

딥러닝은 층을 깊게 한 심층 신경망입니다. 심층 신경망은 지금까지 설명한 신경망을 바탕으로 뒷단에 층을 추가하기만 하면 만들 수 있지만, 커다란 문제가 몇 개 있습니다. 이번 장에서는 딥러닝의 특징과 과제, 그리고 가능성을 살펴봅니다. 또 오늘날의 첨단 딥러닝에 대한 설명도 준비했습니다.

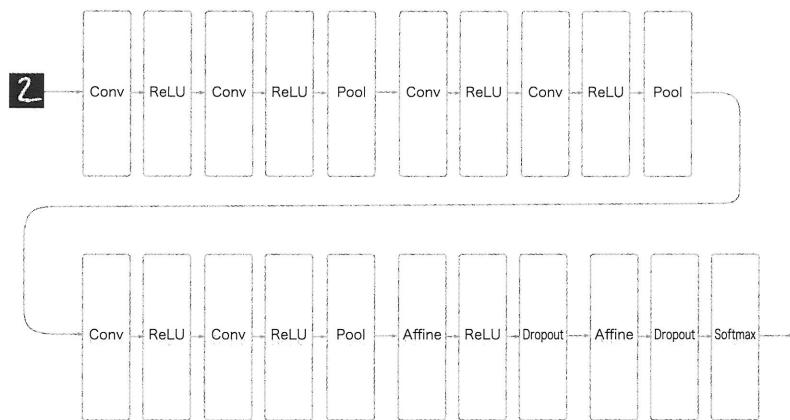
## 8.1 더 깊게

신경망에 관해 그동안 많은 것을 배웠습니다. 신경망을 구성하는 다양한 계층과 학습에 효과적인 기술, 영상 분야에 특히 유효한 CNN과 매개변수 최적화 기법 등이 떠오를 겁니다. 이 모두가 딥러닝에서 중요한 기술입니다. 이번 절에서는 그동안 배운 기술을 집약하고 심층 신경망을 만들어 MNIST 데이터셋의 손글씨 숫자 인식에 도전하려 합니다.

### 8.1.1 더 깊은 신경망으로

거두절미하고, 이번 절에서는 [그림 8-1]과 같이 구성된 CNN을 만들고자 합니다(이 신경망은 다음 절에서 설명하는 VGG 신경망을 참고하였습니다).

그림 8-1 손글씨 숫자를 인식하는 심층 CNN



처 보아도 지금까지 구현한 신경망보다 층이 깊습니다. 여기에서 사용하는 합성곱 계층은 모두  $3 \times 3$  크기의 작은 필터로, 층이 깊어지면서 채널 수가 더 늘어나는 것이 특징입니다(합성곱 계층의 채널 수는 앞 계층에서부터 순서대로 16, 16, 32, 32, 64, 64로 늘어갑니다). 또 그림과 같이 풀링 계층을 추가하여 중간 데이터의 공간 크기를 점차 줄여갑니다. 그리고 마지막 단의 완전연결 계층에서는 드롭아웃 계층을 사용합니다.

가중치 초기값으로 He 초기값을 사용하고, 가중치 매개변수 갱신에는 Adam을 이용합니다. 이상을 정리하면 이 신경망의 특징은 다음과 같습니다.

- $3 \times 3$ 의 작은 필터를 사용한 합성곱 계층
- 활성화 함수는 ReLU
- 완전연결 계층 뒤에 드롭아웃 계층 사용
- Adam을 사용해 최적화
- 가중치 초기값은 'He의 초기값'

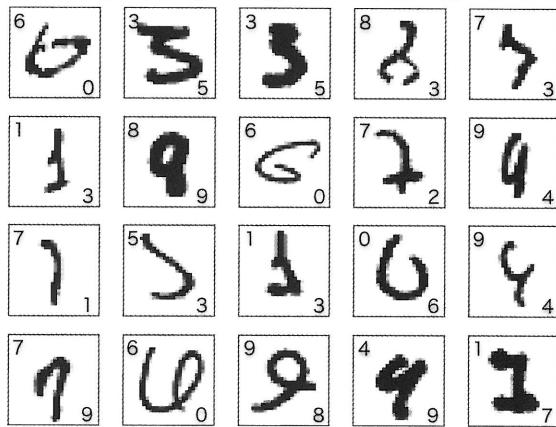
이상의 특징에서 보듯 이 신경망에는 그동안 배운 신경망 기술을 잔뜩 녹였습니다. 그럼 이 신경망을 학습시켜볼까요? 결과부터 말하면 이 신경망의 정확도는 99.38%\*가 되겠습니다. 이 정도면 매우 훌륭한 성능이라고 할 수 있죠!

\* 최종 정확도에는 얼마간 차이가 날 수 있습니다. 다만, 이번 네트워크에서는 대체로 99%를 넘길 겁니다.

**NOTE**\_ 이 신경망을 구현한 소스 코드는 ch08/deep\_convnet.py에, 또 훈련용 코드는 ch08/train\_deepnet.py에 준비되어 있습니다. 이 코드들을 사용하여 학습을 직접 시켜봐도 좋습니다만, 심층 신경망을 학습시키는 데는 시간이 오래 걸립니다(아마 반나절은 걸릴 겁니다). 이 책에서는 학습된 가중치 매개변수를 ch08/deep\_conv\_net\_params.pkl 파일에 준비해놨습니다. 이전의 deep\_convnet.py는 학습된 매개변수를 읽어 들일 수 있으니 적절히 이용하세요.

이 신경망이 잘못 인식할 확률은 겨우 0.62%입니다. 그럼 실제로 어떤 이미지를 인식하지 못했는지 살펴볼까요. [그림 8-2]는 인식에 실패한 예입니다.

**그림 8-2** 인식하지 못한 이미지들 : 각 사진의 왼쪽 위는 정답 레이블, 오른쪽 아래는 이 신경망의 추론 결과



[그림 8-2]의 사진들은 우리 인간도 판단하기 어려운 이미지입니다. 실제로 우리도 똑같이 ‘인식 오류’를 저지르는 이미지가 여럿 포함되어 있지요. 첫 번째 이미지는 ‘0’처럼 보이고(정답은 ‘6’), 두 번째 이미지는 분명히 ‘5’처럼도 보입니다(정답은 ‘3’). 전체적으로 ‘1’과 ‘7’, ‘0’과 ‘6’, ‘3’과 ‘5’의 조합이 헷갈리는데, 이런 예를 보자면 인식을 못 한 것도 이해가 됩니다.

이번의 심층 CNN은 정확도가 높고, 잘못 인식한 이미지들도 인간과 비슷한 인식 오류를 저지르고 있습니다. 이런 점에서도 심층 CNN의 잠재력이 크다는 걸 새삼 느낄 수 있을 거예요.

## 8.1.2 정확도를 더 높이려면

〈What is the class of this image?〉 웹 사이트<sup>[32]</sup>는 다양한 데이터셋을 대상으로 그동안 논문 등에서 발표한 기법들의 정확도 순위를 정리해두었습니다(그림 8-3).

그림 8-3 MNIST 데이터셋에 대한 각 기법의 순위(2016년 12월 시점)<sup>[32]</sup>

MNIST			
who is the best in MNIST ?			
Result	Method	Venue	Details
0.21%	Regularization of Neural Networks using DropConnect	ICML 2013	
0.23%	Multi-column Deep Neural Networks for Image Classification	CVPR 2012	
0.23%	APAC: Augmented PAttern Classification with Neural Networks	arXiv 2015	
0.24%	Batch-normalized Maxout Network in Network	arXiv 2015	Details
0.29%	Generalizing Pooling Functions in Convolutional Neural Networks: Mixed, Gated, and Tree	AISTATS 2016	Details
0.31%	Recurrent Convolutional Neural Network for Object Recognition	CVPR 2015	
0.31%	On the Importance of Normalisation Layers in Deep Learning with Piecewise Linear Activation Units	arXiv 2015	
0.32%	Fractional Max-Pooling	arXiv 2015	Details

[그림 8-3]의 순위를 보면 ‘Neural Networks’나 ‘Deep’, ‘Convolutional’이라는 키워드가 돋보입니다. 사실 상위권은 대부분 CNN을 기초로 한 기법들이 점령했습니다. 참고로 2016년 10월 현재 MNIST 데이터셋에 대한 정확도 1위는 99.79%(오류율 0.21%)이며, 이 기법도 CNN을 기초로 했습니다.<sup>[33]</sup> 다만 이 목록의 기법들이 사용하는 CNN들은 그다지 깊지 않습니다(합성곱 계층 2개에 완전연결 계층 2개 정도인 신경망).

**NOTE** MNIST 데이터셋에 대해서는 층을 아주 깊게 하지 않고도 (현시점에서는) 최고 수준의 결과가 나옵니다. 이는 손글씨 숫자라는 문제가 비교적 단순해서 신경망의 표현력을 극한까지 높일 필요가 없기 때문이라고 생각합니다. 그래서 층을 깊게 해도 헤택이 적다고 할 수 있죠. 반면, 나중에 소개하는 대규모 일반 사물 인식에서는 문제가 훨씬 복잡해지므로 층을 깊게 하면 정확도를 크게 끌어올릴 수 있습니다.

[그림 8-3]의 상위 기법들을 참고하면 정확도를 더 높일 수 있는 기술이나 힌트를 발견 할 수 있습니다. 예를 들어 양상을 학습, 학습률 감소, 데이터 확장 등이 정확도 향상에 공헌하고 있지요. 특히 데이터 확장은 손쉬운 방법이면서도 정확도 개선에 아주 효과적입니다.

**데이터 확장** data augmentation은 입력 이미지(훈련 이미지)를 알고리즘을 동원해 ‘인위적’으로 확장합니다. [그림 8-4]와 같이 입력 이미지를 회전하거나 세로로 이동하는 등 미세한 변화를 주어 이미지의 개수를 늘리는 것이죠. 이는 데이터가 몇 개 없을 때 특히 효과적인 수단입니다.

그림 8-4 데이터 확장의 예



데이터 확장은 [그림 8-4] 같은 변형 외에도 다양한 방법으로 이미지를 확장할 수 있습니다. 예를 들어 이미지 일부를 잘라내는 **crop**이나 좌우를 뒤집는 **flip\*** 등이 있겠죠. 일반적인 이미지에는 밝기 등의 외형 변화나 확대·축소 등의 스케일 변화도 효과적입니다. 어쨌든 데이터 확장을 동원해 훈련 이미지의 개수를 늘릴 수 있다면 딥러닝의 인식 수준을 개선할 수 있습니다. 이것은 쉬운 ‘트릭’이라 가볍게 생각할지도 모르지만, 멋진 결과를 가져오는 경우가 많습니다. 이 책에서는 데이터 확장은 구현하지 않습니다만, 보다시피 어렵지 않은 방식이니 흥미가 있는 분은 한 번 도전해보세요.

### 8.1.3 깊게 하는 이유

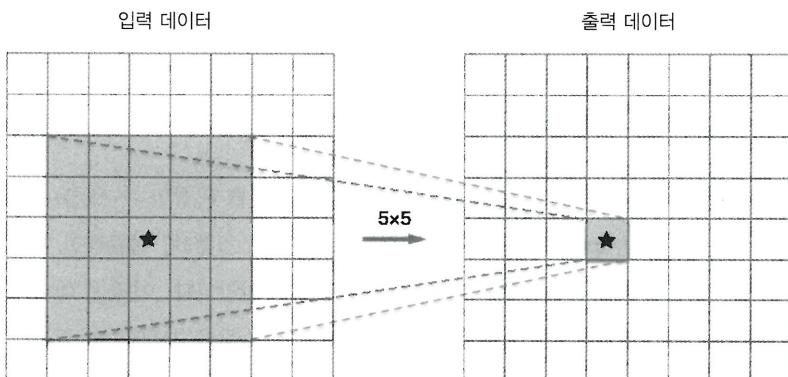
‘층을 깊게 하는 것’이 왜 중요한가에 대한 이론적인 근거는 아직 많이 부족한 것이 사실입니다. 그래도 지금까지의 연구와 실험 결과를 바탕으로 설명할 수 있는 것은 몇 가지 있습니다(다소 직관적이기는 하지만). 이번 절에서는 ‘층을 깊게 하는 것’의 중요성에 대해서, 이를 뒷받침하는 데 이터와 설명을 몇 가지 소개하겠습니다.

\* flip은 이미지의 대칭성을 고려하지 않아도 되는 경우에만 쓸 수 있습니다.

우선 층을 깊게 하는 것의 중요성은 ILSVRC로 대표되는 대규모 이미지 인식 대회의 결과에서 파악할 수 있습니다(자세한 내용은 다음 절을 참고하세요). 이 대회에서 최근 상위를 차지한 기법 대부분은 딥러닝 기반이며, 그 경향은 신경망을 더 깊게 만드는 방향으로 가고 있습니다. 층의 깊이에 비례해 정확도가 좋아지는 것이죠.

이어서 층을 깊게 할 때의 이점을 설명하겠습니다. 그 이점 하나는 신경망의 매개변수 수가 줄어든다는 것입니다. 층을 깊게 한 신경망은 깊지 않은 경우보다 적은 매개변수로 같은 (혹은 그 이상) 수준의 표현력을 달성할 수 있습니다. 합성곱 연산에서의 필터 크기에 주목해 생각해 보면 쉽게 이해될 겁니다. 예를 하나 볼까요? [그림 8-5]는  $5 \times 5$  필터로 구성된 합성곱 계층입니다.

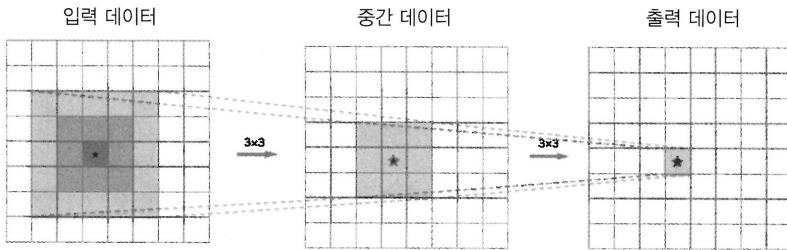
그림 8-5  $5 \times 5$  합성곱 연산의 예



여기에서 주목할 점은 출력 데이터의 각 노드가 입력 데이터의 어느 영역으로부터 계산되었느냐는 것입니다. 당연하지만 [그림 8-5]의 예에서는 각각의 출력 노드는 입력 데이터의  $5 \times 5$  크기 영역에서 계산됩니다.

이어서 [그림 8-6]처럼  $3 \times 3$ 의 합성곱 연산을 2회 반복하는 경우를 생각해봅시다. 이 경우 출력 노드 하나는 중간 데이터의  $3 \times 3$  영역에서 계산됩니다. 그럼 중간 데이터의  $3 \times 3$  영역은 그 전 입력 데이터의 어느 영역에서 계산될까요? [그림 8-6]을 잘 보면  $5 \times 5$  크기의 영역에서 계산되어 나오는 것을 알 수 있지요. 즉, [그림 8-6]의 출력 데이터는 입력 데이터의  $5 \times 5$  영역을 ‘보고’ 계산하게 됩니다.

그림 8-6  $3 \times 3$ 의 합성곱 계층을 2회 반복한 예



$5 \times 5$ 의 합성곱 연산 1회는  $3 \times 3$ 의 합성곱 연산을 2회 수행하여 대체할 수 있습니다. 게다가 전자의 매개변수 수가 25개( $5 \times 5$ )인 반면, 후자는 총 18개( $2 \times 3 \times 3$ )이며, 매개변수 수는 총을 반복할수록 적어집니다. 그리고 그 개수의 차이는 층이 깊어질수록 커집니다. 예를 들어  $3 \times 3$ 의 합성곱 연산을 3회 반복하면 매개변수는 모두 27개가 되지만, 같은 크기의 영역을 1회의 합성곱 연산으로 ‘보기’ 위해서는  $7 \times 7$  크기의 필터, 즉 매개변수 49개가 필요합니다.

**NOTE** 작은 필터를 겹쳐 신경망을 깊게 할 때의 장점은 매개변수 수를 줄여 넓은 수용 영역(receptive field)을 소화할 수 있다는 데 있습니다(수용 영역은 뉴런에 변화를 일으키는 국소적인 공간 영역입니다). 게다가 층을 거듭하면서 ReLU 등의 활성화 함수를 합성곱 계층 사이에 끼움으로써 신경망의 표현력이 개선됩니다. 이는 활성화 함수가 신경망에 ‘비선형’ 힘을 가하고, 비선형 함수가 겹치면서 더 복잡한 것도 표현할 수 있게 되기 때문입니다.

학습의 효율성도 층을 깊게 하는 것의 이점입니다. 층을 깊게 함으로써 학습 데이터의 양을 줄여 학습을 고속으로 수행할 수 있다는 뜻입니다. 이를 (직감적으로) 이해하려면 “7.6 CNN 시각화하기”에서의 설명을 생각하면 좋을 것입니다. 7.6절에서 CNN의 합성곱 계층이 정보를 계층적으로 추출하고 있음을 설명했습니다. 앞단의 합성곱 계층에서는 에지 등의 단순한 패턴에 뉴런이 반응하고 층이 깊어지면서 텍스처와 사물의 일부와 같이 점차 더 복잡한 것에 반응한다고 설명했죠.

그런 네트워크 계층 구조를 기억해두고, ‘개’를 인식하는 문제를 생각해봅시다. 이 문제를 얇은 신경망에서 해결하려면 합성곱 계층은 개의 특징 대부분을 한 번에 ‘이해’해야 합니다. 견종도 다양하고 어느 각도에서 찍은 사진이냐 따라 완전히 다르게 보일 수 있습니다. 그래서 개의 특징을 이해하려면 변화가 풍부하고 많은 학습 데이터가 필요하고, 결과적으로 학습 시간이 오래 걸립니다.

그러나 신경망을 깊게 하면 학습해야 할 문제를 계층적으로 분해할 수 있습니다. 각 층이 학습해야 할 문제를 더 단순한 문제로 대체할 수 있는 것이죠. 예를 들어 처음 층은 에지 학습에 전념하여 적은 학습 데이터로 효율적으로 학습할 수 있습니다. 개가 등장하는 이미지보다 에지를 포함한 이미지는 많고, 에지의 패턴은 개라는 패턴보다 구조가 훨씬 간단하기 때문이죠.

또, 층을 깊게 하면 정보를 계층적으로 전달할 수 있다는 점도 중요합니다. 예를 들어 에지를 추출한 층의 다음 층은 에지 정보를 쓸 수 있고, 더 고도의 패턴을 효과적으로 학습하리라 기대할 수 있습니다. 즉, 층을 깊이 함으로써 각 층이 학습해야 할 문제를 ‘풀기 쉬운 단순한 문제’로 분해할 수 있어 효율적으로 학습하리라 기대할 수 있습니다.

이상이 층을 깊게 하는 것이 왜 중요한가에 대한 보충 설명입니다. 단, 최근 일어나고 있는 층의 심화는 층이 깊어도 제대로 학습할 수 있도록 해주는 새로운 기술과 환경(빅데이터와 컴퓨터 연산 능력 등)이 뒷받침되어 나타난 현상임을 강조해두고자 합니다.

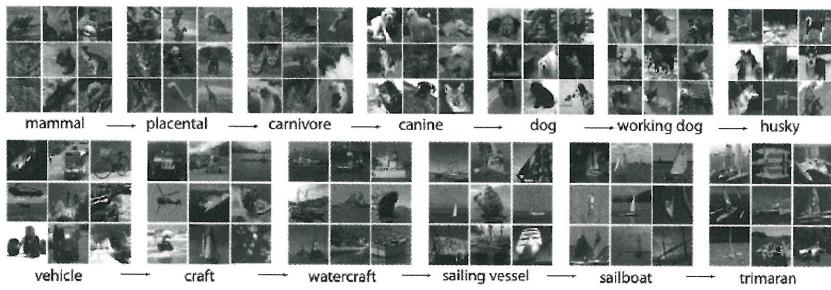
## 8.2 딥러닝의 초기 역사

딥러닝이 지금처럼 큰 주목을 받게 된 계기는 이미지 인식 기술을 겨루는 장<sup>場</sup>인 ILSVRC<sup>ImageNet</sup> Large Scale Visual Recognition Challenge의 2012년 대회입니다. 그해의 대회에서 딥러닝에 기초한 기법, 일명 AlexNet이 압도적인 성적으로 우승하면서 그동안의 이미지 인식에 대한 접근법을 뿌리부터 뒤흔들었습니다. 이 2012년 딥러닝의 역습이 전환점이 되어, 그 후로는 대회의 주역은 항상 딥러닝이었습니다. 이번 절에서는 ILSVRC 대회를 축으로 최근의 딥러닝 트렌드를 살펴보겠습니다.

### 8.2.1 이미지넷

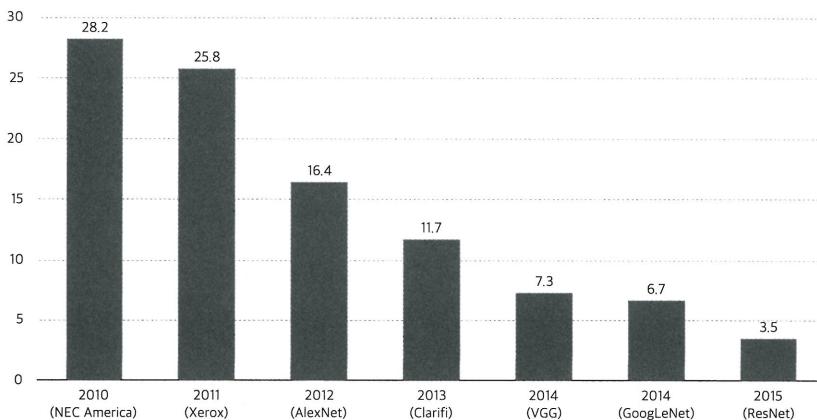
이미지넷<sup>ImageNet</sup>[25]은 100만 장이 넘는 이미지를 담고 있는 데이터셋입니다. [그림 8-7]과 같이 다양한 종류의 이미지를 포함하며 각 이미지에는 레이블(클래스 이름)이 붙어 있습니다. 매년 열리는 ILSVRC는 이 거대한 데이터셋을 사용하여 자웅을 겨루는 천하제일 이미지 인식 기술 대회인 셈이죠.

그림 8-7 대규모 데이터셋 ImageNet의 데이터들<sup>[25]</sup>



ILSVRC 대회에는 시험 항목이 몇 가지 있는데, 그중 하나가 **분류** classification입니다. 분류 부문에서는 1,000개의 클래스를 제대로 분류하는지를 겨릅니다. 그럼 최근 ILSVRC의 분류 시험 결과를 살펴보시죠. [그림 8-8]은 2010년부터 최근까지 ILSVRC의 분류 부분 우승팀의 성적입니다. 여기에서는 **톱-5 오류** top-5 error를 막대 그래프로 나타냈습니다. 톱-5 오류란 확률이 가장 높다고 생각하는 후보 클래스 5개 안에 정답이 포함되지 않은, 즉 5개 모두가 틀린 비율입니다.

그림 8-8 ILSVRC 최우수 팀의 성적 추이 : 세로축은 오류율, 가로축은 연도, 가로축의 괄호 안은 팀 이름(또는 기법 이름)  
이미지넷 분류 톱-5 오류



[그림 8-8]에서 주목할 점은 2012년 이후 선두는 항상 딥러닝 방식이라는 것입니다. 실제로 2012년의 AlexNet이 오류율을 크게 낮췄고, 그 후 딥러닝을 활용한 기법이 꾸준히 정확도를 개선해왔습니다. 특히 2015년에는 150층이 넘는 심층 신경망인 ResNet이 오류율을 3.5%까지 낮췄습니다. 덧붙여서, 이 결과는 일반적인 인간의 인식 능력을 넘어섰다고 합니다.

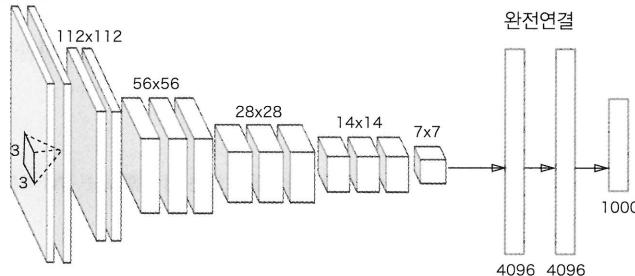
최근 몇 년 빼어난 성적을 거두고 있는 딥러닝 중에서도 VGG, GoogLeNet, ResNet은 특히 유명하며, 다양한 딥러닝 분야에서 활용됩니다. 이어지는 절들에서는 이 세 가지 유명 신경망을 간단히 소개드리겠습니다.

## 8.2.2 VGG

VGG는 합성곱 계층과 풀링 계층으로 구성되는 ‘기본적’인 CNN입니다. 다만, [그림 8-9]와 같이 비중 있는 층(합성곱 계층, 완전연결 계층)을 모두 16층(혹은 19층)으로 심화한 게 특징입니다(층의 깊이에 따라서 ‘VGG16’과 ‘VGG19’로 구분하기도 합니다).

그림 8-9 VGG<sup>[22]</sup>

224x224



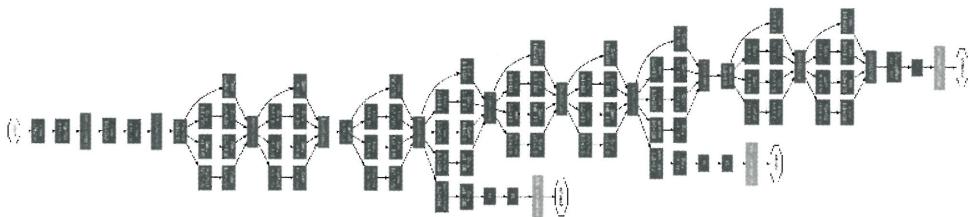
VGG에서 주목할 점은  $3 \times 3$ 의 작은 필터를 사용한 합성곱 계층을 연속으로 거친다는 것입니다. 그림에서 보듯 합성곱 계층을 2~4회 연속으로 풀링 계층을 두어 크기를 절반으로 줄이는 처리를 반복합니다. 그리고 마지막에는 완전연결 계층을 통과시켜 결과를 출력합니다.

**NOTE** VGG는 2014년 대회에서 2위에 올랐습니다. 성능 면에서는 1위인 GoogLeNet에 뒤지지만, VGG는 구성이 간단하여 응용하기 좋습니다. 그래서 많은 기술자가 VGG 기반의 신경망을 즐겨 사용합니다.

### 8.2.3 GoogLeNet

GoogLeNet의 구성은 [그림 8-10]과 같습니다. 그림의 사각형이 합성곱 계층과 풀링 계층 등의 계층을 나타냅니다.

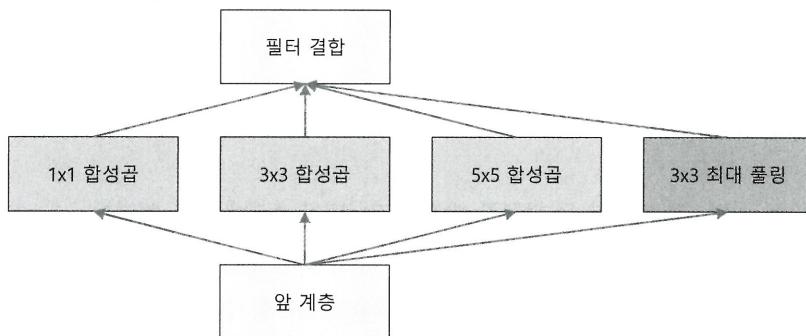
그림 8-10 GoogLeNet<sup>[23]</sup>



그림을 보면 구성이 매우 복잡해 보이는데, 기본적으로는 지금까지 보아온 CNN과 다르지 않습니다. 단, GoogLeNet은 세로 방향 깊이뿐 아니라 가로 방향도 깊다는 점이 특징입니다.

GoogLeNet에는 가로 방향에 ‘폭’이 있습니다. 이를 인셉션 구조라 하며, 그 기반 구조는 [그림 8-11]과 같습니다.

그림 8-11 GoogLeNet의 인셉션 구조<sup>[23]</sup>



인셉션 구조는 [그림 8-11]과 같이 크기가 다른 필터(와 풀링)를 여러 개 적용하여 그 결과를 결합합니다. 이 인셉션 구조를 하나의 빌딩 블록(구성요소)으로 사용하는 것이 GoogLeNet의 특징인 것이죠. 또 GoogLeNet에서는  $1 \times 1$  크기의 필터를 사용한 합성곱 계층을 많은 곳에서 사용합니다. 이  $1 \times 1$ 의 합성곱 연산은 채널 쪽으로 크기를 줄이는 것으로, 매개변수 제거와 고속 처리에 기여합니다(자세한 것은 원논문<sup>[23]</sup>을 참고하세요).

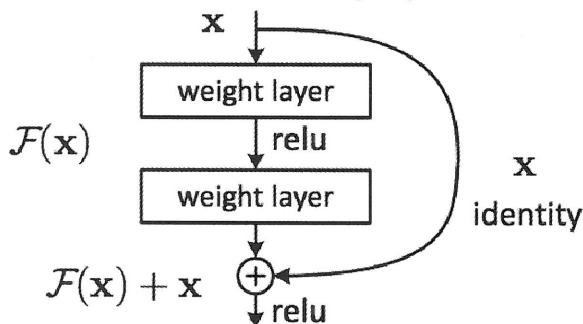
## 8.2.4 ResNet

**ResNet**은 마이크로소프트의 팀이 개발한 네트워크입니다. 그 특징은 지금까지 보다 층을 더 깊게 할 수 있는 특별한 ‘장치’에 있습니다.

지금까지 층을 깊게 하는 것이 성능 향상에 중요하다는 건 알고 있었습니다. 그러나 딥러닝의 학습에서는 층이 지나치게 깊으면 학습이 잘 되지 않고, 오히려 성능이 떨어지는 경우도 많습니다. ResNet에서는 그런 문제를 해결하기 위해서 **스킵 연결** skip connection을 도입합니다. 이 구조가 층의 깊이에 비례해 성능을 향상시킬 수 있게 한 핵심입니다(물론 층을 깊게 하는 데는 여전히 한계가 있습니다).

스킵 연결이란 [그림 8-12]와 같이 입력 데이터를 합성곱 계층을 건너뛰어 출력에 바로 더하는 구조를 말합니다.

그림 8-12 ResNet의 구성요소<sup>[24]</sup> : ‘weight layer’는 합성곱 계층을 말한다.

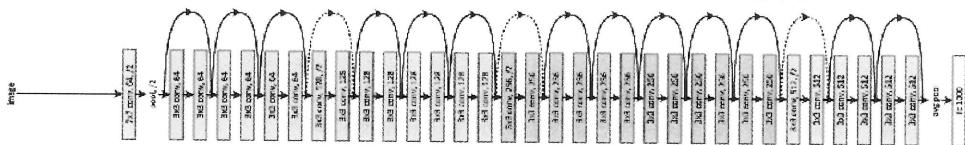


[그림 8-12]에서는 입력  $x$ 를 연속한 두 합성곱 계층을 건너뛰어 출력에 바로 연결합니다. 이 단축 경로가 없었다면 두 합성곱 계층의 출력이  $F(x)$ 가 되나, 스kip 연결로 인해  $F(x) + x$ 가 되는 게 핵심입니다. 스kip 연결은 층이 깊어져도 학습을 효율적으로 할 수 있도록 해주는데, 이는 역전파 때 스kip 연결이 신호 감쇠를 막아주기 때문입니다.

**NOTE** 스kip 연결은 입력 데이터를 ‘그대로’ 흘리는 것으로, 역전파 때도 상류의 기울기를 그대로 하류로 보냅니다. 여기에서의 핵심은 상류의 기울기에 아무런 수정도 가하지 않고 ‘그대로’ 흘린다는 것이죠. 그래서 스kip 연결로 기울기가 작아지거나 지나치게 커질 걱정 없이 앞 층에 ‘의미 있는 기울기’가 전해지리라 기대할 수 있습니다. 층을 깊게 할수록 기울기가 작아지는 소실 문제를 이 스kip 연결이 줄여주는 것입니다.

ResNet은 먼저 설명한 VGG 신경망을 기반으로 스kip 연결을 도입하여 층을 깊게 했습니다. 그 결과는 [그림 8-13]처럼 됩니다.

그림 8-13 ResNet<sup>[24]</sup> : 블록이  $3 \times 3$ 인 합성곱 계층에 대응. 층을 건너뛰는 스kip 연결이 특징이다.



[그림 8-13]과 같이 ResNet은 합성곱 계층을 2개 층마다 건너뛰면서 층을 깊게 합니다. 실험 결과 150층 이상으로 해도 정확도가 오르는 모습을 확인할 수 있습니다. 그리고 ILSVRC 대회에서는 톱-5 오류율이 겨우 3.5%라는 경이적인 결과를 냈습니다.

**NOTE**\_ 이미지넷이 제공하는 거대한 데이터셋으로 학습한 가중치 값들은 실제 제품에 활용해도 효과적이고, 또 많이들 그렇게 이용하고 있습니다. 이를 **전이 학습**<sup>transfer learning</sup>이라고 해서, 학습된 가중치(혹은 그 일부)를 다른 신경망에 복사한 다음, 그 상태로 재학습을 수행합니다. 예를 들어 VGG와 구성이 같은 신경망을 준비하고, 미리 학습된 가중치를 초기값으로 설정한 후, 새로운 데이터셋을 대상으로 재학습(fine tuning)을 수행합니다. 전이 학습은 보유한 데이터셋이 적을 때 특히 유용한 방법입니다.

### 8.3 더 빠르게(딥러닝 고속화)

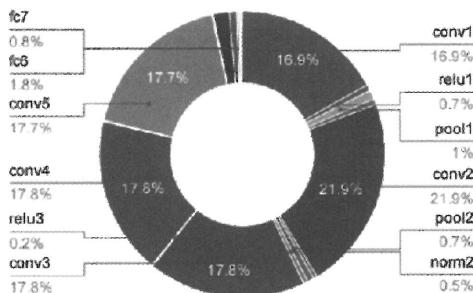
빅데이터와 네트워크의 발전으로 딥러닝에서는 대량의 연산을 수행해야 합니다. 과거에는 주로 CPU가 계산을 담당했으나, CPU만으로 딥러닝을 처리하기는 부족한 게 현실입니다. 실제로 주위를 둘러보면 딥러닝 프레임워크 대부분은 **GPU**<sup>Graphics Processing Unit</sup>를 활용해 대량의 연산을 고속으로 처리할 수 있습니다. 최근 프레임워크에서는 학습을 복수의 GPU와 여러 기기로 분산 수행하기 시작했습니다. 이번 절에서는 딥러닝의 고속화에 관해 이야기해보려 합니다(참고로, 우리의 딥러닝 구현은 8.1절에서 마무리하고, GPU 대응 등 여기에서 설명하는 고속화는 적용하지 않습니다).

### 8.3.1 풀어야 할 숙제

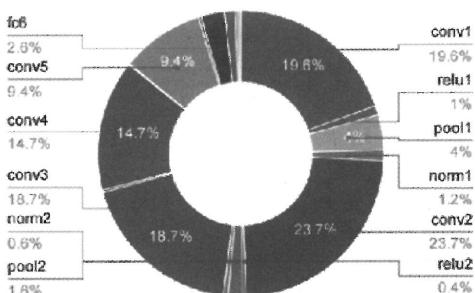
딥러닝의 고속화 얘기를 시작하기 앞서, 딥러닝에서는 어떠한 처리에 시간이 소요되는지를 보겠습니다. [그림 8-14]는 AlexNet의 forward 처리(순전파)에서 각 층이 소비하는 시간을 원 그래프로 보여줍니다.

그림 8-14 AlexNet의 forward 처리 시 각 층의 시간 비율 : 왼쪽이 GPU, 오른쪽이 CPU를 사용한 경우. 'conv'는 합성곱 계층, 'pool'은 풀링 계층, 'fc'는 완전연결 계층, 'norm'은 정규화 계층이다.<sup>[26]</sup>

GPU Forward Time Distribution



CPU Forward Time Distribution



그림에서 보듯 AlexNet에서는 오랜 시간을 합성곱 계층에서 소요합니다. 실제로 합성곱 계층의 처리 시간을 다 더하면 GPU에서는 전체의 95%, CPU에서는 전체의 89%까지 달하지요! 그래서 합성곱 계층에서 이뤄지는 연산을 어떻게 고속으로 효율적으로 하느냐가 딥러닝의 과제입니다. [그림 8-14]는 추론 때의 결과지만, 학습 시에도 마찬가지로 합성곱 계층에서 많은 시간을 소비하게 됩니다.

**NOTE** 합성곱 계층에서 수행하는 연산은 “7.2 합성곱 계층”에서 설명했듯이, 결국 ‘단일 곱셈–누산’입니다. 그래서 딥러닝 고속화라는 주제는 대량의 ‘단일 곱셈–누산’을 어떻게 고속으로 효율적으로 계산하느냐는 것입니다.

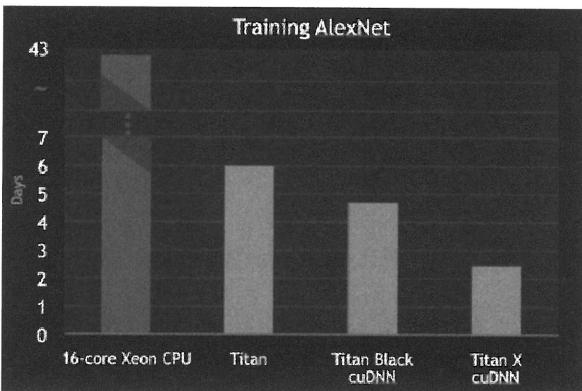
### 8.3.2 GPU를 활용한 고속화

GPU는 원래 그래픽 전용 보드에 이용해왔습니다. 그러나 최근에는 그래픽 처리뿐 아니라 범용 수치 연산에도 이용합니다. GPU는 병렬 수치 연산을 고속으로 처리할 수 있으니, 그 압도적인 힘을 다양한 용도로 활용하자는 것이 **GPU 컴퓨팅**의 목적이죠. 이처럼 GPU로 범용 수치

연산을 수행하는 것을 GPU 컴퓨팅이라고 합니다.

딥러닝에서는 대량의 단일 곱셈–누산(또는 큰 행렬의 곱)을 수행해야 합니다. 이러한 대량 행렬 연산은 GPU의 특기죠(반대로 CPU는 연속적인 복잡한 계산을 잘 처리합니다). 그래서 딥러닝 연산에서는 GPU를 이용하면 CPU만 쓸 때보다 놀라울 정도로 빠르게 결과를 얻을 수 있습니다. 그렇다면 과연 GPU로 어느 정도까지 빨라지는지 예를 보여드리겠습니다. [그림 8-15]는 AlexNet의 학습 시간을 CPU와 GPU에서 비교한 결과입니다.

그림 8-15 AlexNet의 학습 시간을 ‘16코어 제온 CPU’와 엔비디아 ‘타이탄 GPU’에서 비교한 결과<sup>27)</sup>



그림과 같이 CPU에서는 40여 일이나 걸리지만 GPU로는 6일까지 단축되었습니다. 또, cuDNN이라는 딥러닝에 최적화된 라이브러리를 이용하면 더욱 빨라짐을 확인할 수 있습니다.

GPU는 주로 엔비디아와 AMD, 두 회사가 제공합니다. 두 회사의 GPU 모두 범용 수치 연산에 이용할 수는 있지만, 딥러닝과 더 ‘친한’ 쪽은 아직까지는 엔비디아입니다. 실제로 대부분의 딥러닝 프레임워크는 엔비디아 GPU에서만 혜택을 받을 수 있습니다. 엔비디아의 GPU 컴퓨팅용 통합 개발 환경인 **CUDA**를 사용하기 때문이죠. [그림 8-15]에 등장하는 **cuDNN**은 CUDA 위에서 동작하는 라이브러리로, 딥러닝에 최적화된 함수 등이 구현되어 있습니다.

**NOTE** 합성곱 계층에서 행하는 연산은 im2col을 이용해 큰 행렬의 곱으로 변환할 수 있었습니다. 이러한 im2col의 방식은 GPU로 구현하기에도 적합합니다. GPU는 ‘작은’ 단위로 계산하기보다는 큰 덩어리를 한 번에 계산하는 데 유리하기 때문이죠. 즉, im2col로 거대한 행렬의 곱으로 한 번에 계산하여 GPU의 잠재력을 끌어내는 것입니다.

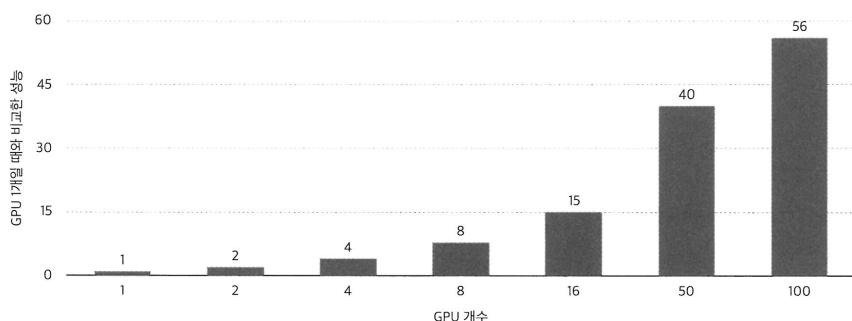
### 8.3.3 분산 학습

GPU로 딥러닝 연산을 꽤 가속할 수 있지만, 그래도 심층 신경망에서는 학습에 며칠 혹은 몇 주가 걸리기도 합니다. 그리고 지금까지 살펴본 것처럼 딥러닝은 많은 시행착오를 동반합니다. 뛰어난 신경망을 만들려면 시험을 수없이 반복해야 하고, 그러려면 1회 학습에 걸리는 시간을 최대한 단축하고 싶다는 요구가 필연적으로 생겨납니다. 그래서 딥러닝 학습을 수평 확장<sup>scale out</sup>하자는 아이디어(즉, ‘분산 학습’)가 중요해지는 것입니다.

딥러닝 계산을 더욱 고속화하고자 다수의 GPU와 기기로 계산을 분산하기도 합니다. 최근에는 다수의 GPU와 컴퓨터를 이용한 분산 학습을 지원한 딥러닝 프레임워크들이 나타나고 있습니다. 그중에서도 구글의 텐서플로와 마이크로소프트의 CNTK<sup>Computational Network Toolkit</sup>는 분산 학습에 역점을 두고 개발하고 있습니다. 거대한 데이터센터의 저지연 · 고처리량<sup>low latency, high throughput</sup> 네트워크 위에서 이 프레임워크들이 수행하는 분산 학습은 놀라운 효과를 보이고 있습니다.

분산 학습까지 더하면 어느 수준까지 고속화할 수 있을까요? [그림 8-16]은 텐서플로로 알아본 분산 학습의 효과입니다.

그림 8-16 텐서플로의 분산 학습 성능 : 가로는 GPU의 수, 세로는 GPU 1개일 때와 비교한 비율<sup>[28]</sup>



[그림 8-16]에서 보듯 GPU 수가 늘어남에 따라 학습도 빨라집니다. 실제로 여러 기기를 연결하여 GPU를 100개까지 사용하니 하나일 때보다 56배가 빨라졌습니다! 7일짜리 작업을 불과 3시간 만에 끝낸다는 것으로, 분산 학습의 놀라운 효과를 증명하고 있습니다.

분산 학습에서도 ‘계산을 어떻게 분산시키느냐’는 뽑시 어려운 문제입니다. 컴퓨터 사이의 통신과 데이터 동기화 등, 쉽게 해결할 수 없는 문제를 얼마든지 끌어안고 있습니다. 그래서 이런 어려운 문제는 텐서플로 같은 뛰어난 프레임워크에 맡기는 것이 좋습니다. 여기에서는 분산 학습의 상세 내용은 다루지 않겠습니다. 분산 학습 관련 기술적인 내용은 텐서플로 기술 논문<sup>[29]</sup> 등을 참고하세요.

### 8.3.4 연산 정밀도와 비트 줄이기

계산 능력 외에도 메모리 용량과 버스 대역폭 등이 딥러닝 고속화에 병목이 될 수 있습니다. 메모리 용량 면에서는 대량의 가중치 매개변수와 중간 데이터를 메모리에 저장해야 한다는 것을 생각해야 합니다. 버스 대역폭 면에서는 GPU(혹은 CPU)의 버스를 흐르는 데이터가 많아져 한계를 넘어서면 병목이 됩니다. 이러한 경우를 고려하면 네트워크로 주고받는 데이터의 비트 수는 최소로 만드는 것이 바람직합니다.

컴퓨터에서는 주로 64비트나 32비트 부동소수점 수를 사용해 실수를 표현합니다. 많은 비트를 사용할수록 계산 오차는 줄어듭니다만, 그만큼 계산에 드는 비용과 메모리 사용량이 늘고 버스 대역폭에 부담을 줍니다.

다행히 딥러닝은 높은 수치 정밀도(수치를 몇 비트로 표현하느냐)를 요구하지 않습니다. 이는 신경망의 중요한 성질 중 하나로, 신경망의 견고성에 따른 특성입니다. 예를 들어 신경망은 입력 이미지에 노이즈가 조금 섞여 있어도 출력 결과가 잘 달라지지 않는 강건함을 보여주죠. 이런 견고성 덕분에 신경망을 흐르는 데이터를 ‘퇴화’시켜도 출력에 주는 영향은 적습니다.

컴퓨터에서 실수를 표현하는 방식으로 **32비트 단정밀도** single-precision과 **64비트 배정밀도** double-precision 부동소수점 등의 포맷이 있지만, 지금까지의 실험으로는 딥러닝은 **16비트 반정밀도** half-precision만 사용해도 학습에 문제가 없다고 알려져 있습니다.<sup>[30]</sup> 실제로 엔비디아의 2016년 GPU인 **파스칼** Pascal 아키텍처는 이 포맷을 지원하여, 이제는 반정밀도 부동소수점이 표준적으로 이용되리라 생각합니다.

**NOTE**\_ 엔비디아의 맥스웰 Maxwell 세대 GPU는 반정밀도 부동소수점 수를 스토리지(데이터를 저장하는 기능)로 지원하고 있었지만, 연산 자체는 16비트로 수행하지 않았습니다. 이것이 파스칼 세대에 와서 연산 역시 16비트로 하기 시작하여, 이전 세대보다 2배 정도 빨라졌습니다.\*

이 책은 지금까지 딥러닝을 구현하며 수치 정밀도에는 특별히 주의하지 않았습니다. 그럼 몇 가지를 짚어봅시다. 우선 파이썬에서는 일반적으로 64비트 배정밀도 부동소수점 수를 사용합니다. 하지만 넘파이는 16비트 반정밀도 부동소수점도 지원하며, 이를 사용해도 정확도가 떨어지지 않는 것을 쉽게 확인할 수 있습니다(단, 스토리지로서 16비트라는 틀이 있을 뿐 연산 자체는 16비트로 수행하지 않습니다). 관심 있는 분은 ch08/half\_float\_network.py를 참고하세요.

딥러닝의 비트 수를 줄이는 연구가 몇 가지 진행되고 있습니다. 최근에는 가중치와 중간 데이터를 1비트로 표현하는 〈Binarized Neural Networks〉라는 방법도 등장했습니다.<sup>[31]</sup> 딥러닝을 고속화하기 위해 비트를 줄이는 기술은 앞으로 주시해야 할 분야이며, 특히 딥러닝을 임베디드용으로 이용할 때 중요한 주제입니다.

## 8.4 딥러닝의 활용

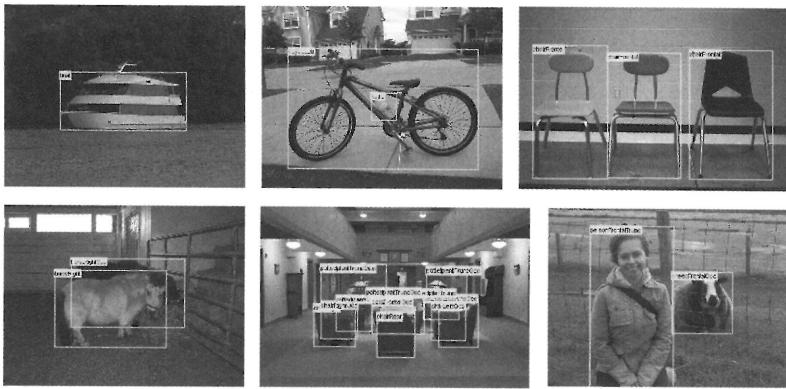
지금까지 딥러닝을 활용한 예를 손글씨 숫자 인식이라는 이미지 분류를 중심으로 살펴봤습니다. 이는 ‘사물 인식’의 한 분야죠. 그러나 딥러닝은 사물 인식뿐 아니라 온갖 문제에 적용할 수 있습니다. 이미지, 음성, 자연어 등 수많은 분야에서 딥러닝은 뛰어난 성능을 발휘합니다. 이번 절에서는 딥러닝이 할 수 있는 것을 컴퓨터 비전 분야를 중심으로 몇 가지 소개하겠습니다.

### 8.4.1 사물 검출

사물 검출은 [그림 8-17]과 같이 이미지 속에 담긴 사물의 위치와 종류(클래스)를 알아내는 기술입니다.

\* 옮긴이\_ AMD GPU 역시 베가(VEGA) 모델부터는 반정밀도 연산을 지원합니다. 또한 게임 콘솔인 PS4 Pro뿐 아니라, 〈애플의 자체 제작 GPU 살펴보기(<http://macnews.tistory.com/5200>)〉에 따르면 애플이 아이폰7에 처음 사용한 A10 퓨전 칩의 GPU도 반정밀도 연산을 지원합니다. 딥러닝이 생활 깊숙이 파고들면서 모바일 기기까지도 본격적인 딥러닝 대응을 시작했다는 빙증일 것입니다.

그림 8-17 사물 검출의 예<sup>[34]</sup>

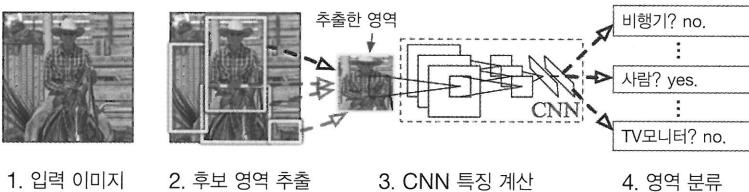


이 그림에서 보듯 사물 검출은 사물 인식보다 어려운 문제입니다. 지금까지 본 사물 인식은 이미지 전체를 대상으로 했는데, 사물 검출에서는 이미지 어딘가에 있을 사물의 위치까지 알아내야 합니다. 게다가 한 이미지에 여러 사물이 존재할 수도 있습니다.

이런 사물 검출 문제에 CNN을 기반으로 한 기법이 몇 가지 제안되었습니다. 이 기법들이 발군의 성능을 보여 사물 검출에도 딥러닝이 효과적임을 시사하고 있습니다.

자, CNN을 이용하여 사물 검출을 수행하는 방식은 몇 가지가 있는데, 그중에서도 **R-CNN**<sup>Regions with Convolutional Neural Network<sup>[35]</sup>이 유명합니다. [그림 8-18]은 R-CNN의 처리 흐름입니다.</sup>

그림 8-18 R-CNN의 처리 흐름<sup>[35]</sup>



R-CNN 그림에서 주목할 곳은 ‘2. 후보 영역 추출’과 ‘3. CNN 특징 계산’입니다. 먼저 사물이 위치한 영역을 (어떤 방법으로) 찾아내고, 추출한 각 영역에 CNN을 적용하여 클래스를 분류하는 것이죠. 여기서 이미지를 사각형으로 변형하거나 분류할 때 서포트 벡터 머신(SVM)을 사용하는 등 실제 처리 흐름은 다소 복잡하지만, 큰 틀에서는 이 두 가지 처리(후보 영역 추출과

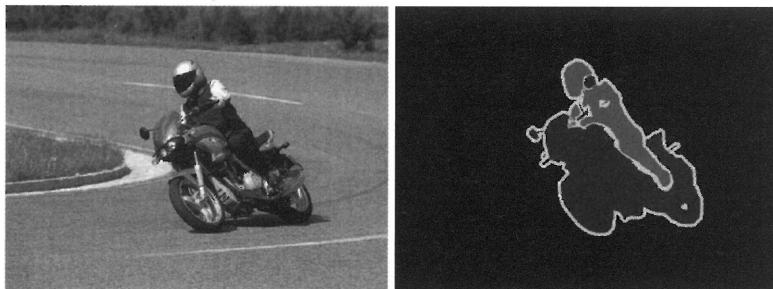
CNN 특징 계산)로 구성됩니다.

후보 영역 추출(사물처럼 보이는 물체를 찾아 처리)에는 컴퓨터 비전 분야에서 발전해온 다양한 기법을 사용할 수 있고, R-CNN 논문에서는 Selective Search 기법을 사용했습니다. 최근에는 이 후보 영역 추출까지 CNN으로 처리하는 **Faster R-CNN**<sup>[36]</sup> 기법도 등장했습니다. Faster R-CNN은 모든 일을 하나의 CNN에서 처리하기 때문에 아주 빠릅니다.

### 8.4.2 분할

**분할** [segmentation]란 이미지를 픽셀 수준에서 분류하는 문제입니다. [그림 8-19]와 같이 픽셀 단위로 객체마다 채색된 지도<sup>supervised</sup> 데이터를 사용해 학습합니다. 그리고 추론할 때 입력 이미지의 모든 픽셀을 분류합니다.

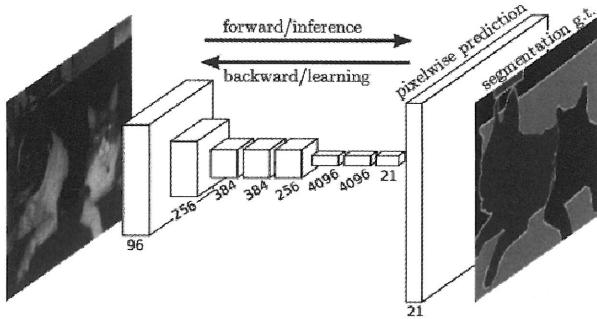
그림 8-19 분할의 예 : 왼쪽이 입력 이미지, 오른쪽이 지도용 이미지<sup>[34]</sup>



지금까지 구현한 신경망은 분류를 이미지 전체를 대상으로 해왔습니다. 이를 픽셀 수준에 적용 하려면 어떻게 하면 될까요?

신경망을 이용해 분할하는 가장 단순한 방법은 모든 픽셀 각각을 추론하는 것입니다. 예를 들어 어떤 직사각형 영역의 중심 픽셀의 클래스를 분류하는 신경망을 만들어서, 모든 픽셀을 대상으로 하나씩 추론 작업을 실행합니다. 짐작한 대로 이런 식으로는 픽셀의 수만큼 forward 처리를 해야 하여 긴 시간이 걸리게 됩니다(정확히는 합성곱 연산에서 많은 영역을 쓸데없이 다시 계산하는 것이 문제가 됩니다). 이러한 낭비를 줄여주는 기법으로 FCN(Fully Convolutional Network)<sup>[37]</sup>이 고안되었습니다. 이는 단 한 번의 forward 처리로 모든 픽셀의 클래스를 분류해주는 놀라운 기법입니다(그림 8-20).

그림 8-20 FCN의 전체 그림<sup>[37]</sup>



Fully Convolutional Network를 직역하면 ‘합성곱 계층만으로 구성된 네트워크’가 됩니다. 일반적인 CNN이 완전연결 계층을 이용하는 반면, FCN은 이 완전연결 계층을 ‘같은 기능을 하는 합성곱 계층’으로 바꿉니다. 사물 인식에서 사용한 신경망의 완전연결 계층에서는 중간 데이터의 공간 볼륨(다차원 형태)을 1차원으로 변환하여 한 줄로 늘어선 노드들이 처리했으나, FCN에서는 공간 볼륨을 유지한 채 마지막 출력까지 처리할 수 있습니다.

FCN은 [그림 8-20]에서 보듯 마지막에 공간 크기를 확대하는 처리를 도입했다는 것도 특징입니다. 이 확대 처리로 인해 줄어든 중간 데이터를 입력 이미지와 같은 크기까지 단번에 확대할 수 있습니다. FCN의 마지막에 수행하는 확대는 이중 선형 보간 bilinear interpolation에 의한 선형 확대입니다. FCN에서는 이 선형 확대를 역합성곱 deconvolution 연산으로 구현해내고 있습니다(자세한 내용은 FCN 논문<sup>[37]</sup>을 참고하세요).

**NOTE** 완전연결 계층에서는 출력이 모든 입력과 연결됩니다. 이와 같은 구성을 합성곱 계층으로도 구현할 수 있습니다. 가령 입력 크기가  $32 \times 10 \times 10$ (채널 32개, 높이 10, 너비 10)인 데이터에 대한 완전연결 계층은 필터 크기가  $32 \times 10 \times 10$ 인 합성곱 계층으로 대체할 수 있습니다. 만약, 완전연결 계층의 출력 노드가 100개라면 합성곱 계층에서는 기존의  $32 \times 10 \times 10$  필터를 100개 준비하면 완전히 같은 처리를 할 수 있습니다. 이처럼 완전연결 계층은 같은 일을 수행하는 합성곱 계층으로 대체할 수 있습니다.

### 8.4.3 사진 캡션 생성

컴퓨터 비전과 자연어를 융합한 재미있는 연구가 있습니다. [그림 8-21]과 같은 사진을 주면, 그 사진을 설명하는 글(사진 캡션)을 자동으로 생성하는 연구입니다.

그림 8-21 딥러닝으로 사진 캡션을 생성하는 예[38]



Describes without errors

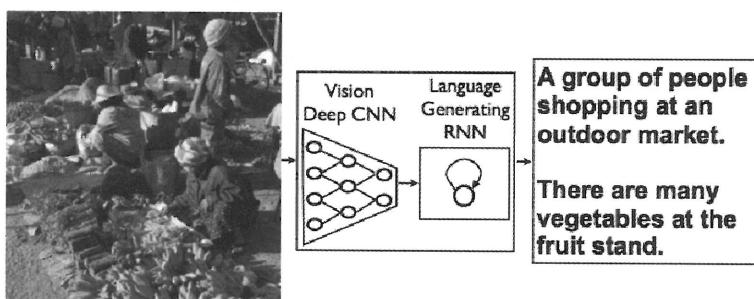
Describes with minor errors

Somewhat related to the image

예를 들어 [그림 8-21]의 첫 번째는 사진만 보고 “비포장도로에서 오토바이를 타는 사람(A person riding a motorcycle on a dirt road)”이라는 문장을 자동으로 생성했습니다. 설명과 사진이 정확히 일치합니다. 오토바이를 타고 있다는 것뿐만이 아니고 거친 비포장도로라는 것도 ‘이해’하고 있다니 놀랍지 않습니까?

딥러닝으로 사진 캡션을 생성하는 방법으로는 NIC(Neural Image Caption) 모델이 대표적입니다. NIC는 [그림 8-22]와 같이 심층 CNN과 자연어를 다루는 순환 신경망(Recurrent Neural Network, RNN)으로 구성됩니다.\* RNN은 순환적 관계를 갖는 신경망으로 자연어나 시계열 데이터 등의 연속된 데이터를 다룰 때 많이 활용합니다.

그림 8-22 NIC의 전체 구성[38]



\* 옮긴이\_ 재귀 신경망으로 쓰기도 합니다.

NIC는 CNN으로 사진에서 특징을 추출하고, 그 특징을 RNN에 넘깁니다. RNN은 CNN이 추출한 특징을 초기값으로 해서 텍스트를 ‘순환적’으로 생성합니다. 여기에서는 더 이상의 상세 기술은 설명하지 않지만, 기본적으로 NIC는 2개의 신경망(CNN과 RNN)을 조합한 간단한 구성입니다. 그래서 놀라울 정도로 정확한 사진 캡션을 만들어내는 것입니다. 또한, 사진이나 자연어와 같은 여러 종류의 정보를 조합하고 처리하는 것을 **멀티모달 처리**<sup>multimodal processing</sup>라고 하여, 최근 주목받는 분야 중 하나입니다.

**NOTE** RNN의 R은 Recurrent(순환적)를 뜻합니다. 여기에서 순환은 신경망의 순환적 네트워크 구조를 말합니다. 이 순환적인 구조로 인해 이전에 생성한 정보에 영향을 받는(바꾸어 말하면, 과거의 정보를 기억하는) 점이 RNN의 특징입니다. 예를 들어 ‘나’라는 단어를 생성한 뒤 ‘잤다’라는 단어를 생성하면 먼저 만든 ‘나’의 영향을 받아 ‘는’이라는 조사가 자동으로 생성되어, 최종적으로 ‘나는 잤다’라는 문장이 완성되는 식입니다. 이처럼 자연어와 시계열 데이터 등 연속성 있는 데이터를 다룰 때 RNN은 과거의 정보를 기억하면서 동작합니다.

## 8.5 딥러닝의 미래

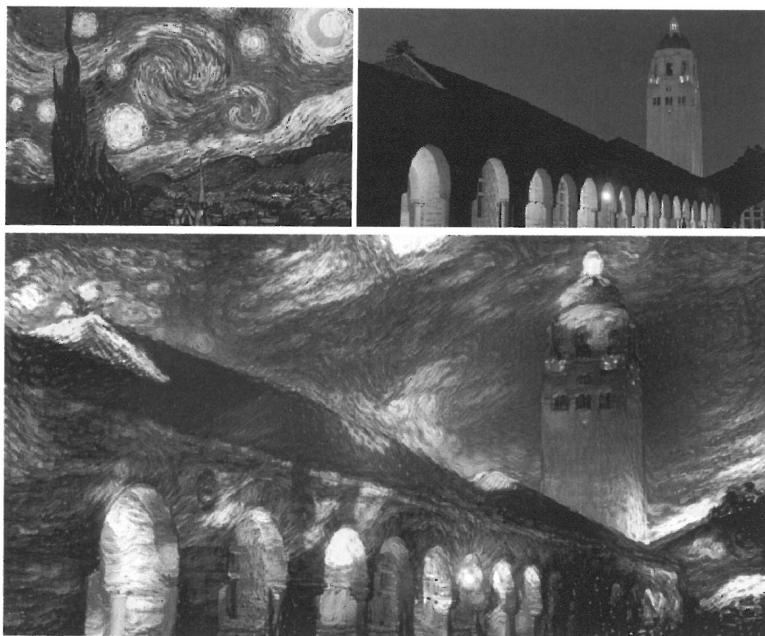
딥러닝은 앞에서 언급한 분야 외에도 여러 분야에서 이용되어 왔습니다. 이번 절에서는 딥러닝의 가능성과 미래를 느낄 만한 연구를 몇 가지 소개하겠습니다.

### 8.5.1 이미지 스타일(화풍) 변환

딥러닝을 활용해 화가처럼 ‘그림을 그리는’ 연구가 있습니다. [그림 8-23]은 두 이미지를 입력해서 새로운 그림을 생성하는 연구입니다. 하나는 ‘콘텐츠 이미지’, 다른 하나는 ‘스타일 이미지’라 부르는데, 이 둘을 조합해 새로운 그림을 그려주죠.

[그림 8-23]과 같이 고흐의 화풍을 콘텐츠 이미지에 적용하도록 지정하면, 이를 기초로 딥러닝이 새로운 그림을 그립니다. 이 기법을 담은 「A Neural Algorithm of Artistic Style」논문<sup>[39]</sup>은 발표되자마자 전 세계에서 많은 이목을 끌었습니다.

그림 8-23 「A Neural Algorithm of Artistic Style」 논문을 구현해 적용한 예 : 왼쪽 위가 '스타일 이미지', 오른쪽 위가 '콘텐츠 이미지', 아래가 새로 생성한 이미지<sup>[40]</sup>



여기에서는 이 연구를 자세히 설명하지는 않겠습니다. 큰 틀만 이야기하면, 이 기술은 네트워크의 중간 데이터가 콘텐츠 이미지의 중간 데이터와 비슷해지도록 학습합니다. 이렇게 하면 입력 이미지를 콘텐츠 이미지의 형태를 흡수할 수 있습니다. 또, 스타일 이미지의 화풍을 흡수하기 위해 '스타일 행렬'이라는 개념을 도입합니다. 그 스타일 행렬의 오차를 줄이도록 학습하여 입력 이미지를 고흐의 화풍과 비슷해지게 만들 수 있는 것입니다.\*

## 8.5.2 이미지 생성

앞의 이미지 스타일 변환 예는 새로운 그림을 생성하려면 이미지 두장을 입력해야 했습니다. 한편 아무런 입력 이미지 없이도 새로운 이미지를 그려내는 연구도 진행 중입니다. 물론 먼저 대량의 이미지를 사용하여 학습하기 하지만, 학습이 끝난 후에는 아무런 입력 이미지 없이도

\* 옮긴이\_ 이런 효과를 적용한 앱들이 이미 많이 등장했습니다. 그중 하나인 Prisma는 이미지 스타일 변환 기법을 대중의 손바닥 위에 올려준 대표적인 앱으로, 애플이 '2016년을 빛낸 최고 앱'으로 뽑았을 만큼 선동적인 인기를 누렸습니다. 이 앱은 합성곱 신경망(CNN) 기술과 인공지능을 조합해 평범한 그림뿐 아니라 동영상에까지 예술가의 흔을 심어줍니다(<http://prisma-ai.com/>).

새로운 그림을 그려냅니다. 가령 딥러닝으로 ‘침실’ 이미지를 무<sup>無</sup>로부터 생성하는 게 가능합니다. [그림 8-24]의 이미지는 DCGAN Deep Convolutional Generative Adversarial Network 기법<sup>[41]</sup>으로 생성한 침실 이미지들입니다.

그림 8-24 DCGAN으로 새롭게 생성한 침실 이미지들<sup>[41]</sup>



[그림 8-24]의 이미지들은 진짜 사진처럼 보일지 모르지만, 모두 DCGAN을 사용해 새롭게 생성한 이미지입니다. 즉, DCGAN이 그런, 아직 아무도 본 적 없는 이미지(학습 데이터에는 존재하지 않는 이미지)이며, 처음부터 새로 생성한 이미지입니다.

자, 진짜와 구분할 수 없는 수준의 이미지를 그리는 DCGAN은 이미지를 생성하는 과정을 모델화합니다. 그 모델을 대량의 이미지(가령 침실이 찍힌 대량의 이미지)를 사용해 학습하고, 학습이 끝나면 그 모델을 이용하여 새로운 그림을 생성할 수 있습니다.

DCGAN도 딥러닝을 사용합니다. DCGAN 기술의 핵심은 생성자 Generator와 식별자 Discriminator로 불리는 2개의 신경망을 이용한다는 점입니다. 생성자가 진짜와 똑같은 이미지를 생성하고 식별자는 그것이 진짜인지(생성자가 생성한 이미지인지, 아니면 실제로 촬영된 이미지인지)를 판정합니다. 그렇게 해서 둘을 겨루도록 학습시켜, 생성자는 더 정교한 가짜 이미지 생성 기술을 학습하고 식별자는 더 정확하게 간파할 수 있는 감정사로 성장하는 것이죠. 이렇게 둘의 능력을 부지런히 갈고닦게 한다는 개념이 GAN Generative Adversarial Network 기술의 재미난 점입니다. 그렇게 절차탁마해서 성장한 생성자는 최종적으로는 진짜와 차각할 정도의 이미지를 그려내는 능력을 기르는 것입니다.

**NOTE** 이전까지 살펴본 기계학습 문제는 **지도 학습**(supervised learning)라는 유형의 문제였습니다. 지도 학습은 손글씨 숫자 인식처럼 이미지 데이터와 정답 레이블을 짹지은 데이터셋을 이용합니다. 그러나 이번 절에서 거론한 문제는 지도용 데이터는 주어지지 않고, 단지 대량의 이미지(이미지의 집합)만 주어집니다. 즉, 지도 없이 스스로 학습하는 **자율 학습**(unsupervised learning) 문제입니다. 자율 학습은 비교적 오래전부터 연구된 분야지만(Deep Belief Network와 Deep Boltzmann Machine이 대표적입니다) 최근에는 그다지 활발하게 연구되지는 않는다는 느낌입니다. 최근 딥러닝을 사용한 DCGAN 등과 같은 기법이 시선을 끌면서, 앞으로 자율 학습도 새로운 도약을 기대할 수 있을지도 모릅니다.

### 8.5.3 자율 주행

사람 대신 컴퓨터가 자동차를 운전하는 **자율 주행** 기술이 눈앞으로 다가왔습니다. 자동차 제조업체뿐 아니라 IT기업과 대학, 연구소 등도 자율 주행 상용화 경쟁에 뛰어들었습니다. 자율 주행은 다양한 기술(주행 경로를 정하는 경로 계획(path plan) 기술과 카메라나 레이저 등의 탐사 기술 등)을 모아 구현하고 있지만, 그중에서도 주위 환경을 올바르게 인식하는 기술이 가장 중요한 문제라고 합니다. 시시각각 변하는 환경과 종횡무진 오가는 다른 차와 사람들을 올바르게 인식하기가 매우 까다롭기 때문이죠.

다양한 환경에서도 안전한 주행 영역을 올바로 인식하게 되면 자율 주행의 실현도 멀지 않을지도 모릅니다. 그리고 최근에는 그런 주위 환경을 인식하는 기술에서 딥러닝이 큰 역할을 해줄 것으로 기대하고 있습니다. 예를 들어 **SegNet**<sup>[42]</sup>이라는 CNN 기반 신경망은 [그림 8-25]와 같이 주변 환경을 정확하게 인식해냅니다.

그림 8-25 딥러닝을 활용한 이미지 분할의 예 : 도로, 차, 건물, 인도 등을 정확하게 인식한다.<sup>[43]</sup>



[그림 8-25]와 같이 입력 이미지를 분할(픽셀 수준에서 판정)하고 있습니다. 결과를 보면 도로와 건물, 보도와 나무, 차량과 오토바이 등을 어느 정도 정확히 판별하고 있습니다. 이런 인식 기술이 딥러닝에 힘입어 향후 한층 정확해지고 빨라지면 자율 주행이 일상에 파고든 영화 속 모습이 현실이 되겠죠.

#### 8.5.4 Deep Q-Network(강화학습)

(자전거를 배울 때처럼) 사람이 시행착오를 겪으며 배우듯 컴퓨터도 시행착오 과정에서 스스로 학습하게 하려는 분야가 있습니다. 이는 ‘가르침’에 의존하는 ‘지도 학습’과는 다른 분야로, 강화학습(reinforcement learning)이라 합니다.

강화학습에서는 에이전트라는 것이 환경에 맞게 행동을 선택하고, 그 행동에 의해서 환경이 변한다는 게 기본적인 틀입니다. 환경이 변화하면 에이전트는 어떠한 보상을 얻습니다. 강화학습의 목적은 더 나은 보상을 받는 쪽으로 에이전트의 행동 지침을 바로잡는 것입니다(그림 8-26).

그림 8-26 강화학습의 기본 틀 : 에이전트는 더 좋은 보상을 받기 위해 스스로 학습한다.

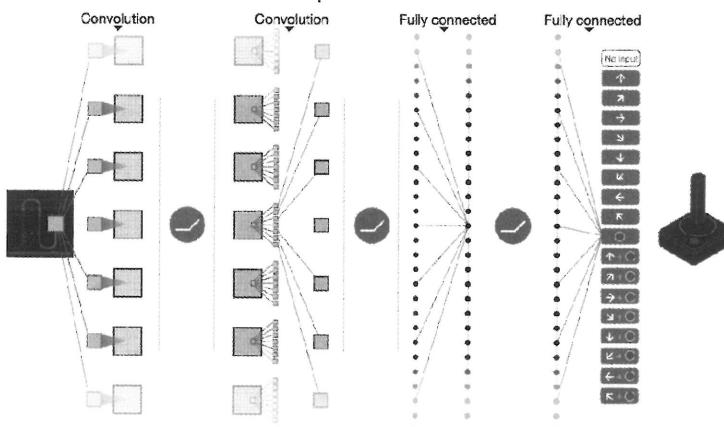


[그림 8-26]은 강화학습의 기본 틀입니다. 여기에서 주의점은 보상은 정해진 것이 아니라 ‘예상 보상’이라는 점입니다. 예를 들면 〈슈퍼 마리오 브라더스〉에서 마리오를 오른쪽으로 이동했을 때 얻는 보상이 항상 명확한 것이 아닙니다. 어떤 상황에서 이동한 것이나에 따라 보상은 천차만별이 될 수 있겠죠. 이런 불명확한 상황에서는 게임 점수(동전을 먹거나 적을 쓰러뜨리는 등)나 게임 종료 등의 명확한 지표로부터 역산해서 ‘예상 보상’을 정해야 합니다. 만약 지도 학습이었다면 행동에 대한 ‘지도’를 통해 올바른 평가를 받을 수 있었을 겁니다.

딥러닝을 사용한 강화학습 중 **Deep Q-Network**(일명 DQN)<sup>[44]</sup>라는 방법이 있습니다. 이는 Q학습이라는 강화학습 알고리즘을 기초로 합니다. Q학습에서는 최적 행동 가치 함수로 최적 인 행동을 정합니다. 이 함수를 딥러닝(CNN)으로 비슷하게 흉내 내어 사용하는 것이 DQN입니다.

DQN 연구 중에는 비디오 게임을 자율 학습시켜 사람을 뛰어넘는 수준의 조작을 실현한 사례가 보고되고 있습니다. [그림 8-27]과 같이 DQN에서 사용하는 CNN은 게임 영상 프레임(4개의 연속한 프레임)을 입력하여 최종적으로는 게임을 제어하는 움직임(조이스틱 이동량이나 버튼 조작 여부)에 대하여 각 동작의 '가치'를 출력합니다.

그림 8-27 Deep Q-Network로 비디오 게임 조작을 학습한다. 비디오 게임 영상을 입력받아 시행착오를 거쳐 프로그래밍 없이 뺑치는 게임 컨트롤을 학습한다.<sup>[44]</sup>



그동안의 비디오 게임 학습에서는 게임의 상태(캐릭터 위치 등)는 미리 추출하는 것이 보통이었습니다. 그러나 DQN에서는 [그림 8-27]과 같이 입력 데이터는 비디오 게임의 영상뿐입니다. 이는 DQN의 주목할 점으로, DQN의 응용 가능성을 현격히 높였다고 할 수 있습니다. 게임마다 설정을 바꿀 필요없이 단순히 DQN에 게임 영상을 보여주기만 하면 되기 때문이죠. 실제 DQN은 구성을 변경하지 않고도 팩맨과 아타리Atari 같은 많은 게임을 학습할 수 있으며, 수많은 게임에서 사람보다 뛰어난 성적을 거두고 있습니다.

**NOTE** 2016년에는 인공지능인 알파고<sup>45)</sup>가 바둑 챔피언 이세돌을 꺾었다는 소식이 화제였습니다. 이 알파고에도 딥러닝과 강화학습이 이용되었습니다. 알파고는 3,000만 개의 프로 기보를 보며 학습한 후, 알파고 스스로 자신과 맞붙는 대결을 반복하면서 수련했습니다. 또한 알파고와 DQN은 모두 구글이 인수한 딥마인드DeepMind가 진행한 연구입니다. 앞으로도 딥마인드의 활약을 기대해봅니다.\*

## 8.6 정리

이번 장에서는 (약간) 깊은 CNN을 구현하고, 손글씨 숫자 인식에서 99%를 넘는 높은 정확도를 얻었습니다. 또, 네트워크를 깊게 하는 동기를 이야기하고 최근의 딥러닝이 더 깊어지는 방향으로 가고 있다는 것도 설명했습니다. 그리고 딥러닝의 트렌드와 실제 활용 예, 고속화를 위한 연구와 미래를 느끼게 해주는 연구 사례를 소개했습니다.

딥러닝 분야에서는 아직 모르는 것이 많고 새로운 연구가 꼬리를 물고 발표되고 있습니다. 전 세계 연구자와 기술자들은 예전부터 활발하게 연구를 계속했고 지금은 상상 속의 기술을 현실화하고 있습니다. 끝까지 읽어주셔서 고맙습니다. 독자 여러분이 책을 통해서 딥러닝을 더 잘 이해하고 딥러닝의 재미를 알아준다면 저자로서 더 이상의 행복은 없습니다.

### 이번 장에서 배운 내용

- 수많은 문제에서 신경망을 더 깊게 하여 성능을 개선할 수 있다.
- 이미지 인식 기술 대회인 ILSVRC에서는 최근 딥러닝 기반 기법이 상위권을 독점하고 있으며, 그 깊이도 더 깊어지는 추세다.
- 유명한 신경망으로는 VGG, GoogLeNet, ResNet이 있다.
- GPU와 분산 학습, 비트 정밀도 감소 등으로 딥러닝을 고속화할 수 있다.
- 딥러닝(신경망)은 사물 인식뿐 아니라 사물 검출과 분할에도 이용할 수 있다.
- 딥러닝의 응용 분야로는 사진의 캡션 생성, 이미지 생성, 강화학습 등이 있다. 최근에는 자율 주행에도 딥러닝을 접목하고 있어 기대된다.

\* 옮긴이\_ 알파고는 2017년 5월 세계 바둑 랭킹 1위 커제 9단마저 3:0으로 꺾고 바둑계를 은퇴했습니다. 이세돌 9단이 알파고를 꺾어본 최후이자 유일한 인간으로 남은 것이죠. 딥마인드의 그 다음 목표는 스타크래프트2 정복입니다. 우리 인간처럼 게임 화면만 보고 플레이하는 인공지능을 목표로 한답니다. 그 과정에서 블리자드와 손잡고 StarCraft II API라는 인공지능 연구 플랫폼을 2017년 여름에 공개했으니 관심 있는 분은 살펴보시기 바랍니다. <https://goo.gl/4c8UBp>

