합성곱

합성곱 : 마치 입력 데이터에 마법의 도장을 찍어서 유용한 특성만 드러나게 하는 것으로 비유 가능

합성곱의 동작 원리

7장에서 사용한 밀집층에서는 뉴런마다 입력 개수만큼의 가중치가 있다.
즉 모든 입력에 가중치를 곱한다.

인공 신경망은 처음에 **w1 ~ w10**과 절편 **b**를 랜덤하게 초기화한 다음(입력이 **10**개까지여서 가중치가 **w10**까지 있다고 가정)
에포크를 반복하면서 경사 하강법 알고리즘을 사용하여
손실이 낮아지도록 최적의 가중치와 절편을 찾는다.
= 모델 훈련

예) **7**장의 예 **:** 패션 **MNIST** 이미지에 있는 **784**개의 픽셀을 입력받는 은닉층의 뉴런 개수가 **100**개면
뉴런마다 하나씩 출력도 **100**개가 된다.

합성곱은 밀집층의 계산과 조금 다르다. 입력 데이터 전체에 가중치를 적용하는 것이 아니라 일부에 가중치를 곱한다.
데이터 예시 **:** `[3, 1, 0, 7, 6, 4, 8, 2, 4, 5]`

처음 **3**개부터 마지막 세개 까지의 앞 뉴런을 묶어 가중치를 곱해 가면서 다음 뉴런에 전달한다.

```
3*w1 + 1*w2 + 0*w3 + b
1*w1 + 0*w2 + 7*w3 + b
,,,
2*w1 + 4*w2 + 5*w3 + b
```

이렇게 한 칸씩 아래로 이동하면서 출력을 만드는 것이 합성곱이다.
여기에서는 뉴런의 가중치가 **3**개이기 때문에 모두 **8**개의 합성곱이 만들어진다.

모두 같은 가중치 **w1 ~ w3**을 사용하고, 같은 **b**를 사용한다.
합성곱 층의 사용할 뉴런 개수 및 편향 역시 하이퍼파라미터이다.

이전에 그렸던 신경망 층의 그림은 뉴런이 길게 늘어져 있고, 서로 조밀하게 연결되었다.
그런데 합성곱에서는 뉴런이 입력 위를 이동하면서 출력을 만들기 때문에 이런 식으로 표현하기가 어렵다. 또 뉴런이라고 부르기도 어색하다.

합성곱 신경망(Convolutional Neural Network, CNN) 에서는 완전 연결 신경망과 달리
뉴런을 *필터* 혹은 *커널*이라 부른다.

완전 연결 신경망은 **7**장에서 만들었던 신경망이다. 완전 연결 층**(**밀집층**)**만을 사용하여 만든 신경망에 해당

뉴런 개수를 의미할 때는 필터라 하고, 가중치를 의미할 때는 커널이라 설명 , 사실 같은 것을 가리키는 용어이다.

입력이 **2**차원 배열이면 필터도 **2**차원이여야 한다. **4** ***4** 크기의 그림을 학습할 때*
*커널 크기를 **3*** **3**으로 가정해 보자. 입력의 **9**개 원소와 커널의 **9**개 가중치를 곱한 후
1개의 출력을 만든다.

그 다음에는 필터가 오른쪽으로 한 칸 이동하여 합성곱을 또 수행한다.
입력의 너비가 **4**이므로 더 이상 오른쪽으로 한 칸 이동할 수 없다.
이럴 때에는 아래로 한 칸 이동한 다음 다시 왼쪽에서부터 합성곱을 수행한다.
그리고 다시 오른쪽으로 한 칸 이동한다.

마치 도장을 찍듯이 왼쪽 위에서 오른쪽 아래까지 이동하면서 출력을 만든다.

위의 예시에서 필터는 모두 4번 이동할 수 있기 때문에 4개(2 *2) 2) 크기의 출력을 만든다.*
*4 4 크기의 입력을 2 * 2로 압축한 느낌이 난다.*
합성곱 계산을 통해 얻은 출력을 특별히 **특징 맵(feature map)** 이라 부른다.

밀집층에서 여러 개의 뉴런을 사용하듯이 합성곱 층에서도 여러 개의 필터를 사용한다.
여러 개의 필터를 사용하여 만들어진 특성 맵은 순서대로 차곡차곡 쌓인다.

2 * 2 크기의 서로 다른 필터 **3**개를 사용하면 출력 층은
(2, 2, 3) 크기의 **3**차원 배열이 된다.

2차원 구조를 그대로 사용하기 때문에 합성곱 신경망이 이미지 처리 분야에서 뛰어난 성능을 발휘한다.
그럼 **keras**에서 합성곱 층을 어떻게 만드는 지 알아보자.

케라스 합성곱 층

케라스의 층은 모두 **keras.layers** 패키지 아래 클래스로 구현되어 있다. 합성곱 층도 마찬가지이다.

닙력 위를 **(**왼쪽 **->** 오른쪽, 위쪽 **->** 아래쪽**)**으로 이동하는 합성곱은 ***Conv2D*** 클래스로 제공한다.
``` **from tensorflow import keras keras.layers.Conv2D(10, kernel_size = (3, 3), actiavtion = 'relu')**

- 매개변수 **1 :** 도장의 개수**(** 필터의 개수 **)**
- 매개변수 **2( kernel_size ) :** 필터에 사용할 커널의 크기
- 필터의 개수와 커널의 크기는 반드시 지정해야 하는 매개변수이다.
- 마지막으로 밀집층에서처럼 활성화 함수를 지정한다. 여기에서는 **relu** 함수를 선택했다.

특성 맵은 활성화 함수를 적용한 후이다.
합성곱 신경망에서도 종종 활성화 함수를 언급하지 않는다.
일반적으로 특성 맵은 활성화 함수를 통과한 값을 나타낸다. 합성곱에서는 활성화 출력이란 표현을 잘 쓰지 않는다.

케라스 **API**를 사용하면 합성곱 층을 사용하는 것이 어렵지 않다. 이전에 **Dense** 층을 사용했던 자리에
대신 **Conv2D** 층을 넣으면 된다. 다만 **kernel_size**와 같이 추가적인 매개변수를 고려해야 한다.

## 합성곱 신경망의 정의 :

- 일반적으로 **1**개 이상의 합성곱 층을 쓴 인공 신경망
- 꼭 합성곱 층만 사용할 필요는 없다.
    - 클래스에 대한 확률을 계산하려면 마지막 층에 클래스 개수만큼의 뉴런을 가진 밀집층을 두는 것이 일반적

합성곱 신경망을 만들려면 조금 더 알아야 할 것이 있다.

## 패딩과 스트라이드

앞에서 예로 두었던 합성곱 계산은 **(4, 4)** 크기의 입력에 **(3, 3)** 크기의 커널을 이용하여 **(2, 2)** 크기의 특성 맵을 만들었었다.
그런데 만약 커널 크기는 **(3, 3)**으로 그대로 두고 출력의 크기를 입력과 동일하게
**(4, 4)**로 만들려면 어떻게 해야 할까 **?**

**(4, 4)** 입력과 동일한 크기의 출력을 만들려면 마치 더 큰 입력에 합성곱하는 척해야 한다.
입력 배열의 주위를 가상의 원소로 채우면 된다.
보통 끝부분의 원소들은 필터의 중앙에 오지 못하고 필터 주변부의 가중치와만 합성곱 하게 된다..
```

중앙부와 모서리 픽셀이 합성곱에 참여하는 비율이 크게 차이날 수밖에 없다.

만약 이미지의 대상이 이미지의 끝부분에 있다면, 해당 대상을 탐색하지 못할 가능성도 있다.
따라서 원본 이미지 주변에 가상의 원소로 채워 가중치에는 의미가 없지만, 끝부분의 이미지로부터 원하는 물체를 인식하게끔 할 수 있다.

이렇게 입력 배열의 주위를 가상의 원소로 채우는 것을 ***패딩***이라 한다.
실제 입력값이 아니기 때문에 **padding**은 **0**으로 채운다.
(4, 4) 크기의 입력에 **0**을 **1**줄 **padding** 하면, **(6, 6)** 크기의 이미지가 된다.

실제 값은 **0**으로 채워져 있기 때문에 계산에 영향을 미치지는 않는다.

입력과 특성 맵의 크기를 동일하게 만들기 위해 입력 주위에 **0**으로 패딩하는 것을 **세임 패딩**이라 한다.
합성곱 신경망에서는 세임 패딩이 가장 많이 사용된다.

패딩 없이 순수나 입력 배열에서만 합성곱을 하여 특성 맵을 만드는 경우를 ***밸리드 패딩***이라 한다.
밸리드 패딩은 특성 맵의 크기가 줄어들 수밖에 없다.

적절한 패딩은 이미지의 주변에 있는 정보를 잃어버리지 않도록 한다.

케라스 **Conv2D** 클래스에서는 **padding** 매개변수로 패딩을 지정할 수 있다.
기본값은 **'valid'**로 밸리드 패딩을 나타낸다.
세임 패딩을 사용하려면 **'same'**으로 지정한다.

```
keras.layers.Conv2D(10, kernel_size = (3, 3), activation = 'relu', padding = 'same')
```

지금까지 본 합성곱 연산은 좌우, 위아래 한 칸씩만 이동했다. 그러나 두 칸, 그 이상씩 건너뛸 수도 있다.
여러 칸씩 이동하면 특성 맵의 크기는 더 작아질 것이다.

이런 이동의 크기를 ***스트라이드***라 한다. 기본으로 스트라이드 = **1**이다. 즉 한 칸씩 이동한다.
이 값이 케라스의 **Conv2D**의 **strides** 매개변수의 기본값이다.

```
keras.layers.Conv2D(10, kernel_size = (3, 3), activation = 'relu', padding = 'same', strides = 1)
```

strides 매개변수는 오른쪽으로 이동하는 크기와 아래쪽으로 이동하는 크기를 **(1, 1)**과 같이 튜플을 사용해 각각 지정할 수 있다.

하지만 커널의 이동 크기를 가로세로 방향으로 다르게 지정하는 경우는 거의 없다.
대부분 기본값을 그대로 사용하기 때문에 **strides** 변수는 잘 사용하지 않는다.

풀링

- 합성곱 층에서 만든 특성 맵의 가로세로 크기를 줄이는 역할

하지만 특성 맵의 개수는 줄이지 않는다. 예를 들면 **(2, 2, 3)** 크기의 특성 맵에
풀링을 적용하면 마지막 차원의 개수는 그대로 유지하고 너비와 높이만 줄어들어
(1, 1, 3) 크기의 특성 맵이 된다.

풀링도 합성곱처럼 입력 위를 지나가면서 도장을 찍는다. 위 그림에서는 **(2, 2)** 크기로 풀링을 한다.
하지만 풀링에는 가중치가 없다. 도장을 찍은 영역에서 가장 큰 값을 고르거나 평균값을 계산한다.

가장 큰 값을 고르는 방식을 **최대 풀링**,
평균값을 고르는 방식을 **평균 풀링**이라 한다.

풀링은 합성곱 층과 뚜렷이 구분되기 때문에 풀링 층이라고 부르겠다.
가령 다음과 같은 **(4, 4)**크기의 특성 맵이 있다고 가정해 보겠다.

여기에 **(2, 2)**풀링을 적용하면, **(2, 2)** 크기 단위로 쪼개서 각 영역을 하나의 값으로 변환한다.
따라서 **(4, 4)** 크기에 **(2, 2)**크기의 풀링을 적용하면 절반으로 크기가 줄어든다.

특성 맵이 여러 개라면 동일한 작업을 반복한다. 즉 **10**개의 특성 맵이 있다면 풀링을 거친 맵도 **10**개가 된다.

풀링에서는 겹치지 않고 이동한다. 한 영역을 하나의 수로 치환하는 것이기 때문에,
그 옆에 새로운 풀링을 붙여서 또 하나의 값으로 줄이는 역할을 한다.

풀링은 가중치가 없고, 풀링 크기와 스트라이드가 같기 때문에 이해하기 쉽다.
또 패딩도 없다.
케라스에서는 **MaxPooling2D** 클래스로 풀링을 수행할 수 있다.

```
keras.layers.MaxPooling2D(2)
```

평균을 계산하는 풀링 클래스는
`keras.layers.AveragePooling2D(2)`

그러나 평균 풀링은 거의 사용하지 않는다. 특정 부분에서의 도드라지는 특성을
다른 부분과 평균을 냄으로써 희석시키기 때문이다. 대부분 **Max Pooling**을 사용한다.

- **1**번째 매개변수 : 풀링의 크기**(** 대부분 풀링의 크기 **= 2 :** 가로세로를 절반으로 줄인다**.)**
- 가로세로 방향의 풀링 크기를 다르게 하려면 첫 번째 매개변수를 정수의 튜플로 지정 **-** 예 **: (2, 3)**
- 그러나 이런 경우는 극히 드물다.

합성곱 층과 마찬가지로 **strides**와 **padding** 매개변수를 제공한다.
그러나 **strides, padding** 변수는 거의 사용하지 않는다.