컴퓨터비전

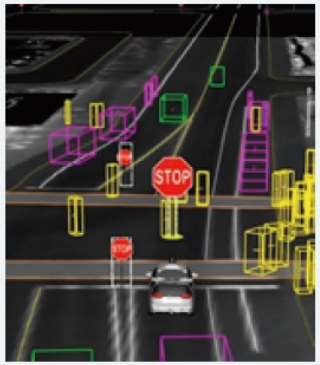
Computer Vision

컨볼루션 신경망과 컴퓨터 비전

본 강의자료는 한빛아카데미 〈파이썬으로 만드는 인공지능〉을 내용을 기반으로 제작 및 보완되었음을 알립니다.

PREVIEW

- 컨볼루션 신경망(CNN, Convolutional neural network)은 딥러닝에서 가장 성공한 모델임
- 컨볼루션 신경망으로 제작한 인공지능 제품의 예



(a) 자율주행차

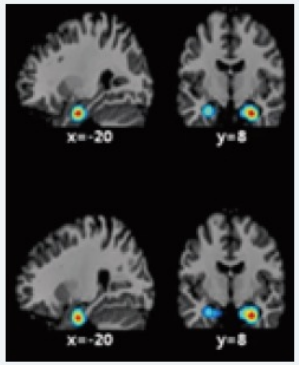
FSD: Fully Self-Driving

(b) 딥드림

Image generation

(c) 영상 주석달기

Image captioning

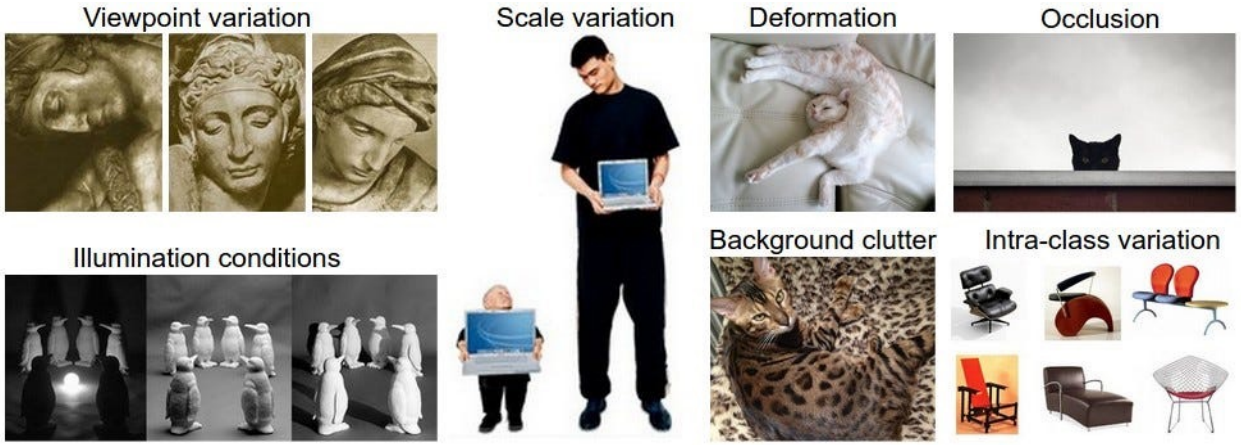


(d) 병변 부위 찾기

Abnormal detection

PREVIEW

- 인공지능의 중요한 분야 중 하나인 컴퓨터 비전은 컴퓨터에 시각을 부여하는 주제를 연구하며, 딥러닝 이전과 이후가 확연히 차이가 나는 분야임
- 가장 두드러진 차이는 “성능”이며, 딥러닝 이전의 컴퓨터 비전 기술로는 앞서 제시된 인공지능 제품의 성능을 내는 것은 불가능함
- 방법론적인 차이도 큼. 딥러닝 이전에는 사람이 일일이 알고리즘을 구상하고 코딩했으므로 “크기, 자세, 조명, 배경, 잡음” 등의 심한 변화에 대처하지 못했음
- 반면 딥러닝은 충분히 많은 영상 데이터가 있으면 자동으로 특징을 추출하고 인식 기능을 학습함으로써 높은 예측 성능을 보장함



6.2 컨볼루션 신경망의 구조와 동작

- **요약**
  - 컨볼루션 신경망의 핵심 연산인 컨볼루션의 원리와 특성을 소개하고 컨볼루션 신경망이 어떻게 복잡한 컴퓨터 비전 문제를 푸는지에 대해 설명함
  - 컨볼루션 신경망을 구성하는 빌딩 블록과 블딩 블록을 쌓아 신경망을 만드는 원리를 설명함

6.2 컨볼루션 신경망의 구조와 동작

6.2.1 컨볼루션 연산으로 특징 맵 추출

- 컨볼루션convolution은 신호에서 특징을 추출하거나 신호를 변환하는 데 사용하는 연산으로 신호 처리, 영상 처리, 컴퓨터 비전 등에 널리 사용됨
- 〈컨볼루션〉은 〈수용장 receptive field〉과 〈커널 kernel〉이 선형 결합 linear combination 인데, 선형 결합이란 해당하는 요소끼리 곱한 결과를 더하는 연산임
  - 예) 1차원 신호 [5]의 위치에서 선형결합 결과는 2*(-1)+9*0+9*1=7
- 컨볼루션 연산을 적용한 결과를 〈특징 맵 feature map〉 이라고 함
  - 단, [0]과 [7]의 위치에서는 커널의 특성상 특징맵을 구할 수 없음

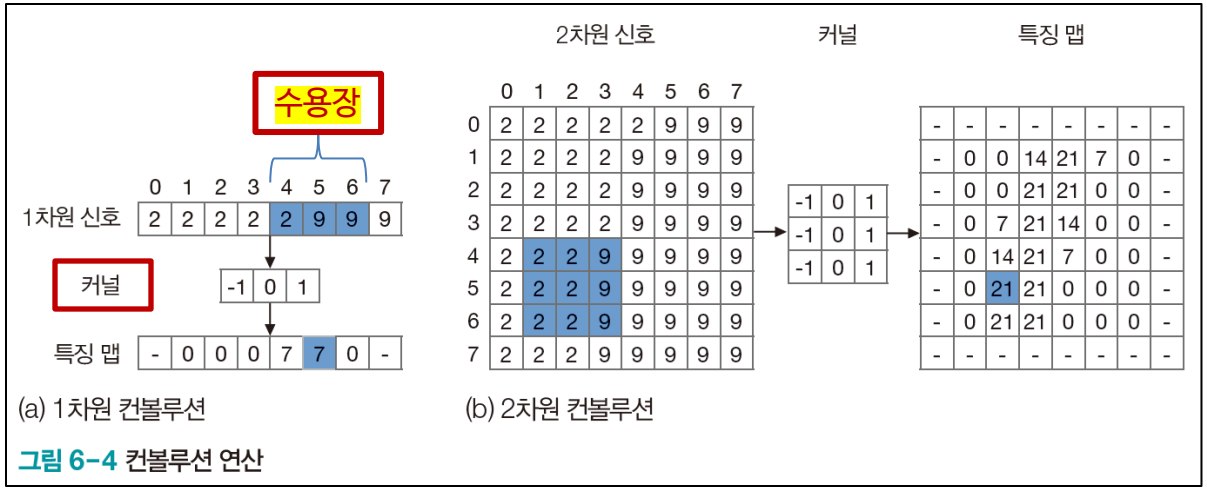


그림 6-4 컨볼루션 연산

6.2 컨볼루션 신경망의 구조와 동작

6.2.1 컨볼루션 연산으로 특징 맵 추출

- 식(6.1)은 1차원 컨볼루션을 식으로 쓴 것이며, z는 입력 신호이고 u는 커널이며, h는 커널의 크기임
  - 보통 커널 크기는 3, 5, 7, … 등 을 사용하는데, 홀 수를 사용해 대칭을 이룸
  - $$f(i) = z(i) \circledast u = \sum_{x=-(h-1)/2}^{(h-1)/2} z(i+x)u(x) \qquad (6.1)$$

- [그림 6-4(b)]는 2차원 컨볼루션을 설명하는 그림이며, 커널과 수용장도 2차원 구조를 가짐
  - 그림은 3x3커널을 예시하고, 위치 [5,2] 에서 계산하는 예를 보여줌
  - $$f(j,i) = z(j,i) \circledast u = \sum_{x=-(h-1)/2}^{(h-1)/2} \sum_{y=-(h-1)/2}^{(h-1)/2} z(j+y, i+x)u(y,x) \qquad (6.2)$$

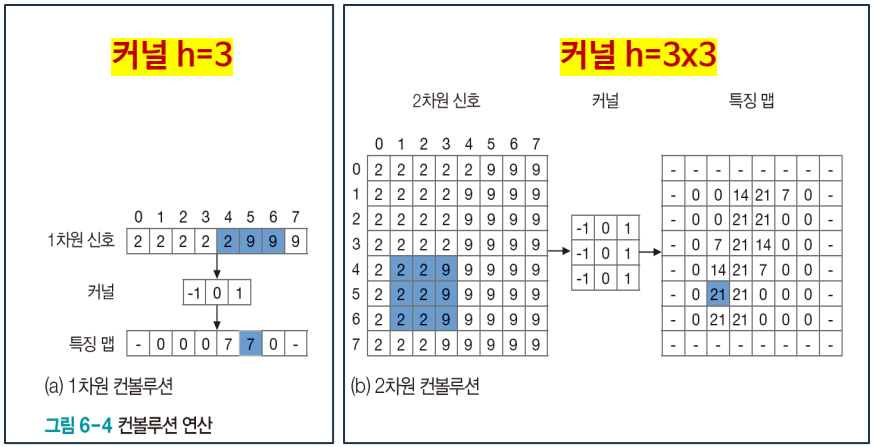
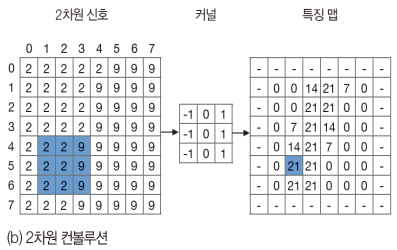


그림 6-4 컨볼루션 연산

6.2 컨볼루션 신경망의 구조와 동작



6.2.1 컨볼루션 연산으로 특징 맵 추출

- [그림6-4(b)]의 2차원 커널에서 오른쪽 열은 모두 1, 왼쪽 열은 모두 -1 이며, 이 커널은 수용장의 오른쪽에서 왼쪽의 밝기 값을 빼는 계산을 하는 셈임. *수용장 : 컨볼루션 연산이 이루어지는 데이터 영역
- 오른쪽이 밝고 왼쪽이 어두운 수용장은 양수가 되고, 반대라면 음수가 되며, 명암 변화가 없으면 수용장은 0이 됨. 즉, 해당 커널은 수직 에지(vertical edge)라는 특징을 추출함
- 연산 결과는 입력 영상과 크기가 같고 입력 영상의 특징을 담고 있기 때문에 〈특징 맵〉이라 부름

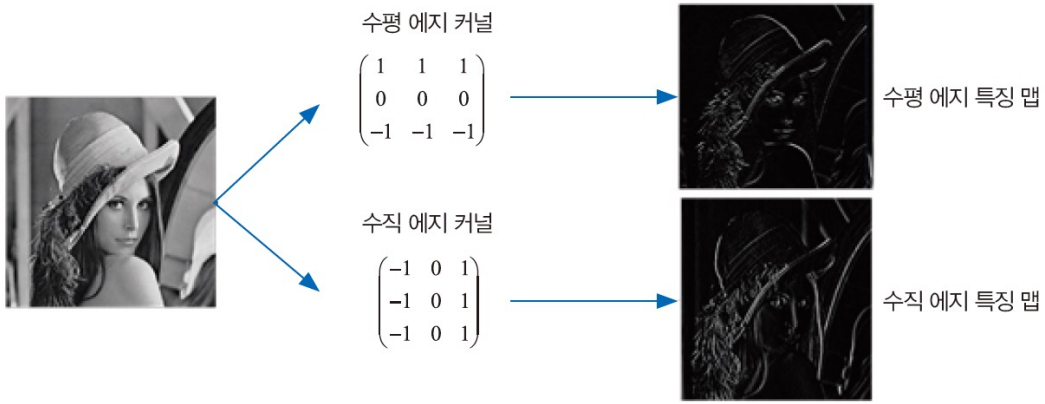


그림 6-5 에지 검출(수평 에지와 수직 에지 특징 맵)

6.2 컨볼루션 신경망의 구조와 동작

6.2.1 컨볼루션 연산으로 특징 맵 추출

- [그림 6-5]에서 보는 바와 같이 원본 영상에 컨볼루션을 적용하면 원본 영상과 형태와 크기가 같은 특징맵에 생성됨
- 이런 장점이 있는 컨볼루션을 신경망에 도입하면 입력 **데이터의 형태를 고스란히 유지한 채** 특징을 추출할 수 있게 됨
- 깊은 다층 퍼셉트론에 영상을 입력하려면 2차원 모양이 영상을 1차원으로 펼쳐야 하며, 화소는 주위 화소 8개와 이웃인데 1차원으로 펼치면 이웃 정보가 사라져서 정보 손실이 큼

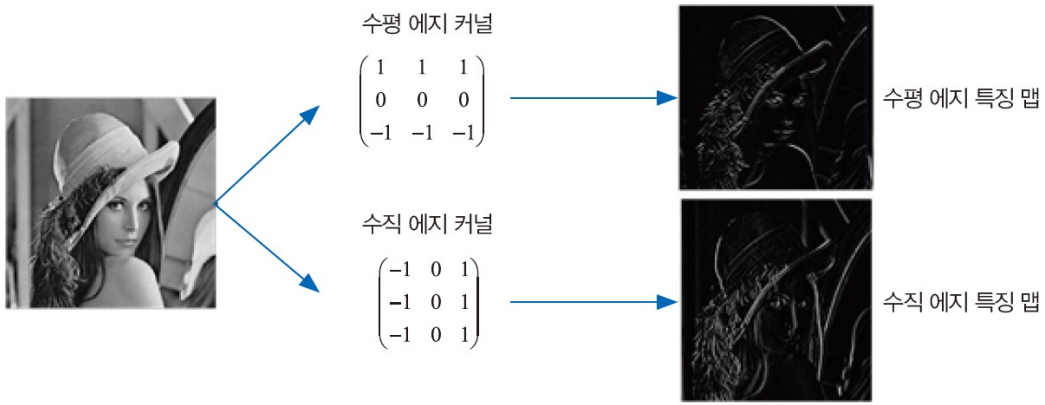


그림 6-5 에지 검출(수평 에지와 수직 에지 특징 맵)

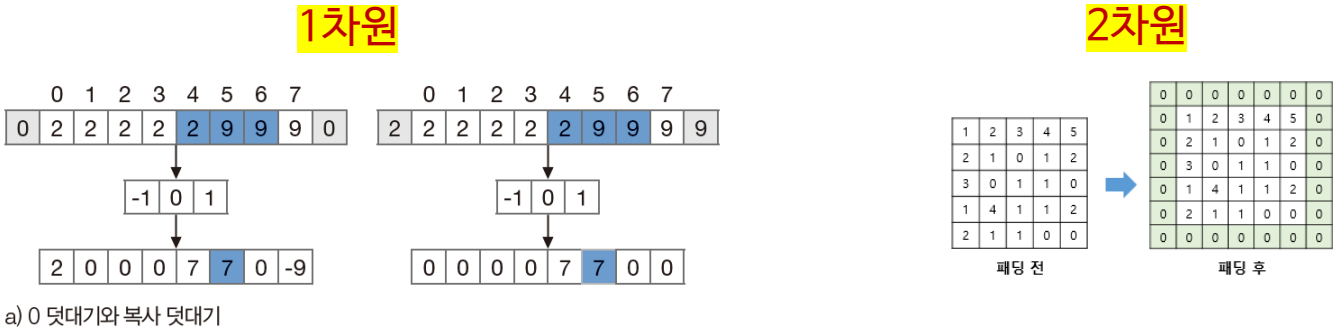
6.2 컨볼루션 신경망의 구조와 동작

6.2.2 컨볼루션층과 풀링층

컨볼루션 신경망은 〈컨볼루션으로 특징 맵을 추출하는 컨볼루션층〉과 〈요약 통계량을 구하는 풀링층〉으로 구성됨

6.2.2.1 컨볼루션층

- 가장자리가 없어지는 문제 및 해결법
  - 딥러닝에서는 컨볼루션층을 많이 쌓기 때문에 데이터의 크기가 점점 작아지는 문제가 심각함
  - 컨볼루션 커널의 크기가 클수록 빨리 작아짐
    - 예를 들어, 11x11 크기의 커널을 사용한다면 상하좌우 모두 5 만큼 씩 줄어드는데 원래 영상이 256x256이라면 컨볼루션을 한번 적용하면 246x246이 됨
  - 아래의 그림과 같이 〈0 덧대기 zero padding〉 혹은 이웃 화솟값을 복사하는 〈복사 덧대기〉를 통해 크기 유지 가능

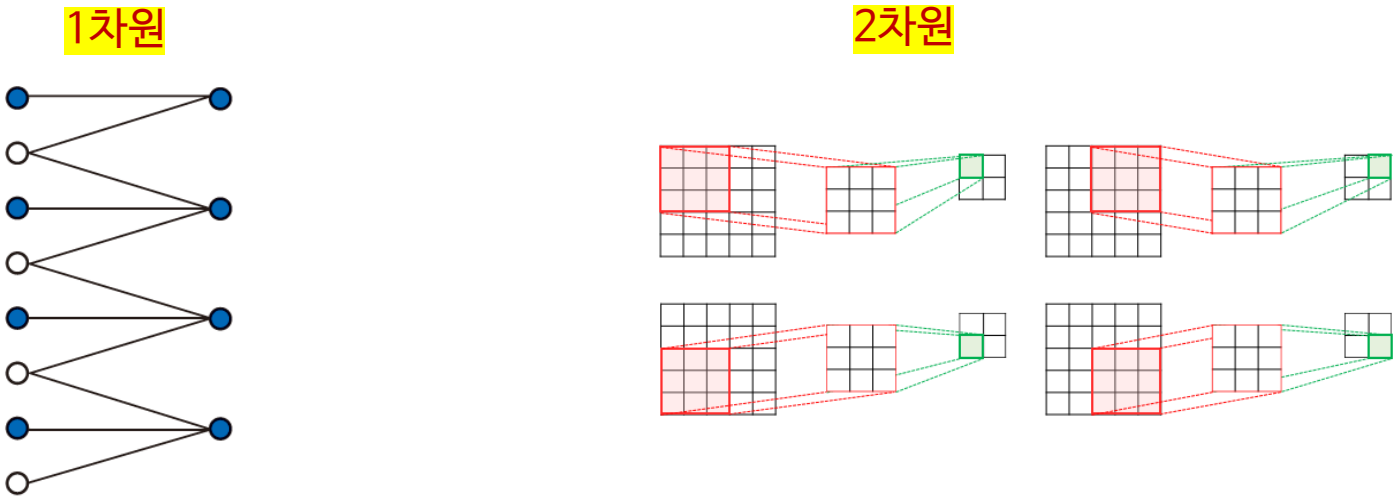


a) 0 덧대기와 복사 덧대기

6.2 컨볼루션 신경망의 구조와 동작

6.2.2.1 컨볼루션층

- 입력영상이 아주 큰 경우 연산량을 줄이기 위해 사용하는 보폭 stride
  - 아래의 그림은 보폭을 2로 설정한 예로, 커널을 두 화소에 한 번씩 적용함
  - 특징 맵은 입력 영상에 비해 반으로 줄었음
  - 일반적으로 보폭을 K로 설정하면 특징 맵의 크기는 1/K로 줄어듬

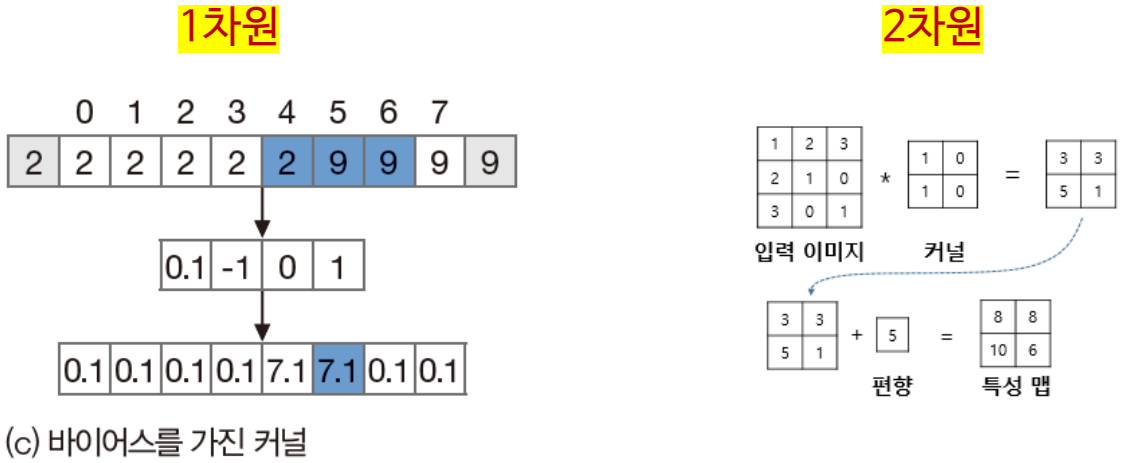


(b) 보폭=2(파란색 화소에 컨볼루션 적용)

6.2 컨볼루션 신경망의 구조와 동작

6.2.2.1 컨볼루션층

- 커널에는 바이어스 bias 가 있음
  - 아래이 그림이 보여주는 바와 같이 커널에는 바이어스가 있음

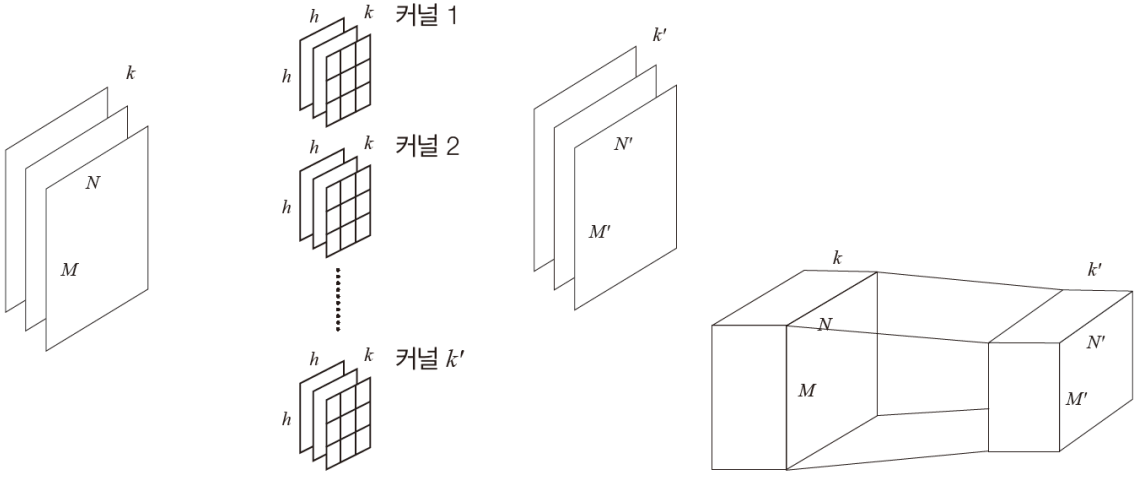


(c) 바이어스를 가진 커널

6.2 컨볼루션 신경망의 구조와 동작

6.2.2.1 컨볼루션층

- 다중 커널의 사용
  - 커널은 한 종류의 특징만 추출하므로 이런 상황에서는 매우 빈약한 특징이 생성됨
  - 영상에서는 변화무쌍한 변화가 나타나므로 풍부한 특징 추출은 컨볼루션 신경망의 성공 여부를 가르는데 매우 중요한 일임
  - 컨볼루션층은 커널을 64개 또는 128개와 같이 아주 많이 사용하여 풍부한 특징 맵을 생성함
  - 커널의 개수가 K`이면 특징맵도 K` 개가 생성됨



(a) 다중 커널 (b) 블록 다이어그램

그림 6-7 컨볼루션층

6.2 컨볼루션 신경망의 구조와 동작

6.2.2.1 컨볼루션층

- 컨볼루션층이 수행하는 연산의 정의
  - 컨볼루션층은 식(6.2)로 구한 **특징맵 f** 에 **활성함수 τ** 를 적용함
  - 활성함수는 특징 맵의 화소 각각에 적용함

컨볼루션층의 연산: $\tau\left(f\left(j,\ i\right)\right)=\tau\left(z\left(j,\ i\right)\circledast u\right)$ (6.3)

$$f\left(j,i\right)=z\left(j,i\right)\circledast u=\sum_{x=-(h-1)/2}^{(h-1)/2}\sum_{y=-(h-1)/2}^{(h-1)/2}z\left(j+y,i+x\right)u\left(y,x\right) \quad (6.2)$$

6.2 컨볼루션 신경망의 구조와 동작

6.2.2.2 풀링층

- 컨볼루션층 다음에는 풀링층이 위치함
- [그림6-8]은 2차원 텐서에 2x2 크기의 커널로 〈최대 풀링max pooling〉을 적용한 사례를 보여줌
- 최대 풀링은 커널 안에 있는 화수 중에서 최댓값을 취하는 연산임
- 풀링은 특징 맵에 나타난 지나친 상세함을 줄여 요약 통계량 summary statistics 을 추출함
- 따라서 보통 보폭을 1보다 크게 하는데, [그림6-8]은 보폭을 2로 설정한 경우임
- 보폭을 s로 설정하면 특징 맵의 크기는 s배 만큼 줄어듬

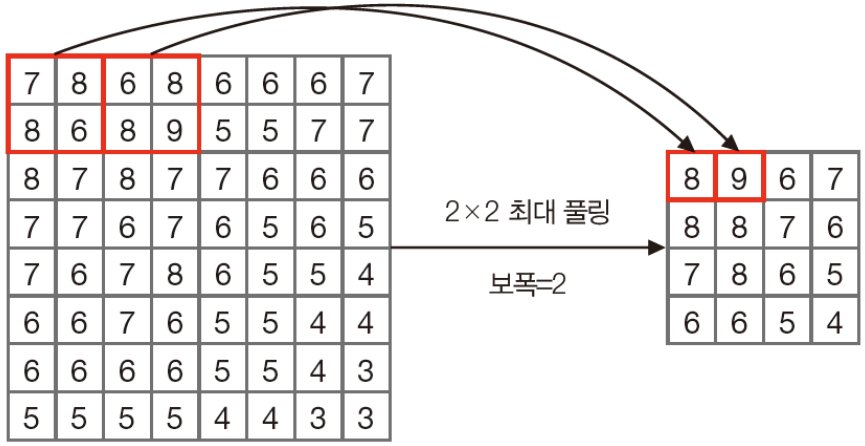


그림 6-8 최대 풀링(보폭이 2인 경우)

6.2 컨볼루션 신경망의 구조와 동작

6.2.2.2 풀링층

- 풀링에는 〈최대 풀링max pooling〉과 〈평균 풀링average pooling〉을 주로 사용함
- 최대 풀링은 커널 안에 있는 화소의 최댓값을 취하고, 평균 풀링은 평균을 취함
- 풀링층은 지나치게 상세한 특징 맵에서 핵심을 추출해 성능을 높여주는 역할뿐 아니라 특징 맵의 크기를 줄여 메모리 효율과 계산 속도를 끌어올리는 효과까지 제공함

6.2 컨볼루션 신경망의 구조와 동작

6.2.2.3 부분 연결성과 가중치 공유

- 컨볼루션층에는 부분 연결성과 가중치 공유라는 좋은 특성이 있음
- [그림6-9(b)]는 오른쪽 층에 있는 노드가 왼쪽 층 노드 3개만 에지로 연결되어 있으며, 이는 [그림5-8]의 깊은 다층 퍼셉트론에서 모든 노드 쌍이 연결된 〈완전 연결 FC: fully connected 구조〉와 확연히 다름
- 왼쪽 층과 오른쪽 층에 n_1과 n_2개의 노드가 있다면, 완전연결의 경우는 $n_1 \times n_2$개의 에지가 있는 반면에 컨볼루션 신경망에는 $h \times n_2$개의 에지가 있음. 커널의 크기 h는 보통 3, 5, 7 정도이므로 컨볼루션 신경망의 가중치가 훨씬 적음. 이런 특성을 부분 연결성이라고 부름

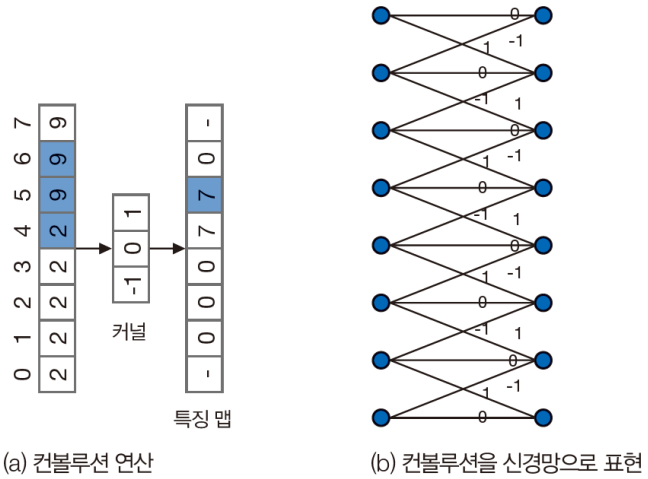


그림 6-9 컨볼루션층의 부분 연결성과 가중치 공유

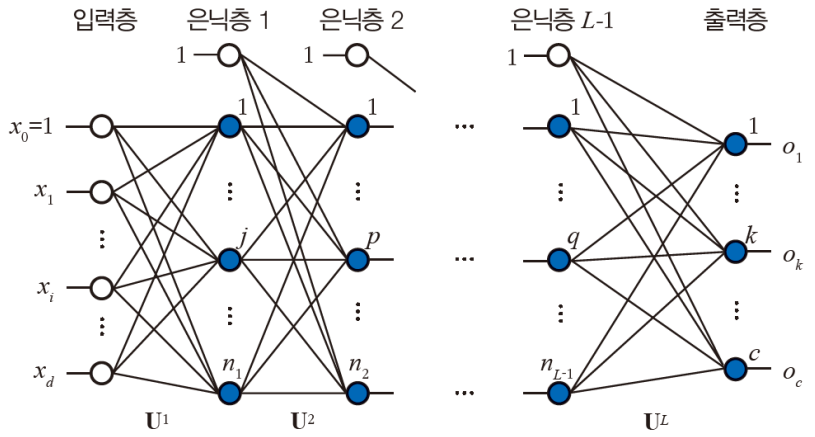


그림 5-8 깊은 다층 퍼셉트론의 구조

6.2 컨볼루션 신경망의 구조와 동작

6.2.2.3 부분 연결성과 가중치 공유

- 컨볼루션 신경망에서는 커널을 구성하는 화소 하나하나가 가중치에 해당하는데, 깊은 다층 퍼셉트론에서는 인접한 두 층의 노드 쌍을 연결하는 에지에 가중치가 있음
- 따라서 컨볼루션 신경망과 깊은 다층 퍼셉트론은 학습 방식이 크게 다름
- [그림6-9]는 학습에 관련된 〈가중치 공유 weight sharing〉라는 컨볼루션 신경망의 또 다른 장점을 설명함

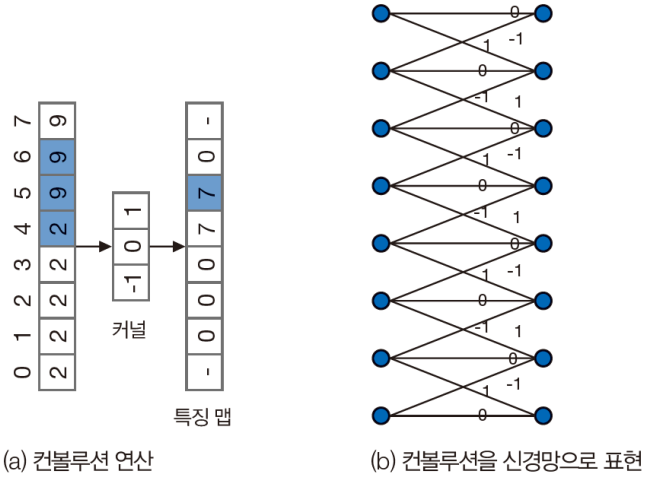


그림 6-9 컨볼루션층의 부분 연결성과 가중치 공유

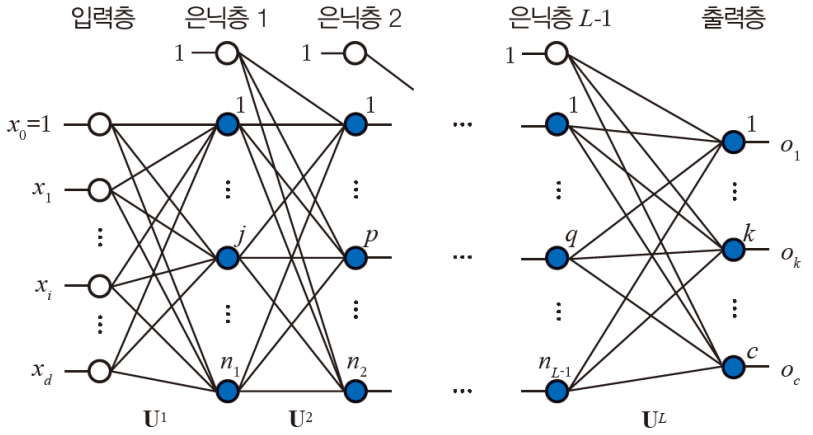
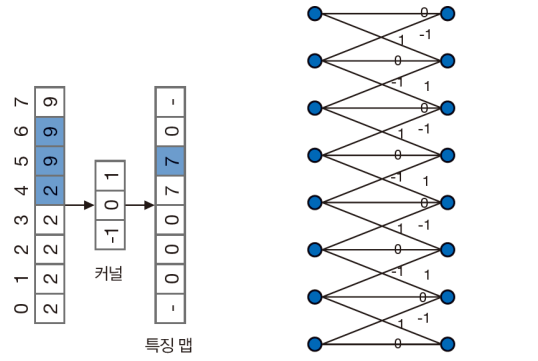


그림 5-8 깊은 다층 퍼셉트론의 구조

6.2 컨볼루션 신경망의 구조와 동작

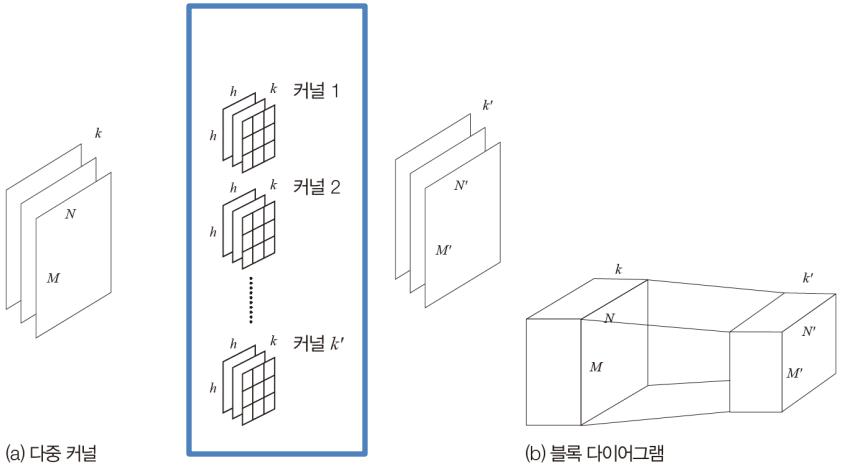
6.2.2.3 부분 연결성과 가중치 공유

- 입력 영상의 모든 곳에 같은 커널을 적용하므로 모든 화소가 **하나의 커널을 공유**하는 셈임
- 가중치 공유는 학습이 최적화해야 하는 매개변수의 개수를 획기적으로 줄여줌
- [그림6-9(b)]에는 hxn_2개의 에지가 있지만, 가중치 공유에 따라 학습이 알아내야 하는 매개변수는 h개에 불과함
- [그림6-7(a)]가 보여주는 컨볼루션층은 hxhxk 크기의 3차원 커널을 k`개 쓰므로 하나의 컨볼루션층에서 학습 알고리즘이 최적화해야 하는 매개변수는 k`xhxhxk 개 임
  - K`은 보통 32, 64, 128 정도이고, h는 3, 5, 7정도이므로 다층 퍼셉트론의 완전연결층에 비하면 매개변수 개수가 획기적으로 줄어듬



(a) 컨볼루션 연산 (b) 컨볼루션을 신경망으로 표현

그림 6-9 컨볼루션층의 부분 연결성과 가중치 공유



(a) 다중 커널 (b) 블록 다이어그램

그림 6-7 컨볼루션층

6.2 컨볼루션 신경망의 구조와 동작

6.2.3 빌딩 블록을 쌓아 만드는 컨볼루션 신경망

- 컨볼루션층과 풀링층을 가지고 컨볼루션 신경망을 구축하며, 빌딩 블록을 쌓아가는 방식으로 만듦

6.2.3.1 빌딩 블록

- [그림5-8]의 깊은 다층 퍼셉트론은 은닉층을 쌓아서 만들며, 은닉층 하나가 빌딩 블록에 해당함
- [그림 6-10(a)]는 컨볼루션 신경망이 사용하는 빌딩 블록임
  - 빌딩 블록의 단위 [컨볼루션-활성함수-풀링] : 예)conv-relu-maxpooling
- 빌딩 블록의 출력 텐서는 다음 빌딩 블록의 입력 텐서로 사용됨

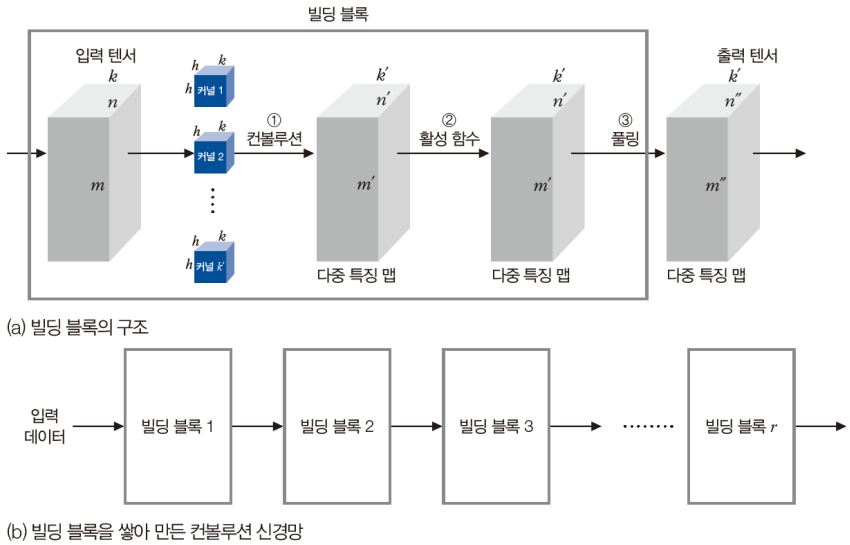
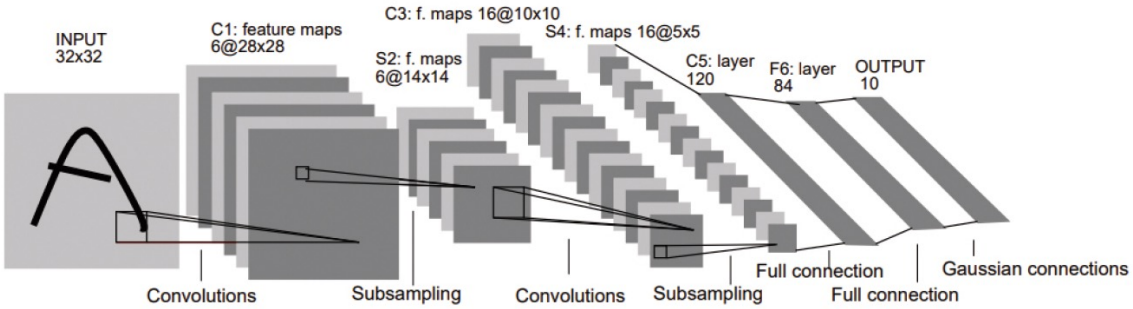


그림 6-10 컨볼루션 신경망의 구조

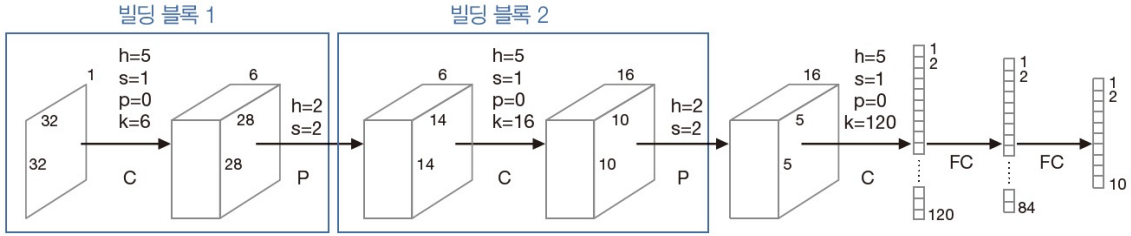
6.2 컨볼루션 신경망의 구조와 동작

6.2.3.2 LeNet-5 사례

- [그림6-11]은 LeNet-5인데, 현재 기준으로 보면 아주 간단한 구조의 컨볼루션 신경망임
- LeNet-5는 필기 숫자 인식 문제에서 컨볼루션 신경망의 가능성을 최초로 입증했다는 점에서 의미가 큼
- 필기 숫자 영상은 컬러가 아닌 명암 영상이므로 입력 데이터이 깊이는 1이고, C는 컨볼루션층, P는 풀링층, FC는 완전연결 층임



(a) [LeCun1998]의 그림



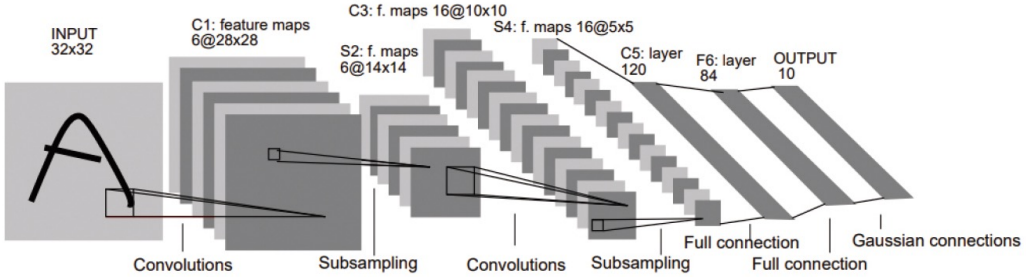
(b) 이 책의 방식에 따른 그림

그림 6-11 LeNet-5의 구조(h는 커널의 크기, s는 보폭, p는 덧대기, k는 커널 개수)

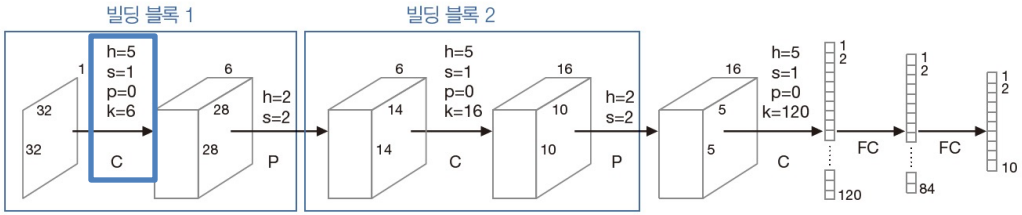
6.2 컨볼루션 신경망의 구조와 동작

6.2.3.2 LeNet-5 사례

- 첫 번째 층은 컨볼루션층인데 5x5커널을 6개 사용하고 보폭은 1, 덧대기는 하지 않음
- 1x32x32 의 텐서에서 28x28 크기의 특징 맵을 6개 추출해 6x28x28의 텐서를 출력함
- 다음은 풀링층인데 2x2커널을 보폭 2로 적용해 6x14x14의 텐서가 출력되며, 여기까지가 빌딩 블록 1임
- 다음은 컨볼루션층인데 입력은 풀링층에서 받은 6x14x14의 텐서이고 출력은 16x10x10텐서임
- 이어지는 풀링층은 2x2커널을 보폭2로 적용해 16x5x5텐서를 출력되며, 여기까지가 빌딩 블록 2임
- 이후에 컨볼루션 연산을 한 번 더 적용하고 완전연결(FC)을 두번 적용함



(a) [LeCun1998]의 그림



(b) 이 책의 방식에 따른 그림

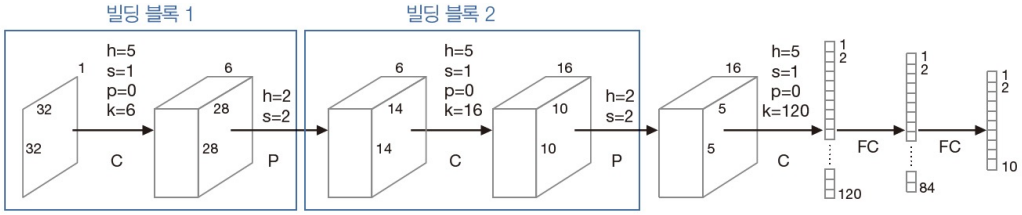
그림 6-11 LeNet-5의 구조(h는 커널의 크기, s는 보폭, p는 덧대기, k는 커널 개수)

6.2 컨볼루션 신경망의 구조와 동작

6.2.3.2 LeNet-5 사례

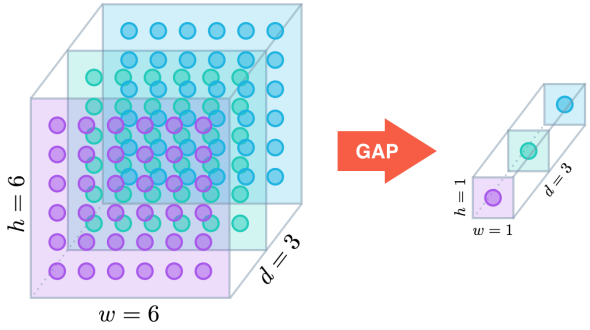
- 전체적으로 LeNet-5는 C-P-C-P-C-FC-FC 층으로 구성되며, 2개의 FC는 다층 퍼셉트론으로 볼 수 있음
- LeNet-5의 학습 알고리즘이 최적화해야 하는 매개변수의 개수는 3,692 + 11,014
  - 3개의 컨볼루션층은 왼쪽에서 오른쪽으로 가면서 5x5커널 6개, 5x5커널 16개, 5x5커널 120개를 사용하므로 매개변수는 총 (5x5+1)x6+(5x5+1)x16+(5x5+1)x120 = 3,692개
  - 완전연결층의 매개변수는 총 (120+1)x84+(84+1)x10=11,014개 ➔ 컨볼루션층의 3배 이상

- 초창기 컨볼루션 신경망은 완전연결층이 과다한 매개변수를 가지는 구조를 사용했는데, 이후에는 완전연결층을 〈전역 평균 풀링 GAP: global average pooling〉과 같은 구조로 대체하여 매개변수 수를 줄이는 방향으로 발전함



(b) 이 책의 방식에 따른 그림

그림 6-11 LeNet-5의 구조(h는 커널의 크기, s는 보폭, p는 덧대기, k는 커널 개수)



6.3 컨볼루션 신경망의 학습

6.3.1 손실 함수와 옵티마이저

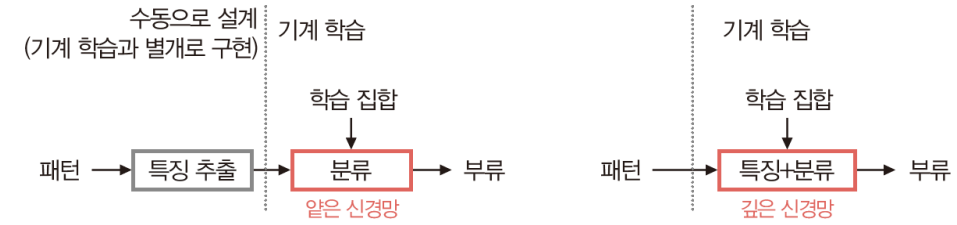
- 손실 함수는 신경망이 범하는 오류를 측정함
- 컨볼루션 신경망의 내부구조는 깊은 다층 퍼셉트론과 다르지만 출력층은 같음
- 예를 들어, 숫자 인식을 하는 신경망이라면 출력층에 노드가 10개 있고 softmax 또는 tanh 활성함수의 결과를 출력
- 따라서 컨볼루션 신경망은 깊은 다층 퍼셉트론과 똑같은 손실 함수를 사용함

- 옵티마이저는 손실 함수의 최저점을 찾아감
- 이때 매개변수가 어느 방향으로 변해야 손실 함수의 최저점에 가까워지는지 알아야 하며, 손실 함수를 매개변수로 미분하여 방향을 알아냄
- 컨볼루션 신경망은 커널이 매개변수에 해당하고 깊은 다층 퍼셉트론은 에지의 가중치가 매개변수라는 점만 다르고 미분으로 방향을 알아내는 원리는 동일함
- 따라서 컨볼루션 신경망과 깊은 다층 퍼셉트론은 같은 옵티마이저를 사용함

6.3 컨볼루션 신경망의 학습

6.3.2 통째 학습 end-to-end learning

- 수작업 특징 hand-crafted features
  - 고전적인 컴퓨터 비전에서는 [그림6-5]와 같이 수평 에지와 수직 에지 특징을 추출하는 커널을 사람이 직접 설계했음
  - 수작업 특징은 사람의 직관으로 설계되므로 어느 정도 수준의 성능을 제공하지만, 자연 영상과 같이 복잡한 데이터에서는 아주 낮은 성능에 머물러 있음
  - [그림6-12(a)] 는 수작업 특징을 사용하는 고전적인 컴퓨터 비전 방식을 보여 줌
- 특징 학습 feature learning
  - 딥러닝은 특징을 추출하는 방식에서 패러다임 변화를 일으킴
  - [그림 6-12(b)]는 특징 추출과 분류를 동시에 학습으로 알아내는 방식이며, 딥러닝은 특징을 학습으로 알아내기 때문에 특징 학습을 한다고 말함
  - 또한 입력 패턴에서 최종 출력에 이르는 전체 과정을 한꺼번에 학습한다는 의미에서 통째 학습end-to-end learning 을 한다고 함
  - 통째 학습은 딥러닝의 월등한 성능을 설명하는 가장 중요한 요임임



(a) 딥러닝 이전의 수작업 특징 (b) 딥러닝 이후의 통째 학습

그림 6-12 딥러닝에 의한 컴퓨터 비전 패러다임 변화

6.3 컨볼루션 신경망의 학습

6.3.2 통째 학습 end-to-end learning

- 컨볼루션 신경망은 딥러닝의 일종이므로 딥러닝 패러다임을 따름
- LeNet-5에 [그림6-12(b)]의 패러다임을 씌워 딥러닝의 원리를 좀 더 구체적으로 설명함
- 앞부분의 컨볼루션층과 풀링층은 특징 추출을, 뒷부분의 완전연결층은 분류를 담당함
- 컨볼루션 신경망의 학습 알고리즘은 깊은 다층 퍼셉트론과 마찬가지로 오류 역전파 알고리즘을 사용함
- 단, 최적화해야 할 매개변수가 완전연결층의 가중치 뿐만 아니라 컨볼루션층이 커널도 포함된다는 점만 다름

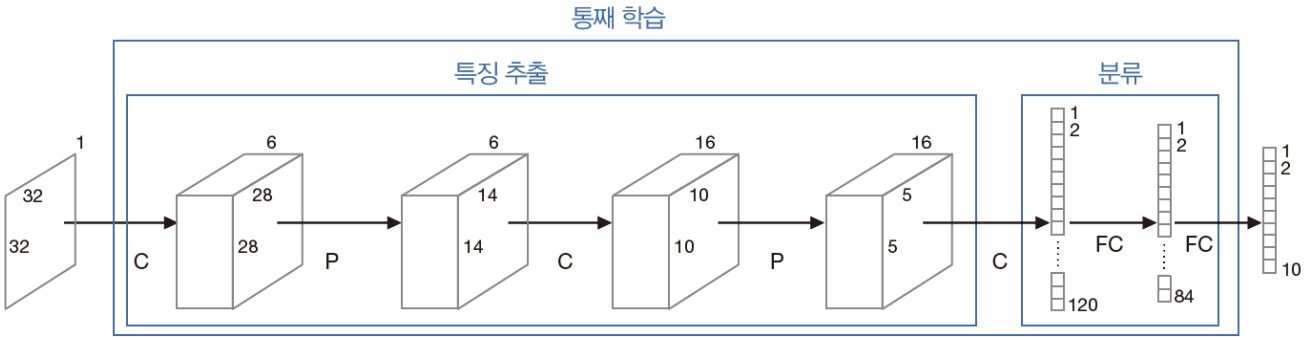


그림 6-13 특징 추출과 분류를 동시에 학습하는 딥러닝의 통째 학습

6.3 컨볼루션 신경망의 학습

6.3.3 컨볼루션 신경망의 성능이 월등한 이유

- “통째 학습을 한다”
  - 특징 추출과 분류를 동시에 최적화하는 학습 방법은 따로 최적화한 후에 결합하는 학습방법보다 뛰어남

- “특징 학습을 한다”
  - 주어진 데이터셋에 최적인 특징 추출 알고리즘을 학습으로 알아내기 때문에 상당한 성능향상이 있음

- “신경망의 깊이를 더욱 깊게 하여 풍부한 특징을 추출한다”
  - 컨볼루션층은 부분 연결성과 가중치 공유로 인해 매개변수 개수가 적음. 따라서 신경망의 깊이를 충분히 깊게 해도 학습이 잘됨

- “컨볼루션 연산은 데이터의 원래 구조를 그대로 유지한 채 특징을 추출한다”
  - 입력 데이터가 컬러 영상의 3차원 텐서라면 컨볼루션층이 출력하는 특징 맵도 3차원 텐서임
  - 영상의 한 화소는 상하좌우에 있는 이웃 화소와 상관관계가 큰데, 컨볼루션 연산은 상관 관계 정보를 그대로 유지하는 강점이 있음.
  - 깊이 다층 퍼셉트론에서는 영상을 1차원으로 펼쳐 입력하기 때문에 이웃 정보가 사라짐

6.4 컨볼루션 신경망 프로그래밍

6.4.1 필기 숫자 인식

6.4.1.1 LeNet-5

[프로그램 6-1] LeNet-5로 MNIST 인식

```
import numpy as np
import torch
from torchvision import datasets
from torchvision.transforms import ToTensor

# MNIST 읽고 텐서 구조 확인 (학습 데이터)
MNIST_training_data = datasets.MNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor()
)
# MNIST 읽고 텐서 구조 확인 (평가 데이터)
MNIST_test_data = datasets.MNIST(
    root="data",
    train=False,
    download=True,
    transform=ToTensor()
)
```

MNIST 데이터를 다운로드하고 ToTensor()로 전처리

6.4 컨볼루션 신경망 프로그래밍

6.4.1 필기 숫자 인식

6.4.1.1 LeNet-5

[프로그램 6-1] LeNet-5로 MNIST 인식

```
1  from torch import nn
2
3  class LeNet5(nn.Module):
4      def __init__(self):
5          super(LeNet5, self).__init__()
6
7          self.input_layer = nn.Sequential(
8              nn.Conv2d(in_channels=1, out_channels=6, kernel_size=(5,5), stride=1, padding=2),
9              nn.ReLU(),
10             nn.MaxPool2d(kernel_size=(2, 2))
11         )
12         self.hidden_layer_1 = nn.Sequential(
13             nn.Conv2d(in_channels=6, out_channels=16, kernel_size=(5,5), stride=1, padding=2),
14             nn.ReLU(),
15             nn.MaxPool2d(kernel_size=(2, 2))
16         )
17         self.hidden_layer_2 = nn.Sequential(
18             nn.Conv2d(in_channels=16, out_channels=120, kernel_size=(5,5), stride=1, padding=2),
19             nn.ReLU(),
20         )
21         self.Flatten = nn.Flatten()  # 3차원 feature map을 1차원 벡터로 펼쳐주는(Flatten) 레이어
22         self.hidden_layer_3 = nn.Sequential(
23             nn.Linear(in_features=120*7*7, out_features=84),
24             nn.ReLU()
25         )
26         self.classifier = nn.Linear(in_features=84, out_features=10)
```

7-11행 1채널의 이미지를 입력받는 첫 컨볼루션 계층 정의

21행 (채널, 너비, 높이) 3채널의 특징 맵을 1차원 벡터로 변환하는 계층

26행 모든 은닉층을 통과한 특징 벡터를 이용해 예측을 수행하는 계층

6.4 컨볼루션 신경망 프로그래밍

6.4.1 필기 숫자 인식

6.4.1.1 LeNet-5

[프로그램 6-1] LeNet-5로 MNIST 인식

```
28      def forward(self, input):
29          """
30          input shape: (1, 28, 28)
31          output shape: (10)
32          """
33          x = self.input_layer(input)  # (1, 28, 28) -> (6, 14, 14)
34          x = self.hidden_layer_1(x)   # (6, 14, 14) -> (16, 7, 7)
35          x = self.hidden_layer_2(x)   # (16, 7, 7)  -> (120, 7, 7)
36          x = self.Flatten(x)          # (120, 7, 7) -> (5880)
37          x = self.hidden_layer_3(x)   # (5880) -> (84)
38          x = self.classifier(x)       # (84) -> (10)
39          return x
40
41  model = LeNet5().cuda()
```

28행 실제로 입력 데이터가 각 계층을 통과하는 과정인 forward 함수 정의

33-39행 각 계층을 통과하며 변화하는 데이터의 형태(shape)에 주목

41행 정의한 모델을 선언하고, GPU로 이동

6.4 컨볼루션 신경망 프로그래밍

6.4.1 필기 숫자 인식

6.4.1.1 LeNet-5

[프로그램 6-1] LeNet-5로 MNIST 인식

```
1  from torch.optim import Adam
2  from torch.nn import CrossEntropyLoss
3  from torch.utils.data import DataLoader
4
5  # 학습을 위한 최적화 알고리즘과 손실 함수 정의
6  optimizer = Adam(model.parameters(), lr=1e-3)
7  loss_fn = CrossEntropyLoss().cuda()
8
9  # 데이터를 배치 단위로 불러오기 위한 데이터로더 정의
10 train_dataloader = DataLoader(MNIST_training_data, batch_size=128, shuffle=True, drop_last=True)
11 val_dataloader = DataLoader(MNIST_test_data, batch_size=256, shuffle=False, drop_last=False)
```

6-7행 학습을 위한 최적화 알고리즘과 손실 함수를 정의

10-11행 데이터를 배치 단위로 불러오기 위해 DataLoader 구성

6.4 컨볼루션 신경망 프로그래밍

6.4.1 필기 숫자 인식

6.4.1.1 LeNet-5

[프로그램 6-1] LeNet-5로 MNIST 인식

```
1  from tqdm import tqdm  # 진행상황 시각화를 위한 라이브러리
2
3  total_epoch = 30
4
5  training_loss_hist = list()
6  training_acc_hist = list()
7
8  val_loss_hist = list()
9  val_acc_hist = list()
10
11 for epoch in range(total_epoch):
12     loss_epoch = 0
13     avg_accuracy_epoch = 0
14
15     model.train()  # 모델을 학습 모드로 설정
16     for x_train_batch, y_train_batch in tqdm(train_dataloader):
17         x_train_batch = x_train_batch.cuda()
18         y_train_batch = y_train_batch.cuda()
19
20         pred = model(x_train_batch)
21         loss = loss_fn(pred, y_train_batch)
22
23         optimizer.zero_grad()  # 옵티마이저 초기화
24         loss.backward()  # 오차 역전파
25         optimizer.step()  # 역전파된 오차를 이용한 파라미터 업데이트 수행
26
27         loss_epoch += loss.item()
28         avg_accuracy_epoch += torch.sum(torch.argmax(pred, dim=1) == y_train_batch).item()
29
30     loss_epoch /= len(val_dataloader.dataset)
31     avg_accuracy_epoch /= len(train_dataloader.dataset)
32
33     # 학습 데이터에서의 손실 값과 accuracy 저장
34     training_loss_hist.append(loss_epoch)
35     training_acc_hist.append(avg_accuracy_epoch)
```

3-9행 학습에 사용할 매개변수와 학습 중 정확률과 손실을 기록할 리스트 정의

15행 모델을 학습 모드로 설정 (중요)

16-25행 배치 단위의 학습을 진행

27-35행 학습 중 손실값과 정확률을 저장

6.4 컨볼루션 신경망 프로그래밍

6.4.1 필기 숫자 인식

6.4.1.1 LeNet-5

[프로그램 6-1] LeNet-5로 MNIST 인식

```
40      model.eval()  # 모델을 테스트 모드로 설정
41      for x_val_batch, y_val_batch in val_dataloader:
42          x_val_batch = x_val_batch.cuda()
43          y_val_batch = y_val_batch.cuda()
44
45          with torch.no_grad():  # 테스트 간 그래디언트 계산 중지
46              pred = model(x_val_batch)
47              loss = loss_fn(pred, y_val_batch)
48
49          loss_epoch += loss.item()
50          avg_accuracy_epoch += torch.sum(torch.argmax(pred, dim=1) == y_val_batch).item()
51
52      loss_epoch /= len(val_dataloader.dataset)
53      avg_accuracy_epoch /= len(val_dataloader.dataset)
54
55      # 테스트 데이터에서의 손실 값과 accuracy 저장
56      val_loss_hist.append(loss_epoch)
57      val_acc_hist.append(avg_accuracy_epoch)
58
59      print(f"Epoch: {epoch+1}/{total_epoch}\ttrain_acc. {training_acc_hist[-1] * 100}"+
60            f"\ttrain_loss {training_loss_hist[-1]}\tval_acc. {val_acc_hist[-1] * 100}\tval_loss {val_loss_hist[-1]}")
```

40행 모델을 평가(eval) 모드로 설정 (중요)

45행 검증 데이터를 다루는 동안 모델의 그래디언트 계산을 중단 (중요)

49-57행 검증 데이터에서의 손실값과 정확률 저장

59행 한 에포크의 학습이 완료되면 해당 에포크의 결과 출력

6.4 컨볼루션 신경망 프로그래밍

6.4.1 필기 숫자 인식

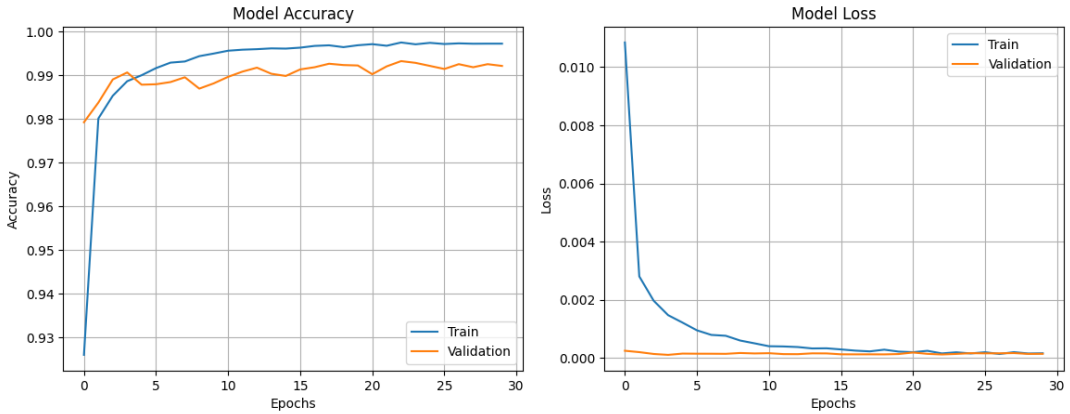
6.4.1.1 LeNet-5

[프로그램 6-1] LeNet-5로 MNIST 인식

```
import matplotlib.pyplot as plt

plt.plot(training_acc_hist)
plt.plot(val_acc_hist)

plt.title("Model Accuracy")
plt.ylabel("Accuracy")
plt.xlabel("Epochs")

plt.legend(["Train", "Validation"], loc="best")
plt.grid()
plt.show()
```

```
plt.plot(training_loss_hist)
plt.plot(val_loss_hist)

plt.title("Model Loss")
plt.ylabel("Loss")
plt.xlabel("Epochs")

plt.legend(["Train", "Validation"], loc="best")
plt.grid()
plt.show()
```

학습을 마친 후 정확률과 손실 값을 시각화



6.4 컨볼루션 신경망 프로그래밍

6.4.1 필기 숫자 인식

6.4.1.2 컨볼루션 신경망의 유연한 구조

[프로그램 6-2] 컨볼루션 신경망으로 MNIST 인식

```
1  from torch import nn
2
3  class CNN(nn.Module):
4      def __init__(self):
5          super(CNN, self).__init__()
6
7          self.input_layer = nn.Sequential(
8              nn.Conv2d(in_channels=1, out_channels=32, kernel_size=(3,3), stride=1, padding=1),
9              nn.ReLU(),
10         )
11         self.hidden_layer_1 = nn.Sequential(
12             nn.Conv2d(in_channels=32, out_channels=64, kernel_size=(3, 3), stride=1, padding=1),
13             nn.ReLU(),
14         )
15         self.maxpool = nn.MaxPool2d(kernel_size=(2, 2))
16         self.dropout_1 = nn.Dropout(p=0.25)
17         self.flatten = nn.Flatten()
18         self.hidden_layer_2 = nn.Sequential(
19             nn.Linear(in_features=64*14*14, out_features=128),
20             nn.ReLU()
21         )
22         self.dropout_2 = nn.Dropout(p=0.5)
23         self.classifier = nn.Linear(in_features=128, out_features=10)
24
25     def forward(self, input):
26         """
27         input shape: (1, 28, 28)
28         output shape: (10)
29         """
30         x = self.input_layer(input)
31         x = self.hidden_layer_1(x)
32         x = self.maxpool(x)
33         x = self.dropout_1(x)
34         x = self.flatten(x)
35         x = self.hidden_layer_2(x)
36         x = self.dropout_2(x)
37         x = self.classifier(x)
38         return x
39
40 model = CNN().cuda()
```

LeNet-5와 다른 구조의 CNN을 설계하여 MNIST 인식 수행
(모델 정의 부분을 제외하고 [프로그램 6-1]과 같은 코드 사용하되, 12에포크 학습)

16, 22행 서로 다른 드롭아웃 확률을 갖는 드롭아웃 계층 사용

30-32행 컨볼루션 층 사이에 풀링을 매번 수행한 LeNet과 다르게
컨볼루션 두 층을 연달아 사용하고 풀링 수행

33행 드롭아웃 계층의 적용

6.4 컨볼루션 신경망 프로그래밍

6.4.3 패션 데이터 인식

[프로그램 6-3] 컨볼루션 신경망으로 Fashion MNIST 인식

```python
import numpy as np
import torch
from torchvision import datasets
from torchvision.transforms import ToTensor

# FashionMNIST 읽고 텐서 구조 확인 (학습 데이터)
Fashion_MNIST_training_data = datasets.FashionMNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor()
)
# FashionMNIST 읽고 텐서 구조 확인 (평가 데이터)
Fashion_MNIST_test_data = datasets.FashionMNIST(
    root="data",
    train=False,
    download=True,
    transform=ToTensor()
)
```

패션 관련 이미지를 다루는 Fashion MNIST에 대한 예측 실습해보기
(데이터셋 로드 부분을 제외하고 [프로그램 6-2]와 같은 코드 사용)

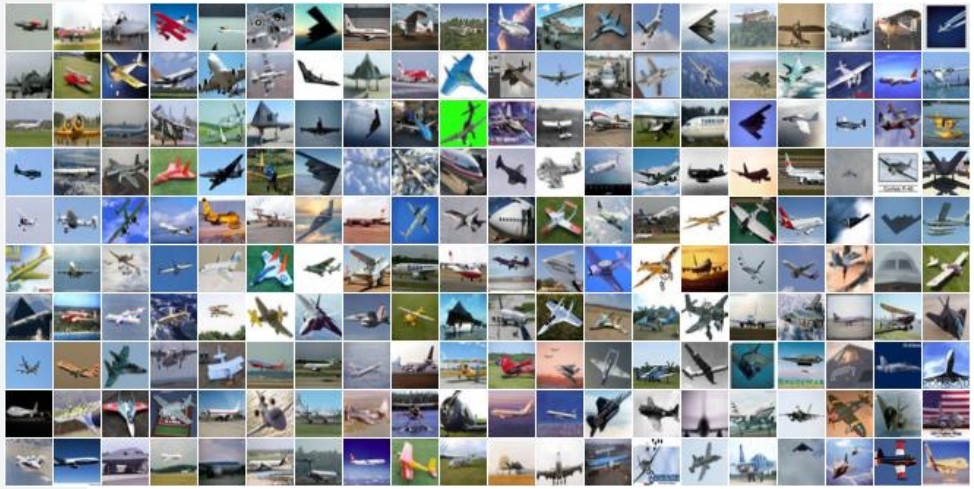
6.4 컨볼루션 신경망 프로그래밍

6.4.4 자연 영상 데이터 인식

[프로그램 6-4] 컨볼루션 신경망으로 CIFAR-10 인식

CIFAR-10 데이터셋 로드하기

```
import numpy as np
import torch
from torchvision import datasets
from torchvision.transforms import ToTensor

# CIFAR 읽고 텐서 구조 확인 (학습 데이터)
CIFAR_training_data = datasets.CIFAR10(
    root="data",
    train=True,
    download=True,
    transform=ToTensor()
)
# CIFAR 읽고 텐서 구조 확인 (평가 데이터)
CIFAR_test_data = datasets.CIFAR10(
    root="data",
    train=False,
    download=True,
    transform=ToTensor()
)
```



6.4 컨볼루션 신경망 프로그래밍

6.4.4 자연 영상 데이터 인식

[프로그램 6-4] 컨볼루션 신경망으로 CIFAR-10 인식

```
from torch import nn

class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()

        self.input_layer = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=32, kernel_size=(3,3), stride=1, padding=1),
            nn.ReLU(),
        )
        self.hidden_layer_1 = nn.Sequential(
            nn.Conv2d(in_channels=32, out_channels=32, kernel_size=(3, 3), stride=1, padding=1),
            nn.ReLU(),
        )
        self.maxpool = nn.MaxPool2d(kernel_size=(2, 2))
        self.dropout_1 = nn.Dropout(p=0.25)
        self.hidden_layer_2 = nn.Sequential(
            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=(3, 3), stride=1, padding=1),
            nn.ReLU(),
            nn.Conv2d(in_channels=64, out_channels=64, kernel_size=(3, 3), stride=1, padding=1),
            nn.ReLU(),
        )
        self.flatten = nn.Flatten()
        self.hidden_layer_3 = nn.Sequential(
            nn.Linear(in_features=64*8*8, out_features=512),
            nn.ReLU()
        )
        self.dropout_2 = nn.Dropout(p=0.5)
        self.classifier = nn.Linear(in_features=512, out_features=10)
```

8행 입력이 RGB 컬러 이미지 이므로 입력 채널은 3으로 설정

6.4 컨볼루션 신경망 프로그래밍

6.4.4 자연 영상 데이터 인식

[프로그램 6-4] 컨볼루션 신경망으로 CIFAR-10 인식

```
31      def forward(self, input):
32          """
33          input shape: (3, 32, 32)
34          output shape: (10)
35          """
36          x = self.input_layer(input)
37          x = self.hidden_layer_1(x)
38          x = self.maxpool(x)
39          x = self.dropout_1(x)
40          x = self.hidden_layer_2(x)
41          x = self.maxpool(x)
42          x = self.dropout_1(x)
43          x = self.flatten(x)
44          x = self.hidden_layer_3(x)
45          x = self.dropout_2(x)
46          x = self.classifier(x)
47          return x
48
49  model = CNN().cuda()
```

36-42행 컨볼루션 계층을 통한 특성 추출

43행 얻어진 특징 맵을 1차원으로 평탄화

44-47행 얻어진 특징을 추가로 은닉층에 입력 후, 최종 예측 결과 반환

[프로그램 6-3]과 동일한 방식으로 30에포크 학습 진행

6.4 컨볼루션 신경망 프로그래밍

6.4.5 학습된 모델 저장과 재활용

[프로그램 6-4] 컨볼루션 신경망으로 CIFAR-10 인식

```
1 torch.save(model.state_dict(), "./my_cnn_state_dict.pth")
```

학습이 완료된 모델의 state_dict를 파일로 저장

[프로그램 6-5] 학습된 모델 불러다 쓰기

[프로그램 6-4]와 동일하게 모델 정의까지 진행

학습을 처음부터 진행하는 대신, 학습된 모델의 state_dict 불러오기

```
1 model.load_state_dict(torch.load("/kaggle/working/my_cnn_state_dict.pth"))
```

```
1  from torch.utils.data import DataLoader
2
3  val_dataloader = DataLoader(CIFAR_test_data, batch_size=256, shuffle=False, drop_last=False)
4
5  model.eval()
6  acc = 0
7  for x, y in val_dataloader:
8      x = x.cuda()
9      y = y.cuda()
10
11     with torch.no_grad():
12         pred = model(x)
13     acc += torch.sum(torch.argmax(pred, dim=1) == y).item()
14
15 acc /= len(val_dataloader.dataset)
16 print("테스트 정확률:", acc*100)
```

불러온 모델로 평가 진행 시 동일한 결과를 얻을 수 있음