3️⃣

합성곱 신경망I- Part 1

날짜	@2024년 10월 2일 → 2024년 10월 7일
범위	5.1~5.2장
상태	완료
주차	3주차

5장 합성곱 신경망I

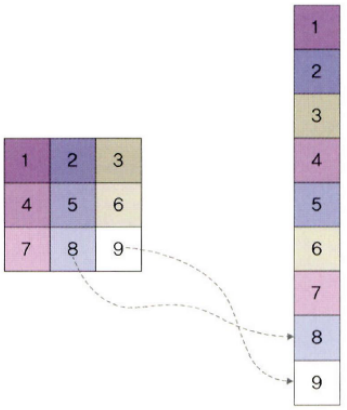
5.1 합성곱 신경망

- 역전파 : 순전파 과정에 따라 계산된 오차 정보가 신경망의 모든 노드(출력층→은닉층→입력층)로 전송됨. 이러한 계산 과정은 복잡하고 많은 자원(CPU, GPU, 메모리) 요구. 계산 시간도 오래 걸림.
- 이 문제를 해결하고자 하는 것이 합성곱 신경망. → 이미지 전체를 한 번에 계산하는 것이 아닌 이미지의 국소적 부분을 계산함으로써 시간과 자원을 절약하여 이미지의 세밀한 부분까지 분석할 수 있는 신경망.

5.1.1 합성곱층의 필요성

- 합성곱 신경망 - 이미지나 영상 처리에 유용함.
    - 예) 3x3 흑백(그레이스케일) 이미지가 있다고 가정해보자.

그림 5-1 합성곱층 원리

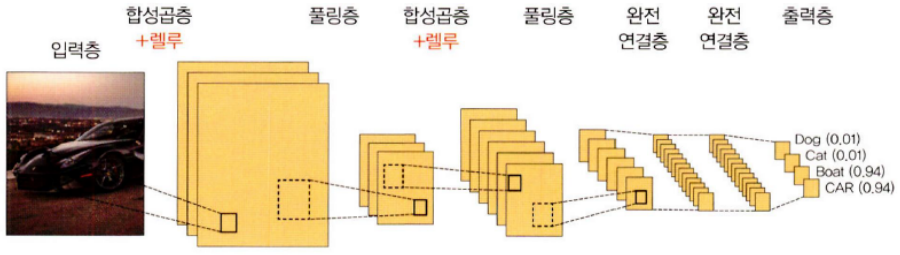


- 이미지 분석은 다음 그림의 왼쪽 같은 3x3 배열을 오른쪽과 같은 **펼쳐서 (flattening)** 각 픽셀에 가중치를 곱하여 은닉층으로 전달하게 됨.
- 하지만 그림에서 보이는 것처럼 이미지를 펼쳐서 분석하면 **데이터의 공간적 구조를 무시**하게 됨. → 이를 방지하려고 도입된 것이 합성곱층임.

5.1.2 합성곱 신경망 구조

- 합성곱 신경망(Convolutional Neural Network, CNN 또는 ConvNet)
  - 음성 인식이나 이미지/영상 인식에서 주로 사용되는 신경망.
  - 다차원 배열 데이터를 처리하도록 구성되어 컬러 이미지 같은 다차원 배열 처리에 특화되어 있음.
  - 계층 다섯 개로 구성됨.
    1. 입력층
    2. 합성곱층
    3. 풀링층
    4. 완전연결층
    5. 출력층

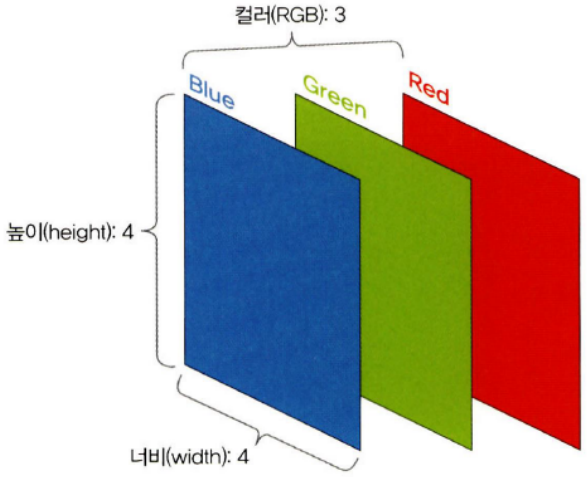
❯ 그림 5-2 합성곱 신경망 구조



합성곱층과 풀링층을 거치면서 입력 이미지의 주요 특성 벡터(feature vector)를 추출함. 그 후 추출된 주요 특성 벡터들은 완전연결층을 거치면서 1차원 벡터로 변화됨. 마지막으로 출력층에서 활성화 함수인 소프트맥스(softmax) 함수를 사용하여 최종 결과가 출력됨.

- 입력층(input layer) : 입력 이미지 데이터가 최초로 거치게 되는 계층.
  - 이미지 - 1차원 아닌 높이, 너비, 채널의 값을 갖는 3차원 데이터. 이때 채널을 이미지가 그레이스케일(gray scale)이면 1 값을 가지며, 컬러(RGB)이면 3 값을 가짐.

❯ 그림 5-3 채널



높이 4, 너비 4, 채널은 RGB를 갖고 있으므로, 3 → shape : (4, 4, 3)

- 합성곱층(convolutional layer) : 입력 데이터에서 특성을 추출하는 역할 수행함.
  - 특성 추출 과정 : 입력 이미지가 들어왔을 때 이미지에 대한 특성을 감지하기 위해 커널(kernel)이나 필터를 사용함. 커널/필터는 이미지의 모든 영역을 훑으면서 특성을 추출하게 되는데, 이렇게 추출된 결과물이 특성 맵(feature map).
  - 커널은 3x3, 5×5 크기로 적용되는 것이 일반적임. 스트라이드(stride)라는 지정된 간격에 따라 순차적으로 이동함.
  - 스트라이드가 1일 때 이동하는 과정

    1. 입력 이미지에 3x3 필터 적용

       입력 이미지와 필터를 포개 놓고 대응되는 숫자끼리 곱한 후 모두 더함.

$$(1\times1) + (0\times0) + (0\times1) + (0\times0) + (1\times1) + (0\times0) + (0\times1) + (0\times0) + (1\times1) = 3$$

❯ 그림 5-4 입력 이미지에 3×3 필터 적용

    2. 필터가 1만큼 이동

$$(0\times1) + (0\times0) + (0\times1) + (1\times0) + (0\times1) + (0\times0) + (0\times1) + (1\times0) + (1\times1) = 1$$

❯ 그림 5-5 입력 이미지에 필터가 1만큼 이동

3. 필터가 1만큼 두 번째 이동

$$(0\times1) + (0\times0) + (0\times1) + (0\times0) + (0\times1) + (1\times0) + (1\times1) + (1\times0) + (0\times1) = 1$$

❯ 그림 5-6 입력 이미지에 필터가 1만큼 두 번째 이동

4. 필터가 1만큼 세 번째 이동

$$(0\times1) + (0\times0) + (1\times1) + (0\times0) + (1\times1) + (0\times0) + (1\times1) + (0\times0) + (0\times1) = 3$$

❯ 그림 5-7 입력 이미지에 필터가 1만큼 세 번째 이동

5. 필터가 1만큼 네 번째 이동

$$(0\times1) + (1\times0) + (0\times1) + (0\times0) + (0\times1) + (1\times0) + (1\times1) + (0\times0) + (0\times1) = 1$$

❯ 그림 5-8 입력 이미지에 필터가 1만큼 네 번째 이동

6. 필터가 1만큼 마지막 이동

$$(0\times1) + (1\times0) + (0\times1) + (0\times0) + (1\times1) + (0\times0) + (0\times1) + (1\times0) + (0\times1) = 1$$

❯ 그림 5-9 입력 이미지에 필터가 1만큼 마지막으로 이동

이미지 크기 (6, 6, 1)이며, 3x3 크기의 커널/필터가 스트라이드 1 간격으로 이동하면서 합성곱 연산을 수행하는 것을 보여줌. 커널은 스트라이드 간격만큼 순회하면서 모든 입력 값과의 합성곱 연산으로 새로운 특성 맵을 만듦. 앞의 그림과 같이 커널과 스트라이드의 상호 작용으로 **(6, 6, 1) 크기가 (4, 4, 1) 크기의 특성 맵으로 줄어듦.**

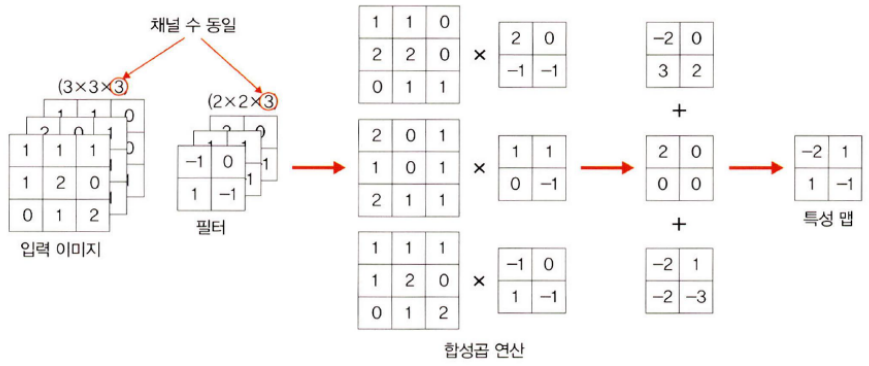
- 컬러 이미지의 합성곱
  - 앞선 그레이스케일 이미지와 구분되는 특징

    1. 필터 채널이 3이라는 것.

    2. RGB 각각에 서로 다른 가중치로 합성곱을 적용한 후 결과를 더해 준다는 것.

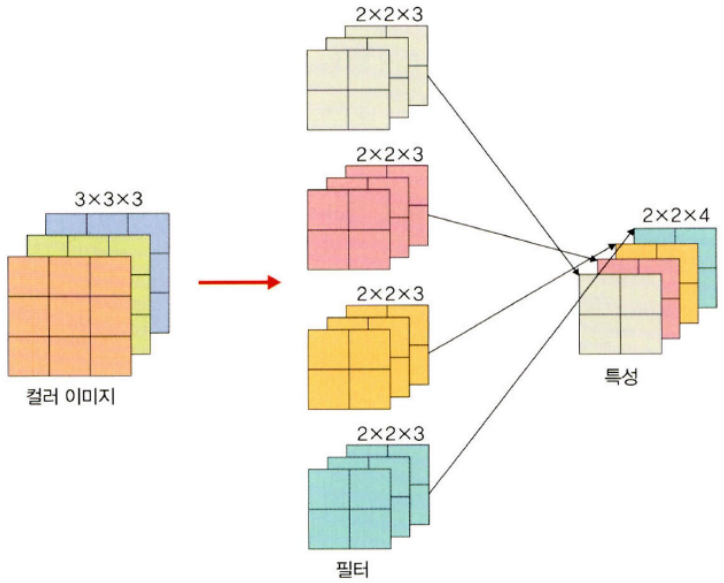
    그 외 스트라이드 및 연산 방법은 동일함. 필터 채널이 3이라고 해서 필터 개수도 3개라고 오해하기 쉬움, 실제로는 필터 개수 1개라는 점 주의!

❯ 그림 5-10 컬러 이미지 합성곱



- 필터가 2개 이상인 합성곱
  - 필터 각각은 특성 추출 결과의 채널이 됨. 각 계산은 앞서 진행했던 방법과 동일함.

❯ 그림 5-11 필터가 2 이상인 합성곱



- 합성곱층 요약
  - 입력 데이터 : W_1 x H_1 x D_1 (W_1 : 가로, xH_1 : 세로, xD_1 : 채널 또는 깊이)
  - 하이퍼파라미터
    - 필터 개수 : K
    - 필터 크기 : F
    - 스트라이드 : S
    - 패딩 : P
  - 출력 데이터

    - $W_2 = (W_1 - F + 2P)/S + 1$
    - $H_2 = (H_1 - F + 2P)/S + 1$
    - $D_2 = K$

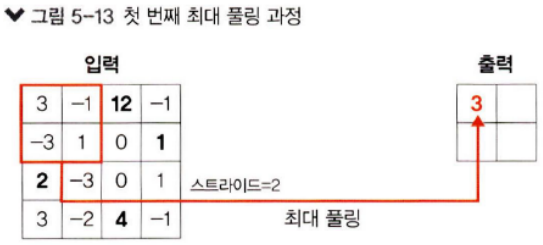
- 풀링층(pooling layer) : 합성곱층과 유사하게 특성 맵의 차원을 다운 샘플링(이미지를 축소하는 것)하여 연산량을 감소시키고, 주요한 특성 벡터를 추출하여 학습을 효과적으로 할 수 있게 함.

  - 최대 풀링(max pooling) : 대상 영역에서 최댓값을 추출
  - 평균 풀링(average pooling) : 대상 영역에서 평균을 반환

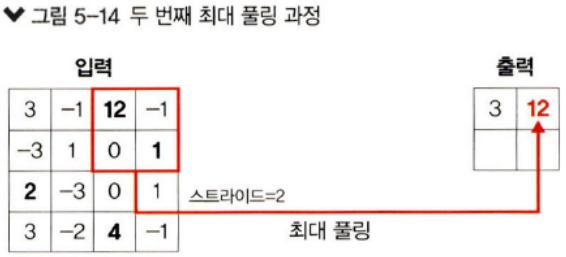
  대부분의 합성곱 신경망에서는 최대 풀링이 사용되는데, 평균 풀링은 각 커널 값을 평균화시켜 중요한 가중치를 갖는 값의 특성이 희미해질 수 있기 때문.

  - 최대 풀링의 연산 과정

    1. 첫 번째 풀링 과정 : 3, -1, 3, -1 값 중에서 최댓값(3) 선택함.



    2. 두 번째 풀링 과정 : 12, -1, 0, 1 값 중에서 최댓값(12) 선택함.



    3. 세 번째 풀링 과정 : 2, -3, 3, -2 값 중에서 최댓값(3) 선택함.

❯ 그림 5-15 세 번째 최대 풀링 과정

4. 네 번째 풀링 과정 : 0, 1, 4, -1 값 중에서 최댓값(4) 선택함.

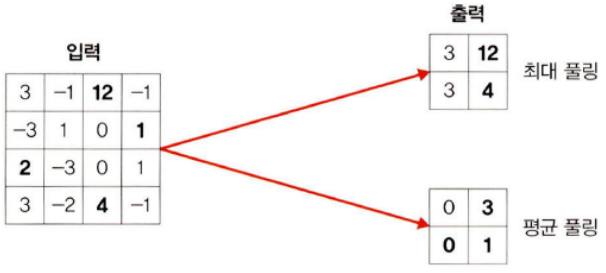
❯ 그림 5-16 네 번째 최대 풀링 과정

- 평균 풀링의 계산 과정 : 각 필터의 평균으로 계산함.

$$0=(3+(-1)+(-3)+1)/4$$
$$3=(12+(-1)+0+1)/4$$
$$0=(2+(-3)+3+(-2))/4$$
$$1=(0+1+4+(-1))/4$$

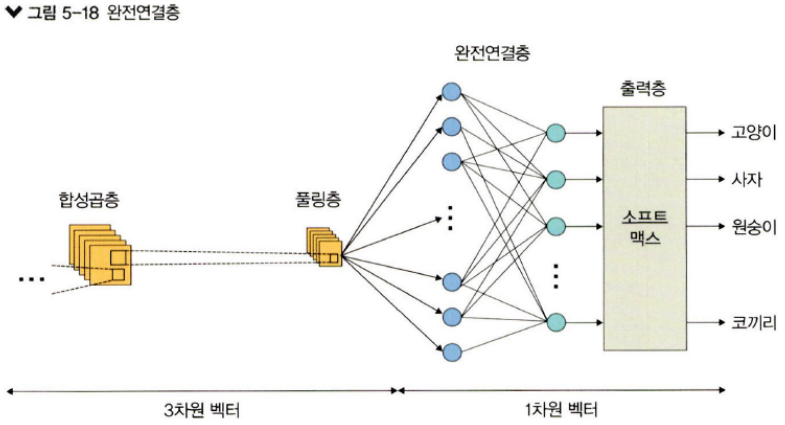
- 최대 풀링과 평균 풀링 결과 비교

❯ 그림 5-17 최대 풀링과 평균 풀링 비교



- 최대 풀링과 평균 풀링 요약
  - 입력 데이터 : W_1 x H_1 x D_1
  - 하이퍼파라미터

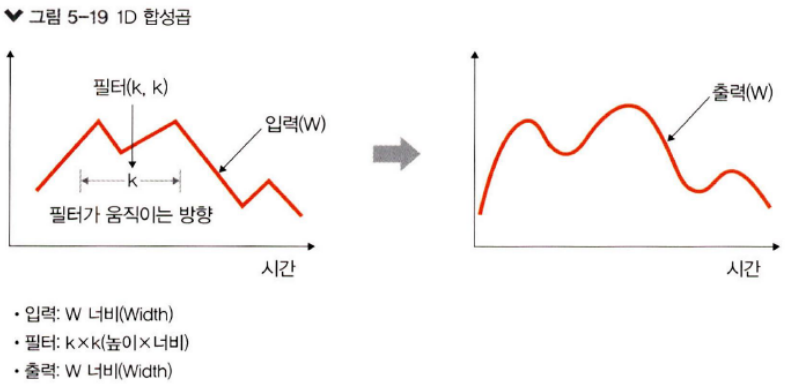
        - 필터 크기 : F
        - 스트라이드 : S
    - 출력 데이터
        - $W_2 = (W_1 - F)/S + 1$
        - $H_2 = (H_1 - F)/S + 1$
        - $D_2 = D_1$
- 완전연결층(fully connected layer) : 이미지 3차원 벡터에서 1차원 벡터로 펼쳐지게 됨.



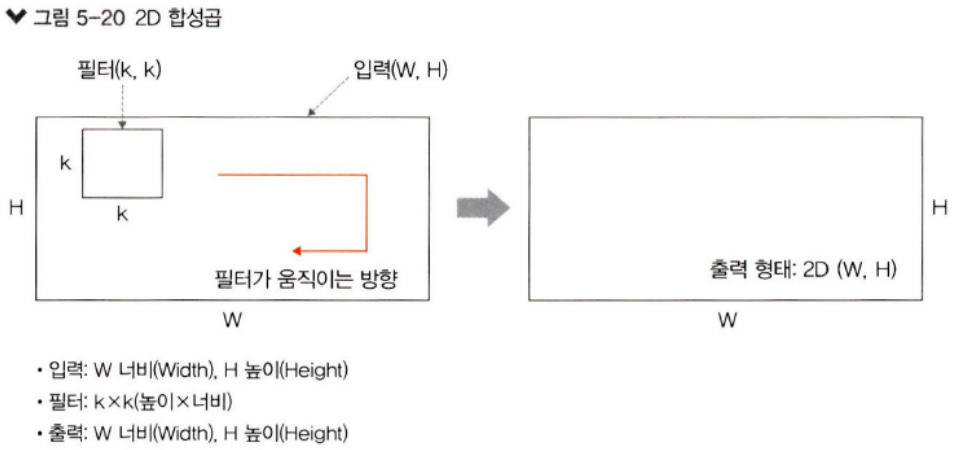
- 출력층(output layer) : 소프트맥스 활성화 함수 사용됨. 입력받은 값을 0~1 사이의 값으로 출력함.
    - 마지막 출력층의 소프트맥스 함수를 사용하여 이미지가 각 레이블(label)에 속할 확률 값이 출력됨. 이때 가장 높은 확률 값을 갖는 레이블이 최종 값으로 선정됨.

5.1.3 1D, 2D, 3D 합성곱

- 이동하는 방향 수와 출력 형태에 따라 1D, 2D, 3D로 분류할 수 있음.
- **1D 합성곱**
    - 필터가 시간을 축으로 좌우로만 이동할 수 있는 합성곱.
    - 입력(W)과 필터(k)에 대한 출력은 W가 됨.
    - 입력이 [1, 1, 1, 1, 1]이고, 필터가 [0.25, 0.5, 0.25]라면, 출력은 [1, 1, 1]
    - 즉, 다음 그림과 같이 출력 형태는 1D 배열이 되며, 그래프 곡선을 완화할 때 많이 사용됨.

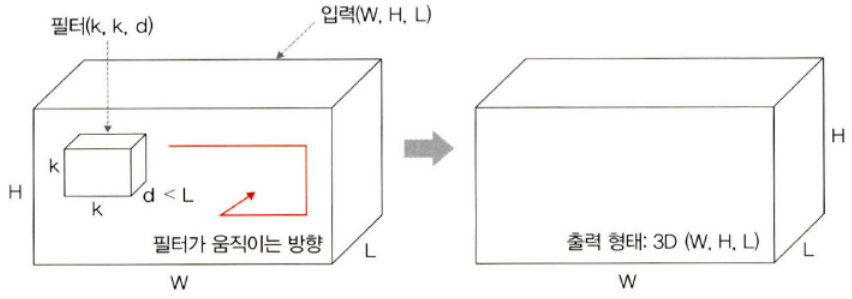


- **2D 합성곱**
  - 필터가 방향 2개로 움직이는 형태.
  - 입력(W, H)과 필터(k, k)에 대한 출력은 (W, H)가 되며, 출력 형태는 2D 행렬.



- **3D 합성곱**
  - 필터가 움직이는 방향이 3개.
  - 입력(W, H, L)에 대해 필터(k, k, d)를 적용하면 출력으로 (W, H, L)을 갖는 형태.
  - 출력은 3D 형태, 이때 $d < L$유지하는 것이 중요함.

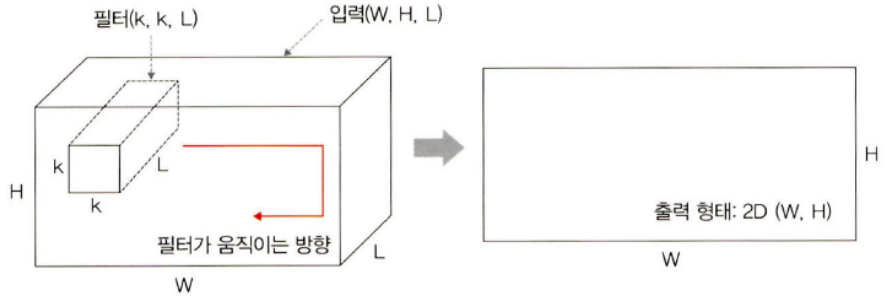
❯ 그림 5-21 3D 합성곱



- 입력: W 너비(Width), H 높이(Height), L 길이(Length)
- 필터: k×k(높이×너비), d: 깊이(depth)
- 출력: W 너비(Width), H 높이(Height), L 길이(Length)

- **3D 입력을 갖는 2D 합성곱**
  - 입력이 (224x224×3, 112×112×32)와 같은 3D 형태임에도 출력 형태가 2D 행렬을 취하는 것.
  - 필터에 대한 길이(L)가 입력 채널의 길이(L)와 같아야 하기 때문에 만들어짐.
  - 즉, 입력(W, H, L)에 필터(k, k, L) 적용하면 출력은 (W, H)가 됨.
  - 필터는 두 방향으로 움직이며, 출력 형태는 2D 행렬.
  - 대표적 사례 : LeNet-5, VGG

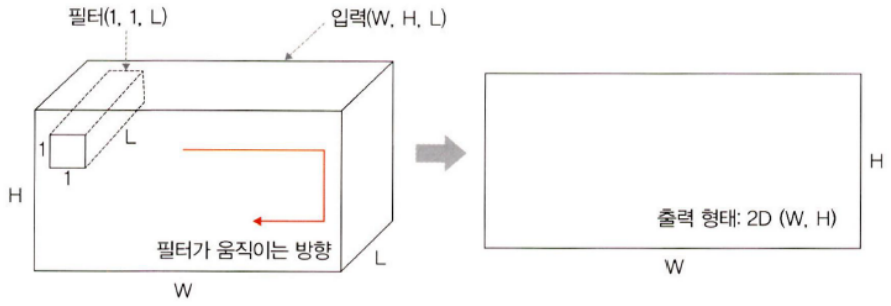
❯ 그림 5-22 3D 입력을 갖는 2D 합성곱



- 입력: W 너비(Width), H 높이(Height), L 길이(Length)
- 필터: k×k(높이×너비), L 길이(Length)
- 출력: W 너비(Width), H 높이(Height)

- **1 × 1 합성곱**
  - 3D 형태로 입력됨.
  - 입력(W, H, L)에 필터(1, 1, L) 적용하면 출력은 (W, H)가 됨.
  - 채널 수를 조정해서 연산량이 감소되는 효과가 있음.
  - 대표적 사례 : GoogLeNet

❤ 그림 5-23 1×1 합성곱



- 입력: W 너비(Width), H 높이(Height), L 길이(Length)
- 필터: 1×1(높이×너비), L 길이(Length)
- 출력: W 너비(Width), H 높이(Height)

5.2 합성곱 신경망 맛보기

- fashion_mnist 데이터셋을 사용하여 합성곱 신경망 직접 구현하기

💡 **fashion_mnist 데이터셋**

토치비전(torchvision)에 내장된 예제 데이터로 운동화, 셔츠, 샌들 같은 작은 이미지의 모음이며, 기본 MNIST 데이터셋처럼 열 가지로 분류될 수 있는 28 X 28 픽셀의 이미지 7만 개로 구성되어 있다.

데이터셋을 자세히 살펴보면 훈련 데이터(train_images)는 0에서 255 사이의 값을 갖는 28 X 28 크기의 넘파이(NumPy) 배열이고, 레이블(정답) 데이터(train_lables)는 0에서 9까지 정수 값을 갖는 배열이다.

0에서 9까지 정수 값은 이미지(운동화, 셔츠 등)의 클래스를 나타내는 레이블이다. 각 레이블과 클래스는 다음과 같다.

- `0 : T-Shirt`
- `1 : Trouser`
- `2 : Pullover`
- `3 : Dress`
- `4 : Coat`
- `5 : Sandal`
- `6 : Shirt`
- `7 : Sneaker`
- `8 : Bag`
- `9 : Ankle Boot`

💡 **GPU 사용**

일반적으로 하나의 GPU 사용할 때 사용하는 코드

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

사용하는 PC에서 다수의 GPU 사용한다면 `nn.DataParallell` 이용한다.

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

nn.DataParallel 사용할 경우 배치 크기(batch size)가 알아서 각 GPU로 분배되는 방으로 작동한다. 따라서 GPU 수만큼 배치 크기도 늘려 주어야 한다.

- fashion_mnist 데이터셋 내려받기
  - `torchvision.datasets`는 `torch.utils.data.Dataset`의 하위 클래스로 다양한 데이터셋(CIFAR, COCO, MNIST, ImageNet 등)을 포함함. torchvision.datasets에서 사용하는 주요한 파라미터는 다음과 같다.
    - `torchvision.datasets.FashionMNIST( file_path , download=True , transform=transforms.Compose([transforms.ToTensor()]) )`
      - 첫 번째 파라미터 : FashionMNIST를 내려받을 위치를 지정함.
      - `download` : `download`를 `True`로 변경해 주면 첫 번째 파라미터의 위치에 해당 데이터셋이 있는지 확인한 후 내려받음.
      - `transform` : 이미지를 텐서(0~1)로 변경함.
- fashion_mnist 데이터를 데이터로더에 전달하기
  - `torch.utils.data.DataLoader()`를 사용하여 원하는 크기의 배치 단위로 데이터를 불러오거나, 순서가 무작위로 섞이도록(shuffle) 할 수 있음. 데이터로더에서 사용하는 파라미터는 다음과 같다.
    - `torch.utils.data.DataLoader( train_dataset , batch_size=100 )`
      - 첫 번째 파라미터 : 데이터를 불러올 데이터셋을 지정함.

- `batch_size` : 데이터를 배치로 묶어 줌. 여기에서는 `batch_size=100`으로 지정했기 때문에 100개 단위로 데이터를 묶어서 불러옴.

- 분류에 사용될 클래스 정의

  1. np.random은 무작위로 데이터를 생성할 때 사용함. 또한, np.random.randint()는 이산형분포를 갖는 데이터에서 무작위 표본을 추출할 때 사용함. 따라서, random.randint(len(train_dataset)) 의미는 0~(train_dataset의 길이) 값을 갖는 분포에서 랜덤한 숫자 한 개를 생성하라는 의미.

  2. train_dataset을 이용한 3차원 배열을 생성함.

     - 배열에 대한 사용 예시

```
import numpy as np
exmap = np.arnage(0, 100, 3)  # 1~99의 숫자에서 3씩 건너뛴
exmap.resize(6,4) # 행렬의 크기를 6x4로 조정
exmap

exmap[3]  # 3행에 해당하는 모든 요소 출력
exmap[3,3]  # 3행 3번째 열에 대한 값 출력
exmap[3][3]  # 3행의 3번째 열에 대한 값을 출력(앞 결과에 동일)
```

- 심층 신경망 모델 생성하기

  1. 클래스(class) 형태의 모델은 항상 `torch.nn.Module` 상속받음. `__init__()`은 객체가 갖는 속성 값을 초기화하는 역할을 하며, 객체가 생성될 때 자동으로 호출됨. `super(FashionDNN, self).__init__()`은 FashionDNN이라는 부모(super) 클래스를 상속받겠다는 의미.

  2. `nn`은 딥러닝 모델(네트워크) 구성에 필요한 모듈이 모여 있는 패키지, `Linear`는 단순 선형 회귀 모델을 만들 때 사용함. 이때 사용하는 파라미터는 다음과 같다.

     `nn.Linear( in_features =784, out_features =256)`

     a. in_features : 입력의 크기(input size)

     b. out_features : 출력의 크기(output size)

     실제로 데이터 연산이 진행되는 `forward()` 부분에는 첫 번째 파라미터 값만 넘겨주게 되며, 두 번째 파라미터에서 정의된 크기가 `forward()` 연산의 결과가 됨.

  3. `torch.nn.Dropout(p)`는 p만큼의 비율로 텐서의 값이 0이 되고, 0이 되지 않는 값들은 기존 값에 (1/(1-p))만큼 곱해져 커짐. 예를 들어 p=0.3이라는 의미는 전체 값 중 0.3의 확률로 0이 된다는 것, 0이 되지 않는 0.7에 해당하는 값은 (1/(1-0.7))만큼 커짐.

4. `forward()` 함수는 모델이 학습 데이터를 입력받아서 순전파 학습을 진행시키며, 반드시 forward라는 이름의 함수여야 함. 객체를 데이터와 함께 호출하면 자동으로 실행됨. 이때 순전파 연산이란 $H(x)$ 식에 입력 x로부터 예측된 y를 얻는 것.
5. 파이토치에서 사용하는 뷰(view) - 넘파이의 reshape과 같은 역할. 텐서의 크기(shape)를 변경해 주는 역할. 따라서 `input_data.view(-1, 784)`는 input_data를 `(?, 784)`의 크기로 변경하라는 의미. 첫 번째 차원(-1)은 사용자가 잘 모르겠으니 파이토치에 맡기겠다는 의미, 두 번째 차원의 길이는 784를 가지도록 하라는 의미.
6. 활성화 함수 지정하는 2가지 방법
   a. `F.relu()` : `forward()` 함수에서 정의
   b. `nn.ReLU()` : `__init__()` 함수에서 정의

   이 둘 간의 차이는 간단히 사용하는 위치라고 할 수 있음. 하지만 근본적으로는 `nn.functional.xx()` (혹은 `F.xx()`)와 `nn.xx()`는 사용 방법에 차이가 있음.

❮ 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

- 심층 신경망에서 필요한 파라미터 정의하기
  1. 옵티마이저를 위한 경사 하강법은 Adam을 사용하며, 학습률을 의미하는 lr은 0.001을 사용한다는 의미.
- 심층 신경망을 이용한 모델 학습하기
  1. 1'. 일반적으로 배열이나 행렬과 같은 리스트(list)를 사용하는 방법은 다음과 같다.
     a. 1과 같이 비어 있는 배열이나 행렬을 만듦.
     b. 1'처럼 `append` 메서드를 이용하여 데이터를 하나씩 추가함.
  2. for 구문을 이용하여 레코드(행, 가로줄)를 하나씩 가져옴. 이때 `for x, y in train:` 과 같이 in 앞에 변수 두 개 지정해 주면 레코드에서 요소 두 개를 꺼내 오겠다는 의미.
  3. 모델이 데이터를 처리하기 위해서는 모델과 데이터가 동일한 장치(CPU, GPU)에 있어야 함. model.to(device)가 GPU 사용했다면, images.to(device), labels.to(device)도 GPU에서 처리되어야 함. 참고로 CPU에서 처리된 데이터를 GPU 모델에 적용하거나 그 반대의 경우 런타임 오류 발생함.

4. Autograd는 자동 미분을 수행하는 파이토치의 핵심 패키지로, 자동 미분에 대한 값을 저장하기 위해 테이프(tape)를 사용함. 순전파 단계에서 테이프는 수행하는 모든 연산을 저장함. 그리고 역전파 단계에서 저장된 값들을 꺼내서 사용함. 즉, Autograd는 Variable을 사용해서 역전파를 위한 미분 값을 자동으로 계산해 줌. 따라서 자동 미분을 계산하기 위해서는 `torch.autograd` 패키지 안에 있는 `Variable` 이용해야 동작함.
5. 분류 문제에 대한 정확도는 전체 예측에 대한 정확한 예측의 비율로 표현할 수 있음.

   `classificationi accuracy = correct predictions / total predictions`

   이때 결과에 100을 곱하여 백분율로 표시하는 코드

   `classification accuracy = correct predictions / total predictions * 100`

   또한, 분류 문제에 대한 정확도는 다음과 같이 값을 반전시켜 오분류율 또는 오류율로 표현할 수 있음.

   `error rate = (1 - (correct predictions / total predictions)) * 100`

   분류 문제에서 클래스가 3개 이상일 때 다음과 같은 사항에 주의하자.

   - 정확도가 80% 이상이었다고 하자. 하지만 80%라는 값이 모든 클래스가 동등하게 고려된 것인지, 특정 클래스의 분류가 높았던 것인지에 대해 알 수 없음에 유의해야 함.
   - 정확도가 90% 이상이었다고 하자. 하지만 100개의 데이터 중 90개가 하나의 클래스에 속할 경우 90%의 정확도는 높다고 할 수 없음. 즉, 모든 데이터를 특정 클래스에 속한다고 예측해도 90%의 예측 결과가 나오기 때문에 데이터 특성에 따라 정확도를 잘 관측해야 함.

- 합성곱 네트워크 생성하기

  1. `nn.Sequential`을 사용하면 `__init__()` 에서 사용할 네트워크 모델들을 정의해 줄 뿐만 아니라, forward() 함수에서 구현될 순전파를 계층(layer) 형태로 조금 더 가독성 뛰어난 코드로 작성할 수 있음. 즉, `nn.Sequential` 은 계층을 차례로 쌓을 수 있도록 $Wx + b$와 같은 수식과 활성화 함수를 연결해 주는 역할. 특히 데이터가 각 계층을 순차적으로 지나갈 때 사용하면 좋은 방법.
  2. 합성곱층(conv layer)은 합성곱 연산을 통해서 이미지의 특성을 추출함. 합성곱이란 커널(또는 필터)이라는 n x m 크기의 행렬이 높이 x 너비 크기의 이미지를 처음부터 끝까지 훑으면서 각 원소 값끼리 곱한 후 모두 더한 값을 출력함. 커널을 일반적으로 3x3이나 5x5를 사용하며 사용하는 파라미터는 다음과 같다.

     `nn.Conv2d( in_channels=1 , out_channels=32 , kernel_size=3 , padding=1 )`

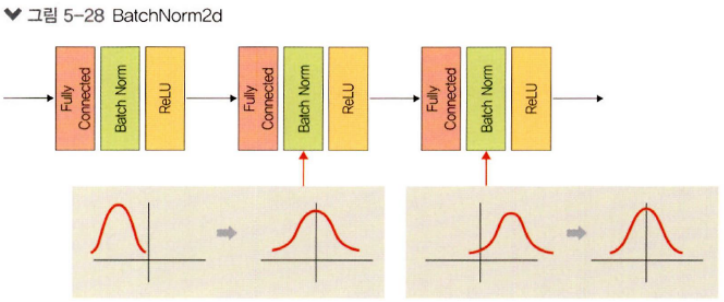
     a. `in_channels` : 입력 채널 수. 흑백 이미지는 1, RGB 값을 가진 이미지는 3을 가진 경우가 많음. (채널이란? 2차원은 행렬, 3차원으로 생각하면 채널은 결국 깊이(depth)를 의미함.)

     b. `out_channels` : 출력 채널의 수.

c. `kernel_size` : 커널 크기를 의미. 논문에 따라 필터라고도 함. 이미지 특징을 찾아내기 위한 공용 파라미터, CNN에서 학습 대상은 필터 파라미터가 됨. 입력 데이터를 스트라이드 간격으로 순회하면서 합성곱을 계산함.

d. `padding` : 패딩 크기를 의미. 출력 크기를 조정하기 위해 입력 데이터 주위에 0을 채움. 패딩 값이 클수록 출력 크기도 커짐.

3. `BatchNorm2d`는 학습 과정에서 각 배치 단위별로 데이터가 다양한 분포를 가지더라도 평균과 분산을 이용하여 정규화하는 것. 다음 그림을 보면 배치 단위나 계층에 따라 입력 값의 분포가 모두 다르지만 정규화를 통해 분포를 가우시안 형태로 만듦. 그러면 평균은 0, 표준편차는 1로 데이터의 분포가 조정됨.



4. MaxPool2d는 이미지 크기를 축소시키는 용도로 사용함. 풀링 계층은 합성곱층의 출력 데이터를 입력으로 받아서 출력 데이터(activation map)의 크기를 줄이거나 특정 데이터를 강조하는 용도. 풀링 계층을 처리하는 방법으로는 최대 풀링과 평균 풀링, 최소 풀링이 있으며, 이때 사용하는 파라미터는 다음과 같다.

```
nn.MaxPool2d( kernel_size=2 , stride=2 )
```

a. `kernel_size` : m x n 행렬로 구성된 가중치

b. `stride` : 입력 데이터에 커널(필터) 적용할 때 이동할 간격. 스트라이드 값이 커지면 출력 크기는 작아짐.

5. 클래스를 분류하기 위해서는 이미지 형태의 데이터를 배열 형태로 변환하여 작업해야 함. 이때 Conv2d에서 사용하는 하이퍼파라미터 값들에 따라 출력 크기(output size)가 달라짐. 즉, 패딩과 스트라이드 값에 따라 출력 크기가 달라짐. 이렇게 줄어든 출력 크기는 최종적으로 분류를 담당하는 완전연결층으로 전달됨.

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

a. `in_features` : 입력 데이터의 크기. 중요한 것은 이전까지 수행했던 Conv2d, MaxPool2d는 이미지 데이터를 입력으로 받아 처리. 하지만 그 출력 결과를 완전연결층으로 보내기 위해서는 1차원으로 변경해주어야 함.

- Conv2d 계층에서의 출력 크기 구하는 공식
  - 출력 크기 = $(W - F + 2P)/S + 1$

        - W : 입력 데이터의 크기(input_volume_size)
        - F : 커널 크기(kernel_size)
        - P : 패딩 크기(padding_size)
        - S : 스트라이드(strides)
    - MaxPool2d 계층에서의 출력 크기 구하는 공식
        - 출력 크기 = IF/F
            - IF : 입력 필터의 크기(input_filter_size, 또한 바로 앞의 Conv2d의 출력 크기이기도 함.)
            - F : 커널 크기(kernel_size)

b. `out_features` : 출력 데이터의 크기.

6. 합성곱층에서 완전연결층으로 변경되기 때문에 데이터 형태를 1차원으로 바꾸어 줌.

   이때 `out.size(0)`은 결국 100을 의미. 따라서 (100, ?) 크기의 텐서로 변경하겠다는 의미.

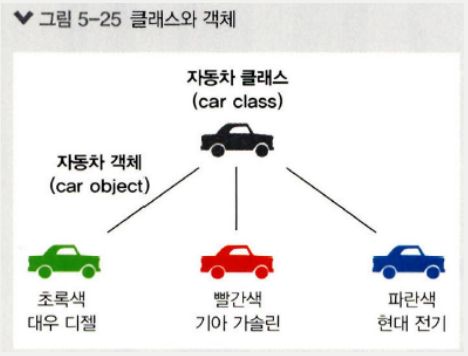
객체

파이토치의 근간 C++. 따라서 C++에서 사용하는 객체 지향 프로그램의 특징들을 파이토치에서도 사용하게 되는데, 대표적인 것이 객체라는 개념.

객체 지향 프로그래밍(object oriented programming)은 프로그래밍에서 필요한 데이터를 추상화하여 속성이나 행동, 동작 특징 등을 객체로 만들고, 그 객체들이 서로 유기적으로 동작하도록 하는 프로그래밍 방법.

좀 더 쉽게 표현하면 클래스하는 붕어빵 틀에서 여러 개의 객체라는 붕어빵을 찍어 내는 것과 같다. 즉, 재사용성의 이유로 객체 지향 프로그래밍을 많이 사용하고 있음.

이때 객체란 메모리를 할당받아 프로그램에서 사용되는 모든 데이터를 의미하기 때문에, 변수, 함수 등은 모두 객체라고 할 수 있음. `객체명 = 클래스명()`



💡 **클래스와 함수**

함수(function)란 하나의 특정 작업을 수행하기 위해 독립적으로 설계된 프로그램 코드

함수의 호출은 특정 작업만 수행할 뿐 그 결괏값을 계속 사용하기 위해서는 반드시 어딘
에 따로 그 값을 저장해야만 함. 즉, 함수를 포함한 프로그램 코드의 일부를 재사용하기
해서는 해당 함수뿐만 아니라 데이터가 저장되는 변수까지도 한꺼번에 관리해야 함.

이처럼 함수뿐만 아니라 관련된 변수까지도 한꺼번에 묶어서 관리하고 재사용할 수 있거
주는 것이 클래스(class).

<클래스와 함수의 차이>

```
def add(num1, num2): # 함수 정의(num1, num2 받아서 더해 주는 함
    result = num1 + num2
    return reslt

print(add(1, 2))
print(add(2, 3))

class Calc:
    def __init__(self):  # 객체를 생성할 때 호출하면 실행되는 초기
        self.result = 0

    def add(self, num1, num2):
        self.result = num1 + num2
        return self.result

obj1 = Calc()
obj2 = Calc()

print(obj1.add(1, 2))
print(obj1.add(2, 3))
print(obj2.add(2, 2))
print(obj2.add(2, 3))

# 두 개의 객체는 독립적으로 연산됨.
# 개별적 함수로 구현했다면 복잡했을 코드가 클래스 사용으로 간결해짐.
```