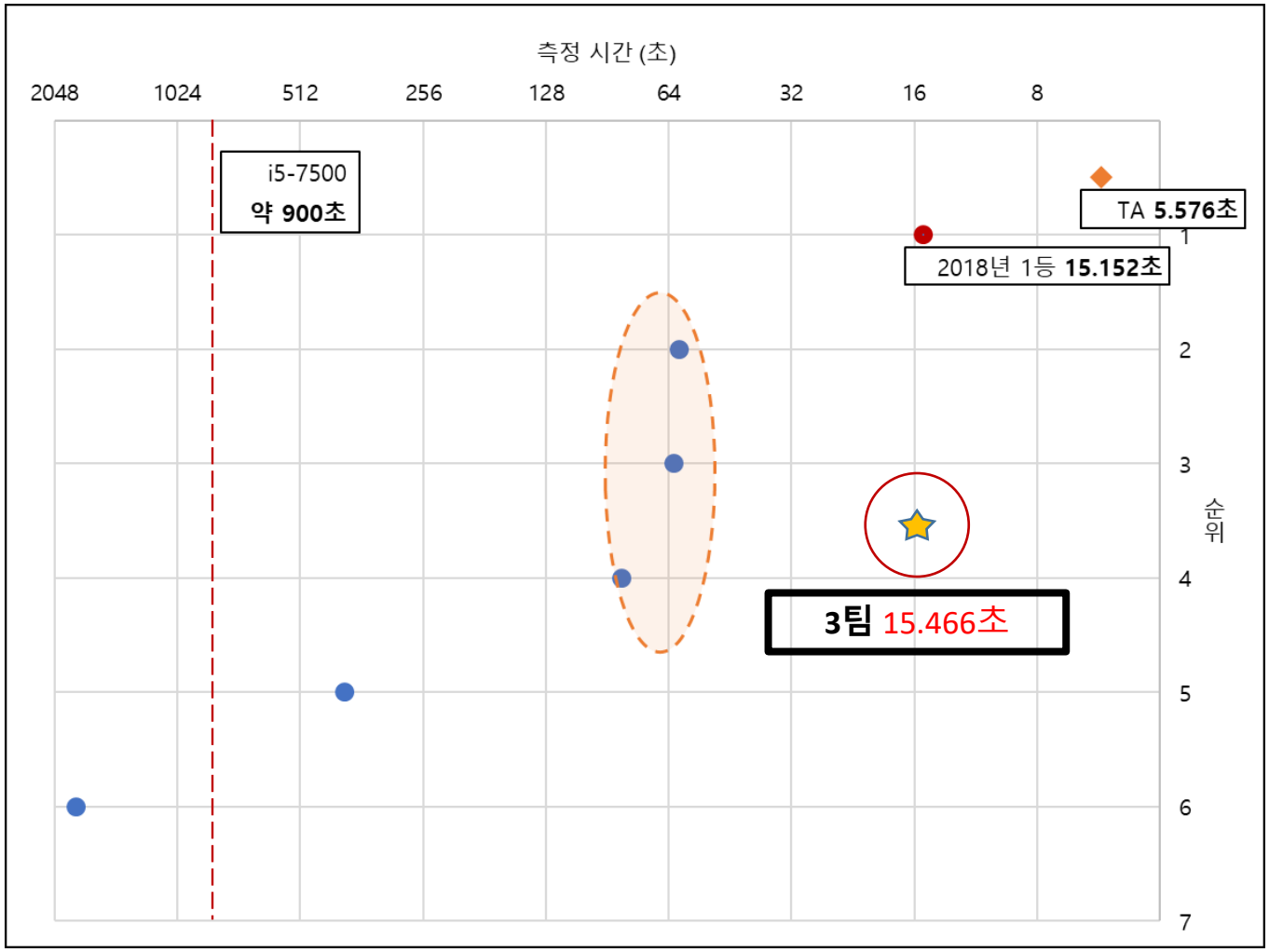
CNN 기반 이미지 분류

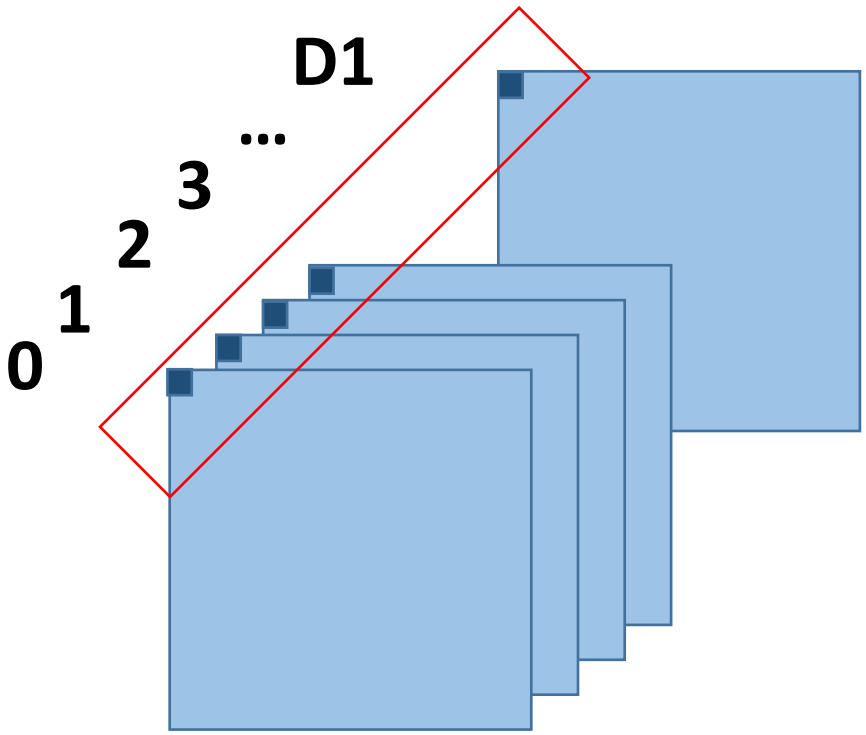
15011044 송창석
15011047 최태호
15011050 조한국




```
Elapsed time: 15.466000 sec
Results are same.
```

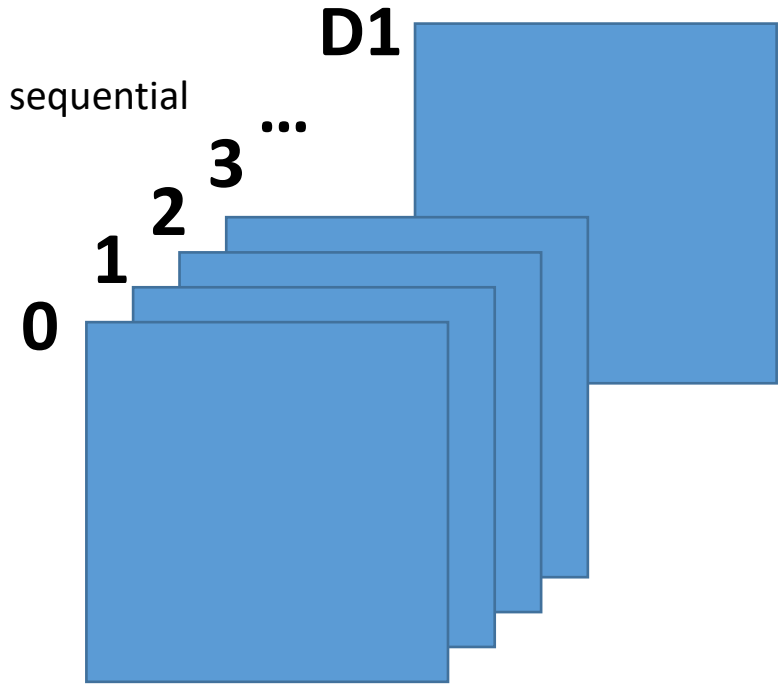
Convolution

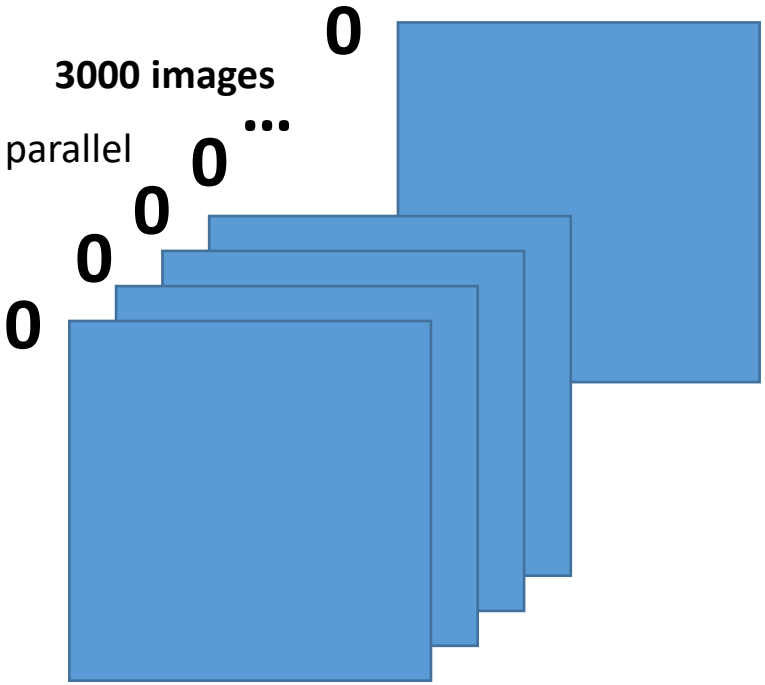
Work item



Convolution

Input order

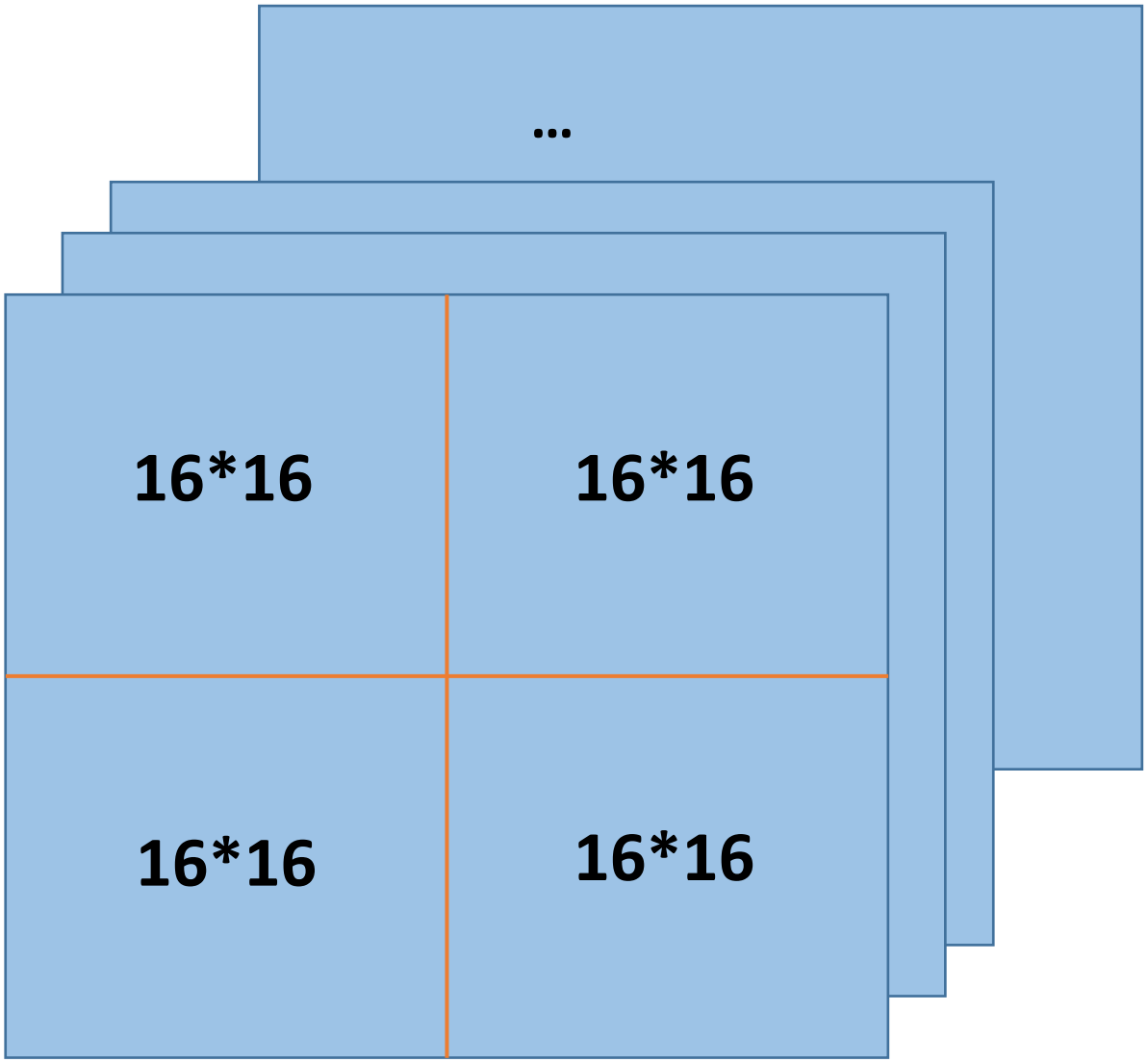




Convolution **{16, 16, 1}**

32*32 image

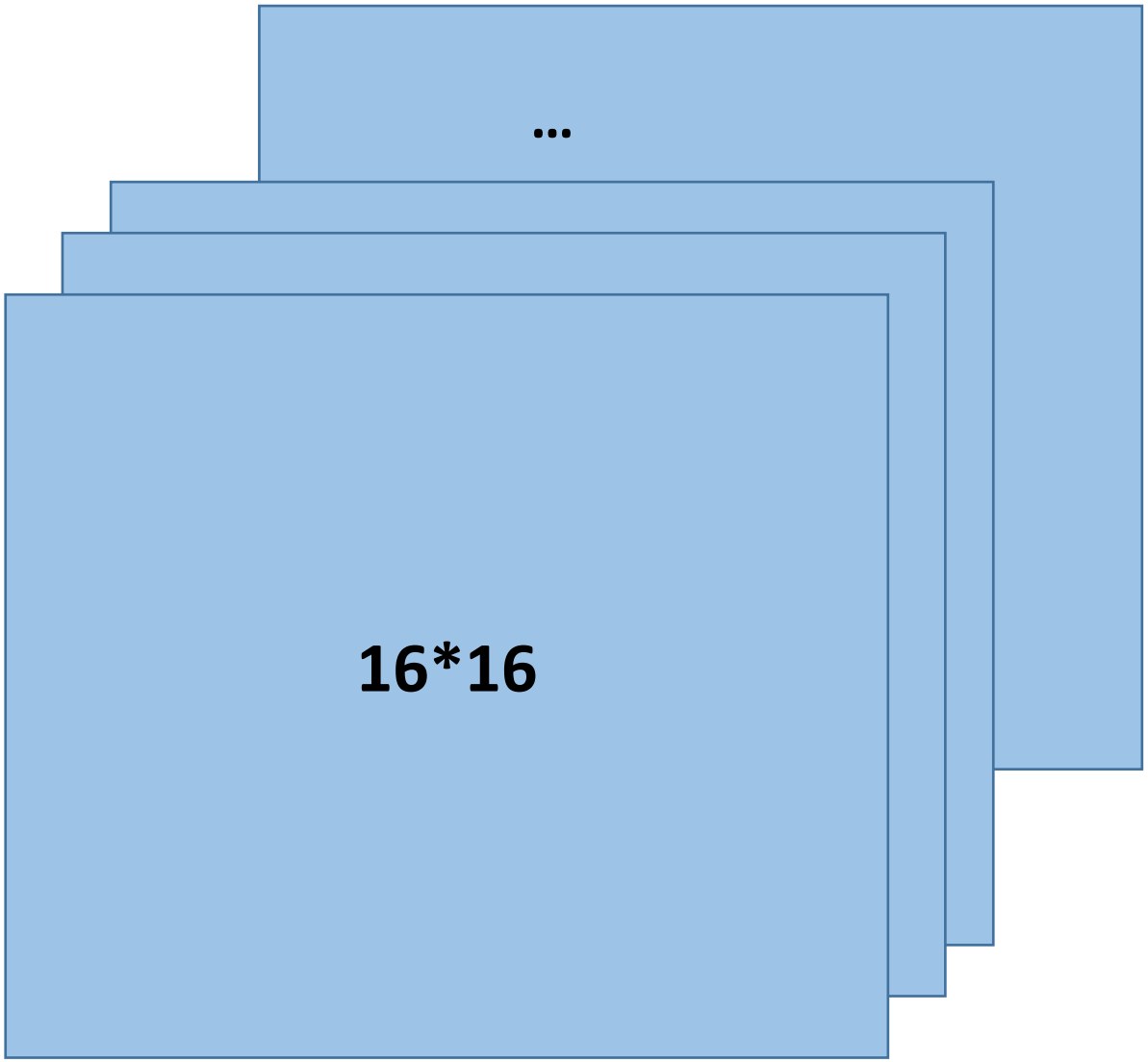
3000 images use same filter



Convolution **{16, 16, 1}**

16*16 image

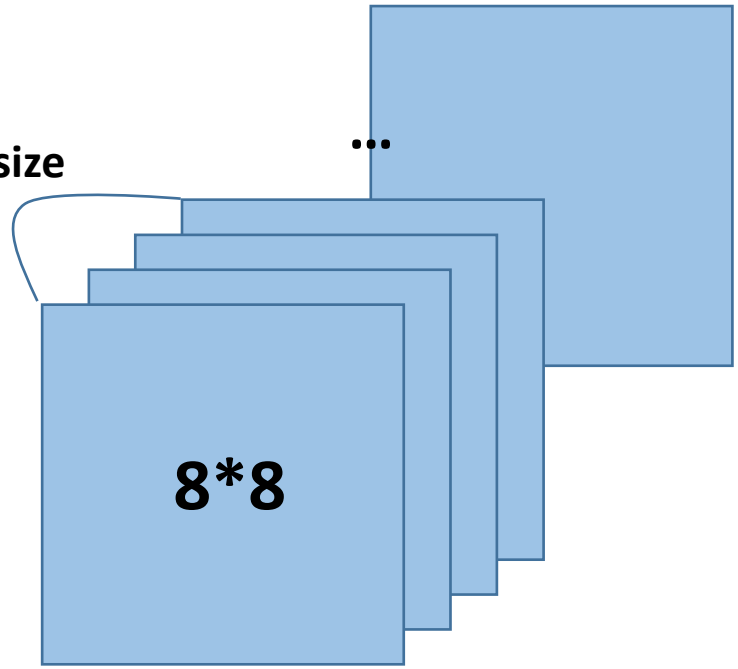
3000 images use same filter



Convolution **{8, 8, 4}**

8*8 image

Max work group size
256/(8*8) = 4



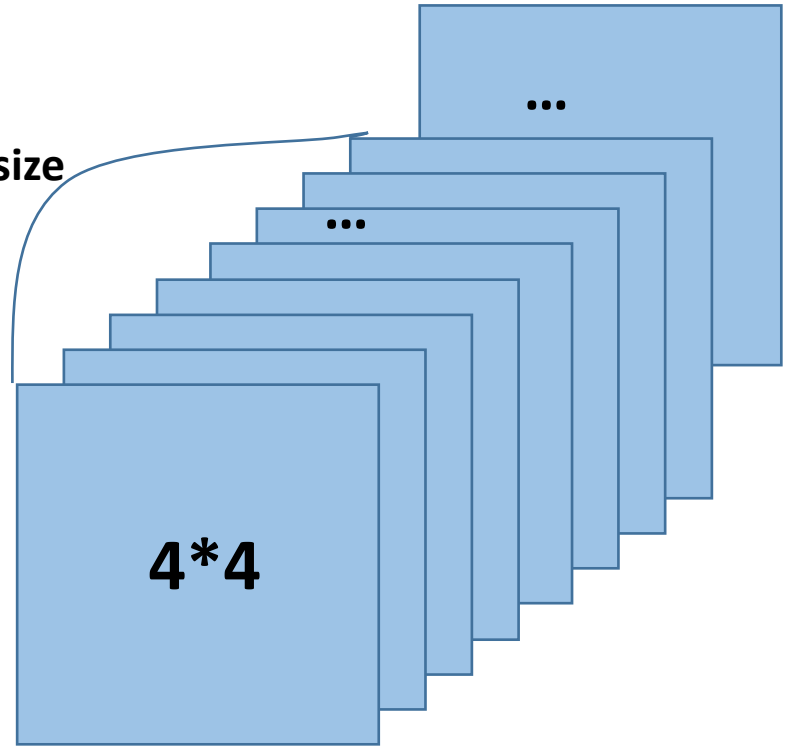
3000 images
use same filter

Work group size: 4 channel
3000%4 == 0

Convolution **{4, 4, 15}**

4*4 image

Max work group size
256/(4*4) = 16



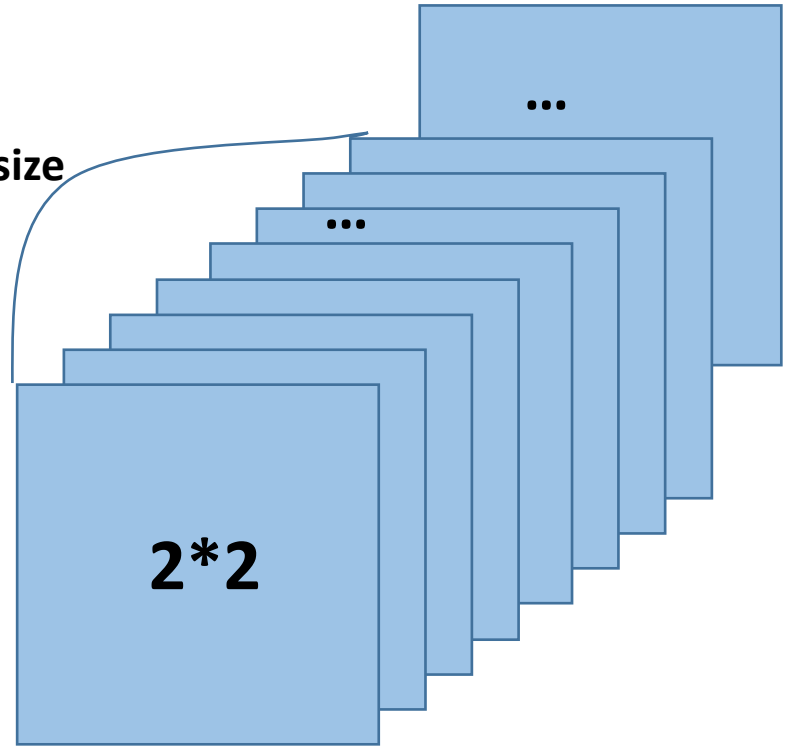
3000 images
use same filter

Max Work group size: 16 channel
3000%16 != 0
So set work group size 15

Convolution **{2, 2, 60}**

2*2 image

Max work group size
256/(2*2) = 64



3000 images
use same filter

Max Work group size: 16 channel
3000%64 != 0
So set work group size 60

Convolution

```
for (int m = 0; m < D1; m++)
{
    if(m%2 == 0)
    {
        if(l_size <= 2 && l_c < 9)
            lFilter1[l_c] = filter[l_c + m*9 + t1];
        else if(l_size > 2 && (l_i*3 + l_j) < 9)
            lFilter1[l_i*3 + l_j] = filter[l_i*3+l_j + m*9 + t1];
    } else
    {
        if(l_size <= 2 && l_c < 9)
            lFilter2[l_c] = filter[l_c + m*9 + t1];
        else if(l_size > 2 && (l_i*3 + l_j) < 9)
            lFilter2[l_i*3 + l_j] = filter[l_i*3+l_j + m*9 + t1];
    }
    barrier(CLK_LOCAL_MEM_FENCE);

    for (int k = 0; k < 3; k++)
    {
        for (int l = 0; l < 3; l++)
        {
            int x = i + k - 1;
            int y = j + l - 1;
            if (x >= 0 && x < N && y >= 0 && y < N)
            {
                if(m%2 == 0)
                    sum += input[x * N + y + m*image_size + t2]* lFilter1[k * 3 + l];
                else
                    sum += input[x * N + y + m*image_size + t2]* lFilter2[k * 3 + l];
            }
        }
    }
}
output[i * N + j + (c/3000)*image_size + (c%3000)*D2*image_size] = ReLU(sum + biases[c/3000]);
```

Convolution local memory

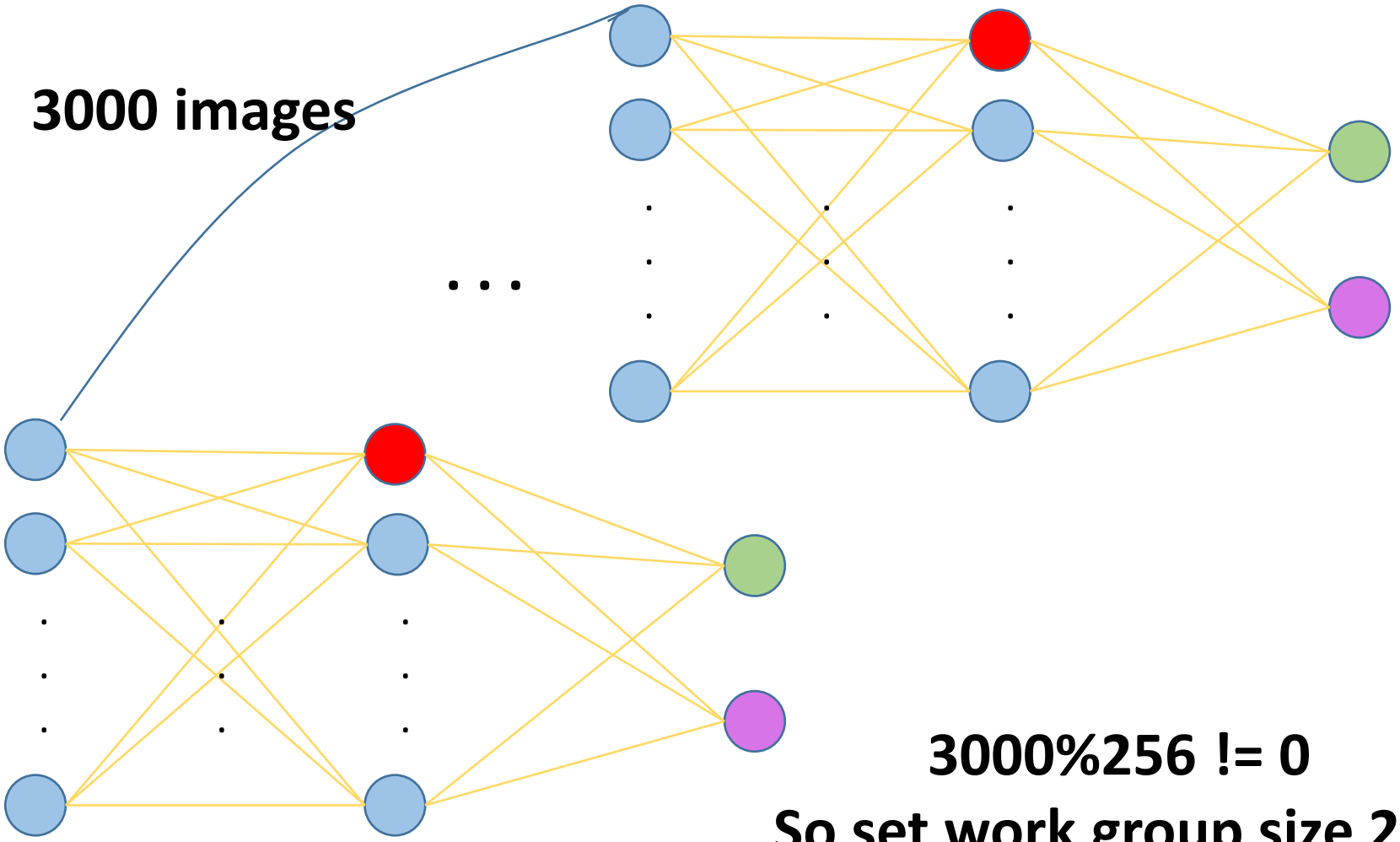
```
for (int m = 0; m < D1; m++)
{
    if(m%2 == 0)
    {
        if(l_size <= 2 && l_c < 9)
            lFilter1[l_c] = filter[l_c + m*9 + t1];
        else if(l_size > 2 && (l_i*3 + l_j) < 9)
            lFilter1[l_i*3 + l_j] = filter[l_i*3+l_j + m*9 + t1];
    } else
    {
        if(l_size <= 2 && l_c < 9)
            lFilter2[l_c] = filter[l_c + m*9 + t1];
        else if(l_size > 2 && (l_i*3 + l_j) < 9)
            lFilter2[l_i*3 + l_j] = filter[l_i*3+l_j + m*9 + t1];
    }
    barrier(CLK_LOCAL_MEM_FENCE);
```

Convolution 연산

```
for (int m = 0; m < D1; m++)
{
    ////////////////////////////////////////////////////
        로컬 필터 할당 부분 생략 (lFilter1 or lFilter2)
    ////////////////////////////////////////////////////

    for (int k = 0; k < 3; k++)
    {
        for (int l = 0; l < 3; l++)
        {
            int x = i + k - 1;
            int y = j + l - 1;
            if (x >= 0 && x < N && y >= 0 && y < N)
            {
                if(m%2 == 0)
                    sum += input[x * N + y + m*image_size + t2]* lFilter1[k * 3 + l];
                else
                    sum += input[x * N + y + m*image_size + t2]* lFilter2[k * 3 + l];
            }
        }
    }
}
output[i * N + j + (c/3000)*image_size + (c%3000)*D2*image_size] = ReLU(sum + biases[c/3000]);
```

Full Connection **{1, 250}**



3000%256 != 0

So set work group size 250

Full Connection

```
void fc(__global float* input_neuron, __global float* output_neuron, __global float* weights, __glob
{
    int j = get_global_id(0);           //0 ~ (M - 1) (0,0) (1,0) (M-1, 0) 512번 동일한 input
    int k = get_global_id(1);           //0 ~ (num_images)

    int l_j = get_local_id(0);
    int l_k = get_local_id(1);          //0 ~ 250-1
    int i = 0;
    float sum = 0.0;

    if(l_k < 250)
    {
        lWeight[l_k] = weights[l_k + j*N];
        lWeight[l_k+250] = weights[(l_k+250) + j*N];
        if(l_k < 12)
        {
            lWeight[l_k + 500] = weights[l_k+500 + j*N];
        }
    }

    barrier(CLK_LOCAL_MEM_FENCE);

    for (i = 0; i < N; i++)
    {
        sum += input_neuron[i + k*N] * lWeight[i];
    }
    output_neuron[j + k*M] = ReLU(sum + biases[j]);
}
```

Some technics

Loop invariant code

```
int image_size = N*N;
int t1 = (c/3000)*D1*9, t2 = (c%3000)*image_size*D1;
```

```
int image_size = N*N;
int t1 = 4*image_size*c;
int t2 = 2*N;
```

Loop unrolling

```
float pixel = input[t1 + t2 * (i * 2) + j * 2];
max = (max > pixel) ? max : pixel;
pixel = input[t1 + t2 * (i * 2) + j * 2 + 1];
max = (max > pixel) ? max : pixel;
pixel = input[t1 + t2 * (i * 2 + 1) + j * 2];
max = (max > pixel) ? max : pixel;
pixel = input[t1 + t2 * (i * 2 + 1) + j * 2 + 1];
max = (max > pixel) ? max : pixel;
```

Some technics

Set kernel argument properly

Convolution 1-1

```
errNum = clSetKernelArg(conv_kernel, 0, sizeof(cl_mem), &conv_memObjects[0])        //g_input
errNum |= clSetKernelArg(conv_kernel, 1, sizeof(cl_mem), &conv_memObjects[1]);      //g_output
```

Convolution 1-2

```
errNum = clSetKernelArg(conv_kernel, 0, sizeof(cl_mem), &conv_memObjects[1]);       //g_input
errNum |= clSetKernelArg(conv_kernel, 1, sizeof(cl_mem), &conv_memObjects[0]);      //g_output
```

QnA