영상처리 실제

Image Processing Practice

신재혁

naezang@cbnu.ac.kr

8장 컨볼루션 신경망

Preview

■ MLP의 한계

- 2차원 구조의 영상을 1차원으로 변환하여 입력. 적지 않은 정보 손실로 인한 성능 저하

■ 인간은 2차원에서 특징 추출

- 수용장receptive field이라는 작은 영역에서 특징을 추출

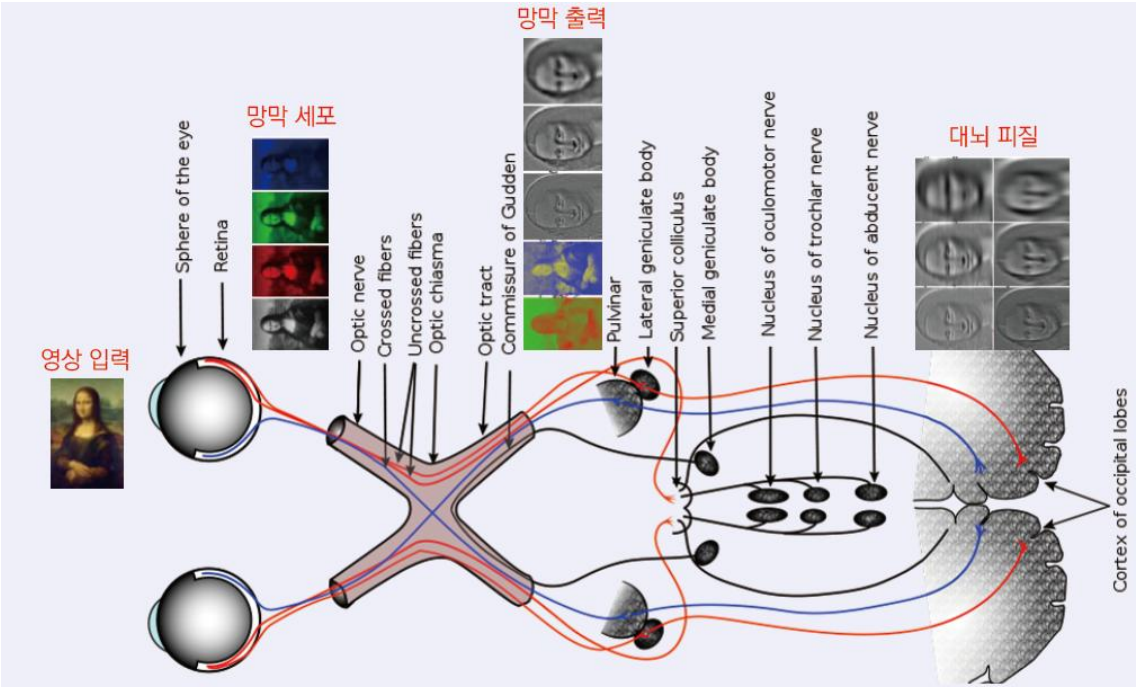


그림 8-1 인간 시각의 특징 추출과 인식

Preview

■ 컨볼루션 신경망CNN:Convolutional Neural Network은 인간 시각을 모방

- 딥러닝 성공에 가장 기여한 모델
- 다양한 응용
  - 컴퓨터 비전에서는 분류, 검출, 분할, 추적 등의 문제 해결
  - 비디오 게임 인공지능에서는 화면 장면을 분석
  - 알파고는 19*19 바둑판 형세 판단

컨볼루션 신경망에 대한 인기 있는 강의는 스탠퍼드 대학의 cs231n이다. 강의 노트는 http://cs231n.stanford.edu/에서 볼 수 있고 컨볼루션 신경망 부분은 https://cs231n.github.io/convolutional-networks/를 참조한다.
유튜브에서 'cs231n'을 검색하면 강의 영상을 찾을 수 있다.

8.1 발상과 전개

■ 7장의 다층 퍼셉트론

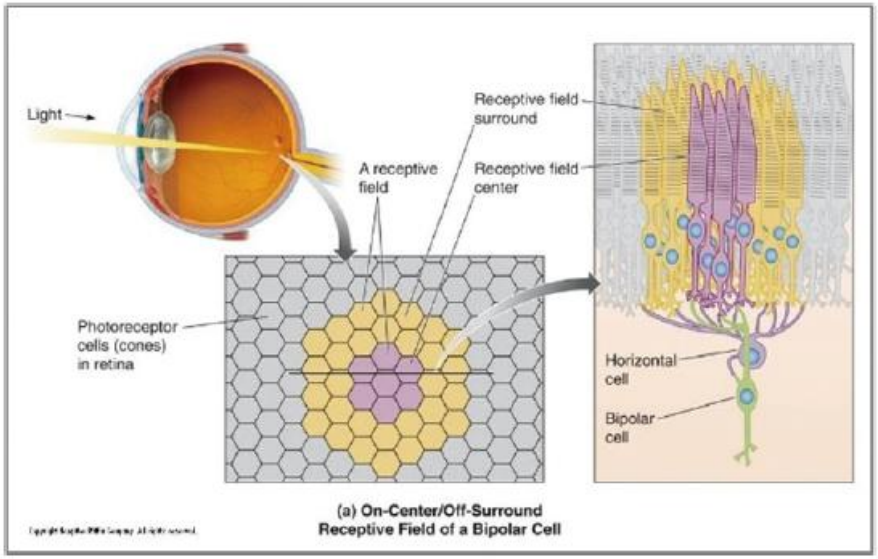
- 영상을 입력하기 위해 1차원으로 펼칠 때 정보 손실
- FC(완전연결)층의 과다한 매개변수
  - 예) 256*256*3 영상을 입력하려면 196,608개의 입력 노드 필요

(a) 2차원 구조의 영상 (b) 1차원 구조로 펼친 경우

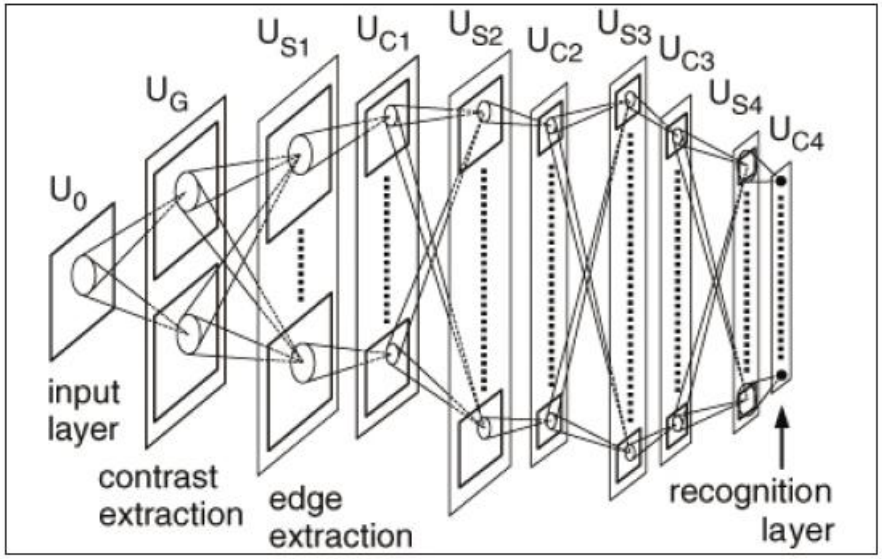
그림 8-2 2차원 구조의 영상을 1차원으로 변환할 때 발생하는 정보 손실

8.1 발상과 전개

■ 인간 수용장receptive field을 모방하는 컨볼루션 신경망 등장



(a) 인간 시각의 수용장



(b) 네오코그니트론[Fukushima1980]

그림 8-3 인간의 수용장과 인공 신경망의 수용장

8.1 발상과 전개

■ 컨볼루션 신경망의 간략 역사

- 1980년 네오코그니트론
- 1998년 르쿤의 논문 [LeCun1998]: LeNet-5를 구현하여 수표 자동 입력 시스템 구현
- 2010년 ImageNet 데이터셋 탄생 (1400만개 영상이 21841 부류로 레이블링)

1000 synsets for Object classification/localization

n02119789: kit fox, Vulpes macrotis
n02100735: English setter
n02096294: Australian terrier
n02066245: grey whale, gray whale, devilfish, Eschrichtius gibbosus, Eschrichtius robustus
n02509815: lesser panda, red panda, panda, bear cat, cat bear, Ailurus fulgens
n02124075: Egyptian cat
n02417914: ibex, Capra ibex
n02123394: Persian cat
n02125311: cougar, puma, catamount, mountain lion, painter, panther, Felis concolor
n02423022: gazelle

(a) ILSVRC 대회의 1000부류 일부

(b) 고양이 부류에 속하는 영상의 일부

그림 8-4 ImageNet 데이터셋

8.1 발상과 전개

- 2010년부터 ILSVRC 대회 개최(1000부류, 120만 훈련, 5만 검증, 15만 테스트 집합)
- 2012년 AlexNet이 15.3% 오류율로 우승. CNN에 대한 폭발적인 관심 불러일으킴
- 이후 층이 깊어진 VGGNet, GoogLeNet, ResNet이 우승
- 물체 검출, 분할, 추적, 자세 추정 등의 다양한 모델 등장
- 2014년 생성 모델 GAN 등장
- 알고리즘 발전
  - ReLU 등 활성 함수
  - 교차 엔트로피 등 손실 함수
  - 드롭 아웃 등 규제 기법
  - Adam 등 옵티마이저

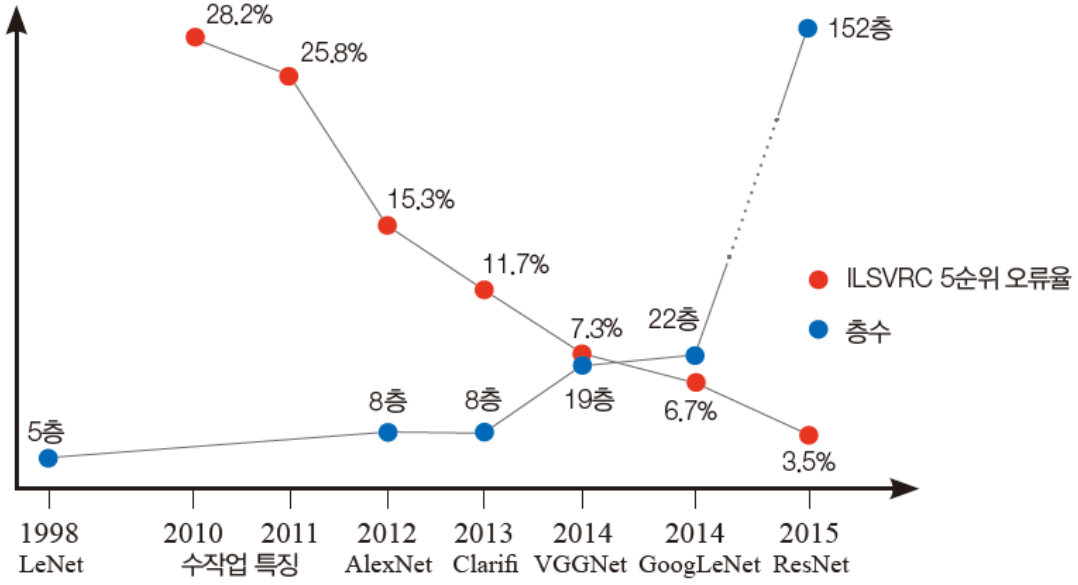


그림 8-5 컨볼루션 신경망의 초기 발전 추세[오일석2017]

8.1 발상과 전개

■ 딥러닝의 성공 요인

- 인터넷으로 인해 커진 데이터셋
- GPU로 인해 값싸게 병렬 처리 가능해짐
- 학습 알고리즘의 발전

8.2 컨볼루션 신경망의 구조

■ 컨볼루션 신경망(CNN; convolutional neural network)

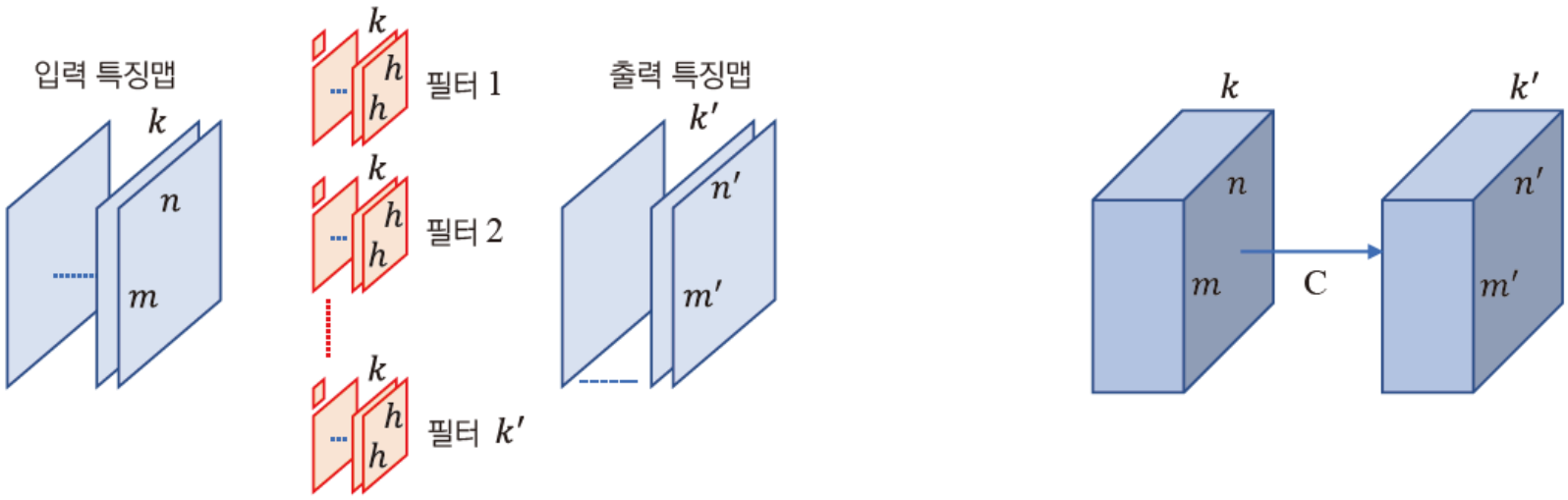
- CNN은 3.4절에서 공부한 컨볼루션 연산이 핵심
- 3.4절에서는 사람이 필터 설계(예, 가우시안 스무딩, 소벨 에지 등)
  - 이들 필터는 어디서 왔는가?
  - 인식에 최적인가?
  - 데이터셋에 맞는 최적 필터를 사용해야 하지 않을까?

■ CNN의 핵심 아이디어는 '최적의 필터를 학습'으로 알아냄

8.2.1 컨볼루션층과 풀링층

■ **컨볼루션층은 표준 컨볼루션에서 몇 가지 확장이 필요**

- 입력 특징 맵이 m*n*k 텐서라면, h*h*k 필터 사용
- 하나의 필터는 바이어스 하나를 가짐 (kh^2+1개의 가중치)
- 필터를 여러 개(k′개) 적용하여 풍부한 특징 맵 추출
- 출력 특징 맵은 m′*n′*k′ 텐서
- 덧대기$_{padding}$와 보폭$_{stride}$



(a) 세부 내용

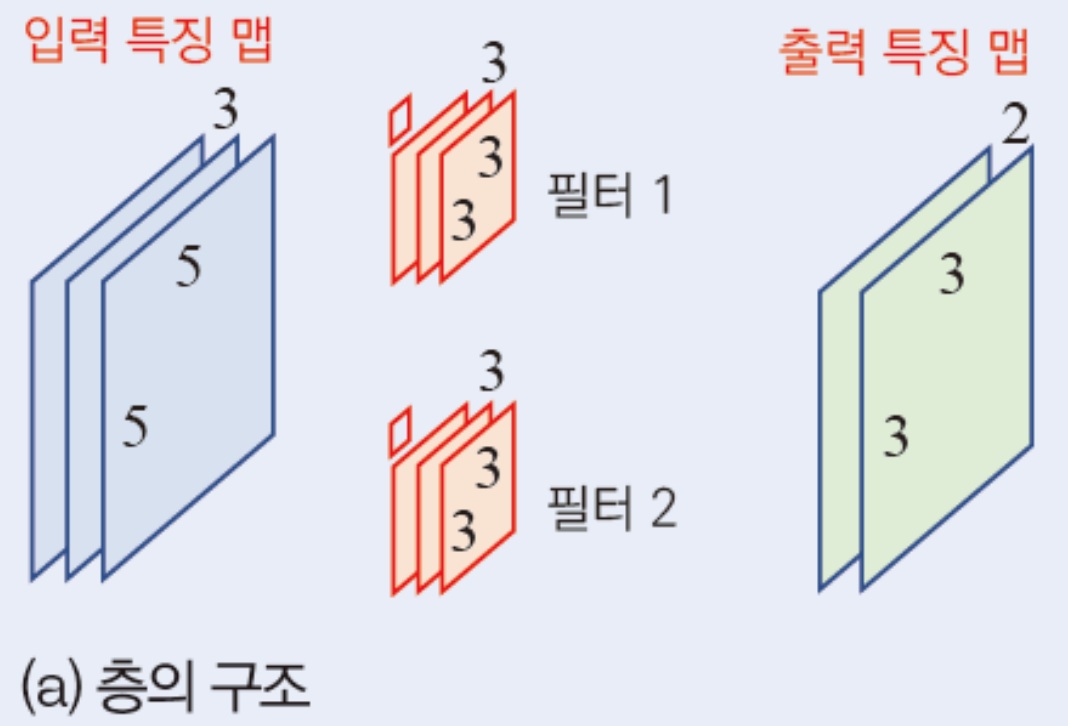
(b) 간결한 블록 표현

그림 8-6 컨볼루션층

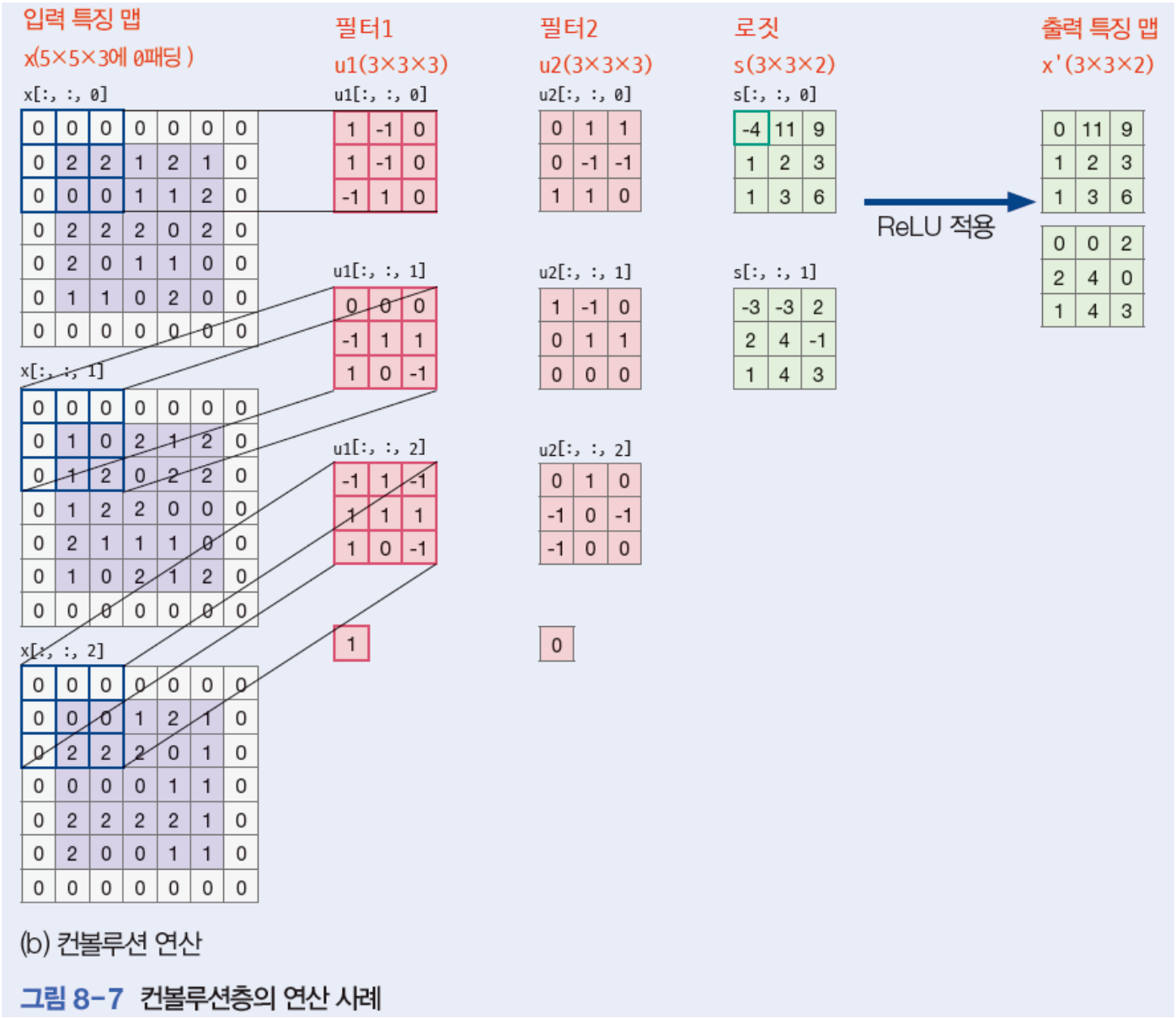
8.2.1 컨볼루션층과 풀링층

■ [예시 8-1] 컨볼루션층의 연산

- 5*5*3 특징 맵에 3*3*3 필터를 2개 적용. 0 패딩 적용, 보폭은 2
- 5*5*3 특징 맵이 3*3*2 특징 맵이 됨([그림 8-7])



8.2.1 컨볼루션층과 풀링층



(b) 컨볼루션 연산

그림 8-7 컨볼루션층의 연산 사례

8.2.1 컨볼루션층과 풀링층

- (0,0) 화소의 계산 사례

$$\underbrace{0\times1+0\times(-1)+0\times0+0\times1+2\times(-1)+2\times0+0\times(-1)+0\times1+0\times0+}_{\text{채널 0}}$$

$$\underbrace{0\times0+0\times0+0\times0+0\times(-1)+1\times1+0\times1+0\times1+1\times0+2\times(-1)+}_{\text{채널 1}}$$

$$\underbrace{0\times(-1)+0\times1+0\times(-1)+0\times1+0\times1+0\times1+0\times1+2\times0+2\times(-1)+}_{\text{채널 2}}\ \underbrace{1}_{\text{바이어스}}=-4$$

■ 가중치 공유weight sharing와 부분 연결성partial connection

- 입력 특징 맵의 모든 화소가 같은 필터 사용하니 가중치를 공유하는 셈
- 필터는 해당 화소 주위에 국한하여 연산 수행. 가중치 개수가 획기적으로 줄어듦
  - k′개의 h*h*k 필터를 쓰는 경우 가중치는 $k'(kh^2+1)$개

8.2.1 컨볼루션층과 풀링층

■ [예제 8-2] 컨볼루션층의 연산량

- 첫번째 층
  - 입력은 256*256*3 텐서. 0 덧대기하고 보폭 2이고 3*3 필터를 64개 사용하므로 출력은 128*128*64 텐서 (필터 모양은 3*3*3)
  - 128*128*28*64번의 곱셈 수행
- 두번째 층
  - 입력은 128*128*64 텐서. 0 덧대기하고 보폭 2이고 5*5 필터를 128개 사용하므로 출력은 64*64*128 텐서 (필터 모양은 5*5*64)
  - 64*64*1601*128번의 곱셈 수행

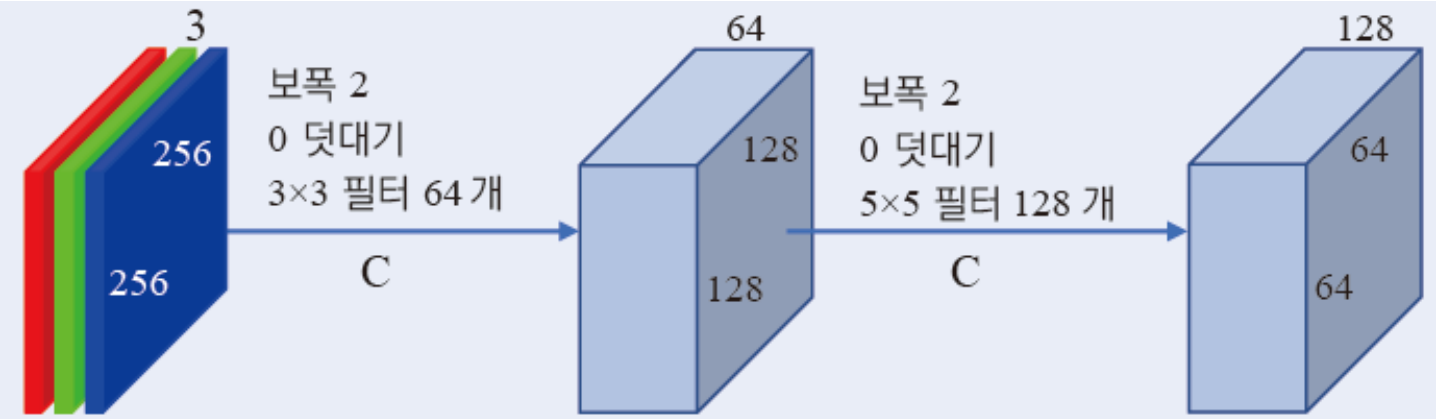


그림 8-8 컨볼루션층을 2개 쌓은 신경망

8.2.1 컨볼루션층과 풀링층

■ 풀링층

- 최대 풀링max pooling은 필터 안의 화소의 최대값 취함
- 평균 풀링average pooling은 필터 안의 화소의 평균을 취함
- 지나친 상세함을 줄이는 효과와 특징 맵의 크기를 줄이는 효과

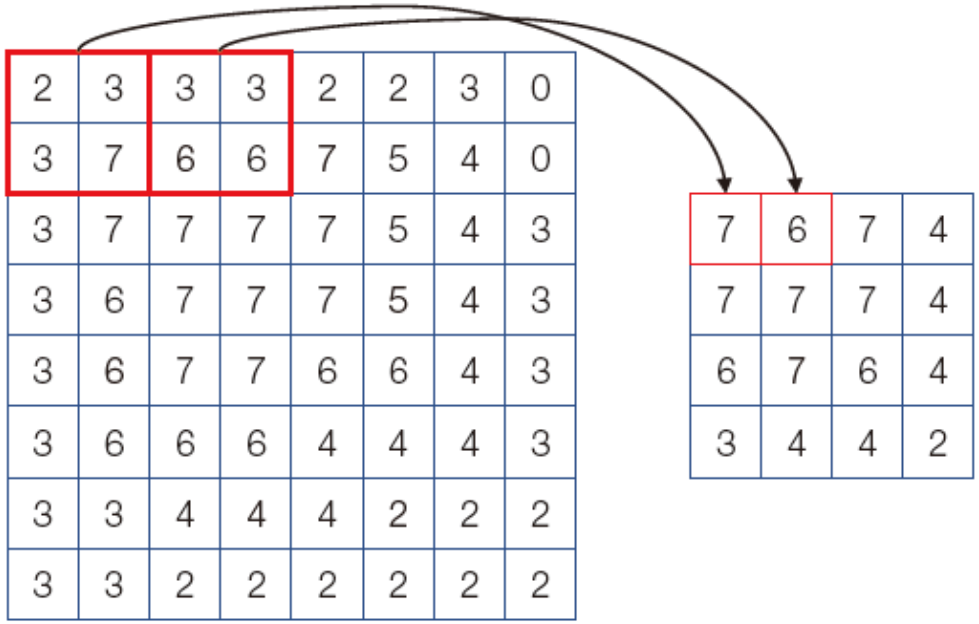
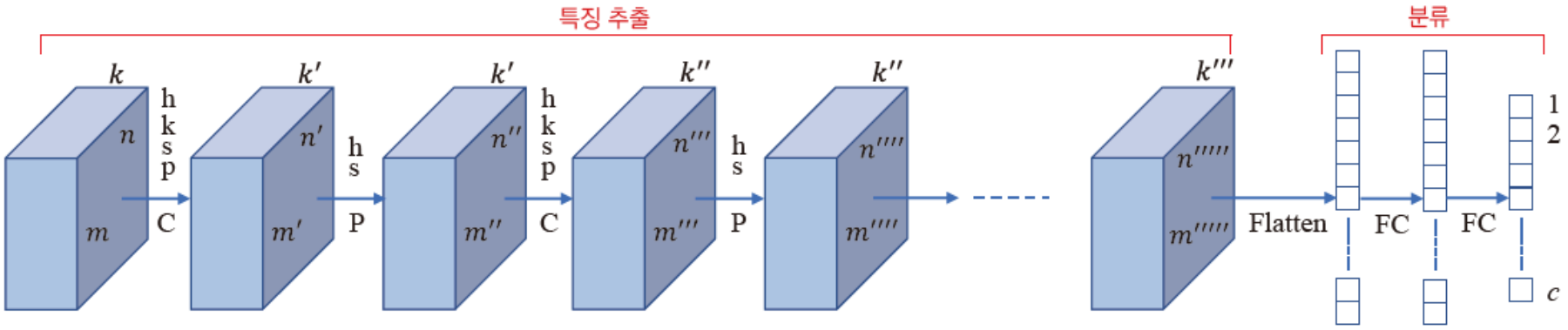


그림 8-9 풀링층(2×2 필터로 최대 풀링 적용, 보폭=2)

8.2.2 빌딩블록을 쌓아 만드는 컨볼루션 신경망

■ 빌딩블록 쌓기

- 보통 컨볼루션층과 풀링층을 번갈아 쌓음
- 풀링층에서는 텐서 깊이가 유지됨
- 신경망 앞 부분은 특징 추출, 뒷부분은 분류 담당



- C: 컨볼루션층
- P: 풀링층
- FC: 완전연결층
- Flatten: 2차원 구조를 1차원 구조로 변환
- h: 커널 크기
- k: 커널 개수
- s: 보폭
- p: 덧대기

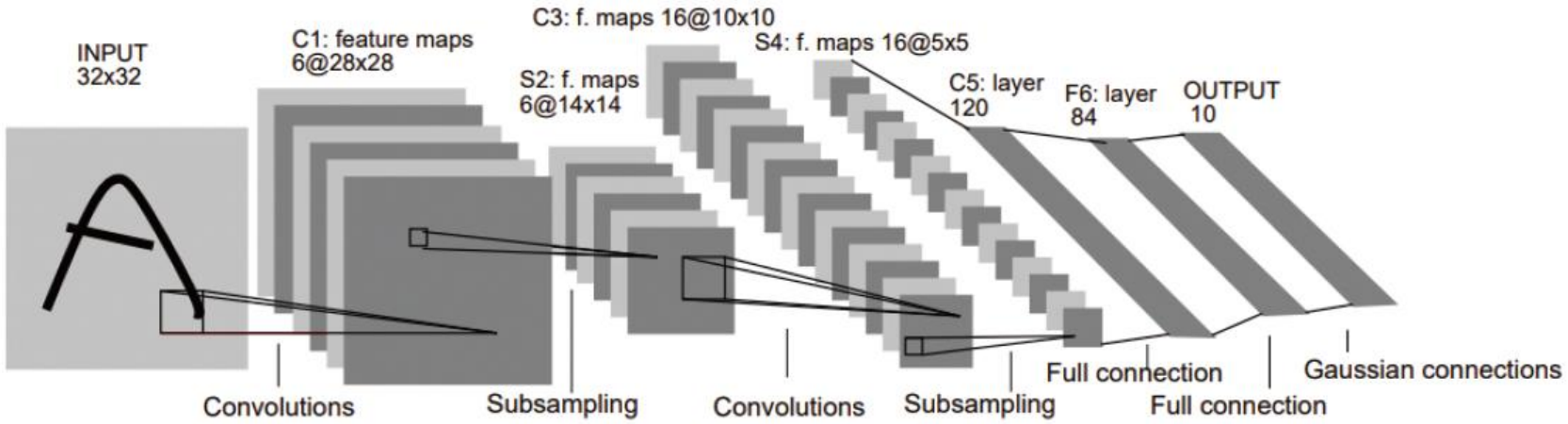
그림 8-10 컨볼루션층과 풀링층을 번갈아 쌓아 만드는 컨볼루션 신경망의 전형적인 구조

8.2.2 빌딩블록을 쌓아 만드는 컨볼루션 신경망

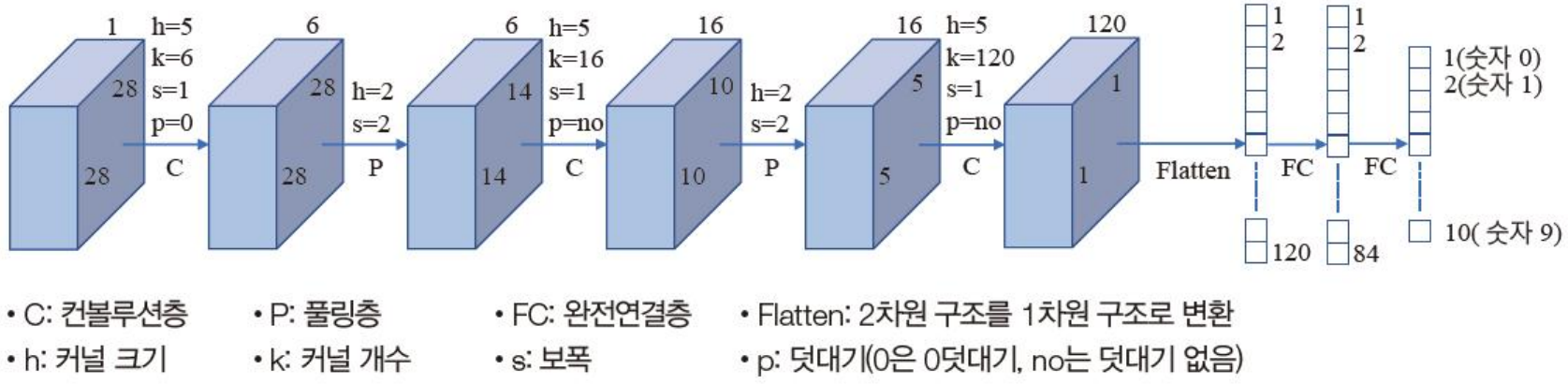
■ LeNet-5 사례[LeCun1998]

- C-P-C-P-C-FC-FC 구조
- 가중치 집합
  - 첫번째 컨볼루션층은 (5*5*1+1)*6개의 가중치
  - 두번째 컨볼루션층은 (5*5*6+1)*16
  - 세번째 컨볼루션층은 (5*5*16+1)*120
  - 첫번째 완전연결층은 (120+1)*84
  - 두번째 완전연결층은 (84+1)*10
  - 총 61,706개의 가중치

8.2.2 빌딩블록을 쌓아 만드는 컨볼루션 신경망



(a) [LeCun1998]의 그림



- C: 컨볼루션층
- P: 풀링층
- FC: 완전연결층
- Flatten: 2차원 구조를 1차원 구조로 변환
- h: 커널 크기
- k: 커널 개수
- s: 보폭
- p: 덧대기(0은 0덧대기, no는 덧대기 없음)

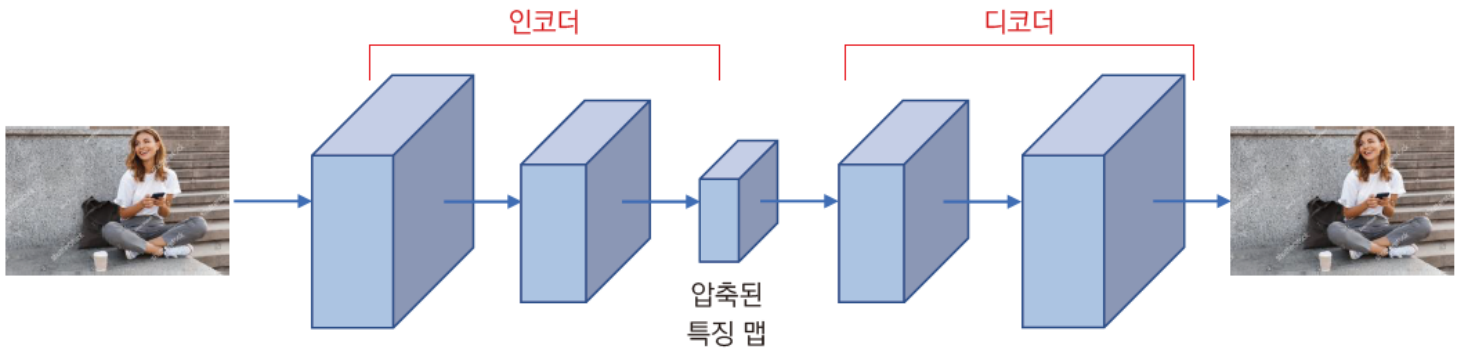
(b) [그림 8-10] 표기에 따른 그림

그림 8-11 LeNet-5의 구조

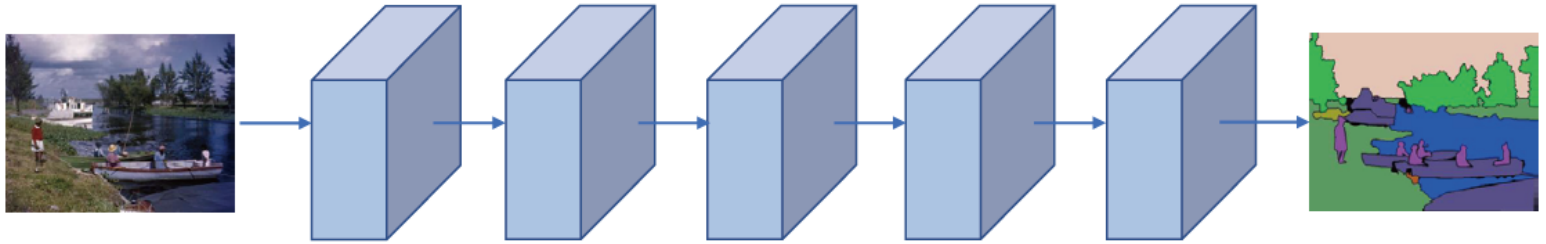
8.2.2 빌딩블록을 쌓아 만드는 컨볼루션 신경망

■ 유연한 구조

- 문제에 따라 다양한 모양으로 조립 가능
- 예) 오토인코더: 입력과 출력이 같은 신경망 (비지도 학습)
- 예) 분할을 위한 신경망: 출력은 분할 맵



(a) 오토인코더



(b) 영상 분할을 위한 컨볼루션 신경망

그림 8-12 컨볼루션 신경망의 유연한 구조

8.3 컨볼루션 신경망의 학습

■ 역전파 학습 알고리즘 사용(다층 퍼셉트론과 비슷)

- 컨볼루션층의 커널 화소와 온전연결층의 에지가 가중치에 대당
- 풀링층은 가중치 없음

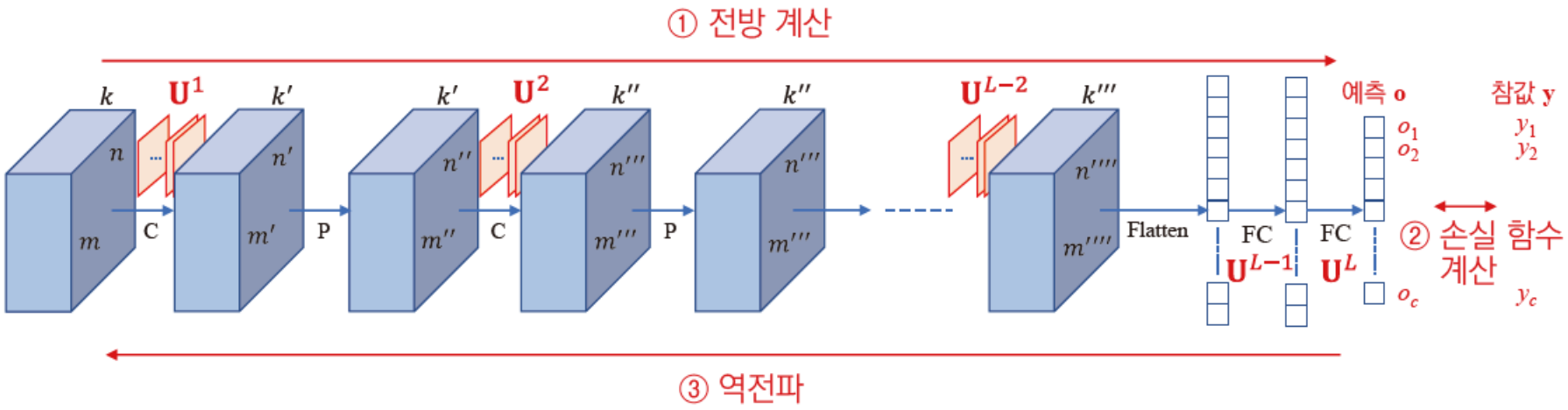


그림 8-13 컨볼루션 신경망을 학습하기 위한 역전파 알고리즘

8.3 컨볼루션 신경망의 학습

■ 특징 학습feature learning

- 학습 알고리즘은 주어진 데이터셋을 인식하는데 최적인 필터를 알아냄

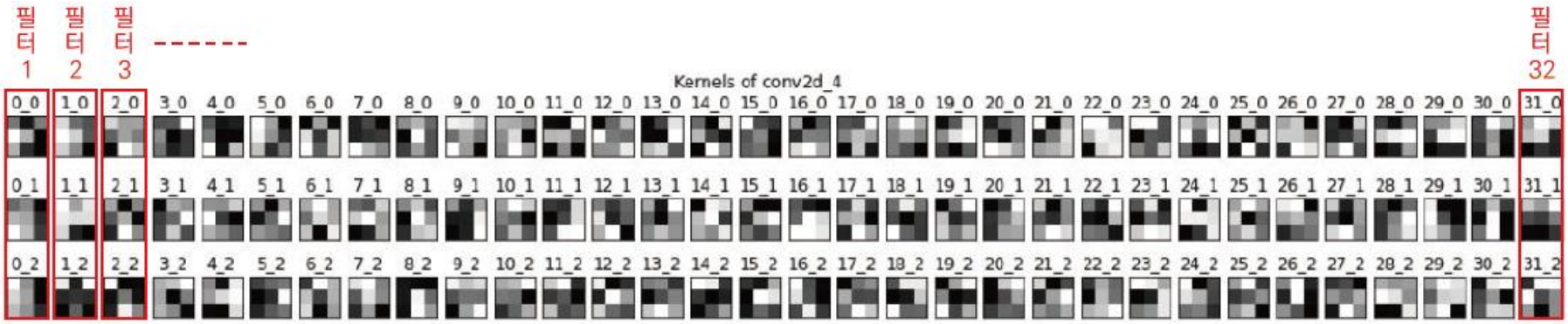


그림 8-14 CIFAR-10 데이터셋으로 학습한 컨볼루션 신경망의 최적 필터

8.3 컨볼루션 신경망의 학습

■ 통째 학습end-to-end learning

- 특징 학습과 분류기 학습을 한꺼번에 진행

(a) 수작업 특징을 사용하는 고전적 패러다임 (b) 통째 학습을 사용하는 딥러닝 패러다임

그림 8-15 딥러닝에 의한 컴퓨터 비전 방법론의 대전환

■ 컨볼루션 신경망이 우수한 이유

- 데이터의 원래 구조를 유지
- 특징 학습을 통해 최적의 특징을 추출
- 신경망의 깊이를 깊게 함

8.4 컨볼루션 신경망 구현

- 텐서플로를 이용한 컨볼루션 신경망 구현은 쉽다.
- LeNet-5 재현부터 시작

8.4.1 LeNet-5 재현

프로그램 8-1 LeNet-5로 MNIST 인식하기

```
01  import numpy as np
02  import tensorflow as tf
03  import tensorflow.keras.datasets as ds
04
05  from tensorflow.keras.models import Sequential
06  from tensorflow.keras.layers import Conv2D,MaxPooling2D,Flatten,Dropout,Dense
07  from tensorflow.keras.optimizers import Adam
08
09  (x_train,y_train),(x_test,y_test)=ds.mnist.load_data()
10  x_train=x_train.reshape(60000,28,28,1)
11  x_test=x_test.reshape(10000,28,28,1)
12  x_train=x_train.astype(np.float32)/255.0
13  x_test=x_test.astype(np.float32)/255.0
14  y_train=tf.keras.utils.to_categorical(y_train,10)
15  y_test=tf.keras.utils.to_categorical(y_test,10)
16
```

2차원 구조로 변환

데이터 준비

8.4.1 LeNet-5 재현

```
17  cnn=Sequential()
18  cnn.add(Conv2D(6,(5,5),padding='same',activation='relu',input_
    shape=(28,28,1)))
19  cnn.add(MaxPooling2D(pool_size=(2,2),strides=2))
20  cnn.add(Conv2D(16,(5,5),padding='valid',activation='relu'))
21  cnn.add(MaxPooling2D(pool_size=(2,2),strides=2))
22  cnn.add(Conv2D(120,(5,5),padding='valid',activation='relu'))
23  cnn.add(Flatten())
24  cnn.add(Dense(units=84,activation='relu'))
25  cnn.add(Dense(units=10,activation='softmax'))
26
27  cnn.compile(loss='categorical_crossentropy',optimizer=Adam
    (learning_rate=0.001),metrics=['accuracy'])
28  cnn.fit(x_train,y_train,batch_size=128,epochs=30,validation_
    data=(x_test,y_test),verbose=2) ①
29
30  res=cnn.evaluate(x_test,y_test,verbose=0)
31  print('정확률=',res[1]*100) ②
```

컨볼루션층(5*5 커널을 6개 사용)

1차원 구조로 변환

모델 선택 (신경망 구조 설계)

학습

예측(성능 측정)

8.4.1 LeNet-5 재현

```
Epoch 1/30 ①
469/469 - 3s - loss: 0.2555 - accuracy: 0.9212 - val_loss: 0.0794 - val_accuracy:
0.9742 - 3s/epoch - 6ms/step
Epoch 2/30
469/469 - 2s - loss: 0.0671 - accuracy: 0.9786 - val_loss: 0.0536 - val_accuracy:
0.9826 - 2s/epoch - 4ms/step
…
Epoch 29/30
469/469 - 2s - loss: 0.0037 - accuracy: 0.9988 - val_loss: 0.0506 - val_accuracy:
0.9902 - 2s/epoch - 5ms/step
Epoch 30/30
469/469 - 2s - loss: 0.0059 - accuracy: 0.9981 - val_loss: 0.0430 - val_accuracy:
0.9910 - 2s/epoch - 5ms/step
정확률= 99.09999966621399 ②
```

[프로그램 7-5]의 깊은 다층 퍼셉트론보다 정확률 0.68% 향상

8.4.2 자연 영상 인식

■ CIFAR-10을 컨볼루션 신경망으로 인식

프로그램 8-2 컨볼루션 신경망으로 자연 영상 인식하기

```
01  import numpy as np
02  import tensorflow as tf
03  import tensorflow.keras.datasets as ds
04
05  from tensorflow.keras.models import Sequential
06  from tensorflow.keras.layers import Conv2D,MaxPooling2D,Flatten,Dense,Dropout
07  from tensorflow.keras.optimizers import Adam
08
09  (x_train,y_train),(x_test,y_test)=ds.cifar10.load_data()
10  x_train=x_train.astype(np.float32)/255.0
11  x_test=x_test.astype(np.float32)/255.0
12  y_train=tf.keras.utils.to_categorical(y_train,10)
13  y_test=tf.keras.utils.to_categorical(y_test,10)
```

8.4.2 자연 영상 인식

규제 기법인 드롭아웃 적용

```
14
15  cnn=Sequential()
16  cnn.add(Conv2D(32,(3,3),activation='relu',input_shape=(32,32,3)))
17  cnn.add(Conv2D(32,(3,3),activation='relu'))
18  cnn.add(MaxPooling2D(pool_size=(2,2)))
19  cnn.add(Dropout(0.25))
20  cnn.add(Conv2D(64,(3,3),activation='relu'))
21  cnn.add(Conv2D(64,(3,3),activation='relu'))
22  cnn.add(MaxPooling2D(pool_size=(2,2)))
23  cnn.add(Dropout(0.25))
24  cnn.add(Flatten())
25  cnn.add(Dense(units=512,activation='relu'))
26  cnn.add(Dropout(0.5))
27  cnn.add(Dense(units=10,activation='softmax'))
28
29  cnn.compile(loss='categorical_crossentropy',optimizer=Adam(learning_
    rate=0.001),metrics=['accuracy'])
30  hist=cnn.fit(x_train,y_train,batch_size=128,epochs=100,validation_data=(x_
    test,y_test),verbose=2) ①
31
```

8.4.2 자연 영상 인식

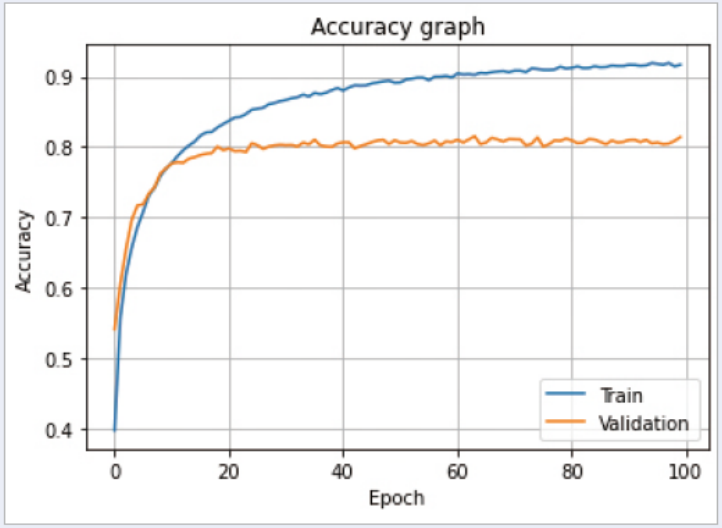
```
32  res=cnn.evaluate(x_test,y_test,verbose=0)
33  print('정확률=',res[1]*100) ②
34
35  import matplotlib.pyplot as plt
36
37  plt.plot(hist.history['accuracy']) ③
38  plt.plot(hist.history['val_accuracy'])
39  plt.title('Accuracy graph')
40  plt.ylabel('Accuracy')
41  plt.xlabel('Epoch')
42  plt.legend(['Train','Validation'])
43  plt.grid()
44  plt.show()
45
46  plt.plot(hist.history['loss']) ④
47  plt.plot(hist.history['val_loss'])
48  plt.title('Loss graph')
49  plt.ylabel('Loss')
50  plt.xlabel('Epoch')
51  plt.legend(['Train','Validation'])
52  plt.grid()
53  plt.show()
```

8.4.2 자연 영상 인식

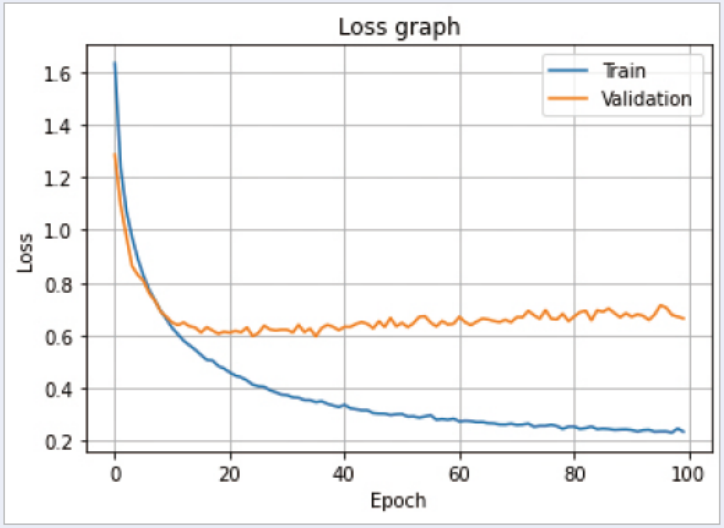
```
Epoch 1/100 ①
391/391 - 3s - loss: 1.6325 - accuracy: 0.3974 - val_loss: 1.2854 - val_accuracy:
0.5411 - 3s/epoch - 9ms/step
Epoch 2/100
391/391 - 3s - loss: 1.2396 - accuracy: 0.5573 - val_loss: 1.0960 - val_accuracy:
0.6054 - 3s/epoch - 7ms/step
…
Epoch 100/100
391/391 - 3s - loss: 0.2350 - accuracy: 0.9168 - val_loss: 0.6647 - val_accuracy:
0.8140 - 3s/epoch - 7ms/step
정확률= 81.40000104904175 ②
```

[프로그램 7-6]의 깊은 다층 퍼셉트론보다 정확률 26.24% 향상

③



④



8.4.3 텐서플로 프로그래밍

■ 텐서플로 프로그래밍의 핵심인 네 가지 모듈

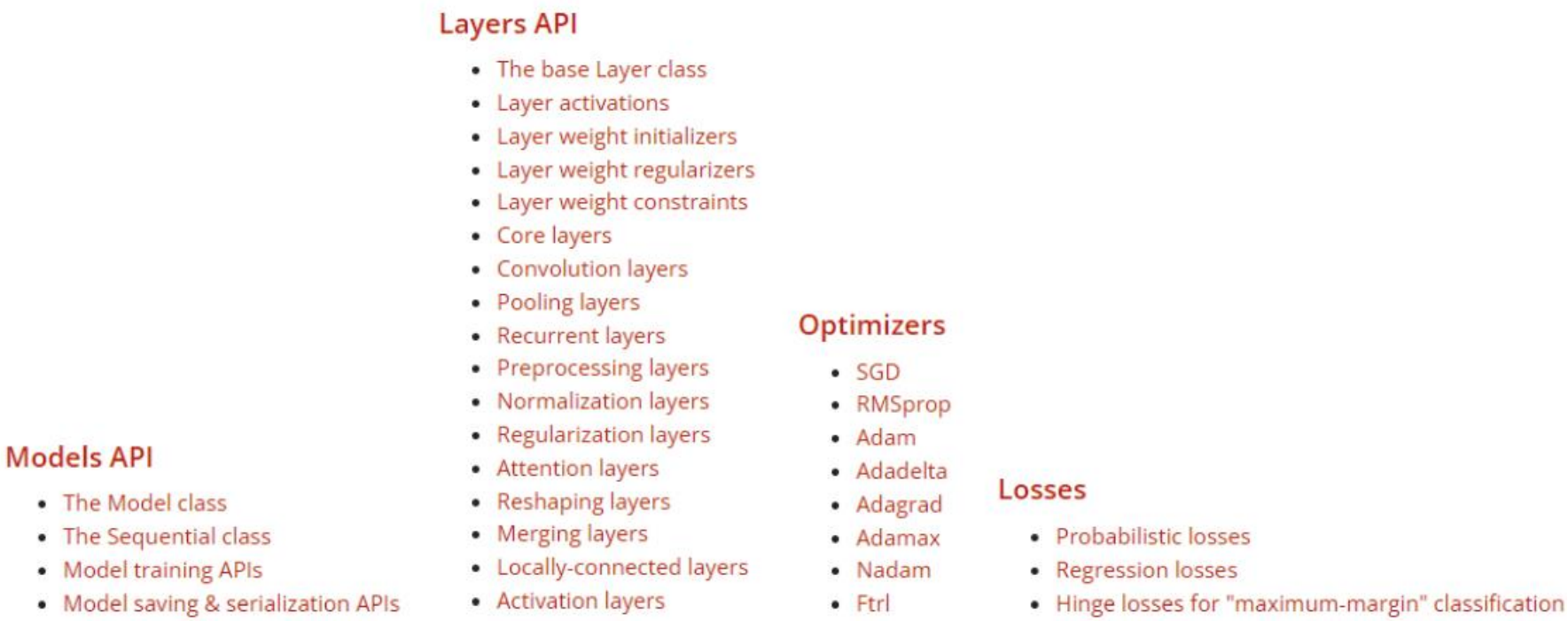


그림 8-16 케라스에서 제공하는 models, layers, optimizers, losses 모듈의 API(https://keras.io/api)

8.4.3 텐서플로 프로그래밍

■ 모델 생성하는 models 모듈

- Sequential과 Functional API
  - Sequential은 한 갈래 텐서가 끝까지 흐르는 경우
  - Functional API는 텐서가 여러 갈래로 나뉘는 경우

```
model=Sequential()
model.add(Conv2D(32,3,3), input_shape(32,32,3))
model.add(Conv2D(64,3,3))
model.add(Flatten())
model.add(Dense(10,activation='softmax')
```

Sequential로 코딩

```
input=Input(shape=(32,32,3))
x1=Conv2D(32,3,3)(input)
x2=Conv2D(64,3,3)(x1)
x3=Flatten()(x2)
output=Dense(10,activation='softmax')(x3)
model=Model(input,output)
```

Functional API로 코딩

8.4.3 텐서플로 프로그래밍

■ 층을 쌓는 layers 모듈

- 완전연결층 Dense, 컨볼루션층 Conv2D, 최대 풀링층 MaxPooling2D, …

■ 손실 함수를 위한 losses 모듈

- 평균제곱오차 MSE, 교차 엔트로피 categorical_crossentropy, 분할을 위한 focal, …

■ 옵티마이저를 위한 optimizers 모듈

- SGD, Adam, AdaGrad, RMSprop 등

8.5 [비전 에이전트 6] 우편번호 인식기 v.2

■ [프로그램 7-7]의 우편번호 인식기 v.1을 컨볼루션 신경망으로 버전업

- [프로그램 8-3]으로 높은 성능의 학습 모델 확보

프로그램 8-3 필기 숫자 인식기 성능 향상하기

```
    01~15행은 [프로그램 8-1]과 같음

17  cnn=Sequential()
18  cnn.add(Conv2D(32,(3,3),activation='relu',input_shape=(28,28,1)))
19  cnn.add(Conv2D(32,(3,3),activation='relu'))
20  cnn.add(MaxPooling2D(pool_size=(2,2)))
21  cnn.add(Dropout(0.25))
22  cnn.add(Conv2D(64,(3,3),activation='relu'))
23  cnn.add(Conv2D(64,(3,3),activation='relu'))
24  cnn.add(MaxPooling2D(pool_size=(2,2)))
25  cnn.add(Dropout(0.25))
26  cnn.add(Flatten())
27  cnn.add(Dense(units=512,activation='relu'))
28  cnn.add(Dropout(0.5))
29  cnn.add(Dense(units=10,activation='softmax'))
```

8.5 [비전 에이전트 6] 우편번호 인식기 v.2

```
30
31  cnn.compile(loss='categorical_crossentropy',optimizer=Adam(learning_
    rate=0.001),metrics=['accuracy'])
32  hist=cnn.fit(x_train,y_train,batch_size=128,epochs=100,validation_data=(x_
    test,y_test),verbose=2)
33
34  cnn.save('cnn_v2.h5')
35
36  res=cnn.evaluate(x_test,y_test,verbose=0)
37  print('정확률=',res[1]*100) ①
```

100세대를 수행하여
성능을 최대로 끌어올림

비전 에이전트에 쓰기 위해 학습된 모델을 저장함

```
...
Epoch 100/100
469/469 - 3s - loss: 0.0341 - accuracy: 0.9904 - val_loss: 0.0191 - val_accuracy:
0.9958 - 3s/epoch - 7ms/step
정확률= 99.58000183105469 ①
```

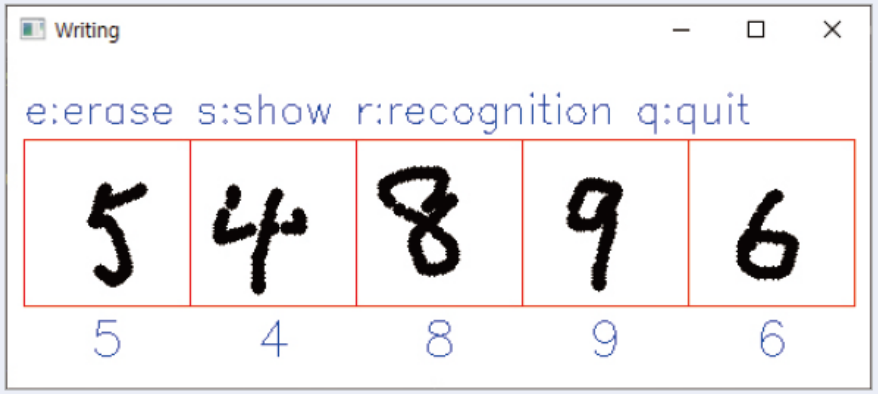
8.5 [비전 에이전트 6] 우편번호 인식기 v.2

프로그램 8-4 우편번호 인식기 v.2(CNN 버전)

[프로그램 7-7]과 같음 (단 아래와 같이 7행에서 'dmlp_trained.h5'를 'cnn_v2.h5'로 바꾸고, 37행에서 (5,784)을 (5,28,28)로 바꿈)

```
07  model=tf.keras.models.load_model('cnn_v2.h5')

37  numerals=numerals.reshape(5,28,28,1)
```

[프로그램 8-3]이 저장해둔 모델을 읽어옴



실용적 시스템을 구현한다면 데이터셋 시프트dataset shift를 해소하여 추가적인 성능 향상 가능

8.6 딥러닝의 학습 알고리즘 향상

■ **8.1절에서 설명한 딥러닝의 성공 요인**

- 데이터셋의 커짐
- 계산이 빨라짐(GPU)
- 학습 알고리즘의 향상

■ **여기서는 세번째 주제를 설명**

- 손실 함수
- 옵티마이저
- 규제

8.6.1 손실 함수

■ **신경망 초기에는 주로 평균제곱오차(MSE)를 사용**

$$J(\mathbf{W}) = \frac{1}{|M|}\sum_{\mathbf{x}\in M}\|\mathbf{y}-\mathbf{o}\|_2^2 \qquad (7.23)$$

■ **딥러닝에서는**

- 분류를 위해서는 교차 엔트로피를 주로 사용
- 과업에 따른 다양한 손실 함수 (물체 검출, 분할, 추적, ...) ← 9장에서 소개

8.6.1 손실 함수

■ 교차 엔트로피

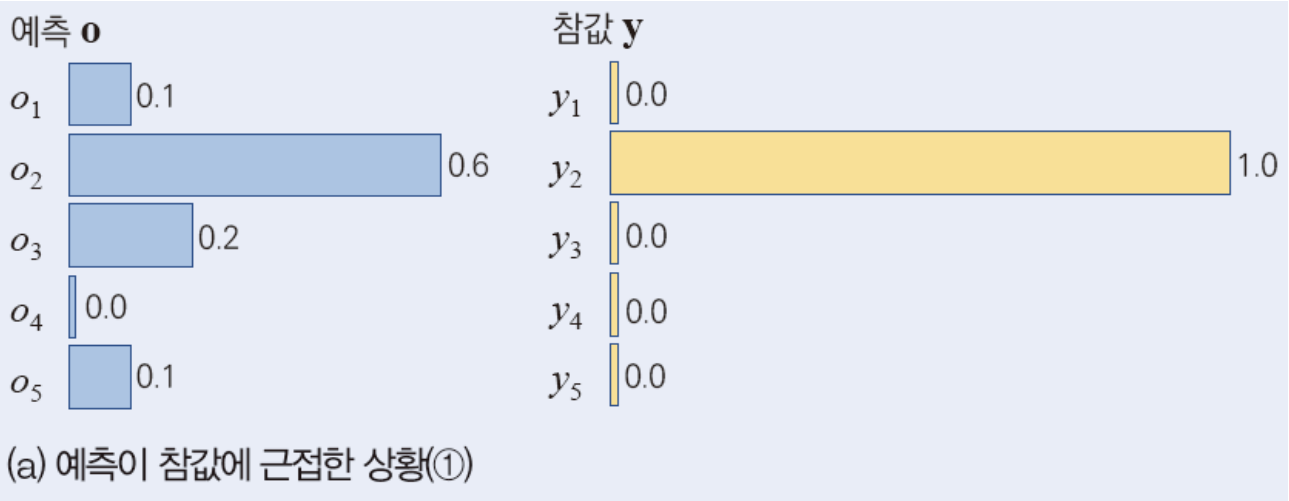
- 참값 벡터 $\mathbf{y} = (y_1, y_2, \cdots, y_c)$와 예측 벡터 $\mathbf{o} = (o_1, o_2, \cdots, o_c)$를 확률 분포로 간주
- 교차 엔트로피는 두 확률 분포의 다른 정도를 측정해줌

교차 엔트로피: $$e = -\sum_{i=1,c} y_i \log(o_i) \qquad (8.1)$$

TIP http://neuralnetworksanddeeplearning.com/chap3.html에서는 평균제곱오차의 불공정한 상황을 애니메이션을 활용하여 설명한다.

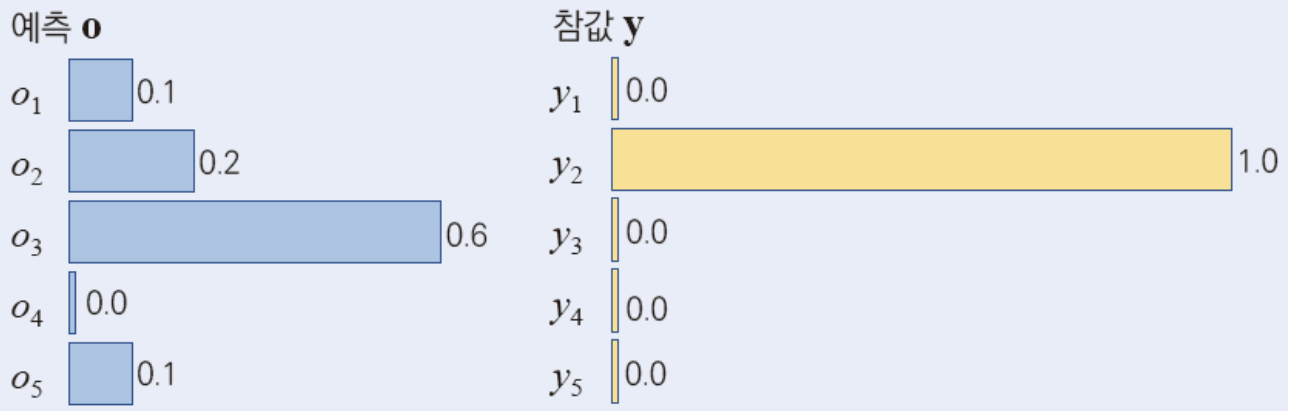
8.6.1 손실 함수

■ [예시 8-3] 평균제곱오차와 교차 엔트로피 손실 함수



(a) 예측이 참값에 근접한 상황(①)

신경망이 맞힌 상황
(**o**와 **y**가 근접)



(b) 예측이 참값에서 벗어난 상황(②)

신경망이 틀린 상황
(**o**와 **y**가 멈)

그림 8-17 손실 함수 계산

8.6.1 손실 함수

평균제곱오차

$$\begin{cases} \text{상황 ①: } \mathrm{MSE} = (0.0-0.1)^2 + (1.0-0.6)^2 + (0.0-0.2)^2 + (0.0-0.0)^2 + (0.0-0.1)^2 = 0.22 \\ \text{상황 ②: } \mathrm{MSE} = (0.0-0.1)^2 + (1.0-0.2)^2 + (0.0-0.6)^2 + (0.0-0.0)^2 + (0.0-0.1)^2 = 1.02 \end{cases}$$

교차 엔트로피

$$\begin{cases} \text{상황 ①: } \mathrm{CE} = -\left(0.0\log(0.1) + 1.0\log(0.6) + 0.0\log(0.2) + 0.0\log(0.0) + 0.0\log(0.1)\right) = 0.5108 \\ \text{상황 ②: } \mathrm{CE} = -\left(0.0\log(0.1) + 1.0\log(0.2) + 0.0\log(0.6) + 0.0\log(0.0) + 0.0\log(0.1)\right) = 1.609 \end{cases}$$

8.6.1 손실 함수

■ Focal 손실 함수

- 부류 불균형이 심한 경우에 효과적
- 의료 영상 분할을 다룬 RetinaNet 논문이 제안 [Lin2017]

참값이 물체인 화소: $\mathbf{o}=(p,1-p),\ \mathbf{y}=(1,0)$

참값이 배경인 화소: $\mathbf{o}=(p,1-p),\ \mathbf{y}=(0,1)$

$$p_t=\begin{cases} p, & \text{참값이 물체인 화소} \\ 1-p, & \text{참값이 배경인 화소} \end{cases} \qquad (8.2)$$

$$e=-\log(p_t) \qquad (8.3)$$

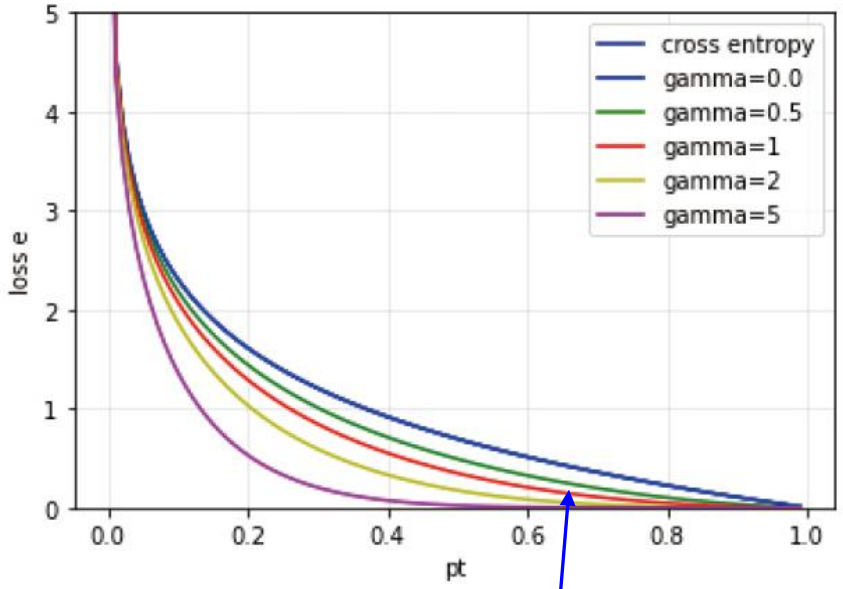


그림 8-18 부류가 2개인 경우의 교차 엔트로피와 Focal 손실 함수

Focal 손실: $e=-(1-p_t)^{\gamma}\log(p_t)$ (8.4)

γ가 클수록 0에 빠르게 가까워짐

8.6.2 옵티마이저

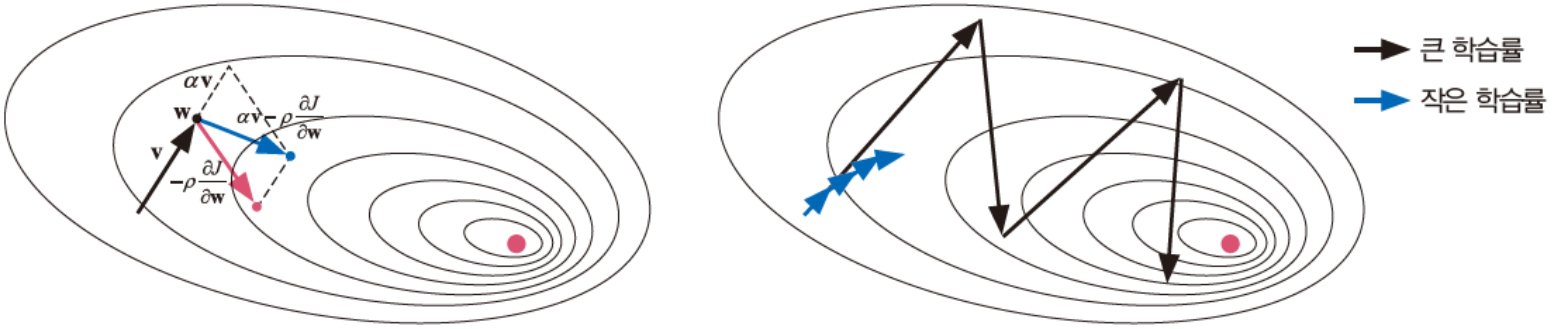
- **기본 옵티마이저 SGD를 개선하는 전략**
  - 모멘텀 적용
  - 적응적 학습률 적용

8.6.2 옵티마이저

■ 모멘텀

- 이전 운동량이 현재에 영향을 미치는 물리 법칙
- 가중치 변경 이력이 현재 가중치에 영향을 미침

$$\text{SGD: } \mathbf{W} = \mathbf{W} - \rho \frac{\partial J}{\partial \mathbf{W}} \longrightarrow \text{모멘텀을 적용한 SGD: } \left.\begin{aligned} \mathbf{V} &= \alpha \mathbf{V} - \rho \frac{\partial J}{\partial \mathbf{W}} \\ \mathbf{W} &= \mathbf{W} + \mathbf{V} \end{aligned}\right\} \quad (8.5)$$



(a) 모멘텀 아이디어 (b) 학습률에 따른 수렴 특성

그림 8-19 딥러닝에서 사용하는 옵티마이저

```
tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.0, nesterov=False, name="SGD",
                        **kwargs)
```

8.6.2 옵티마이저

■ 적응적 학습률

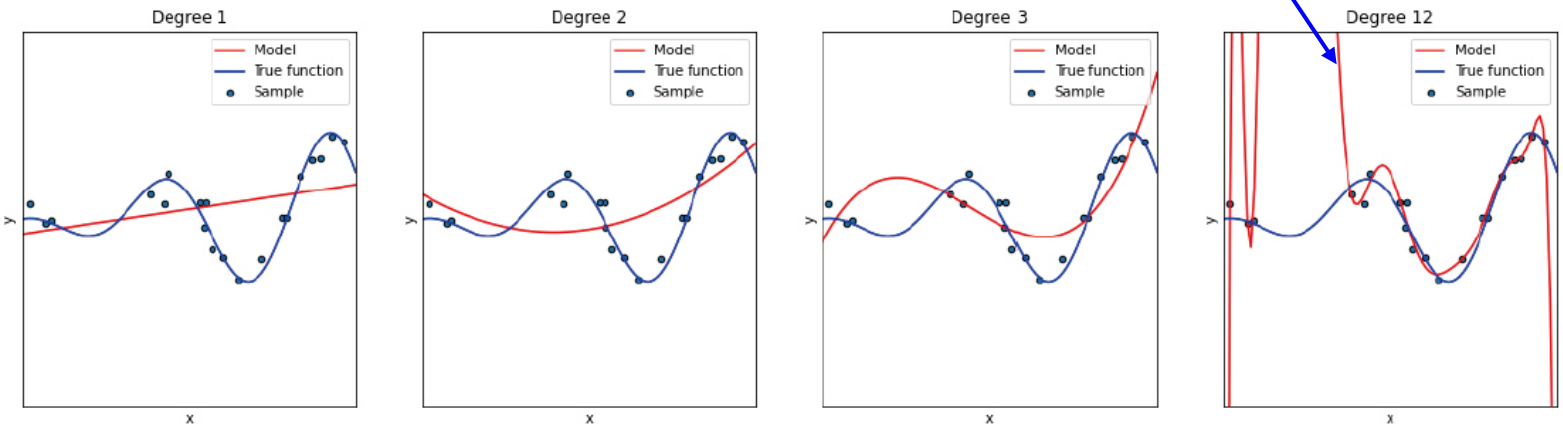
- 식 (8.5)의 표준 SGD는 고정된 학습률 사용
  - 작은 학습률 (0.001, 0.0001과 같은) 사용하여 보수적으로 이동
- 적응적 학습률에서는 상황에 따라 학습률을 조정함[Ruder2016]
  - AdaGrad: 이전 그레이디언트를 누적한 정보를 이용해 학습률을 적응적으로 결정
  - RMSprop: 그레이디언트를 누적할 때 오래될수록 영향력을 감소하여 AdaGrad 개선
  - Adam: RMSprop에 식 (8.5)의 모멘텀을 적용

```
tf.keras.optimizers.Adagrad(learning_rate=0.001, initial_accumulator_value=0.1,
                            epsilon=1e-07, name="Adagrad", **kwargs)
tf.keras.optimizers.RMSprop(learning_rate=0.001, rho=0.9, momentum=0.0,
                            epsilon=1e-07, centered=False, name="RMSprop", **kwargs)
tf.keras.optimizers.Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999,
                         epsilon=1e-07, amsgrad=False, name="Adam", **kwargs)
```

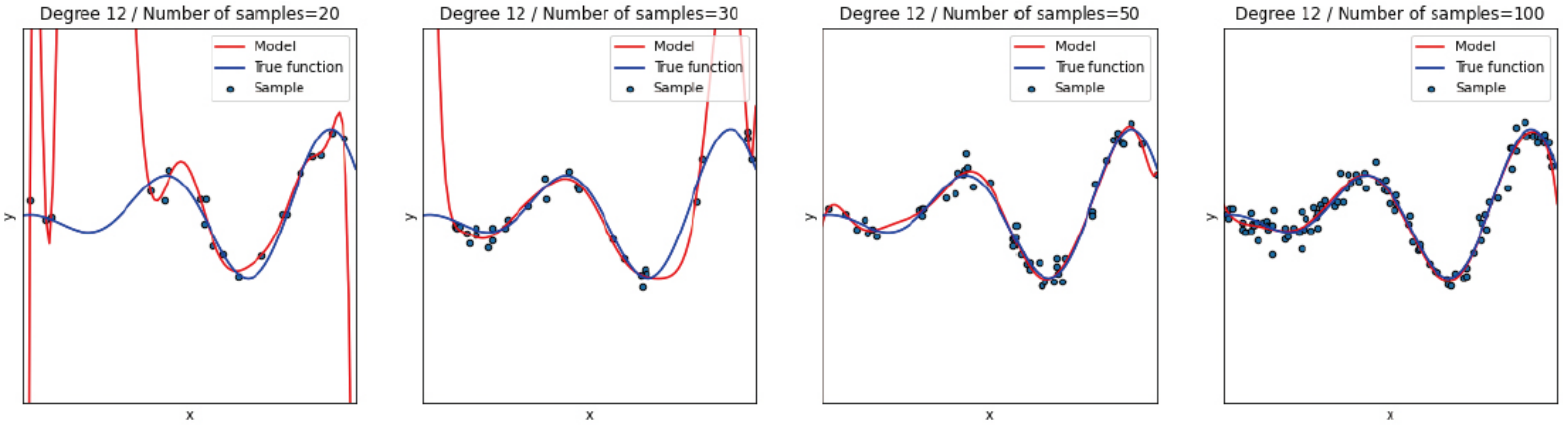
8.6.3 규제

훈련에 참여하지 않은 새로운 데이터에 대해 낮은 성능. 즉 일반화 능력이 약함

■ 과잉 적합이란



(a) 과소 적합과 과잉 적합 현상



(b) 데이터 증강을 통한 과잉 적합 방지

그림 8-20 과잉 적합 현상과 데이터 증강을 통한 방지

8.6.3 규제

■ 다양한 규제 기법

- 데이터 증강data augmentation
  - 훈련 집합을 조금씩 변형하여 인위적으로 늘림
  - 오프라인 방식과 온라인 방식(주로 온라인 사용)
- 드롭아웃
  - 특징 맵을 구성하는 요소 중 일부를 랜덤 선택하여 0으로 설정하여 학습에서 배제
  - 학습할 때만 적용하고 예측(추론) 과정에서는 적용 안함
- 조기 멈춤
  - 성능 향상이 없으면 설정한 세대 수 이전에 학습을 멈춤
- ....

8.6.3 규제

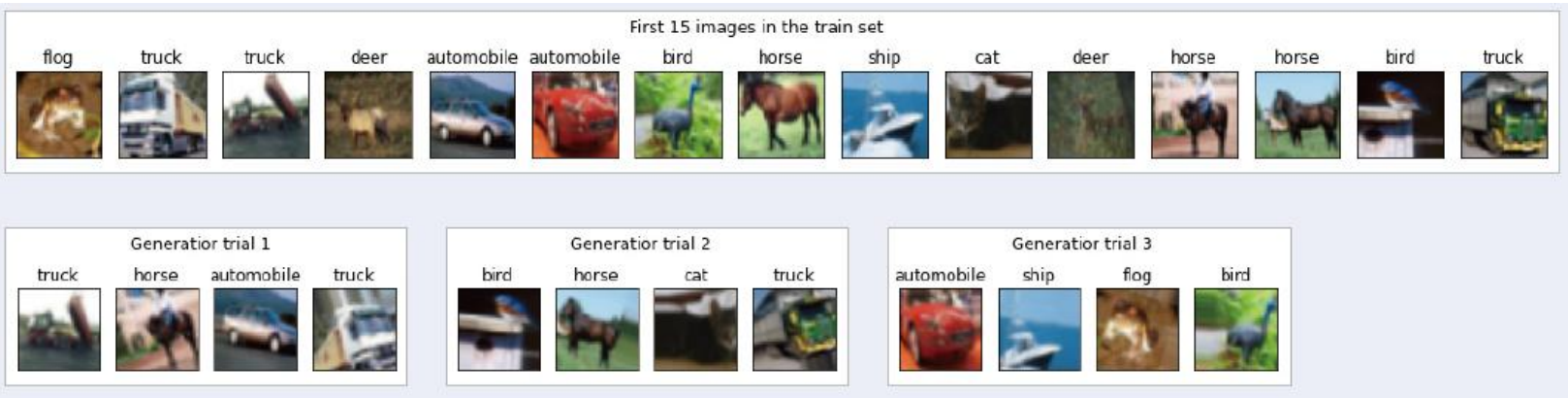
■ [프로그램 8-5]는 증강된 영상 확인하기

프로그램 8-5 증강된 영상 확인하기

```
01  import tensorflow.keras.datasets as ds
02  from tensorflow.keras.preprocessing.image import ImageDataGenerator
03  import matplotlib.pyplot as plt
04
05  (x_train,y_train),(x_test,y_test)=ds.cifar10.load_data()
06  x_train=x_train.astype('float32'); x_train/=255
07  x_train=x_train[0:15,]; y_train=y_train[0:15,]   # 앞 15개에 대해서만 증대 적용
08  class_names=['airplane','automobile','bird','cat','deer','dog','flog','horse',
    'ship','truck']
09
10  plt.figure(figsize=(20,2))
11  plt.suptitle("First 15 images in the train set")
12  for i in range(15):
13      plt.subplot(1,15,i+1)
14      plt.imshow(x_train[i])
15      plt.xticks([]); plt.yticks([])
16      plt.title(class_names[int(y_train[i])])
17  plt.show()
18
```

8.6.3 규제

```
19  batch_siz=4                                          # 한 번에 생성하는 양(미니 배치)
20  generator=ImageDataGenerator(rotation_range=20.0,width_shift_range=0.2,
    height_shift_range=0.2,horizontal_flip=True)
21  gen=generator.flow(x_train,y_train,batch_size=batch_siz)
22
23  for a in range(3):
24      img,label=gen.next()                             # 미니 배치만큼 생성
25      plt.figure(figsize=(8,2.4))
26      plt.suptitle("Generatior trial «+str(a+1))
27      for i in range(batch_siz):
28          plt.subplot(1,batch_siz,i+1)
29          plt.imshow(img[i])
30          plt.xticks([]); plt.yticks([])
31          plt.title(class_names[int(label[i])])
32      plt.show()
```



8.7 전이 학습

■ 전이 학습

- 어떤 도메인의 데이터로 학습한 모델을 다른 도메인에 적용하여 성능을 향상하는 방법

8.7.1 백본 모델

■ 텐서플로가 제공하는 사전 학습 모델

- 전이 학습할 때는 보통 이들 모델을 백본으로 사용

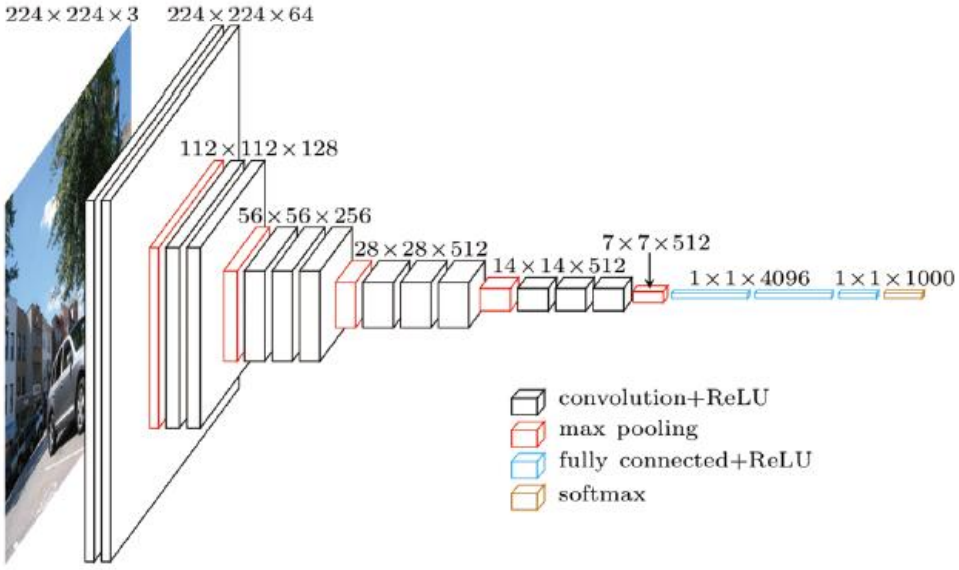
Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
Xception	88	79.0%	94.5%	22.9M	81	109.4	8.1
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
VGG19	549	71.3%	90.0%	143.7M	19	84.8	4.4
ResNet50	98	74.9%	92.1%	25.6M	107	58.2	4.6
ResNet50V2	98	76.0%	93.0%	25.6M	103	45.6	4.4
ResNet101	171	76.4%	92.8%	44.7M	209	89.6	5.2
ResNet101V2	171	77.2%	93.8%	44.7M	205	72.7	5.4
ResNet152	232	76.6%	93.1%	60.4M	311	127.4	6.5
ResNet152V2	232	78.0%	94.2%	60.4M	307	107.5	6.6
InceptionV3	92	77.9%	93.7%	23.9M	189	42.2	6.9
InceptionResNetV2	215	80.3%	95.3%	55.9M	449	130.2	10.0
MobileNet	16	70.4%	89.5%	4.3M	55	22.6	3.4
MobileNetV2	14	71.3%	90.1%	3.5M	105	25.9	3.8
DenseNet121	33	75.0%	92.3%	8.1M	242	77.1	5.4
DenseNet169	57	76.2%	93.2%	14.3M	338	96.4	6.3
DenseNet201	80	77.3%	93.6%	20.2M	402	127.2	6.7

그림 8-22 텐서플로에서 제공하는 사전 학습 모델의 일부(https://keras.io/api/applications)

8.7.1 백본 모델

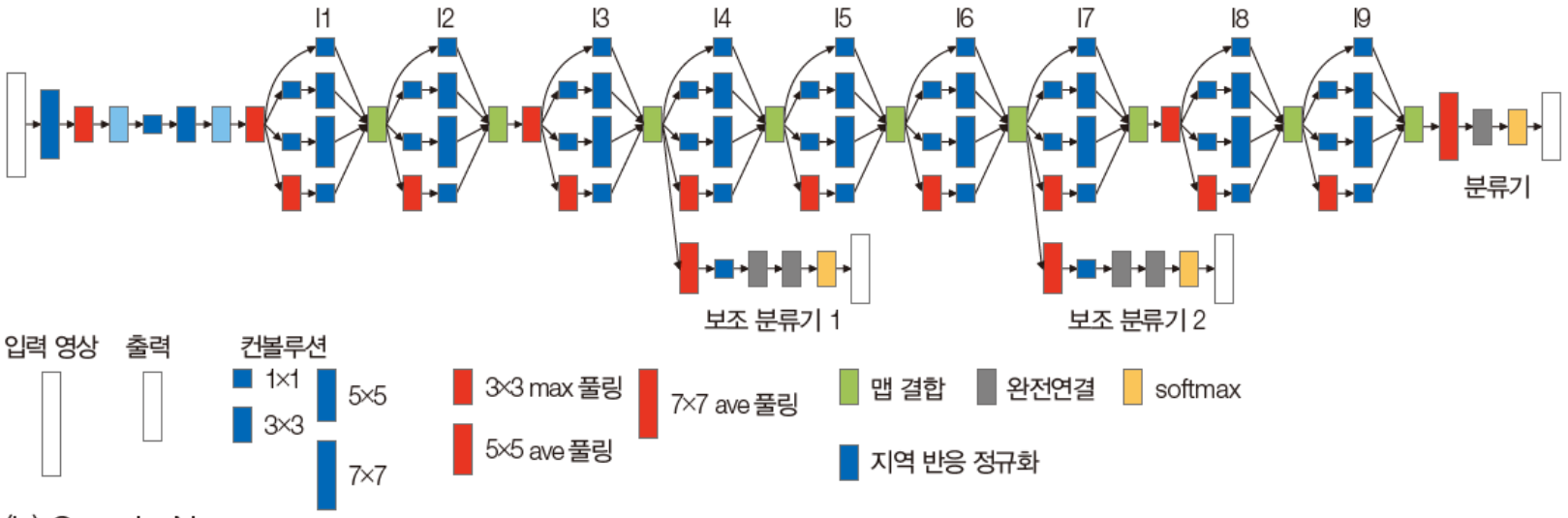
■ 유명한 사전 학습 모델

- VGGNet: 옥스퍼드 대학이 제작. 3*3 작은 마스크 사용하고 층 깊이를 16으로 깊게 만듦
- GoogLeNet: 구글이 제작. 네트워크 속의 네트워크 아이디어 적용
- ResNet: 마이크로소프트가 제작. 지름길 연결 적용

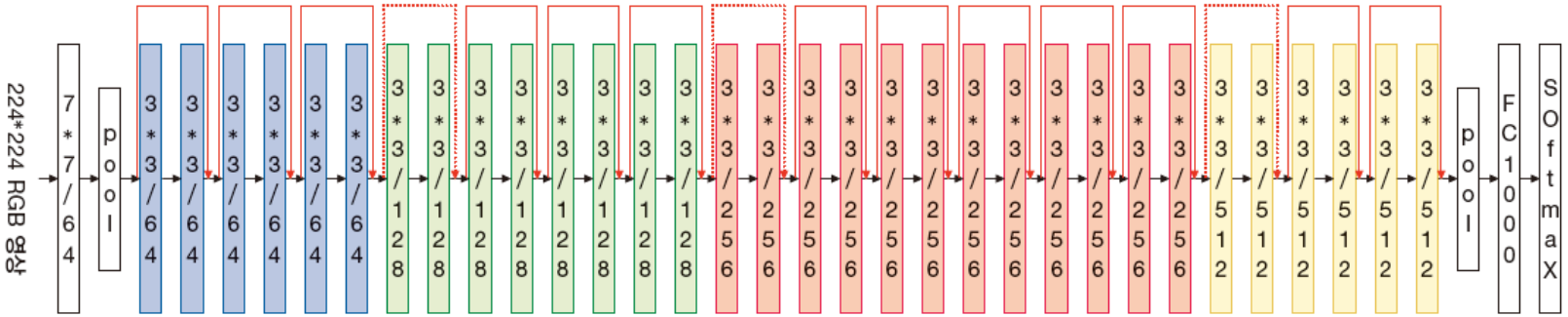


(a) VGGNet

8.7.1 백본 모델



(b) GoogLeNet



(c) ResNet

그림 8-23 VGGNet, GoogLeNet, ResNet의 구조

8.7.2 사전 학습 모델로 자연 영상 인식

프로그램 8-6 ResNet50으로 자연 영상 인식하기

ImageNet으로 학습한 ResNet50을 백본으로 사용

```
01  import cv2 as cv
02  import numpy as np
03  from tensorflow.keras.applications.resnet50 import ResNet50,preprocess_
    input,decode_predictions
04
05  model=ResNet50(weights='imagenet')
06
07  img=cv.imread('rabbit.jpg')
08  x=np.reshape(cv.resize(img,(224,224)),(1,224,224,3))
09  x=preprocess_input(x)
10
11  preds=model.predict(x)
12  top5=decode_predictions(preds,top=5)[0]
13  print('예측 결과:',top5)
14
15  for i in range(5):
16      cv.putText(img,top5[i][1]+':'+str(top5[i][2]),(10,20+i*20),cv.FONT_HERSHEY_
        SIMPLEX,0.5,(255,255,255),1)
17
18  cv.imshow('Recognition result',img)
19
20  cv.waitKey()
21  cv.destroyAllWindows()
```

8.7.2 사전 학습 모델로 자연 영상 인식

```
예측 결과: [('n02325366', 'wood_rabbit', 0.74258107), ('n02326432', 'hare',
0.24038698), ('n02328150', 'Angora', 0.008831766), ('n01877812', 'wallaby',
0.0026911795), ('n02356798', 'fox_squirrel', 0.0012295933)]
```

8.7.3 사전 학습 모델로 견종 인식

■ Stanford dogs 데이터셋

- 미세 분류fine-grained classification 문제
- 부류(개의 품종)가 120개이고 영상이 20,580장

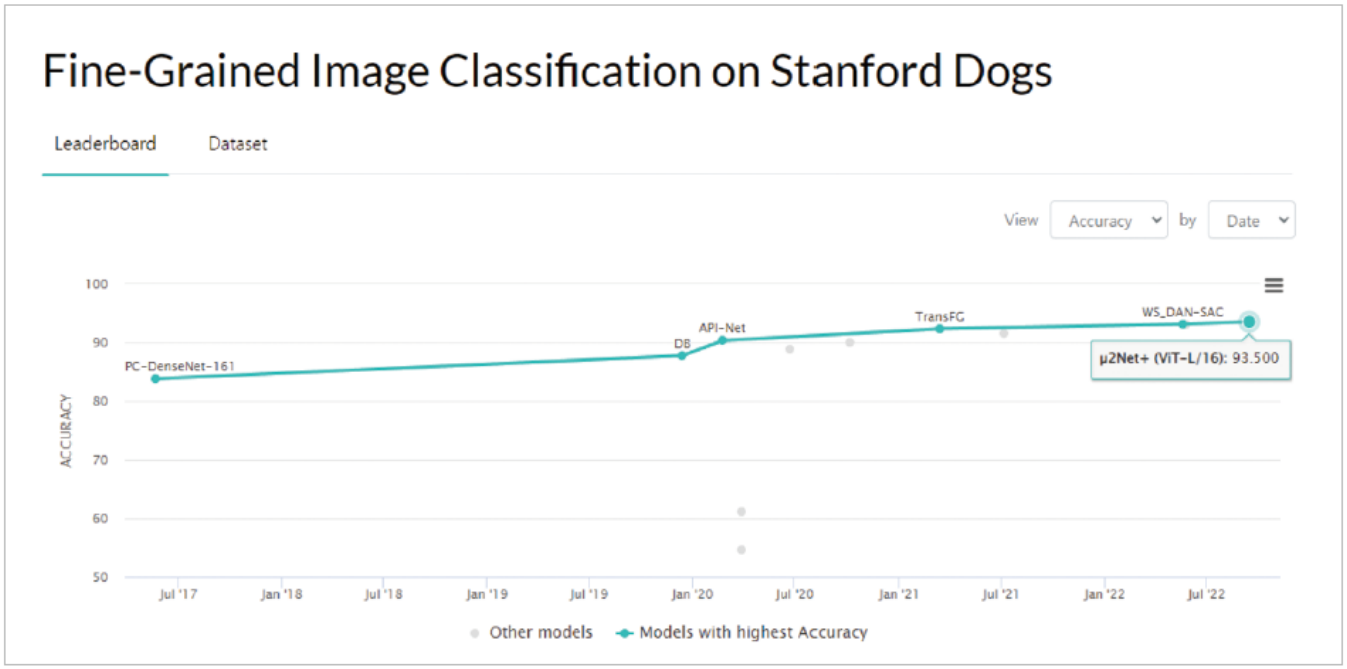


그림 8-24 Stanford dogs 데이터셋의 SOTA 성능

TIP PapersWithCode 사이트(https://paperswithcode.com)에서는 출판된 논문의 성능을 비교해 데이터셋에 대한 최고 성능을 그래프로 제시한다. Stanford dogs에 대한 SOTA는 https://paperswithcode.com/sota/fine-grained-image-classification-on-stanford-1에서 확인할 수 있다.

8.7.3 사전 학습 모델로 견종 인식

■ Stanford dogs 데이터셋 설치

1. 소스코드가 있는 폴더에 datasets/Stanford_dogs 폴더를 만든다.

2. http://vision.stanford.edu/aditya86/ImageNetDogs에 접속해 images.tar, annotations.tar, list.tar 파일을 1에서 만든 Stanford_dogs 폴더에 다운로드한다.

3. 세 파일의 압축을 해제하고 annotations, images, list 폴더를 확인한다.

4. images/images 폴더를 열어 'n02085620-Chihuahua'를 비롯한 120개 폴더가 있는지 확인한다. 폴더 이름 끝에 있는 Chihuahua는 개의 품종, 즉 부류 이름이다.

그림 8-25 Stanford dogs 데이터셋(왼쪽부터 Chihuahua, Japanese_spaniel, Maltese_dog, Pekinese, Shih-Tzu, Blenheim_spaniel)

8.7.3 사전 학습 모델로 견종 인식

프로그램 8-7 DenseNet121로 견종 인식하기

```
01  from tensorflow.keras.models import Sequential
02  from tensorflow.keras.layers import Flatten,Dense,Dropout,Rescaling
03  from tensorflow.keras.optimizers import Adam
04  from tensorflow.keras.applications.densenet import DenseNet121
05  from tensorflow.keras.utils import image_dataset_from_directory
06  import pathlib
07
08  data_path=pathlib.Path('datasets/stanford_dogs/images/images')
09
10  train_ds=image_dataset_from_directory(data_path,validation_split=0.2,subset
    ='training',seed=123,image_size=(224,224),batch_size=16) ①
11  test_ds=image_dataset_from_directory(data_path,validation_split=0.2,subset
    ='validation',seed=123,image_size=(224,224),batch_size=16)
12
13  base_model=DenseNet121(weights='imagenet',include_top=False,input_shape
    =(224,224,3))
14  cnn=Sequential()
15  cnn.add(Rescaling(1.0/255.0))
16  cnn.add(base_model)
17  cnn.add(Flatten())
18  cnn.add(Dense(1024,activation='relu'))
19  cnn.add(Dropout(0.75))
20  cnn.add(Dense(units=120,activation='softmax'))
```

폴더에서 데이터 읽기

분류 층을 배제

DenseNet121을 백본으로 사용

8.7.3 사전 학습 모델로 견종 인식

미세 조정 방식의 전이 학습:
학습률을 아주 작게 설정하여 특징 추출을 담당하는 층의 가중치를 유지

대신 세대 수를 늘려 오래 학습하는 전략

```
21
22  cnn.compile(loss='sparse_categorical_crossentropy',optimizer=Adam(learning_
    rate=0.000001),metrics=['accuracy'])
23  hist=cnn.fit(train_ds,epochs=200,validation_data=test_ds,verbose=2) ②
24
25  print('정확률=',cnn.evaluate(test_ds,verbose=0)[1]*100) ③
26
27  cnn.save('cnn_for_stanford_dogs.h5')        # 미세 조정된 모델을 파일에 저장
28
```

비전 에이전트에 쓰기 위해 학습된 모델을 저장함

8.7.3 사전 학습 모델로 견종 인식

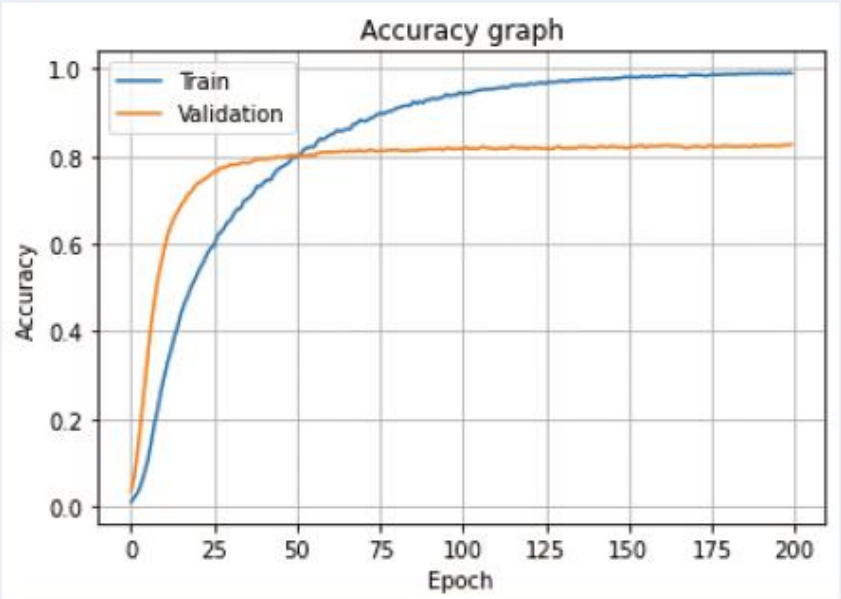
```
29  import pickle
30  f=open('dog_species_names.txt','wb')
31  pickle.dump(train_ds.class_names,f)
32  f.close()
33
34  import matplotlib.pyplot as plt
35
36  plt.plot(hist.history['accuracy']) ④
37  plt.plot(hist.history['val_accuracy'])
38  plt.title('Accuracy graph')
39  plt.ylabel('Accuracy')
40  plt.xlabel('Epoch')
41  plt.legend(['Train','Validation'])
42  plt.grid()
43  plt.show()
44
45  plt.plot(hist.history['loss']) ⑤
46  plt.plot(hist.history['val_loss'])
47  plt.title('Loss graph')
48  plt.ylabel('Loss')
49  plt.xlabel('Epoch')
50  plt.legend(['Train','Validation'])
51  plt.grid()
52  plt.show()
```

8.7.3 사전 학습 모델로 견종 인식

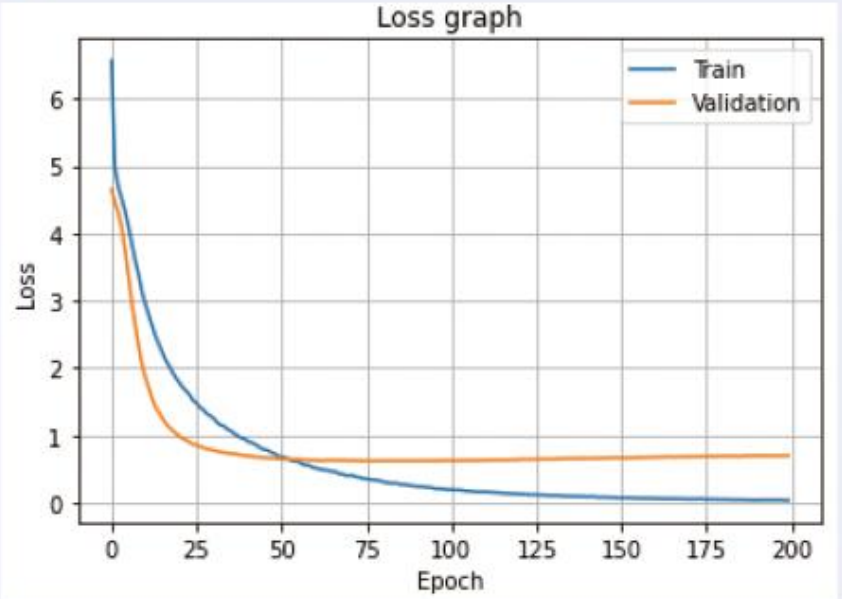
```
Found 20580 files belonging to 120 classes. ①
Using 16464 files for training.
Found 20580 files belonging to 120 classes.
Using 4116 files for validation.

Epoch 1/200 ②
1029/1029 - 181s - loss: 6.5628 - accuracy: 0.0111 - val_loss: 4.6525 - val_accuracy:
0.0355 - 181s/epoch - 176ms/step
Epoch 2/200
1029/1029 - 172s - loss: 4.9830 - accuracy: 0.0214 - val_loss: 4.4681 - val_accuracy:
0.0717 - 172s/epoch - 168ms/step
Epoch 3/200
1029/1029 - 170s - loss: 4.6948 - accuracy: 0.0315 - val_loss: 4.3172 - val_accuracy:
0.1273 - 170s/epoch - 166ms/step
...
Epoch 199/200
1029/1029 - 167s - loss: 0.0360 - accuracy: 0.9907 - val_loss: 0.6933 - val_accuracy:
0.8258 - 167s/epoch - 163ms/step
Epoch 200/200
1029/1029 - 168s - loss: 0.0378 - accuracy: 0.9897 - val_loss: 0.6995 - val_accuracy:
0.8273 - 168s/epoch - 163ms/step

정확률= 82.72594809532166 ③
```

8.7.3 사전 학습 모델로 견종 인식

④



⑤



8.8 [비전 에이전트 7] 견종 인식 프로그램

프로그램 8-8 견종 인식 프로그램 구현하기

```
01  import cv2 as cv
02  import numpy as np
03  import tensorflow as tf
04  import winsound
05  import pickle
06  import sys
07  from PyQt5.QtWidgets import *
08
09  cnn=tf.keras.models.load_model('cnn_for_stanford_dogs.h5')    # 모델 읽기
10  dog_species=pickle.load(open('dog_species_names.txt','rb'))   # 견종 이름
11
12  class DogSpeciesRecognition(QMainWindow):
13      def __init__(self) :
14          super().__init__()
15          self.setWindowTitle('견종 인식')
16          self.setGeometry(200,200,700,100)
17
18          fileButton=QPushButton('강아지 사진 열기',self)
19          recognitionButton=QPushButton('품종 인식',self)
20          quitButton=QPushButton('나가기',self)
21
22          fileButton.setGeometry(10,10,100,30)
23          recognitionButton.setGeometry(110,10,100,30)
24          quitButton.setGeometry(510,10,100,30)
25
26          fileButton.clicked.connect(self.pictureOpenFunction)
27          recognitionButton.clicked.connect(self.recognitionFunction)
28          quitButton.clicked.connect(self.quitFunction)
```

[프로그램 8-7]이 저장해둔 모델을 읽어옴

8.8 [비전 에이전트 7] 견종 인식 프로그램

```
29
30      def pictureOpenFunction(self):
31          fname=QFileDialog.getOpenFileName(self,'강아지 사진 읽기','./')
32          self.img=cv.imread(fname[0])
33          if self.img is None: sys.exit('파일을 찾을 수 없습니다.')
34
35          cv.imshow('Dog image',self.img)
36
37      def recognitionFunction(self):
38          x=np.reshape(cv.resize(self.img,(224,224)),(1,224,224,3))
39          res=cnn.predict(x)[0]    # 예측
40          top5=np.argsort(-res)[:5]
41          top5_dog_species_names=[dog_species[i] for i in top5]
42          for i in range(5):
43              prob='('+str(res[top5[i]])+')'
44              name=str(top5_dog_species_names[i]).split('-')[1]
45              cv.putText(self.img,prob+name,(10,100+i*30),cv.FONT_HERSHEY_
                SIMPLEX,0.7,(255,255,255),2)
46          cv.imshow('Dog image',self.img)
47          winsound.Beep(1000,500)
48
49      def quitFunction(self):
50          cv.destroyAllWindows()
51          self.close()
52
53  app=QApplication(sys.argv)
54  win=DogSpeciesRecognition()
55  win.show()
56  app.exec_()
```

8.8 [비전 에이전트 7] 견종 인식 프로그램