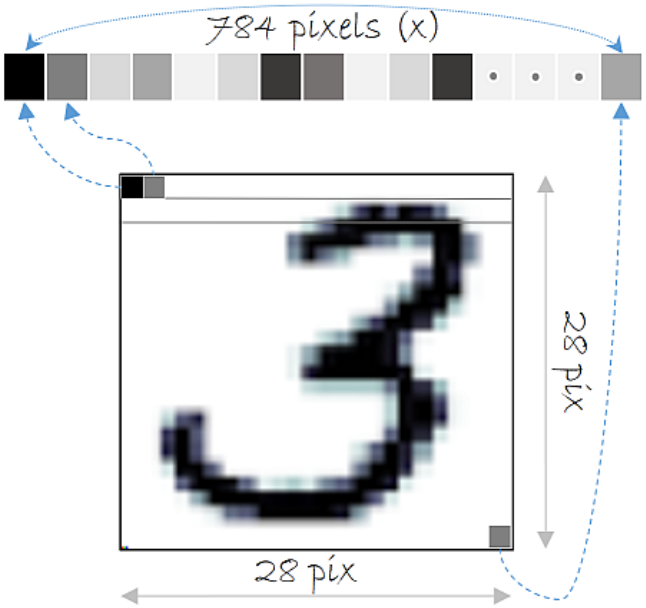
인공지능 1주차-2 CNN

완전 연결 계층

: 한 층(layer)의 모든 뉴런이 그 다음 층의 모든 뉴런과 연결된 상태를 의미한다.

1차원 배열의 형태로 평탄화된 행렬을 통해 이미지를 분류하는데 사용되는 계층이다.



문제점

입력 데이터가 이미지인 경우, 이미지는 세로, 가로, 채널(색상) 3차원 데이터이다. 완전 연결 계층에 입력할 때는 3차원 데이터를 평평한 1차원 데이터로 평탄화 해 줘야 한다.

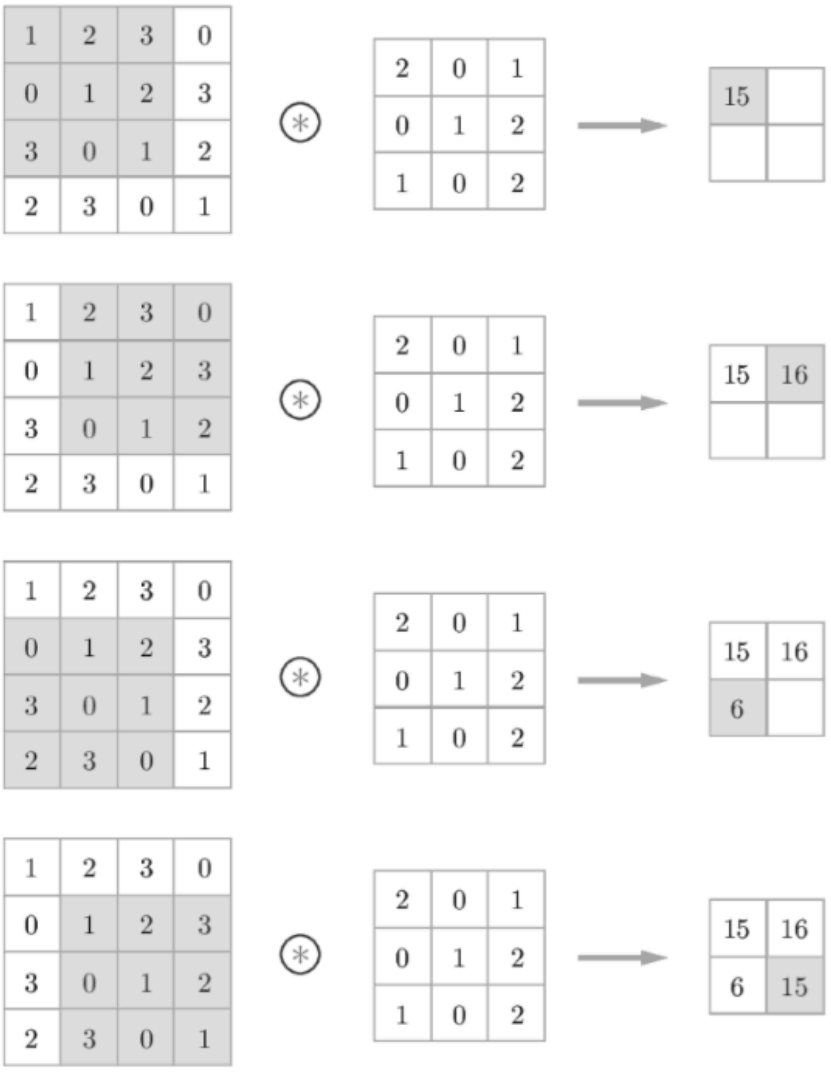
이미지는 3차원 형상에 공간적 정보가 담겨 있는데, 완전 연결 계층은 형상을 무시하고 모든 입력 데이터를 동등한 뉴런으로 취급하여 형상에 담긴 정보를 살릴 수 없다.

합성곱

: 입력 데이터에 필터를 일정한 간격으로 이동시키며, 대응되는 값끼리 곱하고 모두 더하는 연산을 수행한다.

합성곱 연산

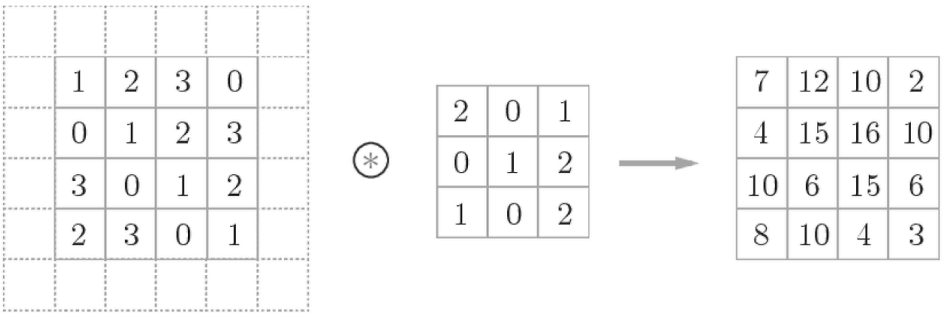
- 합성곱 계층에서는 합성곱 연산을 처리한다. 합성곱 연산은 이미지 처리에서 말하는 필터 연산에 해당한다.
- 필터는 커널(Kenel)이라 부르기도 하며, 가중치에 해당한다.



- 필터에 대응하는 원소와 필터의 원소끼리 곱하여 총합을 계산하는 방식을 단일 곱셈-누산이라고 한다.

패딩

: 데이터 주위에 특정 width의 특정 값을 채워 출력의 크기를 조절하는 방식이다.



패딩을 사용하는 이유

→ 합성곱 연산을 몇번이나 되풀이 하는 심층 신경망에서는 입력 데이터의 크기가 작아지기 때문에 어느 시점에는 출력 크기가 1이 되어 더이상 합성곱 연산을 적용할 수 없게 된다. 이를 방지하기 위해 패딩을 사용하여 출력 크기를 조정한다.

종류

1. **밸리드 패딩(valid padding)**

: 패딩을 사용하지 않음

2. Same padding

: 출력 크기가 입력과 같도록 입력 데이터 주변에 적절한 크기의 패딩을 추가한다.

- 보통 0으로 채워진 픽셀을 사용한다.

3. Full padding

: 입력 데이터의 모든 원소가 동일한 횟수만큼 연산에 참여하도록 패딩을 추가한다.

- 출력 크기가 입력 크기보다 커진다.

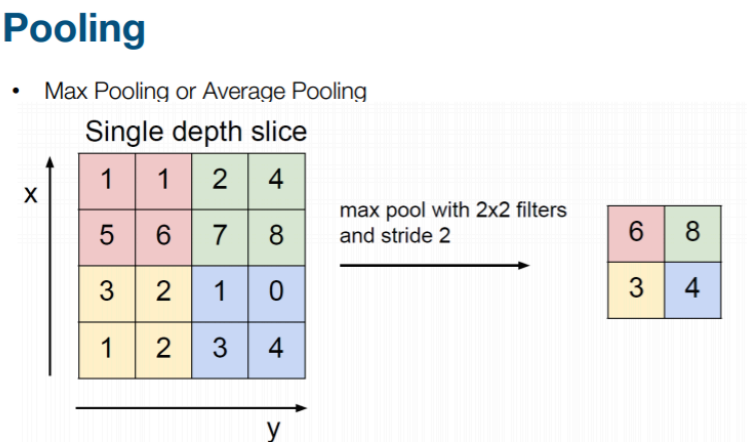
4. Reflection padding

: 입력의 가장자리 값을 반사하여 채운다.

- 가장자리 왜곡 감소, 이미지 처리에서 자주 사용한다.

Pooling layer

: sub sampling은 해당하는 이미지 데이터를 작은 사이즈의 이미지로 줄이는 과정이다.



종류

1. Max Pooling : 정해진 크기 안에서 가장 큰 값만 뽑아낸다.
2. Average Pooling : 정해진 크기 안의 값들의 평균을 뽑아낸다.

사용하는 이유

1. input size를 줄임

: 텐서의 크기를 줄이는 역할을 한다.

2. overfitting을 조절

: input size가 줄어드는 것은 그만큼 필요없는 파라미터의 수가 줄어들어 훈련 데이터에만 높은 성능을 보이는 과적합을 줄일 수 있다.

3. 특징을 잘 뽑음

: pooling을 했을 때, 특정한 모양을 더 잘 인식할 수 있음

4. 지역적 이동에 노이즈를 줌으로써 일반화 성능을 올려준다.

특징

1. training을 통해 훈련되어야 할 파라미터가 없다.
2. Pooling의 결과는 채널 수에는 영향이 없으므로 채널 수는 유지된다.
3. 입력 데이터에 변화가 있어도 pooling 결과의 변화는 적다.

장점

: 맵 크기를 줄임으로써 중요한 데이터만 추출하므로 노이즈를 제거하는 효과가 있다.

단점

: 전체 정보를 반영하지 못하므로 추출되지 않은 정보가 중요한 정보인 경우 중요한 정보를 소실하여 성능을 저하시킬 수 있다.

conv1d

- 시간에 따른 변화 → 시계열, 오디오, 단어 임베딩 시퀀스 등에 사용
- 한 축만 있는 데이터

<입력 데이터의 차원>

- 3차원
- (batch_size, in_channels, width)

batch_size	한 번에 모델에 넣는 샘플 수
in_channels	한 데이터 안에서의 입력 채널 수
width	각 채널의 데이터 길이

<kernel>

- (out_channels, in_channels, kernel_size)
- 1차원 배열로 입력 채널마다 존재하며, 출력 채널 수만큼 반복된다.

out_channels	사용할 필터 개수

in_channels	입력 채널 수
kernel_size	한 채널에서 슬라이딩하는 필터의 길이

conv2d

- 이미지
- 두 축이 있는 데이터

<입력 데이터의 차원>

- 4차원
- (batch_size, in_channels, height, width)

batch_size	한 번에 넣는 이미지 개수
in_channels	이미지 채널 수 (흑백 = 1, RGB=3)
height	이미지 높이
width	이미지 너비

<kernel>

- (out_channels, in_channels, kernel_size)
- 1차원 배열로 입력 채널마다 존재하며, 출력 채널 수만큼 반복된다.

out_channels	사용할 필터 개수
in_channels	입력 채널 수
kernel_height	필터의 세로 크기
kernel_width	필터의 가로 크기

keras에서 MLP 구현하는법

- **Sequential API**

  : 레이어 블록을 순차적으로 쌓아가는 방식으로 아키텍처를 모델링하는 기법이다.

1. keras 모듈의 models 패키지에서 **.Sequential()** 클래스를 불러와 model 인스턴스를 생성한다.
2. model 인스턴스의 **.add()** 메소드를 이용해 층을 추가하는데 입력층은 flatten 클래스로 shape를 일렬로 변경한다.
3. model.add() 메소드를 이용해 완전 연결된(Dense) 층을 생성한다.

- **Functional API**

  : Sequential API보다 더 유연한 적용이 가능하다. 어느 층을 함수의 입력값으로 하여 결과값을 출력하는 방식으로 구현한다.

1. keras 모듈의 layers 패키지에서 **.Input()** 클래스를 불러와 input층의 인스턴스를 생성한다. 입력 층의 노드 형태는 shape 파라미터에 명시한다.
2. layers 패키지에서 concatenate() 클래스를 불러와 입력되는 신호를 결합할 수 있다. → 단일 흐름만 만드는 Sequential API에서는 사용하지 않음
3. 최종적으로 Model을 생성한다. 설계한 입력층과 출력층을 인수로 넘겨준다.

- **Subclassing API**

  : 가장 낮은 수준의 방법으로, keras.Model 클래스를 상속받아 모델을 직접 정의한다. 동적 그래프, 순환 연결, 층 형태 변환 등을 구현할 때 유용하다.

1. keras.Model 클래스를 상속받는다.
2. **init** 메소드를 정의한다. 모델의 구성 요소를 초기화하고 이때, 필요한 레이어들을 생성하고 변수로 저장한다.
3. call 메소드를 정의한다. 입력 데이터를 받아 레이어를 통과시키고 최종 출력을 반환한다.