

Convolution Layer



4개의 이미지를 살펴보면 우리는 각 이미지의 동그라미가 있다는 특징을 발견할 수 있습니다. 동그라미의 특징이 어디서 몇 개 발견되는지 표로 나타내면 우측에 표와 같습니다. 9라는 숫자는 전체의 동그라미가 1개, 이미지를 절반으로 나누어서 위쪽에 동그라미 1개, 아래쪽에는 동그라미가 없습니다. 8이라는 숫자는 전체의 동그라미 2개, 절반으로 나누어서 위쪽에 동그라미 1개, 아래쪽에 동그라미 1개가 있습니다. 6과 0에 대해서도 동일한 방법으로 특징을 확인해 값을 채워 놓았습니다. 이미지상의 어떤 특징이 어느 부분에서 나타나는 지를 알 수 있으면 그 이미지를 분류하는 데 매우 유용합니다. 이미지에서 어떤 특징이 어느 위치에서 발견되는 지를 나타낸 정보는 매우 유용한 정보여서 일을 찾기 위한 목적으로 사용하는 도구가 바로 컨벌루션(Convolution)입니다.

Convolution은 한국어로 합성곱이라 합니다.

※ 합성곱(Convolution) ?

이미지의 형상을 무시하지 않고 이미지를 그대로 인공 신경망이 학습할 수 있게 해주는 수학적 행렬 연산입니다.

- 합성곱에서 원본 이미지는 학습해야 할 사진 데이터(행렬로 변환)
- 필터(filter)는 원본 이미지에서 특징을 잡아내는데 사용되는 행렬입니다.
- 특징을 잡아서 feature map을 생성해 원본 이미지의 형태를 이해하는 것입니다.

즉, 이미지의 특징을 추출하는 것으로 생각하시면 됩니다.

행렬곱 연산입니다.

$$\begin{array}{cccc} 1 & 2 & 3 & 0 \\ 0 & 1 & 2 & 3 \\ 3 & 0 & 1 & 2 \\ 2 & 3 & 0 & 1 \end{array} \odot \begin{array}{ccc} 2 & 0 & 1 \\ 0 & 1 & 2 \\ 1 & 0 & 2 \end{array} = \begin{array}{cc} 15 & 16 \\ 6 & 15 \end{array}$$

입력 데이터 (4,4) 필터 (3,3) 결과 (2,2)

위의 예시에서 크게 4번 이루어집니다.

$$\begin{array}{cccc} 1 & 2 & 3 & 0 \\ 0 & 1 & 2 & 3 \\ 3 & 0 & 1 & 2 \\ 2 & 3 & 0 & 1 \end{array} \odot \begin{array}{ccc} 2 & 0 & 1 \\ 0 & 1 & 2 \\ 1 & 0 & 2 \end{array} = \begin{array}{cc} 15 & 16 \\ 6 & 15 \end{array}$$

입력 데이터 (4,4) 필터 (3,3) 결과 (2,2)

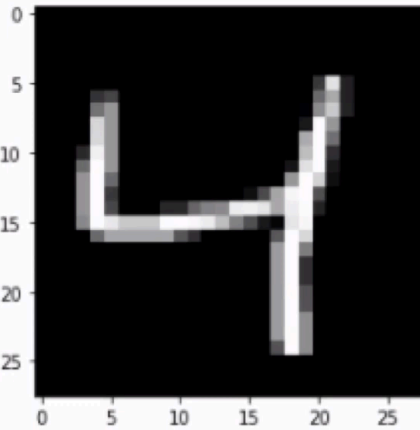
색깔 별 구역의 (3x3) 행렬과 필터가 각각 연산되어 15 , 16 , 6 , 15 라는 결과로 (2x2) 행렬이 만들어집니다. 4x4 의 행렬이 2x2 행렬로 크기가 작아지고, 4x4 행렬의 특징을 가지고 있습니다. 이는 나중에 CNN 학습 시 속도 개선에 도움이 되는 부분입니다.

※ 컨볼루션 연산이 계속되면 데이터가 작아지면서 정보가 유실 될 수 있습니다. 빈 가장자리를 채우는 것을 padding이라고 합니다.

※ 필터의 이동 간격은 stride라고 합니다.

이연산은 이미지 처리와 신호처리 분야에서 매우 유명한 도구입니다. 방금 설명했듯이 이미지 처리에서는 특정 패턴의 특징이 어느 부분에 등장하는지 파악하는데 사용 됩니다.

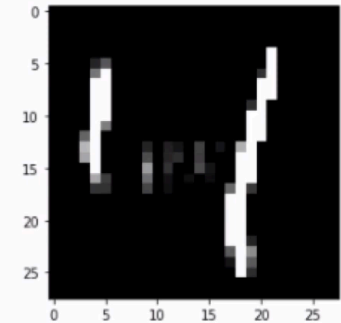
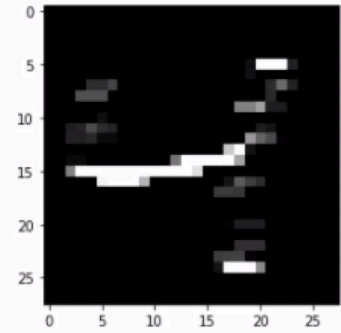
특정한 패턴의 특징이
어디서 나타나는지를 확인하는 도구
“Convolution”



필터



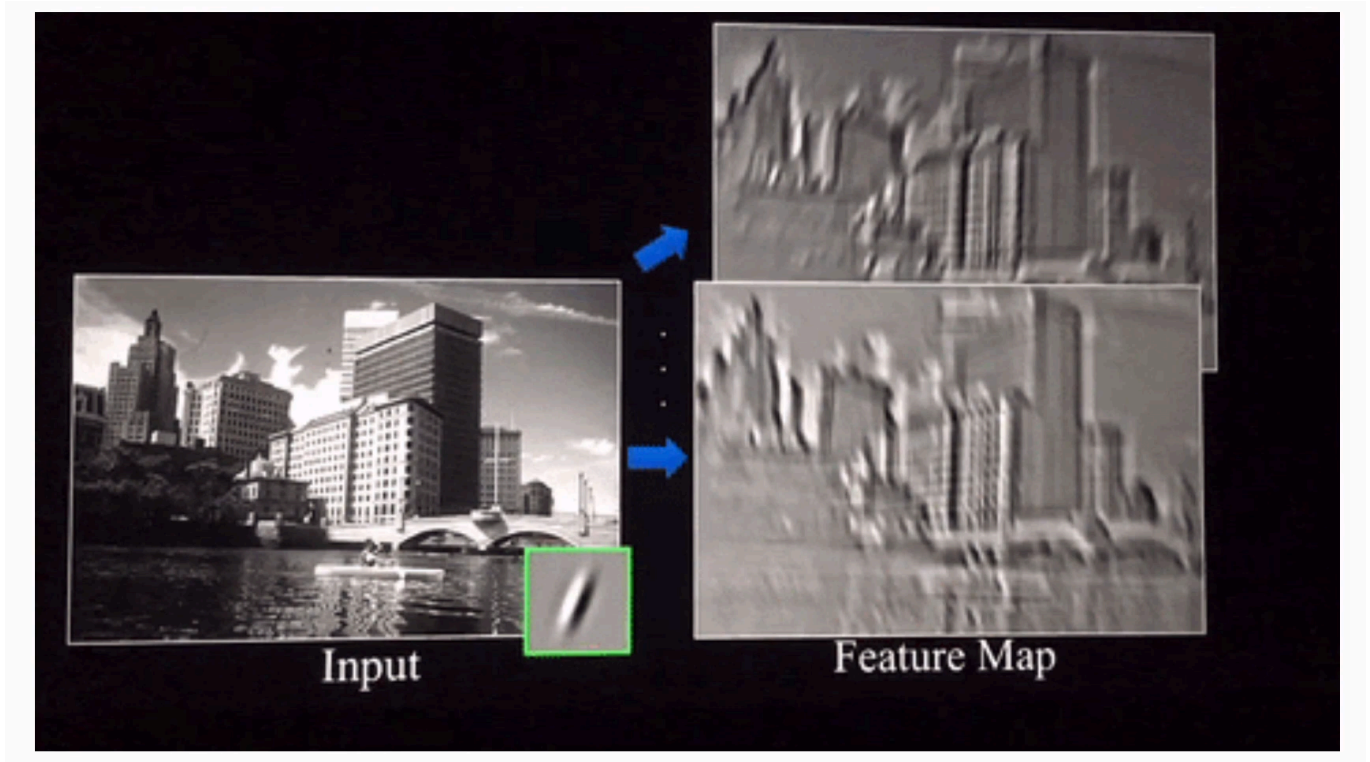
특징맵 feature map



이그림 위에 4라는 손글씨 이미지가 있습니다. 이 이미지가 어떻게 생겼는지는 이미 알고 있습니다. 가로 세로 28, 28 칸에 숫자들이 채워져 있습니다. 특징을 찾기 위한 컨벌루션 필터를 하나 가져왔습니다. 이 필터는 가로로 된 선이 어디서 나타났는지 찾는 필터 입니다. 입력 이미지와 이필터를 가지고 합성곱 연산을 하게 됩니다. 그 연산의 결과는 왼쪽상단에 있는 그림과 같이 2차원 형태의 숫자 집합으로 만들어 진 흑백의 이미지로 표현된 것입니다. 하얗게 보이는 부분은 숫자가 높은 것이고 검정색으로 보이는 부분은 숫자가 0에 가까운 것입니다. 컨벌루션의 결과가 대상 이미지의 가로 선이 있는 부분에서 하얗게 표현된 것이 보이시죠. 이렇게 대상 이미지로 부터 필터를 통해 특징을 잡아 낸 이 컨벌루션의 결과를 특징에 대한 위치정보가 표현된 지도란 의미로 특징맵(feature map)이라 부릅니다.

이번에는 세로 선에 특징을 찾기 위한 컨벌루션 필터를 가져왔습니다. 이 필터의 의해서 만들어진 특징맵은 왼쪽아래와 같습니다. 세로 선이 있던 부분이 하얗게 나타나는 걸 볼수 있습니다. 컨벌루션 필터 하나는 대상 이미지를 보고 특징맵 이미지 하나를 만들게 됩니다. 필터 하나가 이미지 하나를 만든다는 걸 기억해 주세요.

여기 조금 더 의미 있는 예가 있습니다. 컨볼루션 필터가 컨볼루션 연산을 하면서 이미지 한장을 만듭니다.



우하향 방향의 사선 필터로 찾은 특징맵과 우상향 방향의 사선필터로 찾은 특징맵이 만들어진게 보이시죠? 하얗게 표현된 부분이 찾고 싶은 특징이 많이 나타난 위치라고 이해 하면 되겠습니다.

이제 실제 컨볼루션 레이어가 적용된 딥러닝 코드를 확인 해 보겠습니다.

- 1. 과거의 데이터를 준비합니다.

```
(독립, 종속), _ = tf.keras.datasets.mnist.load_data()
독립 = 독립.reshape(60000, 28, 28, 1)
종속 = pd.get_dummies(종속)
print(독립.shape, 종속.shape)
```

- 2. 모델의 구조를 만듭니다.

```
X = tf.keras.layers.Input(shape=[28, 28, 1])
H = tf.keras.layers.Conv2D(3, kernel_size=5, activation='swish')(X) # 컨볼루션
레이어 추가 3개의 필터셋을 사용하므로 3개의 특징맵을 만든다.
H = tf.keras.layers.Conv2D(6, kernel_size=5, activation='swish')(H) # 컨볼루션
레이어 추가 6개의 필터셋을 사용하므로 6개의 특징맵을 만든다.
H = tf.keras.layers.Flatten()(H) # 한줄로 펼침
H = tf.keras.layers.Dense(84, activation='swish')(H)
Y = tf.keras.layers.Dense(10, activation='softmax')(H)
model = tf.keras.models.Model(X, Y)
model.compile(loss='categorical_crossentropy', metrics=['accuracy'])
```

컨벌루션 레이어를 2층 추가 하였습니다. 컨벌루션레이어 에서 우리가 결정할 것은

1. 필터셋을 몇 개 사용할 것인가
2. 필터셋의 사이즈를 얼마로 할 것인가

를 결정하는 것입니다. 3과 6은 사용할 필터셋의 갯수입니다. **kernel_size** 는 필터셋의 크기를 의미합니다. (5,5) 의 필터셋을 사용하기 위해서 5라고 입력한 것입니다. 그리고 컨벌루션 레이어는 기존의 **Dense** 레이어와 같이 활성화 함수 **activation** 을 지정해 주어야 합니다. 첫번째 컨벌루션 레이어는 필터셋이 3개 이기 때문에 3장에 특징 맵을 만듭니다. 다른 표현으로 3채널의 특징 맵을 만든다고 할 수 있습니다. 두번째 컨벌루션 레이어는 필터셋이 6개 이기 때문에 6장에 특징 맵을 만듭니다. 다른 표현으로 6채널의 특징 맵을 만든다고 할 수 있습니다. 그리고 이렇게 만들어진 6채널의 특징 맵모듈을 **Flatten** 레이어에 의해서 픽셀 단위로 한줄로 펼친 후 학습하게 됩니다.

필터의 이해

필터의 대해서 다음의 내용을 먼저 기억해 주세요.

1. 필터셋은 3차원 형태로 된 가중치의 모음
2. 필터셋 하나는 앞선 레이어의 결과인 "특징맵"전체를 본다.

전체를 보고 필터셋 하나가 특징 맵 하나를 만듭니다.

3. 필터셋 개수 만큼 특징 맵을 만든다.

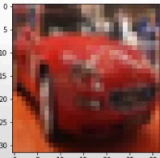
1. 필터셋은 3차원 형태로 된 가중치의 모음

2. 필터셋 하나는 앞선 레이어의 결과인 "특징맵" 전체를 본다.

3. 필터셋 개수 만큼 특징맵을 만든다.

Conv2D(3, kernel_size=5, activation='swish')

Conv2D(6, kernel_size=5, activation='swish')



(32, 32, 3)

F1 F2 F3

(5, 5, 3)

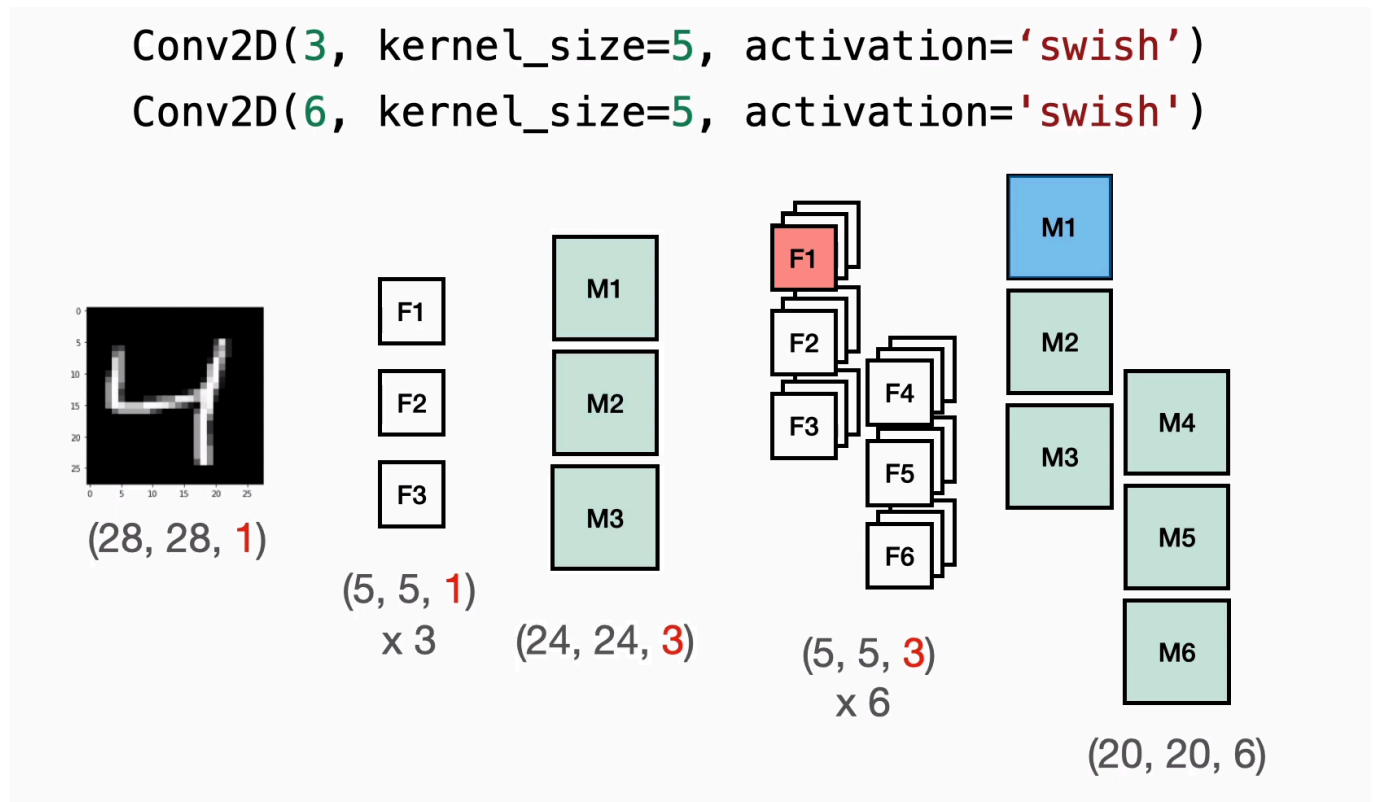
(3, 5, 5, ?)

F1 F2 F3 F4 F5 F6

(6, 5, 5, ?)

우리가 이번에 추가할 컨볼루션 레이어의 코드를 가져왔습니다. 첫번째 레이어는 필터셋 3개, 두번째 레이어는 필터셋 6개죠. 그리고 각 필터는 (5, 5)사이즈라고 했습니다. 3차원 형태의 가중치의 모음이라고 했고요. 레이어가 이렇게 생겼으니 첫번째 레이어의 필터셋은 (3, 5, 5) 두번째 레이어의 필터셋은 (6, 5, 5) 이겠네요. 이렇게 생각하면 안되겠기에 필터셋에 대한

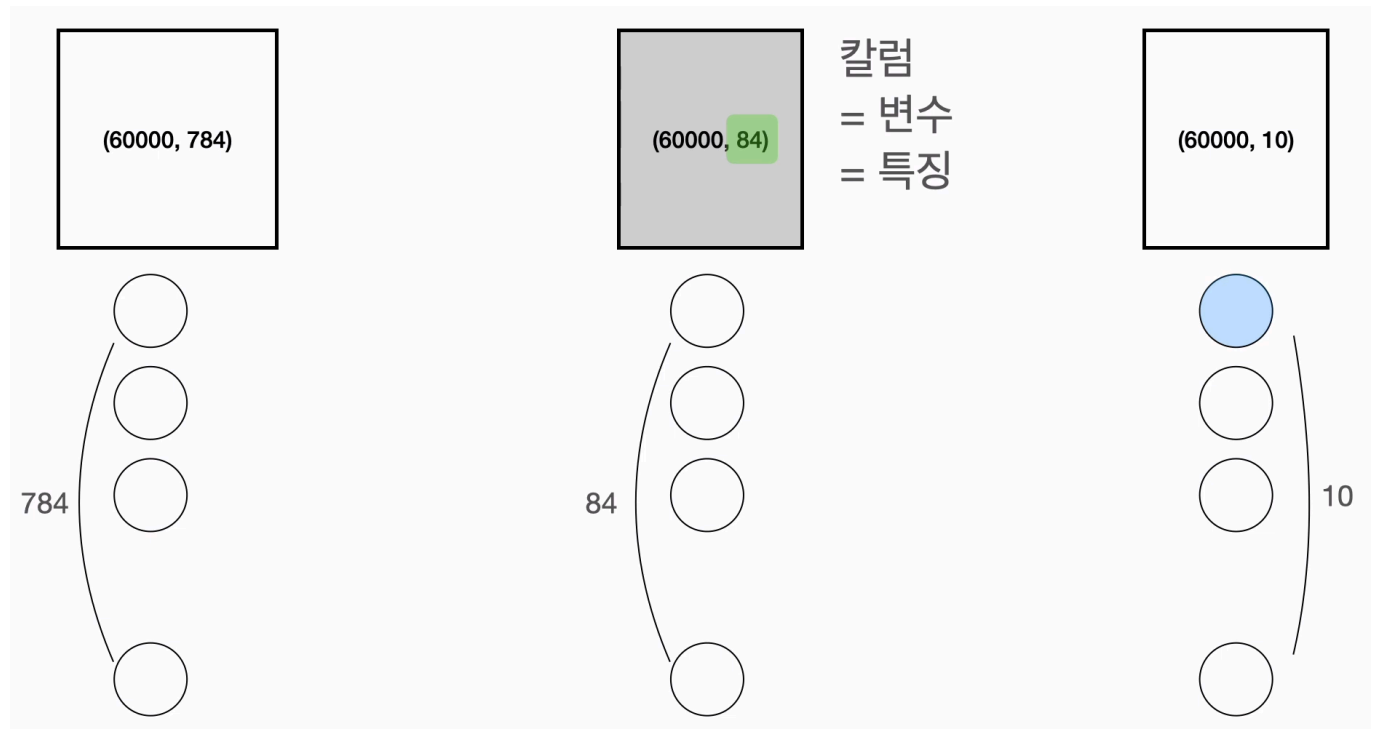
설명을 제대로 하려고 하는 것입니다. 3차원의 필터셋이라고 하는 것은 개별 필터셋 하나가 3차원 형태라고 말씀드리는 거예요. 즉 (5,5)필터가 여러 채널 존재하는 것입니다. 그리고 그 숫자는 앞선 레이어의 "특징맵" 전체를 본다는 두번째 설명과 관련이 있는데요. 전체를 보기 때문에 필터의 채널수가 화살표가 가르키는 곳에 세번째의 특징맵수와 같은 수가 들어가게 됩니다. 입력이 흑백 이라고 생각해 보세요. 흑백의 채널이 하나이니 필터셋의 형태는 (5, 5, 1)이 됩니다. 입력이 만약 컬러 이미지라면 채널이 생기므로 필터셋의 형태도 (5, 5, 3)이 되는 겁니다. 또한 필터셋 전체 모양은 다음 그림 맨 아래와 같이 4차원 형태가 되는 것입니다. 필터셋이 3개이므로 (3, 5, 5, ?) 뒤에 물음표는 앞선 채널의 개수가 동일하니 ?로 적어 놔습니다. 제가 설명을 돕기 위해 지금 이곳에서 필터셋이라는 표현을 쓰고 있는데요. (5,5)필터가 채널 수만큼 겹쳐진 형태이기 때문에 필터셋이라고 표현을 했습니다. 하지만 실제 이분야에서는 저렇게 여러 채널로 구성된 형태의 그대로 필터라고 합니다. 필터셋이라는 용어는 설명을 돕기 위해 이수업에서만 사용하는 용어 입니다. 이 이후에 공부에 참고하시면 됩니다.



입력 이미지가 있습니다. (28, 28)의 흑백 이미지라서 채널이 1개입니다. (5,5,1)의 필터셋이 3개 준비되었고 필터셋 하나는 하나의 특징맵을 생성하므로 이 필터셋에 의하여 특징맵 3장, 3채널의 특징맵이 생성됩니다. 사이즈가 28에서 24로 줄었는 데요. 4만큼 줄었죠. 그,이유는 필터의 사이즈가 5이기 때문이에요. 사이즈에서 일을 뺀 수만큼 사이즈가 감소합니다. 필터가 (5,5)이면 4가 줄고, (3,3)이면 2가 줄어듭니다.

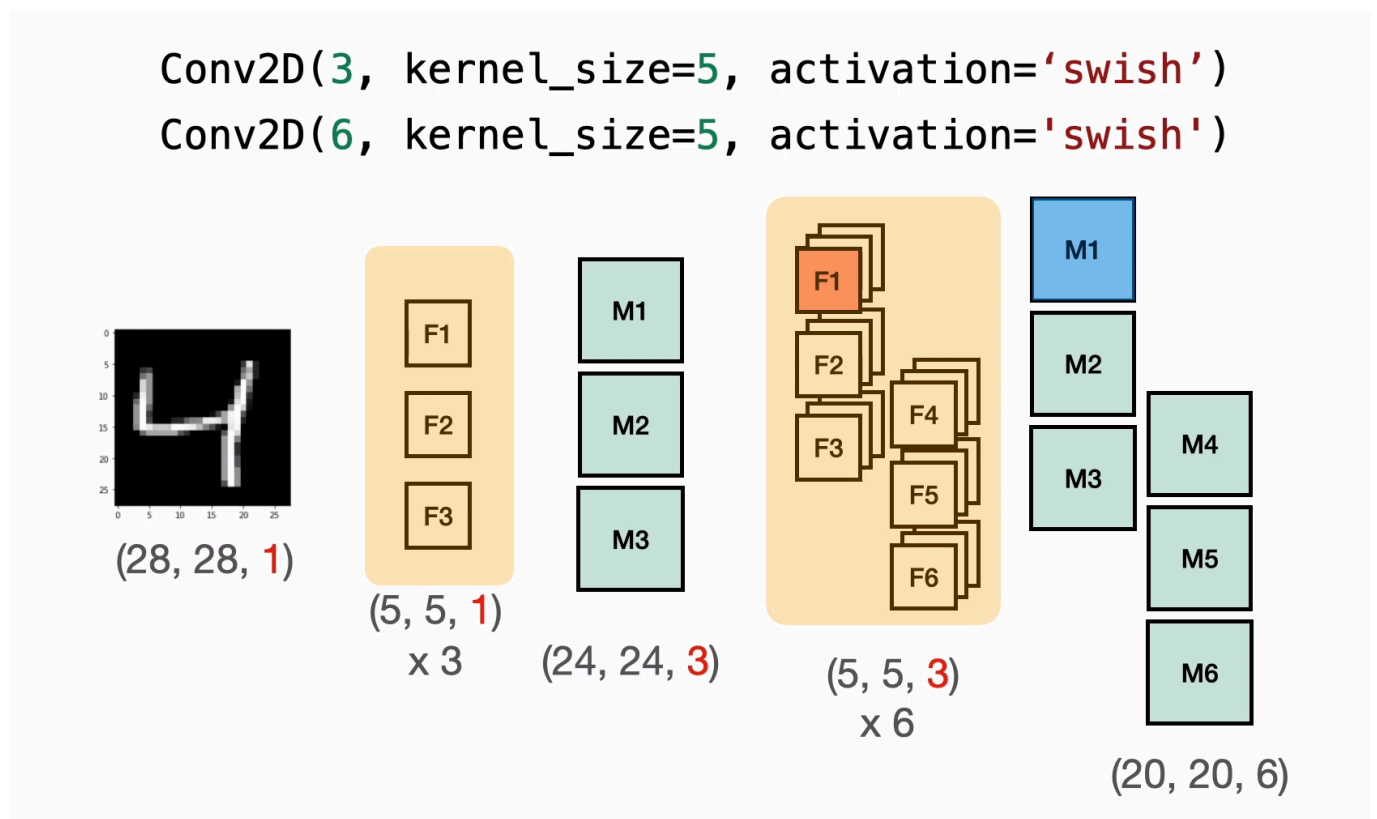
두번째 컨벌루션 레이어에서는 6장의 필터셋을 준비했습니다. 필터셋의 채널수가 앞선 레이어의 결과인 특징맵 채널수와 같은 3으로 되어 있네요. 필터셋이 6개인 6채널의 특징맵이 생성이 됩니다. 특징맵 하나를 만들 때 필터셋 하나는 앞에 특징맵 전체를 보고 새로운 특징맵 하나를 만드는 것입니다.

이전에 표에 특징을 학습 Flatten 모델을 레이어와 노드로 표현한 그림입니다.



하나의 특징을 만들 때 앞선 레이의 노드 전체를 사용하여 학습을 한다고 했습니다. 앞선 특징맵 전체를 보고 새로운 특징맵 하나를 만든다는 것은 이것과 같은 의미예요. 또한 이 노드의 84개의 특징은 컴퓨터가 가중치를 학습해서 스스로 찾아낸 특징이고 그래서 딥러닝에게는 "특징 자동 추출기"라는 별명이 붙어 있다고 했습니다.

컨벌루션 레이어에서는 컨벌루션 필터들을 뜯어보면 가중치 들로 이루어져 있고 컴퓨터는 특징맵을 만들어 내는 컨벌루션 필터를 학습하는 것입니다.



앞서 히든 레이어를 추가 하는 것은 지정한 노드개수만큼 컴퓨터에게 분류를 위한 가장 좋은 특징을 찾아 달라고 요청하는 거라고 말씀 드렸습니다. 컨벌루션 레이어 에서도 마찬가지로 필터를 6개 추가하는 행동이 이렇게 명령하는 걸로 생각 할 수 있습니다.

"컴퓨터야 이 이미지들이 0~9 까지 중 어느 숫자인지 판단하기 위해 가장 좋은 특징맵 6개를 찾아줘"

라고 말이죠.

"특징 자동 추출기" 라는 별명이 딱 어울리지 않나요.

다시한번 이별명을 중심으로 모델을 한번 천천히 음미해 보세요. CNN을 넘어 딥러닝 이란 녀석이 보다 편하게 느껴지는 순간이 올것입니다. 이제 CNN을 정복한 것이나 마찬가지입니다.

```
1  # -*- coding: utf-8 -*-
2  import tensorflow as tf
3  import pandas as pd
4
5  (독립, 종속), _ = tf.keras.datasets.mnist.load_data()
6  print(독립.shape, 종속.shape)
7
8  독립 = 독립.reshape(60000, 28, 28, 1)
9  종속 = pd.get_dummies(종속)
10 print(독립.shape, 종속.shape)
11
12 X = tf.keras.layers.Input(shape=[28, 28, 1])
13 H = tf.keras.layers.Conv2D(3, kernel_size=5, activation='swish')(X)
14 H = tf.keras.layers.Conv2D(6, kernel_size=5, activation='swish')(H)
15 H = tf.keras.layers.Flatten()(H)
16 H = tf.keras.layers.Dense(84, activation='swish')(H)
17 Y = tf.keras.layers.Dense(10, activation='softmax')(H)
18 model = tf.keras.models.Model(X, Y)
19 model.compile(loss='categorical_crossentropy', metrics=['accuracy'])
20
21 model.fit(독립, 종속, epochs=10)
22
23 pred = model.predict(독립[0:5])
24 pd.DataFrame(pred).round(2)
25
26 종속[0:5]
27
28 model.summary()
```


Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 28, 28, 1)	0
conv2d (Conv2D)	(None, 24, 24, 3)	78
conv2d_1 (Conv2D)	(None, 20, 20, 6)	456
flatten (Flatten)	(None, 2400)	0
dense (Dense)	(None, 84)	201,684
dense_1 (Dense)	(None, 10)	850

Total params: 406,138 (1.55 MB)

Trainable params: 203,068 (793.23 KB)

Non-trainable params: 0 (0.00 B)

Optimizer params: 203,070 (793.25 KB)

모델의 모양이 확인해보면 이미지가 (28, 28, 1)이 들어가서 (24, 24)로 줄어드는 것을 볼 수 있습니다. 필터가 (5,5)이면 (4,4)픽셀로 줄어듭니다. 두 번째 컨볼루션을 통과한 다음에 이미지의 모양이 이미지 한장이 input인데 6장의 특징맵이 만들어졌어요. 이미지 하나마다 6장이 만들어진 거예요. 그리고 6장의 이미지 1장에서 특징을 뽑아낸 6개의 특징맵입니다. 그러니까 이미지 한장에 대한 정보가 (20, 20, 6) 이안에 담겨 있는 거예요. 그러면 여기에는 몇개의 픽셀이 들어 있냐면, 이미지 한장당 400 픽셀이니까 2400 픽셀이 들어가 있을 것이고 이것을 한줄로 펴면 컬럼이 2400컬럼이 되는 것입니다.

그리고 그 다음에 텐스레이오로 우리가 84개의 컬럼을 만들라고 했구요. 마지막에 10개의 컬럼으로 만들어 달라고 했습니다.

이렇게 모델을 만들어 보니까 학습하는 파라미터가 어마어마하게 늘어난 것을 볼 수 있습니다. 40만개가 넘어가는 파라미터를 컴퓨터가 학습하게 됩니다. 이렇게 늘어 났다는 것은 학습속도가 굉장히 느려졌다는 얘기고 컴퓨터가 굉장히 많은 연산을 하게 된다는 얘기가 됩니다. 학습 할 때마다 그 다음에 엄청나게 늘어나는 형태의 모델입니다. 이것을 해결하는 것이 다음 예제입니다.

(8, 8, 1)

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

(3, 3, 1)

-1	-1	-1
2	2	2
-1	-1	-1

필터

(6, 6)

1	1	0	1		

$$y = 0 * -1 + 0 * -1 + 0 * -1 + 0 * 2 + 0 * 2 + 1 * 2 + 0 * -1 + 0 * -1 + 1 * -1$$

(8, 8, 1)

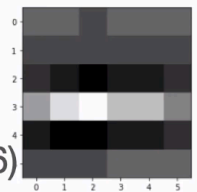
0	0	0	0	0	0	0	0
0	1	0	0	0	1	0	0
0	1	0	0	0	1	0	0
0	1	0	0	0	1	0	0
0	1	1	1	1	1	1	0
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0

(3, 3, 1)

-1	-1	-1
2	2	2
-1	-1	-1

필터

(6, 6)



1	1	0	1	1	1
0	0	0	0	0	0
-1	-2	-3	-2	-2	-1
3	5	6	4	4	2
-2	-3	-3	-2	-2	-1
0	0	0	1	1	1

$$y = * -1 + * -1 + * -1 + * 2 + * 2 + * 2 + * -1 + * -1 + * -1$$