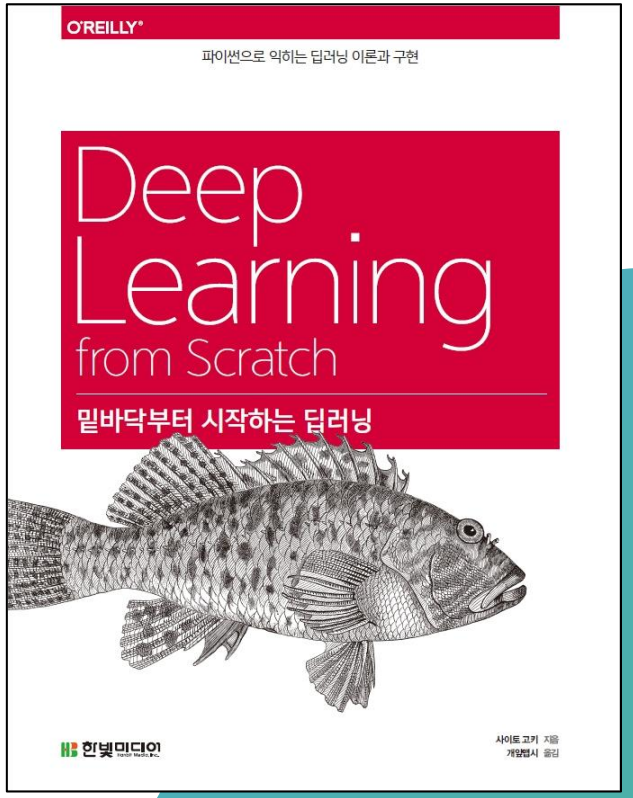
CHAPTER 8 딥러닝

밑바닥부터 시작하는 딥러닝



이 책의 학습 목표

- CHAPTER 1 파이썬에 대해 간략하게 살펴보고 사용법 익히기
- CHAPTER 2 퍼셉트론에 대해 알아보고 퍼셉트론을 써서 간단한 문제를 풀어보기
- CHAPTER 3 신경망의 개요, 입력 데이터가 무엇인지 신경망이 식별하는 처리 과정 알아보기
- CHAPTER 4 손실 함수의 값을 가급적 작게 만드는 경사법에 대해 알아보기
- CHAPTER 5 가중치 매개변수의 기울기를 효율적으로 계산하는 오차역전파법 배우기
- CHAPTER 6 신경망(딥러닝) 학습의 효율과 정확도를 높이기
- CHAPTER 7 CNN의 메커니즘을 자세히 설명하고 파이썬으로 구현하기
- **CHAPTER 8 딥러닝의 특징과 과제, 가능성, 오늘날의 첨단 딥러닝에 대해 알아보기**

Contents

CHAPTER 8 딥러닝

딥러닝의 특징과 과제, 가능성, 오늘날의 첨단 딥러닝에 대해 알아보기

8.1.1 더 깊은 신경망으로

그림 8-1 손글씨 숫자를 인식하는 심층 CNN

- 3×3의 작은 필터를 사용한 합성곱 계층
- 활성화 함수는 ReLU
- 완전연결 계층 뒤에 드롭아웃 계층 사용
- Adam을 사용해 최적화
- 가중치 초깃값은 'He의 초깃값'

SECTION 08 딥러닝

8.1.1 더 깊은 신경망으로

ch08/deep_convnet.py

```
import sys, os
sys.path.append(os.pardir)  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import pickle
import numpy as np
from collections import OrderedDict
from common.layers import *


class DeepConvNet:
    """정확도 99% 이상의 고정밀 합성곱 신경망

    네트워크 구성은 아래와 같음
        conv - relu - conv- relu - pool -
        conv - relu - conv- relu - pool -
        conv - relu - conv- relu - pool -
        affine - relu - dropout - affine - dropout - softmax
    """
    def __init__(self, input_dim=(1, 28, 28),
                 conv_param_1 = {'filter_num':16, 'filter_size':3, 'pad':1, 'stride':1},
                 conv_param_2 = {'filter_num':16, 'filter_size':3, 'pad':1, 'stride':1},
                 conv_param_3 = {'filter_num':32, 'filter_size':3, 'pad':1, 'stride':1},
                 conv_param_4 = {'filter_num':32, 'filter_size':3, 'pad':2, 'stride':1},
                 conv_param_5 = {'filter_num':64, 'filter_size':3, 'pad':1, 'stride':1},
                 conv_param_6 = {'filter_num':64, 'filter_size':3, 'pad':1, 'stride':1},
                 hidden_size=50, output_size=10):
        # 가중치 초기화===========
        # 각 층의 뉴런 하나당 앞 층의 몇 개 뉴런과 연결되는가 (TODO: 자동 계산되게 바꿀 것)
        pre_node_nums = np.array([1*3*3, 16*3*3, 16*3*3, 32*3*3, 32*3*3, 64*3*3, 64*4*4, hidden_size])
        wight_init_scales = np.sqrt(2.0 / pre_node_nums)  # ReLU를 사용할 때의 권장 초깃값

        self.params = {}
        pre_channel_num = input_dim[0]
        for idx, conv_param in enumerate([conv_param_1, conv_param_2, conv_param_3, conv_param_4, conv_param_5, conv_param_6]):
            self.params['W' + str(idx+1)] = wight_init_scales[idx] * np.random.randn(conv_param['filter_num'], pre_channel_num, conv_param['filter_size'], conv_param['filter_size'])
            self.params['b' + str(idx+1)] = np.zeros(conv_param['filter_num'])
            pre_channel_num = conv_param['filter_num']
        self.params['W7'] = wight_init_scales[6] * np.random.randn(64*4*4, hidden_size)
        self.params['b7'] = np.zeros(hidden_size)
        self.params['W8'] = wight_init_scales[7] * np.random.randn(hidden_size, output_size)
        self.params['b8'] = np.zeros(output_size)
```

SECTION 08 딥러닝

8.1.1 더 깊은 신경망으로

```
# 계층 생성===========
self.layers = []
self.layers.append(Convolution(self.params['W1'], self.params['b1'],
                   conv_param_1['stride'], conv_param_1['pad']))
self.layers.append(Relu())
self.layers.append(Convolution(self.params['W2'], self.params['b2'],
                   conv_param_2['stride'], conv_param_2['pad']))
self.layers.append(Relu())
self.layers.append(Pooling(pool_h=2, pool_w=2, stride=2))
self.layers.append(Convolution(self.params['W3'], self.params['b3'],
                   conv_param_3['stride'], conv_param_3['pad']))
self.layers.append(Relu())
self.layers.append(Convolution(self.params['W4'], self.params['b4'],
                   conv_param_4['stride'], conv_param_4['pad']))
self.layers.append(Relu())
self.layers.append(Pooling(pool_h=2, pool_w=2, stride=2))
self.layers.append(Convolution(self.params['W5'], self.params['b5'],
                   conv_param_5['stride'], conv_param_5['pad']))
self.layers.append(Relu())
self.layers.append(Convolution(self.params['W6'], self.params['b6'],
                   conv_param_6['stride'], conv_param_6['pad']))
self.layers.append(Relu())
self.layers.append(Pooling(pool_h=2, pool_w=2, stride=2))
self.layers.append(Affine(self.params['W7'], self.params['b7']))
self.layers.append(Relu())
self.layers.append(Dropout(0.5))
self.layers.append(Affine(self.params['W8'], self.params['b8']))
self.layers.append(Dropout(0.5))

self.last_layer = SoftmaxWithLoss()
```

```
def predict(self, x, train_flg=False):
    for layer in self.layers:
        if isinstance(layer, Dropout):
            x = layer.forward(x, train_flg)
        else:
            x = layer.forward(x)
    return x

def loss(self, x, t):
    y = self.predict(x, train_flg=True)
    return self.last_layer.forward(y, t)

def accuracy(self, x, t, batch_size=100):
    if t.ndim != 1 : t = np.argmax(t, axis=1)

    acc = 0.0

    for i in range(int(x.shape[0] / batch_size)):
        tx = x[i*batch_size:(i+1)*batch_size]
        tt = t[i*batch_size:(i+1)*batch_size]
        y = self.predict(tx, train_flg=False)
        y = np.argmax(y, axis=1)
        acc += np.sum(y == tt)

    return acc / x.shape[0]

def gradient(self, x, t):
    # forward
    self.loss(x, t)

    # backward
    dout = 1
    dout = self.last_layer.backward(dout)

    tmp_layers = self.layers.copy()
    tmp_layers.reverse()
    for layer in tmp_layers:
        dout = layer.backward(dout)

    # 결과 저장
    grads = {}
    for i, layer_idx in enumerate((0, 2, 5, 7, 10, 12, 15, 18)):
        grads['W' + str(i+1)] = self.layers[layer_idx].dW
        grads['b' + str(i+1)] = self.layers[layer_idx].db

    return grads
```

SECTION 08 딥러닝

8.1.1 더 깊은 신경망으로

```
# 계층 생성===========
self.layers = []
self.layers.append(Convolution(self.params['W1'], self.params['b1'],
                   conv_param_1['stride'], conv_param_1['pad']))
self.layers.append(Relu())
self.layers.append(Convolution(self.params['W2'], self.params['b2'],
                   conv_param_2['stride'], conv_param_2['pad']))
self.layers.append(Relu())
self.layers.append(Pooling(pool_h=2, pool_w=2, stride=2))
self.layers.append(Convolution(self.params['W3'], self.params['b3'],
                   conv_param_3['stride'], conv_param_3['pad']))
self.layers.append(Relu())
self.layers.append(Convolution(self.params['W4'], self.params['b4'],
                   conv_param_4['stride'], conv_param_4['pad']))
self.layers.append(Relu())
self.layers.append(Pooling(pool_h=2, pool_w=2, stride=2))
self.layers.append(Convolution(self.params['W5'], self.params['b5'],
                   conv_param_5['stride'], conv_param_5['pad']))
self.layers.append(Relu())
self.layers.append(Convolution(self.params['W6'], self.params['b6'],
                   conv_param_6['stride'], conv_param_6['pad']))
self.layers.append(Relu())
self.layers.append(Pooling(pool_h=2, pool_w=2, stride=2))
self.layers.append(Affine(self.params['W7'], self.params['b7']))
self.layers.append(Relu())
self.layers.append(Dropout(0.5))
self.layers.append(Affine(self.params['W8'], self.params['b8']))
self.layers.append(Dropout(0.5))

self.last_layer = SoftmaxWithLoss()
```

```
def predict(self, x, train_flg=False):
    for layer in self.layers:
        if isinstance(layer, Dropout):
            x = layer.forward(x, train_flg)
        else:
            x = layer.forward(x)
    return x

def loss(self, x, t):
    y = self.predict(x, train_flg=True)
    return self.last_layer.forward(y, t)

def accuracy(self, x, t, batch_size=100):
    if t.ndim != 1 : t = np.argmax(t, axis=1)

    acc = 0.0

    for i in range(int(x.shape[0] / batch_size)):
        tx = x[i*batch_size:(i+1)*batch_size]
        tt = t[i*batch_size:(i+1)*batch_size]
        y = self.predict(tx, train_flg=False)
        y = np.argmax(y, axis=1)
        acc += np.sum(y == tt)

    return acc / x.shape[0]

def gradient(self, x, t):
    # forward
    self.loss(x, t)

    # backward
    dout = 1
    dout = self.last_layer.backward(dout)

    tmp_layers = self.layers.copy()
    tmp_layers.reverse()
    for layer in tmp_layers:
        dout = layer.backward(dout)

    # 결과 저장
    grads = {}
    for i, layer_idx in enumerate((0, 2, 5, 7, 10, 12, 15, 18)):
        grads['W' + str(i+1)] = self.layers[layer_idx].dW
        grads['b' + str(i+1)] = self.layers[layer_idx].db

    return grads
```

8.1.1 더 깊은 신경망으로

ch08/train_deepnet.py

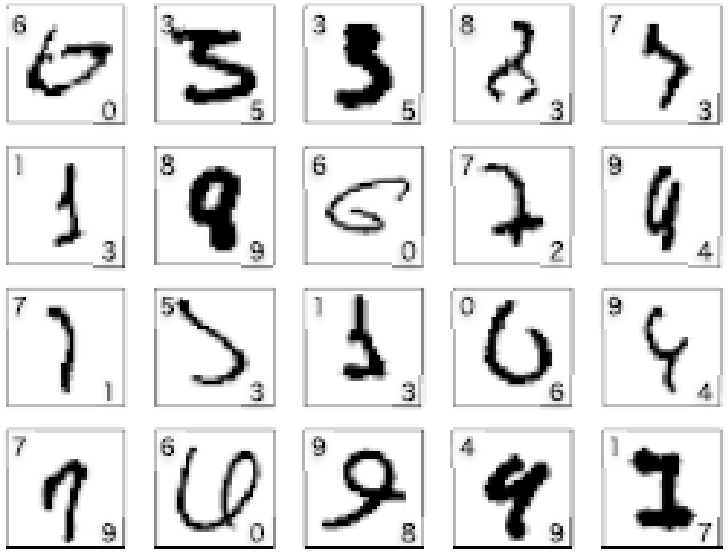
```
import sys, os
sys.path.append(os.pardir)  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np
import matplotlib.pyplot as plt
from dataset.mnist import load_mnist
from deep_convnet import DeepConvNet
from common.trainer import Trainer

(x_train, t_train), (x_test, t_test) = load_mnist(flatten=False)

network = DeepConvNet()
trainer = Trainer(network, x_train, t_train, x_test, t_test,
                  epochs=20, mini_batch_size=100,
                  optimizer='Adam', optimizer_param={'lr':0.001},
                  evaluate_sample_num_per_epoch=1000)
trainer.train()

# 매개변수 보관
network.save_params("deep_convnet_params.pkl")
print("Saved Network Parameters!")
```

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

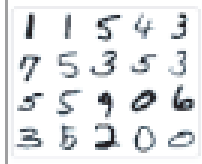


8.1.2 정확도를 더 높이려면

그림 8-3 MNIST 데이터셋에 대한 각 기법의 순위(2016년 12월 시점)[93]

MNIST

who is the best in MNIST ?



MNIST 88 results collected

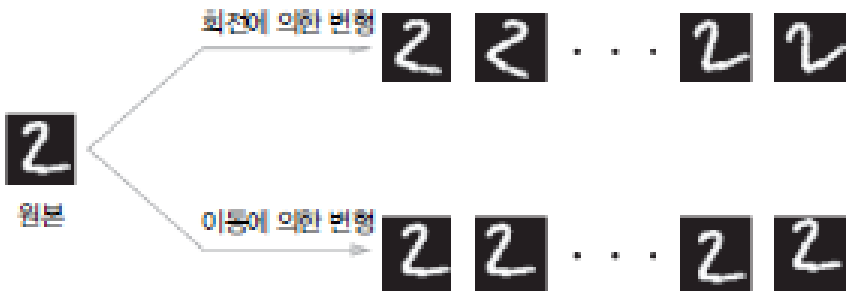
Units: error %

Classify handwritten digits. Some additional results are available on the original dataset page.

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

데이터 확장data augmentation은 입력 이미지(훈련 이미지)를 알고리즘을 동원해 '인위적'으로 확장한다.

그림 8-4 데이터 확장의 예



Data Augmentation
데이터가 부족할 때 효과적인 수단
Crop: 이미지 일부 잘라냄
Flip: 좌우 뒤집기
* 쉬운 트릭 멋진 결과

8.1.3 깊게 하는 이유

'층을 깊게 하는 것'의 중요성에 대해서, 이를 뒷받침하는 데이터와 설명을 몇 가지 소개하겠다.

그림 8-5 5×5 합성곱 연산의 예

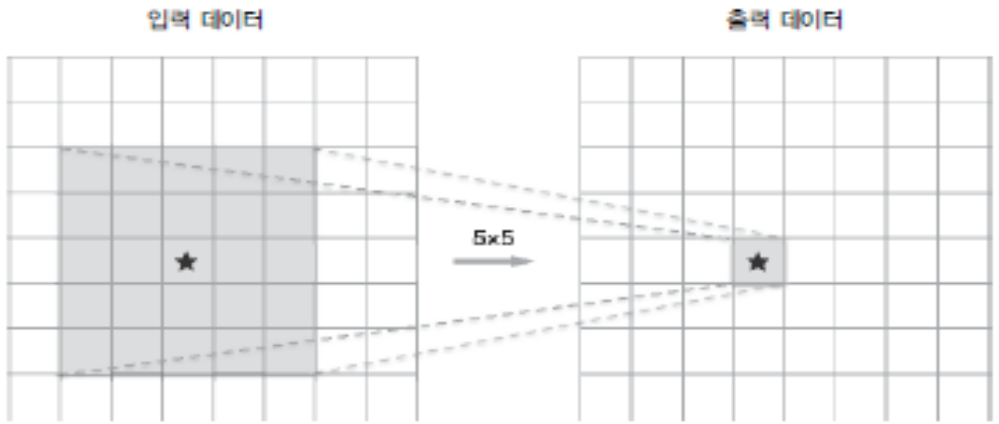
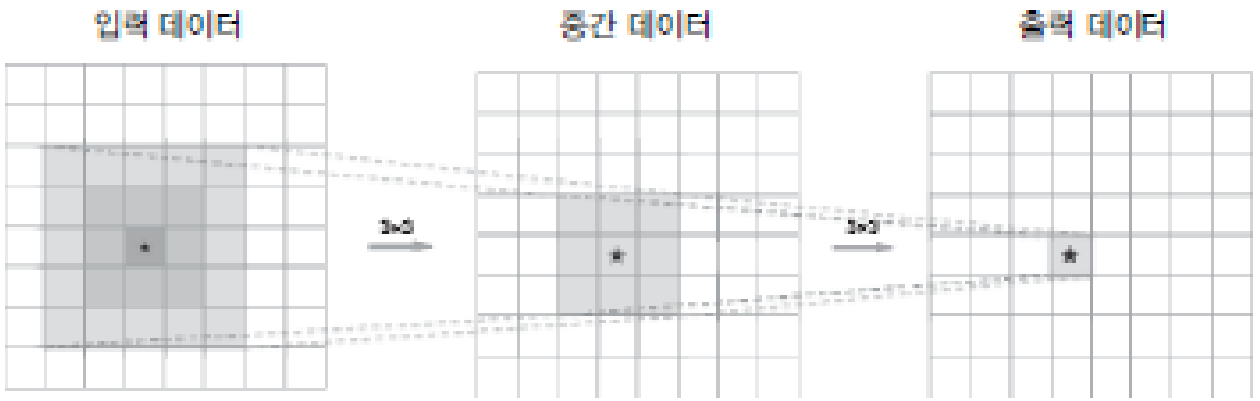


그림 8-6 3×3의 합성곱 계층을 2회 반복한 예



층을 깊게 할 때의 이점

1. 매개변수 수의 감소
   5x5에 대한 컨볼루션 기준으로 보면
   그림 8-5(25개:5x5) 그림 8-6(18개:2x3x3)

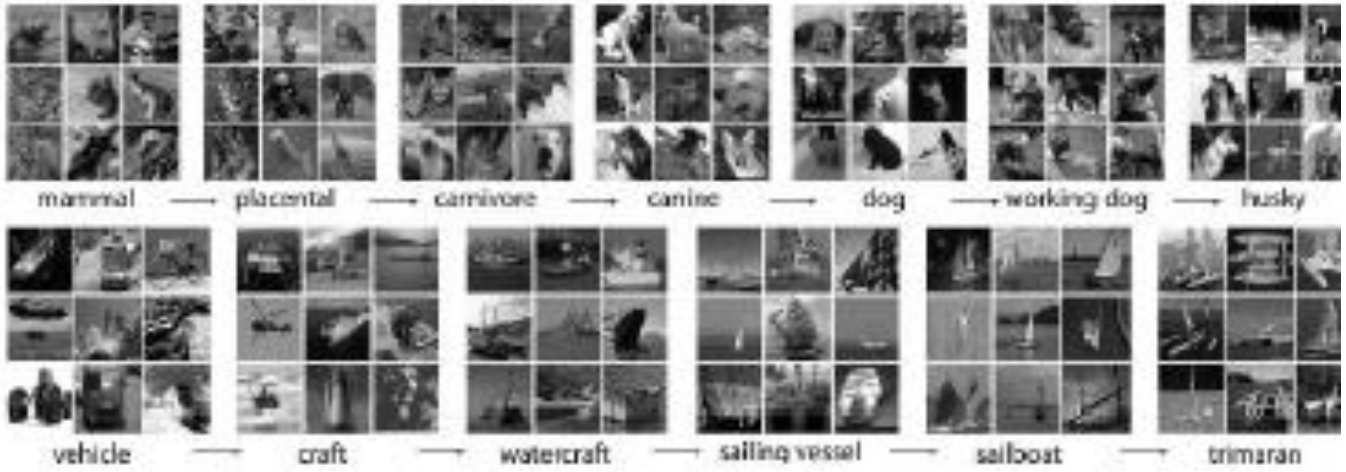
2. 학습의 효율성
   전단: 단순한 특징...후단: 복잡한 특징 반응
3. 정보의 계층적 전달
   학습해야 할 문제의 계층적 분해
   정보의 계층적 전달(에지→ 고도의 패턴)

SECTION 08 딥러닝

8.2 딥러닝의 초기 역사

8.2.1 이미지넷

그림 8-7 대규모 데이터셋 ImageNet의 데이터들[25]

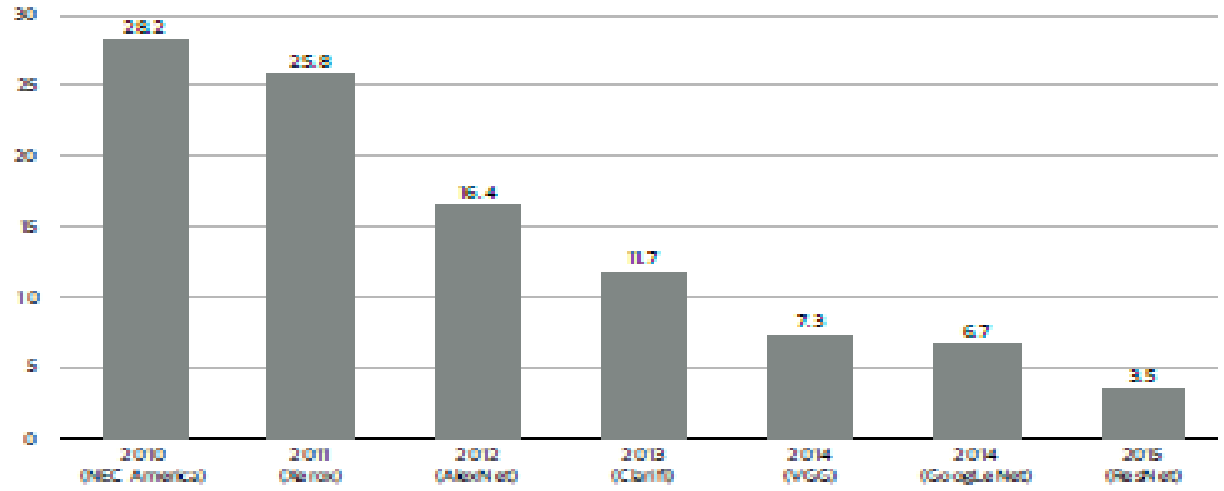


ImageNet:100만장이 넘는 이미지 데이터셋
ILSVRC(ImageNet Large Scale Visual Recognition Challenge)

Classification: 1000개의 class

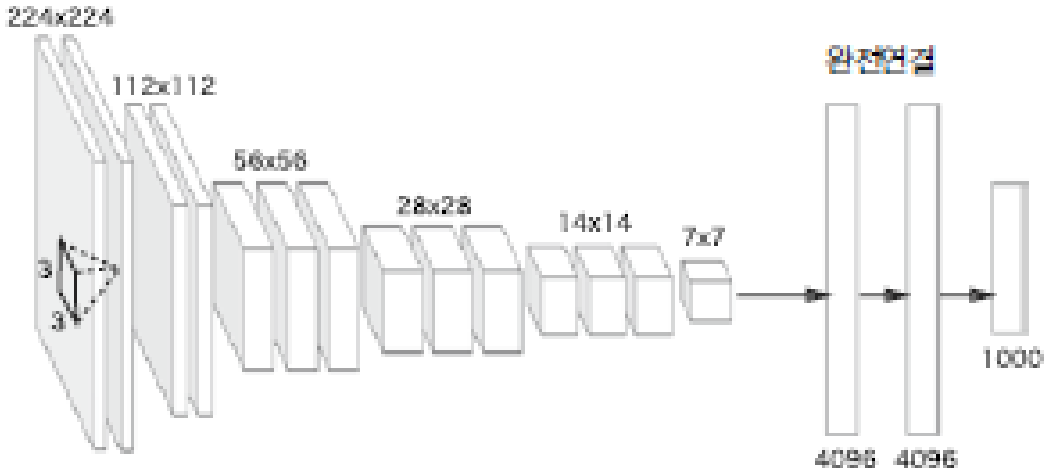
그림 8-8 ILSVRC 최우수 팀의 성적 추이 : 세로축은 오류율, 가로축은 연도, 가로축의 괄호 안은 팀 이름(또는 기법 이름)

이미지넷 분류 톱-5 오류



8.2.2 VGG

그림 8-9 VGG[22]



VGG에서 주목할 점은 3×3의 작은 필터를 사용한 합성곱 계층을 연속으로 거친다는 것이다.

8.2.3 GoogLeNet

그림 8-10 GoogLeNet[23]

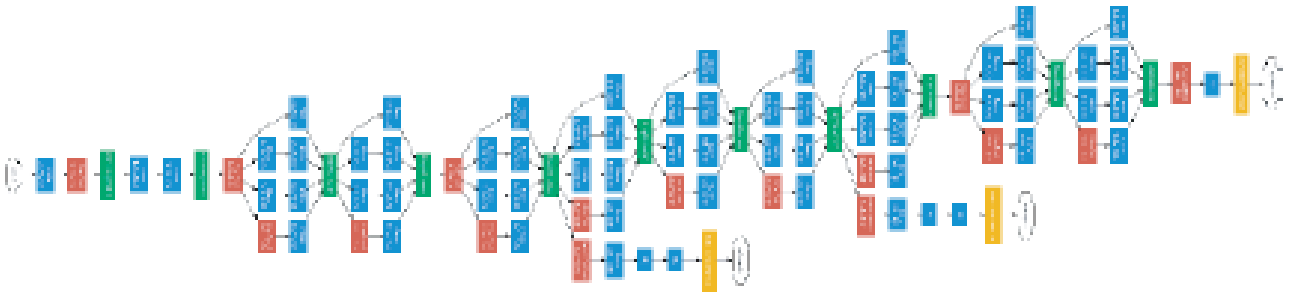
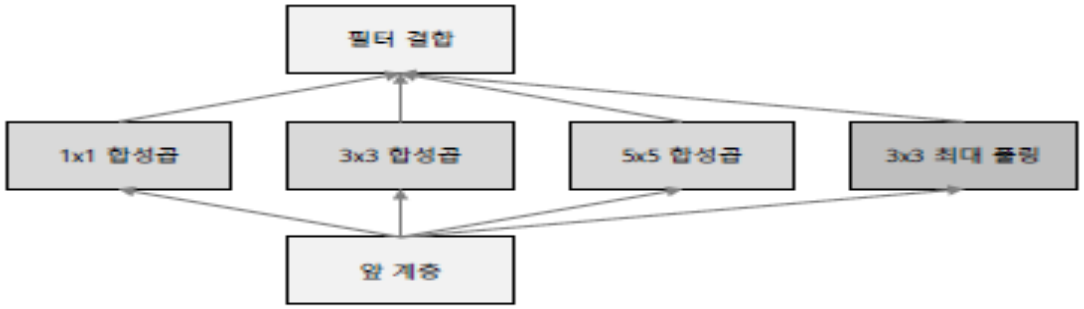


그림 8-11 GoogLeNet의 인셉션 구조[23]



8.2.4 ResNet

그림 **8-12** ResNet의 구성요소[24] : 'weight layer'는 합성곱 계층을 말한다

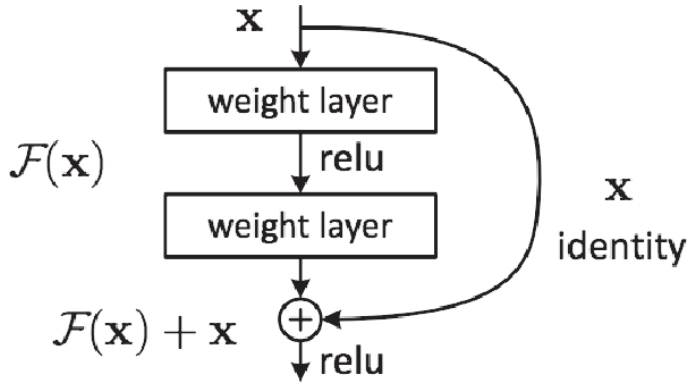
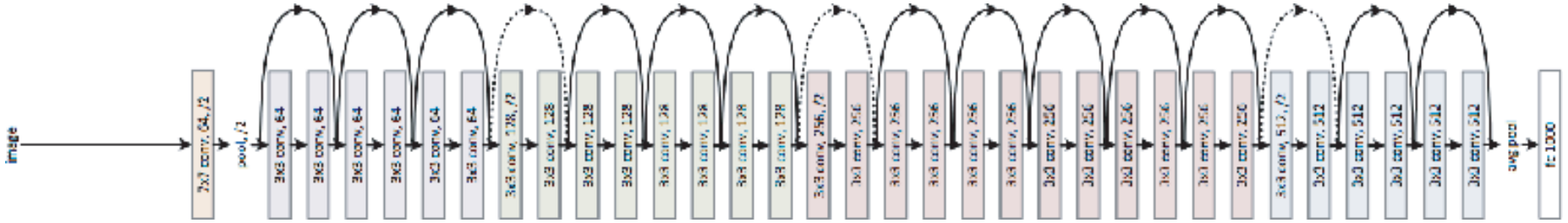


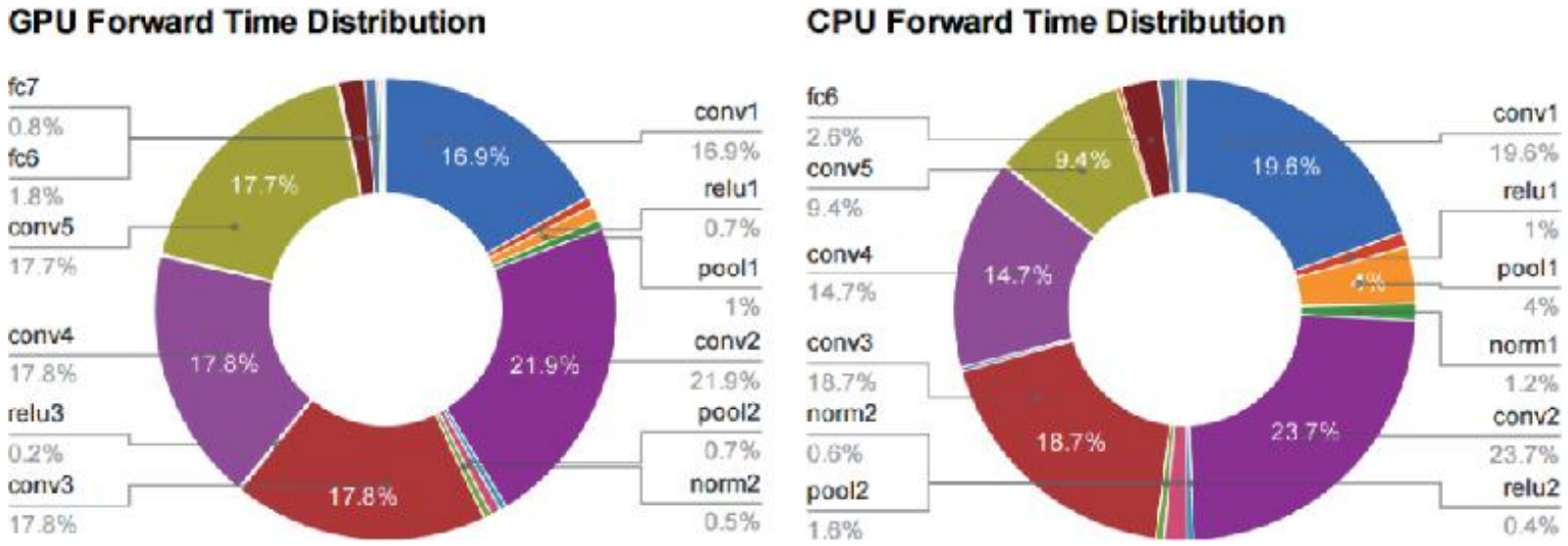
그림 **8-13** ResNet[24] : 블록이 3×3인 합성곱 계층에 대응. 층을 건너뛰는 스킵 연결이 특징이다.



8.3 더 빠르게(딥러닝의 고속화)

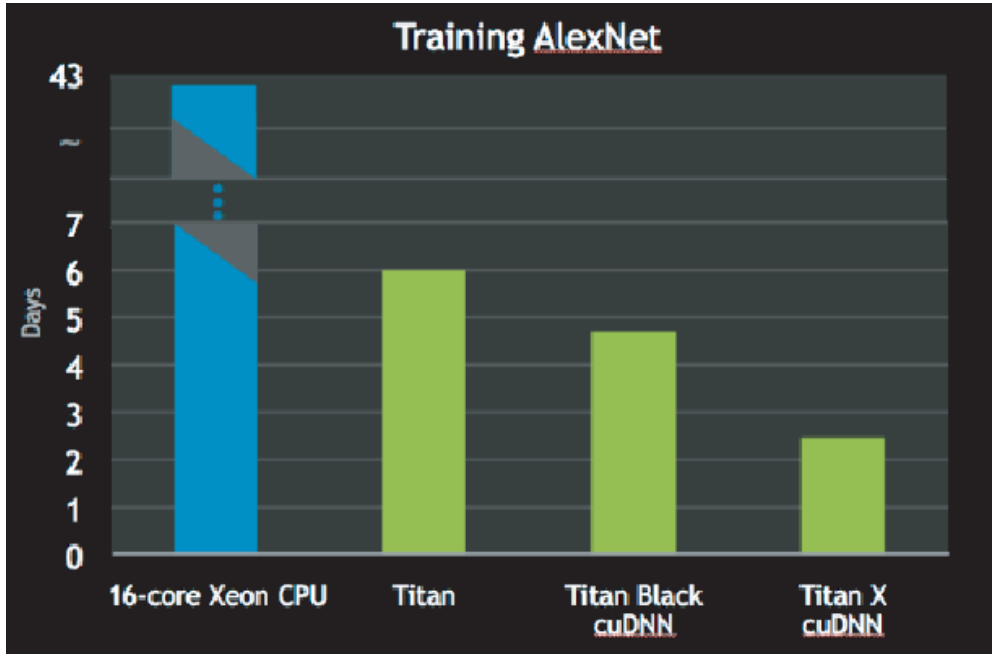
8.3.1 풀어야 할 숙제

그림 **8-14** AlexNet의 forward 처리 시 각 층의 시간 비율 : 왼쪽이 GPU, 오른쪽이 CPU를 사용한 경우. ‘conv’는합성곱 계층, ‘pool’은 풀링 계층, ‘fc’는 완전연결 계층, ‘norm’은 정규화 계층이다.[26



8.3.2 GPU를 활용한 고속화

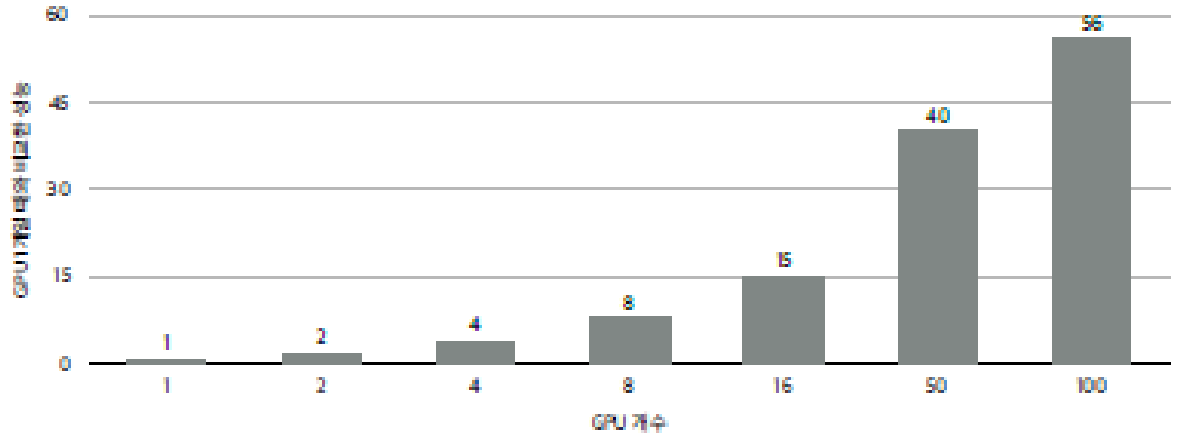
그림 **8-15** AlexNet의 학습 시간을 '16코어 제온 CPU'와 엔비디아 '타이탄 GPU'에서 비교한 결과[27]



엔비디아의 GPU 컴퓨팅용 통합 개발 환경인 **CUDA**를 사용하기 때문.
[그림 8-15]에 등장하는 **cuDNN**은CUDA 위에서 동작하는 라이브러리로, 딥러닝에 최적화된 함수 등이 구현되어 있다

8.3.3 분산 학습

그림 8-16 텐서플로의 분산 학습 성능 : 가로는 GPU의 수, 세로는 GPU 1개일 때와 비교한 배율[28]



[그림 8-16]에서 보듯 GPU 수가 늘어남에 따라 학습도 빨라진다.

8.3.4 연산 정밀도와 비트 줄이기

계산 능력 외에도 메모리 용량과 버스 대역폭 등이 딥러닝 고속화에 병목이 될 수 있다

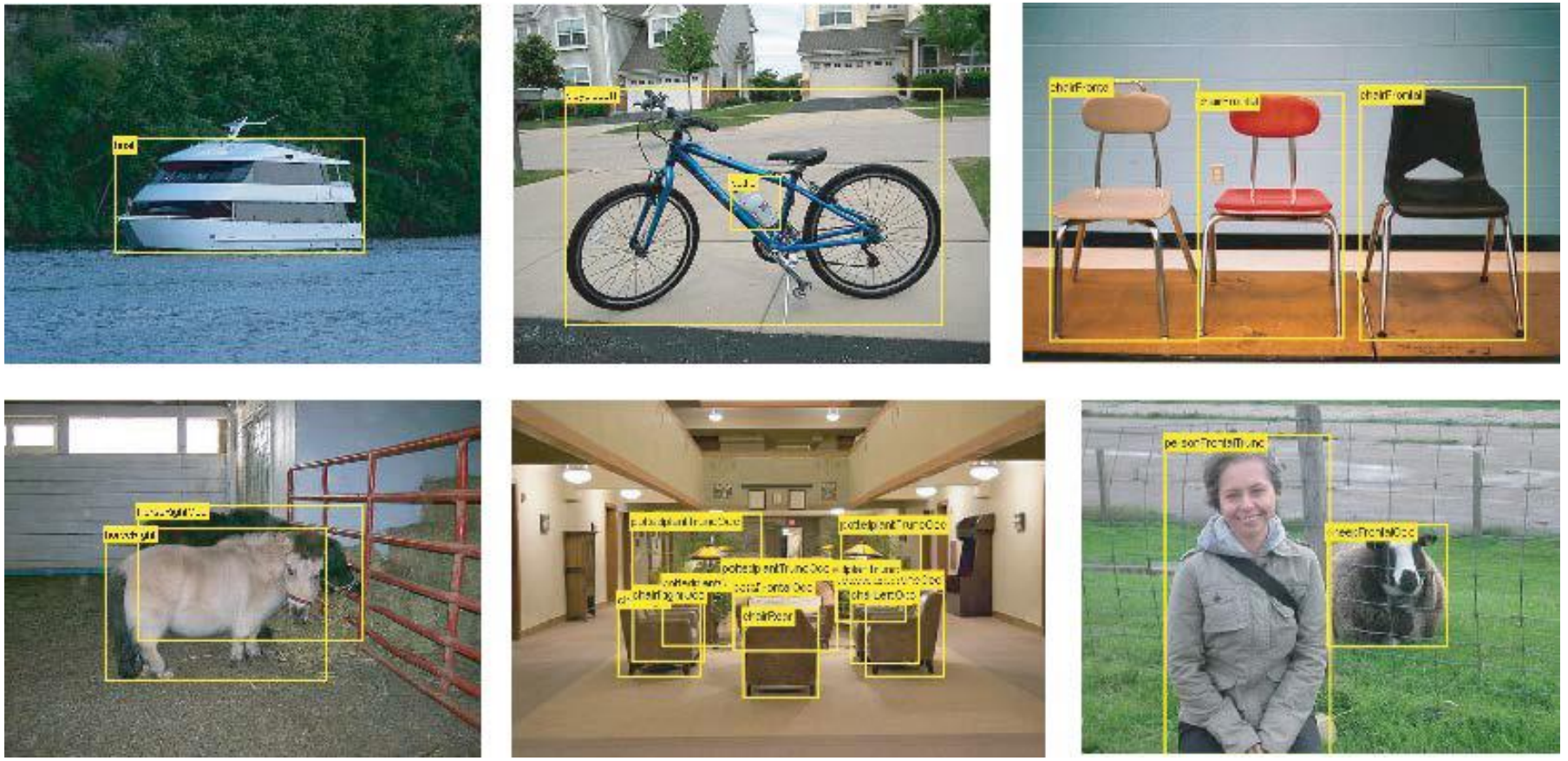
버스 대역폭 면에서는 GPU(혹은 CPU)의 버스를 흐르는 데이터가 많아져 한계를 넘어서면 병목이 된다. 이러한 경우를 고려하면 네트워크로 주고받는 데이터의 비트
수는 최소로 만드는 것이 바람직하다

다행히 딥러닝은 높은 수치 정밀도(수치를 몇 비트로 표현하느냐)를 요구하지 않는다. 이는 신경망의 중요한 성질 중 하나로, 신경망의 견고성에 따른 특성이다.
예를 들어 신경망은 입력 이미지에 노이즈가 조금 섞여 있어도 출력 결과가 잘 달라지지 않는 강건함을 보여준다

8.4 딥러닝의 활용

8.4.1 사물 검출

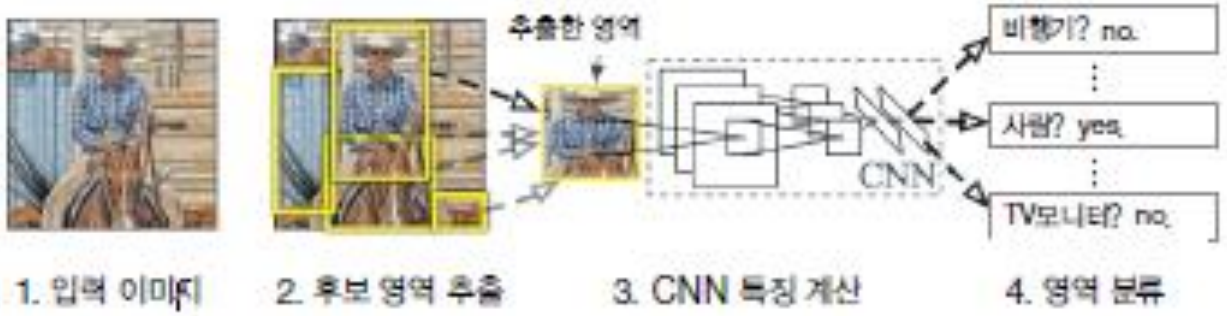
그림 **8-17** 사물 검출의 예[34



8.4.1 사물 검출

CNN을 이용하여 사물 검출을 수행하는 방식은 몇 가지가 있는데,
그중에서도**R-CNN**Regions with Convolutional Neural Network[35]이 유명하다.
[그림 8-18]은 R-CNN의 처리 흐름이다.

그림 8-18 R-CNN의 처리 흐름[35]

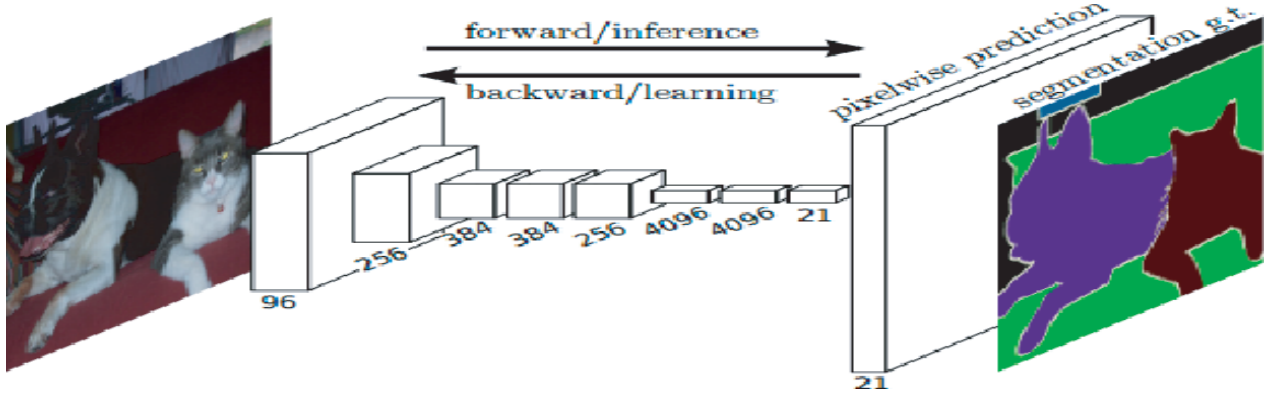


8.4.2 분할

분할segmentation이란 이미지를 픽셀 수준에서 분류하는 문제이다

그림 **8-19** 분할의 예 : 왼쪽이 입력 이미지, 오른쪽이 지도용 이미지

그림 **8-20** FCN의 전체 그림[37]



8.4.3 사진 캡션 생성

그림 8-21 딥러닝으로 사진 캡션을 생성하는 예[39]

[그림 8-22]와 같이 심층 CNN과 자연어를 다루는 순환 신경망Recurrent Neural Network, RNN으로 구성된다.

- RNN은 순환적 관계를 갖는 신경망으로 자연어나 시계열 데이터 등의 연속된 데이터를 다룰 때 많이 활용한다

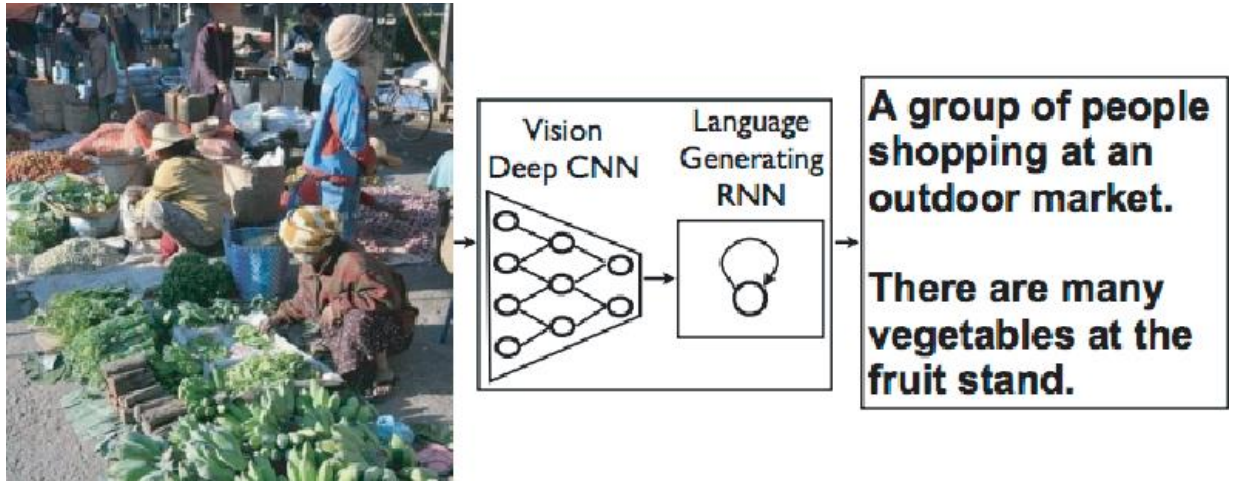


그림 8-22 NIC의 전체 구성[

SECTION 08 딥러닝

8.5 딥러닝의 미래

8.5.1 이미지 스타일(화풍) 변환

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[49]

8.5.2 이미지 생성

딥러닝으로 '침실' 이미지를 무無로부터 생성하는 게 가능하다.
[그림 8-24]의 이미지는 **DCGAN**Deep Convolutional Generative Adversarial Network 기법[41]으로 생성한 침실 이미지들이다.

그림 8-24 DCGAN으로 새롭게 생성한 침실 이미지들[41]

8.5.3 자율 주행

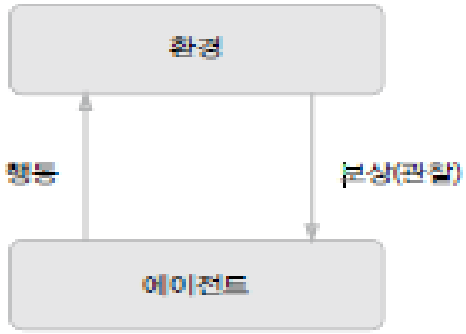
SegNet[42]이라는 CNN 기반 신경망은 [그림 8-25]와 같이 주변 환경을 정확하게 인식해낸다.

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

8.5.4 Deep Q-Network(강화학습)

이는 '가르침'에 의존하는 '지도 학습'과는 다른 분야로,
강화학습reinforcement learning이라 한다.

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



딥러닝을 사용한 강화학습 중 **Deep Q-Network**(일명 DQN)[44]라는 방법이 있다.

그림 8-27 Deep Q-Network로 비디오 게임 조작을 학습한다. 비디오 게임 영상을 입력받아 시행착오를 거쳐 프로 게이머 뺨치는 게임 컨트롤을 학습한다.[45]