생성 모델과 시각 지능

Generative Model and Visual Intelligence

03 주차 | 딥러닝 심화

상명대학교 컴퓨터과학과
민 경 하

학습목차

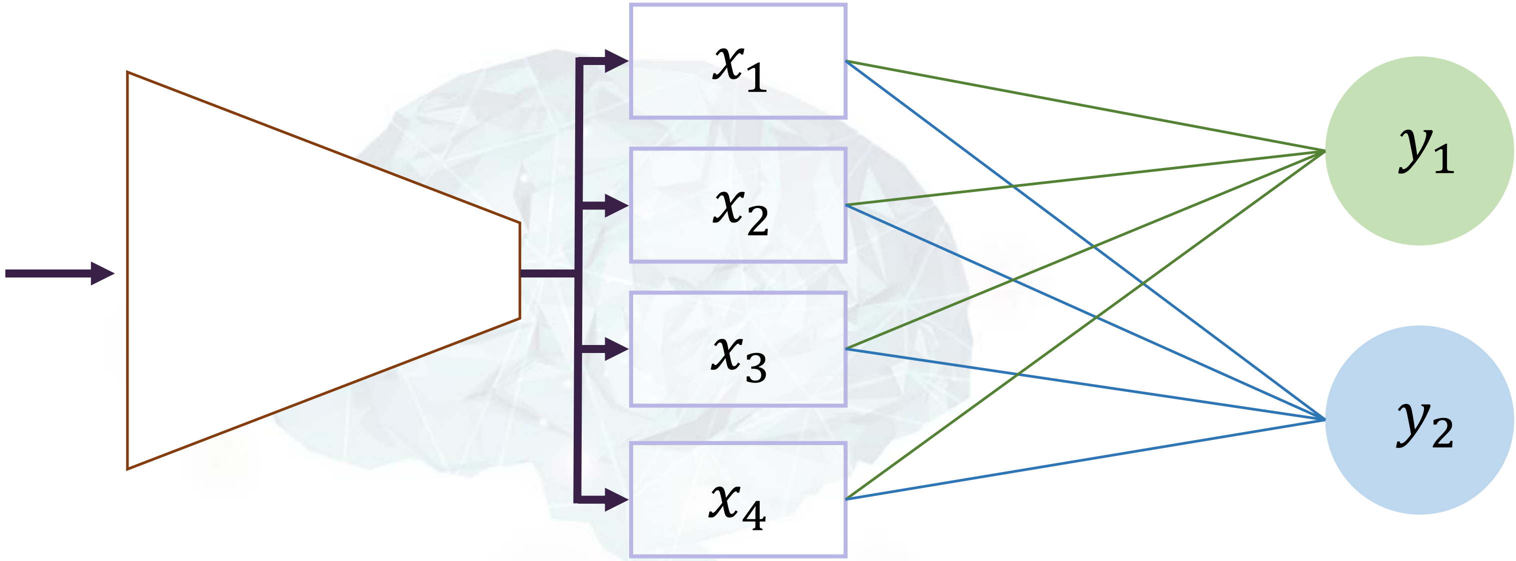
1. 세상 단순한 영상 인식의 원리
2. 세상 단순한 Convolutional Neural Network
3. Convolutional layer (feat. PyTorch)
4. 세상 단순한 CNN의 구현 (feat. PyTorch)

1. 세상 단순한 영상 인식의 원리

세상에서 가장 단순한 영상 인식 문제

영상 인식의 원리

- 영상으로부터 추출한 특징을 비교해서 각 Label의 Predicted probability를 계산함
- 단순한 특징에서 복합적인 특징까지 단계별로 특징을 추출함



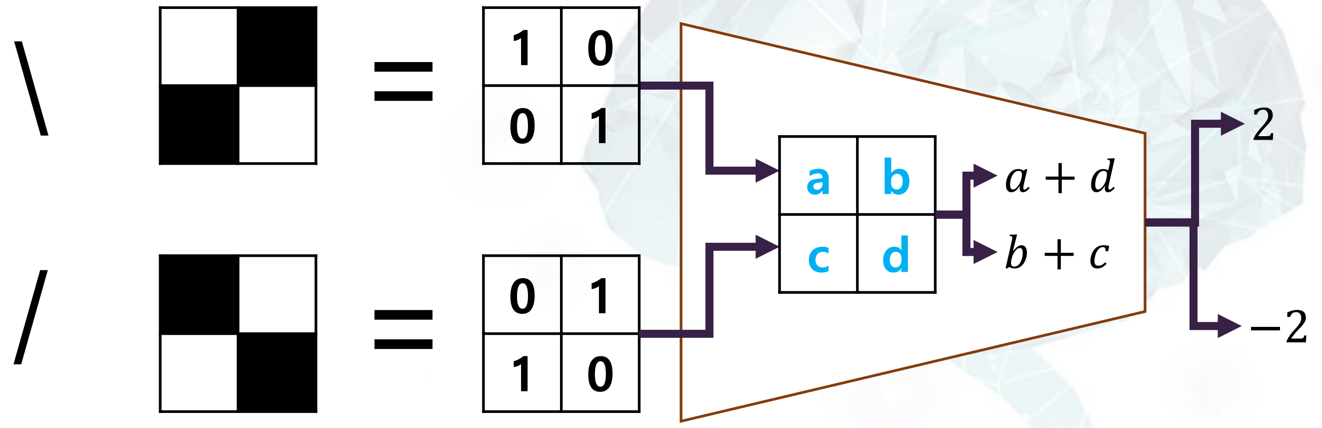
출처: https://unsplash.com/photos/LATYeZyw88c
출처: https://unsplash.com/photos/1AIYdIb3O5M

세상에서 가장 단순한 영상 인식 문제

특징 추출

- 적절한 필터 (filter)를 곱해서 나온 값 → 특징 (Feature)
- 필터를 적용한 특징 추출
  - » image의 각 원소에 값을 곱해서 더하는 연산 (convolution)을 수행

두 영상의 특징을 가장 잘 보여주는 필터 $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$는?



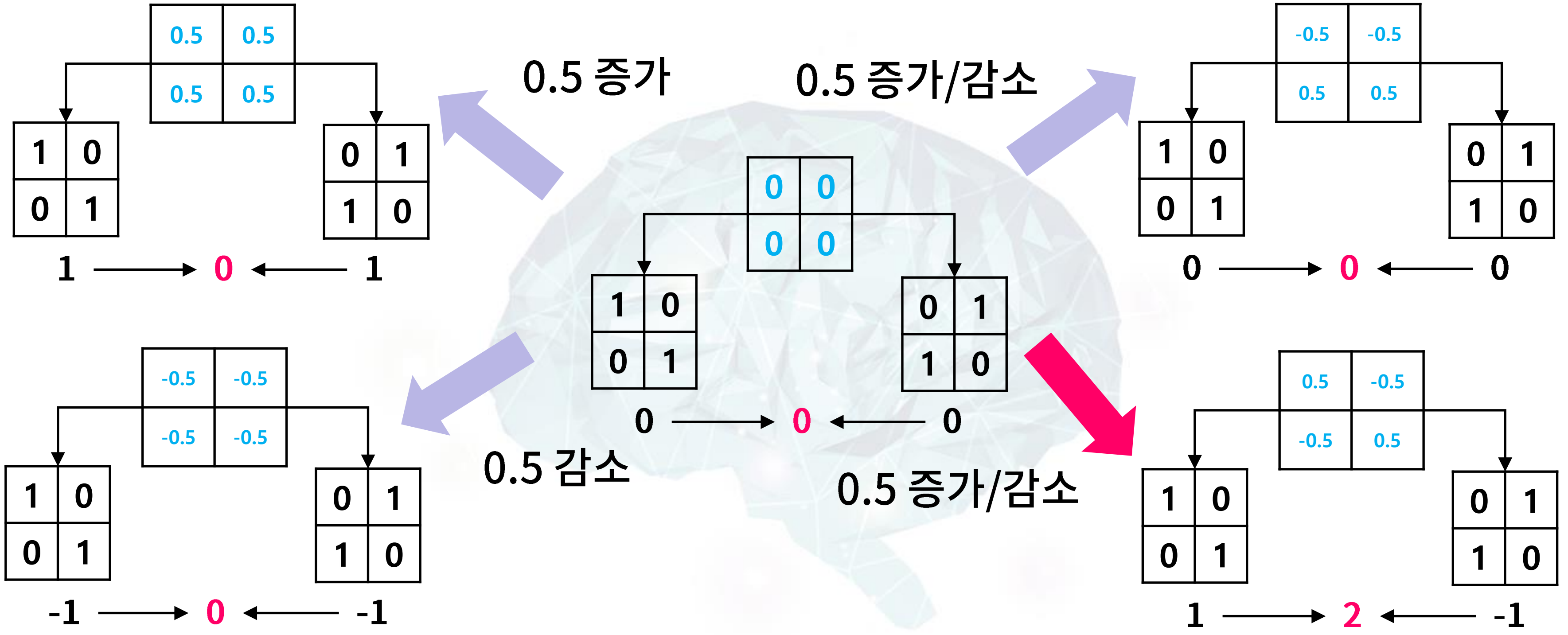
두 영상의 특징을
가장 잘 대비시키는 $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$는?

1	-1
-1	1

세상에서 가장 단순한 영상 인식 문제

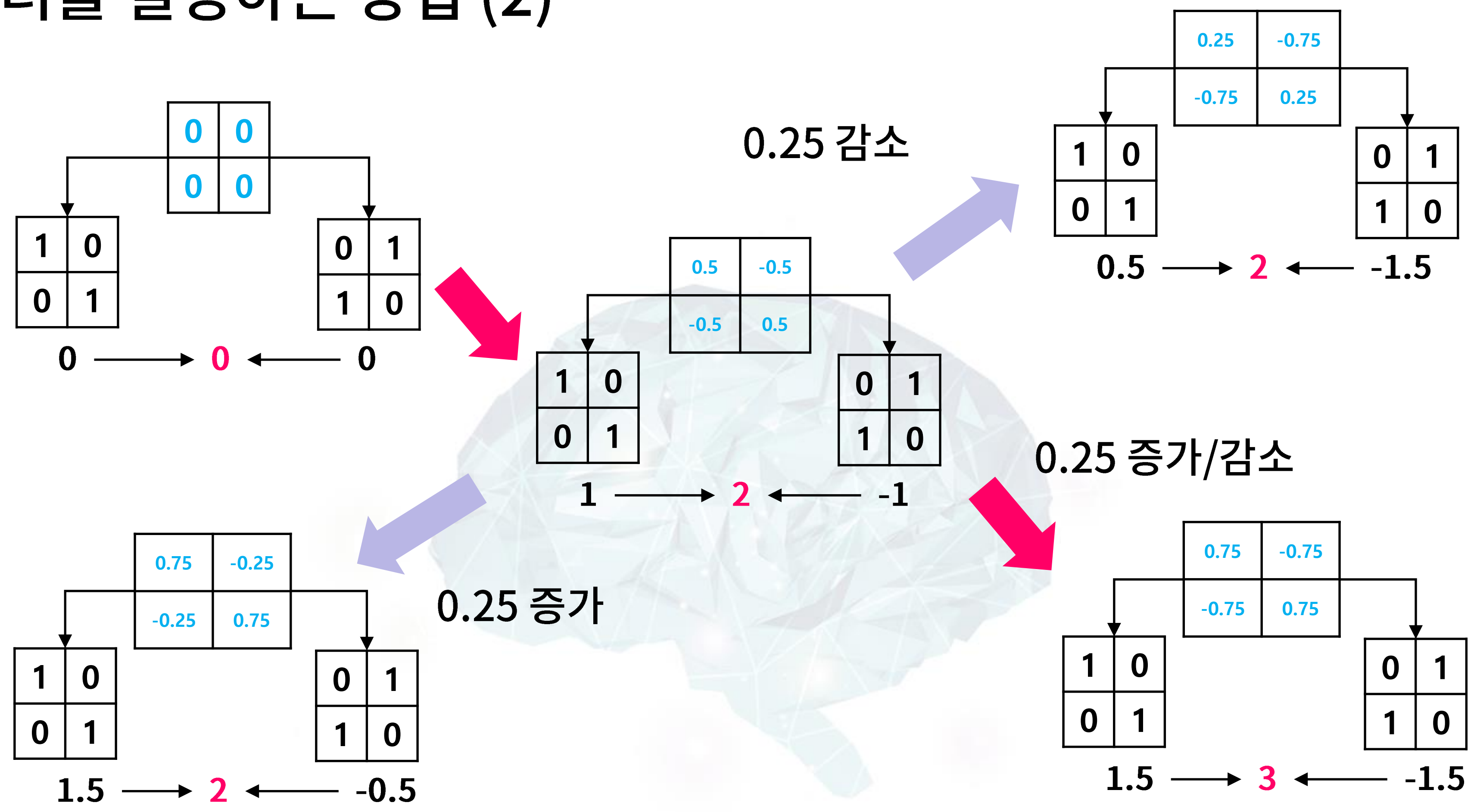
필터를 결정하는 방법 (1)

- explicit method: 사람이 적절한 필터 값을 계산 (고전적)
- implicit method: 학습을 통해서 적절한 필터 값을 결정



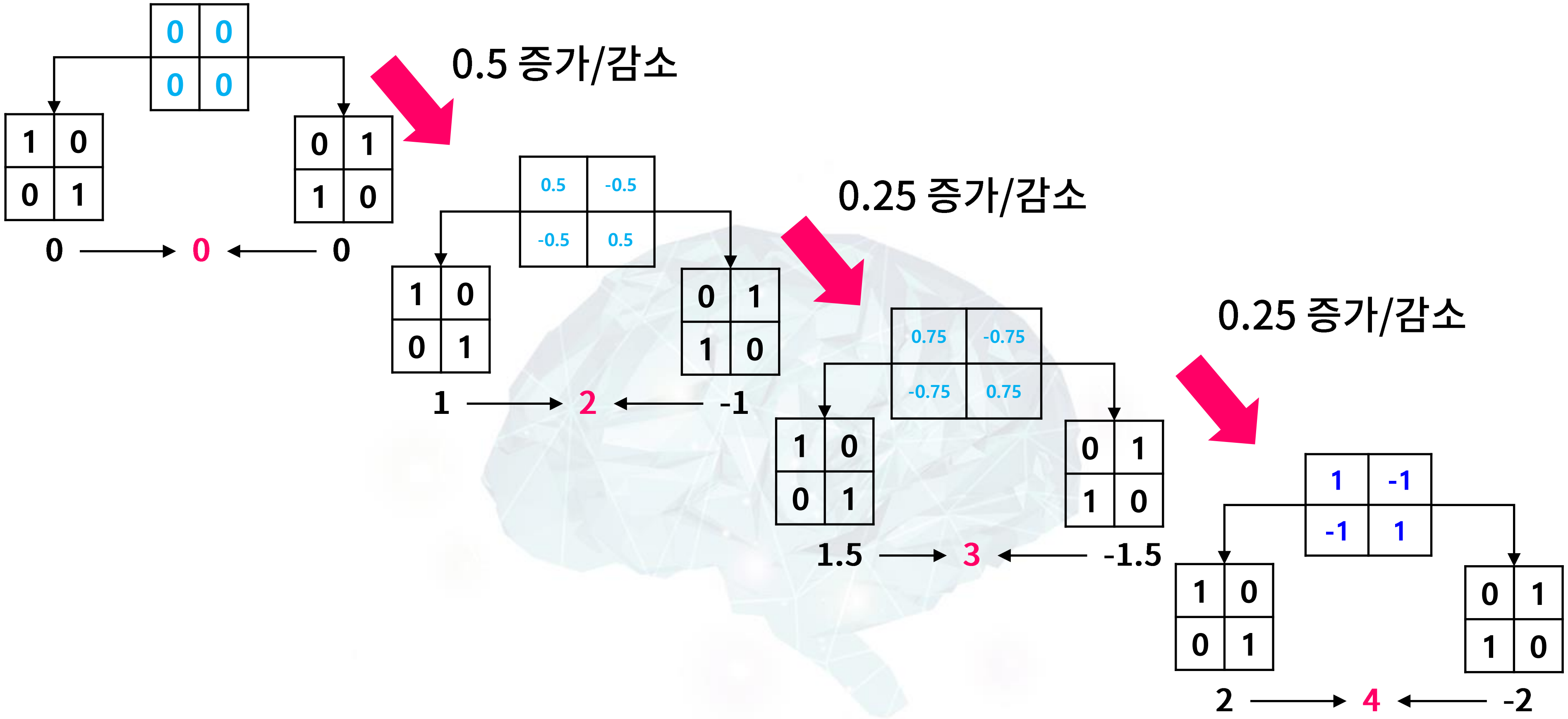
세상에서 가장 단순한 영상 인식 문제

필터를 결정하는 방법 (2)



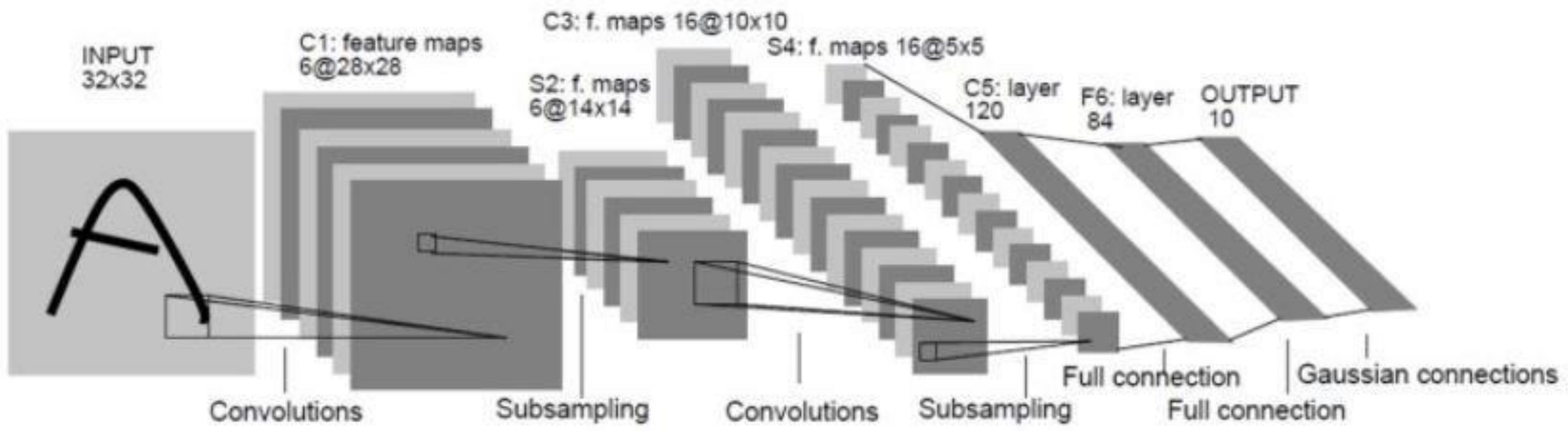
세상에서 가장 단순한 영상 인식 문제

필터를 결정하는 방법 (3)



2. 세상 단순한 Convolutional Neural Network

세계 최초의 CNN → LeNet5 (1998)

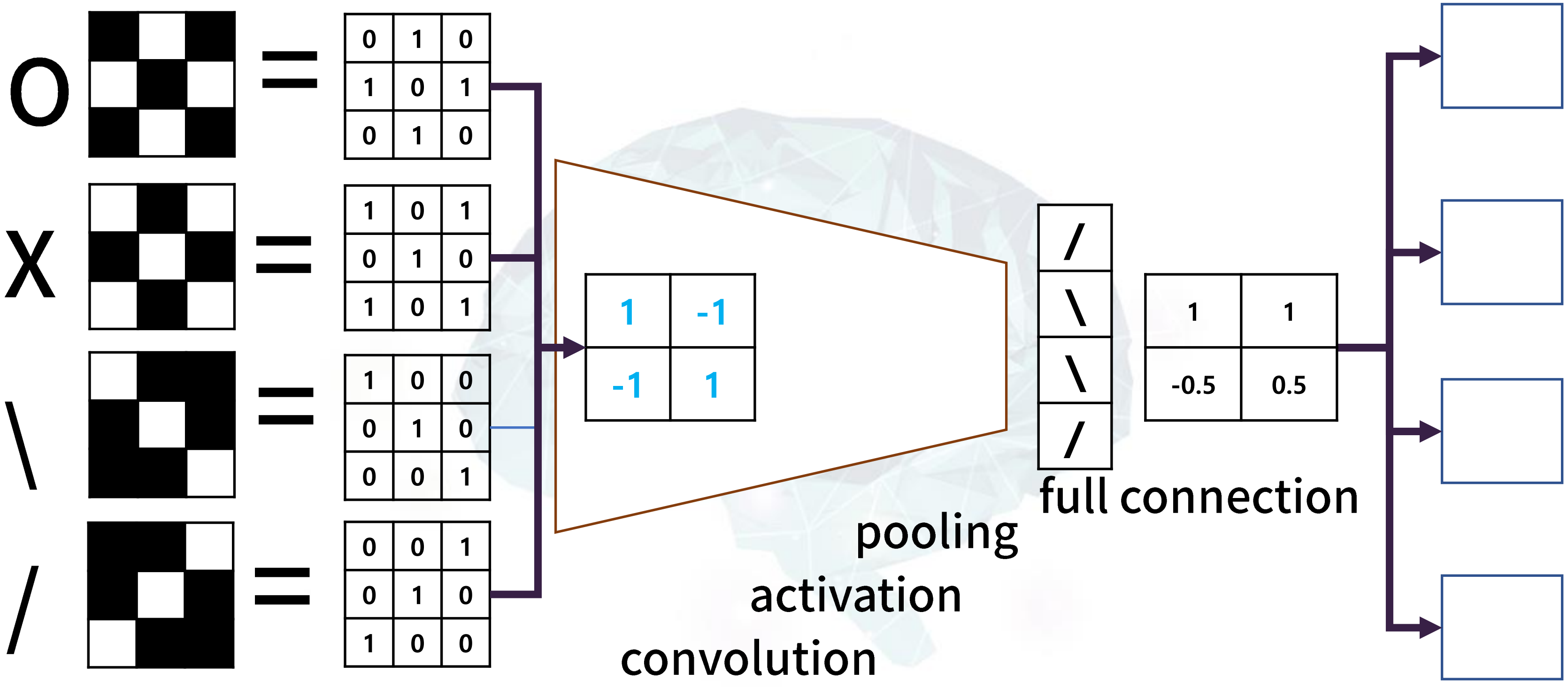


- Colvolution → 특징 (feature map) 추출
- Subsampling (Pooling) → 특징의 정보 압축
- Activation → 중요하지 않은 특징 제거
- Full connection → 최종 결정

출처: https://www.ytimes.co.kr/news/articleView.html?idxno=6013

세계 최초의 CNN → LeNet5 (1998)

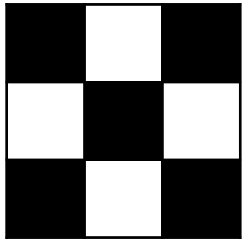
- 예) 3x3 영상에서의 (0, X, \, /) 인식
  - 세상 단순한 필터를 적용한 추출
  - 고정된 필터를 사용하는 인식 문제 (단순)



CNN의 4 가지 layer의 역할을 고려한 영상 인식

- Convolution → 특징 추출 (O의 경우)

$$\begin{bmatrix}0&1&0\\1&0&1\\0&1&0\end{bmatrix}$$



$$\begin{bmatrix}0&1&0\\1&0&1\\0&1&0\end{bmatrix} \otimes \begin{bmatrix}1&-1\\-1&1\end{bmatrix} = -2 = /$$

$$\begin{bmatrix}0&1&0\\1&0&1\\0&1&0\end{bmatrix} \otimes \begin{bmatrix}1&-1\\-1&1\end{bmatrix} = 2 = \backslash$$

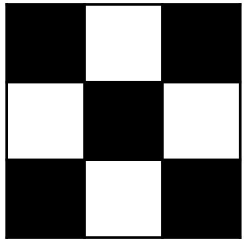
$$\begin{bmatrix}0&1&0\\1&0&1\\0&1&0\end{bmatrix} \otimes \begin{bmatrix}1&-1\\-1&1\end{bmatrix} = 2 = \backslash$$

$$\begin{bmatrix}0&1&0\\1&0&1\\0&1&0\end{bmatrix} \otimes \begin{bmatrix}1&-1\\-1&1\end{bmatrix} = -2 = /$$

CNN의 4 가지 layer의 역할을 고려한 영상 인식

- Activation → 중요하지 않은 특징 제거 (0의 경우)

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$



$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} = -2 = /$$

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} = 2 = \backslash$$

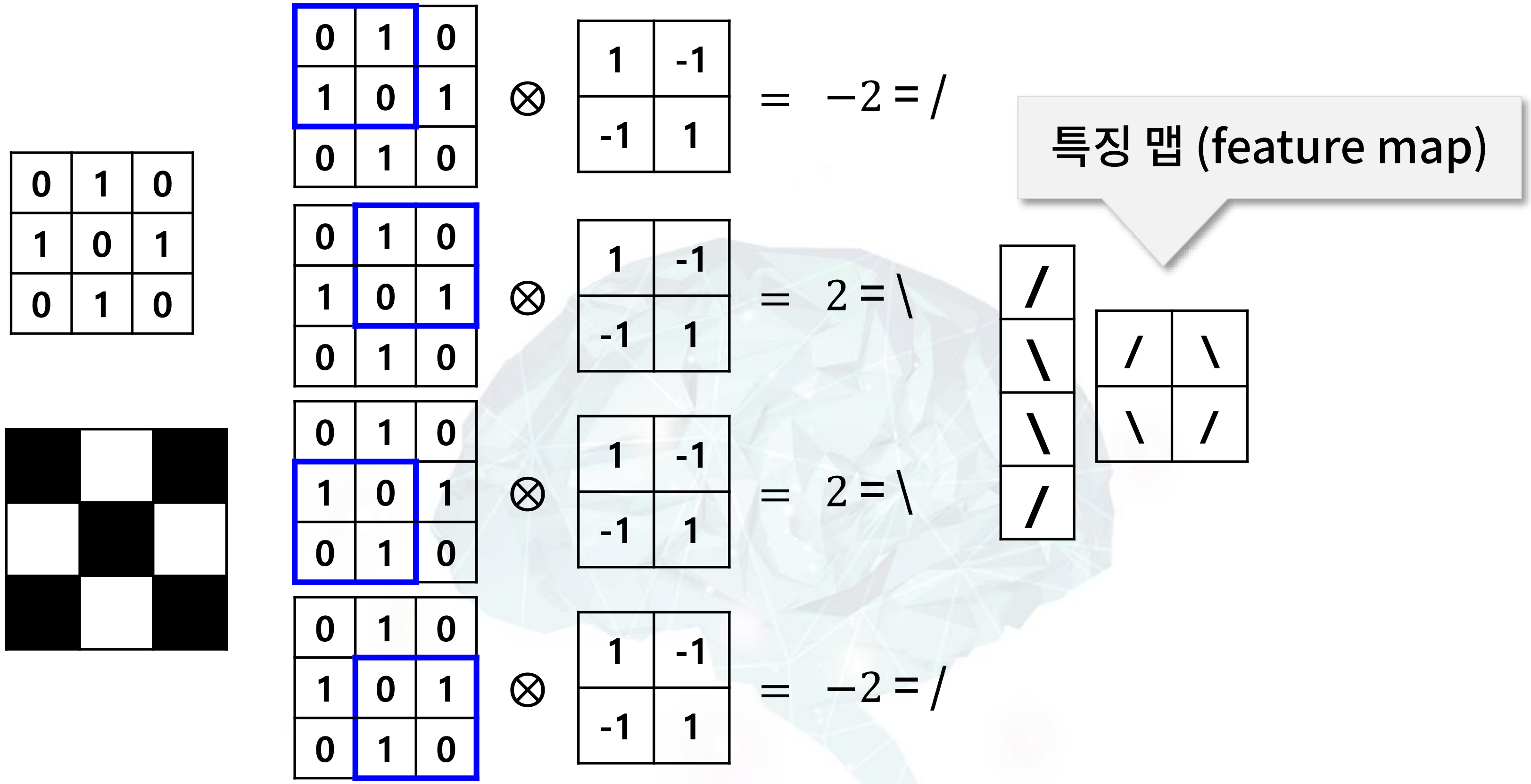
$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} = 2 = \backslash$$

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} = -2 = /$$

/	\
\	/

CNN의 4 가지 layer의 역할을 고려한 영상 인식

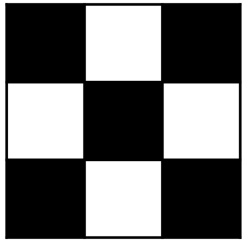
- Pooling → 픽셀들의 정보 압축 (O의 경우)



CNN의 4 가지 layer의 역할을 고려한 영상 인식

- Fully connected → 최종 결정 (O의 경우)

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$



$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} = -2 = /$$

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} = 2 = \backslash$$

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} = 2 = \backslash$$

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} = -2 = /$$

$$\begin{bmatrix} / \\ \backslash \\ \backslash \\ / \end{bmatrix} \quad \begin{bmatrix} / & \backslash \\ \backslash & / \end{bmatrix}$$

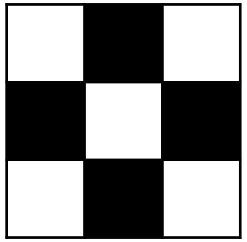
특징 맵 (feature map)

$$\begin{bmatrix} -2 & 2 \\ 2 & -2 \end{bmatrix} \otimes \begin{bmatrix} 1 & 1 \\ -0.5 & 0.5 \end{bmatrix} = -2$$

CNN의 4 가지 layer의 역할을 고려한 영상 인식

- X의 경우는?

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$



$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} = -2 = /$$

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} = 2 = \backslash$$

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} = 2 = \backslash$$

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} = -2 = /$$

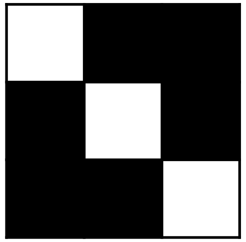
$$\begin{bmatrix} \backslash \\ / \\ / \\ \backslash \end{bmatrix} \quad \begin{bmatrix} \backslash & / \\ / & \backslash \end{bmatrix}$$

$$\begin{bmatrix} 2 & -2 \\ -2 & 2 \end{bmatrix} \otimes \begin{bmatrix} 1 & 1 \\ -0.5 & 0.5 \end{bmatrix} = -2$$

CNN의 4 가지 layer의 역할을 고려한 영상 인식

- \의 경우는?

$$\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}$$



$$\begin{bmatrix}\boxed{1}&\boxed{0}&0\\\boxed{0}&\boxed{1}&0\\0&0&1\end{bmatrix} \otimes \begin{bmatrix}1&-1\\-1&1\end{bmatrix} = 2 = \backslash$$

$$\begin{bmatrix}1&\boxed{0}&\boxed{0}\\0&\boxed{1}&\boxed{0}\\0&0&1\end{bmatrix} \otimes \begin{bmatrix}1&-1\\-1&1\end{bmatrix} = -1 = 0$$

$$\begin{bmatrix}1&0&0\\\boxed{0}&\boxed{1}&0\\\boxed{0}&\boxed{0}&1\end{bmatrix} \otimes \begin{bmatrix}1&-1\\-1&1\end{bmatrix} = -1 = 0$$

$$\begin{bmatrix}1&0&0\\0&\boxed{1}&\boxed{0}\\0&\boxed{0}&\boxed{1}\end{bmatrix} \otimes \begin{bmatrix}1&-1\\-1&1\end{bmatrix} = 2 = \backslash$$

$$\begin{bmatrix}\backslash\\0\\0\\\backslash\end{bmatrix}$$

Activation 적용

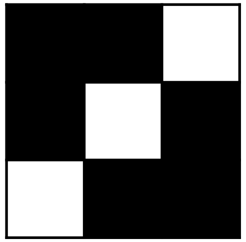
$$\begin{bmatrix}\backslash&0\\0&\backslash\end{bmatrix}$$

$$\begin{bmatrix}2&0\\0&2\end{bmatrix} \otimes \begin{bmatrix}1&1\\-0.5&0.5\end{bmatrix} = 3$$

CNN의 4 가지 layer의 역할을 고려한 영상 인식

- /의 경우는?

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$



$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} = 1 = 0$$

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} = -2 = /$$

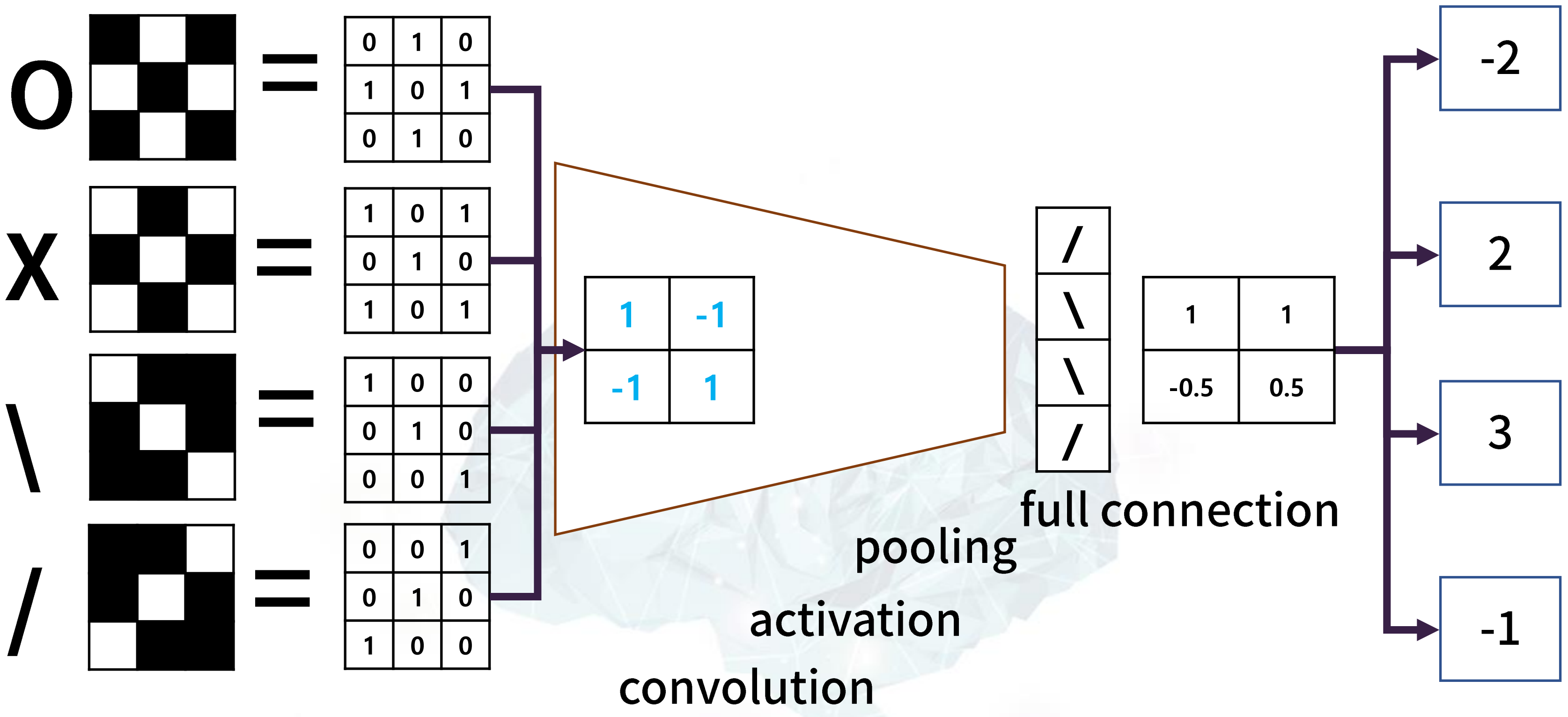
$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} = -2 = /$$

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} = 1 = 0$$

$$\begin{bmatrix} 0 \\ / \\ / \\ 0 \end{bmatrix} \quad \begin{bmatrix} 0 & / \\ / & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & -2 \\ -2 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & 1 \\ -0.5 & 0.5 \end{bmatrix} = -1$$

Convolutional neural network를 이용한 (0,x,/,\) 분류



3. Convolutional layer (feat. PyTorch)

Convolution (합성곱)

- 주어진 함수에 g (kernel, filter)를 곱해서 더하는 연산

$$(f * g)(t) \equiv \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

$$\begin{bmatrix} 99 & 99 & 99 & 0 & 0 & 0 \\ 99 & 99 & 99 & 0 & 0 & 0 \\ 99 & 99 & 99 & 0 & 0 & 0 \\ 99 & 99 & 99 & 0 & 0 & 0 \\ 99 & 99 & 99 & 0 & 0 & 0 \\ 99 & 99 & 99 & 0 & 0 & 0 \end{bmatrix} \otimes \underbrace{\frac{1}{3}\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}}_{\text{kernel}} = \begin{bmatrix} \; & \; & \; & \; \\ \; & \; & \; & \; \\ \; & \; & \; & \; \\ \; & \; & \; & \; \end{bmatrix}$$

Convolution (합성곱)

- 주어진 함수에 g (kernel, filter)를 곱해서 더하는 연산

$$(f * g)(t) \equiv \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

- 영상에서는 입력 영상에 작은 크기의 kernel 영상을 pixel-by-pixel로 곱해서 더한 영상을 생성하는 과정을 의미함

Convolution (합성곱)

- 주어진 함수에 g (kernel, filter)를 곱해서 더하는 연산

$$(f * g)(t) \equiv \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \otimes \underbrace{\frac{1}{3}\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}}_{\text{kernel}} = \begin{bmatrix} \boxed{0} & & & \\ & & & \\ & & & \\ & & & \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

$$\Downarrow$$

$$\frac{1}{3}\begin{bmatrix} 1\text{x}1 & 1\text{x}0 & 1\text{x}(-1) \\ 1\text{x}1 & 1\text{x}0 & 1\text{x}(-1) \\ 1\text{x}1 & 1\text{x}0 & 1\text{x}(-1) \end{bmatrix}$$

$$+$$

$$\Downarrow$$

$$0$$

Convolution (합성곱)

- 주어진 함수에 g (kernel, filter)를 곱해서 더하는 연산

$$(f * g)(t) \equiv \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \otimes \underbrace{\frac{1}{3}\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}}_{\text{kernel}} = \begin{bmatrix} 0 & \boxed{1} & & \\ & & & \\ & & & \\ & & & \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

$$\Downarrow$$

$$\frac{1}{3}\begin{bmatrix} 1\text{x}1 & 1\text{x}0 & 0\text{x}(-1) \\ 1\text{x}1 & 1\text{x}0 & 0\text{x}(-1) \\ 1\text{x}1 & 1\text{x}0 & 0\text{x}(-1) \end{bmatrix}$$

$$+$$

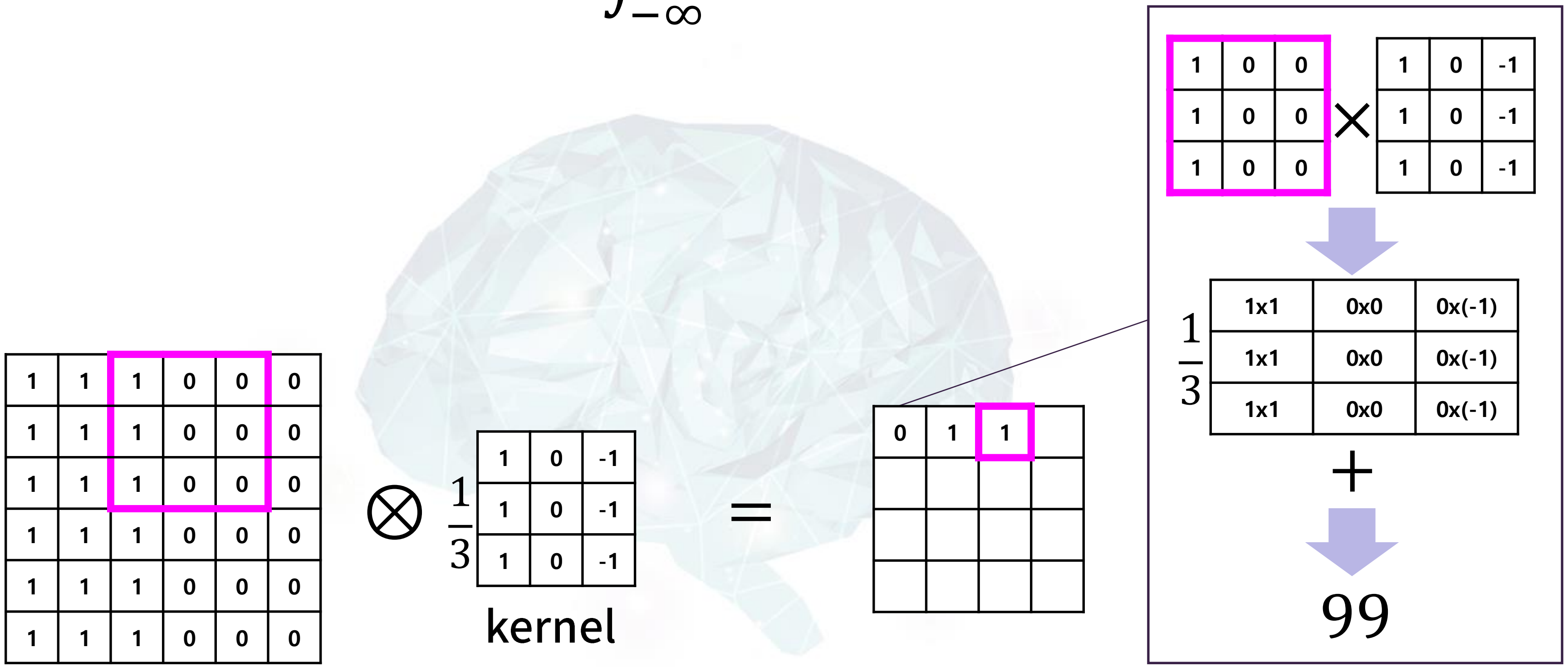
$$\Downarrow$$

$$99$$

Convolution (합성곱)

- 주어진 함수에 g (kernel, filter)를 곱해서 더하는 연산

$$(f * g)(t) \equiv \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$



Convolution (합성곱)

- 주어진 함수에 g (kernel, filter)를 곱해서 더하는 연산

$$(f * g)(t) \equiv \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \otimes \underbrace{\frac{1}{3}\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}}_{\text{kernel}} = \begin{bmatrix} 0 & 1 & 1 & \boxed{0} \\ & & & \\ & & & \\ & & & \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

$$\downarrow$$

$$\frac{1}{3}\begin{bmatrix} 0\text{x}1 & 0\text{x}0 & 0\text{x}(-1) \\ 0\text{x}1 & 0\text{x}0 & 0\text{x}(-1) \\ 0\text{x}1 & 0\text{x}0 & 0\text{x}(-1) \end{bmatrix}$$

$$+$$

$$\downarrow$$

$$0$$

Convolution (합성곱)

- 주어진 함수에 g (kernel, filter)를 곱해서 더하는 연산

$$(f * g)(t) \equiv \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \otimes \underbrace{\frac{1}{3}\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}}_{\text{kernel}} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

Convolution (합성곱)

Convolution의 주요 속성

- size (kernel의 크기): 3x3, 4x4, 5x5, etc
- stride (kernel의 적용 단위): 1, 2, 3, etc
- padding (입력 영상의 주변): None, 0, 1, etc

size	3x3
stride	1
padding	None

$$
\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \otimes \underbrace{\frac{1}{3}\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}}_{\text{kernel}} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}
$$

Convolution (합성곱)

- Convolution의 주요 속성
  - size (kernel의 크기): 3x3, 4x4, 5x5, etc
  - stride (kernel의 적용 단위): 1, 2, 3, etc
  - padding (입력 영상의 주변): None, 0, 1, etc

size	4x4
stride	1
padding	None

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \otimes \frac{1}{4} \underbrace{\begin{bmatrix} 1 & 0 & 0 & -1 \\ 1 & 0 & 0 & -1 \\ 1 & 0 & 0 & -1 \\ 1 & 0 & 0 & -1 \end{bmatrix}}_{\text{kernel}} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Convolution (합성곱)

- Convolution의 주요 속성
  - size (kernel의 크기): 3x3, 4x4, 5x5, etc
  - stride (kernel의 적용 단위): 1, 2, 3, etc
  - padding (입력 영상의 주변): None, 0, 1, etc

size	5x5
stride	1
padding	None

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \otimes \frac{1}{5} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 & -1 \end{bmatrix}}_{\text{kernel}} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

Convolution (합성곱)

Convolution의 주요 속성

- size (kernel의 크기): 3x3, 4x4, 5x5, etc
- stride (kernel의 적용 단위): 1, 2, 3, etc
- padding (입력 영상의 주변): None, 0, 1, etc

size	3x3
stride	2
padding	None

$$\begin{bmatrix} \boxed{\begin{matrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{matrix}} & \begin{matrix}0 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 0\end{matrix} \\ \begin{matrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{matrix} & \begin{matrix}0 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 0\end{matrix} \end{bmatrix} \otimes \underbrace{\frac{1}{3}\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}}_{\text{kernel}} = \begin{bmatrix} \boxed{0} & \\ & \end{bmatrix}$$

Convolution (합성곱)

Convolution의 주요 속성

- size (kernel의 크기): 3x3, 4x4, 5x5, etc
- stride (kernel의 적용 단위): 1, 2, 3, etc
- padding (입력 영상의 주변): None, 0, 1, etc

size	3x3
stride	2
padding	None

$$\begin{bmatrix}1&1&1&0&0&0\\1&1&1&0&0&0\\1&1&1&0&0&0\\1&1&1&0&0&0\\1&1&1&0&0&0\\1&1&1&0&0&0\end{bmatrix} \otimes \underset{\text{kernel}}{\frac{1}{3}\begin{bmatrix}1&0&-1\\1&0&-1\\1&0&-1\end{bmatrix}} = \begin{bmatrix}0&1\\ & \end{bmatrix}$$

Convolution (합성곱)

Convolution의 주요 속성

- size (kernel의 크기): 3x3, 4x4, 5x5, etc
- stride (kernel의 적용 단위): 1, 2, 3, etc
- padding (입력 영상의 주변): None, 0, 1, etc

size	3x3
stride	2
padding	None

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \otimes \underbrace{\frac{1}{3}\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}}_{\text{kernel}} = \begin{bmatrix} 0 & 1 \\ 0 & \end{bmatrix}$$

Convolution (합성곱)

Convolution의 주요 속성

- size (kernel의 크기): 3x3, 4x4, 5x5, etc
- stride (kernel의 적용 단위): 1, 2, 3, etc
- padding (입력 영상의 주변): None, 0, 1, etc

size	3x3
stride	2
padding	None

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \otimes \underbrace{\frac{1}{3}\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}}_{\text{kernel}} = \begin{bmatrix} 0 & 1 \\ 0 & \boxed{1} \end{bmatrix}$$

Convolution (합성곱)

- Convolution의 주요 속성
  - size (kernel의 크기): 3x3, 4x4, 5x5, etc
  - stride (kernel의 적용 단위): 1, 2, 3, etc
  - padding (입력 영상의 주변): None, 0, 1, etc

size	3x3
stride	1
padding	None

$$\begin{array}{|c|c|c|c|c|c|}\hline 1&1&1&0&0&0\\\hline 1&1&1&0&0&0\\\hline 1&1&1&0&0&0\\\hline 1&1&1&0&0&0\\\hline 1&1&1&0&0&0\\\hline 1&1&1&0&0&0\\\hline\end{array} \otimes \frac{1}{3}\begin{array}{|c|c|c|}\hline 1&0&-1\\\hline 1&0&-1\\\hline 1&0&-1\\\hline\end{array} = \begin{array}{|c|c|c|c|}\hline 0&1&1&0\\\hline 0&1&1&0\\\hline 0&1&1&0\\\hline 0&1&1&0\\\hline\end{array}$$

kernel

Convolution (합성곱)

- Convolution의 주요 속성
  - size (kernel의 크기): 3x3, 4x4, 5x5, etc
  - stride (kernel의 적용 단위): 1, 2, 3, etc
  - padding (입력 영상의 주변): None, 0, 1, etc

size	3x3
stride	1
padding	0-padding

0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0
0	1	1	1	0	0	0	0
0	1	1	1	0	0	0	0
0	1	1	1	0	0	0	0
0	1	1	1	0	0	0	0
0	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0

$$\otimes \ \frac{1}{3}\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} =$$

kernel

-.6	0	.6	.6	0	0
-1	0	1	1	0	0
-1	0	1	1	0	0
-1	0	1	1	0	0
-1	0	1	1	0	0
-.6	0	.6	.6	0	0

Convolution (합성곱)

Convolution의 주요 속성

- size (kernel의 크기): 3x3, 4x4, 5x5, etc
- stride (kernel의 적용 단위): 1, 2, 3, etc
- padding (입력 영상의 주변): None, 0, 1, etc

size	3x3
stride	1
padding	1-padding

1	1	1	1	1	1	1	1
1	1	1	1	0	0	0	1
1	1	1	1	0	0	0	1
1	1	1	1	0	0	0	1
1	1	1	1	0	0	0	1
1	1	1	1	0	0	0	1
1	1	1	1	0	0	0	1
1	1	1	1	1	1	1	1

$$\otimes \frac{1}{3}\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} =$$

kernel

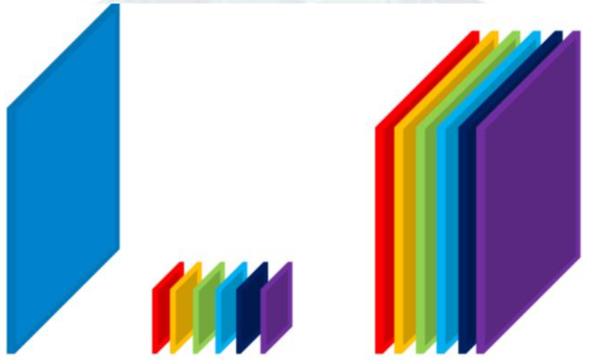
0	0	.6	.6	0	-.6
0	0	1	1	0	-1
0	0	1	1	0	-1
0	0	1	1	0	-1
0	0	1	1	0	-1
0	0	.6	.6	0	-.6

Convolution (합성곱)

Convolution in pytorch

```
torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1,
padding=0,

dilation=1, groups=1, bias=True, padding_mode='zeros')
```

Conv2d(1, 5, 5);



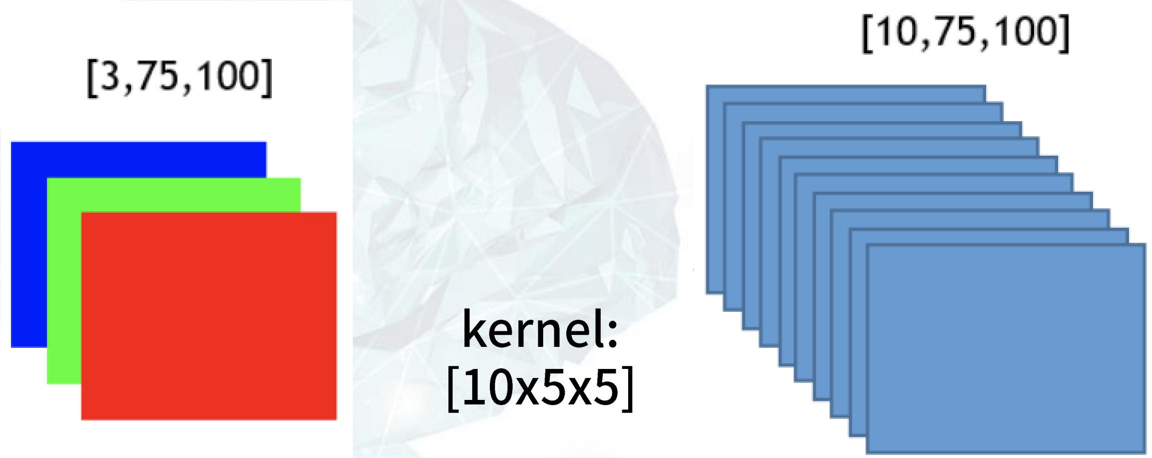
출처: https://velog.io/@gyuho/Convolution-Neural-Network

Convolution (합성곱)

Convolution in pytorch

```
torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0,

dilation=1, groups=1, bias=True, padding_mode='zeros')
```

Conv2d(3, 10, 5, 1, 2);



Convolution과 행렬 곱

- Convolution with 3x3 filter

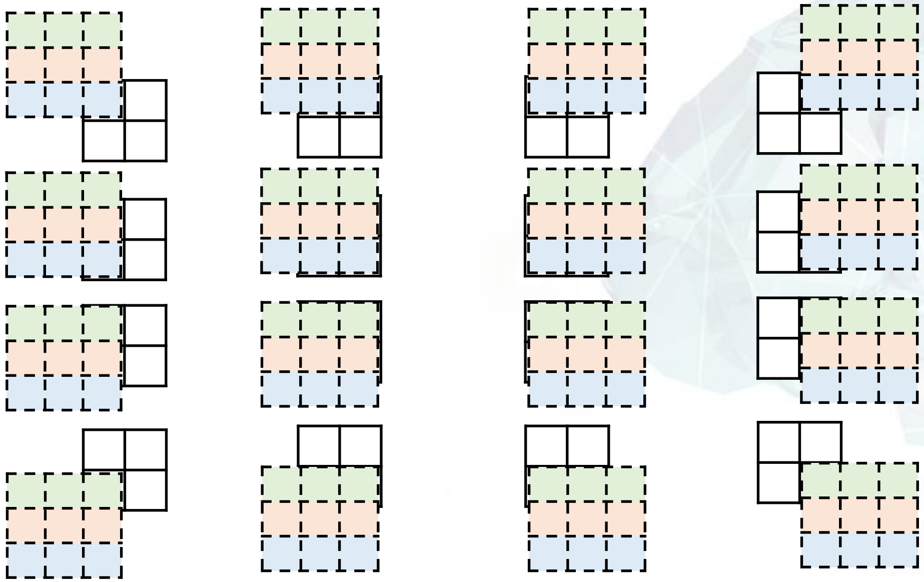
$$\begin{bmatrix} x_0 & x_1 & x_2 & X_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{bmatrix} \otimes \underbrace{\begin{bmatrix} w_1 & w_2 & w_3 \\ w_4 & w_5 & w_6 \\ w_7 & w_8 & w_9 \end{bmatrix}}_{\text{kernel}} \rightarrow \begin{bmatrix} y_0 & y_1 \\ y_2 & y_3 \end{bmatrix}$$

$$\begin{bmatrix} w_1 & w_2 & w_3 & 0 & w_4 & w_5 & w_6 & 0 & w_7 & w_8 & w_9 & 0 & 0 & 0 & 0 & 0 \\ 0 & w_1 & w_2 & w_3 & 0 & w_4 & w_5 & w_6 & 0 & w_7 & w_8 & w_9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_1 & w_2 & w_3 & 0 & w_4 & w_5 & w_6 & 0 & w_7 & w_8 & w_9 & 0 \\ 0 & 0 & 0 & 0 & 0 & w_1 & w_2 & w_3 & 0 & w_4 & w_5 & w_6 & 0 & w_7 & w_8 & w_9 \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ X_{10} \\ x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{15} \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

Transposed Convolution

- Inverse Convolution with 3x3 filter

$$\begin{bmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{bmatrix} \leftarrow \begin{bmatrix} w_1 & w_2 & w_3 \\ w_4 & w_5 & w_6 \\ w_7 & w_8 & w_9 \end{bmatrix} \otimes \begin{bmatrix} y_0 & y_1 \\ y_2 & y_3 \end{bmatrix}$$



$$\begin{bmatrix} w_9 & 0 & 0 & 0 \\ w_8 & w_9 & 0 & 0 \\ w_7 & w_8 & 0 & 0 \\ 0 & w_7 & 0 & 0 \\ w_6 & 0 & w_9 & 0 \\ w_5 & w_6 & w_8 & w_9 \\ w_4 & w_5 & w_7 & w_8 \\ 0 & w_4 & 0 & w_7 \\ w_3 & 0 & w_6 & 0 \\ w_2 & w_3 & w_5 & w_6 \\ w_1 & w_2 & w_4 & w_5 \\ 0 & w_1 & 0 & w_4 \\ 0 & 0 & w_3 & 0 \\ 0 & 0 & w_2 & w_3 \\ 0 & 0 & w_1 & w_2 \\ 0 & 0 & 0 & w_1 \end{bmatrix} \times \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{15} \end{bmatrix}$$

Transposed Convolution

- Inverse Convolution with 3x3 filter
- Convolution filter의 transposed matrix를 곱함
- 영상의 크기가 커지는 연산

w_1	w_2	w_3	0	w_4	w_5	w_6	0	w_7	w_8	w_9	0	0	0	0	0
0	w_1	w_2	w_3	0	w_4	w_5	w_6	0	w_7	w_8	w_9	0	0	0	0
0	0	0	0	w_1	w_2	w_3	0	w_4	w_5	w_6	0	w_7	w_8	w_9	0
0	0	0	0	0	w_1	w_2	w_3	0	w_4	w_5	w_6	0	w_7	w_8	w_9

$\leftrightarrow$

w_9	0	0	0
w_8	w_9	0	0
w_7	w_8	0	0
0	w_7	0	0
w_6	0	w_9	0
w_5	w_6	w_8	w_9
w_4	w_5	w_7	w_8
0	w_4	0	w_7
w_3	0	w_6	0
w_2	w_3	w_5	w_6
w_1	w_2	w_4	w_5
0	w_1	0	w_4
0	0	w_3	0
0	0	w_2	w_3
0	0	w_1	w_2
0	0	0	w_1

상명대학교
SANGMYUNG UNIVERSITY

4. 세상 단순한 CNN의 구현 (feat. PyTorch)

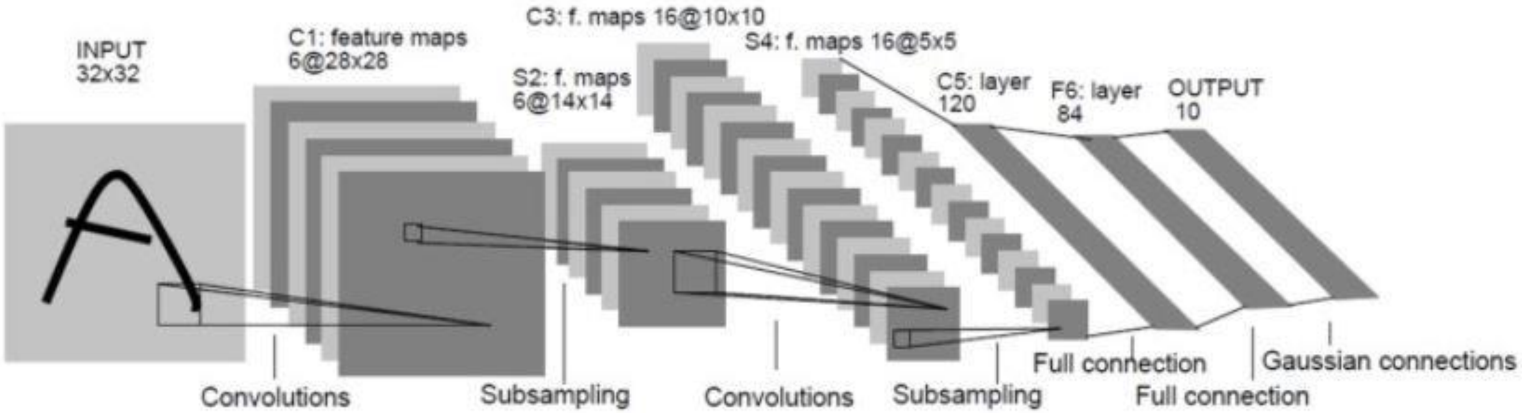
세상 단순한 CNN은 최초의 CNN

LeNet5

구현 과정

1. 모델 정의
2. 데이터 로딩
3. Training
4. Test

(1) 모델 정의



3 Convolution layer
- C1: 1x32x32 → 6x28x28
- C2: 6x14x14 → 16x10x10
- C3: 16x5x5 → 120x1x1

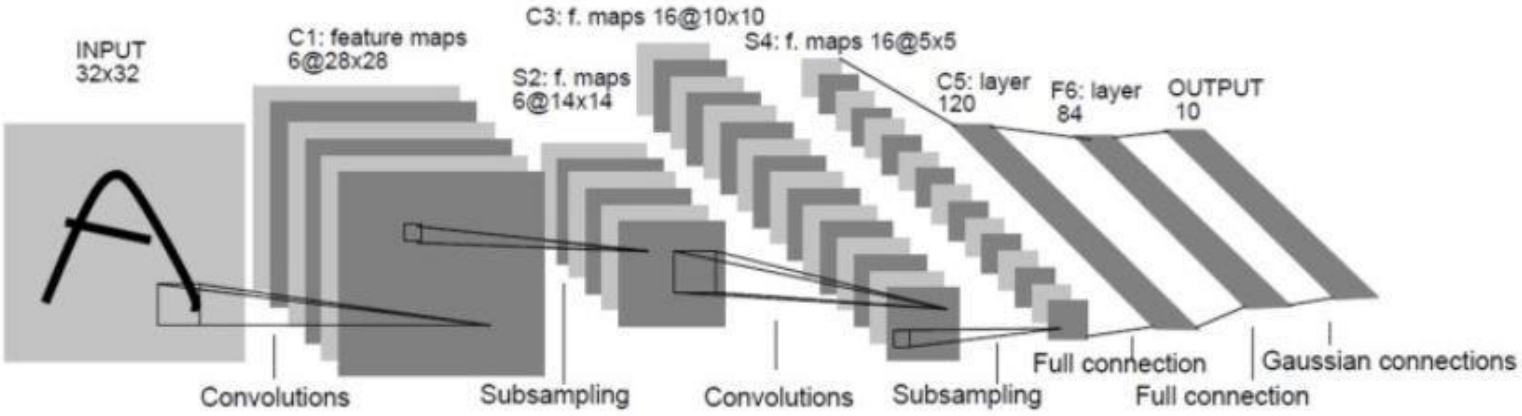
2 Pooling layer
- S1: 28x28 → 14x14
- S2: 10x10 → 5x5

2 Fully connected layer
- F1: 120 → 84
- F2: 84 → 10

- Activation function
  - tanh ( ) 함수 사용

출처: https://medium.com/@jaimin-k/traffic-sign-classification-using-convolutional-neural-network-258a67128c0f

(1) 모델 정의



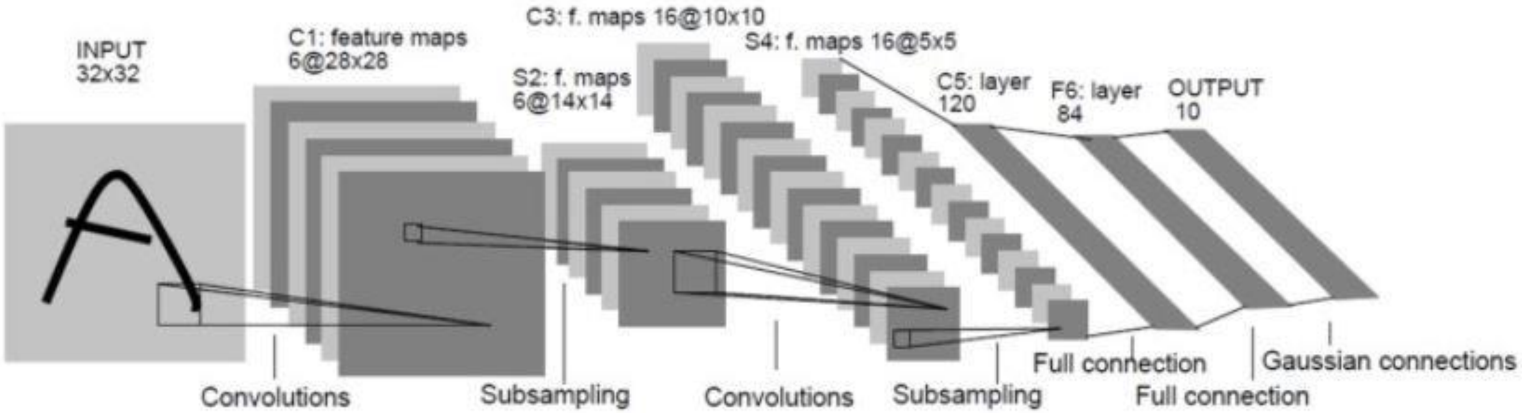
3 Convolution layer
- C1: 1x32x32 → 6x28x28
- C2: 6x14x14 → 16x10x10
- C3: 16x5x5 → 120x1x1

2 Pooling layer
- S1: 28x28 → 14x14
- S2: 10x10 → 5x5

2 Fully connected layer
- F1: 120 → 84
- F2: 84 → 10

3 Convolution layer
- C1: nn.Conv2d (1, 6, kernel_size = 5, stride = 1)
- C2: nn.Conv2d (6, 16, kernel_size = 5, stride = 1)
- C3: nn.Conv2d (16, 120, kernel_size = 5, stride = 1)

(1) 모델 정의



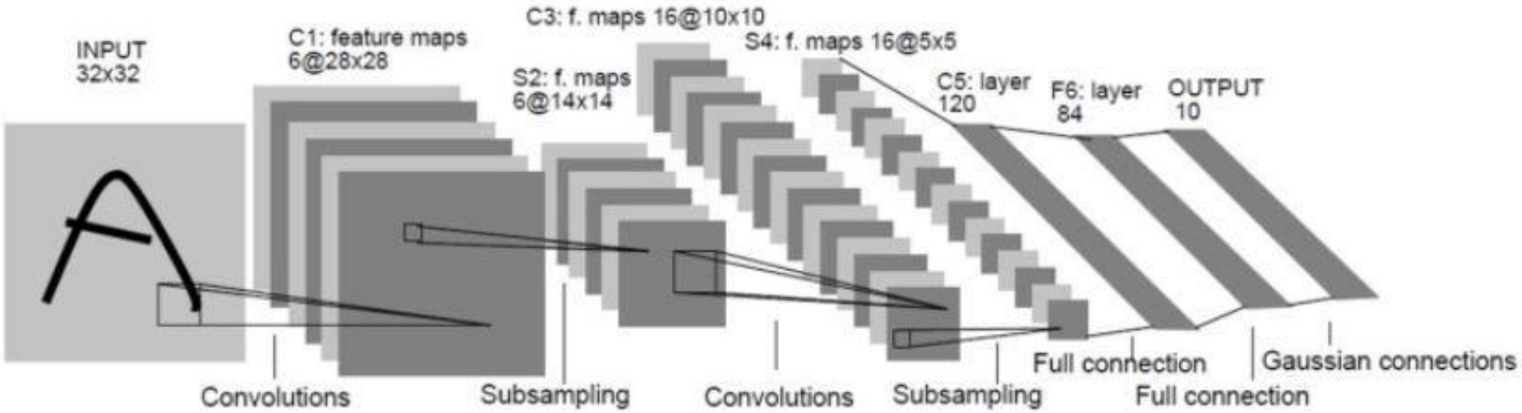
3 Convolution layer
- C1: 1x32x32 → 6x28x28
- C2: 6x14x14 → 16x10x10
- C3: 16x5x5 → 120x1x1

2 Pooling layer
- S1: 28x28 → 14x14
- S2: 10x10 → 5x5

2 Fully connected layer
- F1: 120 → 84
- F2: 84 → 10

2 Pooling layer
- S1: F.avg_pool2d (x, 2, 2)
- S2: F.avg_pool2d (x, 2, 2)

(1) 모델 정의



3 Convolution layer
C1: 1x32x32 → 6x28x28
C2: 6x14x14 → 16x10x10
C3: 16x5x5 → 120x1x1

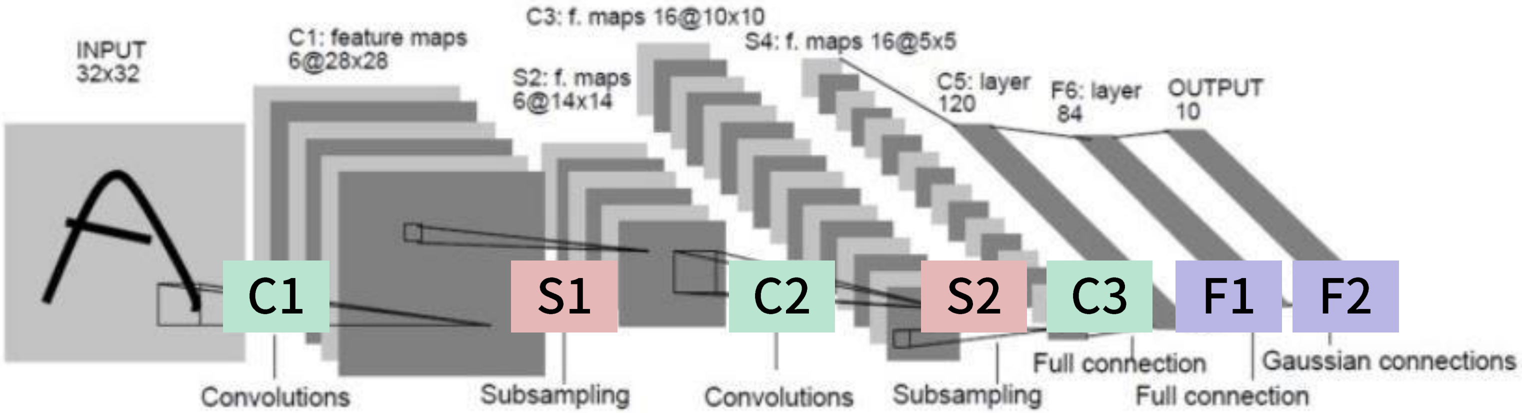
2 Pooling layer
S1: 28x28 → 14x14
S2: 10x10 → 5x5

2 Fully connected layer
F1: 120 → 84
F2: 84 → 10

2 Pooling layer
S1: F.avg_pool2d (x, 2, 2)
S2: F.avg_pool2d (x, 2, 2)

2 Fully connected layer
F1: nn.Linear(120, 84)
F2: nn.Linear(84, 10)

(1) 모델 정의

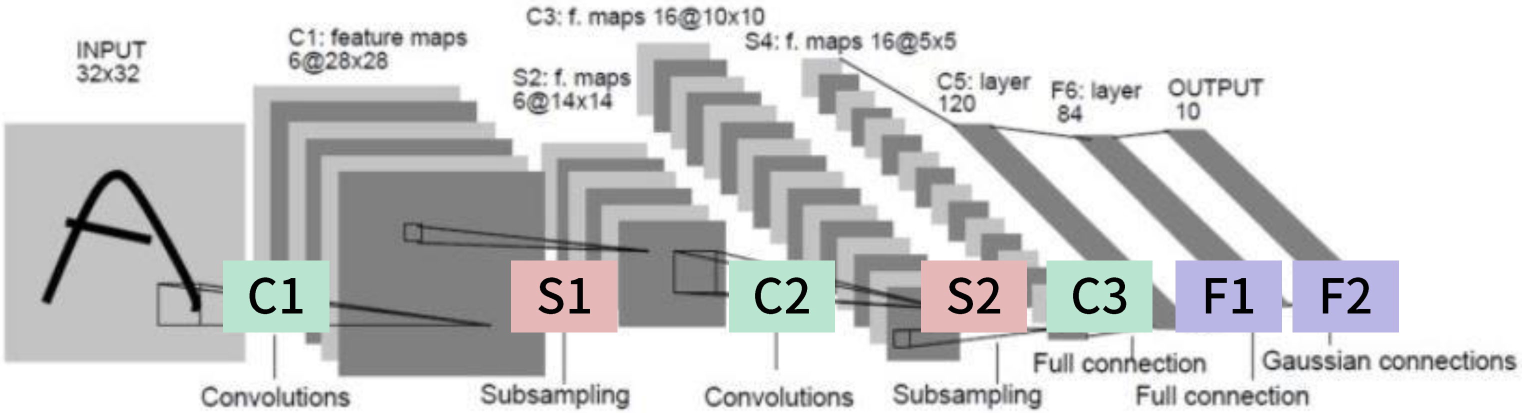


C1: 1x32x32 → 6x28x28	C1: nn.Conv2d (1, 6, kernel_size = 5, stride = 1)
S1: 28x28 → 14x14	S1: F.avg_pool2d (x, 2, 2)
C2: 6x14x14 → 16x10x10	C2: nn.Conv2d (6, 16, kernel_size = 5, stride = 1)
S2: 10x10 → 5x5	S2: F.avg_pool2d (x, 2, 2)
C3: 16x5x5 → 120x1x1	C3: nn.Conv2d (16, 120, kernel_size = 5, stride = 1)
F1: 120 → 84	F1: nn.Linear(120, 84)
F2: 84 → 10	F2: nn.Linear(84, 10)

(1) 모델 정의

```
from torch import nn
import torch.nn.functional as F

class LeNet5 ( nn.Module ):
    def __init__ (self):
        super(LeNet5.self).__init__()
        self.conv1 = nn.Conv2d (1, 6, kernel_size = 5, stride = 1)
        self.conv2 = nn.Conv2d (6, 16, kernel_size = 5, stride = 1)
        self.conv3 = nn.Conv2d (16, 120, kernel_size = 5, stride = 1)
        self.fc1 = nn.Linear (120, 84)
        self.fc2 = nn.Linear (84, 10)
    def forward(self, x):
        x = F.tanh (self.conv1(x))
        x = F.avg_pool2d (x, 2, 2)
        x = F.tanh (self.conv2(x))
        x = F.avg_pool2d (x, 2, 2)
        x = F.tanh (self.conv3(x))
        x = x.view(-1, 120)
        x = F.tanh (self.fc1(x))
        x = self.fc2(x)
        return F.softmax(x, dim=1)
```

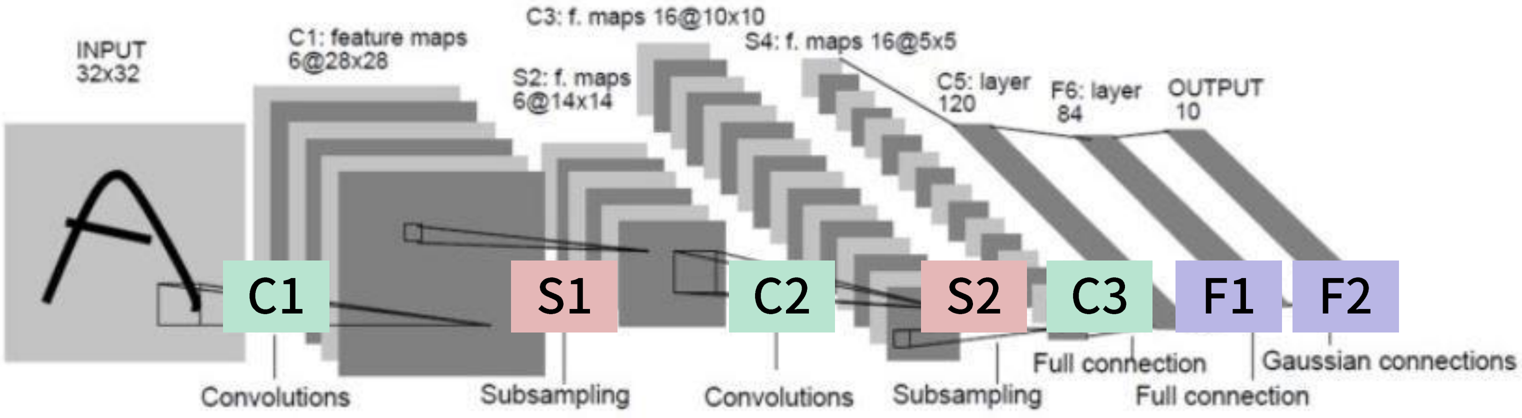
(1) 또 다른 모델 정의 (nn.Sequential을 사용)



feature extractor
- C1: nn.Conv2d (1, 6, kernel_size = 5, stride = 1)
- S1: F.avg_pool2d (x, 2, 2)
- C2: nn.Conv2d (6, 16, kernel_size = 5, stride = 1)
- S2: F.avg_pool2d (x, 2, 2)
- C3: nn.Conv2d (16, 120, kernel_size = 5, stride = 1)

classifier
- F1: nn.Linear(120, 84)
- F2: nn.Linear(84, 10)

(1) 또 다른 모델 정의 (nn.Sequential을 사용)



feature extractor

```
self.feature_extractor = nn.Sequential (
    nn.Conv2d (1, 6, kernel_size = 5, stride = 1),
    nn.Tanh ( ),
    nn.AvgPool2d(kernel_size = 2),
    nn.Conv2d (6, 16, kernel_size = 5, stride = 1),
    nn.Tanh ( ),
    nn.AvgPool2d(kernel_size = 2),
    nn.Conv2d (16, 120, kernel_size = 5, stride = 1),
    nn.Tanh ( )
)
```

classifier

```
self.classifier = nn.Sequential (
    nn.Linear(120, 84),
    nn.Tanh( ),
    nn.Linear(84, 10)
)
```

(1) 또 다른 모델 정의 (nn.Sequential을 사용)

```
from torch import nn
import torch.nn.functional as F

class LeNet5 ( nn.Module ):
    def __init__ (self):
        super(LeNet5.self).__init__()

    self.feature_extractor = nn.Sequential (
        nn.Conv2d (1, 6, kernel_size = 5, stride = 1),
        nn.Tanh ( ),
        nn.AvgPool2d(kernel_size = 2),
        nn.Conv2d (6, 16, kernel_size = 5, stride = 1),
        nn.Tanh ( ),
        nn.AvgPool2d(kernel_size = 2),
        nn.Conv2d (16, 120, kernel_size = 5, stride = 1),
        nn.Tanh ( )
    )

    self.classifier = nn.Sequential (
        nn.Linear(120, 84),
        nn.Tanh( ),
        nn.Linear(84, 10)
    )
```

(1) 또 다른 모델 정의 (nn.Sequential을 사용)

```
def forward (self, x):
        x = self.feature_extractor (x)
        x = torch.flatten(x, 1)
        x = self.classifier (x)
        return F.softmax(x, dim-1)
```

(2) 데이터 로딩

- 많이 이용하는 MNIST 데이터셋 이용
- Pytorch에서 지원하는 Torchvision에서 제공
- Torchvision에서 제공하는 함수를 이용할 경우, 실시간으로 다운로드 받아서 사용할 수 있음

(2) 데이터 로딩

- 데이터를 정해진 크기 (32x32)의 Tensor 포맷으로 변경하여 사용

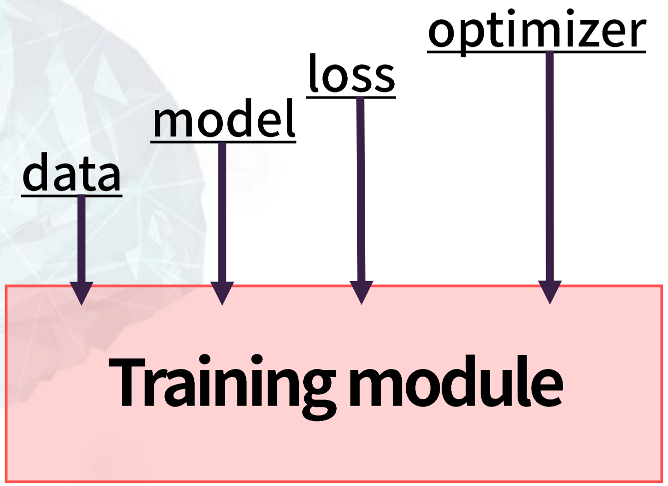
```
from torchvision import transforms

data_transform = transforms.Compose([
            transforms.Resize((32, 32)),
            transforms.ToTensor(),
])
```

(2) 데이터 로딩

```
from torchvision import transforms
from torchvision import datasets

data_transform = transforms.Compose([
            transforms.Resize((32, 32)),
            transforms.ToTensor(),
])

datapath = '/data/MNIST'

train_data = datasets.MNIST(datapath, train=True,
             download=True, transform=data_transform)
```

(3) Training

- 환경 설정
- Data
- Model
- Loss
- Optimizer
- Training



(3) Training

- 환경 설정

```
RANDOM_SEED = 42
LEARNING_RATE = 1e-4
BATCH_SIZE = 32
EPOCH = 20
IMAGE_SIZE = 32
```

- Data

```
train_loader = DataLoader (dataset=train_data,
             batch_size = BATCH_SIZE, shuffle = True)
```

- Model

```
model = LeNet5().to(DEVICE)
```

- Loss

```
criterion = nn.CrossEntropyLoss ( )
```

- Optimizer

```
optimizer = torch.optim.Adam(model.parameters(),
            lr = LEARNING_RATE)
```

- Training

(3) Training

- Training

```
for epoch in range(0, EPOCH):
    for x, y in train_loader:
        optimizer.zero_grad ( )

        y = y.to (device)
        y_pred = model (x)
        loss = criterion (y_pred, y)

        loss.backward ( )

        optimizer.step ( )
        optimizer.zero_grad ( )
```

```
for t in range (500):
    y_pred = model (x)
    loss = torch.nn.functional.mse_loss (y_pred, y)

    loss.backward ( )

    optimizer.step()
    optimizer.zero_grad
```

(3) Training

- Training

```
RANDOM_SEED = 42
LEARNING_RATE = 1e-4
BATCH_SIZE = 32
EPOCH = 20
IMAGE_SIZE = 32

train_loader = DataLoader (dataset=train_data,
             batch_size = BATCH_SIZE, shuffle = True)
model = LeNet5().to(DEVICE)
criterion = nn.CrossEntropyLoss ( )
optimizer = torch.optim.Adam(model.parameters(),
            lr = LEARNING_RATE)

for epoch in range(0, EPOCH):
    for x, y in train_loader:
        optimizer.zero_grad ( )

        y = y.to (device)
        y_pred = model (x)
        loss = criterion (y_pred, y)

        loss.backward ( )

        optimizer.step ( )
        optimizer.zero_grad ( )
```