

CNN 최종 발표

사슴조

14010976 이현식

14010990 김정혁

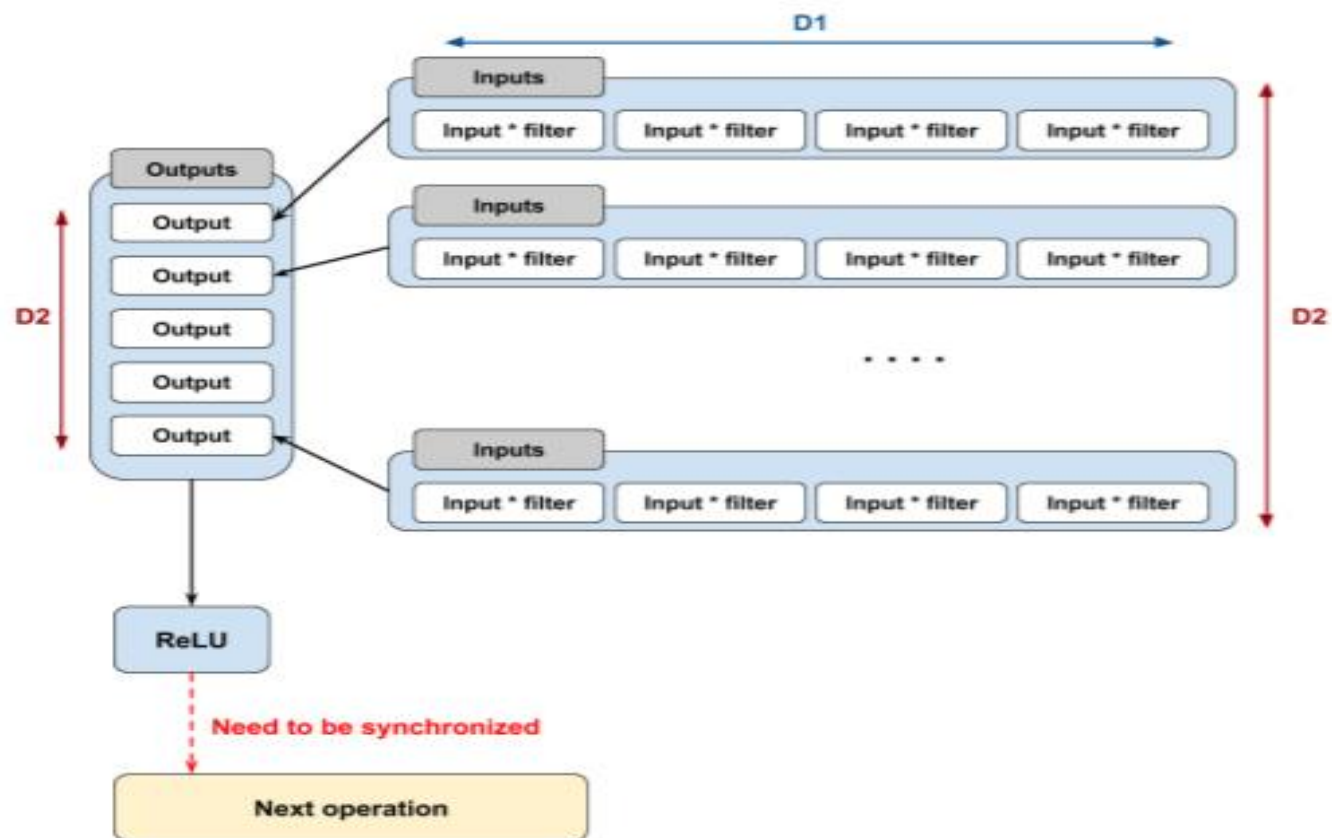
16011033 위 진

목차

1. 동작 설명
2. 최적화 포인트
3. 문제 발생
4. 해결 방안
5. 최적화 시도
6. 최적화 성능 평가
7. 역할 분배
8. 느낀점



동작 설명



최적화 포인트

- Convolution layer에서의 시간이 가장 많이 걸림

- $N \times N \times D1 \times D2$ 연산에서 $D1 \times D2$ 를 병렬로 처리
-> $N \times N$ 만 순차로 처리

```
static void convolution3x3(float *input, float *output, float *filter, int N) {
    int i, j, k, l;
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            float sum = 0;
            for (k = 0; k < 3; k++) {
                for (l = 0; l < 3; l++) {
                    int x = i + k - 1;
                    int y = j + l - 1;
                    if (x >= 0 && x < N && y >= 0 && y < N)
                        sum += input[x * N + y] * filter[k * 3 + l];
                }
            }
            output[i * N + j] += sum;
        }
    }
}
```

```
static void convolution_layer(float *inputs, float *outputs, float *filters, float *biases, int D2, int D1, int N) {
    int i, j;

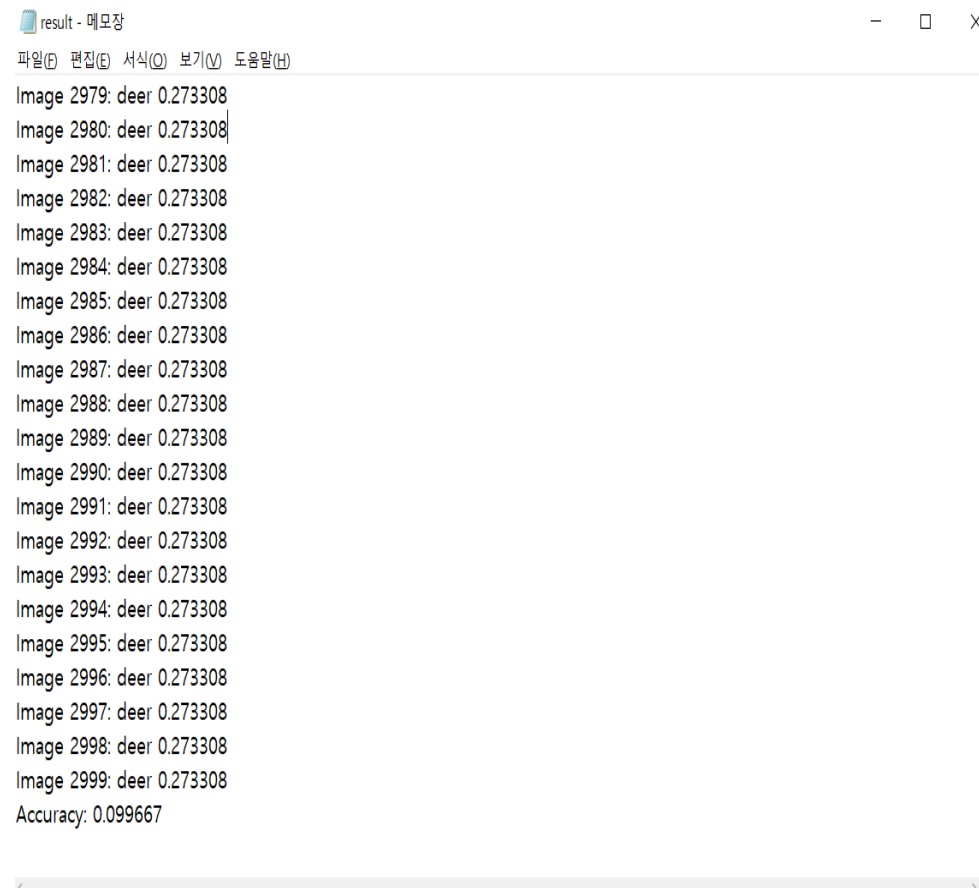
    memset(outputs, 0, sizeof(float) * N * N * D2);

    for (j = 0; j < D2; j++) {
        for (i = 0; i < D1; i++) {
            float *input = inputs + N * N * i;
            float *output = outputs + N * N * j;
            float *filter = filters + 3 * 3 * (j * D1 + i);
            convolution3x3(input, output, filter, N);
        }
    }

    for (i = 0; i < D2; i++) {
        float *output = outputs + N * N * i;
        float bias = biases[i];
        for (j = 0; j < N * N; j++) {
            output[j] = ReLU(output[j] + bias);
        }
    }
}
```

문제 발생

- 결과 값으로 사슴만 출력됨
- output에 D1개의 $N * N$ 배열을 더하는 과정에서 input끼리 종속성이 발생



```
result - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
Image 2979: deer 0.273308
Image 2980: deer 0.273308
Image 2981: deer 0.273308
Image 2982: deer 0.273308
Image 2983: deer 0.273308
Image 2984: deer 0.273308
Image 2985: deer 0.273308
Image 2986: deer 0.273308
Image 2987: deer 0.273308
Image 2988: deer 0.273308
Image 2989: deer 0.273308
Image 2990: deer 0.273308
Image 2991: deer 0.273308
Image 2992: deer 0.273308
Image 2993: deer 0.273308
Image 2994: deer 0.273308
Image 2995: deer 0.273308
Image 2996: deer 0.273308
Image 2997: deer 0.273308
Image 2998: deer 0.273308
Image 2999: deer 0.273308
Accuracy: 0.099667
```

해결 방안

- $N \times N \times D1 \times D2$ 연산에서 $N \times N \times D2$ 를 병렬로 처리
-> D1만 순차로 처리함으로 종속성 해결

```
__kernel void convolution(
    __global float *inputs,
    __global float *outputs,
    __global float *filters,
    int D1, int D2, int N)
{
    int index = get_global_id(0); //처리하고자 하는 행 (0 ~ N * N * D1)
    int output_index = get_global_id(1); //0 ~ (D2 - 1)

    int index_of_matrix = index % (N * N); //하나의 이미지 내에서의 인덱스
    int x_index_of_matrix = index_of_matrix % N; //처리하고자 하는 곳의 열 좌표
    int y_index_of_matrix = index_of_matrix / N; //처리하고자 하는 곳의 행 좌표
    //좌표는 (y,x) 로 표현한다.

    int i, j, iter; //for문 용
    float sum = 0.0;

    for (iter = 0; iter < D1; iter++)
    {
        for (j = 0; j < 3; j++)
        {
            for (i = 0; i < 3; i++)
            {
                ((y_index_of_matrix + (j - 1)) >= 0) && (y_index_of_matrix + (j - 1) < N) && (x_index_of_matrix + (i - 1) >= 0) && (x_index_of_matrix + (i - 1) < N) && (sum += inputs[N * N * iter +
                    (y_index_of_matrix + (j - 1)) * N + x_index_of_matrix + (i - 1)] * filters[output_index * 3 * 3 * D1 + iter * 3 * 3 + j * 3 + i]);

                //컨볼루션 구현
            }
        }
    }

    outputs[(N * N) * output_index + y_index_of_matrix * N + x_index_of_matrix] = sum;
}
```

최적화 시도

- Local memory 사용 -> memory를 한번에 넣으려고 시도함

```
__kernel void convolution(__global float *inputs, __local float *l_inputs, __global float *outputs, __global float *filters, __local float *l_filters, int D1, int D2, int N)
{
    int index = get_global_id(0); //처리하고자 하는 좌표 (0 ~ N*N-1)
    int output_index = get_global_id(1); //출력은 몇 번째 이미지에 해야 하는가를 나타냄 (0 ~ D2)

    int x_index_of_matrix = index % N; //처리하고자 하는 곳의 열 좌표
    int y_index_of_matrix = index / N; //처리하고자 하는 곳의 행 좌표
    //좌표는 (y,x)로 표현한다.

    int i, j, iter; //for문 용
    float sum = 0.0;

    for (iter = 0; iter < D1; iter++)
    {
        for (j = 0; j < 3; j++)
        {
            for (i = 0; i < 3; i++)
            {
                if ((y_index_of_matrix + (j - 1)) >= 0 && (y_index_of_matrix + (j - 1)) < N && (x_index_of_matrix + (i - 1)) >= 0 && (x_index_of_matrix + (i - 1)) < N)
                {
                    l_inputs[(iter * 3 * 3) + (j * 3) + i] = inputs[(iter * N * N) + ((y_index_of_matrix + (j - 1)) * N) + (x_index_of_matrix + (i - 1))];
                }
                else
                {
                    l_inputs[(iter * 3 * 3) + (j * 3) + i] = 0;
                }
            }
        }

        for (iter = 0; iter < D1; iter++)
        {
            for (j = 0; j < 3; j++)
            {
                for (i = 0; i < 3; i++)
                {
                    l_filters[(iter * 3 * 3) + (j * 3) + i] = filters[(output_index * D1 * 3 * 3) + (iter * 3 * 3) + (j * 3) + i];
                }
            }
        }

        barrier(CLK_LOCAL_MEM_FENCE);

        sum = 0.0;
        for (iter = 0; iter < D1; iter++)
        {
            for (j = 0; j < 3; j++)
            {
                for (i = 0; i < 3; i++)
                {
                    sum = sum + l_inputs[(iter * 3 * 3) + (j * 3) + i] * l_filters[(iter * 3 * 3) + (j * 3) + i];
                    //컨볼루션 구현
                }
            }
        }

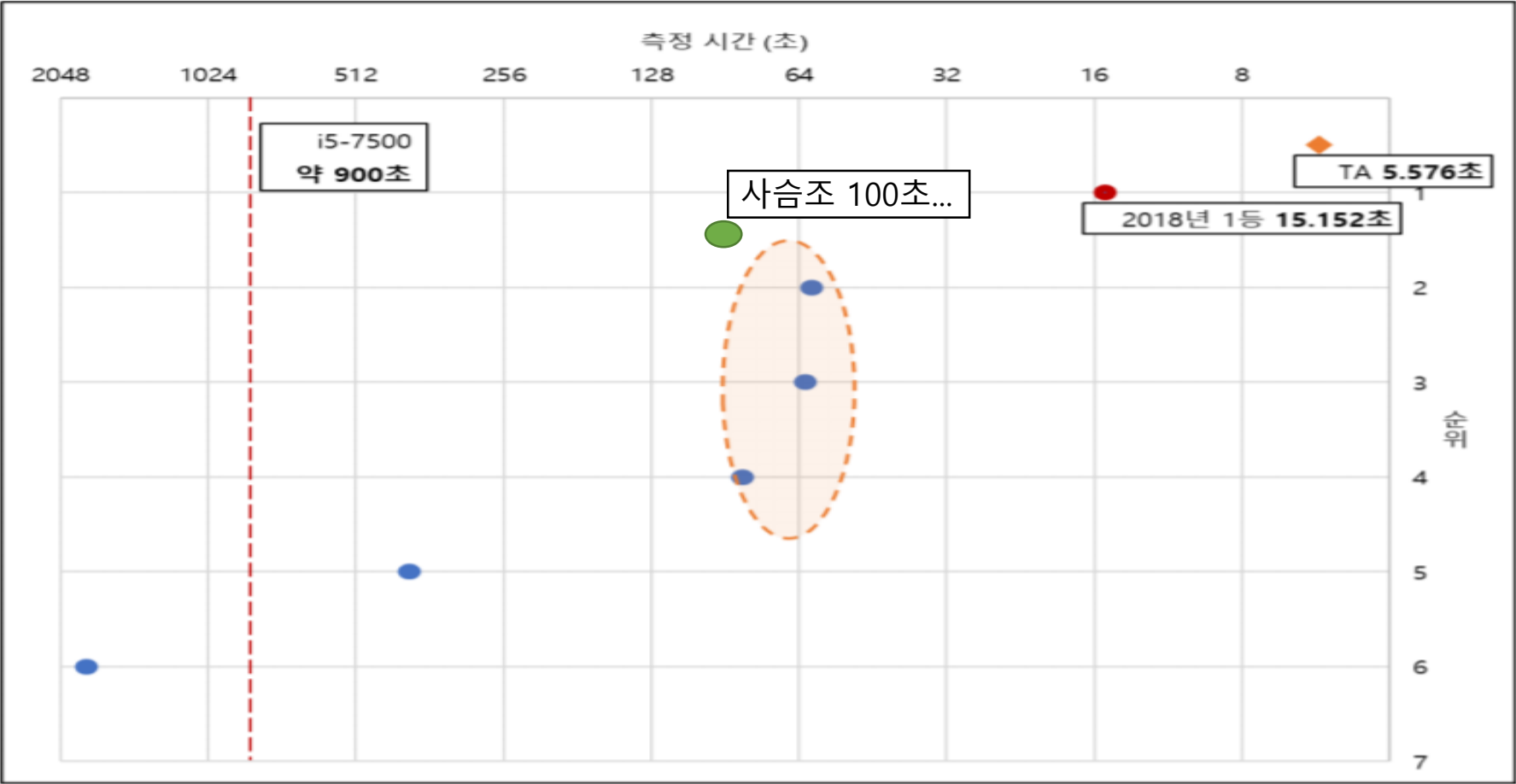
        outputs[(output_index * (N * N)) + (y_index_of_matrix * N) + (x_index_of_matrix)] = sum;
    }
}
```

최적화 시도

- Unrolling 기법 사용 -> padding의 문제로 불가능

```
if ((y_index_of_matrix >= 1) && (y_index_of_matrix < N - 1) && (x_index_of_matrix >= 1) && (x_index_of_matrix < N - 1)) {  
    sum += inputs[N * N * iter + (y_index_of_matrix + (-1 - 1)) * N + x_index_of_matrix + (-1 - 1)] * filters[output_index * 3 * 3 * D1 + iter * 3 * 3 + (-1 * 3) - 1];  
    sum += inputs[N * N * iter + (y_index_of_matrix + (-1 - 1)) * N + x_index_of_matrix + (0 - 1)] * filters[output_index * 3 * 3 * D1 + iter * 3 * 3 + (-1 * 3) + 0];  
    sum += inputs[N * N * iter + (y_index_of_matrix + (-1 - 1)) * N + x_index_of_matrix + (1 - 1)] * filters[output_index * 3 * 3 * D1 + iter * 3 * 3 + (-1 * 3) + 1];  
  
    sum += inputs[N * N * iter + (y_index_of_matrix + (0 - 1)) * N + x_index_of_matrix + (-1 - 1)] * filters[output_index * 3 * 3 * D1 + iter * 3 * 3 + (0 * 3) - 1];  
    sum += inputs[N * N * iter + (y_index_of_matrix + (0 - 1)) * N + x_index_of_matrix + (0 - 1)] * filters[output_index * 3 * 3 * D1 + iter * 3 * 3 + (0 * 3) + 0];  
    sum += inputs[N * N * iter + (y_index_of_matrix + (0 - 1)) * N + x_index_of_matrix + (1 - 1)] * filters[output_index * 3 * 3 * D1 + iter * 3 * 3 + (0 * 3) + 1];  
  
    sum += inputs[N * N * iter + (y_index_of_matrix + (1 - 1)) * N + x_index_of_matrix + (-1 - 1)] * filters[output_index * 3 * 3 * D1 + iter * 3 * 3 + (1 * 3) - 1];  
    sum += inputs[N * N * iter + (y_index_of_matrix + (1 - 1)) * N + x_index_of_matrix + (0 - 1)] * filters[output_index * 3 * 3 * D1 + iter * 3 * 3 + (1 * 3) + 0];  
    sum += inputs[N * N * iter + (y_index_of_matrix + (1 - 1)) * N + x_index_of_matrix + (1 - 1)] * filters[output_index * 3 * 3 * D1 + iter * 3 * 3 + (1 * 3) + 1];  
}
```


최적화 성능 평가



역할 분배

1. 시퀀셜 코드를 돌려보고 최적화 포인트 선정
2. Convolution layer 병렬화
3. Local memory 문제
4. Image 문제

느낀점

이현식: global memory와 local memory 그리고 kernel영역과 host영역 사이의 상관관계를 이해하고 실제로 프로젝트로 적용시키는게 너무 복잡해서 힘들었지만, 병렬 처리를 어떤식으로 코딩해야 되는지 전반적인 느낌을 알게 되었다. 병렬 처리는 진입장벽이 높다고 생각하고, 그만큼 열심히 하면 경쟁력이 있는 분야일 것 같다고 느꼈다.

김정혁: 코드를 최적화할 때 나아가야할 방향을 제대로 잡아야 한다는 것을 절실히 느꼈다. 더 나은 방향이 있음에도 불구하고 다른 방향으로 나아가고 있었다면 그 다른, 더 나은 방향으로 가기는 정말 힘들다는 것을 느꼈다. 한편으로는 사소하지만 본인에게는 큰 산들을 넘어가면서 스스로의 발전이 있었던 것 같다.

위진: 최적화를 하는 데에 있어서 성능 향상을 원해서 사용했던 기법들이 오히려 역효과를 불러냈고 일부만 사용하고 발상을 전환한 방법으로 종속성 문제를 해결하여 좋은 결과를 얻었다. 기법의 선택과 포기의 밸런스를 맞추는 것이 정말 중요하다고 느꼈다.
(사슴이라는 동물이 미워졌다..)

Thank you .