Convolution Neural Network

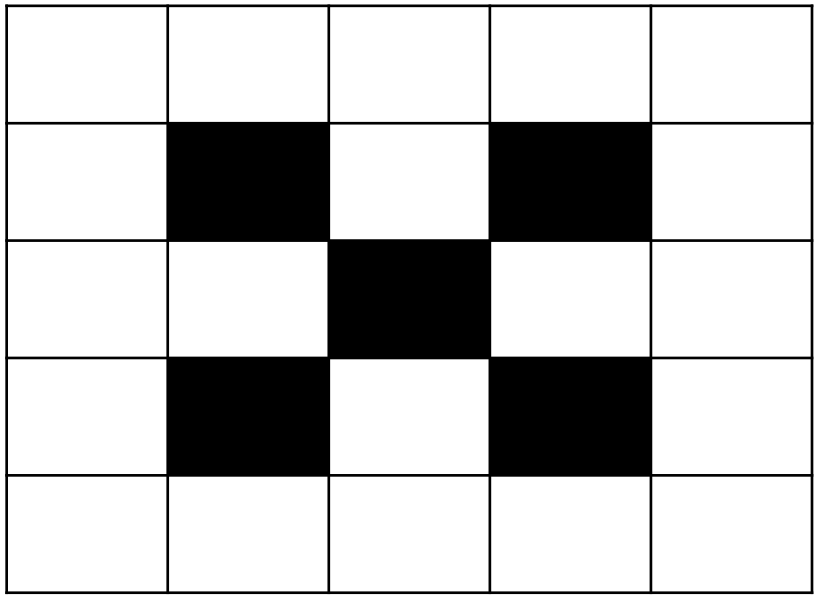
김균엽

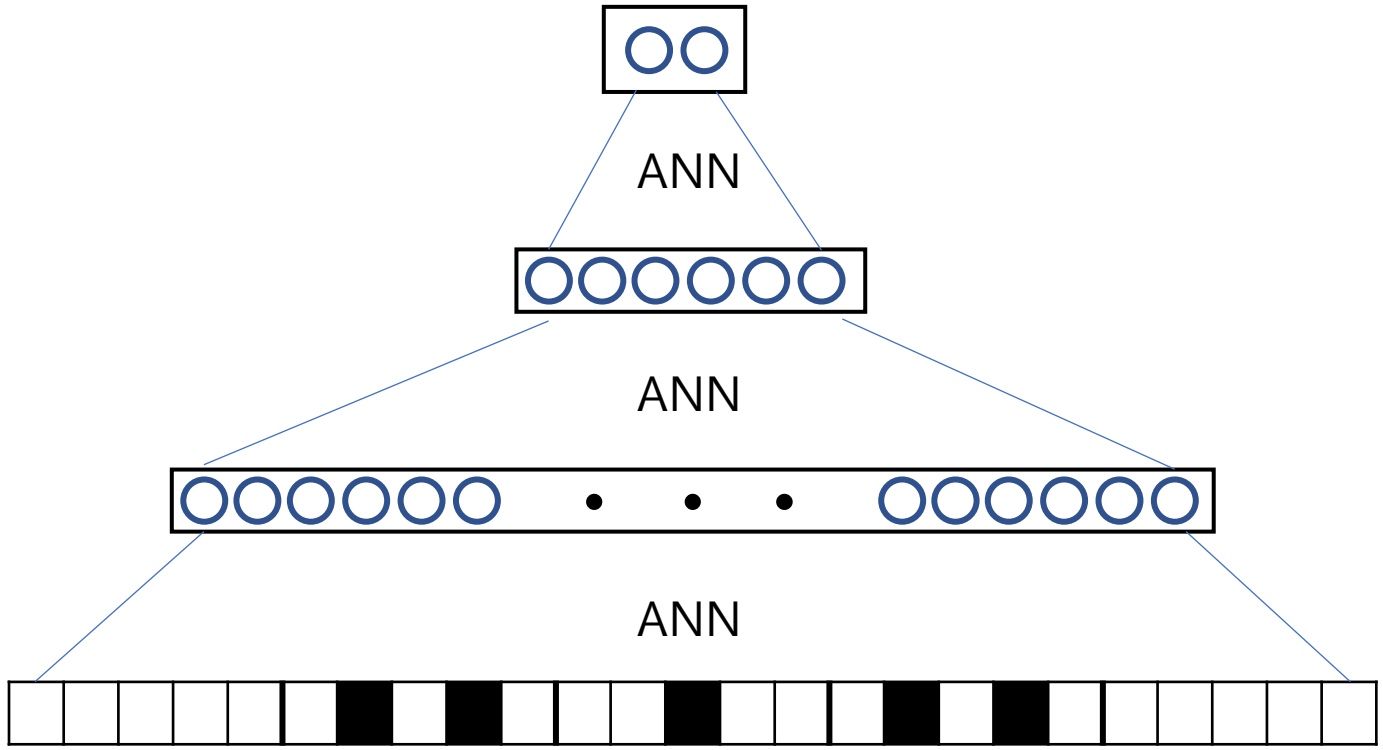
CNN

- **Problem of MLP**
  - MLP은 공간정보를 이해할 수 없다.
  - 모든 data를 1차원으로 변환하여 학습하기 때문에 2차원, 3차원 공간에 대한 학습이 어렵다.
    - 2차원 그림에 X가 존재하는지 여부 판별
    - → CNN
  - 또한 ANN은 순서가 바뀌어도 다른데이터라 인지할 수 없다
    - 데이터의 순서가 키,나이, 성적 순으로 입력되는 것과 나이, 키, 성적 순으로 입력되는 것을 다른데이터라 판별할 수 없다.
    - → RNN

CNN

MLP를 통해서 그림을 판별하는 방법?
Ex) 그림에 X가 있는지 판별



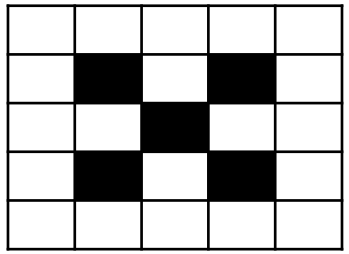


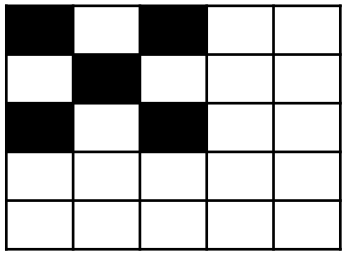
CNN

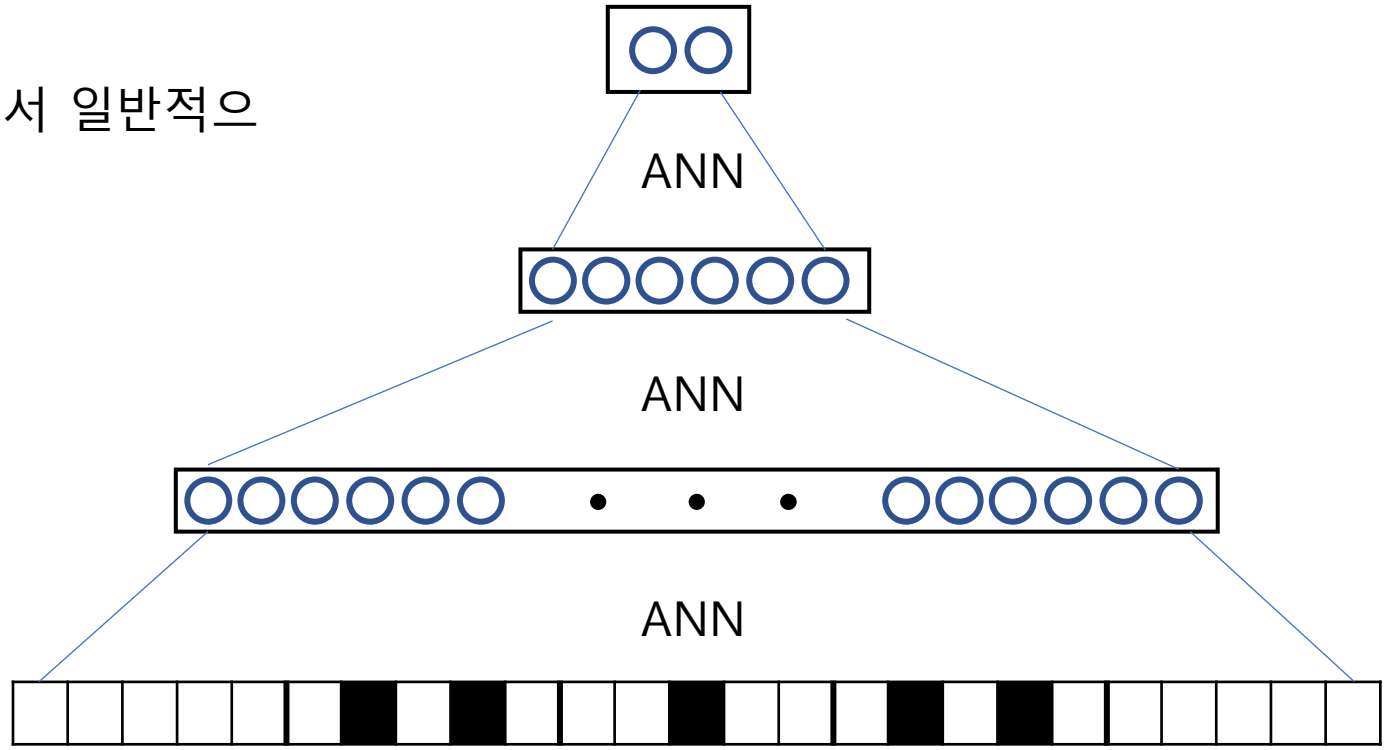
Problem?

1. 이미지의 크기는 적어도 160000개의 픽셀에서 일반적으로 1000*1000을 넘는 크기를 가진다

2. X의 위치가 변할때 대응이 가능할까?







둘은 ANN에서는 연관 관계가 전혀 없는 데이터일 것이라 예측

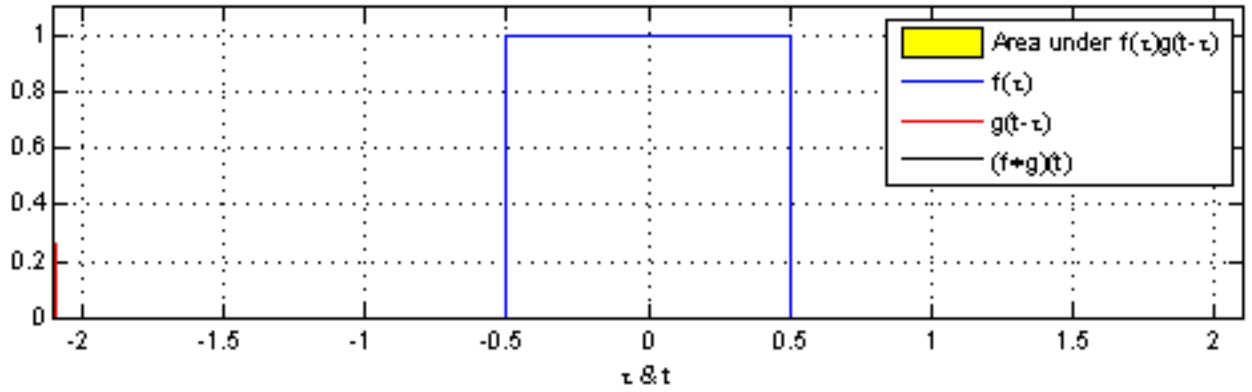
CNN

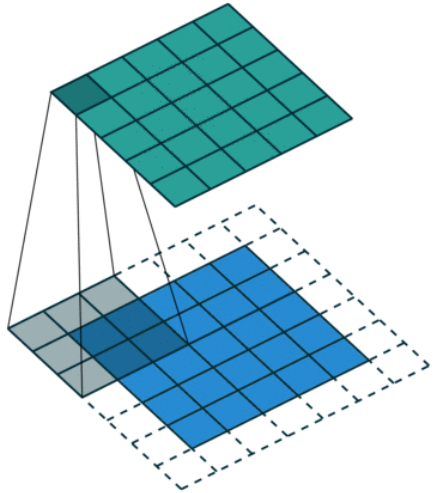
극단적인 예시로 MLP는 아래의 두 데이터를 전혀 다른 데이터라 인지한다.

feature1	feature2	feature3	feature4	feature5	feature6
0	1	1	0	0	0
0	0	1	1	0	0

CNN

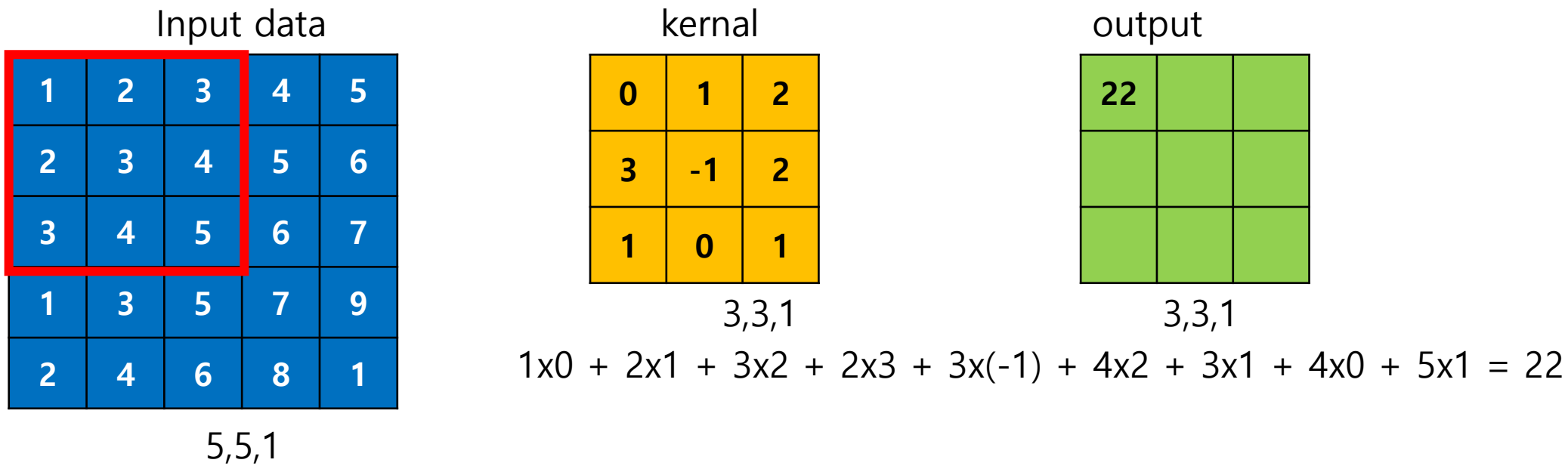
- **CNN**
  - 그림과 같은 정보는 공간적인 정보가 중요하다.
  - 그렇기에 전체 feature(각 pixel)을 연산하는 것이 아닌.
  - 주변 pixel만을 계층적으로 연산함으로써 내 픽셀의 주변 정보 즉, 공간정보를 파악한다.
  - 가까운 픽셀을 연속적으로 연산하여 전체 그림에 대한 정보를 학습할 수 있도록 한다.





CNN

- **CNN**
  - 2D CNN에 대한 예시
  - 입력으로 2d의 1channel 데이터가 입력되었다 가정
  - 입력으로 5,5데이터가 입력되고, kernal이 3,3형태의 parameter가 존재한다고 할때
  - Data에 kernal을 매칭하며 모든 픽셀을 연산



CNN

- **CNN**
  - 2D CNN에 대한 예시
  - 입력으로 2D의 1channel 데이터가 입력되었다 가정
  - 입력으로 5,5데이터가 입력되고, kernal이 3,3형태의 parameter가 존재한다고 할때
  - Data에 kernal을 매칭하며 모든 픽셀을 연산

Input data

1	2	3	4	5
2	3	4	5	6
3	4	5	6	7
1	3	5	7	9
2	4	6	8	1

5,5,1

kernal

0	1	2
3	-1	2
1	0	1

3,3,1

output

22	30	

3,3,1

2x0 + 3x1 + 4x2 + 3x3 + 4x(-1) + 5x2 + 4x1 + 5x0 + 6x1 = 30

CNN

- **CNN**
  - Image의 각 pixel값을 tensor로 변환한 데이터를 input으로 입력
  - Kernel은 parameter로 구성되어있어 deep learning학습을 진행하며 parameter가 최적화 된다.
  - Output으로는 image형태의 tensor가 반환된다.

Input data

1	2	3	4	5
2	3	4	5	6
3	4	5	6	7
1	3	5	7	9
2	4	6	8	1

5,5,1

kernel

w11	w12	w13
w21	w22	w23
w31	w32	w33

3,3,1

output

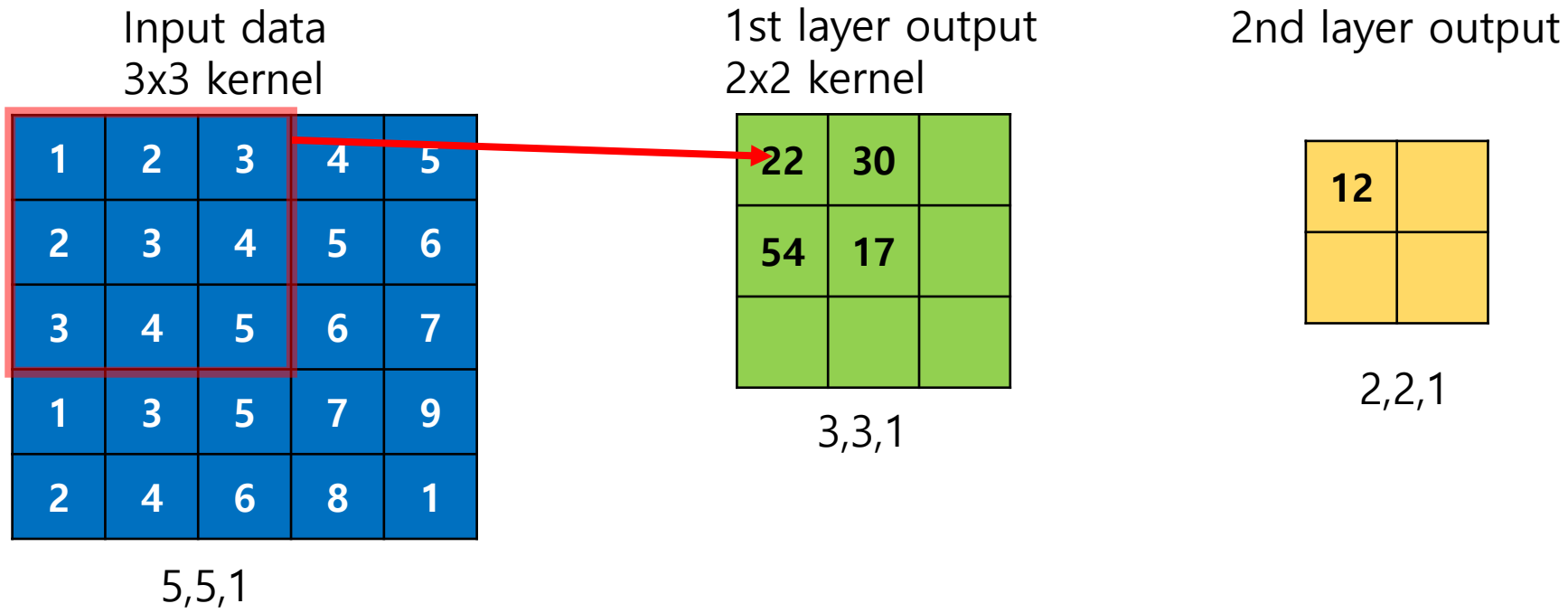
y11	y12	y13
y21	y22	y23
y31	y32	y33

3,3,1

2x0 + 3x1 + 4x2 + 3x3 + 4x(-1) + 5x2 + 4x1 + 5x0 + 6x1 = 30

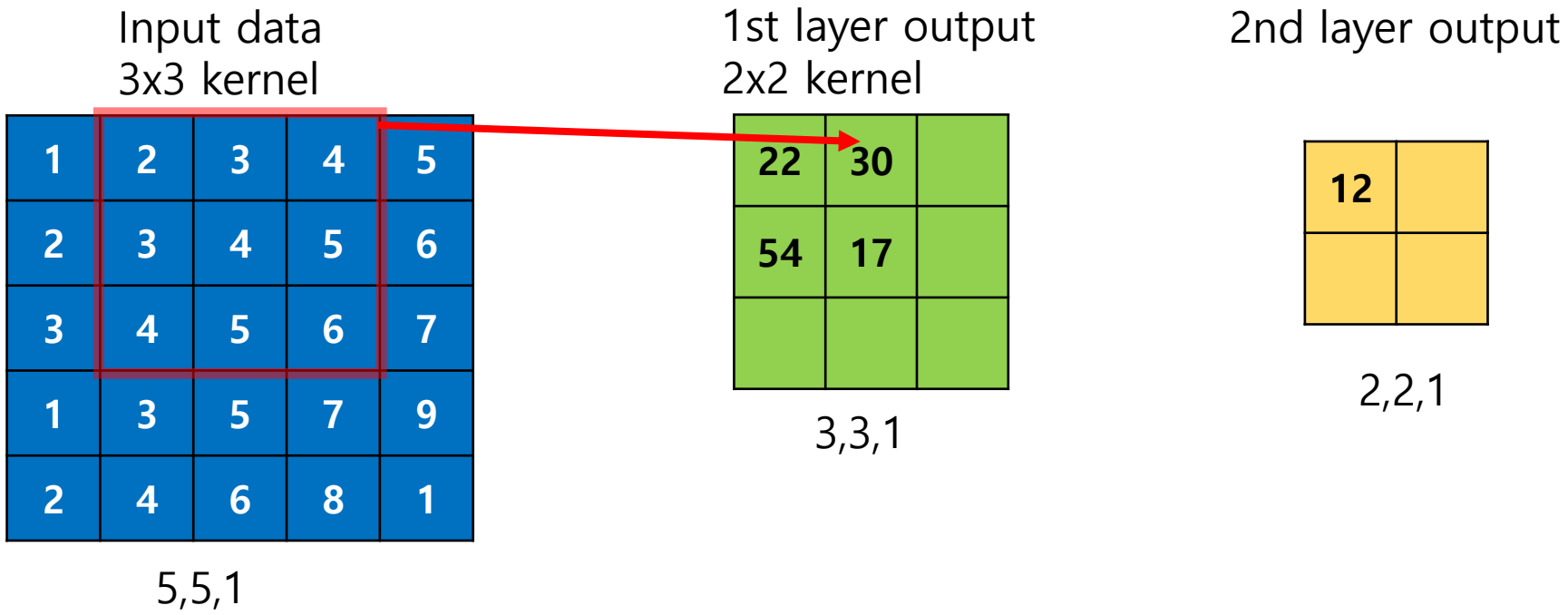
CNN

- **CNN**
  - 초기 layer에서는 주변 픽셀에 대한 정보만 학습되지만
  - Layer가 증가할 수록 점점더 넓은 범위의 정보가 학습됨



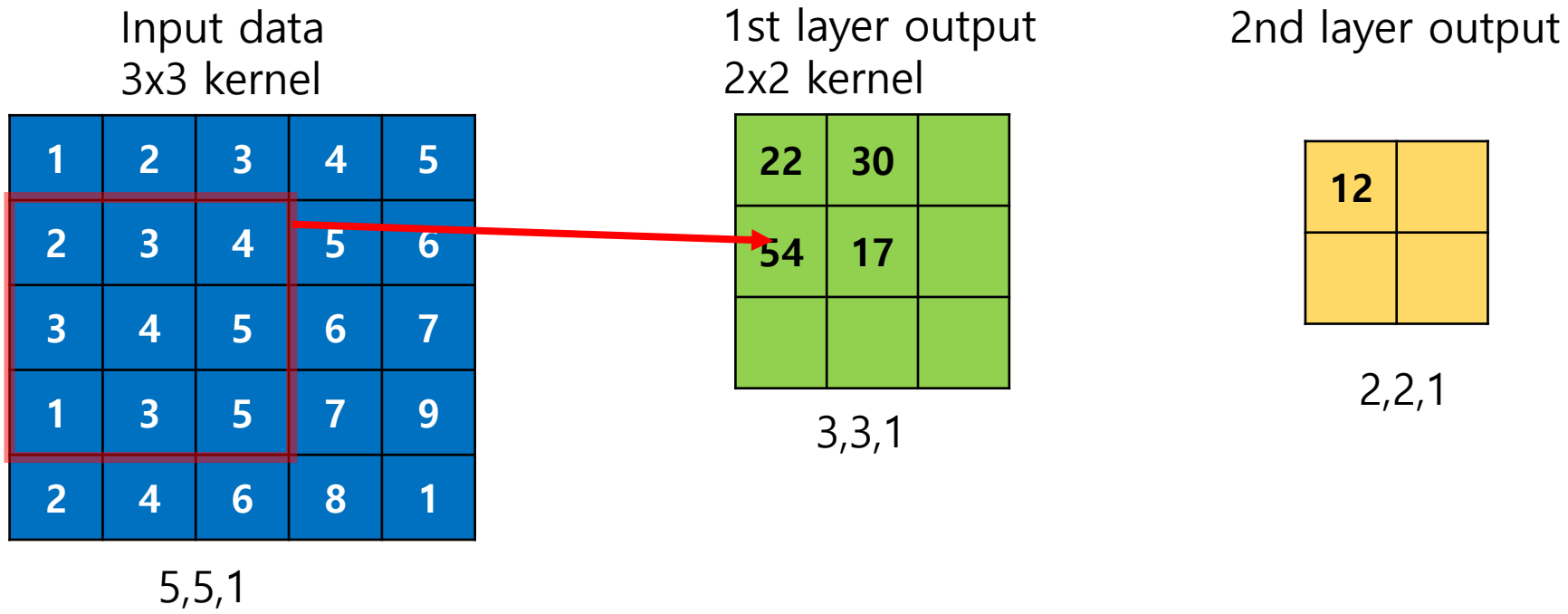
CNN

- **CNN Data**
  - 초기 layer에서는 주변 픽셀에 대한 정보만 학습되지만
  - Layer가 증가할 수록 점점더 넓은 범위의 정보가 학습됨



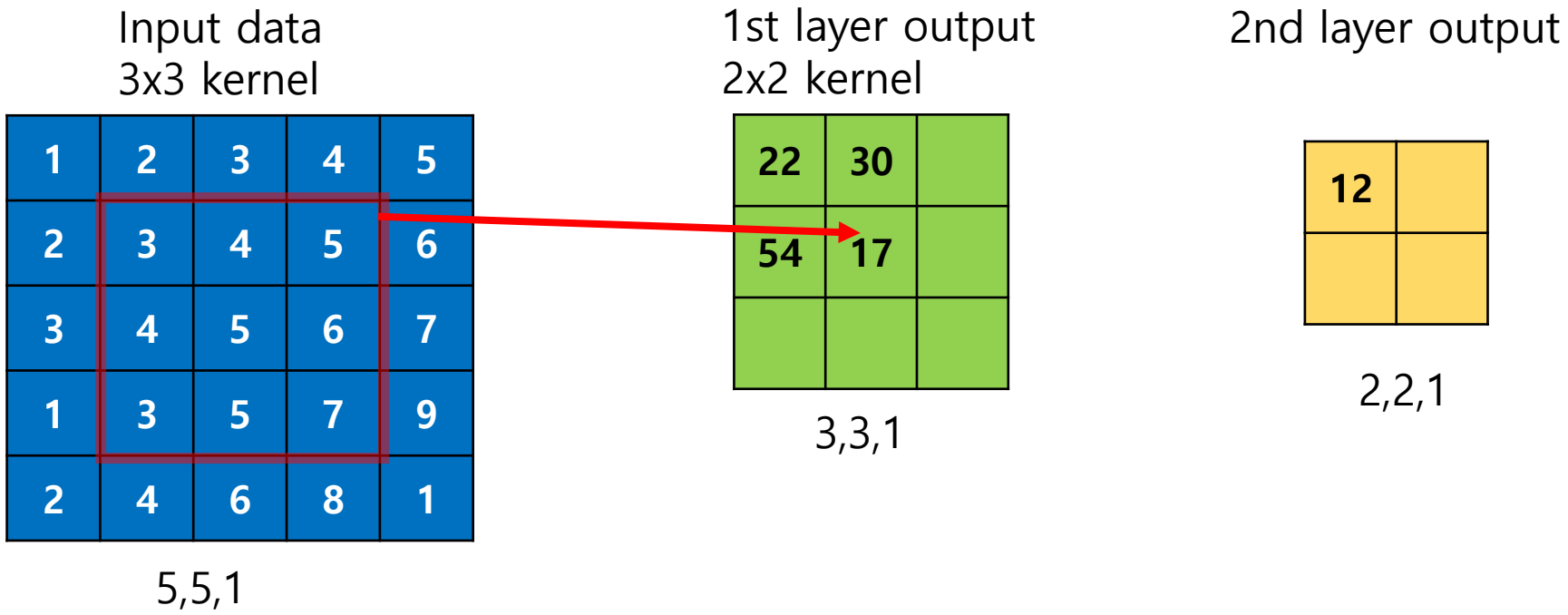
CNN

- **CNN**
  - 초기 layer에서는 주변 픽셀에 대한 정보만 학습되지만
  - Layer가 증가할 수록 점점더 넓은 범위의 정보가 학습됨



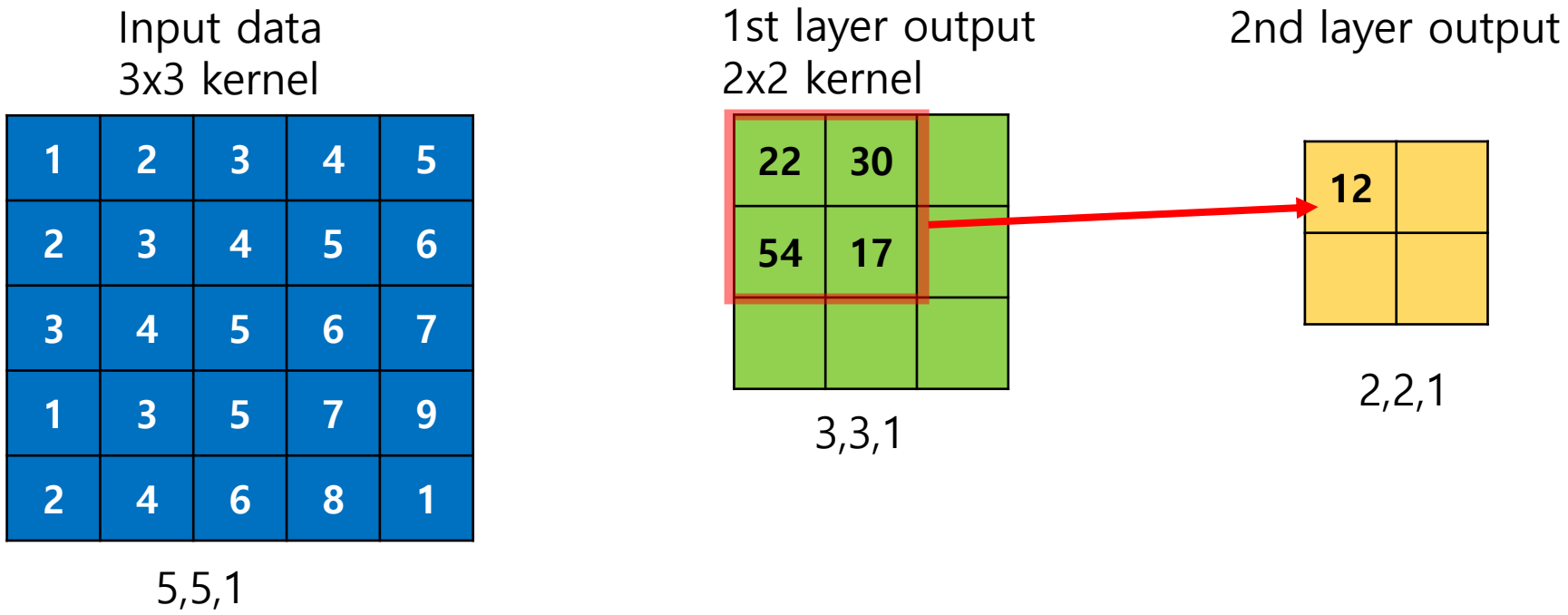
CNN

- **CNN**
  - 초기 layer에서는 주변 픽셀에 대한 정보만 학습되지만
  - Layer가 증가할 수록 점점더 넓은 범위의 정보가 학습됨



CNN

- **CNN**
  - 초기 layer에서는 주변 픽셀에 대한 정보만 학습되지만
  - Layer가 증가할 수록 점점더 넓은 범위의 정보가 학습됨



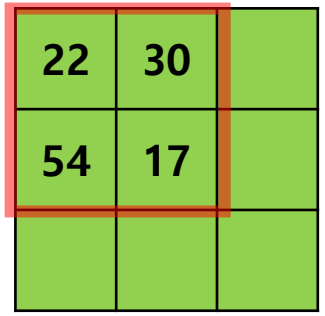
CNN

- **CNN**
  - 초기 layer에서는 주변 픽셀에 대한 정보만 학습되지만
  - Layer가 증가할 수록 점점더 넓은 범위의 정보가 학습됨

Input data
3x3 kernel

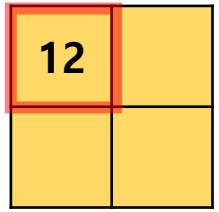
5,5,1

1st layer output
2x2 kernel



3,3,1

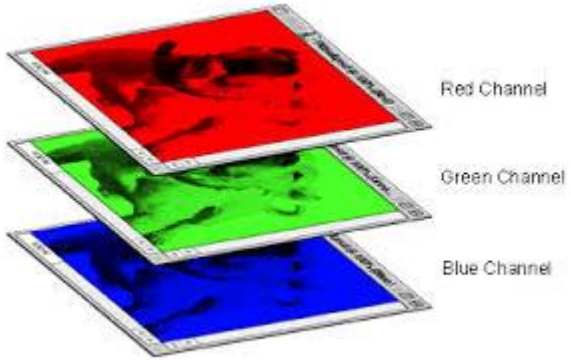
2nd layer output



2,2,1

CNN

- **Channel**
  - RGB와같이 컴퓨터는 데이터를 저장할때 한픽셀에 각 색깔에 대한 정보를 저장한다.
  - 일반적으로 우리가 보는 컬러이미지는 RGB로 3channel의 이미지이다.
  - 28x28크기의 1channel의 image에 포함된 값의 수는 28*28개이다.
  - 그에비해 동일한 크기의 3channel에 포함된 값의 수는 28*28*3개이다.
  - 일반적으로 image를 tensor로 표현할때 (channel, width, height)또는 (width, height, channel)의 형태로 나타낸다



RGB image

grayscale image

CNN

- **CNN with channel**
  - 2D CNN이란 각 kernel 적용 후의 output이 2D인 경우를 얘기한다.

Input data

1	2	3	4	5
2	3	4	5	6
3	4	5	6	7
1	3	5	7	9
2	4	6	8	1

5,5,1

kernal

0	1	2
3	-1	2
1	0	1

3,3,1

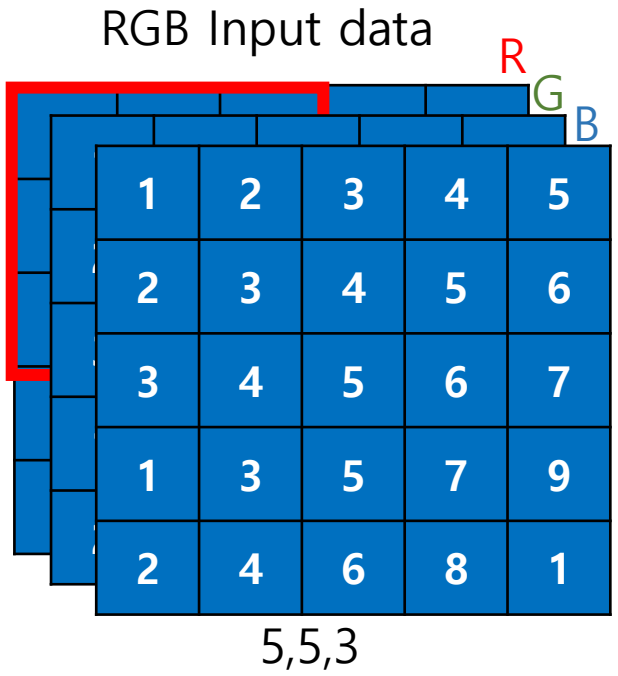
output

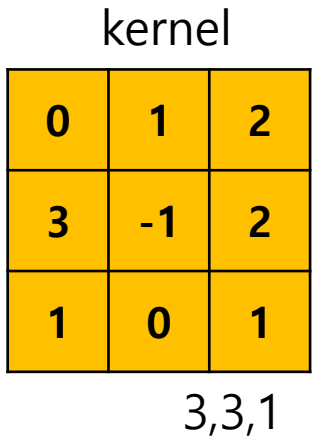
22		

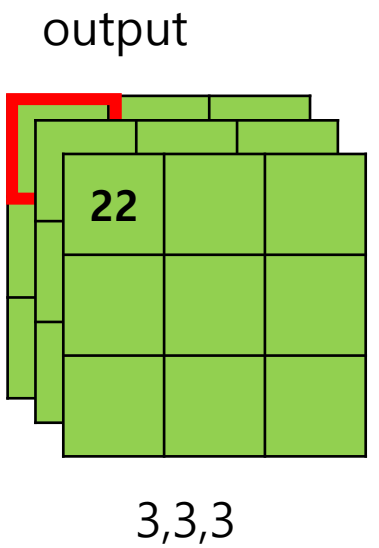
3,3,1

CNN

- **CNN with channel**
  - 만약 input이 RGB data가 입력될 때는 입력데이터가 2D가 아닌 3channel이 포함된 3D tensor가 입력된다.
  - 만약 input data의 channel과 kernel의 channel이 다르면 3D output이 도출된다.

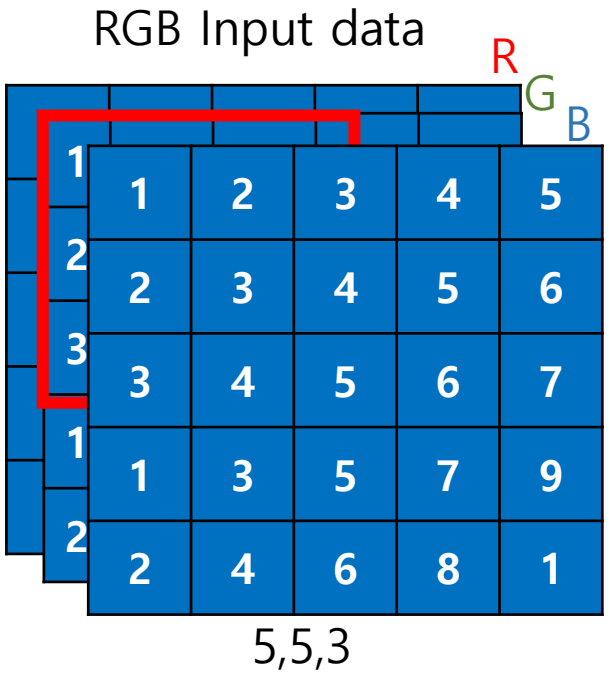


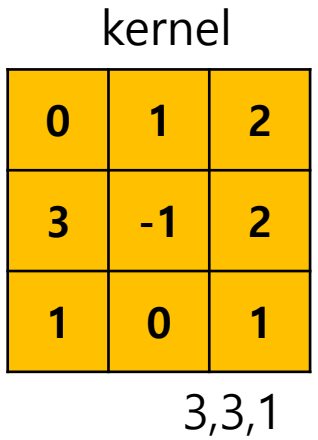


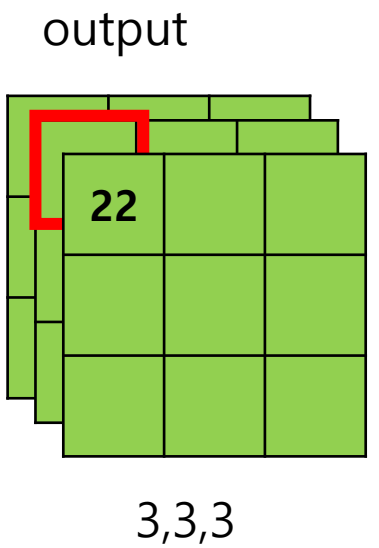


CNN

- **CNN with channel**
  - 만약 input이 RGB data가 입력될 때는 입력데이터가 2D가 아닌 3channel이 포함된 3D tensor가 입력된다.
  - 만약 input data의 channel과 kernel의 channel이 다르면 3D output이 도출된다.

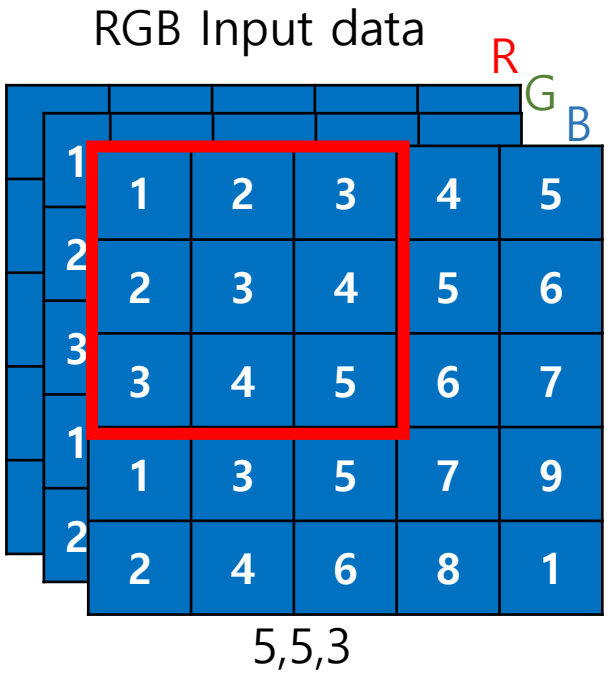


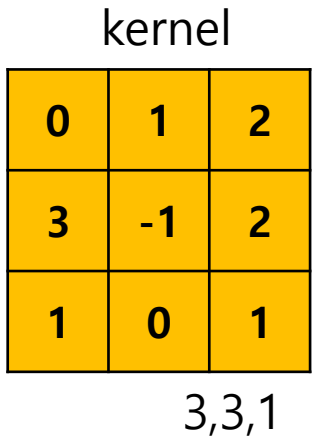


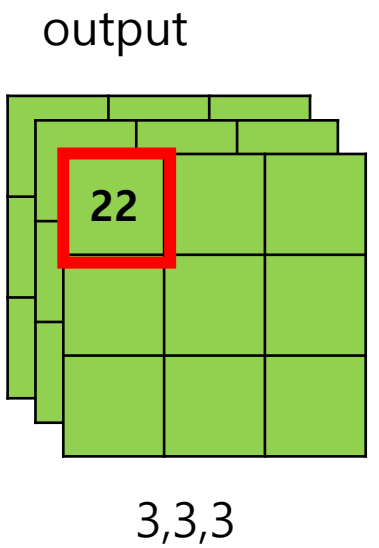


CNN

- **CNN with channel**
  - 만약 input이 RGB data가 입력될 때는 입력데이터가 2D가 아닌 3channel이 포함된 3D tensor가 입력된다.
  - 만약 input data의 channel과 kernel의 channel이 다르면 3D output이 도출된다.

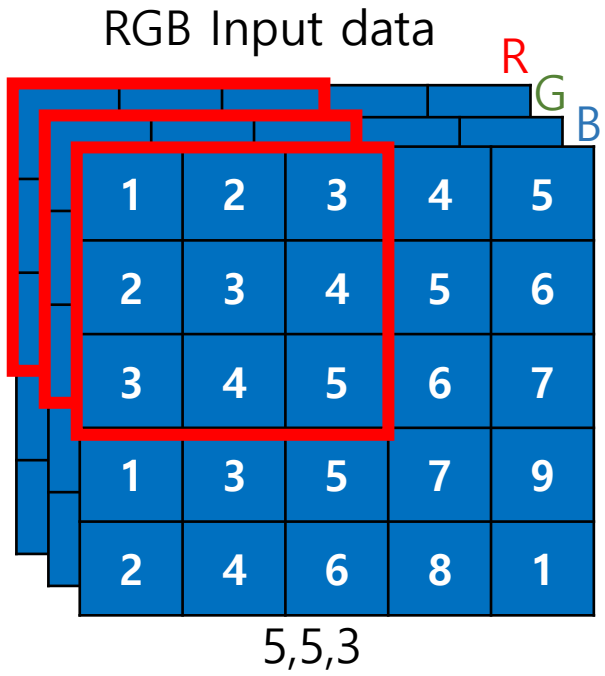


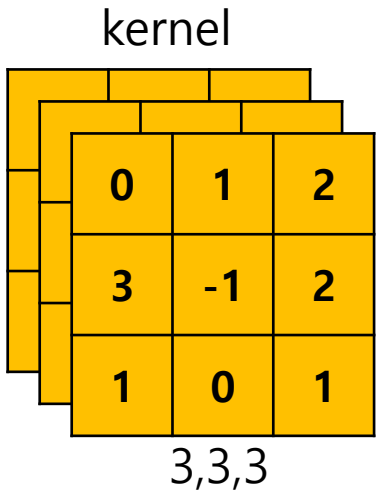


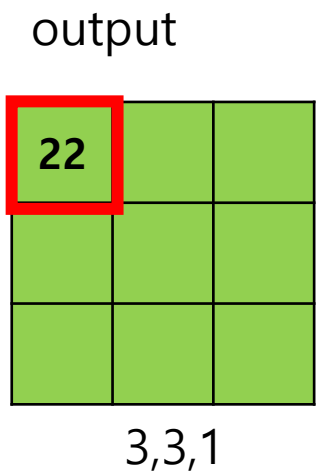


CNN

- **CNN with channel**
  - 2D CNN을 수행하기 위해서는 kernel의 channel이 data의 channel과 동일해야한다.
  - 모든 방향으로 shift하는 convolution연산의 특성상 kernel의 channel이 data의 channel과 동일하면 앞뒤로 shift하지않아 결과가 1 channel로 도출된다.

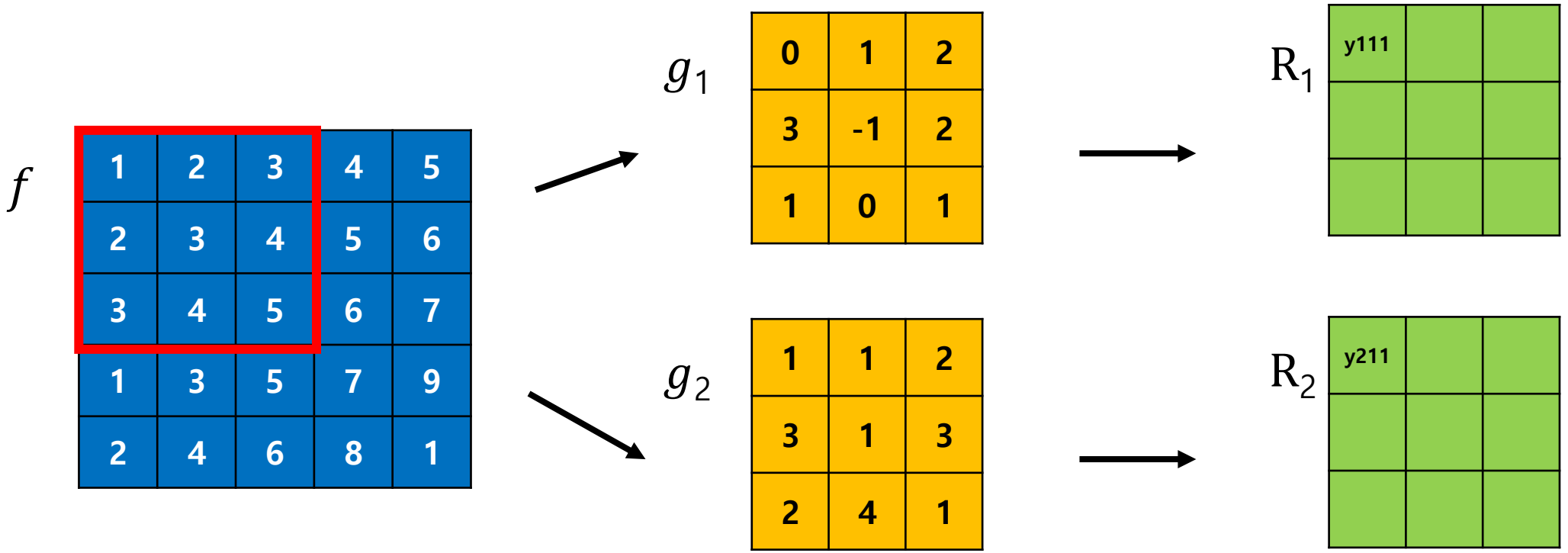






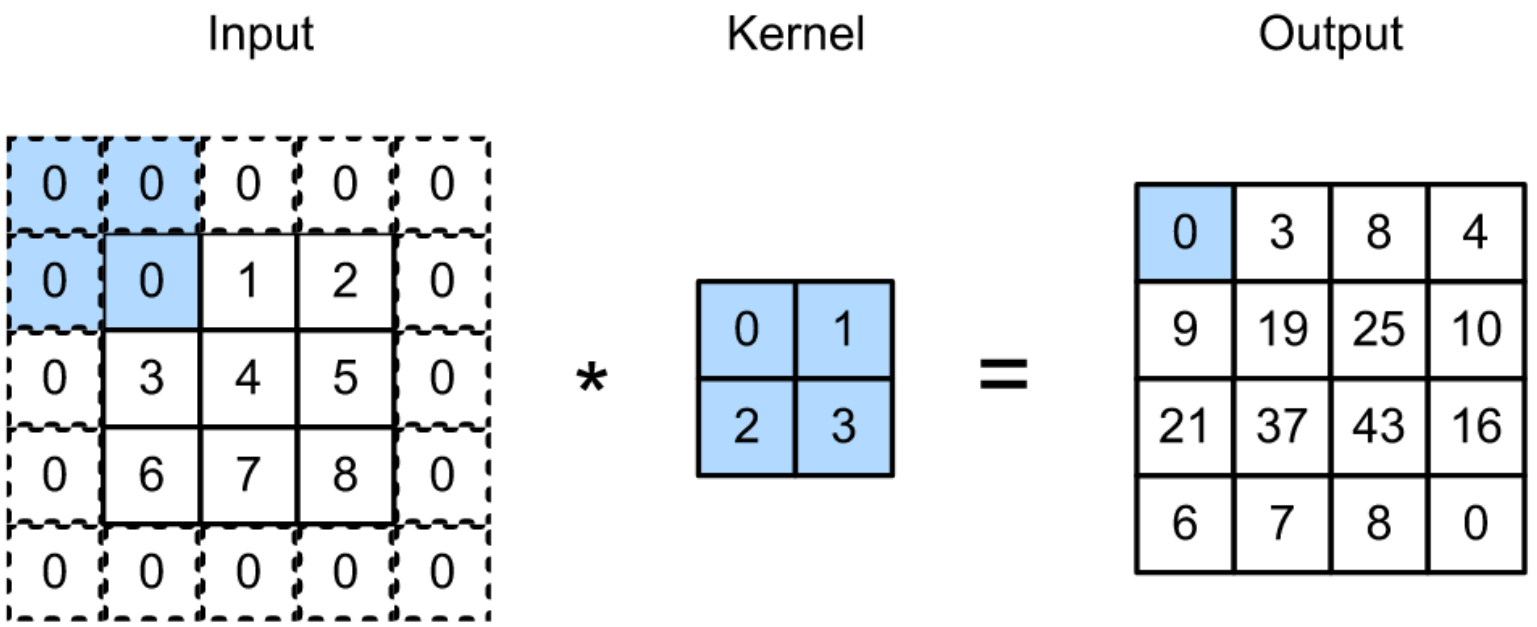
CNN

- **CNN with channel**
  - 2D CNN에서 Output의 channel을 설정하는 방법은 kernel의 개수다.
  - 각 kernel은 1channel의 output을 도출하기 때문에 kernel의 개수를 조정하여 output의 channel을 조정할 수 있다.



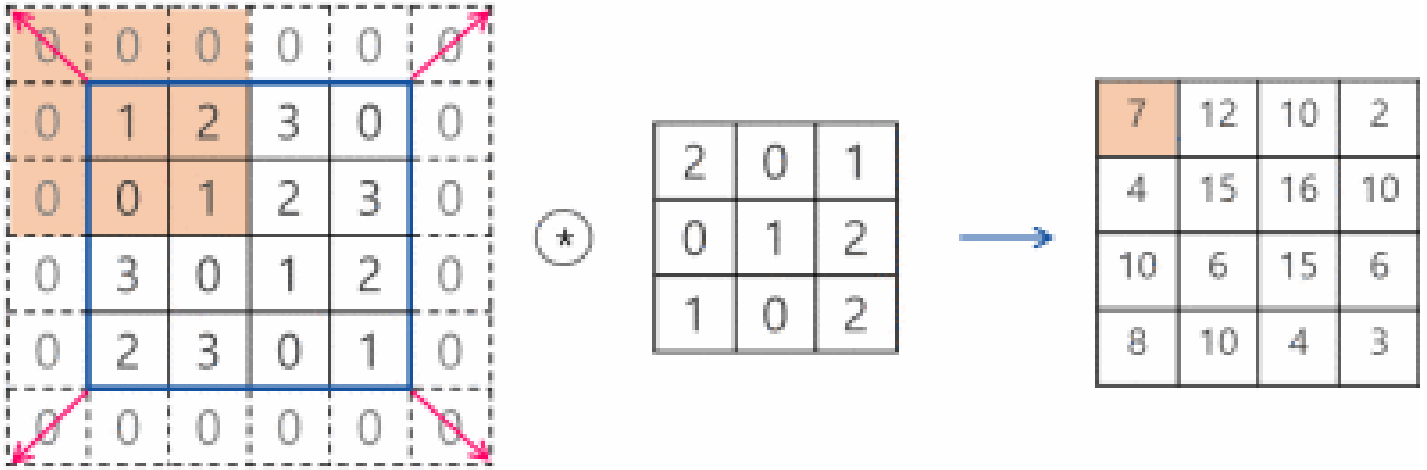
CNN

- **padding**
  - Padding을 사용하지 않은 CNN은 매 layer가 지날때 마다 tensor size가 줄어든다.
  - 하지만 image의 삭제된 부분을 복구하는 inpainting과 같은 경우에는 image의 크기를 줄여서는 안된다.
  - 패딩(Padding)은 합성곱 연산을 수행하기 전, 입력데이터 주변을 특정값으로 채워 늘리는 것을 말한다.
  - 이럴때 padding을 줌으로써 size를 유지하거나 늘릴 수 있다.



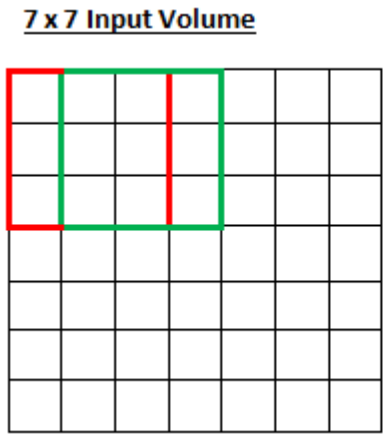
CNN

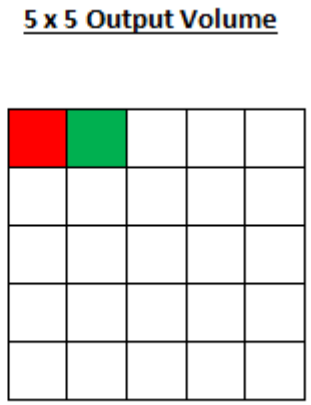
- **padding**
  - Padding을 사용하지 않은 CNN은 매 layer가 지날때 마다 tensor size가 줄어든다.
  - 하지만 image의 삭제된 부분을 복구하는 inpainting과 같은 경우에는 image의 크기를 줄여서는 안된다.
  - 이럴때 padding을 줌으로써 size를 유지하거나 늘릴 수 있다.



CNN

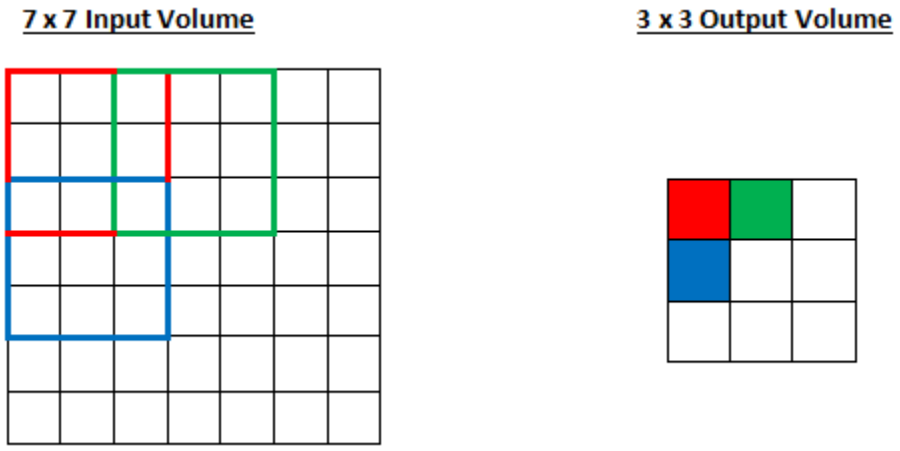
- **stride**
  - 스트라이드는 입력데이터에 kernel를 적용할 때 이동할 간격을 조절하는 것, 즉 kernel이 이동할 간격을 말한다.
  - 7x7 input과 3x3 kernel에서 stride가 1인경우에는 다음과 같이 연산된다.





CNN

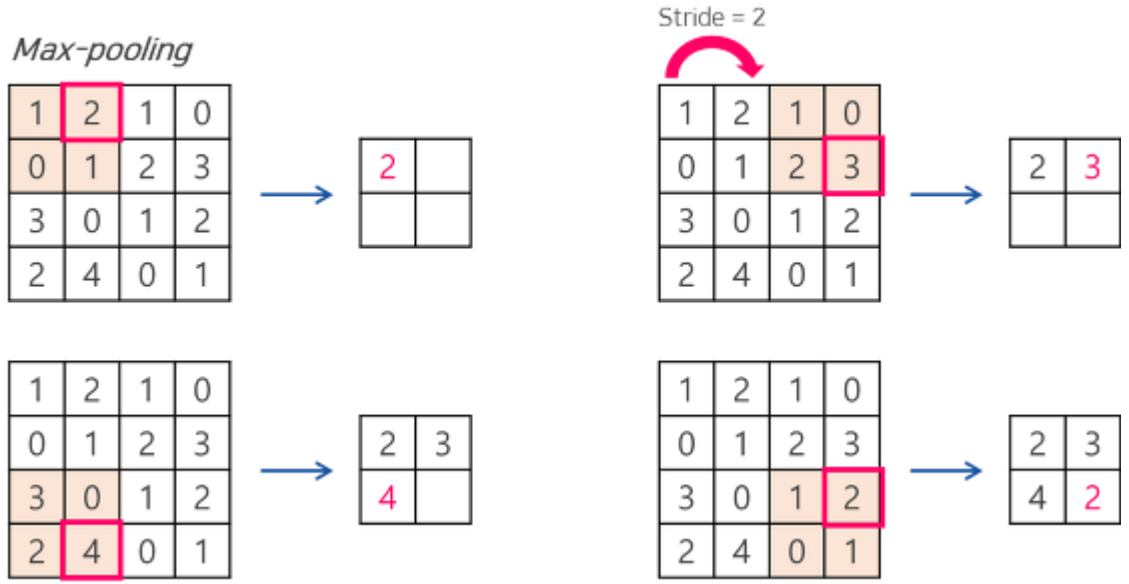
- **stride**
  - 스트라이드는 입력데이터에 kernel를 적용할 때 이동할 간격을 조절하는 것, 즉 kernel이 이동할 간격을 말한다.
  - 7x7 input과 3x3 kernel에서 **stride가 2인경우**에는 다음과 같이 연산된다.



CNN

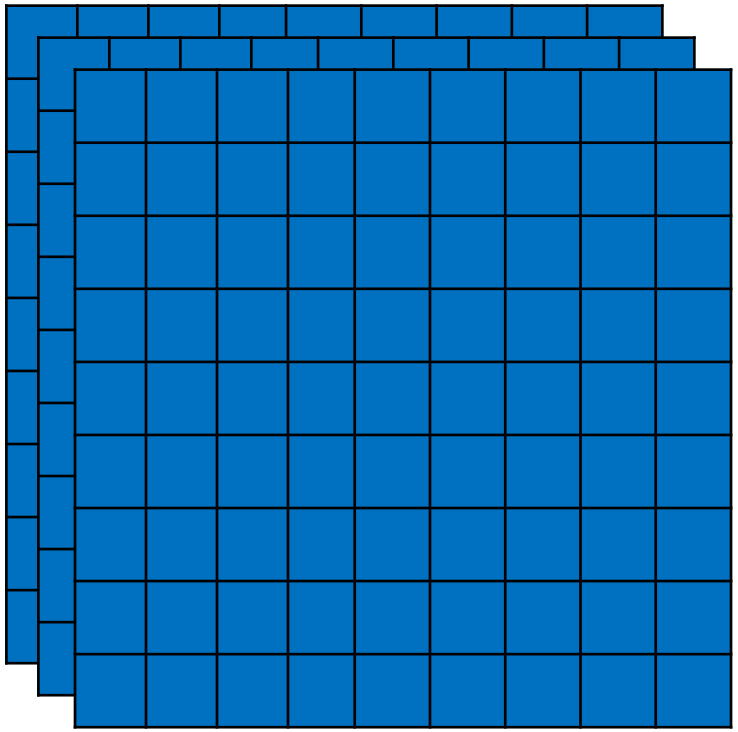
- **Pooling layer**
  - Pooling layer는 스트라이드와 같이 데이터의 tensor 크기를 조정하는것을 목적으로하는 layer이다.
  - 주로 합성곱 계층(CNN)에서 출력데이터의 크기를 입력데이터의 크기 그대로 유지하거나 적은 변동을 주고, 풀링계층(Pooling layer) 에서만 크기를 조절한다.
  - 풀링에는 Max-Pooling과 Average pooling이 있는데 Max-Pooling은 해당영역에서 최대값을 찾는 방법이고, Average-Pooling은 해당영역의 평균값을 계산하는 방법이다.

Window size가 2,2
Stride가 2인 max pooling

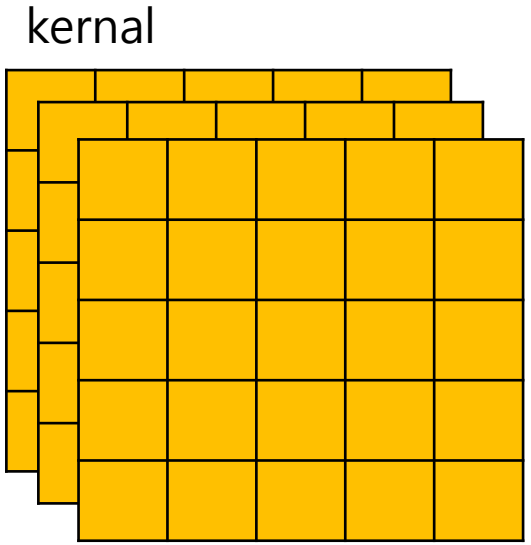


CNN

- **CNN in-out shape**
  - Input이 9*9*3이고 kernel의 size가 5*5*3, kernel이 두개일때



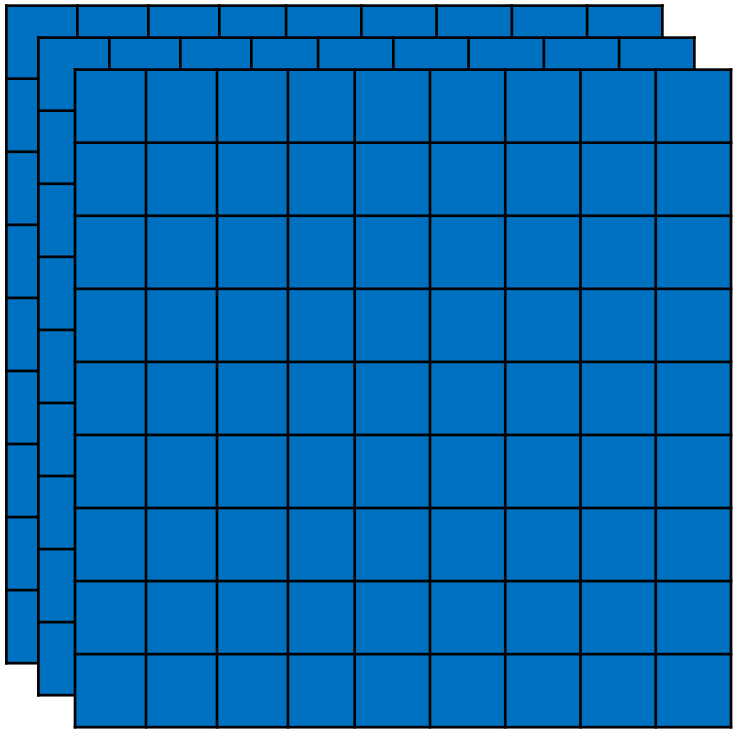
9,9,3

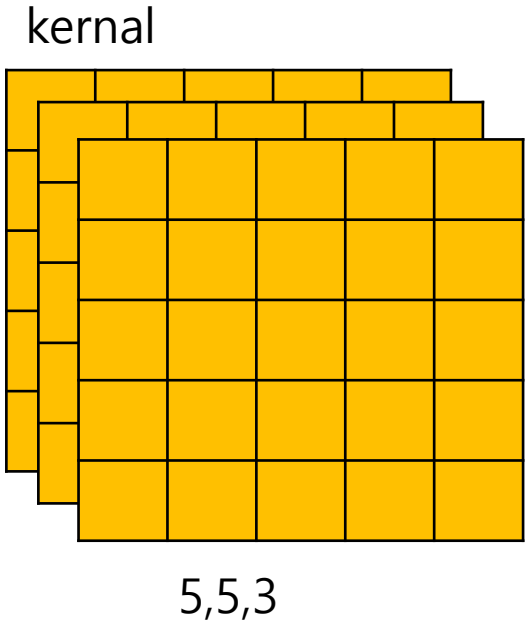


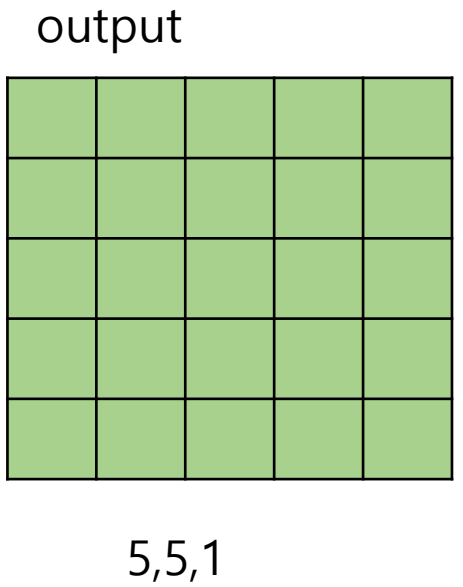
5,5,3

CNN

- **CNN in-out shape**
  - Input이 9*9*3이고 kernel의 size가 5*5*3, kernel이 두개일때







CNN

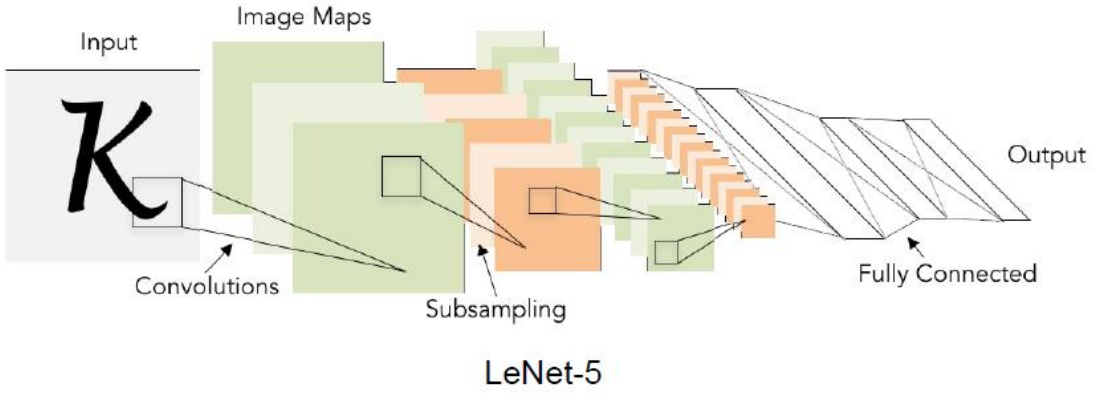
- **CNN in-out shape**
  - Input이 9*9*3이고 kernel의 size가 5*5*3, kernel이 두개일때

$$(OH, OW) = \left(\frac{H + 2P - FH}{S} + 1 \ , \frac{W + 2P - FW}{S} + 1\right)$$

- $(H,\ W)$: 입력크기
- $(FH,\ FW)$: 필터크기
- $(OH,\ OW)$: 출력크기
- P : 패딩
- S : 스트라이드

LeNet

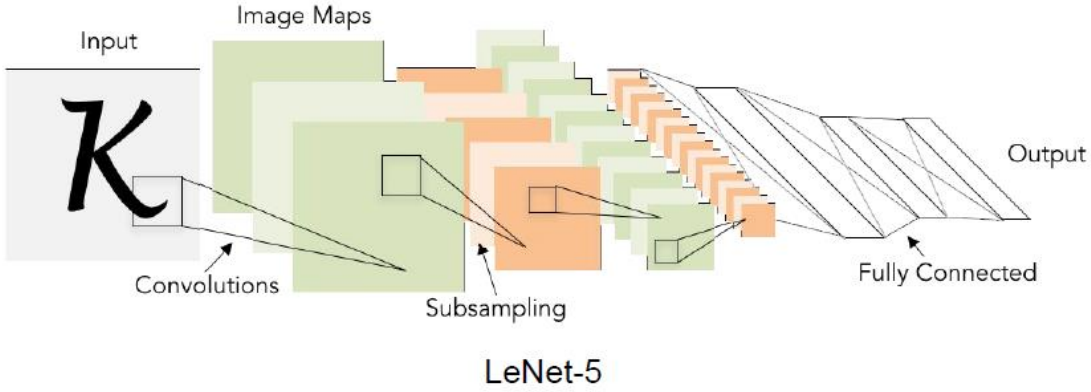
- LeNet-5



LeNet-5

- Input: 32x32 (grayscale) → N x 1 x 32 x 32
- First layer: 5x5 conv, 6 kernels, stride: 1, padding: 0
- Q) What is outputsize of the first layer?
- Second layer: average pooling 2x2
- Third layer: 5x5 conv, 16 kernels, stride: 1, padding: 0
- Fourth layer: 2x2 average pooling
- Fifth layer: 5x5 conv, 120 kernels.

LeNet

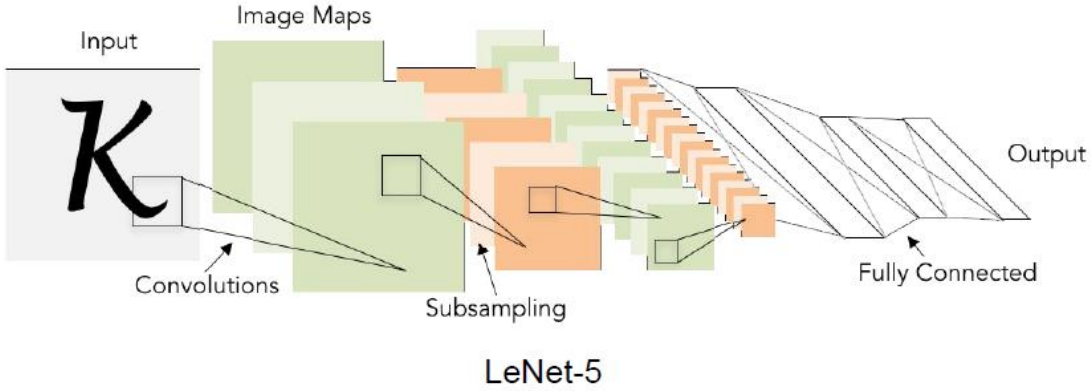


LeNet-5

- LeNet-5

	output
• Input: 32x32 (grayscale) → N x 1 x 32 x 32	N x 1 x 32 x 32
• First layer: 5x5 conv, 6 kernels, stride: 1, padding: 0	N x 6 x 28 x 28
• Q) What is outputsize of the first layer?	
• Second layer: average pooling 2x2	N x 6 x 14 x 14
• Third layer: 5x5 conv, 16 kernels, stride: 1, padding: 0	N x 16 x 10 x 10
• Fourth layer: 2x2 average pooling	N x 16 x 5 x 5
• Fifth layer: 5x5 conv, 120 kernels.	N x 120 x 1 x 1

LeNet



- CIFAR10 with LeNet-5

CIFAR10 dataset을 LeNet을 통해 이미지 분류

LeNet은 Classification을 위한 모델이기 때문에 최종 output의 형태가 class의 개수와 같아야 한다.

LeNet의 구조와 동일하지만 input data만 3channel이라 가정

이후 MLP를 사용하여 목적 class의 차원수로 변환하는 과정을 거침

구현에 진행할 MLP는 2개의 layer로 구성

1st-layer MLP는 100개의 차원의 결과 도출

2st-layer는 output layer로 사용하여 class갯수에 맞게 도출

각 CNNlayer와 MLP는 activation function으로 ReLU사용

Image processing in pytorch

- **Image Load**
  - torchvision을 이용한 image load
    - torchvision.io.read_image(path): 경로의 image를 array like한 형태로 불러오는 함수
    - torchvision.datasets: torchvision에서 제공되는 dataset을 불러오는 함수
      torch.utils.data.Dataset의 형태로 반환되어 바로 사용가능
      https://pytorch.org/vision/stable/datasets.html

```
#torchvision을 통한 dataset load
trainDataset = torchvision.datasets.FashionMNIST(root='./data', train=True,
                                                 download=True, transform=transform)
testDataset = torchvision.datasets.FashionMNIST(root='./data', train=False,
                                                download=True, transform=transform)
```

Image processing in pytorch

- **Image Load**
  - Torchvision.transforms
  - image의 크기조정이나 scaling, normalization과 같은 전처리를 지원하는 함수

  - transforms.compose([transform list]):
    여러 개의 transform을 수행해야 할때 파이프라인화 하는 함수

  - transforms.ToTensor():
    array like한 data를 tensor 형태로 변환하는 함수
  - transforms.Normalize(mean,std):
    각 data를 정규분포의 형태에 맞춰 scaling해주는 함수
  - transforms.Resize([new image size]):
    이미지의 크기를 변경하는 함수
  - 이외 다양한 기능 존재 https://pytorch.org/vision/stable/transforms.html 참조

```
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize(0.5,0.5)])
```

CNN in pytorch

- **Torch.nn.2DConv**
  - 2D CNN을 수행하는 클래스
  - 입력되는 파라미터로는 input_channel, out_channel, kernel_size 등이 있다.
  - kernal의 크기 및 개수를 결정하는 요인인 input_channel, out_channel, kernel_size는 필수 paramter로 사용됨

```
self.conv1 = nn.Conv2d(in_channels=1, out_channels=6, kernel_size=5)
```

```
x = self.relu(self.conv1(x))
```

pooling in pytorch

- **Torch.nn.2DMaxPooling**
  - 2D max pooling을 수행하는 class
  - 입력되는 parameter은 window size가 입력된다.

```
self.pool = nn.MaxPool2d(2, 2)
```