Week3_예습과제_팽소원

5장 합성곱 신경망 I

5.1 합성곱 신경망

역전파는 순전파 과정에 따라 계산된 오차 정보가 신경망의 모든 노드로 전송

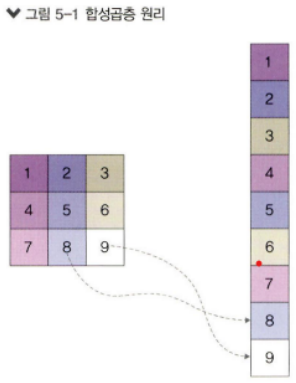
→ 계산 과정 복잡, 시간 ↑, 많은 자원 요구

이러한 문제를 해결하기 위해 합성곱 신경망 등장

합성곱 신경망: 이미지의 국소적 부분 계산 → 시간&자원 절약

5.1.1 합성곱층의 필요성

합성곱 신경망 : 이미지나 영상을 처리하는데 유용



이미지 분석은 3*3 배열을 오른쪽과 같이 펼쳐서 각 픽셀에 가중치를 곱하여 은닉층으로 전달

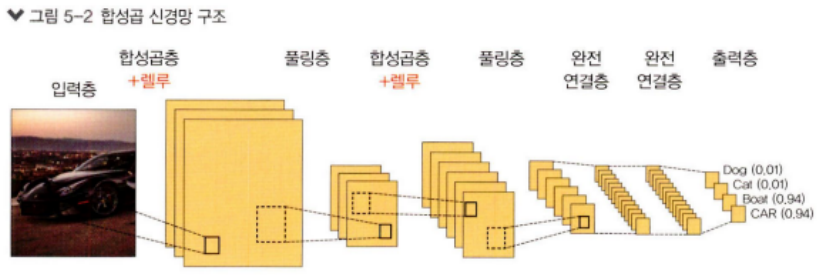
BUT 이렇게 이미지를 펼쳐서 분석하면 데이터의 공간적 구조 무시

⇒ 이러한 문제를 방지하고자 합성곱층 등장

5.1.2 합성곱 신경망 구조

합성곱 신경망 : 음성인식이나 이미지/영상 인식에서 주로 사용

①입력층, ②합성곱층, ③풀링층, ④완전연결층, ⑤출력층으로 구성되어 있음



*합성곱층*과 *풀링층*을 거치면서 *입력 이미지*의 주요 특성 벡터를 추출

추출된 주요 특성 벡터들은 *완전연결층*을 거치면서 1차원 벡터로 변환

*출력층*에서 활성화 함수인 소프트맥스 함수를 사용하여 최종 결과 출력

입력층

입력층: 입력이미지 데이터가 최초로 거치게 되는 계층

이미지는 단순 1차원 데이터가 아닌 높이, 너비, 채널을 갖는 3차원 데이터임

채널은 이미지가 gray scale이면 1, 컬러면 3 값을 가짐

합성곱층

합성곱층: 입력 데이터에서 특성을 추출하는 역할

입력 이미지가 들어왔을 때, 이미지에 대한 특성을 감지하기 위해 커널이나 필터 사용

커널/필터 : 이미지의 모든 영역을 훑으면서 특성을 추출 (주로 3*3, 5*5 크기)

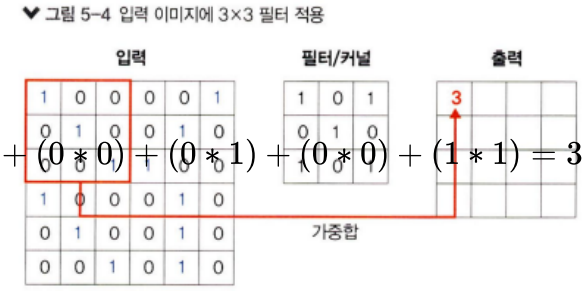
이렇게 추출된 결과물 = 특성 맵(feature map)

이미지 크기가 (6,6,1)이며, 3*3 크기의 커널이 스트라이드(필터를 적용하는 위치의 간격)이 1일때 예시 과정

1단계. 입력 이미지에 3*3 필터 적용

입력 이미지와 필터를 포개 놓고 대응되는 숫자끼리 곱한 후 모두 더함

$$(1*1)+(0*0)+(0*1)+(0*0)+(1*1)+(0*0)+(0*1)+(0*0)+(1*1)=3$$



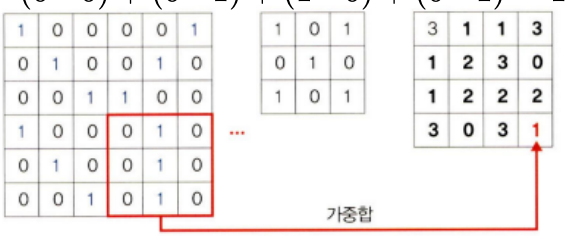
2단계. 필터가 1만큼 이동

$$(0*1)+(0*0)+(0*1)+(1*0)+(0*1)+(0*0)+(0*1)+(1*0)+(1*1)=1$$

3단계. 필터가 1만큼 마지막으로 이동

$$(0*1)+(1*0)+(0*1)+(0*0)+(1*1)+(0*0)+(0*1)+(1*0)+(0*1)=1$$

❯ 그림 5-9 입력 이미지에 필터가 1만큼 마지막으로 이동



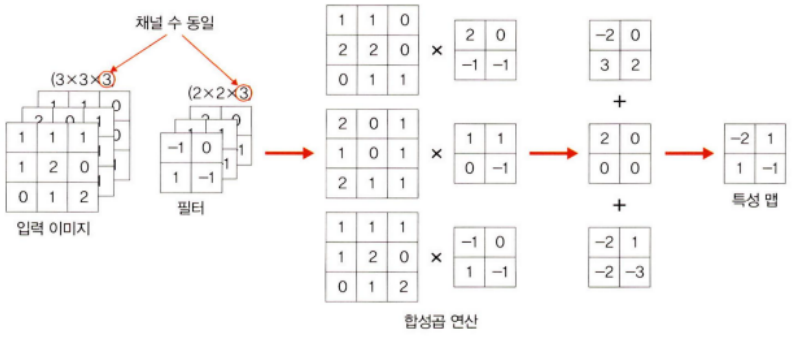
커널은 스트라이드 간격만큼 순회하면서 모든 입력값과의 합성곱 연산으로 새로운 특성 맵을 만들게 됨

위에 예시 같이 커널과 스트라이드의 상호 작용으로 원본(6,6,1) 크기가 (4,4,1) 크기의 특성 맵으로 줆

컬러 이미지는 그레이스케일 이미지와 다르게 ①필터 채널이 3개이며, ② RGB 각각에 서로 다른 가중치로 합성곱을 적용한 후 결과를 더해준다.

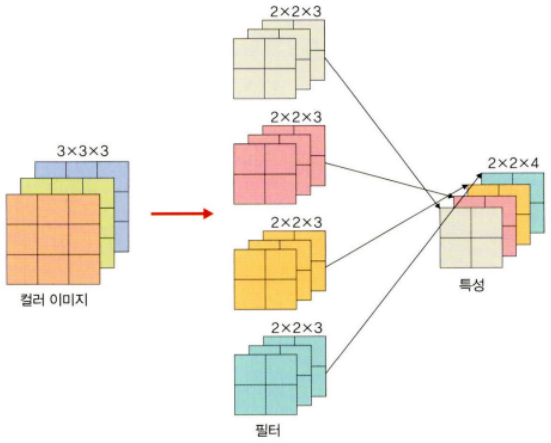
필터 채널 = 3, 필터 개수 = 3이 아니라 1

❯ 그림 5-10 컬러 이미지 합성곱



필터 개수가 2개 이상이라면 필터 각각은 특성 추출 결과의 채널이 됨

❯ 그림 5-11 필터가 2 이상인 합성곱



- **입력 데이터**: $W_1 \times H_1 \times D_1$ (W_1: 가로, $\times H_1$: 세로, $\times D_1$: 채널 또는 깊이)
- **하이퍼파라미터**
  - 필터 개수: K
  - 필터 크기: F
  - 스트라이드: S
  - 패딩: P
- **출력 데이터**
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$
  - $D_2 = K$

풀링층

풀링층: 합성곱층과 유사하게 특성 맵의 차원을 다운 샘플링하여 연산량 ↓, 주요 특성 벡터를 추출하여 효과적으로 학습

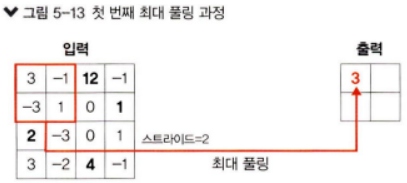
풀링 연산에는 주로 두가지가 사용

1. 최대 풀링 : 대상 영역에서 최댓값을 추출 (대부분 최대 풀링 사용)
2. 평균 풀링 : 대상 영역에서 평균을 반환 (특성이 희미해질 수 있음)

최대 풀링의 연산 과정 예시

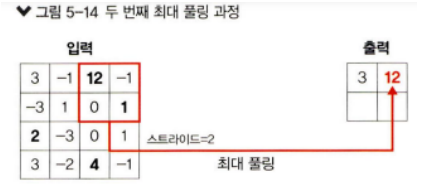
첫 번째 최대 풀링 과정

3, -1, -3, 1 값 중에서 최댓값(=3) 선택



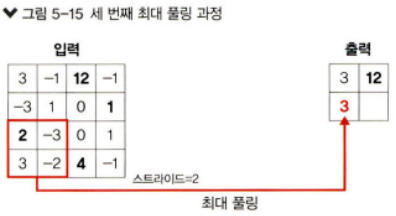
두 번째 최대 풀링 과정

12, -1, 0, 1 값 중에서 최댓값(=12) 선택



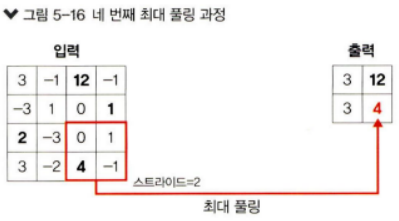
세 번째 최대 풀링 과정

2, -3, 3, -2 값 중에서 최댓값(=3) 선택



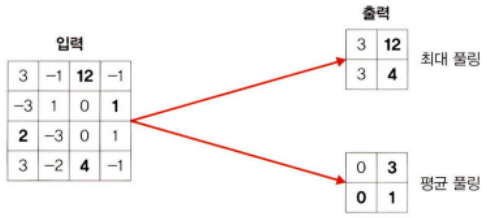
네 번째 최대 풀링 과정

0, 1, 4, -1 값 중에서 최댓값(-4) 선택



평균 풀링의 계산 과정은 최대 풀링과 유사한 방식으로 진행하지만 각 필터의 평균으로 계산함

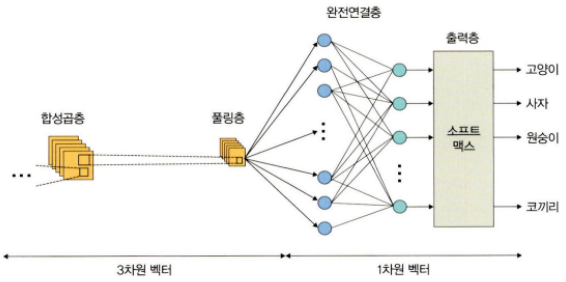
❯ 그림 5-17 최대 풀링과 평균 풀링 비교



- **입력 데이터**: $W_1 \times H_1 \times D_1$
- **하이퍼파라미터**
  - 필터 크기: F
  - 스트라이드: S
- **출력 데이터**
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$

완전연결층

❯ 그림 5-18 완전연결층



합성곱층과 풀링층을 거치면서 차원이 축소된 특성 맵은 최종적으로 완전연결층으로 전달된다.

완전연결층에서 이미지는 3차원 벡터에서 1차원 벡터로 펼쳐진다.

출력층

출력층: 소프트맥스 함수(입력받은 값을 0~1로 출력)가 사용하여 이미지가 각 레이블에 속할 확률값 출력
→ 가장 높은 확률 값을 갖는 레이블이 최종 값으로 선정

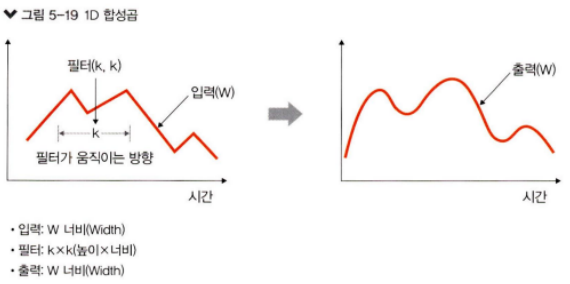
5.1.3 1D, 2D, 3D 합성곱

합성곱은 이동하는 방향의 수와 출력 형태에 따라 1D, 2D, 3D로 분류 가능

1D 합성곱

1D 합성곱: 필터가 시간을 축으로 좌우로만 이동할 수 있는 합성곱

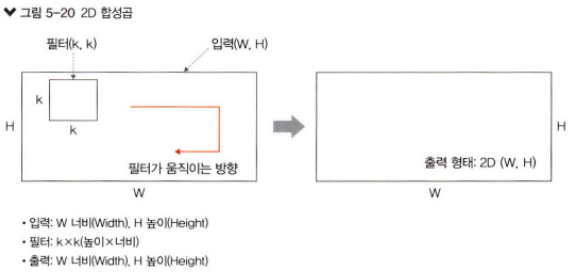
❯ 그림 5-19 1D 합성곱



EX) 입력이 [1, 1, 1, 1, 1] 이고 필터가 [0.25, 0.5, 0.25]라면, 출력은 [1, 1, 1]

그래프 곡선을 완화할 때 많이 사용

2D 합성곱

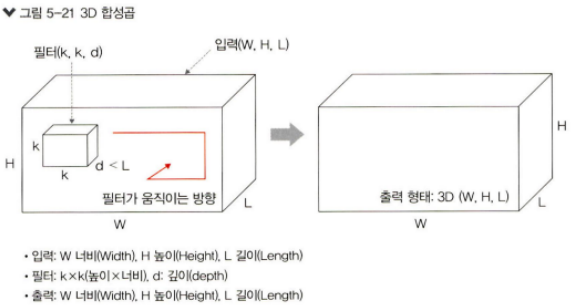


2D 합성곱: 방향 두 개로 움직이는 형태

입력(W, H)과 필터(k, k)에 대한 출력(W, H)

출력 형태는 2D 행렬

3D 합성곱



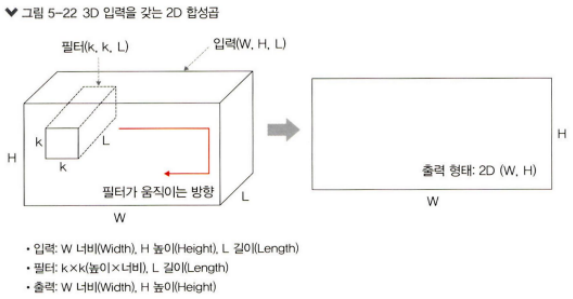
3D 합성곱: 필터가 움직이는 방향이 3개

입력(W, H, L)에 대해 필터(k, k, d)를 적용하면 출력(W, H, L)을 갖는 형태

출력은 3D 형태이며, d < L을 유지하는것이 중요

3D 입력을 갖는 2D 합성곱

3D 입력을 갖는 2D 합성곱: 입력이 3D 형태임에도 출력 형태가 3D가 아닌 2D 행렬을 취하는 것



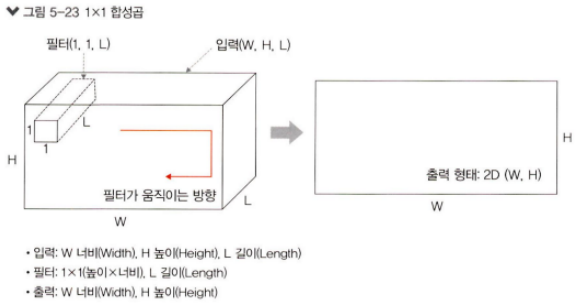
필터에 대한 길이가 입력 채널의 길이와 같으면 이와 같은 합성곱 형태가 만들어짐

입력(W, H, L)에 필터 (k, k, L)를 적용하면 출력은 (W, H)

필터는 두 방향으로 움직이며 출력 형태는 2D 행렬

3D 입력을 갖는 2D 합성곱의 대표적 사례로는 LeNet-5와 VGG가 있음

1 × 1 합성곱



1 × 1 합성곱은 3D 형태로 입력

입력(W, H, L)에 필터(1, 1, L)를 적용하면 출력은 (W, H)가 됨

1 × 1 합성곱에서 채널 수를 조정해서 연산량이 감소

대표적 사례) GoogLeNet

5.2 합성곱 신경망 맛보기

fashion_mnist 데이터셋

fashion_mnist 데이터셋: 토치비전에 내장된 예제 데이터로 운동화, 셔츠, 샌들 같은 작은 이미지의 모음

기본 mnist 데이터셋처럼 열 가지로 분류될 수 있는 28*28 픽셀의 이미지 7만개로 구성

훈련 데이터셋 : 0~255 사이의 값을 갖는 28*28 크기의 넘파이 배열

테스트 데이터셋 : 0~9까지 정수 값을 갖는 배열

GPU 사용

일반적으로 하나의 GPU를 사용할 때는 다음과 같은 코드를 이용

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model = Net()
model.to(device)
```

하지만 사용하는 PC에서 다수의 GPU를 사용한다면 다음 코드와 같이 nn.DataParallel을 사용

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = Net()
if torch.cuda.device_count() > 1:
    model = nn.DataParallel(net)
model.to(device)
```

nn.DataParallel을 사용할 경우 배치 크기가 알아서 각 GPU로 분배된다. 따라서 GPU 수만큼 배치 크기도 늘려 주어야 한다.

torch. utils.data.DataLoader(train_dataset, batch_size=100)

torch.utils.data.DataLoader() : 원하는 크기의 배치 단위로 데이터를 불러오거나, 순서가 무작위로 섞이도록 할 수 있음

train_dataset : 데이터를 불러올 데이터셋을 지정

batch_size : 데이터를 배치로 묶어 줌

np.random : 무작위로 데이터를 생성할 때 사용

np.random.randint() : 이산형 분포를 갖는 데이터에서 무작위 표본을 추출할 때 사용

np.random.rand() : 0~1 사이의 정규표준분포 난수를 출력할 때 사용

객체

파이토치의 근간은 C++이기 때문에 C++에서 사용하는 객체 지향 프로그램의 특징들을 파이토치에서도 사용

객제 지향 프로그래밍 : 프로그래밍에서 필요한 데이터를 추상화하여 속성이나 행동, 동작, 특징 등을 객체로 만들고, 그 객체들이 서로 유기적으로 동작하도록하는 프로그래밍 방법

객체 : 메모리를 할당받아 프로그램에서 사용되는 모든 데이터를 의미함 (변수, 함수 모두 객체임)

클래스와 함수

함수: 하나의 특정 작업을 수행하기 위해 독립적으로 설계된 프로그램 코드

BUT 함수를 호출하면 특정 작업만 수행하고, 그 결과값을 계속 사용할려면 따로 그 값을 저장해야함

클래스: 함수 뿐만 아니라 관련되니 변수까지도 한꺼번에 묶어서 관리하고 재사용 가능

코드 5-6 심층 신경망 모델 생성

```
class FashionDNN(nn.Module):
    def __init__(self): ------①
        super(FashionDNN, self).__init__()
        self.fc1 = nn.Linear(in_features=784, out_features=256) ------②
        self.drop = nn.Dropout(0.25) ------③
        self.fc2 = nn.Linear(in_features=256, out_features=128)
        self.fc3 = nn.Linear(in_features=128, out_features=10)

    def forward(self, input_data): ------④
        out = input_data.view(-1, 784) ------⑤
        out = F.relu(self.fc1(out)) ------⑥
        out = self.drop(out)
        out = F.relu(self.fc2(out))
        out = self.fc3(out)
        return out
```

클래스 형태의 모델은 항상 torch.nn.Module을 상속받음

__ init __() : 객체가 갖는 속성 값을 초기화하는 역할

super(FashionDNN, self).__init__() : FashionDNN이라는 부모 클래스를 상속받는다는 의미

nn : 딥러닝 모델 구성에 필요한 모듈이 모여 있는 패키지

torch.nn.Dropout(p) : p만큼의 비율로 텐서의 값이 0이 되고, 0이 되지 않는 값들은 기존 값에 (1/(1-p)) 만큼 곱해져 커짐

forward() : 모델이 학습 데이터를 입력받아 순전파 학습을 진행시키는 함수

view : 넘파이의 reshape과 같은 역할로 텐서의 크기를 변경해주는 역할

활성화 함수를 지정할 때는 두가지 방법이 가능함

1. F.relu() : forward() 함수에서 정의
2. nn.ReLU() : __ init __() 함수에서 정의

❤ 표 5-1 nn.xx와 nn.functional.xx의 사용 방법 비교

구분	nn.xx	nn.functional.xx
형태	nn.Conv2d: 클래스 nn.Module 클래스를 상속받아 사용	nn.functional.conv2d: 함수 def function (input)으로 정의된 순수한 함수
호출 방법	먼저 하이퍼파라미터를 전달한 후 함수 호출을 통해 데이터 전달	함수를 호출할 때 하이퍼파라미터, 데이터 전달
위치	nn.Sequential 내에 위치	nn.Sequential에 위치할 수 없음
파라미터	파라미터를 새로 정의할 필요 없음	가중치를 수동으로 전달해야 할 때마다 자체 가중치를 정의

모델이 데이터를 처리하기 위해서는 모델과 데이터가 동일한 장치에 있어야 함 CPU에서 처리된 데이터를 GPU 모델에 적용하거나 그 반대의 경우 런타임 오류가 발생

Autograd : 자동 미분을 수행하는 파이토치의 핵심 패키지로 Varible을 사용해서 역전파를 위한 미분 값을 자동으로 계산

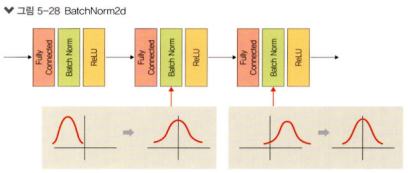
자동 미분에 대한 값을 저장하기 위해 테이프(순전파 단계에서 수행하는 모든 연산을 저장 후 역전파 단계에서 저장된 값을 사용)를 사용

코드 5-9 합성곱 네트워크 생성

```
class FashionCNN(nn.Module):
    def __init__(self):
        super(FashionCNN, self).__init__()
        self.layer1 = nn.Sequential( ····· ①
            nn.Conv2d(in_channels=1, out_channels=32, kernel_size=3, padding=1), ····· ②
            nn.BatchNorm2d(32), ····· ③
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2) ····· ④
        )
        self.layer2 = nn.Sequential(
            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(2)
        )
        self.fc1 = nn.Linear(in_features=64*6*6, out_features=600) ····· ⑤
        self.drop = nn.Dropout2d(0.25)
        self.fc2 = nn.Linear(in_features=600, out_features=120)
        self.fc3 = nn.Linear(in_features=120, out_features=10) ·····
                                        마지막 계층의 out_features는 클래스 개수를 의미
    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = out.view(out.size(0), -1) ····· ⑥
        out = self.fc1(out)
        out = self.drop(out)
        out = self.fc2(out)
        out = self.fc3(out)
        return out
```

nn.Sequential : 여러 개의 계층을 하나의 컨테이너에 구현하는 방법

BatchNorm2d : 학습 과정에서 각 배치 단위별로 데이터가 다양한 분포를 가지더라도 평균과 분산을 이용하여 정규화하는 것을 의미



클래스를 분류하기 위해서는 이미지 형태→ 배열 형태 로 변환해야 함

하지만 Conv2d에서 사용하는 하이퍼파라미터 값들의 따라 출력 사이즈가 달라짐

nn.Linear(in_features=64*6*6, out_features=600)

in_features : 입력 데이터의 크기를 의미하며 출력 크기를 계산해서 구해야 함

Conv2d 계층에서 출력 크기 구하는 공식

출력 크기 = (W-F+2P)/S+1

W: 입력 데이터의 크기

F: 커널 크기

P: 패딩 크기

S: 스트라이드

MaxPool2d 계층에서의 출력 크기 구하는 공식

출력 크기 = IF/F

IF : 입력 필터의 크기

F: 커널 크기

심층 신경망과 합성곱 신경망의 정확도를 비교해보면 합성곱 신경망의 정확도가 더 높다. 이미지 데이터가 많아지면 심층 신경망으로는 정확한 특성 추출 및 분류가 불가능하므로 합성곱 신경망을 생성하여 학습하는 것이 좋다