합성곱 신경망 소개

컨브넷(convnet) = 합성곱 신경망(convolutional neural network) 대부분 컴퓨터 비전에 사용된다.

간단한 커브넷 만들기

```
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

컨브넷이 (image_height, image_width, image_channels) 크기의 입력 텐서를 사용한다.

컨브넷의 구조

```
>>> model.summary()

Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 26, 26, 32)        320

 max_pooling2d_2 (MaxPooling  (None, 13, 13, 32)       0
 2D)

 conv2d_4 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_3 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 conv2d_5 (Conv2D)           (None, 3, 3, 64)          36928

=================================================================
Total params: 55,744
Trainable params: 55,744
```

```
Non-trainable params: 0
_________________________________________________________________
```

Conv2D와 MaxPooling2D 층의 출력은 (height, width, channels)크기의 3D 텐서이다.

이 분류기는 1D 벡터를 처리하는데, 이전 층의 출력이 3D 텐서이다. 그래서 먼저 3D출력을 1D 텐서로 펼쳐야 한다. 그 다음 몇 개의 Dense 층을 추가한다.

컨브넷 위에 분류기 추가하기

```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

>>> model.summary()

Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 26, 26, 32)        320

 max_pooling2d_2 (MaxPooling  (None, 13, 13, 32)       0
 2D)

 conv2d_4 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_3 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 conv2d_5 (Conv2D)           (None, 3, 3, 64)          36928

 flatten (Flatten)           (None, 576)               0

 dense (Dense)               (None, 64)                36928

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
_________________________________________________________________
```

여기서 볼 수 있듯이, (3, 3, 64) 출력이 (576,) 크기의 벡터로 펼쳐진 후 Dense 층으로 주입된다.

이제 MNIST 숫자 이미지에 컨브넷을 훈련 시키자. 2장의 예제 코드를 재사용하자.

MNIST 이미지에 컨브넷 훈련하기

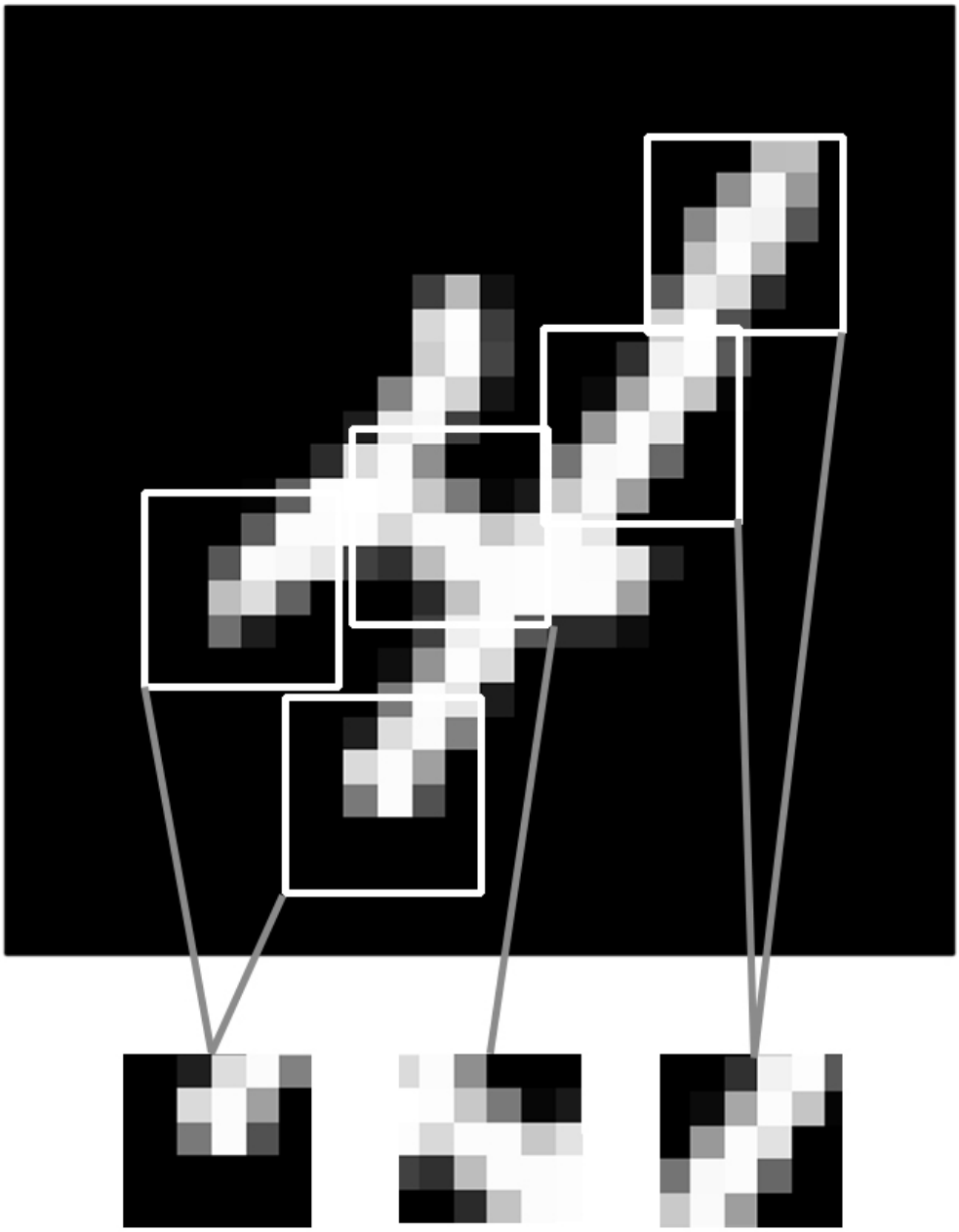
```
from keras.datasets import mnist
from keras.utils import to_categorical

# MNIST 데이터셋을 로드합니다.
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# 학습 이미지 데이터를 (60000, 28, 28, 1) 크기로 변환합니다.
train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255

# 테스트 이미지 데이터를 (10000, 28, 28, 1) 크기로 변환합니다.
test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255

# 학습 레이블을 원-핫 인코딩합니다.
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

# 모델을 컴파일합니다.
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# 모델을 학습합니다.
model.fit(train_images, train_labels, epochs=5, batch_size=64)

>>> test_loss, test_acc = model.evaluate(test_images, test_labels)
>>> test_acc

313/313 [==============================] - 1s 3ms/step - loss: 0.0383 - accuracy: 0.9913
0.9912999868392944
```

2장의 완전 연결 네트워크는 97.8% 정확도를 얻었지만, 기본적인 컨브넷은 99.2%의 테스트 정확도를 얻었다. 완전 연결된 모델보다 왜 간단한 컨브넷이 더 잘 작동할까?

Conv2D와 MaxPooling2D 층이 어떤 일을 하는지 알아보자.

합성곱 연산

완전 열결 층과 합성곱 층의 차이는 Dense층은 입력 특성 공간에 있는 전역 패턴을 학습하지만, 합성곱 층은 지역 패턴을 학습한다. 이미지일 경우 2D 윈도우로 입력에서 패턴을 찾는다. 앞의 예에서 윈도우는 3 X 3 크기였다.



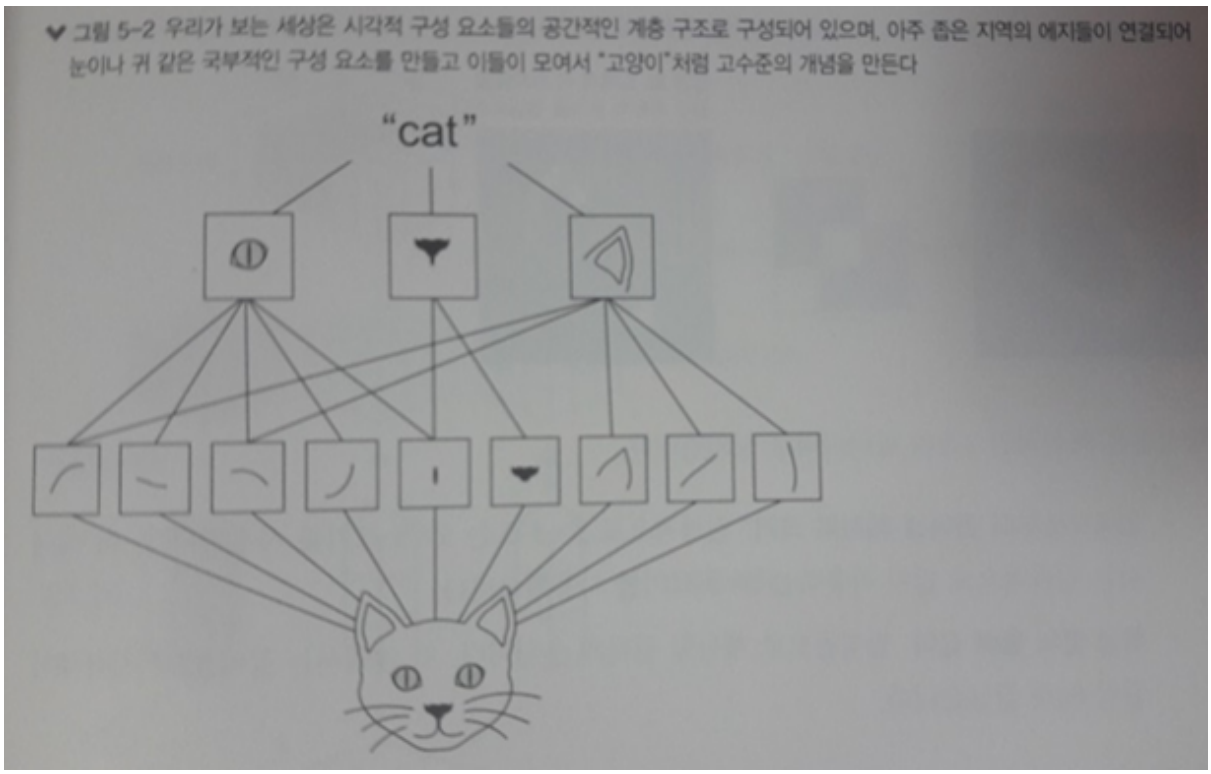
이미지는 에지(edge), 질감(texture) 등 지역 패턴으로 분해될 수 있다.

이 핵심 특징은 컨브넷에 두 가지 흥미로운 성질을 제공한다.

- **학습된 패턴은 평행 이동 불변성을 가진다** : 컨브넷이 이미지의 어떤 곳에서 어떤 패턴을 학습했다면 다른 곳에서도 이 패턴을 인식할 수 있다. 하지만 완전 연결 네트워크는 새로운 위치에 나타난 것은 새로운 패턴으로 학습해야 한다. 이런 컨브넷의 장점은 이미지를 효율적으로 처리하게 만들어준다. 적은 수의 훈련 샘플을 사용해서 일반화 능력을 가진 표현을 학습할 수 있다.

- **컨브넷은 패턴의 공간적 계층 구조를 학습할 수 있다** : 첫 번째 합성곱 층이 에지 같은 작은 지역 패턴을 학습한다. 두 번째 합성곱 층은 첫 번째 층의 특성으로 구성된 더 큰 패턴을 학습하는 식이다. 이런 방식을 사용하여 컨브넷은 매우 복잡하고 추상적인 시각적 개념을 효과적으로 학습할 수 있다.

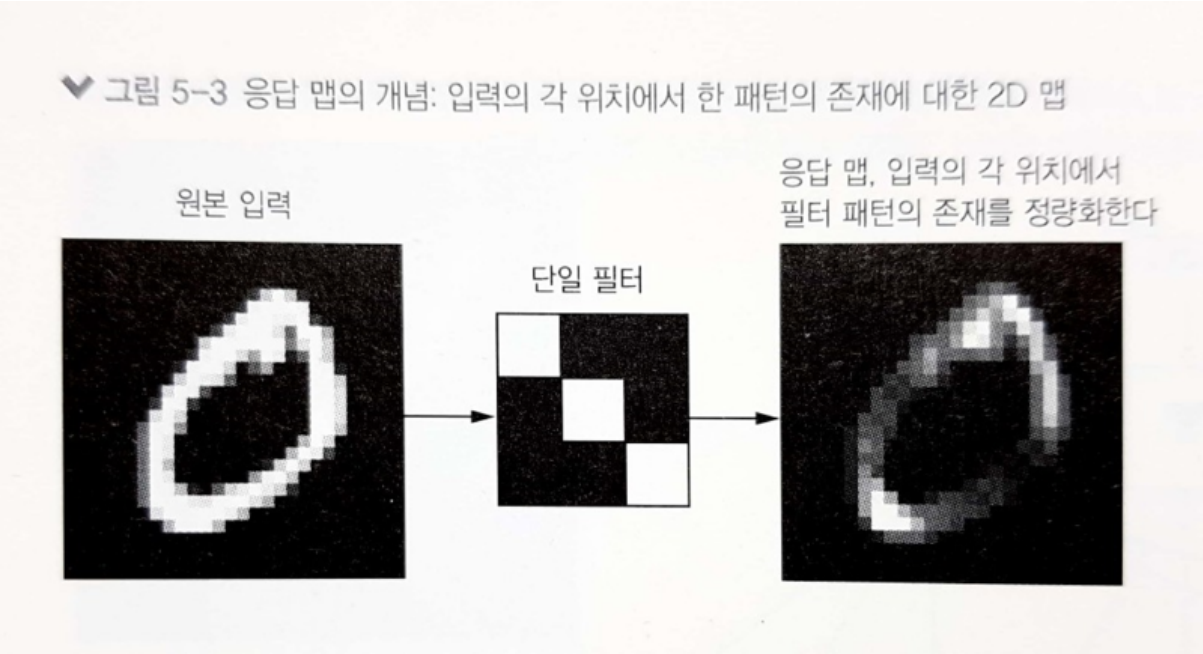
합성곱 연산은 특성 맵이라고 부르는 3D 텐서에 적용된다. 이 텐서는 2개의 공간 축(높이와 너비)과 깊이 축(채널 축이라고도 한다.)으로 구성된다. RGB 이미지는 3개의 컬러채널(빨간색, 녹색, 파란색)을 가지므로 깊이 축의 차원이 3이 된다. MNIST숫자처럼 흑백 이미지는 깊이 축의 차원이 1이다.



출력 특성 맵도 높이와 너비를 가진 3D 텐서이다. 출력 텐서의 깊이는 층의 매개변수로 결정되기 때문에 상황에 따라 다르다. 일종의 **필터**라고 한다. 필터는 입력 데이터의 어떤 특성을 인코딩한다. 예를 들어 고수준으로 보면 하나의 필터가 '입력에 얼굴이 있는지'를 인코딩할 수 있다.

MNIST 예제에서는 첫 번째 합성곱 층이 (28, 28, 1) 크기의 특성 맵을 입력으로 받아 (26, 26, 32) 크기의 특성 맵을 출력한다. 즉 입력에 대해 32개의 필터를 적용한다. 32개의 출력 채널은 각각 26 x 26 크기의 배열 값을 가진다. 이 값은 입력에 대한 필터의 응답 맵이다. = 입력의 각 위치에서 필터 패턴에 대한 응답을 나타낸다.

특성 맵이란 말이 의미하는 것은, 깊이 축에 있는 각 차원은 하나의 특성(필터)이고, 2D 텐서 output[:, :, n]은 입력에 대한 이 필터 응답을 나타내는 2D 공간상의 맵이다.



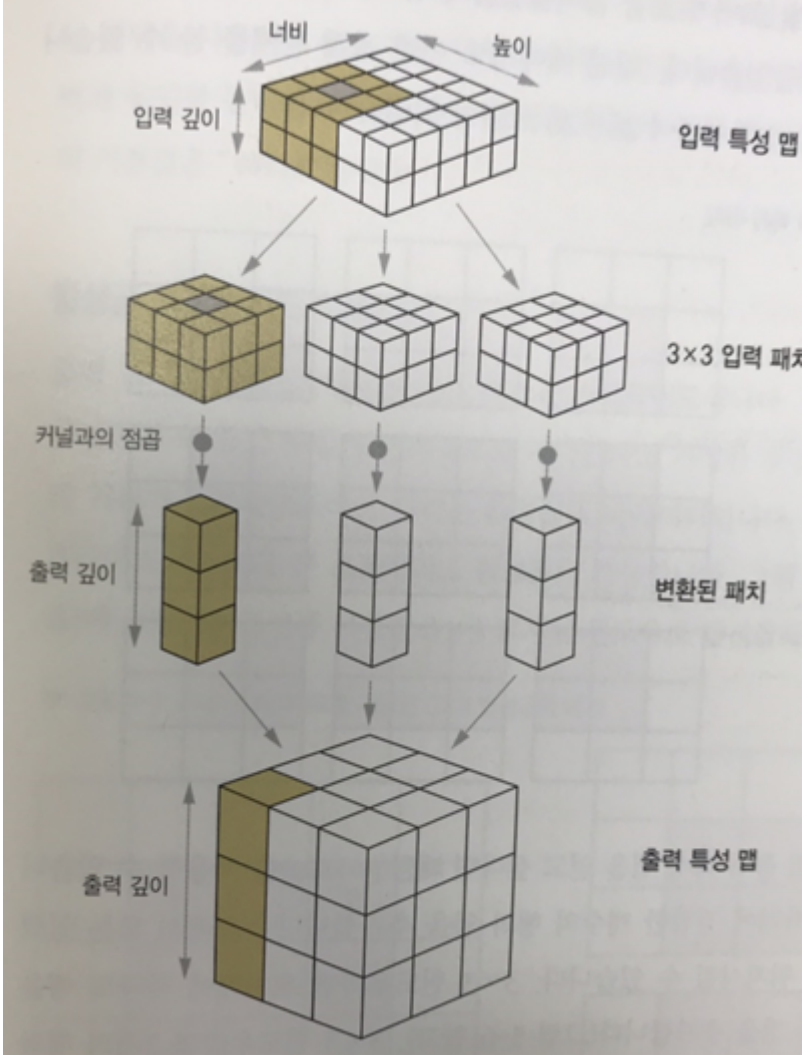
합성곱은 핵심적인 2개의 파라미터로 정의된다.

- **입력으로부터 뽑아낼 패치의 크기** : 전형적으로 3x3 또는 5x5 크기를 사용한다.
- **특성 맵의 출력 깊이** : 합성곱으로 계산할 필터의 수이다. 이 예에선 깊이 32로 시작해 64로 끝났다.

케라스의 Conv2D층에서 이 파라미터는 Conv2D(깊이, (패치의 세로, 패치의 가로)처럼 천 번째와 두 번째의 매개변수로 전달된다.

3D 입력 특성 맵 위를 3x3 Ehssm 5x5크기의 윈도우가 슬리이딩하면서 모든 위치에서 3D 특성패치Conv2D(깊이, (패치의 세로, 패치의 가로)의 크기를 추출하는 방식으로 합성곱이 작동한다.

이런 3D 패치는 출력 깊이 크기의 1D 벡터로 변환된다. 합성곱 커널이라고 불리는 하나의 학습된 가중치 행렬과의 텐서 곱셈을 통하여 변환된다.

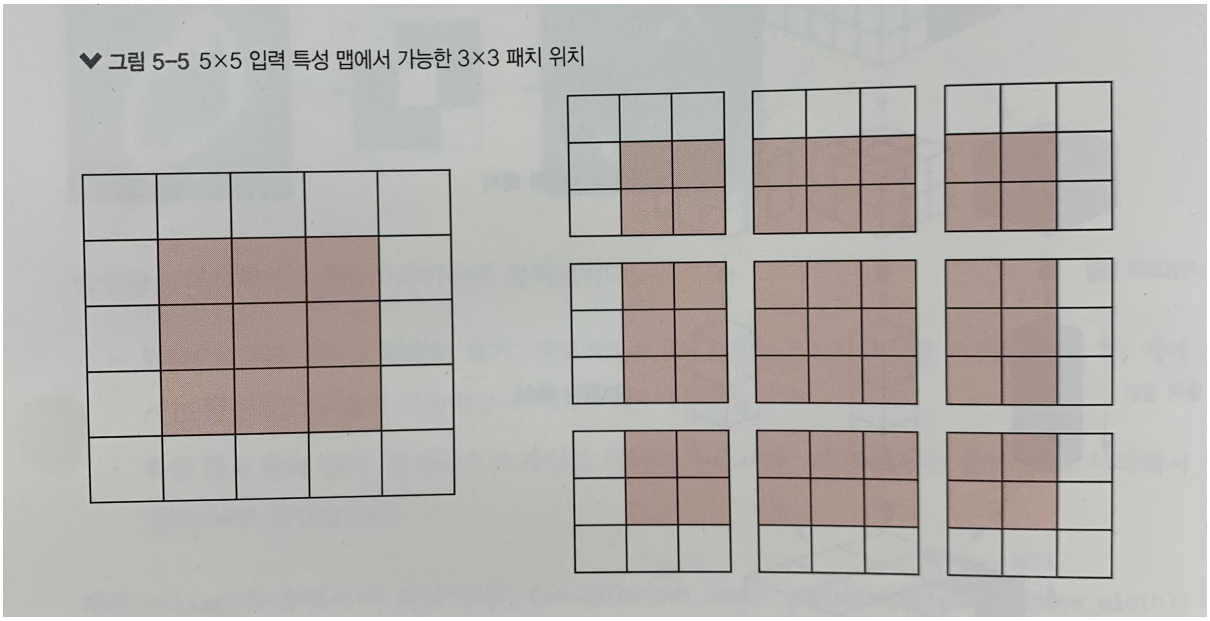


출력 높이와 너비는 입력의 높이. 너비와 다를 수 있다. 여기엔 두 가지 이유가 있다.

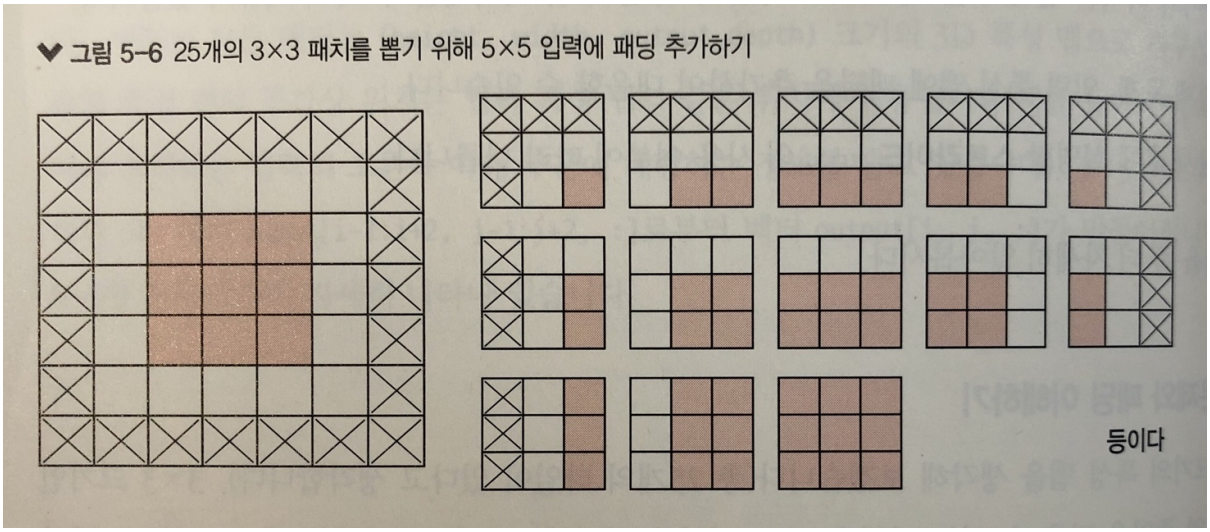
- 경계 문제, 입력 특성 맵에 패딩을 추가하여 대응할 수 있다
- 뒤에 설명할 스트라이드의 사용 여부에 따라 다르다.

경계 문제와 패딩 이해하기

5x5 크기의 특성 맵을 생각해보자. (총 25개의 타일이 있는 것이다.) 3x3 크기인 윈도우의 중앙을 맞출 수 있는 타일은 3x3 격자를 형성하는 9개 뿐이다. 따라서 출력 특성 맵은 3x3 크기가 된다. 여기서는 높이와 너비 차원을 따라 정확히 2개의 타일이 줄었다. 앞선 예에서도 이런 경계 문제를 볼 수 있다. 첫 번째 합성곱 층에서 28 x 28 크기의 입력이 26 x 26 크기가 되었다.



입력과 동일한 높이와 너비를 가진 출력 특성 맵을 얻고 싶다면 **패딩**을 사용할 수 있다.



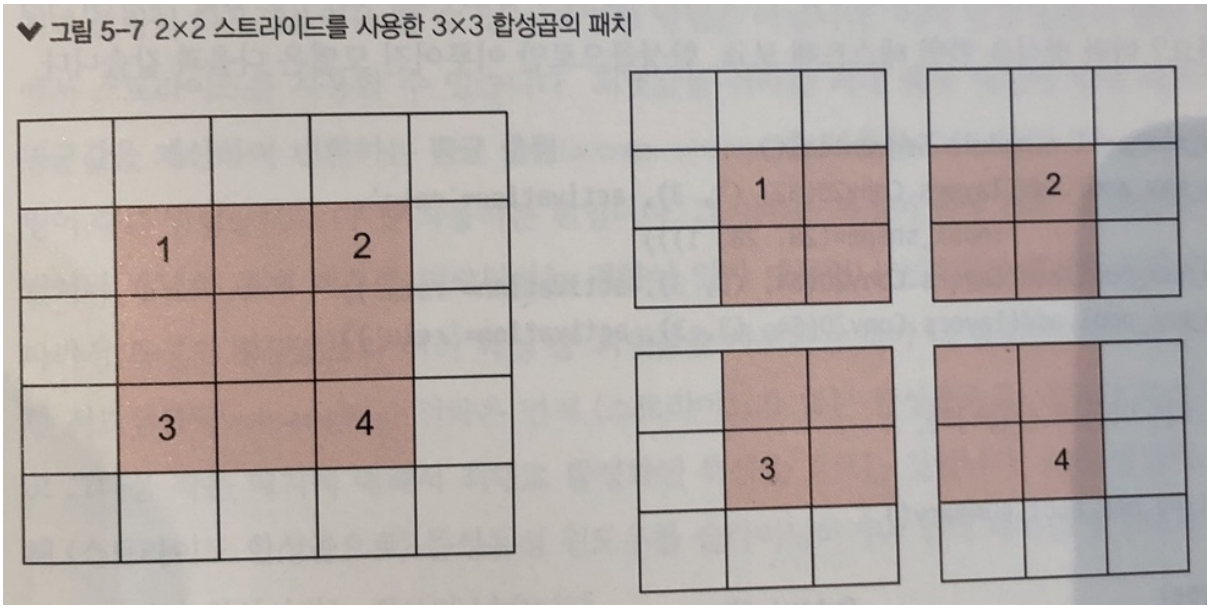
패딩은 입력 데이터의 가장자리에 추가 정보를 제공하여 경계 문제를 완화하고, 모델의 성능을 향상시키는 데 도움을 준다.

Conv2D 층에서 패딩은 padding 매개변수로 설정할 수 있다. 2개의 값이 가능하다. “valid”는 패딩을 사용하지 않는다는 뜻이다. “same” 은 입력과 동일한 높이와 너비를 가진 출력을 만들기 위해 패딩한다. 라는 뜻이다.

합성곱 스트라이드 이해하기

합성곱 연산에서 스트라이드(stride)는 필터가 입력 데이터를 얼마나 이동하는지를 결정하는 파라미터이다. 두 번의 연속적인 윈도우 사이의 거리가 스트라이드라고 불리는 합성곱의 파라미터이다. 기본 값은 1이다, 스트라이드가 1보다 큰 스트라이드 합성곱도 가능하다.

2x2 스트라이드를 사용한 3x3 합성곱의 패치



스트라이드 2를 사용했다는 것은 특성 맵의 너비와 높이가 2의 배수로 다운샘플링되었다는 뜻이다. 스트라이드 합성곱은 실전에서 드물게 사용된다. 하지만 유용하게 사용될 수 있으므로 기억하자. 스트라이드 대신 최대 풀링 연산을 하는 경우가 많다. 좀 더 자세히 알아보자.

최대 풀링 연산

최대 풀링(MaxPooling)은 합성곱 신경망(CNN)에서 사용되는 풀링 연산 중 하나로, 입력 데이터의 영역에서 최대값을 선택하여 특징 맵의 크기를 줄이는 연산이다. 주로 이미지 처리 작업에서 특징을 추출하고 정보를 압축하기 위해 사용된다.

최대 풀링은 입력 특성 맵에서 윈도우에 맞는 패치를 추출하고 각 채널별로 최댓값을 출력한다.

합성곱과 개념적으로 비슷하지만 추출한 패치에 학습된 선형 변환을 적용하는 대신 하드코딩된 최댓값 추출 연산을 사용한다. 합성곱과 가장 큰 차이점은 최대 풀링은 보통 2x2 윈도우와 스트라이드 2를 사용하여 특성 맵을 절반 크기로 다운샘플링한다는 것이다. 이에 반해 합성곱은 전형적으로 3x3 윈도우와 스트라이드 1을 사용한다.

다운 샘플링을 사용하는 이유는 처리할 특성 맵의 가중치 개수를 줄이기 위해서이다. 또 연속적인 합성곱 층이 점점 커진 윈도우를 통해 바라보도록 만들어 필터의 공간적인 계층 구조를 구성한다.

최댓값을 취하는 최대 풀링 대신에 입력 패치의 채널별 평균값을 계산하여 변환하는 **평균 풀링**을 사용할 수도 있다.