

4주차 요약본

작성자

이의진(엘텍공과대학 휴먼기계바이오공학부)

Chapter 08 컨볼루션 신경망

다층 퍼셉트론의 한계:

2차원 구조의 영상을 1차원으로 변환하여 입력해야함.

→ 이 과정에서 정보 손실로 인한 성능 저하 발생

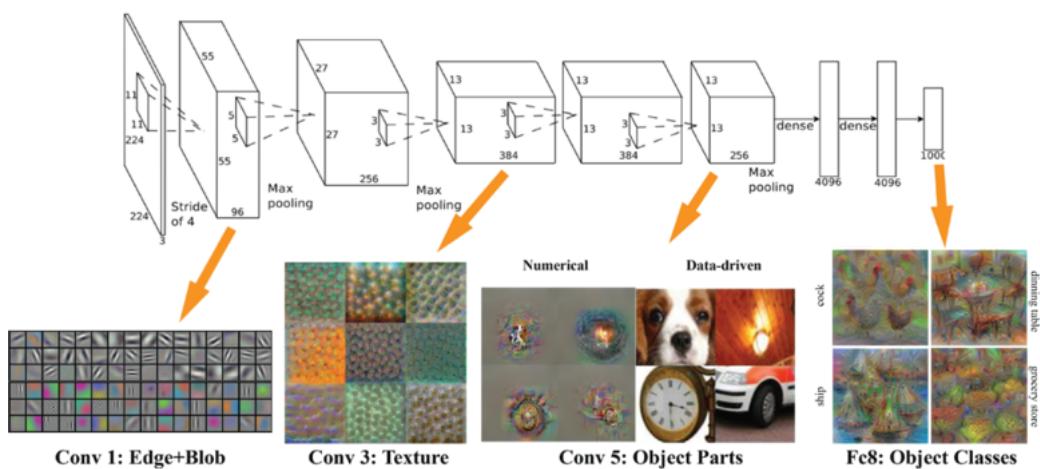
컨볼루션 신경망(CNN: Convolutional Neural Network)의 응용

컴퓨터 비전: 분류, 검출, 분할, 추적 등의 물체 인식, 음성인식과 자연어 처리 등

ex) 알파고가 19X19 바둑판의 형세 분석 시 이용

8.1 발상과 전개

AlexNet



→ 5개의 CNN, 3개의 FC layer로 구성.



2012년 ILSVRC 대회에서 AlexNet이 우승을 차지하며 CNN이 중심인 딥러닝 시대로 대전환

◆발상

- 2차원 구조의 영상을 1차원으로 변환할 때 발생하는 정보 손실

A	B																																
<table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>0</td></tr> </table>	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	<table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>1</td><td>1</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>0</td></tr> </table>	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0
0	0	0	0																														
0	1	0	0																														
1	1	0	0																														
0	0	0	0																														
0	0	0	0																														
0	0	1	1																														
0	0	0	1																														
0	0	0	0																														
(a) 2차원 구조의 영상																																	
(b) 1차원 구조로 펼친 경우																																	

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

(a) A의 삼각형이 회전, 이동한 영상 B에서도 삼각형 검출 가능

(b) 2차원을 1차원으로 펼치면 삼각형 모양이 사라짐

👉다중 퍼셉트론은 화소의 연결성을 쓸 수 없어 개별 픽셀을 보고 분류해야함 ⇒ 정확도 매우 낮음.

- MLP - 매우 많은 가중치

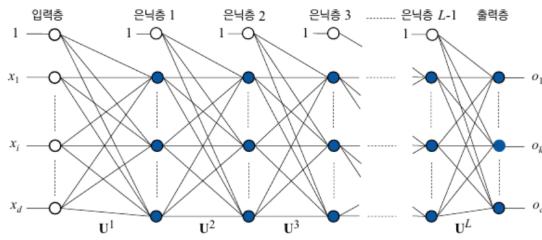


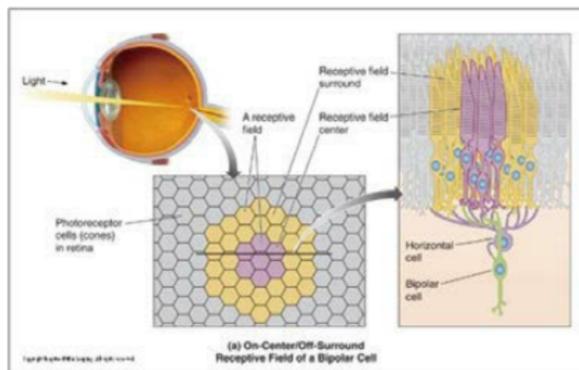
그림 7-17 깊은 다중 퍼셉트론(층 신경망)

👉FC layer 층으로 구성됨.

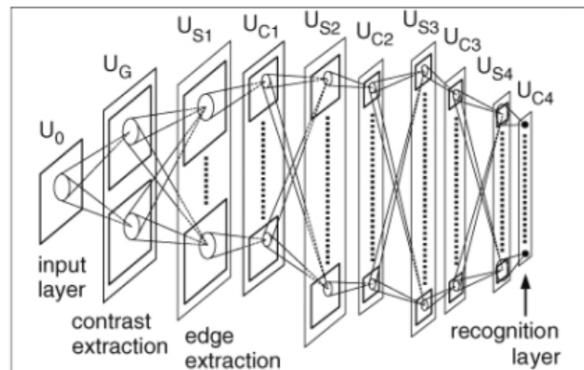
컬러영상 입력하려면 입력층에만 $256 \times 256 \times 3$ 개의 노드가 필요. 은닉층에 2048개 노드가 있다면,

그 층에만 4억 개 이상의 가중치가 있다(데이터셋을 작은 영상으로 국한한 이유)

- 인간의 수용장과 인공 신경망의 수용장



(a) 인간 시각의 수용장



(b) 네오코그니트론[Fukushima1980]

그림 8-3 인간의 수용장과 인공 신경망의 수용장

→ 수용장 하나는 영상의 특정 위치를 담당하며 망막에 분포한 수많은 수용장이 영상 전체를 관리.

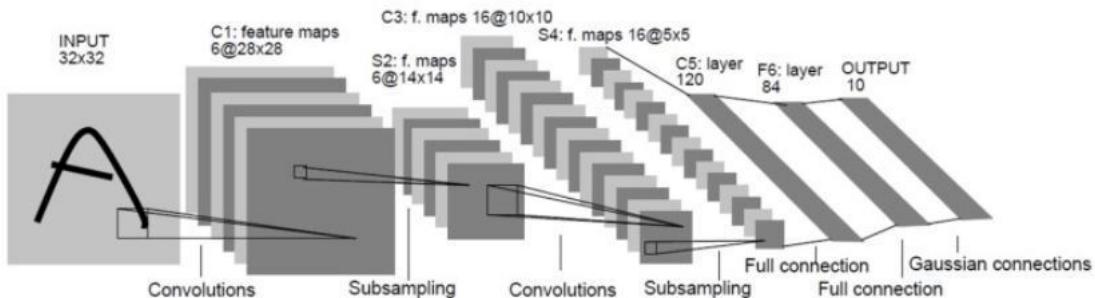
♦간략 역사

1. 네오코그니트론(1980년 후쿠시마)

- 원이 수용장, 첫 번째 층은 대비 특징, 두 번째 층은 에지 특징 추출
- 특징을 계층적으로 추출하다가 마지막 층(recognition layer)에서 분류 수행

2. 총이 5개인 **LeNet-5** 컨볼루션 신경망 제안(1998년 르쿤)

- 수표에 적힌 필기 문자를 인식하는 시스템 개발
- CNN이 실제 응용에 성공적으로 적용한 첫 사례
- LeNet-5의 구조
 - input, 3개의 Conv-layer(C1, C3, C5), 2개의 서브샘플링 레이어(S2, S4), 1층의 FC-layer(F6), output layer로 구성되어 있다. 참고로 C1부터 F6까지 활성화 함수로 tanh를 사용한다.



3. 방대한 ImageNet 데이터셋 구축(2010년 페이페이)

- 이중 1000부류를 뽑아 2010년부터 **ILSVRC**(ImageNet Large Scale Visual Recognition Challenge) 대회 개최
 - 120만장의 훈련 집합, 5만 장의 검증 집합, 15만 장의 테스트 집합을 제공.
 - ImageNet 태스크의 이미지 분류 성능은 Top-1, Top-5 오류로 측정 된다.
 - Top-1 error : 가장 높은 가능성을 보이는 레이블이 정답 레이블이 아닌 테스트 이미지의 비율
 - Top-5 error: 가장 높은 가능성을 보이는 5개의 레이블 중 정답 레이블이 없는 테스트 이미지의 비율
- ⇒ CNN기반의 분류기가 나오기 이전에는 수작업 특징과 SVM 분류기 사용. (오류율: 25.8%)

4. **AlexNet** (2012)

- 오류율: 15.3%
- 컨볼루션 신경망의 가능성을 입증

5. **VGGNet** (2014년 옥스퍼드 대학교)

- 오류율: 7.3%

6. **GoogLeNet**(2014년 구글)

- 오류율: 6.7%

7. **ResNet**(2015년 마이크로소프트)

- 오류율: 3.5%

◆CNN의 초기 발전 추세

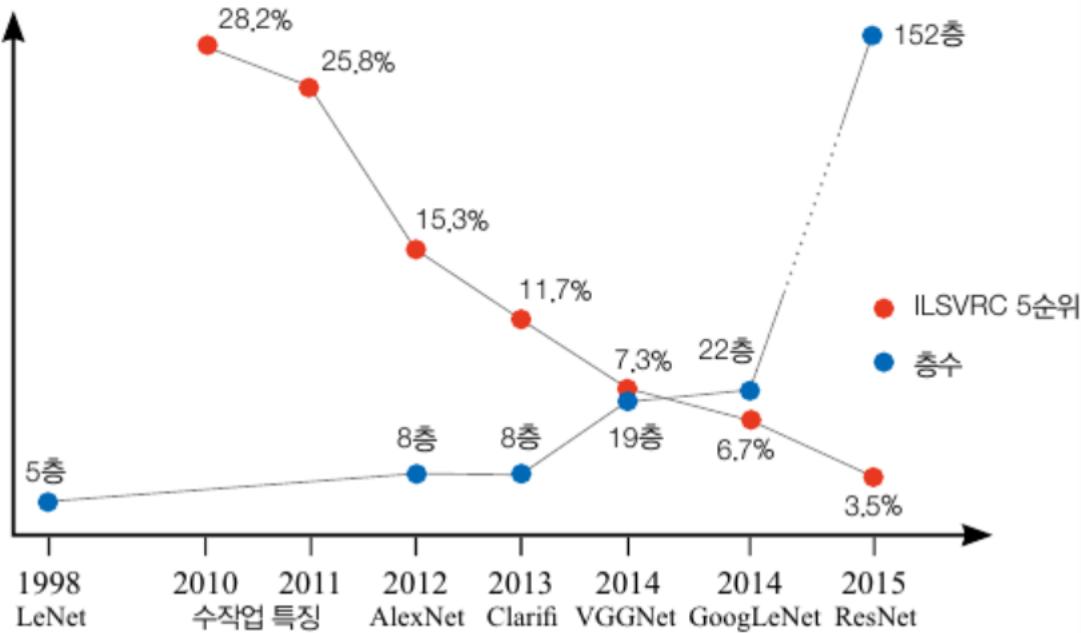


그림 8-5 컨볼루션 신경망의 초기 발전 추세[오일석2017]

→ 신경망의 구조 개선

1. CNN의 layer은 점점 깊어진다.
2. 초기는 주로 분류 문제 → 점점 검출, 분할, 추적으로 확장
3. 2014년 물체 검출을 위한 RCNN 등장 → fast RCNN, faster RCNN, mask RCNN으로 발전
4. 2014년 굿펠로우가 GAN 생성모델 발표 → InfoGAN, cycleGAN, SAGAN 등을 발전

→ 학습 알고리즘 발전

- ReLU 활성함수 발견되고 다양한 변종 개발
- 손실 함수로 교차 엔트로피와 여러 변종 개발
- optimizer로는 표준에 해당하는 SGD를 넘어 모멘텀 Adam 널리 쓰임.
- 규제 기법 개발: 드롭아웃, 배치 정규화, 데이터 증강 등

▼ 참고 자료

https://www.dbpia.co.kr/pdf/pdfView.do?nodeId=NODE07587658&googleIPsandBox=false&mark=0&ipRange=false&accessgl=Y&language=ko_KR&hasTopBanner=false

◆ 딥러닝의 성공 요인

- 데이터셋이 커짐. ImageNet은 초기 발전에 가장 큰 공헌을 함.
- GPU의 병렬 처리로 학습 시간 빨라짐.
- 좋은 학습 알고리즘 개발됨.
ex) 활성 함수, 규제 기법, 손실 함수, 옵티마이저의 발전

8.2 컨볼루션 신경망의 구조



CNN: 최적의 필터를 학습으로 알아낸다.

8.2.1 컨볼루션층과 풀링층

→ 신경망에 표준 컨볼루션 연산을 적용하려면,

1. 컬러 영상은 3차원 구조의 텐서이므로 필터를 3차원으로 확장해야 한다.
2. 풍부한 특징을 추출하도록 많은 필터를 배치해야 한다.

컨볼루션층

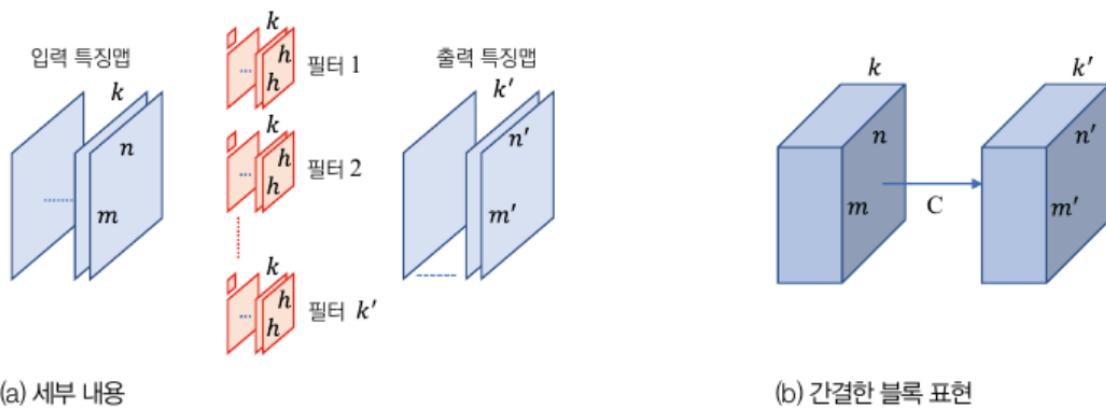


그림 8-6 컨볼루션층

그림 8-6

1. Conv-layer: 입력 특징 맵에 convolution을 적용해 얻은 특징 맵을 출력한다.
2. 입력 특징 맵: 깊이가 k (k 개 채널로 구성)인 $m \times n$ 맵 $\Rightarrow m \times n \times k$ 모양의 3차원 텐서
3. 필터: 필터의 깊이는 입력과 같이 k 고 크기는 $h \times h$, 보통은 3 or 5 사용. 필터는 하나의 바이어스 값을 추가로 가진다.
 - 필터는 $kh^2 + 1$ 개의 가중치를 가짐
 - 필터는 상하좌우방향으로 이동하며 convolution을 수행함 \rightarrow 로짓(logit)을 구하고 활성 함수를 적용해 출력 특징 맵에 쓴다.
 - 보통 활성함수로는 ReLU나 그 변종을 사용한다.
4. 컨볼루션층은 필터를 여러 개 사용하여 풍부한 특징을 추출한다.
5. 필터 개수가 k' 이면, 필터 하나가 특징 맵 하나를 생성하므로 출력 특징 맵은 필터의 개수에 해당하는 k' 깊이의 맵을 가짐.

→ Padding

맵의 경계에서 필터를 대면 일부 픽셀이 밖으로 나간다.

경계를 제외하면 층이 깊어질수록 맵이 점점 작아짐.

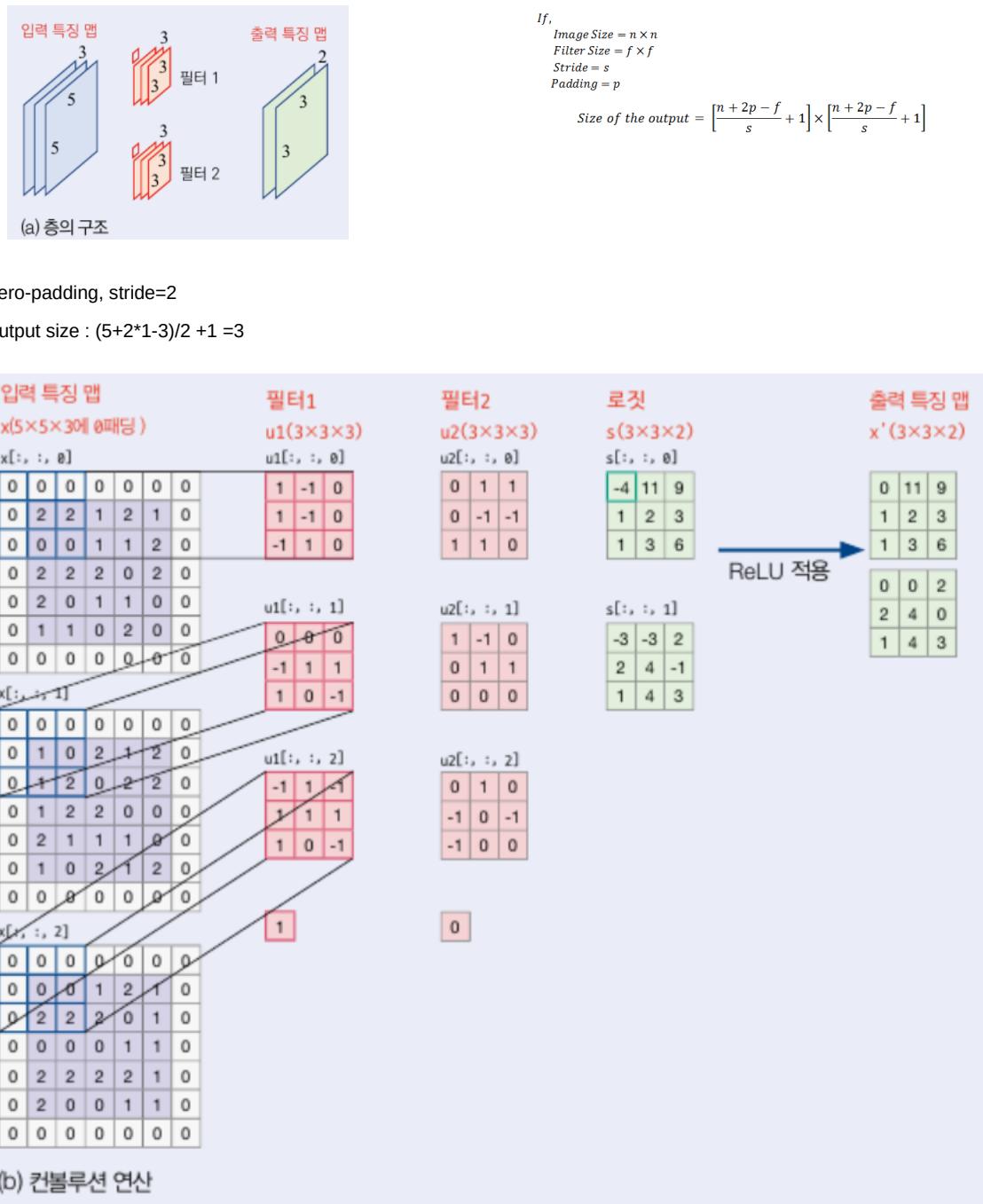
⇒ 0 덧대기, 복사 덧대기로 해결

→ Stride

보폭을 s 로 설정하면 s 픽셀씩 건너 필터를 적용한다. 출력 맵의 크기를 줄이는 효과 있음.

\Rightarrow 보폭이 s 면, $m \times n$ 맵이 $(m/s) \times (n/s)$ 로 줄어드다.

◆ [예시 8-1] 컨볼루션 층의 연산 사례



$$\frac{0 \times 1 + 0 \times (-1) + 0 \times 0 + 0 \times 1 + 2 \times (-1) + 2 \times 0 + 0 \times (-1) + 0 \times 1 + 0 \times 0 +}{\text{채널 0}} \\ \frac{0 \times 0 + 0 \times 0 + 0 \times 0 + 0 \times 0 + 0 \times (-1) + 1 \times 1 + 0 \times 1 + 0 \times 1 + 1 \times 0 + 2 \times (-1) +}{\text{채널 1}} \\ \frac{0 \times (-1) + 0 \times 1 + 0 \times (-1) + 0 \times 1 + 0 \times 1 + 0 \times 1 + 0 \times 1 + 2 \times 0 + 2 \times (-1) + 1}{\text{채널 2}} = -4$$

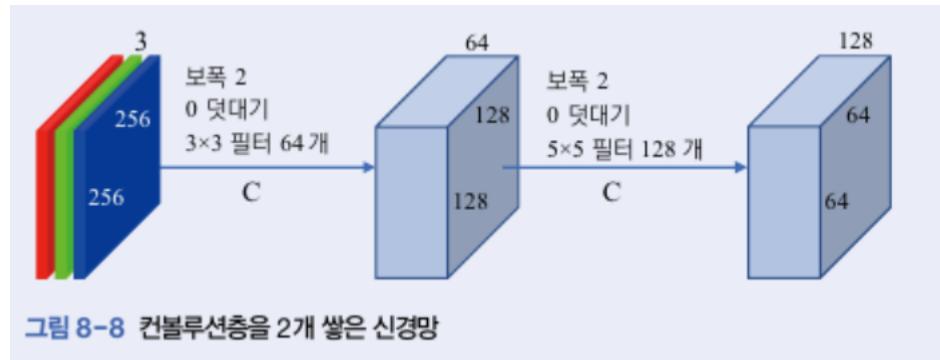
바이어스

필터의 개수 3개 → 출력 특징 맵에서 깊이가 3

◆ 컨볼루션층의 특징

- 가중치 공유(weight sharing) : 입력 특징 맵의 모든 픽셀이 같은 필터를 사용한다.
- 부분 연결성(partial connection): 필터는 해당 픽셀 주위로 국한하여 연산한다.

◆ [예시 8-2] 컨볼루션층의 연산량



1. 입력 특징 맵: $256 \times 256 \times 3$ (RGB 컬러영상 → 깊이 3)
2. 첫 번째 Conv-layer: zero-padding, stride=2
3. 출력 특징 맵: $128 \times 128 \times 64$

→ 첫 번째 conv-layer의 연산량 산정

픽셀마다 $3 \times 3 \times 3$ kernel 사용 → 픽셀당 $27+1=28$ 번의 곱셈
픽셀이 256×256 이고 stride=2 → $128 \times 128 \times 28$ 번의 곱셈
필터가 64개 → 총 $128 \times 128 \times 28 \times 64$

풀링층

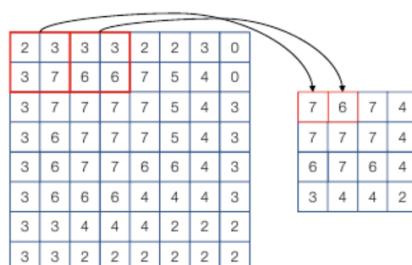


그림 8-9 풀링층(2×2 필터로 최대 풀링 적용, 보폭=2)

→ 보통 컨볼루션층 뒤에 풀링층이 따른다.

stride = 2 → 특징 맵이 반으로 줄어듦.

- **max-pooling**: 필터 안의 픽셀 중 최댓값을 취하는 연산. (주로 사용)
- **average-pooling**: 필터 안의 픽셀값의 평균을 취하는 연산.

→ pooling의 효과

특징 맵에는 인식에 불필요한 상세한 내용이 많음 → 풀링은 상세함을 줄이고 특징 맵의 크기를 줄여 신경망의 효율을 높임

8.2.2 빌딩블록을 쌓아 만드는 컨볼루션 신경망

다층 퍼셉트론: FC-layer를 구성요소로 사용

CNN: Conv-layer, Pooling을 구성요소로 사용

◆ 빌딩블록 쌓기

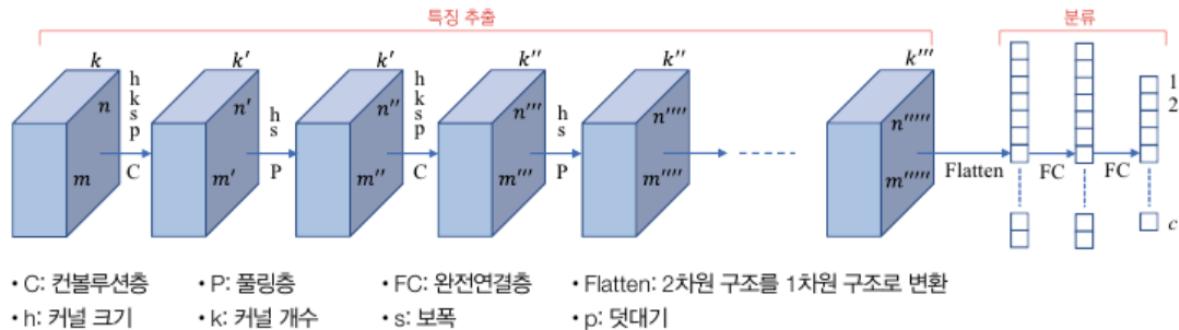
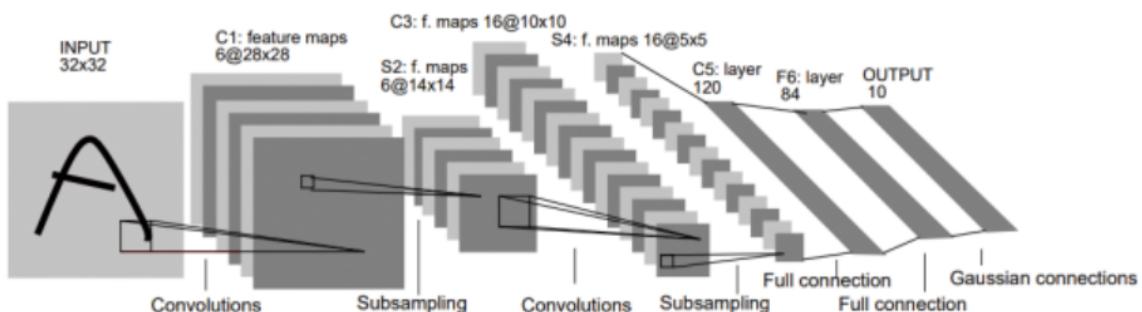


그림 8-10 컨볼루션층과 풀링층을 번갈아 쌓아 만드는 컨볼루션 신경망의 전형적인 구조

- 보통 CNN은 conv-layer, pooling-layer를 번갈아 쌓아 만든다.
 - ➡ 특징 맵의 크기 $m \times n$ 은 conv-layer, pooling-layer의 stride에 따라 바뀐다.
 - ➡ 깊이 k 는 conv-layer에서는 필터 개수와 같고, 풀링층에서는 그대로 유지된다.
- 신경망의 앞 부분
 - conv-layer, pooling-layer → 특징을 추출
- 신경망 뒷 부분
 - FC-layer를 쌓아 → 분류를 수행
- 앞부분과 뒷부분 사이
 - Flatten연산(컨볼루션층의 다차원 구조를 1차원 구조로 변환) → FC-layer에 입력
- 출력층의 부류 개수에 해당하는 만큼 노드를 배치한다.

◆ LeNet-5

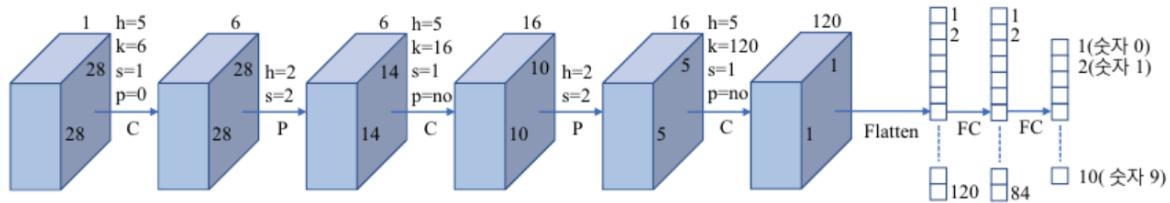


(a) [LeCun1998]의 그림

Convolutions : 컨볼루션층, Sub-sampling: 풀링층,

Full connection: 완전연결층, 출력층에 10개의 노드(필기 숫자 인식)

숫자 맵은 명암이므로 입력 특징 맵의 깊이는 1



- 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

C-P-C-P-C-FC-FC 구조

첫 번째 완전 연결층의 특징 벡터는 84개 노드를 가진 은닉층을 통하여 10개 노드를 가진 출력층에 도달

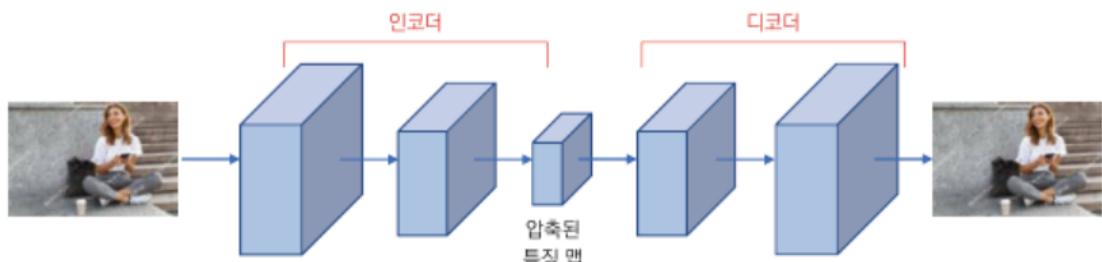
→ 가중치 계산

- 첫번째 컨볼루션 층: $(5 \times 5 \times 1 + 1) \times 6$
- 첫 번째 완전 연결층: $(120 + 1) \times 84$

◆ 컨볼루션 신경망의 유연한 구조

CNN은 Conv-layer, Pooling-layer, FC-layer를 쌓아 만들 → 데이터나 풀어야 할 문제에 따라 다양한 모양으로 조립 가능.

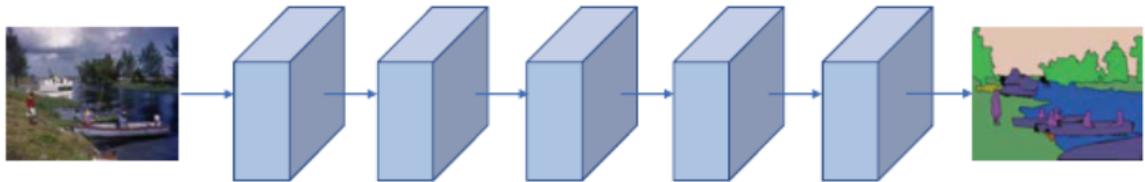
→ 오토인코더(auto-encoder)



(a) 오토인코더

- 입력 영상을 그대로 출력으로 내놓는다.
 - 인코더: 특징 맵을 점점 작게 함
 - 디코더: 다시 키워 원래 영상을 복원
- 인코더, 디코더는 대칭을 이루도록 설계한다.
- 압축된 특징 맵
 - 원래 영상보다 크기가 훨씬 축소된 특징 맵. 디코더를 통한 영상 복원이 가능할 정도의 핵심 정보를 모두 가지고 있음
 - 영상의 특징 추출이나 영상 압축기로 활용.

→ 영상 분할을 위한 컨볼루션 신경망



(b) 영상 분할을 위한 컨볼루션 신경망

8.3 컨볼루션 신경망의 학습

◆ 컨볼루션 신경망을 위한 역전파 알고리즘

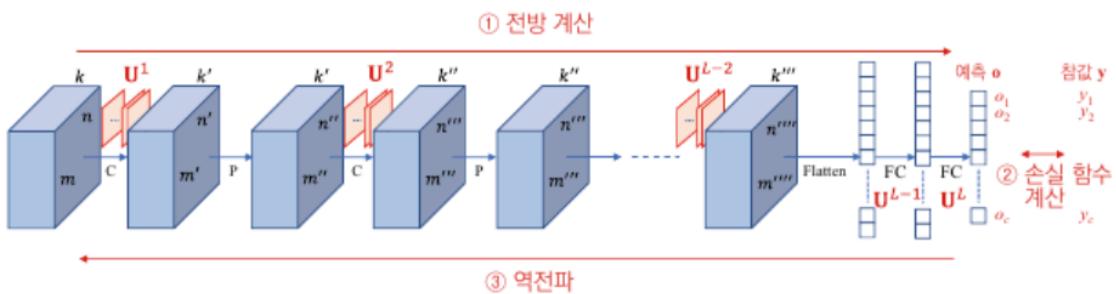


그림 8-13 컨볼루션 신경망을 학습하기 위한 역전파 알고리즘

① 전방 계산 : C, P, FC를 거쳐 출력벡터 o 출력

② 손실 함수 계산: 손실 함수를 통해 o 와 참값 벡터 y 의 오류 계산

③ 역전파: 오류를 줄이는 방향으로 가중치 갱신

가중치: C와 FC에 있는 필터 $U^1 \sim U^L$, 풀링층은 가중치 없다.

→ 다층 퍼셉트론을 위한 역전파 알고리즘이 그대로 적용됨.

◆ 특징 학습과 통째 학습

- CNN은 특징학습이다. 즉, 특징 추출을 담당하는 필터를 학습한다.

➡ 학습 알고리즘은 주어진 데이터셋을 인식하는 데 최적인 필터를 찾아낸다.

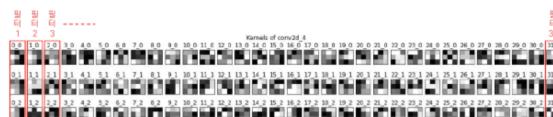


그림 8-14 CIFAR-10 데이터셋으로 학습한 컨볼루션 신경망의 최적 필터

- CNN은 통째학습



그림 8-15 딥러닝에 의한 컴퓨터 비전 방법론의 대전환

- (a) 수작업 특징(SIFT, LBP등)으로 분류 모델을 학습 → 특징 추출과 분류기를 결합해 인식기를 완성
- (b) 딥러닝 패러다임: 특징 학습과 분류기 학습을 통째로 진행

◆ 컨볼루션 신경망이 우수한 이유

1. 데이터의 원래 구조 유지

MLP : 모델에 맞춰 1차원 구조로 변환하여 입력
CNN: 3차원 구조를 그대로 입력

2. 특징 학습을 통해 최적의 특징을 추출

통째 학습으로 특징과 분류를 동시에 최적화한다.

3. 신경망의 깊이를 깊게 할 수 있다

CNN은 가중치 공유와 부분 연결성으로 가중치가 적기 때문에 무리 없이 학습 가능
층을 깊게 하면 세밀한 계층 구조의 특징을 추출하므로 인식 성능 높아짐.

8.4 컨볼루션 신경망 구현

8.4.1 LeNet-5의 재현

◆ 프로그램 8-1 : LeNet-5로 MNIST인식하기

```

import numpy as np
import tensorflow as tf
import tensorflow.keras.datasets as ds

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dropout, Dense
from tensorflow.keras.optimizers import Adam

# 1. 데이터 준비
(x_train, y_train), (x_test, y_test)=ds.mnist.load_data()
x_train=x_train.reshape(60000, 28, 28, 1)
x_test=x_test.reshape(10000, 28, 28, 1)
x_train=x_train.astype(np.float32)/255.0
x_test=x_test.astype(np.float32)/255.0
y_train=tf.keras.utils.to_categorical(y_train, 10)
y_test=tf.keras.utils.to_categorical(y_test, 10)

# 2. 모델 선택(신경망 구조 설계)
cnn=Sequential()
cnn.add(Conv2D(6, (5,5), padding='same', activation='relu', input_shape=(28,28,1)))
cnn.add(MaxPooling2D(pool_size=(2,2), strides=2))
cnn.add(Conv2D(16, (5,5), padding='valid', activation='relu'))
cnn.add(MaxPooling2D(pool_size=(2,2), strides=2))
cnn.add(Conv2D(120, (5,5), padding='valid', activation='relu'))
cnn.add(Flatten())
cnn.add(Dense(units=84, activation='relu'))
cnn.add(Dense(units=10, activation='softmax'))

# 3. 학습
cnn.compile(loss='categorical_crossentropy', optimizer=Adam(learning_rate=0.001), metrics=['accuracy'])
cnn.fit(x_train, y_train, batch_size=128, epochs=30, validation_data=(x_test, y_test), verbose=2)

# 4. 예측(성능 측정)
res=cnn.evaluate(x_test, y_test, verbose=0)
print('정확률=', res[1]*100)

```

▼ 데이터 준비

```

# MNIST 데이터셋을 읽고, 신경망에 입력할 형태로 변환
(x_train, y_train), (x_test, y_test)=ds.mnist.load_data()

```

```

x_train=x_train.reshape(60000,28,28,1)
x_test=x_test.reshape(10000,28,28,1)
x_train=x_train.astype(np.float32)/255.0
x_test=x_test.astype(np.float32)/255.0
y_train=tf.keras.utils.to_categorical(y_train,10)
y_test=tf.keras.utils.to_categorical(y_test,10)

```

x_train.shape = (60000,28,28) → 입력 특징 맵이 28x28의 2차원 구조 → 3차원 변환 필요

▼ 모델 선택(LeNet-5 신경망 구조 설계)

```

#cnn객체 생성
cnn=Sequential()
#add함수와 Conv2D,MaxPooling2D이용해 cnn객체에 컨볼루션층 추가
cnn.add(Conv2D(6,(5,5),padding='same',activation='relu',input_shape=(28,28,1)))
cnn.add(MaxPooling2D(pool_size=(2,2),strides=2))
cnn.add(Conv2D(16,(5,5),padding='valid',activation='relu'))
cnn.add(MaxPooling2D(pool_size=(2,2),strides=2))
cnn.add(Conv2D(120,(5,5),padding='valid',activation='relu'))
cnn.add(Flatten()) #텐서를 1차원 구조로 변환해 FC층에 입력
cnn.add(Dense(units=84,activation='relu')) #FC층 추가.
cnn.add(Dense(units=10,activation='softmax'))#FC층 추가. 출력노드 10개 활성함수로 softmax사용

```

`Conv2D(6,(5,5),padding='same',activation='relu',input_shape=(28,28,1))`

- 1,2번째 인수: 5x5 필터를 6개 사용하라
- 3번째 인수: zero-padding 적용하라
 - padding='valid'는 패딩을 적용X
- 4번째 인수: ReLU 활성 함수 사용 지시
- 5번째 인수: 신경망에 최초로 입력되는 텐서 모양 지정
- 두 번째 컨볼루션 층부터는 input_shape인수 생략 가능

`MaxPooling2D(pool_size=(2,2),strides=2)`

- (2,2) 필터를 사용하며 보폭은 2

▼ 학습

```

# 성능 조사 지시
cnn.compile(loss='categorical_crossentropy',optimizer=Adam(learning_rate=0.001),metrics=[ 'accuracy'])
# 실제 학습 진행
cnn.fit(x_train,y_train,batch_size=128,epochs=30,validation_data=(x_test,y_test),verbose=2)

```

`cnn.fit(x_train,y_train,batch_size=128,epochs=30,validation_data=(x_test,y_test),verbose=2)`

- 미니 배치 크기를 128로 하고 최대 30세대 반복한다.
- 매 세대마다 validation에 설정한 x_test, y_test를 가지고 성능 평가하여 화면에 표시

▼ 예측(성능 측정)

```

# x_test,y_test를 테스트 집합으로 성능을 측정하여 res에 저장한다.
res=cnn.evaluate(x_test,y_test,verbose=0)
print('정확률=',res[1]*100)

```

실행 결과: 앞의 깊은 다층 퍼셉트론보다 성능이 향상되었음.

8.4.2 자연 영상 인식

◆ 프로그램 8-2 : CNN으로 자연 영상 인식하기

```

import numpy as np
import tensorflow as tf
import tensorflow.keras.datasets as ds

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D,MaxPooling2D,Flatten,Dense,Dropout
from tensorflow.keras.optimizers import Adam

```

```

(x_train,y_train),(x_test,y_test)=ds.cifar10.load_data()
# CIFAR-10 데이터셋은 이미 32x32x3 텐서로 표현되어있어 모양 변환할 필요 없다.
x_train=x_train.astype(np.float32)/255.0
x_test=x_test.astype(np.float32)/255.0
y_train=tf.keras.utils.to_categorical(y_train,10)
y_test=tf.keras.utils.to_categorical(y_test,10)

# C-C-P-D-C-C-P-D-FC-D-FC
cnn=Sequential()
#C-C-P-D 구조 추가
cnn.add(Conv2D(32,(3,3),activation='relu',input_shape=(32,32,3)))
cnn.add(Conv2D(32,(3,3),activation='relu'))
cnn.add(MaxPooling2D(pool_size=(2,2)))
cnn.add(Dropout(0.25))
#C-C-P-D 구조 추가
cnn.add(Conv2D(64,(3,3),activation='relu'))
cnn.add(Conv2D(64,(3,3),activation='relu'))
cnn.add(MaxPooling2D(pool_size=(2,2)))
cnn.add(Dropout(0.25))
# Flatten으로 특징 맵을 펼침. FC-D-FC구조 추가
cnn.add(Flatten())
cnn.add(Dense(units=512,activation='relu'))
cnn.add(Dropout(0.5))
cnn.add(Dense(units=10,activation='softmax'))

#신경망 학습
cnn.compile(loss='categorical_crossentropy',optimizer=Adam(learning_rate=0.001),metrics=['accuracy'])
hist=cnn.fit(x_train,y_train,batch_size=128,epochs=100,validation_data=(x_test,y_test),verbose=2)

#성능 측정
res=cnn.evaluate(x_test,y_test,verbose=0)
print('정확률=',res[1]*100)

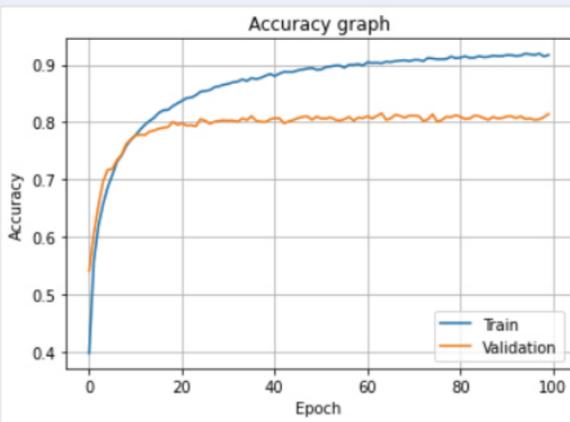
import matplotlib.pyplot as plt

plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('Accuracy graph')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train','Validation'])
plt.grid()
plt.show()

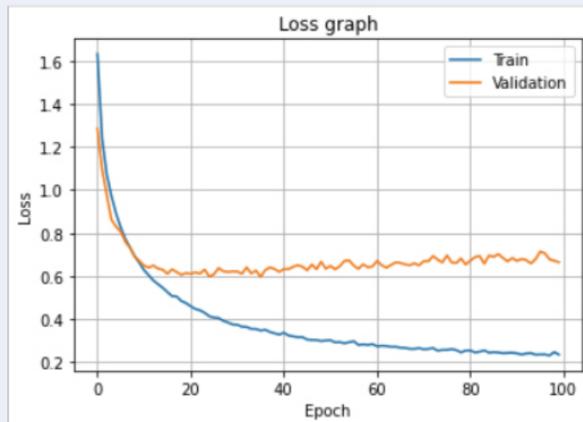
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('Loss graph')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train','Validation'])
plt.grid()
plt.show()

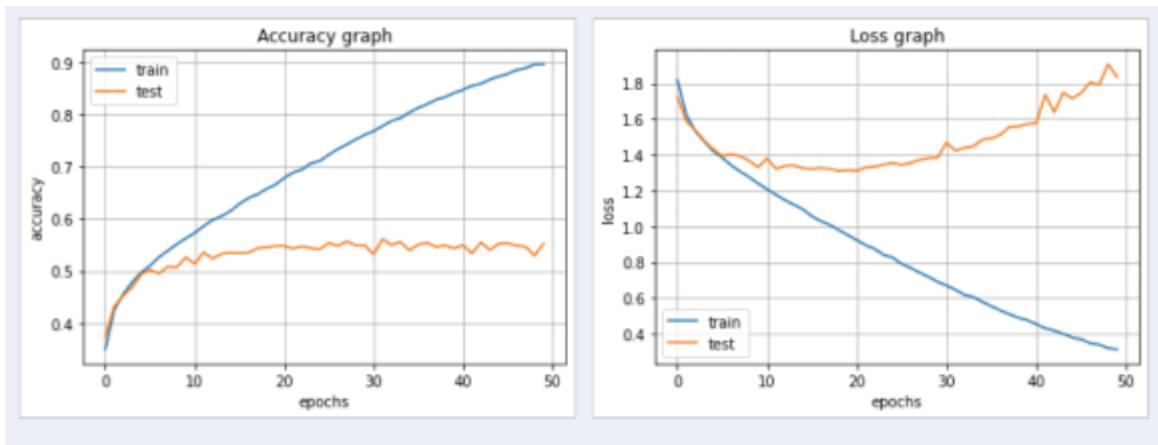
```

③



④





→ MLP보다 과잉 적합이 상당이 누그러졌다.

8.4.3 텐서플로 프로그래밍

◆ models 모듈 : 모델 생성

- Sequential : 한 갈래 텐서가 끝까지 흐름

```
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'))
```

층을 흐르는 텐서 중간에 접근X

- Functional API: 중간에 여러갈래로 나뉠 수 있음.

```
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)
```

중간 텐서가 변수로 노출되므로 쉽게 접근 가능

◆ layers 모듈 : 신경망 구성하는 다양한 종류의 층 제공

Dense, Conv2D, MaxPooling2D, Flatten, Dropout 등

→ Conv3D는 필터가 세 방향(수평, 수직, 깊이)으로 이동한다. 이때, 필터의 깊이가 입력 텐서의 깊이보다 작아야한다.

◆ losses 모듈 : MSE, categorical_crossentropy 등

◆ optimizer 모듈 : SGD, Adam, AdaGrad, RMSprop 등

SGD는 가장 기본적인 옵티마이저, 다른 애들은 SGD의 변종

Adam이 가장 좋은 성능을 보인다는 보고가 여럿 있다고 함.

8.5 [비전 에이전트6] 우편번호 인식기 v.2

◆ 좋은 하이퍼 매개변수 설정으로 정확률 개선

```
import numpy as np
import tensorflow as tf
import tensorflow.keras.datasets as ds
```

```

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dropout, Dense
from tensorflow.keras.optimizers import Adam

(x_train,y_train),(x_test,y_test)=ds.mnist.load_data()
x_train=x_train.reshape(60000,28,28,1)
x_test=x_test.reshape(10000,28,28,1)
x_train=x_train.astype(np.float32)/255.0
x_test=x_test.astype(np.float32)/255.0
y_train=tf.keras.utils.to_categorical(y_train,10)
y_test=tf.keras.utils.to_categorical(y_test,10)

cnn=Sequential()
cnn.add(Conv2D(32,(3,3),activation='relu',input_shape=(28,28,1)))
cnn.add(Conv2D(32,(3,3),activation='relu'))
cnn.add(MaxPooling2D(pool_size=(2,2)))
cnn.add(Dropout(0.25))
cnn.add(Conv2D(64,(3,3),activation='relu'))
cnn.add(Conv2D(64,(3,3),activation='relu'))
cnn.add(MaxPooling2D(pool_size=(2,2)))
cnn.add(Dropout(0.25))
cnn.add(Flatten())
cnn.add(Dense(units=512,activation='relu'))
cnn.add(Dropout(0.5))
cnn.add(Dense(units=10,activation='softmax'))

cnn.compile(loss='categorical_crossentropy',optimizer=Adam(learning_rate=0.001),metrics=['accuracy'])
hist=cnn.fit(x_train,y_train,batch_size=128,epochs=100,validation_data=(x_test,y_test),verbose=2)

cnn.save('cnn_v2.h5')
# 학습한 모델을 저장해 비전 에이전트 5를 업그레이드하는데
# 쓸 수 있게 준비한다.

res=cnn.evaluate(x_test,y_test,verbose=0)
print('정확률=',res[1]*100)

```

◆ 비전 에이전트 버전업

```

import numpy as np
import tensorflow as tf
import cv2 as cv
import matplotlib.pyplot as plt
import winsound

#7-7과 비슷
model=tf.keras.models.load_model('cnn_v2.h5')

def reset():
    global img

    img=np.ones((200,520,3),dtype=np.uint8)*255
    for i in range(5):
        cv.rectangle(img,(10+i*100,50),(10+(i+1)*100,150),(0,0,255))
        cv.putText(img,'e:erase s:show r:recognition q:quit',(10,40),cv.FONT_HERSHEY_SIMPLEX,0.8,(255,0,0),1)

def grab_numerals():
    numerals=[]
    for i in range(5):
        roi=img[51:149,11+i*100:9+(i+1)*100,0]
        roi=255-cv.resize(roi,(28,28),interpolation=cv.INTER_CUBIC)
        numerals.append(roi)
    numerals=np.array(numerals)
    return numerals

def show():
    numerals=grab_numerals()
    plt.figure(figsize=(25,5))
    for i in range(5):
        plt.subplot(1,5,i+1)
        plt.imshow(numerals[i],cmap='gray')
        plt.xticks([]); plt.yticks([])
    plt.show()

def recognition():
    numerals=grab_numerals()
    numerals=numerals.reshape(5,28,28,1)
    # 샘플의 텐서를 CNN의 입력층에 맞추는 일
    numerals=numerals.astype(np.float32)/255.0
    res=model.predict(numerals) # 신경망 모델로 예측
    class_id=np.argmax(res,axis=1)
    for i in range(5):
        cv.putText(img,str(class_id[i]),(50+i*100,180),cv.FONT_HERSHEY_SIMPLEX,1,(255,0,0),1)

```

```

winsound.Beep(1000,500)

BrushSiz=4
LColor=(0,0,0)

def writing(event,x,y,flags,param):
    if event==cv.EVENT_LBUTTONDOWN:
        cv.circle(img,(x,y),BrushSiz,LColor,-1)
    elif event==cv.EVENT_MOUSEMOVE and flags==cv.EVENT_FLAG_LBUTTON:
        cv.circle(img,(x,y),BrushSiz,LColor,-1)

reset()
cv.namedWindow('Writing')
cv.setMouseCallback('Writing',writing)

while(True):
    cv.imshow('Writing',img)
    key=cv.waitKey(1)
    if key==ord('e'):
        reset()
    elif key==ord('s'):
        show()
    elif key==ord('r'):
        recognition()
    elif key==ord('q'):
        break

cv.destroyAllWindows()

```

데이터셋 시프트: 학습에 사용한 데이터와 실제 현장에서 발생한 데이터의 분포가 다른 상황

8.6 딥러닝의 학습 알고리즘 향상

8.6.1 손실 함수

◆ **교차 엔트로피**: 두 확률 분포가 얼마나 다른지 측정

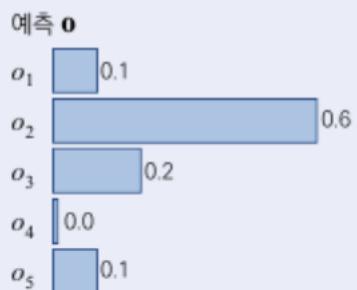
신경망의 경우, 신경망이 예측한 벡터와 참값 벡터의 다른 정도를 측정한다.

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

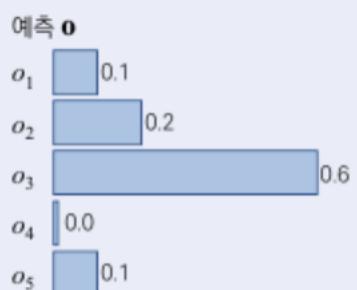
c는 부류 개수, 위 식은 한 샘플에 대한 교차 엔트로피이므로, 손실 함수로 쓸 때는 미니 배치에 대해 평균한다.

[예시 8-3] 평균제곱오차와 교차 엔트로피 손실 함수

[그림 8-17]은 $c=5$ 인 경우에 대해 예측 벡터 \mathbf{o} 와 참값 벡터 \mathbf{y} 를 예시한다. [그림 8-17(a)]는 \mathbf{o} 가 \mathbf{y} 에 근접해서 신경망이 맞힌 상황이고 [그림 8-17(b)]는 \mathbf{o} 가 \mathbf{y} 에서 벗어나 신경망이 틀린 상황이다. 미니 배치가 샘플 하나만 가진다고 가정하고 손실 함수를 계산해본다.



(a) 예측이 참값에 근접한 상황(①)



(b) 예측이 참값에서 벗어난 상황(②)

그림 8-17 손실 함수 계산

평균제곱오차

$$\left\{ \begin{array}{l} \text{상황 ①: } \text{MSE} = (0.0 - 0.1)^2 + (1.0 - 0.6)^2 + (0.0 - 0.2)^2 + (0.0 - 0.0)^2 + (0.0 - 0.1)^2 = 0.22 \\ \text{상황 ②: } \text{MSE} = (0.0 - 0.1)^2 + (1.0 - 0.2)^2 + (0.0 - 0.6)^2 + (0.0 - 0.0)^2 + (0.0 - 0.1)^2 = 1.02 \end{array} \right.$$

교차 엔트로피

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

두 손실 함수 모두 잘 예측한 상황에서 값이 더 낮다.

→ 단계2에서 오류 계산 시 손실 함수 사용 → 이때 값이 클수록 가중치를 갱신하는 양이 커짐

단, MSE는 손실 함수값이 더 큰데 미분값이 더 작아 가중치 갱신이 더 작은 상황이 발생하기도 함 → 교차 엔트로피로 해결

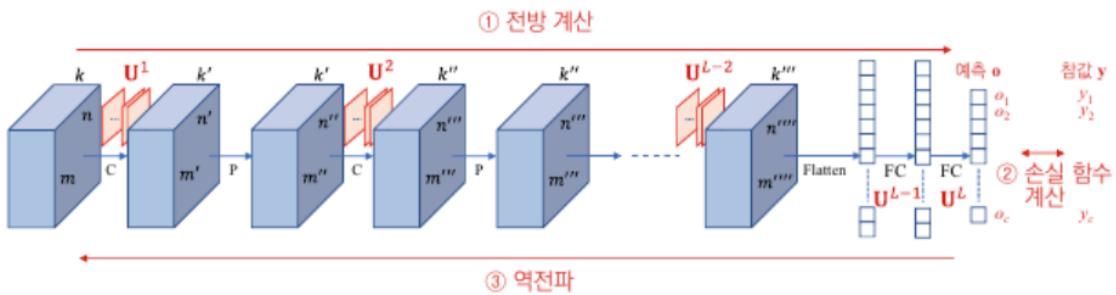
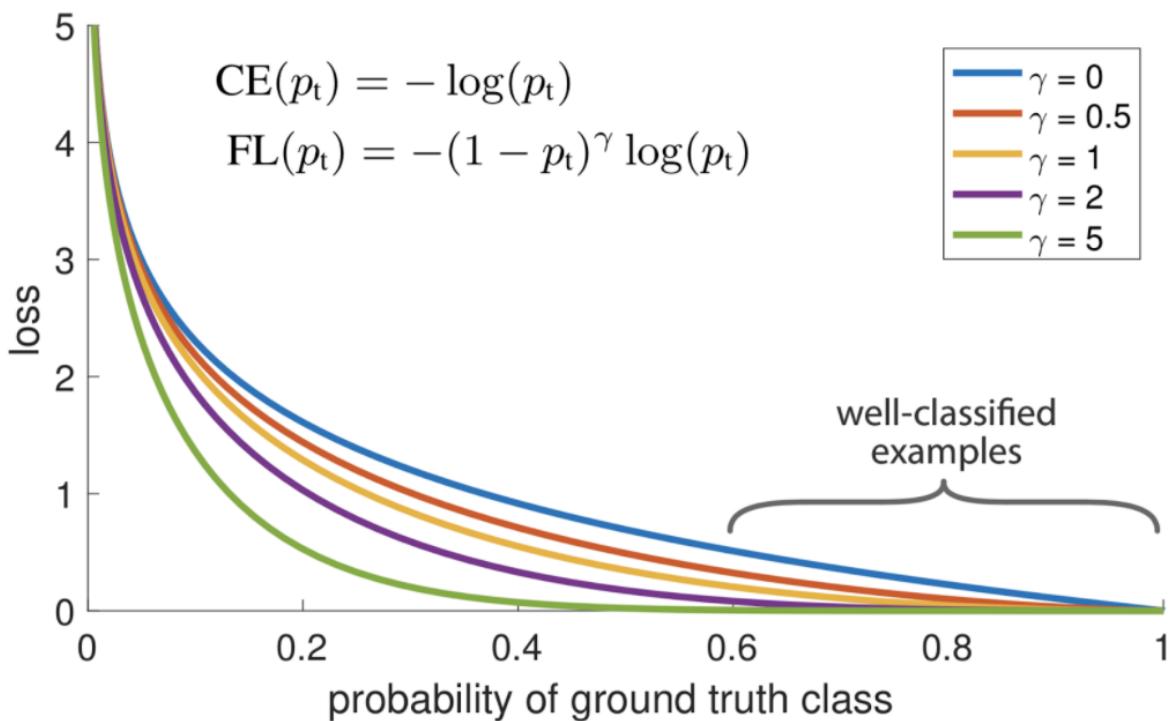


그림 8-13 컨볼루션 신경망을 학습하기 위한 역전파 알고리즘

◆ Focal 손실 함수: 클래스 불균형 문제를 해결.

EX) 물체와 배경을 구분하는 물체 분할 작업



Cross-entropy

- 만약 p 의 값이 1이면 $\text{CE}(p,y)=0$
잘 예측하였지만 보상은 없으며 단지, 패널티가 없음
- 반면 p 의 값을 0에 가깝게 예측하게 되면 $\text{CE}(p,y)\approx\infty$
페널티가 굉장히 커지게 됩니다.
- 모든 픽셀의 엔트로피를 평균하므로,
배경(분류 쉬움)만 잘 분류하면 물체(분류 어려움)를 구별 못해도 교차 엔트로피가 낮아짐.

Focal loss

- 물체(분류 어려움)를 잘 분류 못했을 때 → 더 큰 가중치 (객체 일부분만 있거나, 실제 분류해야 되는 객체들)
배경(분류 쉬움)을 잘 분류 못했을 때 → 낮은 가중치(background object가 이에 해당)

- p_t 가 커지면 빠르게 0에 가깝게 만들. $\gamma=2$ 에서 최고성능

8.6.2 옵티마이저

신경망 학습: 손실 함수의 최저점, 오류가 최저인 점을 찾아가는 과정. 이 최적화 문제의 표준 알고리즘이 SGD
SGD의 한계는 모멘텀, 적응적 학습률로 극복.

◆ 모멘텀

기본개념 : 이전 운동량을 현재에 반영한다.

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

모멘텀은: 속도(운동량)를 먼저 생성하고, 속도로 가중치를 생성
알파는 오래된 운동량의 영향력을 줄여줌.

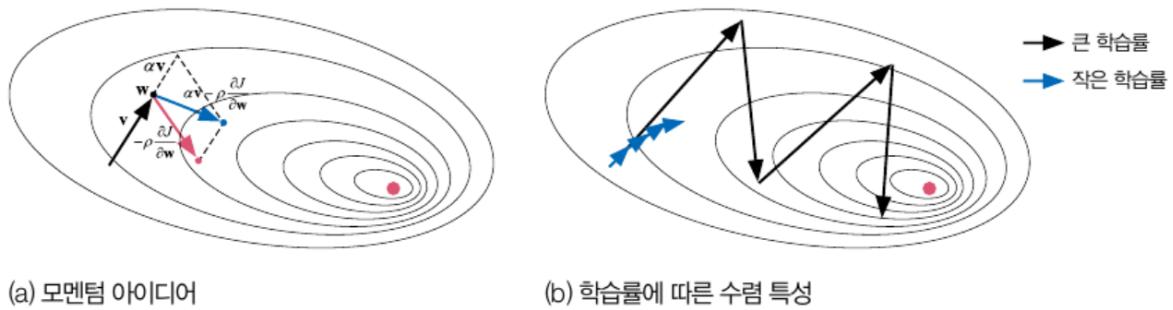
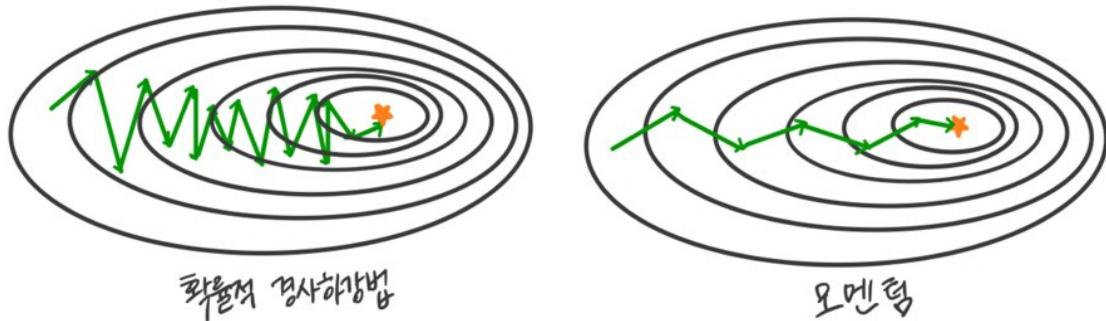


그림 8-19 딥러닝에서 사용하는 옵티마이저

- SGD : 빨간색 점이 새로운 가중치
- SGD with 모멘텀: 파란색 점이 새로운 가중치



- 텐서플로의 SGD클래스 선언에서 momentum매개변수 값을 지정하면 됨. 0이면 SGD

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

◆ 적응적 학습률

세대와 그레디언트에 따라 적응적으로 학습률을 정함. 앞에선 고정된 학습률을 사용하였음.

종류: AdaGrad, RMSprop, Adam

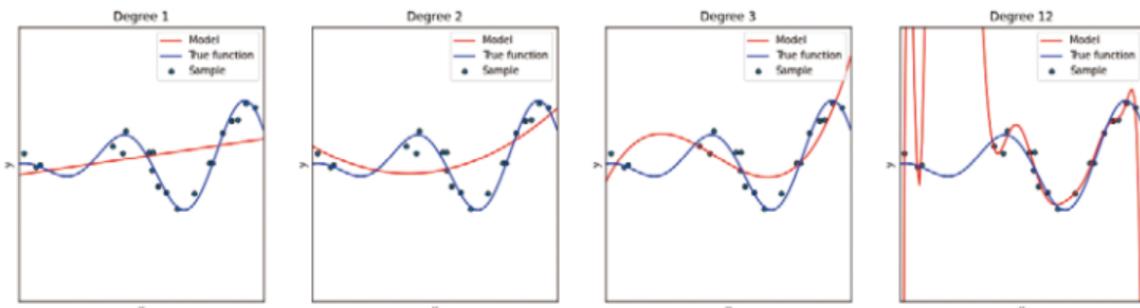
```
tf.keras.optimizers.Adagrad(learning_rate=0.001, initial_accumulator_value=0.1,  
                           epsilon=1e-07, name="Adagrad", **kwargs)  
tf.keras.optimizers.RMSprop(learning_rate=0.001, rho=0.9, momentum=0.0,  
                           epsilon=1e-07, centered=False, name="RMSprop",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

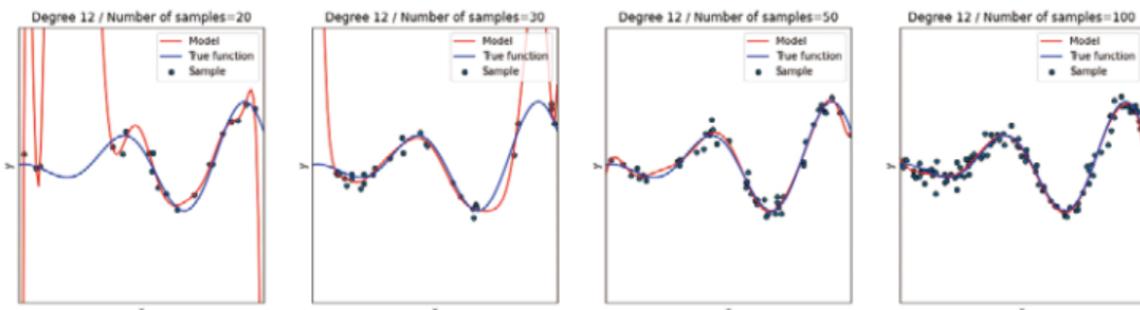
딥러닝은 깊은 신경망을 사용하므로 가중치가 많다

데이터 부족 → 과잉 적합(overfitting)

과잉 적합



(a) 과소 적합과 과잉 적합 현상



(b) 데이터 증강을 통한 과잉 적합 방지

그림 8-20 과잉 적합 현상과 데이터 증강을 통한 방지

(a) 1차원 모델: 데이터에 비해 용량이 작아 모델링이 제대로 되지 않음 → 과소 적합

12차원 모델: 훈련 집합에 대해서는 완벽하게 모델링. 일반화능력이 낮음 → 과대 적합

(b) 데이터 증강: 훈련 집합 크기를 늘리면, 과잉적합이 누그러지는 모습

⇒ 데이터 수집에는 많은 비용과 시간이 들 → 딥러닝에서는 주어진 훈련 집합을 변형하여 인위적으로 늘림.

데이터 증강(data augmentation)

영상에 이동, 회전, 크기, 명암 변환을 랜덤하게 적용해 무한대로 증강 가능

- 오프라인 증강 방식: 데이터 한꺼번에 증강하여 폴더 저장해놓고, 미니 배치 단위로 읽어 학습한다.
- 온라인 증강 방식(주로 사용): 필요할 때마다 새로운 미니 배치를 랜덤 증강으로 생성한다.

◆ 프로그램 8-5 증강된 영상 확인하기

```
import tensorflow.keras.datasets as ds
from tensorflow.keras.preprocessing.image import ImageDataGenerator
# 데이터 증강을 지원하는 ImageDataGenerator 클래스 불러옴.
import matplotlib.pyplot as plt

(x_train,y_train),(x_test,y_test)=ds.cifar10.load_data()
x_train=x_train.astype('float32'); x_train/=255
x_train=x_train[0:15]; y_train=y_train[0:15] # 앞 15개에 대해서만 증대 적용
class_names=['airplane','automobile','bird','cat','deer','dog','flog','horse','ship','truck']

plt.figure(figsize=(20,2))
plt.suptitle("First 15 images in the train set")
for i in range(15):
    plt.subplot(1,15,i+1)
    plt.imshow(x_train[i])
    plt.xticks([]); plt.yticks([])
    plt.title(class_names[int(y_train[i])])
plt.show()

batch_size=4
# 증강을 통해 한 번에 생성해낼 미니 배치 크기
generator=ImageDataGenerator(rotation_range=20.0,width_shift_range=0.2,height_shift_range=0.2,horizontal_flip=True)
# 회전은 -20~20도, 가로방향이동은 20% 이내, 좌우 반전 시도
gen=generator.flow(x_train,y_train,batch_size=batch_size)
# x_train에서 증강하라고 지시하고 미니 배치 크기 설정

for a in range(3): #세번에 걸쳐 데이터를 생성
    img,label=gen.next() # 미니 배치만큼 생성
    # 호출할 때마다 데이터를 증강해 생성하는 역할.
    plt.figure(figsize=(8,2.4))
    plt.suptitle("Generatior trial "+str(a+1))
    for i in range(batch_size):
        plt.subplot(1,batch_size,i+1)
        plt.imshow(img[i])
        plt.xticks([]); plt.yticks([])
        plt.title(class_names[int(label[i])])
    plt.show()
```



First 15 images in the train set

automobile



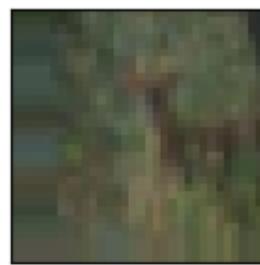
truck



ship

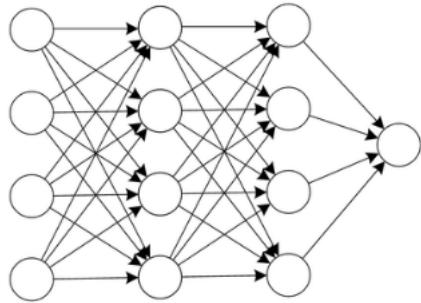


deer

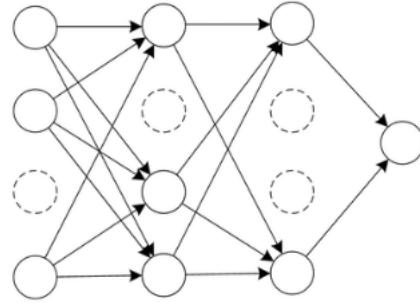


드롭아웃

- 특징 맵을 구성하는 요소 중 일부를 랜덤 선택하여 0으로 설정한다.



(a) Standard Neural Network



(b) Network after Dropout

- 드롭아웃 비율은 0으로 만들 요소의 비율
- 특징 맵의 모든 요소의 합을 유지하기 위해 남은 요소에 $1/(1-r)$ 을 곱한다.
- 학습할 때 적용, 예측할 땐 적용하지 않는다.

조기 멈춤

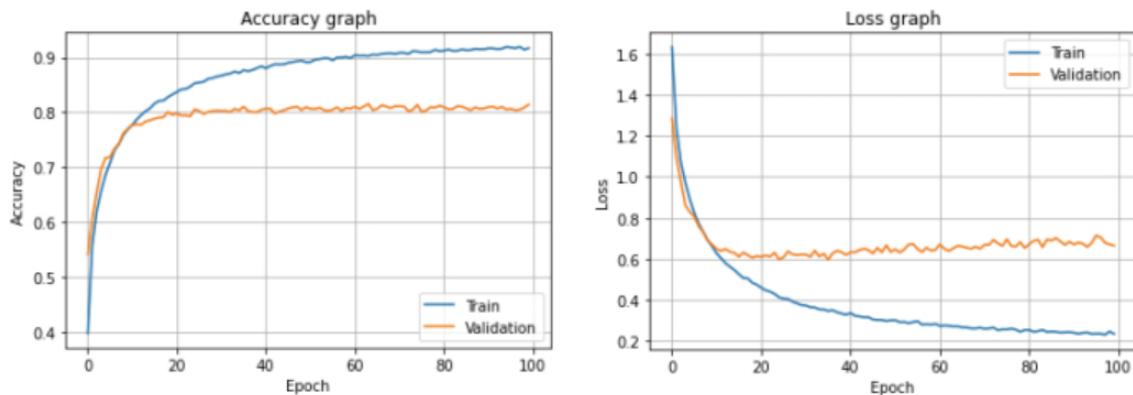


그림 8-21 [프로그램 8-2]의 성능 그래프(왼쪽은 정확률, 오른쪽은 손실 함수)

- 과잉적합 나타난다
⇒ epoch = 40에서 early stopping을 적용하면 효과적
- EarlyStopping 클래스 선언

```
tf.keras.callbacks.EarlyStopping(monitor="val_loss", min_delta=0, patience=0,
                                 verbose=0, mode="auto", baseline=None, restore_
                                 best_weights=False)
```

- monitor : 어떤 측정치를 기준으로 삼을 것인지
기본값은 'val_loss' - 검증 집합의 손실 함수를 사용
- min_delta : 그것보다 작은 개선은 개선으로 여기지 말라.
- patience: 그 값에 해당하는 세대 동안 개선이 없으면 조기에 학습을 멈춰라(지정한 세대에 도달하지 않았더라도)
- restore_best_weights: true면 가장 좋았던 세대의 가중치, False면 멈추는 순간의 가중치를 취하라.

8.7 전이 학습

전이 학습(transfer learning): 어떤 도메인의 데이터로 학습한 모델을 다른 도메인에 적용해 성능을 높이는 방법

ex) ImageNet 데이터셋으로 학습한 모델로 개의 품종 인식하는데 전이 학습을 한다.

8.7.1 백본 모델

사전 학습 모델(pretrained model): 대용량 데이터로 미리 학습되어 있어 전이학습에 활용가능한 모델

→ 사전 학습 모델을 특정 응용에 전이 학습하면 백본 모델로 사용했다고 표현한다.

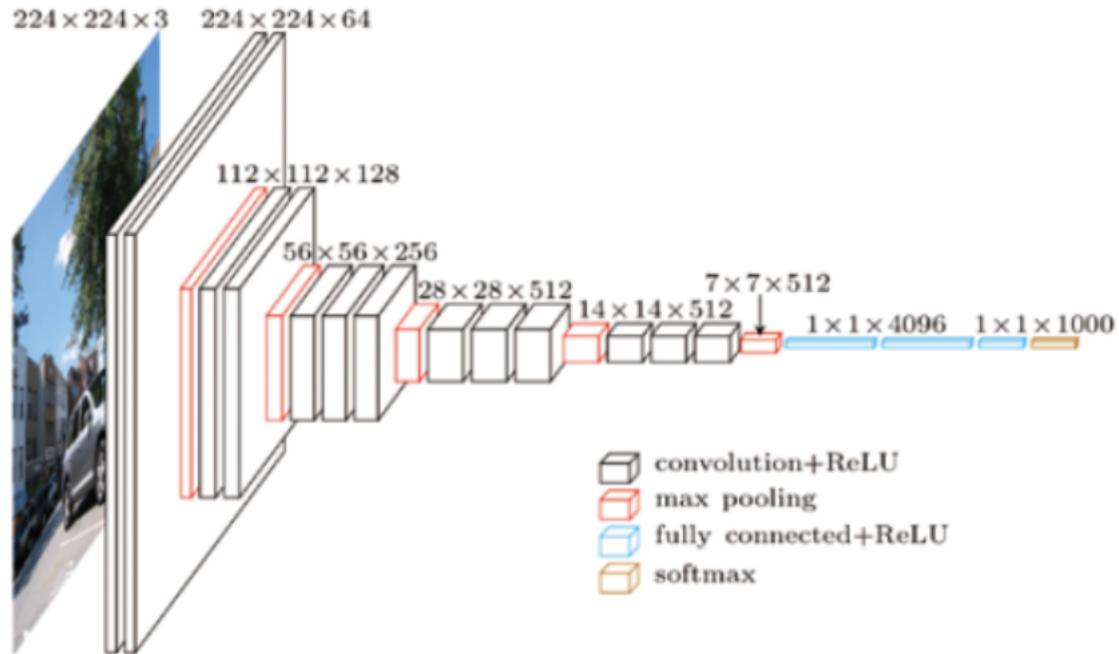
→ DenseNet을 개의 품종을 인식하는 데 전이했다면, 그 프로그램이 DenseNet을 백본 모델로 사용했다.

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>)

VGGNet

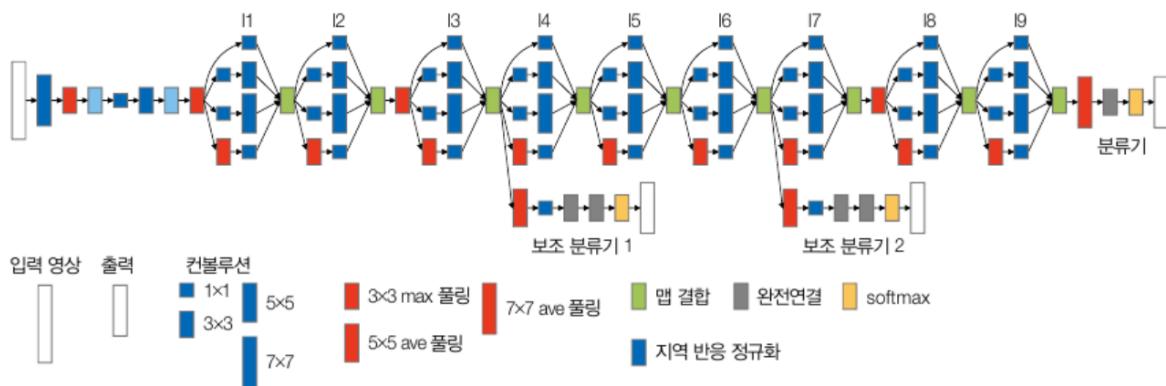
3x3 의 작은 마스크 사용. 층의 깊이를 16으로 깊게 함



(a) VGGNet

GoogLeNet

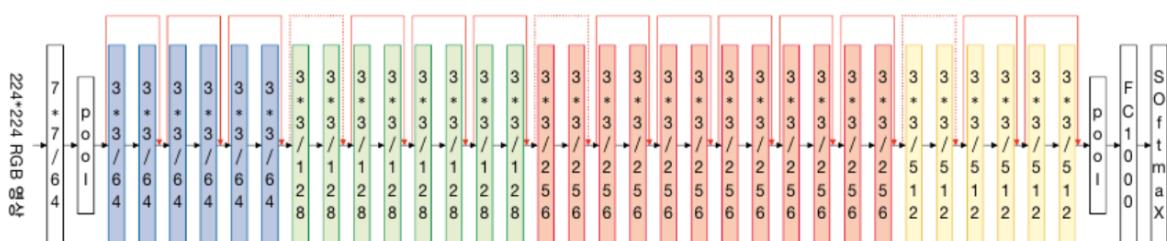
CNN안에 MLP를 둔 네트워크 속의 네트워크



(b) GoogLeNet

ResNet

- skip connection(지름길 연결): 이전 층의 특징 맵을 현재 층의 결과에 더함.
- 지름길로 도달한 특징맵과 현재 특징맵은 요소별 덧셈(element-wise addition)을 사용하여 결합한다.



(c) ResNet

8.7.2 사전 학습 모델로 자연 영상 인식

◆ 프로그램 8-6 ResNet50으로 자연 영상 인식하기

```
import cv2 as cv
import numpy as np
from tensorflow.keras.applications.resnet50 import ResNet50, preprocess_input, decode_predictions

model=ResNet50(weights='imagenet')
# ImageNet으로 학습된 가중치를 읽어오라.

# 테스트 영상을 읽고 신경망에 입력할 수 있는 형태로 변환
img=cv.imread('rabbit.jpg')
x=np.reshape(cv.resize(img,(224,224)),(1,224,224,3))
# cv.resize: ResNet50모델의 입력 크기인 224X224로 변환
# np.reshape: 224x224x3 텐서를 1x224x224x3텐서로 변환
# 변환이유: predict함수가 여러 장으로 구성된 미니 배치 단위로 입력 받기 때문
x=preprocess_input(x)
# ResNet50모델에 영상 입력전 수행하는 전처리 적용

preds=model.predict(x)
# preds : 1x1000배열로 출력됨(imageNet이 1000개의 부류이므로)
# 1장 말고 16장으로 구성된 미니 배치였다면 16x1000(영상마다 확률가지므로)
top5=decode_predictions(preds,top=5)[0]
#1000개 확률 중 가장 큰 5개 확률을 취함
print('예측 결과:',top5)

for i in range(5):
    cv.putText(img,top5[i][1]+':'+str(top5[i][2]),(10,20+i*20),cv.FONT_HERSHEY_SIMPLEX,0.5,(255,255,255),1)

cv.imshow('Recognition result',img)

cv.waitKey()
cv.destroyAllWindows()
```

예측 결과: [('n02325366', 'wood_rabbit', 0.74275386), ('n02326432', 'hare', 0.24023663), ('n02328150', 'Angora', 0.008815719), ('n01877812', 'wallaby', 0.0026892424), ('n02356798', 'fox_squirrel', 0.0012279281)]



8.7.3 사전 학습 모델로 견종 인식

◆ 프로그램 8-7 DenseNet121로 견종 인식하기

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten,Dense,Dropout,Rescaling
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications.densenet import DenseNet121
from tensorflow.keras.utils import image_dataset_from_directory
import pathlib #폴더를 다룰 때 씀

data_path=pathlib.Path('datasets/stanford_dogs/images/images')
# Stanford dogs 데이터셋이 있는 폴더 지정

train_ds=image_dataset_from_directory(data_path,validation_split=0.2,subset='training',seed=123,image_size=(224,224),batch_size=16)
test_ds=image_dataset_from_directory(data_path,validation_split=0.2,subset='validation',seed=123,image_size=(224,224),batch_size=16)
'''

1번째 인수: 데이터 저장된 폴더 경로 지정
2번째 인수: 훈련집합과 검증 집합을 8:2로 분할
3번째 인수: 훈련 집합을 취하라고 지시
4번째 인수: 난수 씨앗 고정
5번째 인수: 영상을 읽으면서 해당 크기로 변환
마지막 인수: 미니 배치크기 설정
'''


base_model=DenseNet121(weights='imagenet',include_top=False,input_shape=(224,224,3))
'''
```

```

1번쨰 인수: ImageNet데이터셋으로 사전 학습된 가중치를 가져오라
2번째 인수: 모델 뒤쪽의 FC-layer을 포함시키지 말라
3번째 인수: 신경망 입력이 224x224x3 텐서임을 알려줌
...
cnn=Sequential()
cnn.add(Rescaling(1.0/255.0)) #입력텐서를 255로 나누어 [0,1]범위로 변환하는 층 추가
cnn.add(base_model)
cnn.add(Flatten())
cnn.add(Dense(1024,activation='relu'))
cnn.add(Dropout(0.75))
cnn.add(Dense(units=120,activation='softmax'))

cnn.compile(loss='sparse_categorical_crossentropy',optimizer=Adam(learning_rate=0.000001),metrics=['accuracy'])
hist=cnn.fit(train_ds,epochs=200,validation_data=test_ds,verbose=2)

print('정확률=',cnn.evaluate(test_ds,verbose=0)[1]*100)

cnn.save('cnn_for_stanford_dogs.h5') # 미세 조정된 모델을 파일에 저장
#fine-tuning: 사전 학습된 모델 뒤에 새로운 층을 붙여 신경망을 구성하
#고 학습률을 낮게 설정해 다시 학습하는 방식

import pickle #개의 품종이름을 파일에 저자해 나중에 쓸수있게 함.
f=open('dog_species_names.txt','wb')
pickle.dump(train_ds.class_names,f)
#pickle.dump()는 파이썬 객체를 파일에 저장해줌
f.close()

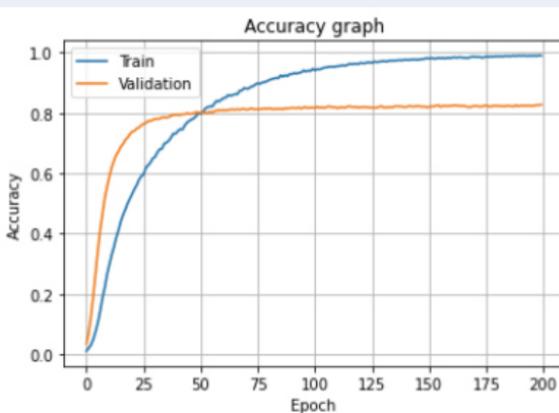
import matplotlib.pyplot as plt

plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('Accuracy graph')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train','Validation'])
plt.grid()
plt.show()

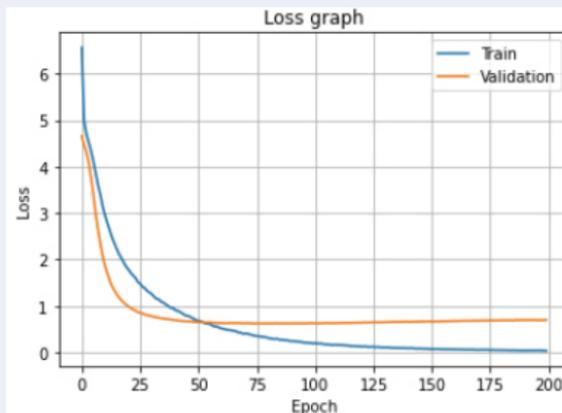
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('Loss graph')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train','Validation'])
plt.grid()
plt.show()

```

④



⑤



8.8 견종 인식 프로그램

◆ 프로그램 8-7 DenseNet121로 견종 인식하기

```

import cv2 as cv
import numpy as np
import tensorflow as tf
import winsound
import pickle
import sys
from PyQt5.QtWidgets import *

```

```

#앞에서 저장해 둔 신경망 모델 읽기
cnn=tf.keras.models.load_model('cnn_for_stanford_dogs.h5') # 모델 읽기
#파일에 저장되어 있는 품종이름 읽기
dog_species=pickle.load(open('dog_species_names.txt','rb')) # 견종 이름

class DogSpeciesRecognition(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('견종 인식')
        self.setGeometry(200,200,700,100)

        fileButton=QPushButton('강아지 사진 열기',self)
        recognitionButton=QPushButton('품종 인식',self)
        quitButton=QPushButton('나가기',self)

        fileButton.setGeometry(10,10,100,30)
        recognitionButton.setGeometry(110,10,100,30)
        quitButton.setGeometry(510,10,100,30)

        fileButton.clicked.connect(self.pictureOpenFunction)
        recognitionButton.clicked.connect(self.recognitionFunction)
        quitButton.clicked.connect(self.quitFunction)

    def pictureOpenFunction(self): #강아지 사진 열기 버튼
        #폴더에서 사진 파일을 브라우징
        fname=QFileDialog.getOpenFileName(self,'강아지 사진 읽기','..')
        #파일 읽어와 객체 저장
        self.img=cv.imread(fname[0])
        if self.img is None: sys.exit('파일을 찾을 수 없습니다.')

        cv.imshow('Dog image',self.img)

    def recognitionFunction(self): # 품종 인식 버튼
        #이미지를 신경망에 입력할 준비(224x224->1x224x224x3텐서로 만들)
        x=np.reshape(cv.resize(self.img,(224,224)),(1,224,224,3))
        res=cnn.predict(x)[0] # 예측
        top5=np.argsort(-res)[:5] #오름차순 정렬하고 확률이 가장 큰 5개 인덱스만 취함
        top5_dog_species_names=[dog_species[i] for i in top5]
        #견종이름을 저장해둔 객체를 이용해 인덱스를 견종 이름으로 변환
        for i in range(5):#영상에 품종 이름과 확률을 씀
            prob='('+str(res[top5[i]])+')'
            name=str(top5_dog_species_names[i]).split('-')[1]
            cv.putText(self.img,prob+name,(10,100+i*30),cv.FONT_HERSHEY_SIMPLEX,0.7,(255,255,255),2)
        cv.imshow('Dog image',self.img)
        winsound.Beep(1000,500)

    def quitFunction(self):
        cv.destroyAllWindows()
        self.close()

app=QApplication(sys.argv)
win=DogSpeciesRecognition()
win.show()
app.exec_()

```

Chapter 09 인식

9.1 인식이란

9.1.1 인식의 세부문제

사람과 달리 컴퓨터 비전은 인식 문제를 **분류**, **검출**, **분할**, **추적**, **행동** 분류의 세부 문제로 구분해 봄다

- **분류**: 영상에 있는 물체의 부류 정보를 알아냄
 - **사례 분류**: 내 차나 어린이 교통표지판 같이 특정한 물체를 알아냄
 - **범주 분류**: 고양이나 자전거처럼 물체 부류를 알아냄
- **검출**: 물체를 찾아 직사각형(bounding box)으로 위치 표현. 신뢰도를 같이 출력함
 - **보통** 물체의 종류는 미리 정해짐
- **분할**: 물체가 점유하는 영역, 픽셀 집합을 지정하고 부류 확률을 같이 출력
 - **의미 분할**: 픽셀 각각에 대해 물체 부류를 지정

- 사례 분할: 같은 부류에 속하는 물체가 여럿이면 서로 다른 번호를 부여
⇒ 영역을 알면 박스를 구할 수 있으므로 검출은 분할의 부분문제임
- 추적: 비디오에 나타난 물체의 이동 궤적을 표시
 - 시각 물체 추적(VOT), 다중 물체 추적(MOT)
- 행동 분류: 물체가 수행하는 행동의 종류를 알아냄



그림 9-2 인식을 여러 세부 문제로 나누는 컴퓨터 비전

9.1.3 사람과 컴퓨터 비전의 인식 과정 비교

사람

1. 분할, 분류, 추적, 행동 분류를 동시에 수행한다
 2. 선택적 주의 집중을 발휘한다
 3. 문맥을 이용하는데 능숙하다.
- 영상의 특정 지역에 집중해서 알아낸 **지역 정보**와 영상 전체를 해석해서 알아낸 **문맥 정보**를 결합하여 최적의 의사결정을 한다.

컴퓨터 비전

1. 인식을 세부문제로 나누고 독립적으로 해결
 2. 의도 없고 선택적 주의 집중 할 수 없다
- 사람의 의도를 고안하고 컴퓨터 비전의 학습 과정에 반영하는 연구는 아직 시작되지도 않았다. 먼 훗날의 연구주제~

9.2 분류

9.2.1 고전 방법론

- 사례분류: 모양과 텍스처가 고정된 특정 물체를 찾는 문제(객체) ex) 차, 교통 표지판
- 범주분류: 일반 부류에 속하는 물체를 알아내는 문제. 모양이 변하는 물체까지 포함. ex) 자전거, 코끼리

9.2.2 딥러닝 방법론

사례 분류 문제는 이미 풀렸다고 간주하고 **범주 분류**에 공을 들인다.

범주 분류를 다루는 대표적인 대회: PASCAL VOC, ILSVRC

- ILSVRC대회에서 2012년 AlexNet이 우승한 이후 CNN이 주류를 이룬다.

- 미세 분류(fine-grained classification): 개나 새의 품종 알아내기



Laysan Albatross



Rusty Blackbird

Fish Crow

Brewer Blackbird

Shiny Cowbird

그림 9-11 부류 내 변화가 크고 부류 간 변화가 작은 미세 분류 문제(CUB-200 데이터셋)

첫째 행: 부류 내 변화가 크다

둘째 행: 서로 다른 품종에서 부류 간 변화가 적다.

→ 현재는 ResNet을 이용한 전이학습을 적용해 82%의 정확률을 달성

◆ 설명가능

인간은 의사결정의 이유를 설명하는 능력이 뛰어남.

딥러닝은 분류 성능에는 획기적인 향상을 가져왔지만 분류 결과에 대한 이유를 설명하지 못한다.



91.5% 확률로 붉은 날개
검은 새로 분류

하이아이트 영역을 보고
91.5% 확률로 붉은 날개
검은 새로 분류

노란 띠와 붉은 반점을
보고 91.5% 확률로 붉은
날개 검은 새로 분류

그림 9-12 딥러닝의 설명 수준(왼쪽부터 설명 없음, 시각화 수준 설명, 문장 수준 설명)

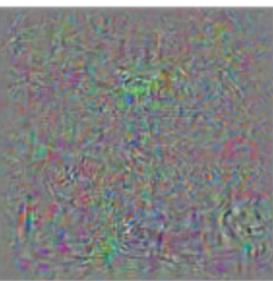
설명 기능 없는 모델 / 시각화 수준의 설명 가능 모델/ 문장 수준의 설명 가능한 모델

⇒ 설명 기능 확보하려는 많은 연구가 진행되고 있다.

CAM(Class activation map): CNN의 뒷부분에 있는 FC-layer층을 전역평균풀링층(GAP: global average pooling layer)로 대체한 신경망에서만 작동하는 한계

GradCAM: CAM의 한계를 극복해 다양한 신경망 구조에 적용 가능한 기법

◆ 적대적 공격과 방어 전략



(a) 자연 영상 교란[Szegedy2013]

(b) 교통 표지판에 스티커 부착

그림 9-13 적대적 공격

(a) 딥러닝은 오른쪽 영상을 타조로 분류함

영상을 적절히 조작하면 모델을 속여 원하는 부류로 분류하게 할 수 있다

(b) 교통 표지판 영상에 적절한 형태로 스티커 붙이면 모델을 속일 수 있다.

→ 딥러닝 모델의 적대적 공격에 대한 기술적 취약성을 이해하고 이에 대응하기 위한 방어 전략 또한 활발하게 연구되고 있음

9.3 검출

검출: 영상에서 여러 개의 물체를 찾아 위치와 함께 부류를 지정해야 함 → 분류에 비해 훨씬 어렵다.

9.3.1 성능 척도

검출 모델: 입력 영상에 있는 물체를 **bounding box**라는 직사각형으로 지정하는데, 박스마다 물체 부류와 신뢰도를 같이 출력한다.

성능 지표: **mAP(mean Average Precision)**

→ 물체 부류 각각에 대해 AP계산하고 이를 AP를 모든 부류에 대해 평균하여 구한다.

9.3.2 고전 방법

9.3.3 컨볼루션 신경망: RCNN계열

9.3.4 컨볼루션 신경망: YOLO계열

9.3.4 프로그래밍 실습: YOLO v3으로 물체 검출

9.4 분할

9.4.1 성능 척도와 데이터셋

9.4.2 의미 분할을 위한 FCN

9.4.3 FCN을 개선한 신경망: DeConvNet, U-net, DeepLabv3+

9.4.4 프로그래밍 실습: U-net을 이용한 분할 학습

9.4.4 pixellib 라이브러리를 이용한 프로그래밍 실습

9.5 [비전 에이전트8] 배경을 내 맘대로 바꾸기

9.6 사람 인식