CNN 기반 이미지 분류

박 기호

2019년 2학기

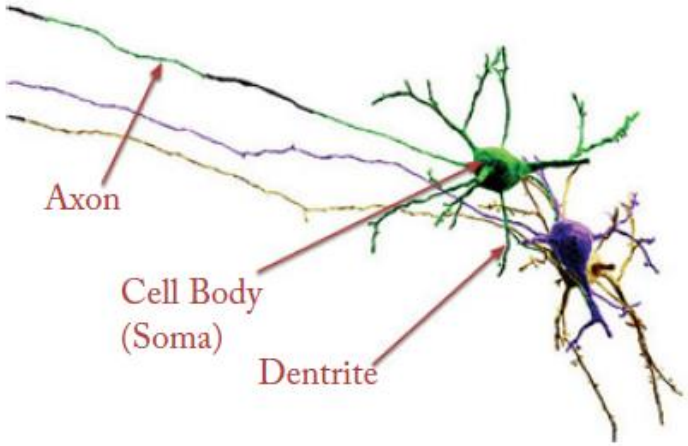
세종대학교 컴퓨터공학과

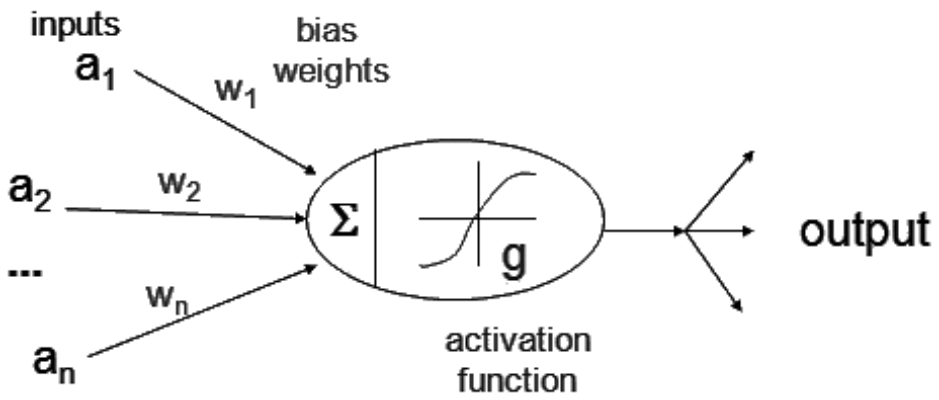
프로젝트 개요

- 인공 신경망을 활용하여 이미지를 분류하는 예제
  - 사전 학습된 모델을 사용해 이미지 분류를 실행
- CNN 모델(VGG16)을 활용해 CIFAR-10 데이터셋에 대한 테스트 수행
- GPU를 통한 가속을 수행하며, 결과 값을 유지하되 수행시간이 짧은 팀 위주로 점수 부여

인공 신경망의 개요

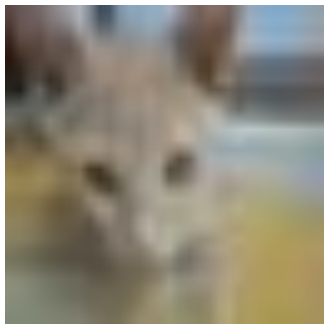
❖ 뉴런의 동작을 모사한 구조 및 수행

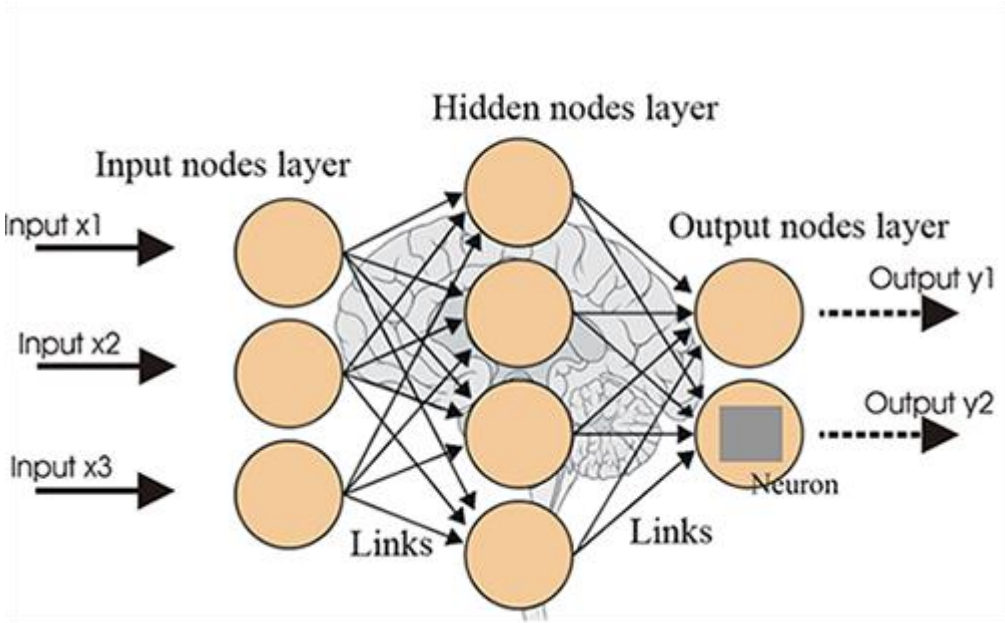




인공 신경망의 개요

❖ 인공 신경망의 인식 과정



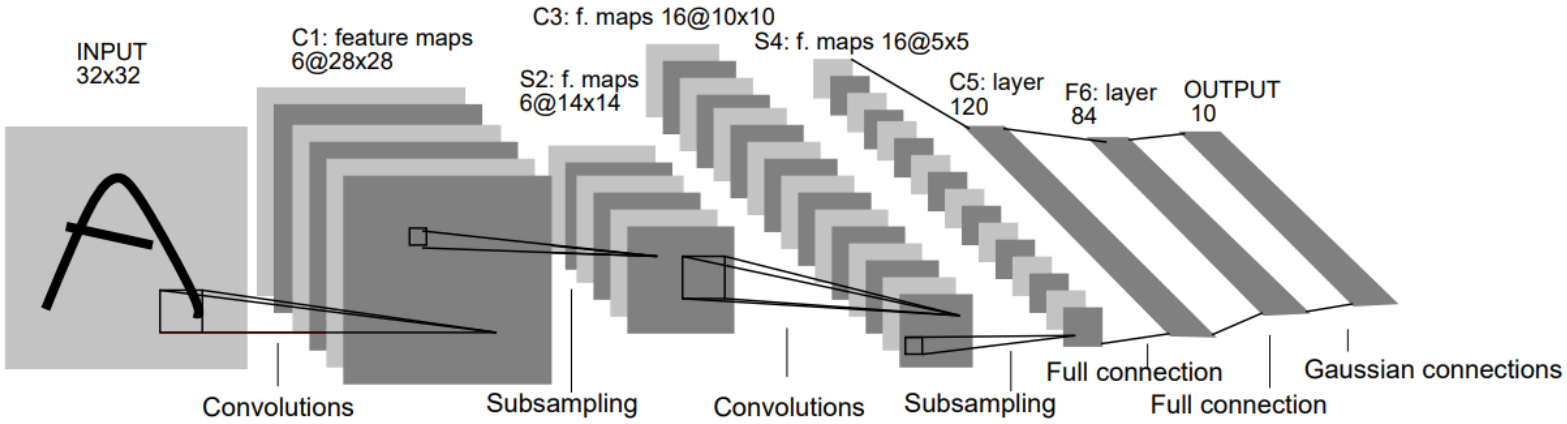


Cat

CNN이란

Convolutional Neural Networks

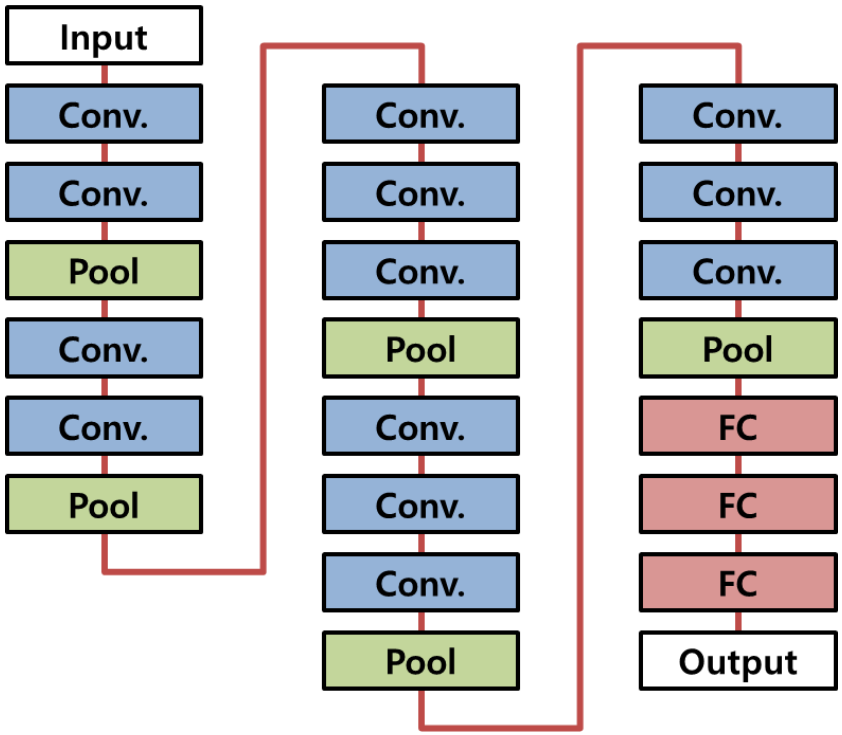
- 컨볼루션 연산을 통해 데이터의 특징을 추출하여 학습하는 신경망



CNN 모델의 구성

- CNN 모델 VGG16을 CIFAR10 데이터셋에 맞게 가공

- 모델의 구조



소스코드

- **CNN 레이어 구조**
  - **function cnn( ) in cnn.c**
  - **VGG16**
    - convolution layer
    - pooling layer
    - fc layer

```
// run network
for(int i = 0; i < num_images; ++i)
{
    float *image = images + i * 3 * 32 * 32;

    convolution_layer(image, c1_1, w1_1, b1_1, 64, 3, 32);
    convolution_layer(c1_1, c1_2, w1_2, b1_2, 64, 64, 32);
    pooling_layer(c1_2, p1, 64, 16);

    convolution_layer(p1, c2_1, w2_1, b2_1, 128, 64, 16);
    convolution_layer(c2_1, c2_2, w2_2, b2_2, 128, 128, 16);
    pooling_layer(c2_2, p2, 128, 8);

    convolution_layer(p2, c3_1, w3_1, b3_1, 256, 128, 8);
    convolution_layer(c3_1, c3_2, w3_2, b3_2, 256, 256, 8);
    convolution_layer(c3_2, c3_3, w3_3, b3_3, 256, 256, 8);
    pooling_layer(c3_3, p3, 256, 4);

    convolution_layer(p3, c4_1, w4_1, b4_1, 512, 256, 4);
    convolution_layer(c4_1, c4_2, w4_2, b4_2, 512, 512, 4);
    convolution_layer(c4_2, c4_3, w4_3, b4_3, 512, 512, 4);
    pooling_layer(c4_3, p4, 512, 2);

    convolution_layer(p4, c5_1, w5_1, b5_1, 512, 512, 2);
    convolution_layer(c5_1, c5_2, w5_2, b5_2, 512, 512, 2);
    convolution_layer(c5_2, c5_3, w5_3, b5_3, 512, 512, 2);
    pooling_layer(c5_3, p5, 512, 1);

    fc_layer(p5, fc1, w1, b1, 512, 512);
    fc_layer(fc1, fc2, w2, b2, 512, 512);
    fc_layer(fc2, fc3, w3, b3, 10, 512);

    softmax(fc3, 10);

    labels[i] = find_max(fc3, 10);
    confidences[i] = fc3[labels[i]];
}
```

소스코드

- **CNN Weights**
  - Filters, Bias
  - Network 변수에 저장

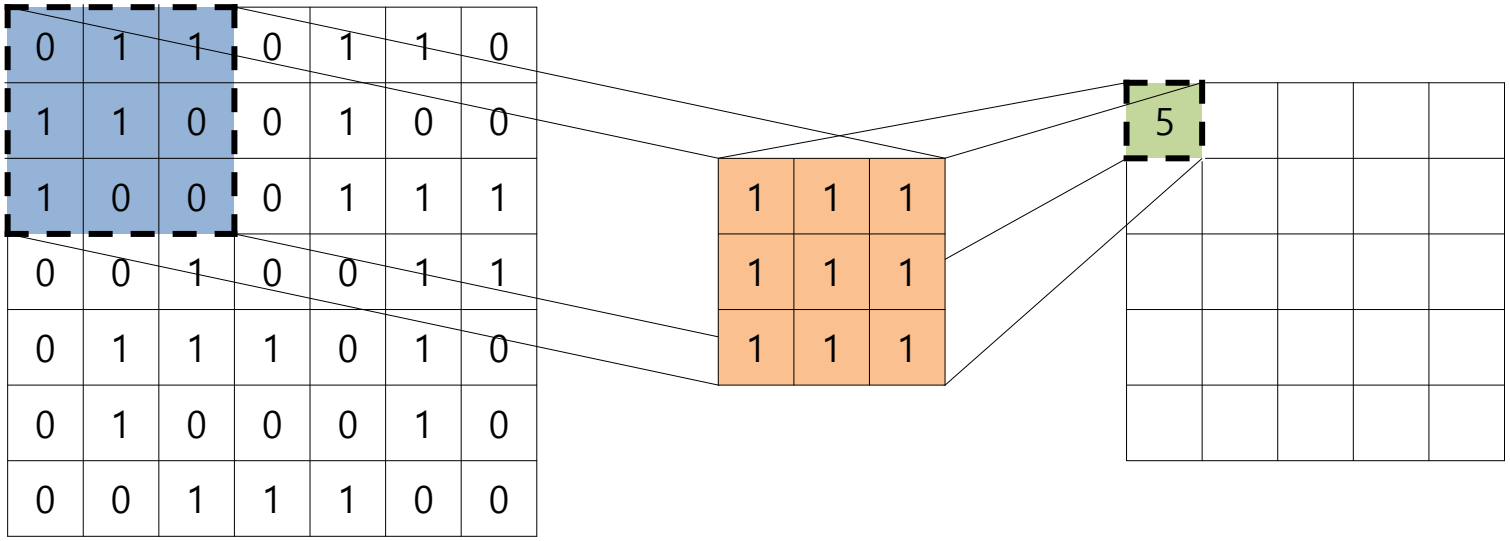
```
w1_1 = network[0]; b1_1 = network[1];
w1_2 = network[2]; b1_2 = network[3];
w2_1 = network[4]; b2_1 = network[5];
w2_2 = network[6]; b2_2 = network[7];
w3_1 = network[8]; b3_1 = network[9];
w3_2 = network[10]; b3_2 = network[11];
w3_3 = network[12]; b3_3 = network[13];
w4_1 = network[14]; b4_1 = network[15];
w4_2 = network[16]; b4_2 = network[17];
w4_3 = network[18]; b4_3 = network[19];
w5_1 = network[20]; b5_1 = network[21];
w5_2 = network[22]; b5_2 = network[23];
w5_3 = network[24]; b5_3 = network[25];
w1 = network[26]; b1 = network[27];
w2 = network[28]; b2 = network[29];
w3 = network[30]; b3 = network[31];
```

```
const int NETWORK_SIZES[] = {
    64 * 3 * 3 * 3, 64,
    64 * 64 * 3 * 3, 64,
    128 * 64 * 3 * 3, 128,
    128 * 128 * 3 * 3, 128,
    256 * 128 * 3 * 3, 256,
    256 * 256 * 3 * 3, 256,
    256 * 256 * 3 * 3, 256,
    512 * 256 * 3 * 3, 512,
    512 * 512 * 3 * 3, 512,
    512 * 512 * 3 * 3, 512,
    512 * 512 * 3 * 3, 512,
    512 * 512 * 3 * 3, 512,
    512 * 512 * 3 * 3, 512,
    512 * 512, 512,
    512 * 512, 512,
    10 * 512, 10
};
```

CNN Layers (1)

Convolution

- 이미지 내에서 필터를 활용하여 특징을 추출
  ex) 선 검출 필터 → convolution → 선으로 표현된 그림 도출

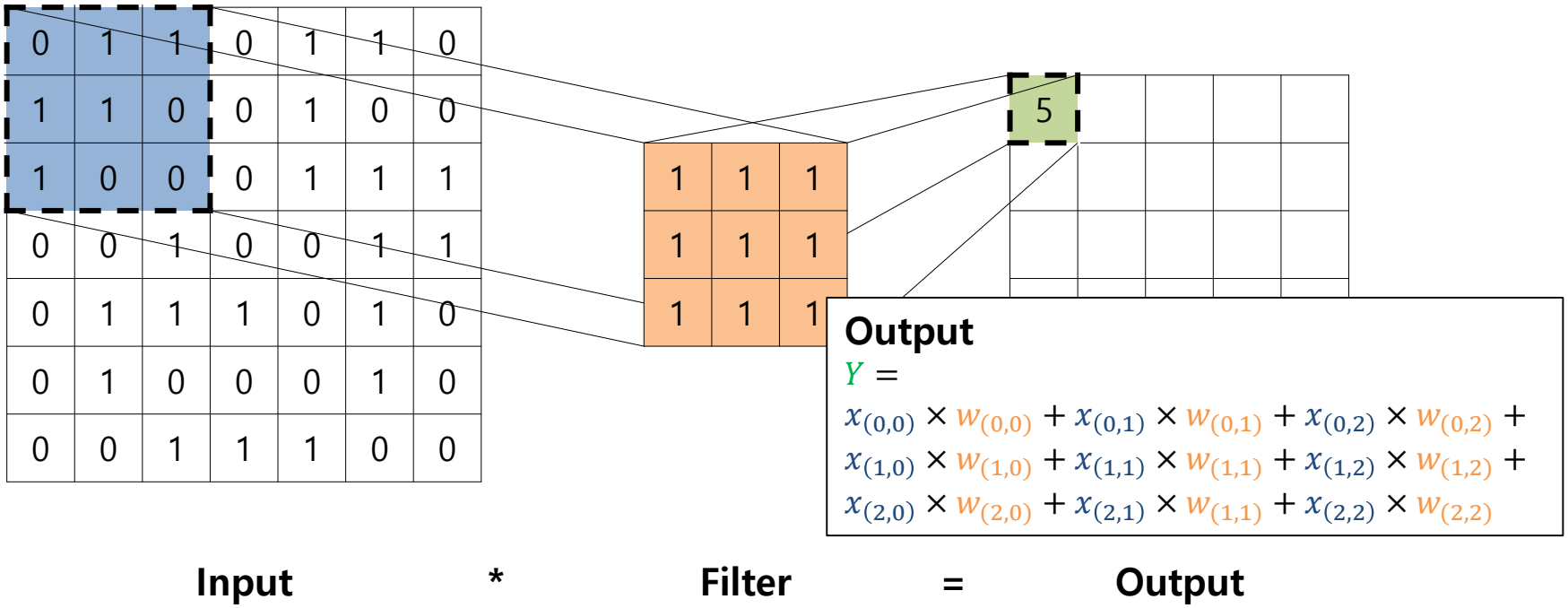


Input * **Filter** = **Output**

CNN Layers (1)

Convolution

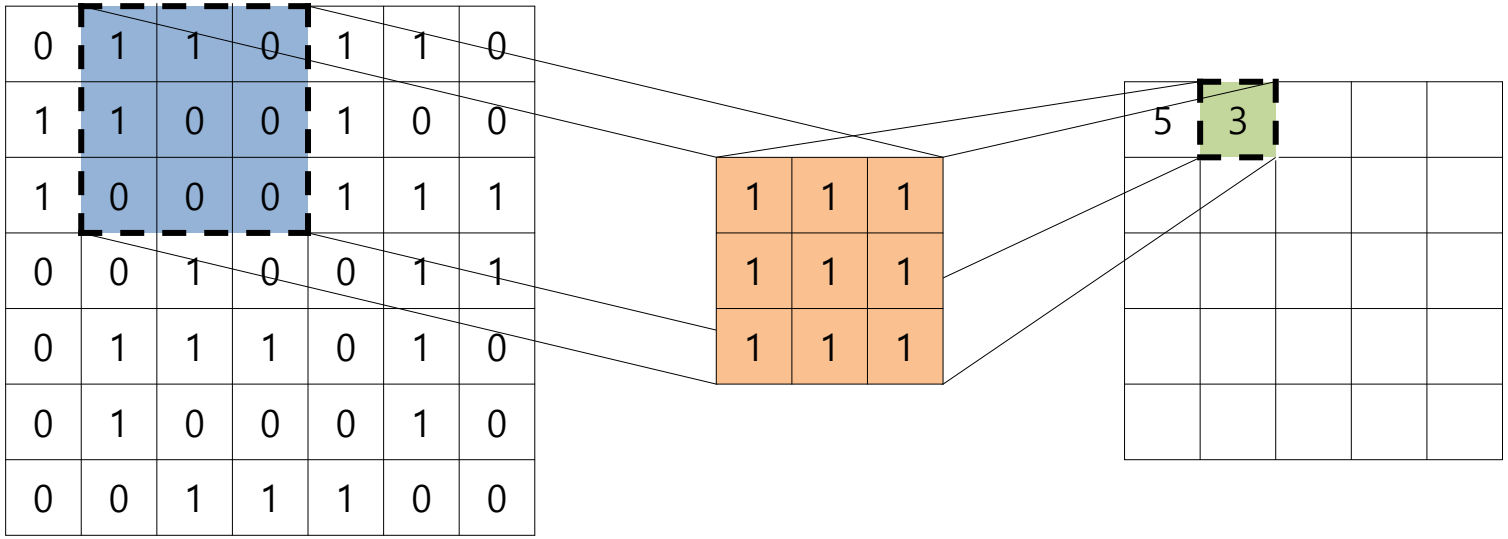
- 이미지 내에서 필터를 활용하여 특징을 추출
  ex) 선 검출 필터 → convolution → 선으로 표현된 그림 도출



CNN Layers (1)

Convolution

- 이미지 내에서 필터를 활용하여 특징을 추출
  ex) 선 검출 필터 → convolution → 선으로 표현된 그림 도출

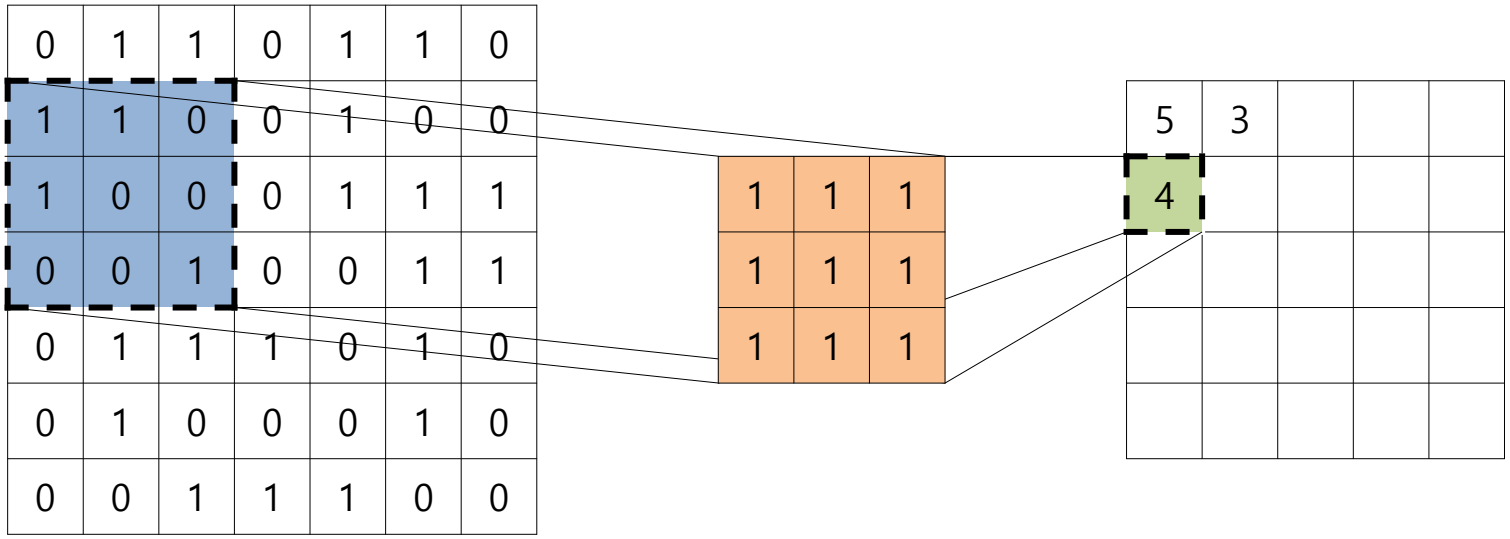


Input * **Filter** = **Output**

CNN Layers (1)

Convolution

- 이미지 내에서 필터를 활용하여 특징을 추출
  ex) 선 검출 필터 → convolution → 선으로 표현된 그림 도출

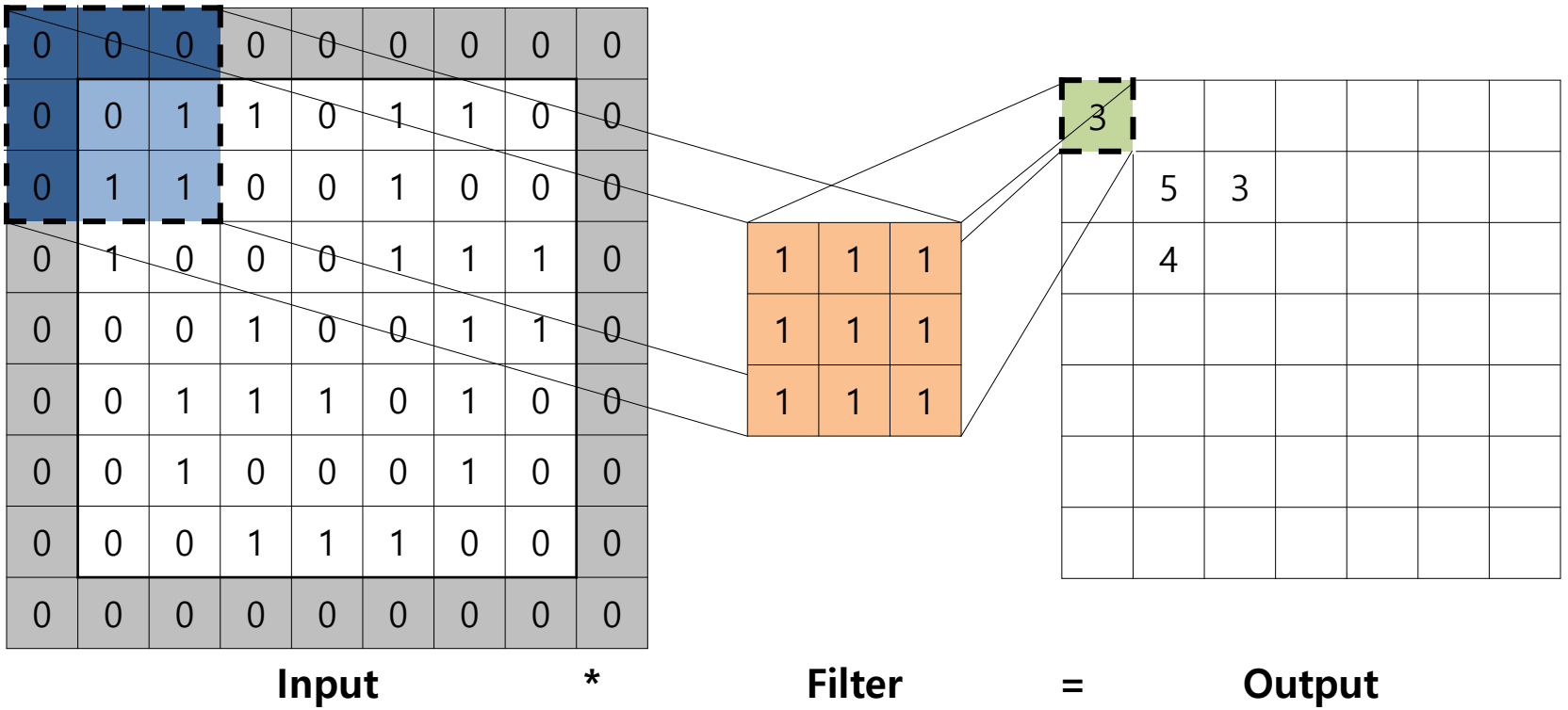


Input * **Filter** = **Output**

CNN Layers (1)

Convolution

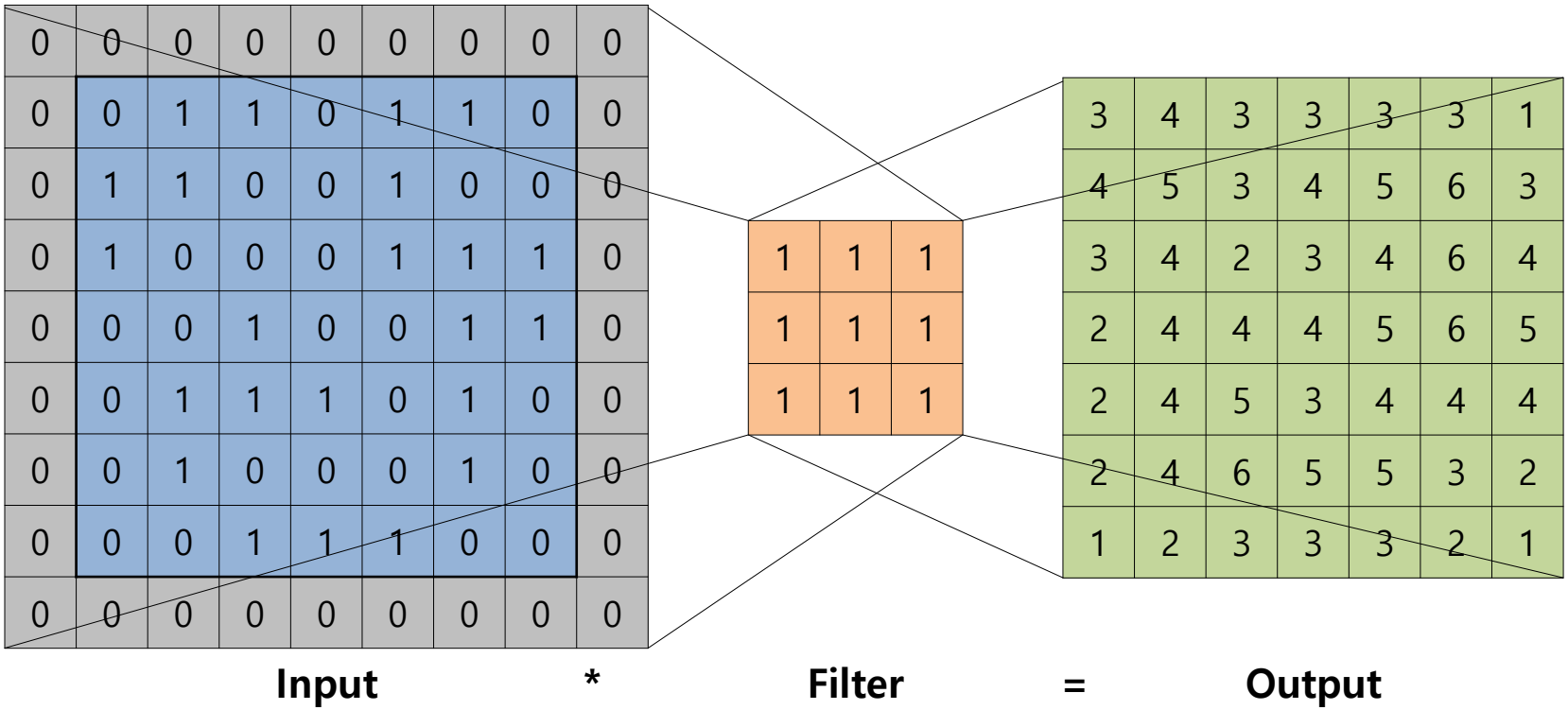
- 이미지 내에서 필터를 활용하여 특징을 추출
  ex) 선 검출 필터 → convolution → 선으로 표현된 그림 도출



CNN Layers (1)

Convolution

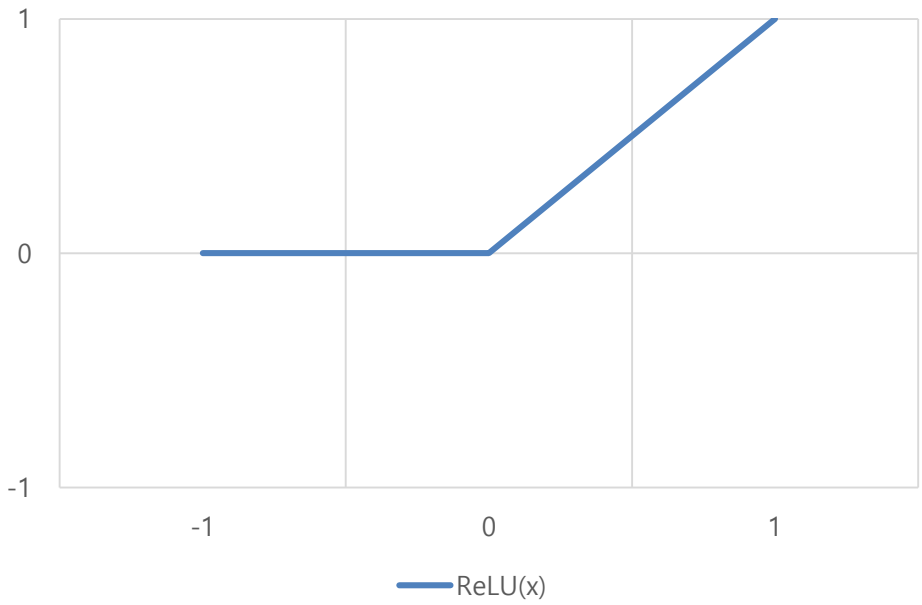
- 이미지 내에서 필터를 활용하여 특징을 추출
  ex) 선 검출 필터 → convolution → 선으로 표현된 그림 도출



CNN Layers (1)

ReLU

- 입력 값을 활성화 여부를 결정하는 함수
- 주로 각 신경망 레이어의 출력 값에 사용



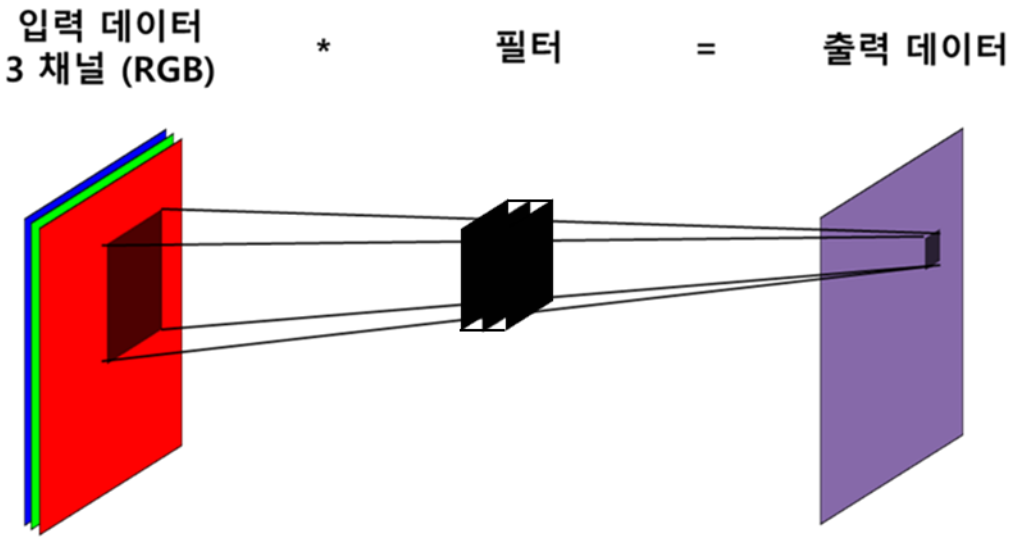
Output

$$Y = \begin{cases} 0 & (x < 0) \\ x & (x \geq 0) \end{cases}$$

CNN Layers (1)

Convolution

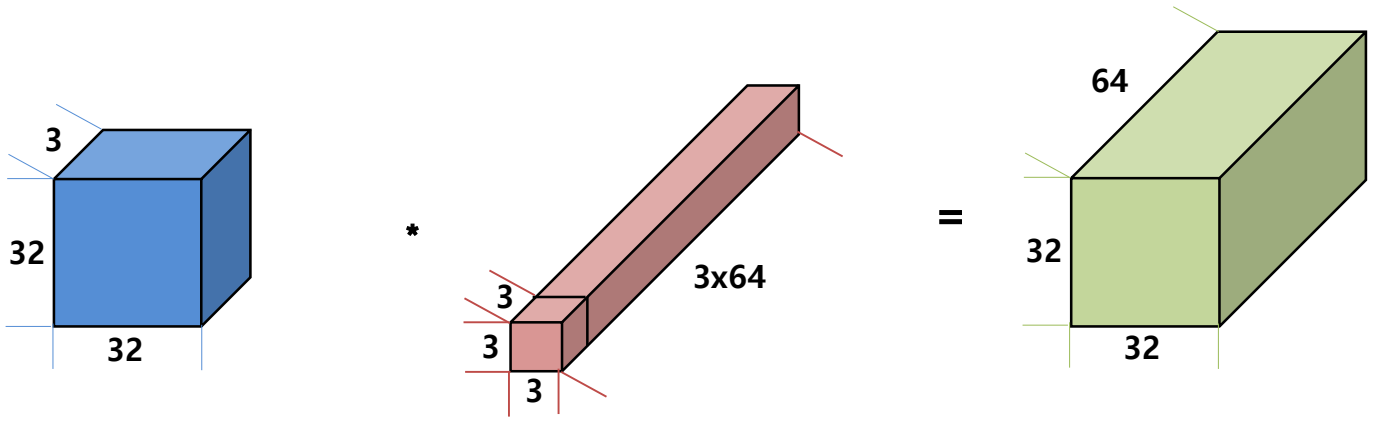
- 필터마다 입력 데이터의 모든 채널과 연산을 거쳐 출력 데이터를 생성



$$Input_{[1:3]} * Filter_{[1:3]} = Output_{[1]}$$

CNN Layers (1)

- **Convolution (예시)**
  - **Input:** $N \times N \times D_1$ $(32 \times 32 \times 3)$
  - **Filter:** $3 \times 3 \times D_1 \times D_2$ $(3 \times 3 \times 3 \times 64)$
  - **Output:** $N \times N \times D_2$ $(32 \times 32 \times 64)$



소스코드

❖ Convolution layer (1)

convolution_layer(float *inputs, float *outputs, float *filters, float *biases, int D2, int D1, int N)

```
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);
    }
}
```

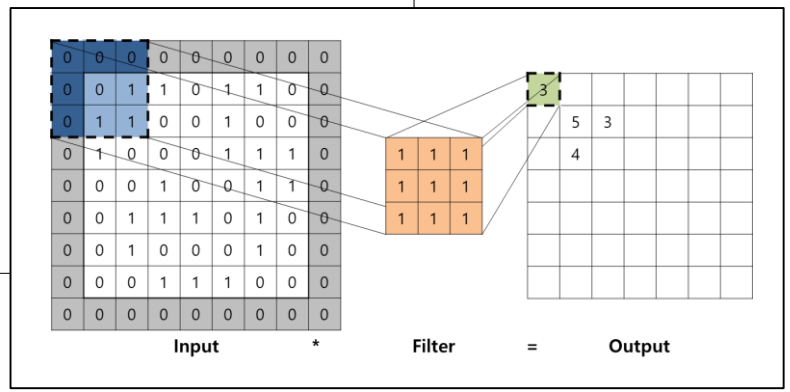
D2: output channels
D1: Input Channels
N: Input size (N x N)

소스코드

❖ Convolution layer (2)

convolution3x3(float *input, float *output, float *filter, int N)

```
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;
    }
}
```



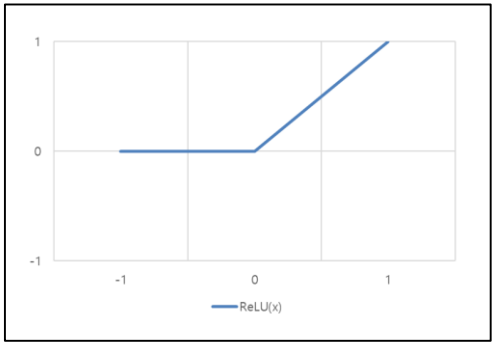
소스코드

❖ Convolution layer (3)

convolution_layer(float *inputs, float *outputs, float *filters, float *biases, int D2, int D1, int N)

```
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);
    }
}
```

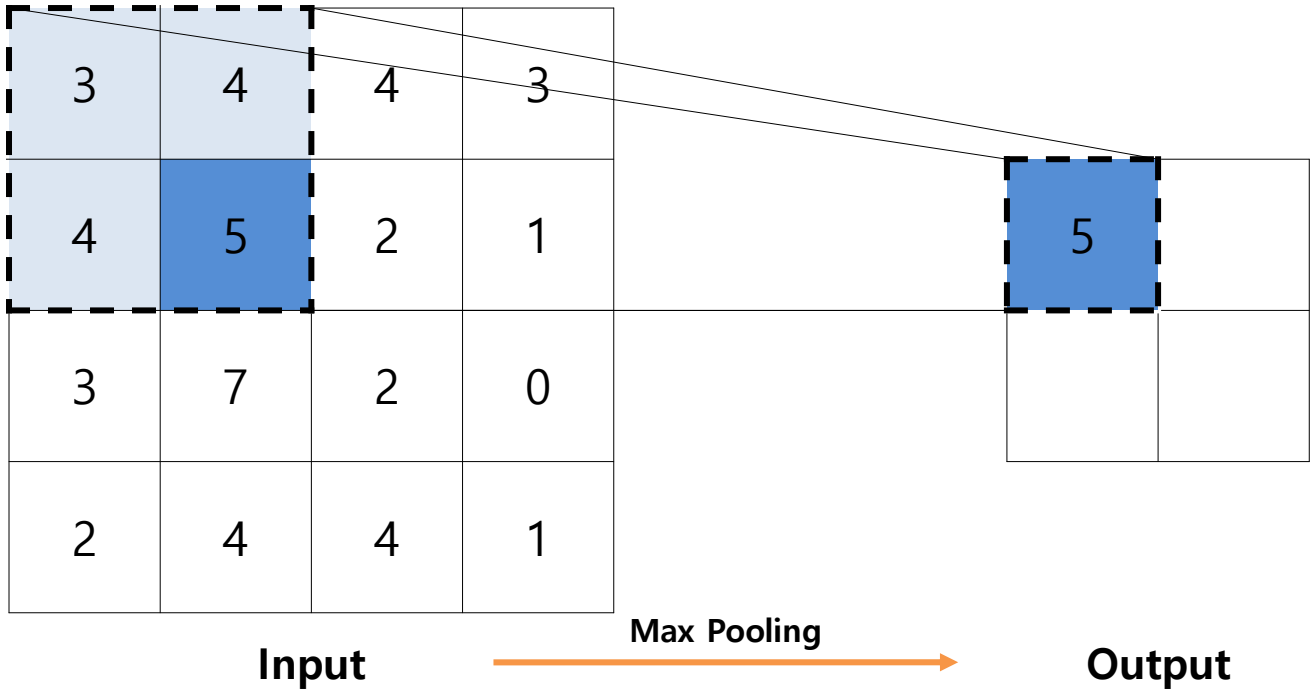


#define ReLU(x) (((x)>0)?(x):0)

CNN Layers (2)

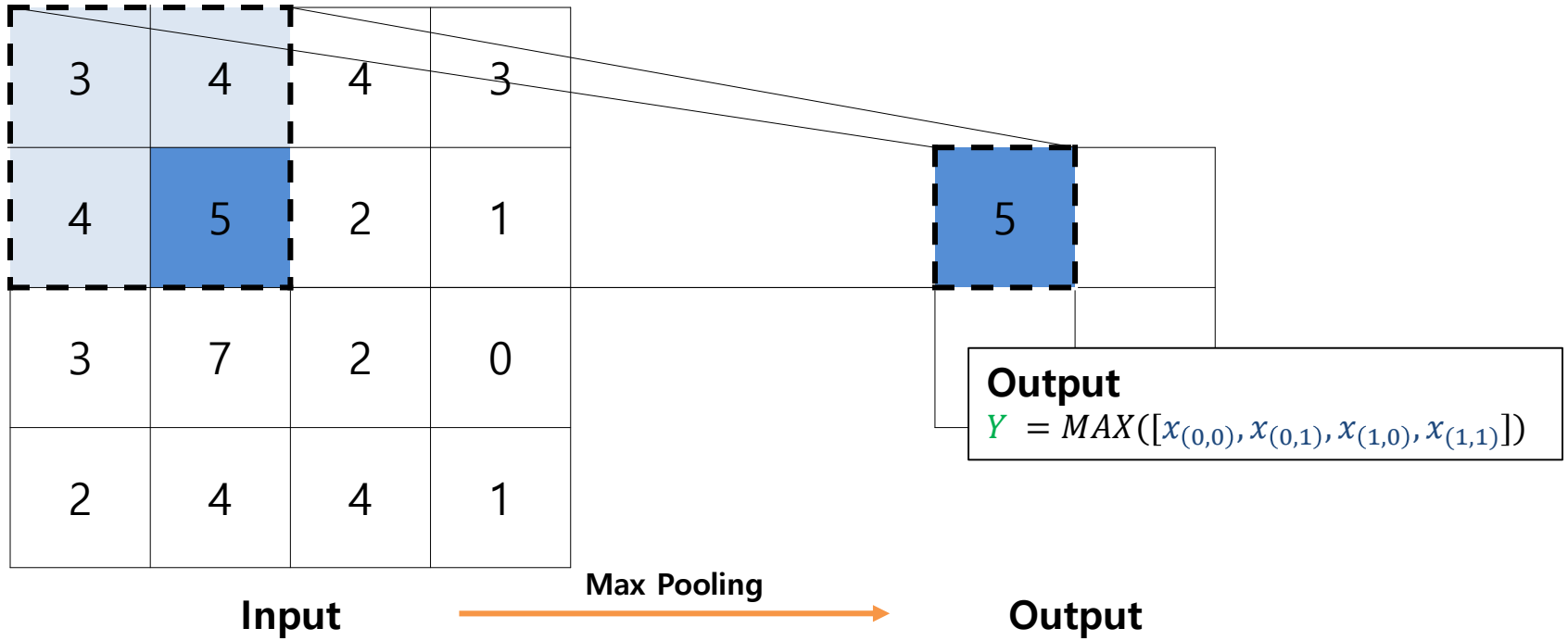
Pooling

- 이미지 크기를 줄이는 연산
- 인접한 n x n 사이지의 픽셀 중 가장 값이 큰 픽셀을 선택 (Max pooling)



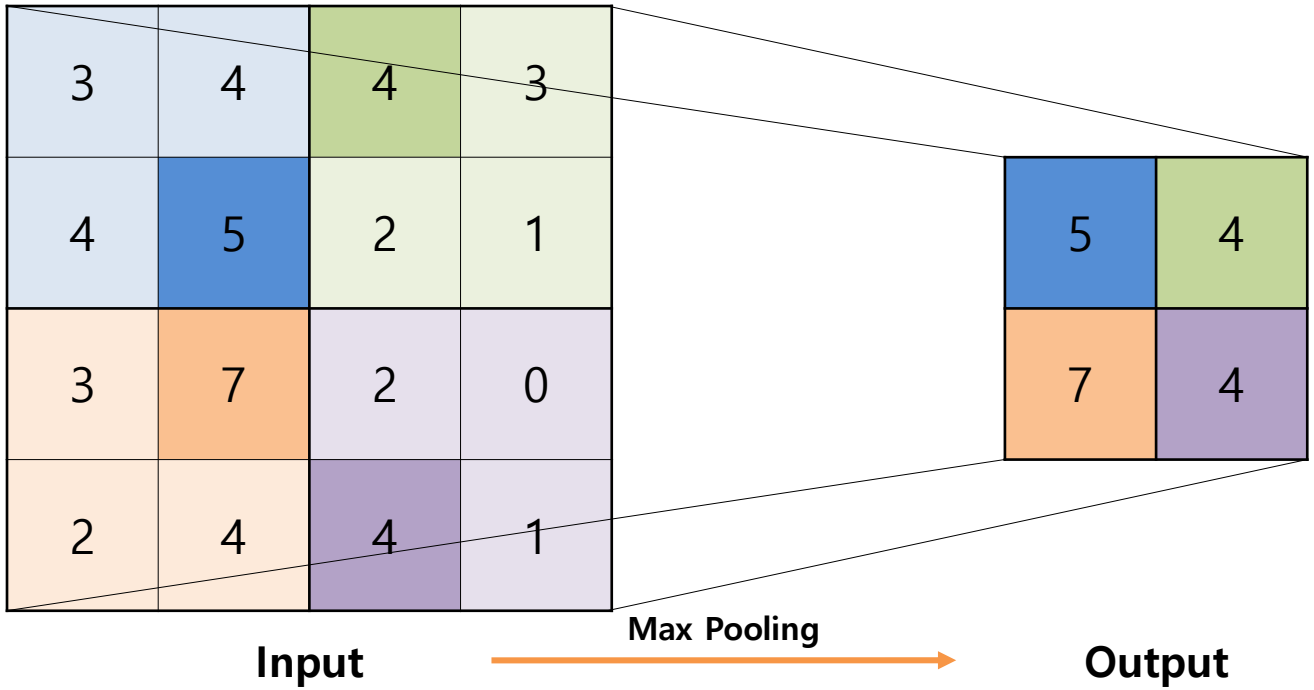
CNN Layers (2)

- **Pooling**
  - 이미지 크기를 줄이는 연산
  - 인접한 n x n 사이지의 픽셀 중 가장 값이 큰 픽셀을 선택 (Max pooling)



CNN Layers (2)

- **Pooling**
  - 이미지 크기를 줄이는 연산
  - 인접한 n x n 사이지의 픽셀 중 가장 값이 큰 픽셀을 선택 (Max pooling)



소스코드

❖ Pooling layer (1)

pooling_layer(float *inputs, float *outputs, int D, int N)

```
for (i = 0; i < D; i++) {
    float * input = inputs + i * N * N * 4;
    float * output = outputs + i * N * N;
    pooling2x2(input, output, N);
}
```

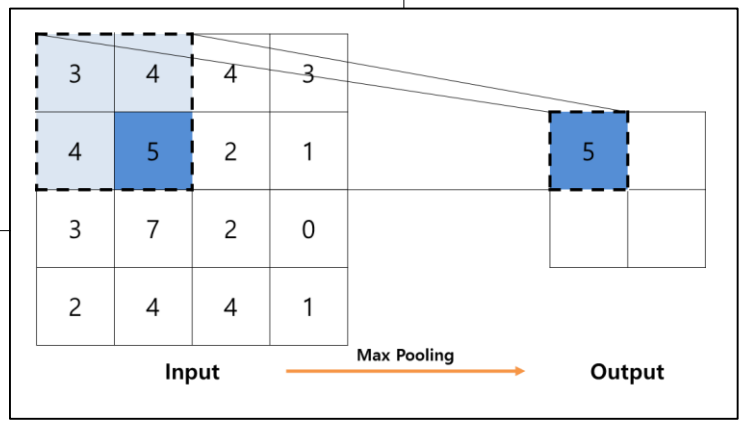
D: channels
N: output size (N x N)

소스코드

❖ Pooling layer (2)

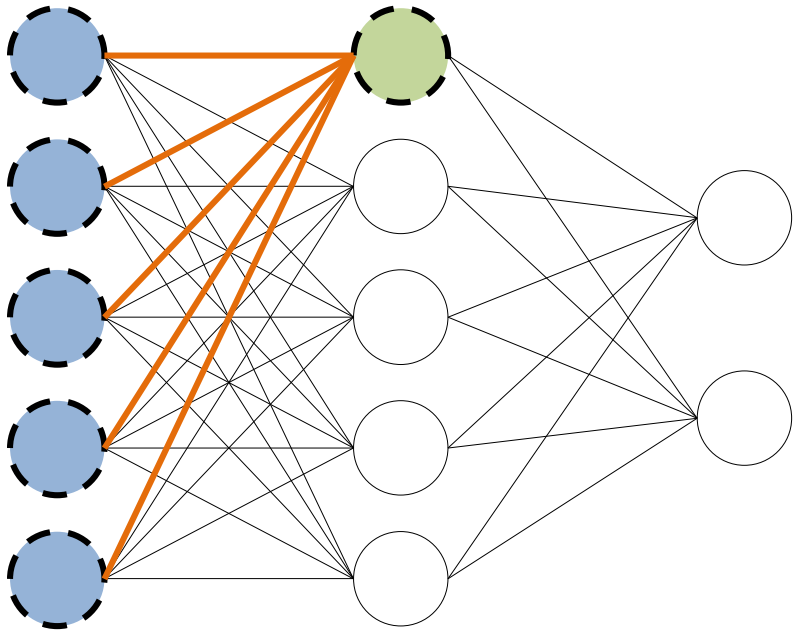
pooling2x2(float *inputs, float *outputs, int N)

```
for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
        float max = 0;
        for (k = 0; k < 2; k++) {
            for (l = 0; l < 2; l++) {
                float pixel = input[(i * 2 + k) * 2 * N + j * 2 + l];
                max = (max > pixel) ? max : pixel;
            }
        }
        output[i * N + j] = max;
    }
}
```



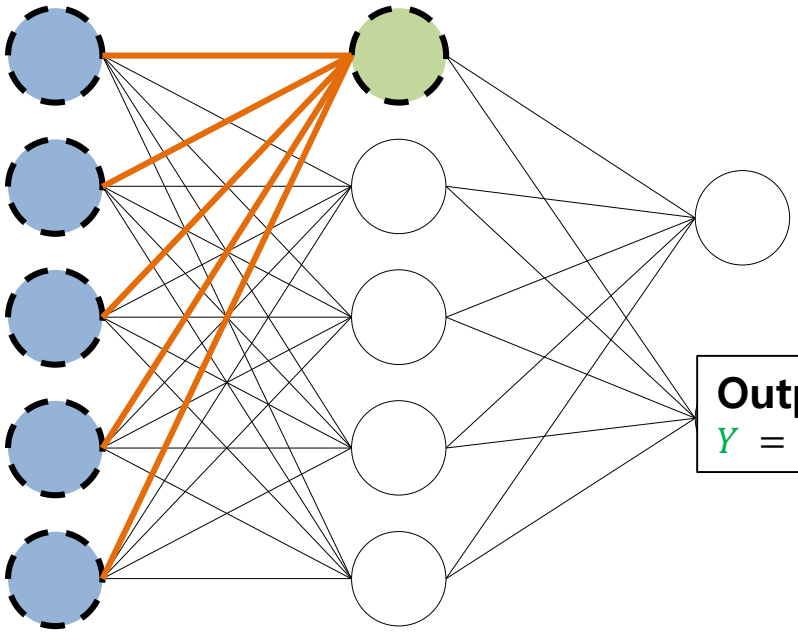
CNN Layers (3)

- **Fully-connected**
  - 모든 노드들이 서로 연결된 형태의 기본적인 신경망



CNN Layers (3)

- **Fully-connected**
  - 모든 노드들이 서로 연결된 형태의 기본적인 신경망

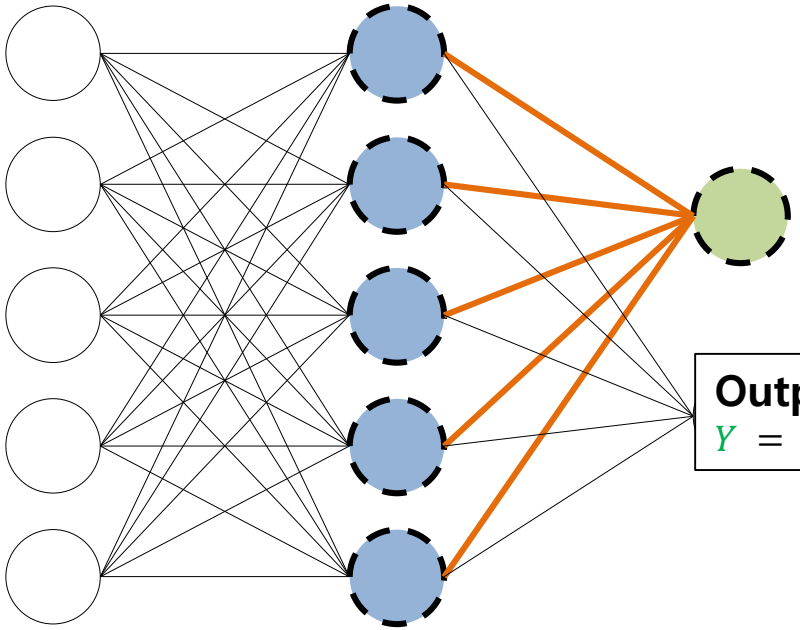


Output

$$Y = W_1 X_1 + W_2 X_2 + W_3 X_3 + W_4 X_4 + W_5 X_5$$

CNN Layers (3)

- **Fully-connected**
  - 모든 노드들이 서로 연결된 형태의 기본적인 신경망



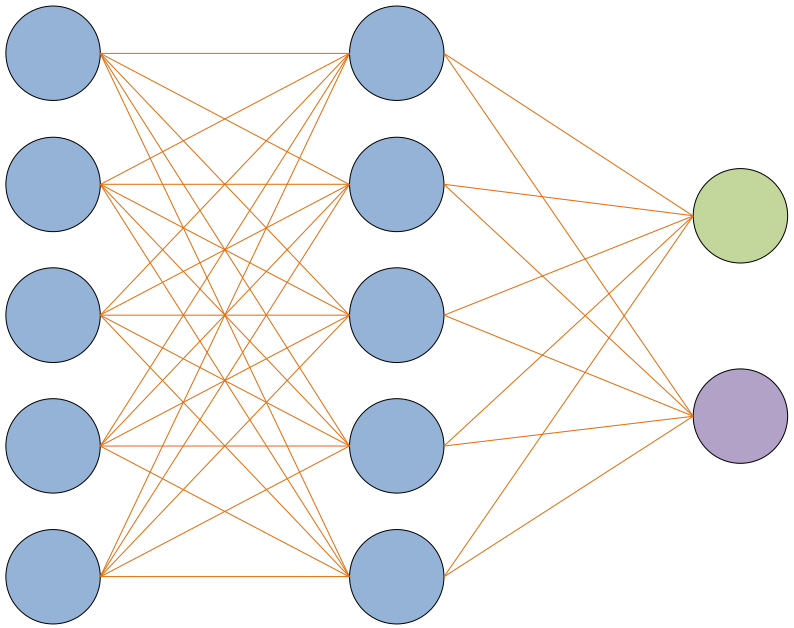
0.2104

Output

$$Y = W_1 X_1 + W_2 X_2 + W_3 X_3 + W_4 X_4 + W_5 X_5$$

CNN Layers (3)

- **Fully-connected**
  - 모든 노드들이 서로 연결된 형태의 기본적인 신경망



0.2104

-1.3163

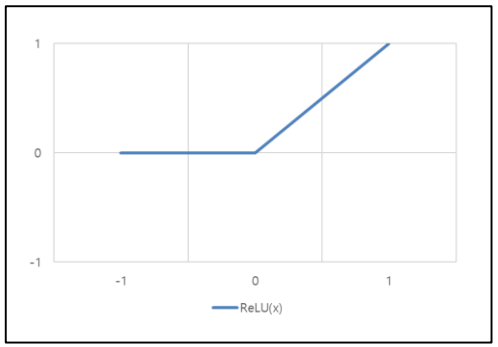
소스코드 (8)

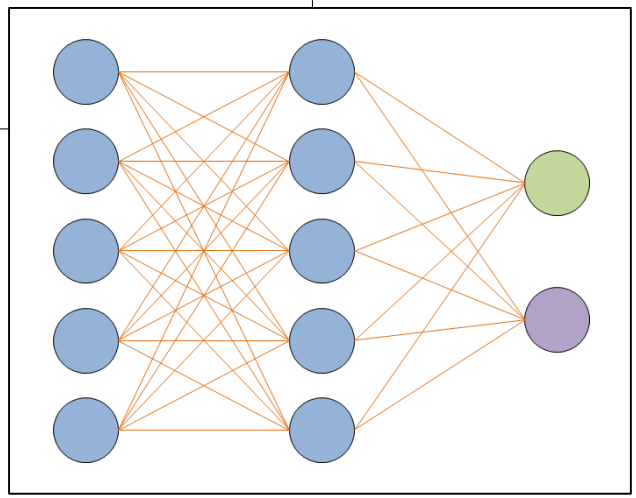
❖ Fully connected layer

fc_layer(float *input_neuron, float *output_neuron, float *weights, float *biases, int M, int N)

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

M: Output size
N: Input size

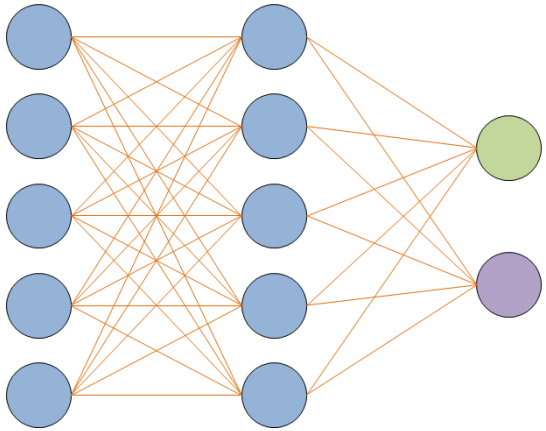




CNN Layers (4)

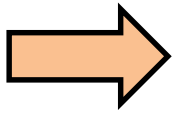
Softmax

- 입력 값을 0과 1 사이의 값으로 정규화 하는 함수 (출력 값들의 합은 1)
- 주로 신경망의 최종 출력 값에 사용



0.2104

-1.3163



0.8215

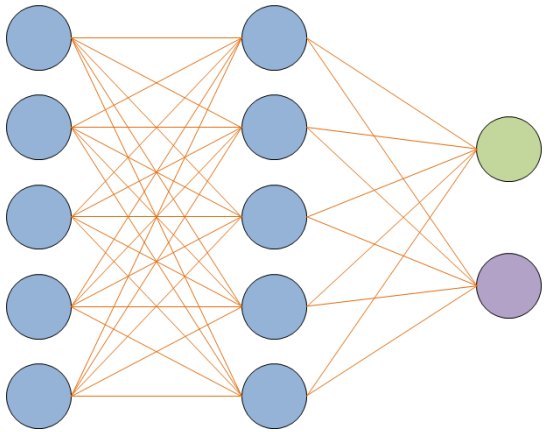
0.1785

Output

$$Y = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

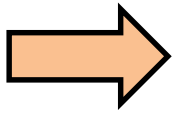
CNN Layers (4)

- **Softmax**
  - 입력 값을 0과 1 사이의 값으로 정규화 하는 함수 (출력 값들의 합은 1)
  - 주로 신경망의 최종 출력 값에 사용



0.2104

-1.3163

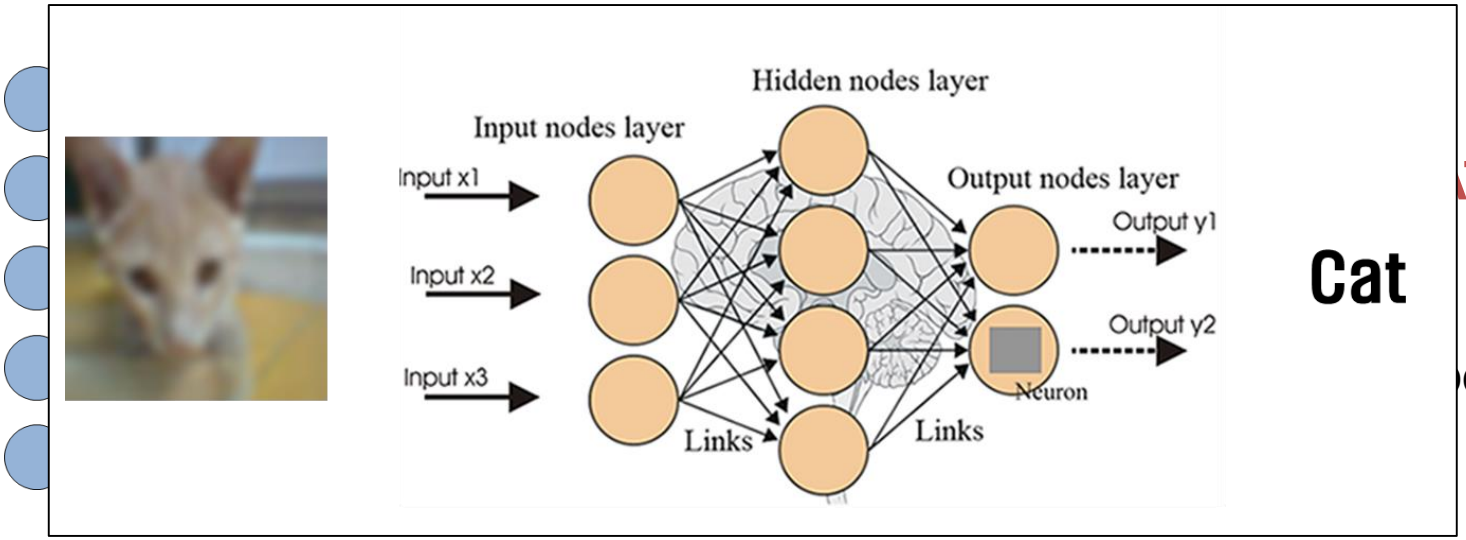


0.8215 CAT

0.1785 DOG

CNN Layers (4)

- **Softmax**
  - 입력 값을 0과 1 사이의 값으로 정규화 하는 함수 (출력 값들의 합은 1)
  - 주로 신경망의 최종 출력 값에 사용



프로젝트 수행 내용 (1)

- 순차 코드를 가속할 수 있는 kernel 코드 작성 및 성능 평가 수행

- cifar10_image.bin, cifar10_label.bin
  - 10,000개의 이미지 및 레이블

- 프로그램 실행
  - ./cnn 실행파일 <이미지 수> <출력파일>
    - ./cnn_seq 3000 result.out
    - 이미지 10,000장이 아닌 3,000장 처리 시간 비교
  - ./compare 실행 파일 <출력파일1> <출력파일2>
    - ./compare_result answer.out result.out
    - 결과 값이 같은지 출력

프로젝트 수행 내용 (2)

- 구현
  - cnn_opencl.c 파일의 TODO부분 구현

- 조건
  - OpenCL 디바이스 GPU만 활용(실습실 GPU기준)
  - 알고리즘 자체의 변경은 불가
  - 결과 값은 최대 floating point 오차 0.01까지 허용
  - 만약 오차 허용 값에 의해서 결과가 순차코드와 다른 경우에는 해당 이미지 명시
  - 팀 별 제출 (2~3인 팀 구성)
  - 주에 1번(화요일 또는 목요일)은 저녁 실습실 출석 체크

프로젝트 수행 내용 (3)

- 중간 발표
  - 11월 27일 중간 결과물 및 발표 자료 제출, 11월 28일 중간 발표
  - 중간 발표 자료 구성
    - CNN 코드 이해한 만큼 정리해서 발표
    - 현재까지 진행한 최적화 기법 간단하게 발표
    - 최적화 코드가 동작하는 경우, 동작 시간 체크해서 발표자료에 추가, 소스코드 제출
  - 중간 발표 및 최종 발표 2:8 비율로 프로젝트 점수 반영

- 최종 발표
  - 12월 9일 최종 결과물 및 보고서, 발표 자료 제출, 12월 10일 최종 발표
  - 데이터셋을 제외한 프로젝트 파일 전체
  - 보고서 작성 (word, 한글)
    - 최적화 방법 → 적용 전, 후의 성능 차이 (적용한 방법들 각각을 상세하게 작성)
    - 느낀 점 정리
  - 프로젝트 발표자료 (ppt or pdf)

프로젝트 수행 결과

❖ 프로젝트 결과 분포

- 2018년 프로젝트 결과
  - 총 8개 팀 중 7개 팀 제출
  - 1위 팀 15.152초 달성
  - 1개 팀 정확도 미 일치