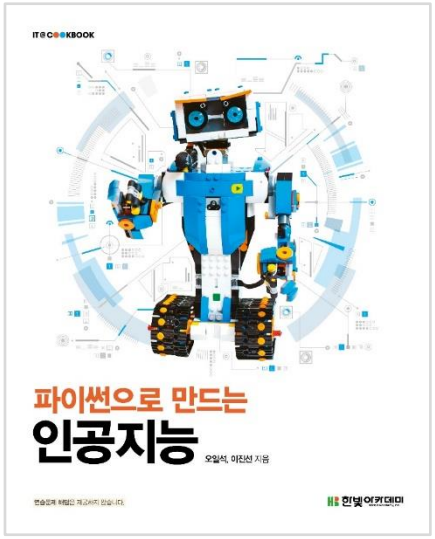
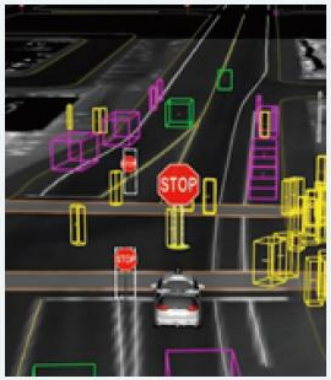
6장

컨볼루션 신경망과 컴퓨터 비전



Preview

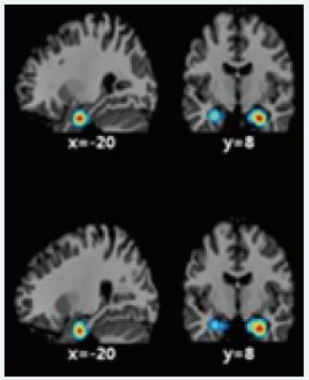
- 컨볼루션 신경망은 딥러닝에서 가장 성공한 모델



(a) 자율주행차

(b) 딥드림

(c) 영상 주석달기



(d) 병변 부위 찾기

그림 6-1 컨볼루션 신경망으로 제작한 인공지능 제품

- 컴퓨터 비전은 인공지능의 가장 중요한 연구 분야 중 하나
  - 시각은 인간의 가장 강력한 인지 기능. 컴퓨터 비전은 인간의 시각 기능 모방
  - 딥러닝 이후 컴퓨터 비전 연구 패러다임 변화: 수작업 규칙 기반 → 대용량 영상 데이터로 기계 학습
  - 컴퓨터 비전(문제)과 딥러닝(해결 도구)은 상호작용을 통해 공진화

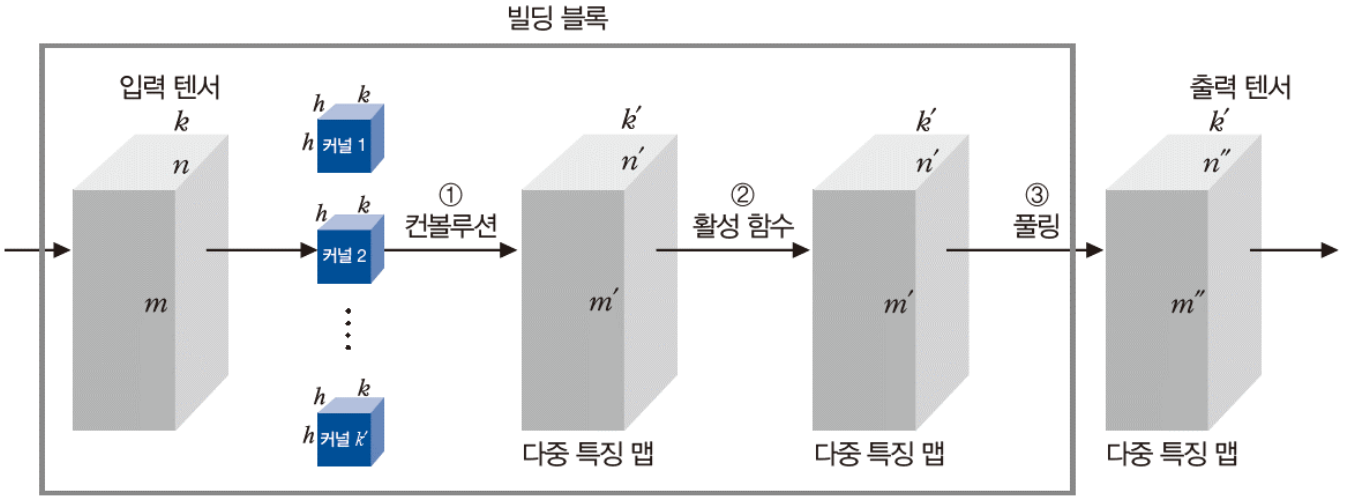
6.2 컨볼루션 신경망의 구조와 동작

- 이 절은
  - 컨볼루션 연산의 원리와 특성을 소개
  - 컨볼루션 신경망이 어떻게 복잡한 컴퓨터 비전 문제를 푸는지 설명
  - 컨볼루션 신경망을 구성하는 빌딩 블록과 빌딩 블록을 쌓아 신경망을 만드는 원리 설명

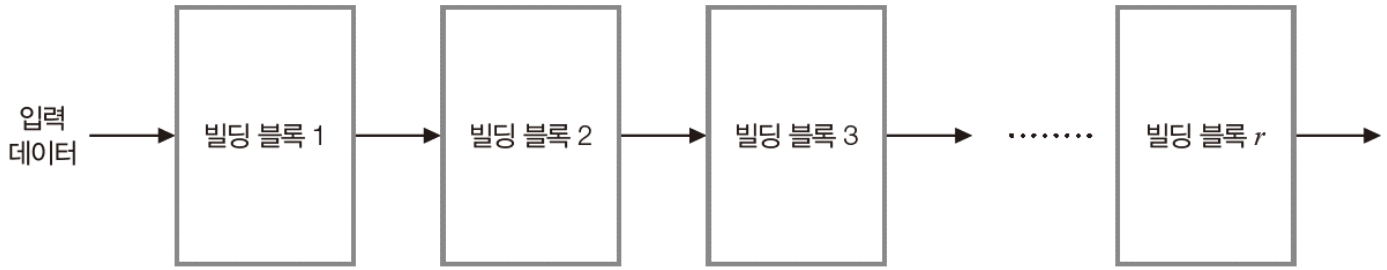
6.2.3 빌딩 블록을 쌓아 만드는 컨볼루션 신경망

빌딩 블록

- k*m*n 입력 텐서에 k′개의 커널을 적용하면 k′*m′*n′ 크기의 특징 맵이 됨
- 풀링을 적용하면 k′*m″*n″ 특징 맵이 됨. 이 텐서는 다음 빌딩 블록의 입력이 됨



(a) 빌딩 블록의 구조

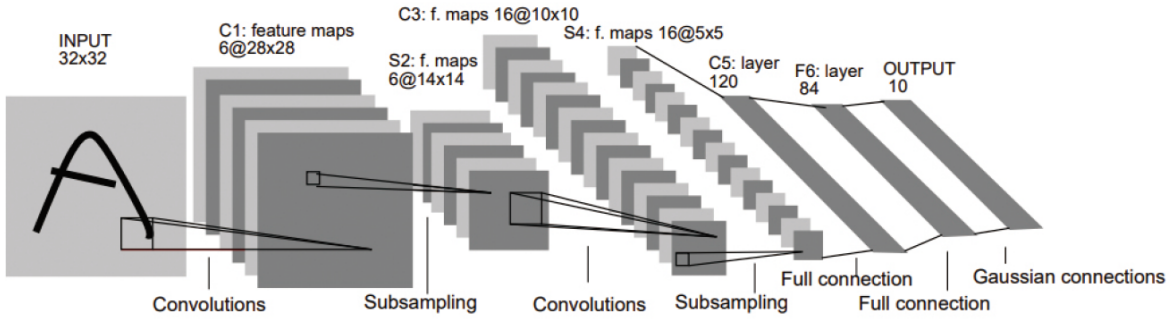


(b) 빌딩 블록을 쌓아 만든 컨볼루션 신경망

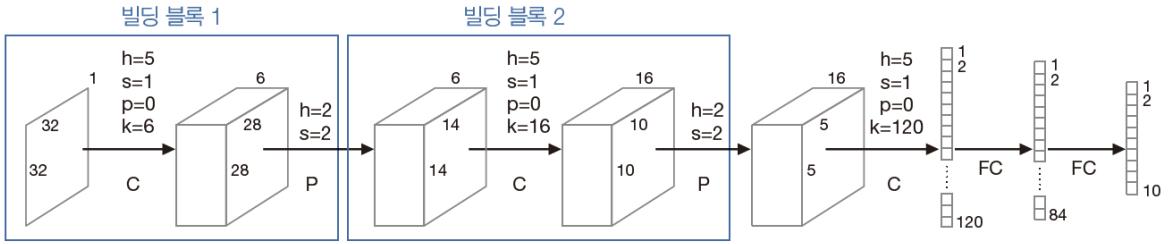
그림 6-10 컨볼루션 신경망의 구조

6.2.3 빌딩 블록을 쌓아 만드는 컨볼루션 신경망

- LeNet-5 사례 [LeCun1998]
  - 현재 기준으로 보면 아주 간단한 컨볼루션 신경망(C-P-C-P-C-FC-FC)
  - 입력은 1*32*32 텐서(32*32 크기의 필기 숫자 맵)
  - 빌딩 블록1
    - 컨볼루션은 5*5 커널을 6개 사용(보폭 1, 덧대기 안함): 1*32*32 텐서 → 6*28*28 텐서
    - 풀링은 2*2 커널을 사용(보폭 2): 6*28*28 텐서 → 6*14*14 텐서



(a) [LeCun1998]의 그림



(b) 이 책의 방식에 따른 그림

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

6.2.3 빌딩 블록을 쌓아 만드는 컨볼루션 신경망

- LeNet-5 사례(...앞에서 계속)
  - 빌딩 블록 2
    - 컨볼루션은 5*5 커널을 16개 사용(보폭 1, 덧대기 안함): 6*14*14 → 16*10*10
    - 풀링은 2*2 커널을 사용(보폭 2): 16*10*10 → 16*5*5
  - 컨볼루션 적용(5*5 커널을 120개 사용, 보폭 1, 덧대기 안함): 16*5*5 → 1*120
  - FC(완전연결) 적용: 1*120 → 1*84
  - FC(완전연결) 적용: 1*84 → 1*10

- LeNet-5의 가중치(매개변수) 수
  - 컨볼루션층은 5*5 커널 6개+5*5 커널 16개 + 5*5 커널 120개 =(5*5+1)*(6+16+120)=3692개
  - FC 층은 (120+1)*84+(84+1)*10=11014개
  - 학습 알고리즘은 14706개의 매개변수를 최적화해야 함
  - FC 층이 컨볼루션 층보다 3배 가량 많음(딥러닝은 FC 층을 전역 평균 풀링 등으로 대치하여 매개변수 수를 줄이는 방향으로 발전)

6.3 컨볼루션 신경망의 학습

- 컨볼루션 신경망은 커널을 학습함
  - [그림 6-10]은 커널을 파란 색으로 표시하여 이 사실을 강조([그림 5-10]의 깊은 다층 퍼셉트론은 노드를 연결하는 가중치를 학습함)
  - 둘의 학습 알고리즘이 다를 것 같지만 다행이 같은 손실 함수와 같은 옵티마이저 사용

6.3.1 손실 함수와 옵티마이저

손실 함수

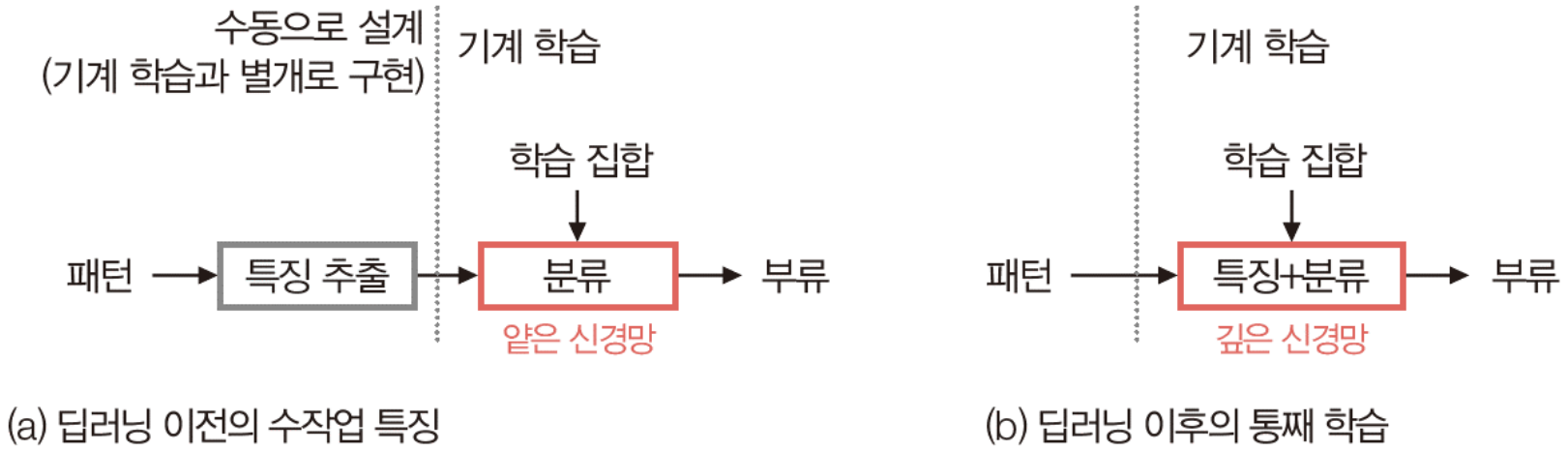
- 다층 퍼셉트론과 컨볼루션 신경망은 중간 층은 다르지만 입력과 출력은 동일하므로 같은 손실 함수 사용
- 손실 함수 목록은 5.7절 참조

옵티마이저

- 다층 퍼셉트론은 에지 가중치, 컨볼루션 신경망은 커널의 화소 값을 최적화한다는 점만 다르지 학습 원리는 동일하므로 같은 옵티마이저 사용
- 옵티마이저는 5.8절 참조

6.3.2 통째 학습

- 고전적인 컴퓨터 비전은 수작업$_{\text{hand-crafted}}$ 특징 사용([그림 6-12(a)])
  - 예) [그림 6-5]의 수직 에지와 수평 에지
  - 사람의 직관으로 설계하기 때문에 어느 정도 성능 달성 가능
  - 주어진 데이터에 최적이 아니기 때문에 자연 영상처럼 복잡한 데이터에서 낮은 성능



(a) 딥러닝 이전의 수작업 특징 (b) 딥러닝 이후의 통째 학습

그림 6-12 딥러닝에 의한 컴퓨터 비전 패러다임 변화

6.3.2 통째 학습

- 딥러닝은 특징을 학습([그림 6-12(b)])
  - 특징 추출에서 패러다임 변화: 특징 추출과 분류를 동시에 학습으로 알아냄
  - 입력에서 출력까지 전 과정을 한꺼번에 학습하므로 통째 학습$_{\text{end-to-end learning}}$ 이라 부름
- [그림 6-13]은 컨볼루션 신경망의 특징 학습을 설명
  - 앞 부분(컨볼루션층과 풀링층)은 특징 추출, 뒤 부분(FC 층)은 분류를 담당

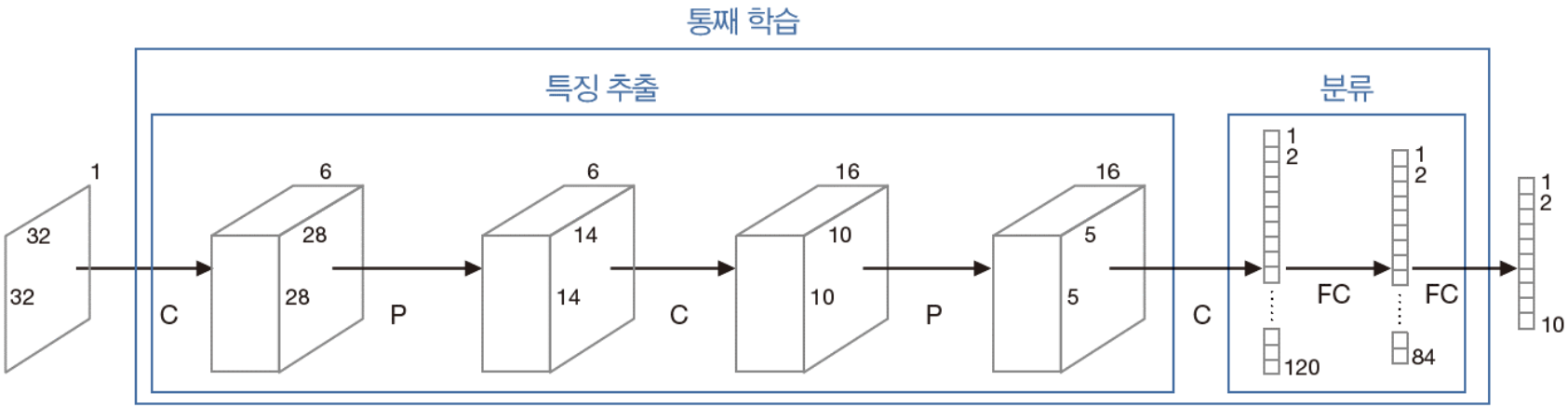


그림 6-13 특징 추출과 분류를 동시에 학습하는 딥러닝의 통째 학습

6.3.3 컨볼루션 신경망의 성능이 월등한 이유

- 네 가지 이유
  - 통째 학습: 따로 최적화하여 붙이는 이전 방식보다 우수
  - 특징 학습: 수작업 특징보다 우수
  - 신경망 깊이를 깊게 하여 풍부한 특징을 추출: 수십~수백 층 사용
  - 데이터의 원래 구조를 유지한 채 특징 추출

6.4 컨볼루션 신경망 프로그래밍

- 여러 가지 실험
  - 실험 데이터
    - MNIST 필기 숫자
    - Fashion MNIST
    - 자연 영상 CIFAR-10
  - 여러 종류의 층을 사용해봄으로써 텐서플로의 유연성 확인

6.4.1 필기 숫자 인식

- LeNet-5 재현
  - C-P-C-P-C-FC-FC 구조
  - [프로그램 5-9]와 비슷함(음영 표시한 부분만 달라짐)
    - 2차원 구조를 유지해야 하므로 reshape(60000,28,28,1)을 사용

프로그램 6-1 LeNet-5로 MNIST 인식

TIP (60000,28,28)이 아니라 (60000,28,28,1)인 이유는 6.4.2항에서 설명한다. 층을 쌓아 신경망을 구축하는 18~26행에 대해서도 6.4.2항에서 구체적으로 설명한다.

```
01  import numpy as np
02  import tensorflow as tf
03  from tensorflow.keras.datasets import mnist
04  from tensorflow.keras.models import Sequential
05  from tensorflow.keras.layers import Conv2D,MaxPooling2D,Flatten,Dense
06  from tensorflow.keras.optimizers import Adam
07
08  # MNIST 데이터셋을 읽고 신경망에 입력할 형태로 변환
09  (x_train,y_train),(x_test,y_test)=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
```

6.4.1 필기 숫자 인식

C-P-C-P-C-FC-FC 구조의
컨볼루션 신경망 설계

```
17  # LeNet-5 신경망 모델 설계
18  cnn=Sequential()
19  cnn.add(Conv2D(6,(5,5),padding='same',activation='relu',input_shape=(28,28,1)))
20  cnn.add(MaxPooling2D(pool_size=(2,2)))
21  cnn.add(Conv2D(16,(5,5),padding='same',activation='relu'))
22  cnn.add(MaxPooling2D(pool_size=(2,2)))
23  cnn.add(Conv2D(120,(5,5),padding='same',activation='relu'))
24  cnn.add(Flatten())
25  cnn.add(Dense(84,activation='relu'))
26  cnn.add(Dense(10,activation='softmax'))
27
28  # 신경망 모델 학습
29  cnn.compile(loss='categorical_crossentropy',optimizer=Adam(),metrics=['accuracy'])
30  hist=cnn.fit(x_train,y_train,batch_size=128,epochs=30,validation_data=(x_test,
    y_test),verbose=2)
31
32  # 신경망 모델 정확률 평가
33  res=cnn.evaluate(x_test,y_test,verbose=0)
34  print("정확률은",res[1]*100)
35
```

6.4.1 필기 숫자 인식

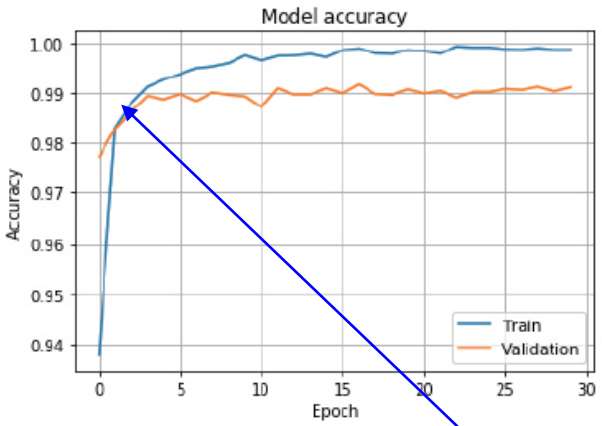
```
36  import matplotlib.pyplot as plt
37
38  # 정확률 그래프
39  plt.plot(hist.history['accuracy'])
40  plt.plot(hist.history['val_accuracy'])
41  plt.title('Model accuracy')
42  plt.ylabel('Accuracy')
43  plt.xlabel('Epoch')
44  plt.legend(['Train','Validation'],loc='best')
45  plt.grid()
46  plt.show()
47
48  # 손실 함수 그래프
49  plt.plot(hist.history['loss'])
50  plt.plot(hist.history['val_loss'])
51  plt.title('Model loss')
52  plt.ylabel('Loss')
53  plt.xlabel('Epoch')
54  plt.legend(['Train','Validation'],loc='best')
55  plt.grid()
56  plt.show()
```

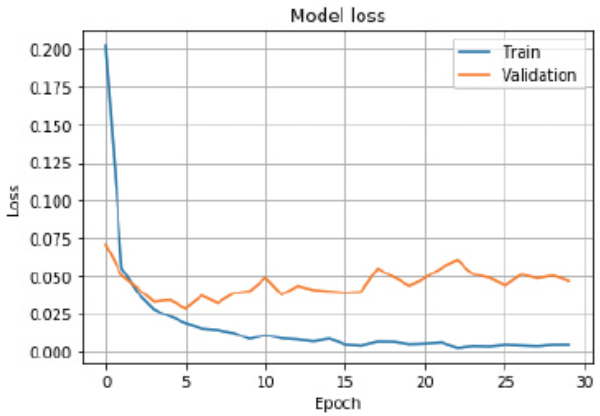
6.4.1 필기 숫자 인식

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/30
60000/60000 - 29s - loss: 0.2021 - accuracy: 0.9378 - val_loss: 0.0708 - val_accuracy: 0.9772
Epoch 2/30
60000/60000 - 31s - loss: 0.0552 - accuracy: 0.9830 - val_loss: 0.0501 - val_accuracy: 0.9828
Epoch 3/30
60000/60000 - 49s - loss: 0.0385 - accuracy: 0.9879 - val_loss: 0.0417 - val_accuracy: 0.9862
...
Epoch 29/30
60000/60000 - 30s - loss: 0.0043 - accuracy: 0.9987 - val_loss: 0.0500 - val_accuracy: 0.9903
Epoch 30/30
60000/60000 - 32s - loss: 0.0044 - accuracy: 0.9987 - val_loss: 0.0465 - val_accuracy: 0.9911

정확률은 99.1100013256073
```

99.11% 정확률을 얻어 97.99%의
깊은 다층 퍼셉트론에 비해 1.12% 우수





수렴 속도도 빠름

6.4.1 필기 숫자 인식

- 컨볼루션 신경망의 유연한 구조
  - [프로그램 6-2]는 keras.io 사이트가 제공하는 신경망 구조와 하이퍼 매개변수 사용
  - C-C-P-dropout-FC-dropout-FC 구조(dropout은 규제 기법이므로 C-C-P-FC-FC 구조)

TIP keras.io에는 유용한 예제 프로그램이 아주 많다. https://keras.io/examples/mnist_cnn에 접속하면 MNIST 인식을 위한 컨볼루션 신경망 프로그램을 구할 수 있다. 이 프로그램은 옵티마이저로 Adadelta를 쓰고 있는데, Adadelta를 쓰면 최종 성능이 80%대를 벗어나지 못한다. 따라서 29행에서 Adam을 설정했다.

프로그램 6-2 컨볼루션 신경망으로 MNIST 인식

```
01  import numpy as np
02  import tensorflow as tf
03  from tensorflow.keras.datasets import mnist
04  from tensorflow.keras.models import Sequential
05  from tensorflow.keras.layers import Conv2D,MaxPooling2D,Flatten,Dense,Dropout
06  from tensorflow.keras.optimizers import Adam
07
08  # MNIST 데이터셋을 읽고 신경망에 입력할 형태로 변환
09  (x_train,y_train),(x_test,y_test)=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
```

6.4.1 필기 숫자 인식

C-C-P-FC-FC 구조의
컨볼루션 신경망 설계

```
17  # 신경망 모델 설계
18  cnn=Sequential()
19  cnn.add(Conv2D(32,(3,3),activation='relu',input_shape=(28,28,1)))
20  cnn.add(Conv2D(64,(3,3),activation='relu'))
21  cnn.add(MaxPooling2D(pool_size=(2,2)))
22  cnn.add(Dropout(0.25))
23  cnn.add(Flatten())
24  cnn.add(Dense(128,activation='relu'))
25  cnn.add(Dropout(0.5))
26  cnn.add(Dense(10,activation='softmax'))
27
28  # 신경망 모델 학습
29  cnn.compile(loss='categorical_crossentropy',optimizer=Adam(),metrics=['accuracy'])
30  hist=cnn.fit(x_train,y_train,batch_size=128,epochs=12,validation_data=(x_test,
    y_test),verbose=2)
31
32  # 신경망 모델 정확률 평가
33  res=cnn.evaluate(x_test,y_test,verbose=0)
34  print("정확률은",res[1]*100)
35
```

TIP 드롭아웃은 학습 도중에 일정 비율의 가중치를 무작위로 골라 불능으로 만드는 규제 기법이다. 드롭아웃에 대해서는 6.6.2항에서 설명한다.

성능 향상을 위해 Adam
옵티마이저 사용

6.4.1 필기 숫자 인식

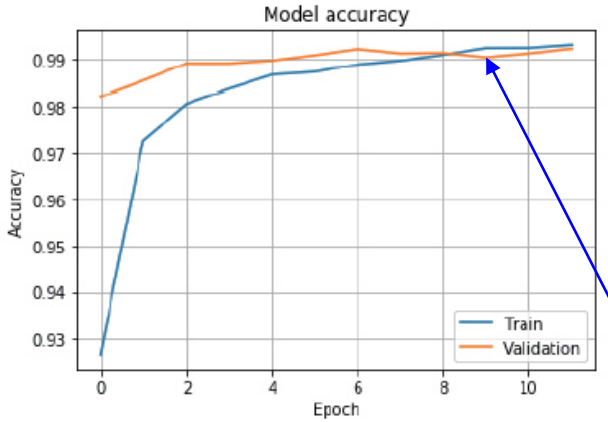
```
36  import matplotlib.pyplot as plt
37
38  # 정확률 그래프
39  plt.plot(hist.history['accuracy'])
40  plt.plot(hist.history['val_accuracy'])
41  plt.title('Model accuracy')
42  plt.ylabel('Accuracy')
43  plt.xlabel('Epoch')
44  plt.legend(['Train','Validation'], loc='best')
45  plt.grid()
46  plt.show()
47
48  # 손실 함수 그래프
49  plt.plot(hist.history['loss'])
50  plt.plot(hist.history['val_loss'])
51  plt.title('Model loss')
52  plt.ylabel('Loss')
53  plt.xlabel('Epoch')
54  plt.legend(['Train','Validation'], loc='best')
55  plt.grid()
56  plt.show()
```

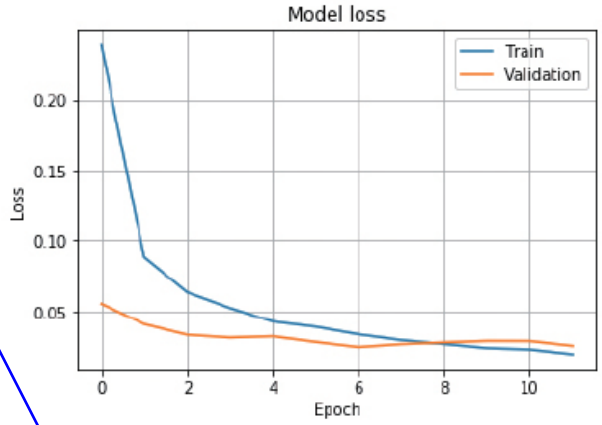
6.4.1 필기 숫자 인식

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 - 81s - loss: 0.2378 - accuracy: 0.9268 - val_loss: 0.0553 - val_accuracy: 0.9822
Epoch 2/12
60000/60000 - 85s - loss: 0.0879 - accuracy: 0.9728 - val_loss: 0.0411 - val_accuracy: 0.9856
...
Epoch 10/12
60000/60000 - 80s - loss: 0.0245 - accuracy: 0.9924 - val_loss: 0.0295 - val_accuracy: 0.9905
Epoch 11/12
60000/60000 - 80s - loss: 0.0235 - accuracy: 0.9925 - val_loss: 0.0294 - val_accuracy: 0.9913
Epoch 12/12
60000/60000 - 79s - loss: 0.0202 - accuracy: 0.9931 - val_loss: 0.0260 - val_accuracy: 0.9923
```

정확률은 99.22999739646912

99.23% 정확률을 얻어 99.11%의 LeNet-5에 비해 0.12% 우수





세대 수를 늘리면 성능 향상 여지 있음

6.4.2 텐서플로 프로그래밍

- 텐서플로 프로그래밍에서 중요한 네 가지 클래스
  - models 클래스(여기서 설명)
  - layers 클래스(여기서 설명)
  - loss 클래스(5.7절에서 설명):
  - optimizers 클래스(5.8절에서 설명)

← 텐서플로 프로그램의 기본이니 잘 기억하길 바람
(keras.io 사이트 도움말 중요)

- models 클래스: Sequential과 functional API
  - Sequential 함수
    - 순차적으로 층을 쌓아 모델을 만드는 경우에 사용
    - 대부분 신경망이 이 구조에 해당
    - [프로그램 6-2]에서는
      - 18행이 Sequential 함수로 cnn이라는 객체 생성
      - 19~26행은 add 함수로 층을 쌓아 신경망 설계
  - functional API 함수
    - 텐서가 흐르다가 중간에서 여러 개로 갈라지는 경우(출력이 여러 개일 수 있음)
    - 시계열이나 생성 모델에서 종종 발생(예, [프로그램 10-2])

6.4.2 텐서플로 프로그래밍

- layers 클래스: Dense, Conv2D, MaxPooling2D, Flatten 함수
  - 입력 층을 쌓는 [프로그램 6-2]의 19행

```
19   cnn.add(Conv2D(32,(3,3),activation='relu',input_shape=(28,28,1)))
```

    - input_shape=(28,28,1) 매개변수는 신경망에 (28*28*1) 텐서가 입력된다는 사실 알려줌
      - (28*28) 대신 (28*28*1)을 사용하는 이유는 일반성 유지(RGB 영상의 경우 (28*28*3)으로 확장)
    - 맨 앞 두 개 매개변수 32,(3,3)은 3*3 크기의 커널을 32개 사용하라는 뜻
      - [그림 6-7]에 따르면 k′=32이고 h=3인 셈
    - activation='relu'는 컨볼루션 결과에 ReLU 활성 함수를 적용하라는 뜻
  - Conv2D의 API

[Conv2D 함수의 API]

```
Tensorflow.layers.Conv2D(filters, kernel_size, strides=(1,1),
padding='valid', data_format=None, dilation_rate=(1,1), activation=None,
use_bias=True, kernel_initializer='glorot_uniform', bias_
initializer='zeros', kernel_regularizer=None, bias_regularizer=None,
activity_regularizer=None, kernel_constraint=None, bias_constraint=None)
```

6.4.2 텐서플로 프로그래밍

- layers 클래스 (...앞에서 계속)
  - 은닉층과 출력층을 쌓는 [프로그램 6-2]의 20~26행

```
20  cnn.add(Conv2D(64,(3,3),activation='relu'))
21  cnn.add(MaxPooling2D(pool_size=(2,2)))
22  cnn.add(Dropout(0.25))
23  cnn.add(Flatten())
24  cnn.add(Dense(128,activation='relu'))
25  cnn.add(Dropout(0.5))
26  cnn.add(Dense(10,activation='softmax'))
```

  - 19행의 입력층 이후에는 input_shape 매개변수 필요 없음(이전 층의 텐서 구조를 알고 있기 때문)
  - 22행과 25행은 드롭아웃 적용
  - 23행의 Flatten은 다차원 구조를 1차원 구조로 변환
  - 24행과 26행은 완전연결층을 쌓음

TIP 드롭아웃은 학습 도중에 일정 비율의 가중치를 무작위로 골라 불능으로 만드는 규제 기법이다. 드롭아웃에 대해서는 6.6.2항에서 설명한다.

6.4.2 텐서플로 프로그래밍

TIP 함수의 API를 살펴보면서 자신이 설계한 신경망 구조를 정확히 이해하는 일은 매우 중요하다. 예제 프로그램을 따라 하는 데서 그치면 딥러닝 프로그래밍 능력이 갖춰지지 않는다. 테니스를 배울 때 책에 있는 포즈만 연습하면 프로가 되지 못하는 것과 같다. 책에 있는 내용을 존중하고 충실히 따라함과 동시에 자신의 포즈를 이리저리 궁리해야 하는 것과 마찬가지로 딥러닝 프로그래밍을 잘 하려면 여러가지 선택사항을 바꿔가며 반복적으로 연습해야 한다.

6.4.3 패션 인식

- [프로그램 6-3]은 fashion MNIST 인식하는 컨볼루션 신경망
  - [프로그램 6-2]에서 달라지는 부분은 데이터를 읽어들이는 부분 뿐

프로그램 6-3 컨볼루션 신경망으로 fashion MNIST 인식

```
    # 03행과 09행을 제외하고 나머지는 [프로그램 6-2]와 같음
…
03  from tensorflow.keras.datasets import fashion_mnist
…
09  (x_train,y_train),(x_test,y_test)=fashion_mnist.load_data()
…
```

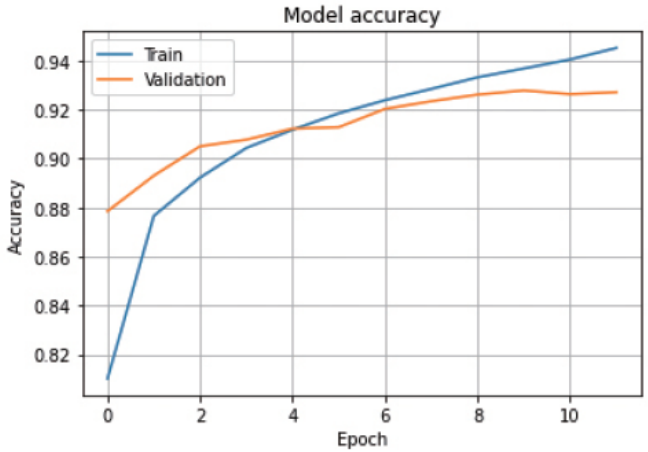
TIP fashion MNIST 데이터에 대해서는 5.4.3항과 [프로그램 5-8], [프로그램 5-11]을 참조한다.

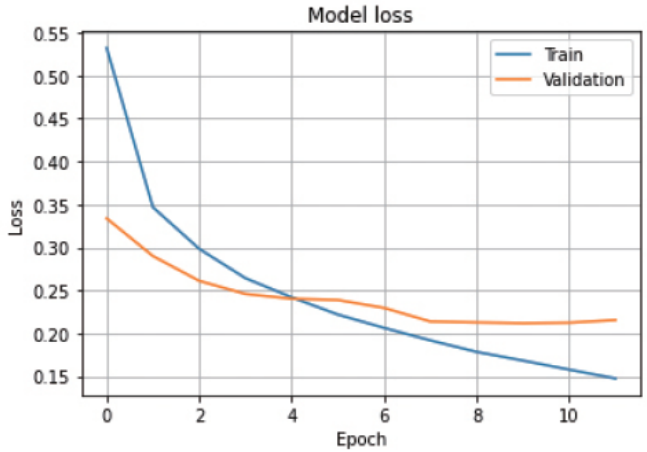
6.4.3 패션 인식

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 - 41s - loss: 0.5322 - accuracy: 0.8102 - val_loss: 0.3338 - val_accuracy: 0.8785
Epoch 2/12
60000/60000 - 40s - loss: 0.3467 - accuracy: 0.8766 - val_loss: 0.2899 - val_accuracy: 0.8930
...
Epoch 11/12
60000/60000 - 40s - loss: 0.1577 - accuracy: 0.9405 - val_loss: 0.2124 - val_accuracy: 0.9263
Epoch 12/12
60000/60000 - 42s - loss: 0.1474 - accuracy: 0.9452 - val_loss: 0.2154 - val_accuracy: 0.9271
```

정확률은 92.71000027656555 ←

92.71% 정확률을 얻어 89.67%의
[프로그램 5-12]에 비해 3.04% 우수





6.4.4 자연 영상 인식

- 딥러닝 프로그래밍에서 주로 사용하는 자연 영상 데이터베이스
  - ImageNet
  - MSCoCo
  - CIFAR: 작아서 MNIST 다음에 주로 사용
- CIFAR-10
  - {airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck}의 10부류
  - 영상은 32*32 맵으로 표현

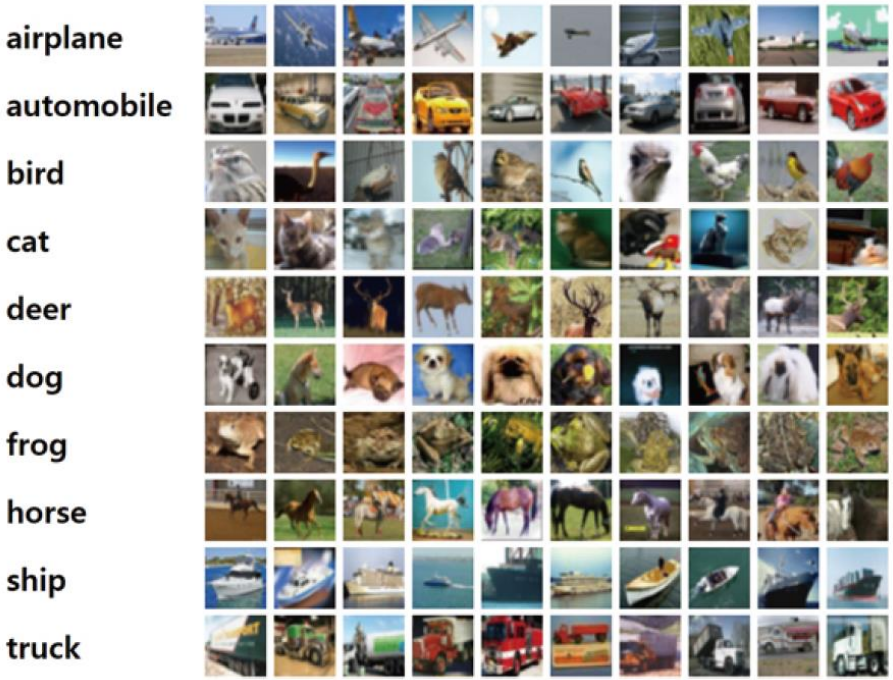


그림 6-14 CIFAR-10 데이터셋

6.4.4 자연 영상 인식

- [프로그램 6-4(a)]는 CIFAR-10을 인식하는 컨볼루션 신경망
  - MNIST를 인식하는 [프로그램 6-2]와 다른 곳은 데이터셋 읽는 부분 뿐
    - CIFAR-10은 텐서 구조가 이미 (32*32*3)이어서 reshape 적용할 필요 없음

Colab에서 실행

프로그램 6-4(a) CNN으로 CIFAR-10 인식

```
01  import numpy as np
02  import tensorflow as tf
03  from tensorflow.keras.datasets import cifar10
04  from tensorflow.keras.models import Sequential
05  from tensorflow.keras.layers import Conv2D,MaxPooling2D,Flatten,Dense,Dropout
06  from tensorflow.keras.optimizers import Adam
07
08  # CIFAR-10 데이터셋을 읽고 신경망에 입력할 형태로 변환
09  (x_train,y_train),(x_test,y_test)=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)
14
```

CIFAR-10은 읽은 데이터가 (32,32,3) 텐서로 표현되므로 [프로그램 6-2]와 달리 reshape 적용할 필요가 없음

6.4.4 자연 영상 인식

```
15  # 신경망 모델 설계
16  cnn=Sequential()
17  cnn.add(Conv2D(32,(3,3),activation='relu',input_shape=(32,32,3)))
18  cnn.add(Conv2D(32,(3,3),activation='relu'))
19  cnn.add(MaxPooling2D(pool_size=(2,2)))
20  cnn.add(Dropout(0.25))
21  cnn.add(Conv2D(64,(3,3),activation='relu'))
22  cnn.add(Conv2D(64,(3,3),activation='relu'))
23  cnn.add(MaxPooling2D(pool_size=(2,2)))
24  cnn.add(Dropout(0.25))
25  cnn.add(Flatten())
26  cnn.add(Dense(512,activation='relu'))
27  cnn.add(Dropout(0.5))
28  cnn.add(Dense(10,activation='softmax'))
29
30  # 신경망 모델 학습
31  cnn.compile(loss='categorical_crossentropy',optimizer=Adam(),metrics=['accuracy'])
32  hist=cnn.fit(x_train,y_train,batch_size=128,epochs=30,validation_data=(x_test,
    y_test),verbose=2)
33
34  # 신경망 모델 정확률 평가
35  res=cnn.evaluate(x_test,y_test,verbose=0)
36  print("정확률은",res[1]*100)
```

6.4.4 자연 영상 인식

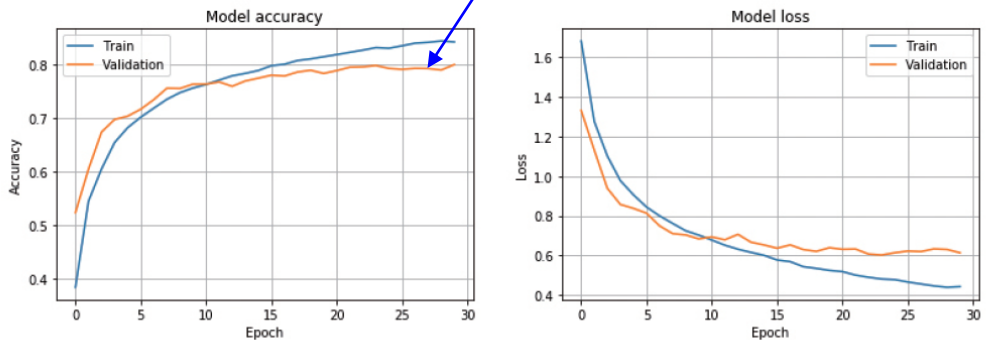
```
37
38  import matplotlib.pyplot as plt
39
40  # 정확률 그래프
41  plt.plot(hist.history['accuracy'])
42  plt.plot(hist.history['val_accuracy'])
43  plt.title('Model accuracy')
44  plt.ylabel('Accuracy')
45  plt.xlabel('Epoch')
46  plt.legend(['Train','Validation'], loc='best')
47  plt.grid()
48  plt.show()
49
50  # 손실 함수 그래프
51  plt.plot(hist.history['loss'])
52  plt.plot(hist.history['val_loss'])
53  plt.title('Model loss')
54  plt.ylabel('Loss')
55  plt.xlabel('Epoch')
56  plt.legend(['Train','Validation'],loc='best')
57  plt.grid()
58  plt.show()
```

6.4.4 자연 영상 인식

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/30
50000/50000 - 126s - loss: 1.6821 - accuracy: 0.3850 - val_loss: 1.3316 - val_
accuracy: 0.5239
Epoch 2/30
50000/50000 - 110s - loss: 1.2766 - accuracy: 0.5451 - val_loss: 1.1318 - val_
accuracy: 0.6046
Epoch 3/30
50000/50000 - 103s - loss: 1.1022 - accuracy: 0.6050 - val_loss: 0.9390 - val_
accuracy: 0.6736
…
Epoch 28/30
50000/50000 - 103s - loss: 0.4458 - accuracy: 0.8403 - val_loss: 0.6328 - val_
accuracy: 0.7916
Epoch 29/30
50000/50000 - 101s - loss: 0.4391 - accuracy: 0.8427 - val_loss: 0.6300 - val_
accuracy: 0.7889
Epoch 30/30
50000/50000 - 100s - loss: 0.4424 - accuracy: 0.8411 - val_loss: 0.6132 - val_
accuracy: 0.7989

정확률은 79.89000082015991
```

79.89% 정확률
세대수를 늘리면 추가 성능 향상 여지



6.4.5 학습된 모델 저장과 재활용

- 학습된 모델 저장([프로그램 6-4(b)])
  - 학습에 많은 시간이 걸리므로 저장해 두고 필요할 때 불러 쓸 필요성
  - save 함수로 간단히 달성(신경망의 구조 정보와 가중치 저장해 줌. HDF5 파일 형식 사용)

프로그램 6-4(b) CNN으로 CIFAR-10 인식: 모델 저장

```
59    cnn.save("my_cnn.h5")
```

6.4.5 학습된 모델 저장과 재활용

- 저장한 모델 불러오기([프로그램 6-5])

프로그램 6-5 예비 학습된 모델 불러다 쓰기

```
01  import numpy as np
02  import tensorflow as tf
03  from tensorflow.keras.datasets import cifar10
04
05  # 신경망 구조와 가중치를 저장하고 있는 파일을 읽어 옴
06  cnn=tf.keras.models.load_model("my_cnn.h5")
07  cnn.summary()
08
09  # CIFAR-10 데이터셋을 읽고 신경망에 입력할 형태로 변환
10  (x_train,y_train),(x_test,y_test)=cifar10.load_data()
11  x_train=x_train.astype(np.float32)/255.0
12  x_test=x_test.astype(np.float32)/255.0
13  y_train=tf.keras.utils.to_categorical(y_train,10)
14  y_test=tf.keras.utils.to_categorical(y_test,10)
15
16  res=cnn.evaluate(x_test,y_test,verbose=0)
17  print("정확률은",res[1]*100)
```

← load_model 함수로 저장해둔 모델 불러옴

← summary 함수로 불러온 모델 구조 확인

← 학습할 때와 같은 성능인지 확인

6.4.5 학습된 모델 저장과 재활용

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_16 (Conv2D)           (None, 30, 30, 32)        896
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 28, 28, 32)        9248
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
dropout_12 (Dropout)         (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 12, 12, 64)        18496
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 10, 10, 64)        36928
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
dropout_13 (Dropout)         (None, 5, 5, 64)          0
_________________________________________________________________
flatten_4 (Flatten)          (None, 1600)              0
_________________________________________________________________
dense_8 (Dense)              (None, 512)               819712
_________________________________________________________________
dropout_14 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_9 (Dense)              (None, 10)                5130
=================================================================
Total params: 890,410
Trainable params: 890,410
Non-trainable params: 0
_________________________________________________________________

 정확률은 79.89000082015991
```

6.5 컨볼루션 신경망의 시각화

- 두 가지 시각화를 시도
  - 컨볼루션층에 있는 커널을 시각화
  - 컨볼루션층이나 풀링층이 추출해주는 특징 맵을 시각화

NOTE 설명 가능 인공지능 1

이 절에서 제시하는 커널의 시각화와 특징 맵의 시각화는 설명 가능 인공지능(XAI, eXplaining Artificial Intelligence)과 관련이 깊다. 딥러닝으로 구현한 인공지능은 성능이 뛰어나지만 왜 그런 의사결정을 했는지 설명하는 능력은 매우 뒤떨어진다. 방대한 수치 계산으로 의사결정이 이루어져서 의사결정에 대한 이유를 해석할 방법이 마땅히 없기 때문이다. 설명 가능을 달성하려는 많은 연구 결과가 있는데, 커널과 특징 맵의 시각화는 가장 낮은 수준의 방법으로 볼 수 있다. 낮은 수준이지만 쉽게 적용이 가능하고 많은 정보를 주기 때문에 애용하는 방법이다. 본격적인 설명 가능 인공지능은 12장에서 설명한다.

6.5.1 커널의 시각화 프로그래밍

- [프로그램 6-6]의 36~57행
  - CIFAR-10으로 학습한 컨볼루션 신경망의 커널을 시각화

프로그램 6-6 커널과 특징 맵의 시각화: CIFAR-10을 인식하는 컨볼루션 신경망

```
01
…   [프로그램 6-4(a)]의 Sequential과 compile로 모델을 구축하고 fit로 학습하는 부분
32
33
34  cnn.summary()                                  # cnn 모델의 정보 출력
35
36  for layer in cnn.layers:                       ❶ 컨볼루션층의 커널을 시각화
37      if 'conv' in layer.name:
38          kernel,biases=layer.get_weights()
39          print(layer.name,kernel.shape)         # 커널의 텐서 모양을 출력
40
41  kernel,biases=cnn.layers[0].get_weights()      # 층 0의 커널 정보를 저장
42  minv,maxv=kernel.min(),kernel.max()
43  kernel=(kernel-minv)/(maxv-minv)
44  n_kernel=32
45
46  import matplotlib.pyplot as plt
47
48  plt.figure(figsize=(20,3))
49  plt.suptitle("Kernels of conv2d_4")
50  for i in range(n_kernel):                      # i번째 커널
51      f=kernel[:,:,:,i]
52      for j in range(3):                         # j번째 채널
53          plt.subplot(3,n_kernel,j*n_kernel+i+1)
54          plt.imshow(f[:,:,j],cmap='gray')
55          plt.xticks([]); plt.yticks([])
56          plt.title(str(i)+'_'+str(j))
57  plt.show()
```

이름에 conv가 포함된 층(컨볼루션층)에 대해 커널의 텐서 모양 출력

맨 앞에 있는 컨볼루션층의 커널 정보를 추출

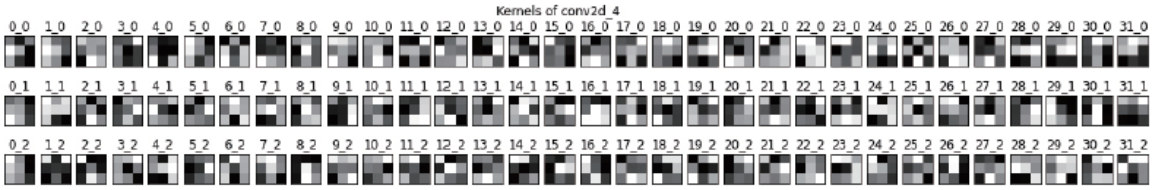
6.5.1 커널의 시각화 프로그래밍

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 30, 30, 32)        896
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 28, 28, 32)        9248
_________________________________________________________________
...
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5130
=================================================================
Total params: 890,410
Trainable params: 890,410
Non-trainable params: 0
_________________________________________________________________
```

❶

```
conv2d (3, 3, 3, 32)
conv2d_1 (3, 3, 32, 32)
conv2d_2 (3, 3, 32, 64)
conv2d_3 (3, 3, 64, 64)
```

← 36~39행의 실행 결과로 알아낸 컨볼루션층 정보



↖ 맨 앞에 있는 컨볼루션층의 커널 시각화

6.5.2 특징 맵의 시각화 프로그래밍

- [프로그램 6-6]의 59~80행
  - CIFAR-10으로 학습한 컨볼루션 신경망의 특징 맵을 시각화

이름에 conv가 포함된 층(컨볼루션층)에 대해 특징 맵의 텐서 모양 출력

```
58
59  for layer in cnn.layers:                          ❷ 컨볼루션층의 특징 맵을 시각화
60      if 'conv' in layer.name:
61          print(layer.name,layer.output.shape)    # 특징 맵의 텐서 모양을 출력
62
63  from tensorflow.keras.models import Model
64
65  partial_model=Model(inputs=cnn.inputs,outputs=cnn.layers[0].output) # 층 0만 떼어냄
66  partial_model.summary()
67
68  feature_map=partial_model.predict(x_test)       # 부분 모델로 테스트 집합을 예측
69  fm=feature_map[1]                               # 1번 영상의 특징 맵을 시각화
70
71  plt.imshow(x_test[1])                           # 1번 영상을 출력
72
73  plt.figure(figsize=(20,3))
74  plt.suptitle("Feature maps of conv2d_4")
75  for i in range(32):                             # i번째 특징 맵
76      plt.subplot(2,16,i+1)
77      plt.imshow(fm[:,:,i],cmap='gray')
78      plt.xticks([]); plt.yticks([])
79      plt.title("map"+str(i))
80  plt.show()
```

층 0만 떼내어 partial_model 객체에 저장 (층 0의 특징 맵을 가로채기 위해)

테스트 집합의 1번 영상에 대한 특징 맵을 시각화

6.5.2 특징 맵의 시각화 프로그래밍

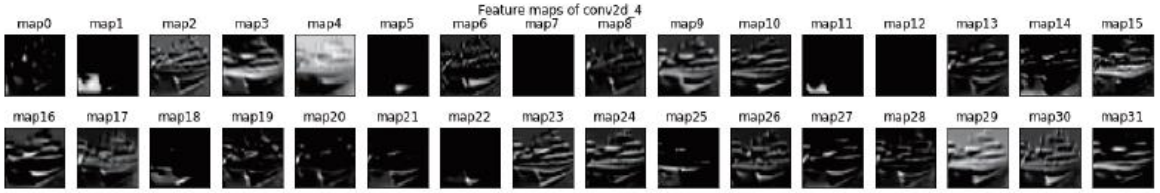
❷

```
conv2d (None, 30, 30, 32)
conv2d_1 (None, 28, 28, 32)
conv2d_2 (None, 12, 12, 64)
conv2d_3 (None, 10, 10, 64)

Model: "model_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_input (InputLayer)    [(None, 32, 32, 3)]       0
_________________________________________________________________
conv2d (Conv2D)              (None, 30, 30, 32)        896
=================================================================
Total params: 896
Trainable params: 896
Non-trainable params: 0
_________________________________________________________________
```

이름에 conv가 포함된 층(컨볼루션층)에 대해 커널의 텐서 모양 출력

맨 앞에 있는 컨볼루션층의 커널 정보를 추출



6.6 딥러닝의 규제

- 딥러닝의 전략
  - 충분히 큰 신경망 구조를 사용하되, 다양한 규제 기법을 적용하여 과잉 적합을 방지
  - 데이터 증대, 드롭아웃, 가중치 감쇠, 앙상블, 배치 정규화 등의 다양한 규제 기법이 있음

6.6.1 데이터 증대

- 과잉 적합을 방지하는 가장 확실한 방법은 큰 훈련 집합 사용
  - 대부분 상황에서 데이터를 늘리는 일은 많은 비용이 소요
  - 딥러닝에서는 주어진 데이터를 인위적으로 늘리는 데이터 증대data augmentation를 적용
    - 영상을 이동, 회전 또는 좌우 반전
    - 명암 조정 등
    - 텐서플로는 훌륭한 함수 제공

6.6.1 데이터 증대

- 증대된 영상 확인하기 [프로그램 6-7]

데이터 증대에 쓸
ImageDataGenerator 클래스를 불러옴

프로그램 6-7 ImageDataGenerator로 영상 데이터셋 증대

```
01  from tensorflow.keras.datasets import cifar10
02  from tensorflow.keras.preprocessing.image import ImageDataGenerator
03  import matplotlib.pyplot as plt
04
05  # CIFAR-10의 부류 이름
06  class_names=['airplane','automobile','bird','cat','deer','dog','flog','horse','ship','truck']
07
08  # CIFAR-10 데이터셋을 읽고 신경망에 입력할 형태로 변환
09  (x_train, y_train), (x_test, y_test)=cifar10.load_data()
10  x_train=x_train.astype('float32'); x_train/=255
11  x_train=x_train[0:12,]; y_train=y_train[0:12,]      # 앞 12개에 대해서만 증대 적용
12
13  # 앞 12개 영상을 그려줌
14  plt.figure(figsize=(16,2))
15  plt.suptitle("First 12 images in the train set")
16  for i in range(12):
17      plt.subplot(1,12,i+1)
18      plt.imshow(x_train[i])
19      plt.xticks([]); plt.yticks([])
20      plt.title(class_names[int(y_train[i])])
21
```

증대를 확인할 목적으로
앞 12개 영상만 취함

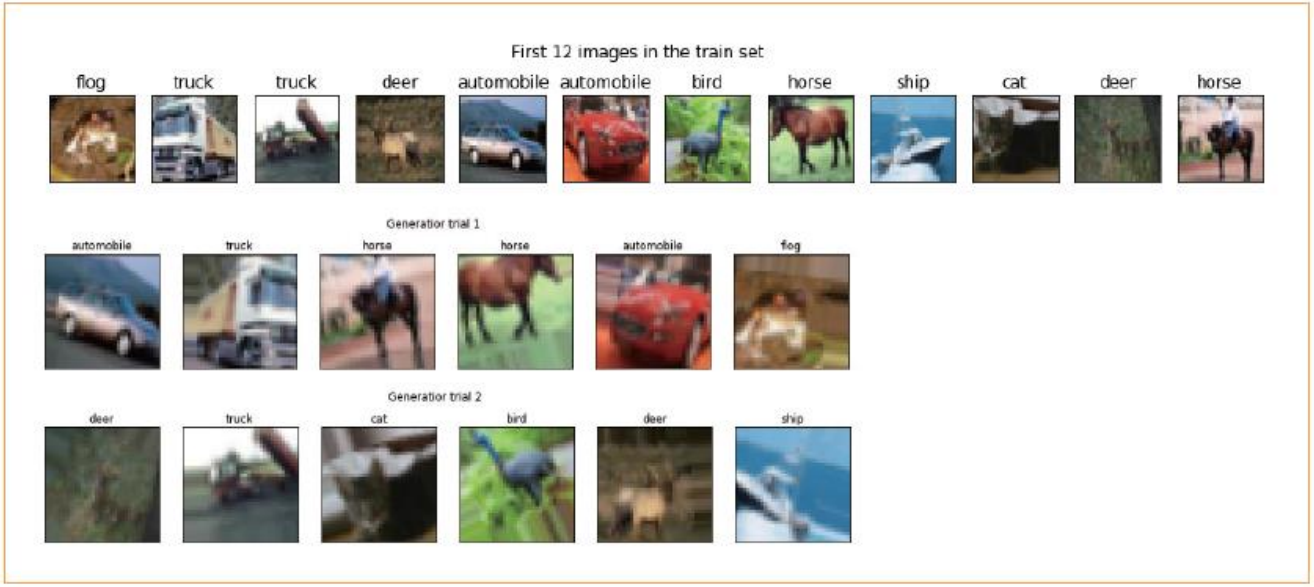
6.6.1 데이터 증대

```
22  # 영상 증대기 생성
23  batch_siz=6                                          # 한 번에 생성하는 양
24  generator=ImageDataGenerator(rotation_range=30.0,width_shift_
    range=0.2,height_shift_range=0.2,horizontal_flip=True)
25  gen=generator.flow(x_train,y_train,batch_size=batch_siz)
26
27  # 첫 번째 증대하고 그리기
28  img,label=gen.next()
29  plt.figure(figsize=(16,3))
30  plt.suptitle("Generatior trial 1")
31  for i in range(batch_siz):
32      plt.subplot(1,batch_siz,i+1)
33      plt.imshow(img[i])
34      plt.xticks([]); plt.yticks([])
35      plt.title(class_names[int(label[i])])
36
37  # 두 번째 증대하고 그리기
38  img,label=gen.next()
39  plt.figure(figsize=(16,3))
40  plt.suptitle("Generatior trial 2")
41  for i in range(batch_siz):
42      plt.subplot(1,batch_siz,i+1)
43      plt.imshow(img[i])
44      plt.xticks([]); plt.yticks([])
45      plt.title(class_names[int(label[i])])
```

변형 방식을 설정하여 객체 생성

Next 함수는 호출할 때마다
Batch_size 매개변수가 지정한 수만큼 영상 생성

6.6.1 데이터 증대



[ImageDataGenerator 함수의 API]

```
tensorflow.keras.preprocessing.image.ImageDataGenerator(featurewise_
center=False, samplewise_center=False, featurewise_std_
normalization=False, samplewise_std_normalization=False, zca_
whitening=False, zca_epsilon=1e-06, rotation_range=0, width_shift_
range=0.0, height_shift_range=0.0, brightness_range=None, shear_range=0.0,
zoom_range=0.0, channel_shift_range=0.0, fill_mode='nearest', cval=0.0,
horizontal_flip=False, vertical_flip=False, rescale=None, preprocessing_
function=None, data_format='channels_last', validation_split=0.0,
interpolation_order=1, dtype='float32')
```

다양한 변형을 위한
많은 종류의 매개변수

6.6.1 데이터 증대

- 증대된 영상으로 신경망 학습
  - 온라인과 오프라인 두 종류(현대 딥러닝은 온라인을 주로 사용)
    - 온라인은 학습을 진행하면서 실시간으로 샘플 생성
    - 오프라인은 한꺼번에 생성하여 디스크에 저장한 다음 사용
- [프로그램 6-8] CIFAR-10에 데이터 증대 적용

프로그램 6-8 CNN으로 CIFAR-10 인식: 데이터 증대 적용

```
01  import numpy as np
02  import tensorflow as tf
03  from tensorflow.keras.datasets import cifar10
04  from tensorflow.keras.models import Sequential
05  from tensorflow.keras.layers import Conv2D,MaxPooling2D,Flatten,Dense,Dropout
06  from tensorflow.keras.optimizers import Adam
07  from tensorflow.keras.preprocessing.image import ImageDataGenerator
08
09  # CIFAR-10 데이터셋을 읽고 신경망에 입력할 형태로 변환
10  (x_train, y_train), (x_test, y_test) = cifar10.load_data()
11  x_train=x_train.astype(np.float32)/255.0
12  x_test=x_test.astype(np.float32)/255.0
13  y_train=tf.keras.utils.to_categorical(y_train,10)
14  y_test=tf.keras.utils.to_categorical(y_test,10)
15
```

6.6.1 데이터 증대

```
16  # 신경망 모델 설계
17  cnn=Sequential()
18  cnn.add(Conv2D(32,(3,3),activation='relu',input_shape=(32,32,3)))
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(512,activation='relu'))
28  cnn.add(Dropout(0.5))
29  cnn.add(Dense(10,activation='softmax'))
30
31  # 신경망 모델 학습(영상 증대기 활용)
32  cnn.compile(loss='categorical_crossentropy',optimizer=Adam(),metrics=['accuracy'])
33  batch_siz=128
34  generator=ImageDataGenerator(width_shift_range=0.1,height_shift_
    range=0.1,horizontal_flip=True)
35  hist=cnn.fit_generator(generator.flow(x_train,y_train,batch_size=batch_
    siz),epochs=50,validation_data=(x_test,y_test),verbose=2)
36
37  # 신경망 모델 정확률 평가
38  res=cnn.evaluate(x_test,y_test,verbose=0)
39  print("정확률은",res[1]*100)
```

fit 대신 fit_generator 사용

6.6.1 데이터 증대

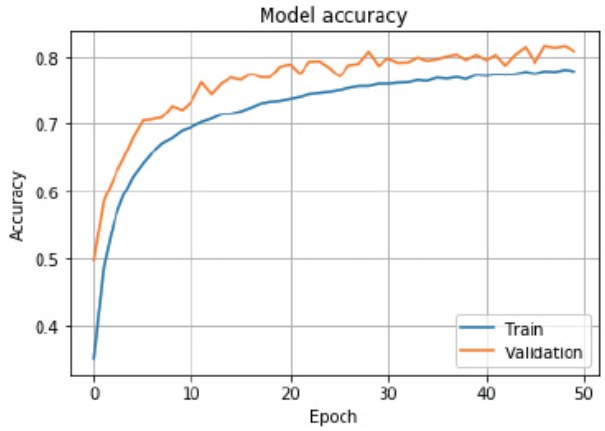
```
40
41  import matplotlib.pyplot as plt
42
43  # 정확률 그래프
44  plt.plot(hist.history['accuracy'])
45  plt.plot(hist.history['val_accuracy'])
46  plt.title('Model accuracy')
47  plt.ylabel('Accuracy')
48  plt.xlabel('Epoch')
49  plt.legend(['Train','Validation'], loc='best')
50  plt.grid()
51  plt.show()
52
53  # 손실 함수 그래프
54  plt.plot(hist.history['loss'])
55  plt.plot(hist.history['val_loss'])
56  plt.title('Model loss')
57  plt.ylabel('Loss')
58  plt.xlabel('Epoch')
59  plt.legend(['Train','Validation'], loc='best')
60  plt.grid()
61  plt.show()
```

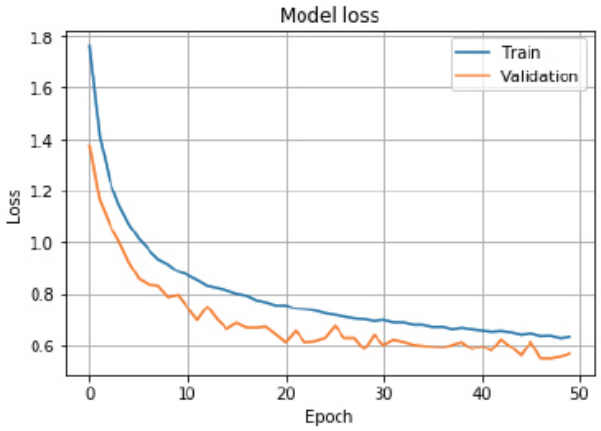
6.6.1 데이터 증대

```
Epoch 1/50
391/391 - 131s - loss: 1.7603 - accuracy: 0.3500 - val_loss: 1.3752 - val_accuracy: 0.4976
Epoch 2/50
391/391 - 133s - loss: 1.4144 - accuracy: 0.4882 - val_loss: 1.1636 - val_accuracy: 0.5858
...
Epoch 48/50
391/391 - 137s - loss: 0.6366 - accuracy: 0.7780 - val_loss: 0.5458 - val_accuracy: 0.8127
Epoch 49/50
391/391 - 138s - loss: 0.6276 - accuracy: 0.7809 - val_loss: 0.5531 - val_accuracy: 0.8154
Epoch 50/50
391/391 - 137s - loss: 0.6330 - accuracy: 0.7791 - val_loss: 0.5641 - val_accuracy: 0.8081

정확률은 80.80999851226807
```

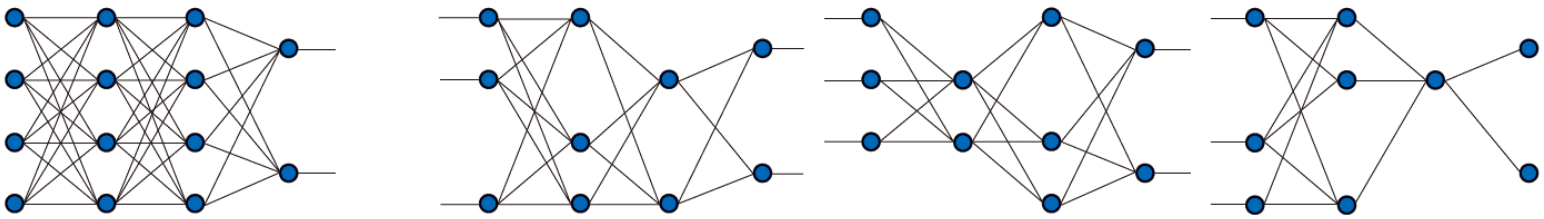
80.81% 정확률을 얻어 데이터 증대를 적용하지 않은 [프로그램 6-4]의 79.89%에 비해 0.92% 성능 향상





6.6.2 드롭아웃

- 드롭아웃dropout
  - 일정 비율의 가중치를 임의로 선택하여 불능으로 만들고 학습하는 규제 기법
  - 불능이 될 에지는 샘플마다 독립적으로 정하는데 난수를 이용하여 랜덤하게 선택
  - [프로그램 6-2]~[프로그램 6-8]에서 이미 적용함



(a) 원래 신경망(4-4-4-2 구조) (b) 드롭아웃된 3개의 신경망 예시

그림 6-15 드롭아웃의 원리

6.6.2 드롭아웃

- 드롭아웃의 성능 향상을 측정하는 [프로그램 6-9]
  - 교차 검증을 적용하여 실험 결과의 신뢰성을 높임([프로그램 6-4]에 [프로그램 5-12]의 교차 검증을 씌움)

프로그램 6-9 교차 검증으로 드롭아웃의 성능 향상 효과 측정

```
01  import numpy as np
02  import tensorflow as tf
03  from tensorflow.keras.datasets import cifar10
04  from tensorflow.keras.models import Sequential
05  from tensorflow.keras.layers import Conv2D,MaxPooling2D,Flatten,Dense,Dropout
06  from tensorflow.keras.optimizers import Adam
07  from tensorflow.keras.preprocessing.image import ImageDataGenerator
08  from sklearn.model_selection import KFold
09
10  # CIFAR-10 데이터셋을 읽고 신경망에 입력할 형태로 변환
11  (x_train, y_train), (x_test, y_test) = cifar10.load_data()
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
17  # 하이퍼 매개변수 설정
18  batch_siz=128
19  n_epoch=10
20  k=5     # k-겹 교차 검증
21
```

5-겹 교차 검증

6.6.2 드롭아웃

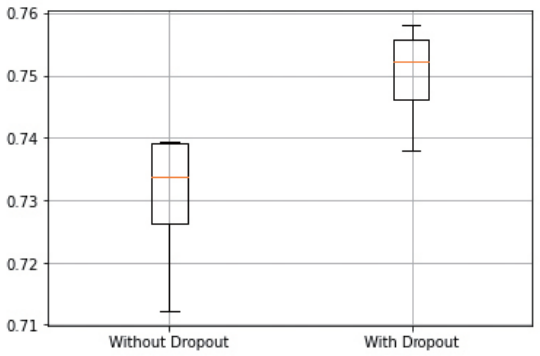
```
22  # 드롭아웃 비율에 따라 교차 검증을 수행하고 정확률을 반환하는 함수
23  def cross_validation(dropout_rate):
24      accuracy=[]
25      for train_index,val_index in KFold(k).split(x_train):
26        # 훈련 집합과 검증 집합으로 분할
27        xtrain,xval=x_train[train_index],x_train[val_index]
28        ytrain,yval=y_train[train_index],y_train[val_index]
29
30        # 신경망 모델 설계
31        cnn=Sequential()
32        cnn.add(Conv2D(32,(3,3),activation='relu',input_shape=(32,32,3)))
33        cnn.add(Conv2D(32,(3,3),activation='relu'))
34        cnn.add(MaxPooling2D(pool_size=(2,2)))
35        cnn.add(Dropout(dropout_rate[0]))
36        cnn.add(Conv2D(64,(3,3),activation='relu'))
37        cnn.add(Conv2D(64,(3,3),activation='relu'))
38        cnn.add(MaxPooling2D(pool_size=(2,2)))
39        cnn.add(Dropout(dropout_rate[1]))
40        cnn.add(Flatten())
41        cnn.add(Dense(512,activation='relu'))
42        cnn.add(Dropout(dropout_rate[2]))
43        cnn.add(Dense(10,activation='softmax'))
44
45        # 신경망 모델을 학습하고 평가하기
46        cnn.compile(loss='categorical_crossentropy',optimizer=Adam(),metrics=['a
          ccuracy'])
47        cnn.fit(xtrain,ytrain,batch_size=batch_siz,epochs=n_epoch,verbose=0)
48        accuracy.append(cnn.evaluate(xval,yval,verbose=0)[1])
49    return accuracy
```

6.6.2 드롭아웃

```
50
51  # 드롭아웃 비율을 달리하며 신경망을 평가
52  acc_without_dropout=cross_validation([0.0,0.0,0.0])
53  acc_with_dropout=cross_validation([0.25,0.25,0.5])
54
55  print("드롭아웃 적용 안 할 때:",np.array(acc_without_dropout).mean())
56  print("드롭아웃 적용할 때:",np.array(acc_with_dropout).mean())
57
58  import matplotlib.pyplot as plt
59
60  # 박스 플롯으로 정확률 표시
61  plt.grid()
62  plt.boxplot([acc_without_dropout,acc_with_dropout],labels=["Without Dropout","
    With Dropout"])
```

드롭아웃 비율을
다르게 하여 성능 비교

```
드롭아웃 적용 안 할 때: 0.73016
드롭아웃 적용할 때: 0.75003994
```



1.984% 성능 향상 (5-겹 교차 검증을
사용하여 높은 신뢰성)

6.6.3 가중치 감쇠

- 과잉 적합에서는 가중치 값이 아주 큰 현상이 나타남
- 가중치 감쇠$_{\text{weight decay}}$는 성능을 유지한 채로 가중치 크기를 낮추는 규제 기법

6.6.3 가중치 감쇠

- 표준 손실 함수(평균제곱오차 MSE)

$$J\left(\mathbf{U}^1,\mathbf{U}^2,\cdots,\mathbf{U}^L\right)=\frac{1}{|M|}\sum_{\mathbf{x}\in M}\|\mathbf{y}-\mathbf{o}\|^2 \qquad (6.4)$$

- 가중치 감쇠를 적용한 MSE

규제 항

L1 가중치 감쇠 또는 라소 규제

$$J\left(\mathbf{U}^1,\mathbf{U}^2,\cdots,\mathbf{U}^L\right)=\frac{1}{|M|}\sum_{\mathbf{x}\in M}\|\mathbf{y}-\mathbf{o}\|^2+\lambda|\mathbf{U}|$$

$$=\frac{1}{|M|}\sum_{\mathbf{x}\in M}\|\mathbf{y}-\mathbf{o}\|^2+\lambda\left(\left|u_{10}^1\right|+\left|u_{11}^1\right|+\cdots++\cdots+\left|u_{n_Ln_{L-1}}^L\right|\right) \qquad (6.5)$$

L2 가중치 감쇠 또는 리지 규제

$$J\left(\mathbf{U}^1,\mathbf{U}^2,\cdots,\mathbf{U}^L\right)=\frac{1}{|M|}\sum_{\mathbf{x}\in M}\|\mathbf{y}-\mathbf{o}\|^2+\lambda\|\mathbf{U}\|^2$$

$$=\frac{1}{|M|}\sum_{\mathbf{x}\in M}\|\mathbf{y}-\mathbf{o}\|^2+\lambda\left(\left(u_{10}^1\right)^2+\left(u_{11}^1\right)^2+\cdots+\left(u_{10}^2\right)^2+\cdots+\left(u_{n_Ln_{L-1}}^L\right)^2\right) \qquad (6.6)$$

6.6.3 가중치 감쇠

- 가중치 감쇠를 위한 텐서플로 프로그래밍
  - 세 가지 매개변수
    - Kernel_regularizer(가중치에 적용)
    - Bias_regularizer(바이어스에 적용)
    - Activity_regularizer(활성 함수 결과에 적용)
  - 적용 예

```
from tensorflow.keras import regularizers

model.add(Dense(64,input_dim=64,kernel_regularizer=regularizers.l2(0.01),
activity_regularizer=regularizers.l1(0.01)))
```

가중치에 L2, 활성 함수 결과에 L1 적용하라는 지시
(λ로 0.01 사용)

6.6.4 성능 평가 쟁점

- 성능 평가의 중요성
  - 하이퍼 매개변수 최적화나 모델 선택에서 중요
  - 현장 설치 여부를 정할 때 중요
- 객관적인 성능 평가 방법
  - 교차 검증
    - 데이터를 분할하여 여러 번 반복함으로써 통계적 신뢰성을 높임
  - 제거 조사ablation study
    - 여러 선택 사항이 있을 때, 선택 사항을 하나씩 빼고 성능을 측정해봄으로써 선택 사항 각각의 기여도를 평가

6.6.4 성능 평가 쟁점

- [프로그램 6-10]은 세 가지 옵션에 대해 제거 조사
  - 교차 검증으로 제거 조사 결과에 대한 신뢰성 높임

프로그램 6-10 교차 검증으로 제거 조사

```
01  import numpy as np
02  import tensorflow as tf
03  from tensorflow.keras.datasets import cifar10
04  from tensorflow.keras.models import Sequential
05  from tensorflow.keras.layers import Conv2D,MaxPooling2D,Flatten,Dense,Dropout
06  from tensorflow.keras.optimizers import Adam
07  from tensorflow.keras.preprocessing.image import ImageDataGenerator
08  from sklearn.model_selection import KFold
09  from tensorflow.keras import regularizers
10
11  # CIFAR-10 데이터셋을 읽고 신경망에 입력할 형태로 변환
12  (x_train,y_train),(x_test,y_test)=cifar10.load_data()
13  x_train=x_train.astype(np.float32)/255.0
14  x_test=x_test.astype(np.float32)/255.0
15  y_train=tf.keras.utils.to_categorical(y_train,10)
16  y_test=tf.keras.utils.to_categorical(y_test,10)
17
18  # 하이퍼 매개변수 설정
19  batch_siz=128
20  n_epoch=10
21  k=5 # k-folds
22
```

6.6.4 성능 평가 쟁점

```
# 하이퍼 매개변수에 따라 교차 검증을 수행하고 정확률을 반환하는 함수
def cross_validation(data_gen,dropout_rate,l2_reg):
    accuracy=[]
    for train_index,val_index in KFold(k).split(x_train):
        xtrain,xval=x_train[train_index],x_train[val_index]
        ytrain,yval=y_train[train_index],y_train[val_index]

        # 신경망 모델 설계
        cnn=Sequential()
        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(dropout_rate[0]))
        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(dropout_rate[1]))
        cnn.add(Flatten())
        cnn.add(Dense(512,activation='relu'))
        cnn.add(Dropout(dropout_rate[2]))
        cnn.add(Dense(10,activation='softmax',kernel_
        regularizer=regularizers.l2(l2_reg)))

        # 신경망을 학습하고 정확률 평가
        cnn.compile(loss='categorical_crossentropy',optimizer=Adam(),metrics=[
        'accuracy'])
        if data_gen:
            generator=ImageDataGenerator(rotation_range=3.0,width_shift_
            range=0.1,height_shift_range=0.1,horizontal_flip=True)
            cnn.fit_generator(generator.flow(x_train,y_train,batch_size=batch_
            siz),epochs=n_epoch,validation_data=(x_test,y_test),verbose=2)
        else:
            cnn.fit(xtrain,ytrain,batch_size=batch_siz,epochs=n_
            epoch, validation_data=(x_test,y_test),verbose=2)
        accuracy.append(cnn.evaluate(xval,yval,verbose=0)[1])
    return accuracy
```

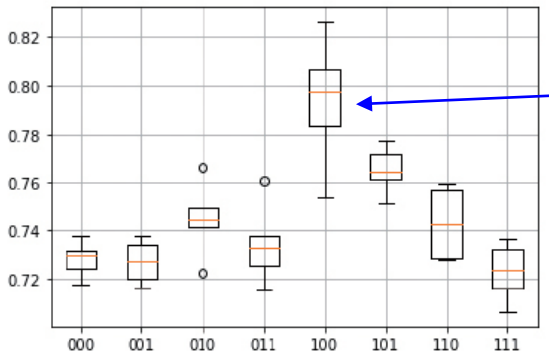
6.6.4 성능 평가 쟁점

```
54
55  # 하이퍼 매개변수를 달리 하며 신경망 모델을 평가
56  acc_000=cross_validation(False,[0.0,0.0,0.0],0.0)
57  acc_001=cross_validation(False,[0.0,0.0,0.0],0.01)
58  acc_010=cross_validation(False,[0.25,0.25,0.5],0.0)
59  acc_011=cross_validation(False,[0.25,0.25,0.5],0.01)
60  acc_100=cross_validation(True,[0.0,0.0,0.0],0.0)
61  acc_101=cross_validation(True,[0.0,0.0,0.0],0.01)
62  acc_110=cross_validation(True,[0.25,0.25,0.5],0.0)
63  acc_111=cross_validation(True,[0.25,0.25,0.5],0.01)
64
65  print("출력 형식: [Data augmentation-Dropout-l2 regularizer] (교차검증 시도/평균)")
66  print("[000] (",acc_000,"/",np.array(acc_000).mean(),")")
67  print("[001] (",acc_001,"/",np.array(acc_001).mean(),")")
68  print("[010] (",acc_010,"/",np.array(acc_010).mean(),")")
69  print("[011] (",acc_011,"/",np.array(acc_011).mean(),")")
70  print("[100] (",acc_100,"/",np.array(acc_100).mean(),")")
71  print("[101] (",acc_101,"/",np.array(acc_101).mean(),")")
72  print("[110] (",acc_110,"/",np.array(acc_110).mean(),")")
73  print("[111] (",acc_111,"/",np.array(acc_111).mean(),")")
74
75  import matplotlib.pyplot as plt
76
77  # 박스 플롯으로 정확률 표시
78  plt.grid()
79  plt.boxplot([acc_000,acc_001,acc_010,acc_011,acc_100,acc_101,acc_110,acc_111],
    labels=["000","001","010","011","100","101","110","111"])
```

6.6.4 성능 평가 쟁점

```
Train on 40000 samples, validate on 10000 samples
Epoch 1/10
40000/40000 - 35s - loss: 1.5782 - accuracy: 0.4252 - val_loss: 1.4161 - val_accuracy: 0.4958
Epoch 2/10
40000/40000 - 34s - loss: 1.1971 - accuracy: 0.5735 - val_loss: 1.1219 - val_accuracy: 0.6033
Epoch 3/10
40000/40000 - 35s - loss: 0.9914 - accuracy: 0.6525 - val_loss: 0.9464 - val_accuracy: 0.6696
...


출력 형식: [Data augmentation-Dropout-l2 regularizer] (교차검증 시도/평균)
[000] ( [0.7316, 0.7174, 0.7378, 0.7243, 0.73] / 0.72822 )
[001] ( [0.7338, 0.7167, 0.7375, 0.7274, 0.7202] / 0.72712004 )
[010] ( [0.7499, 0.7412, 0.7659, 0.7224, 0.7444] / 0.74476 )
[011] ( [0.7325, 0.7257, 0.7608, 0.7154, 0.7378] / 0.73443997 )
[100] ( [0.7972, 0.7838, 0.8257, 0.7539, 0.8064] / 0.79340005 )
[101] ( [0.7614, 0.7714, 0.7772, 0.7517, 0.764] / 0.76513994 )
[110] ( [0.7572, 0.7594, 0.7427, 0.7281, 0.7285] / 0.74318 )
[111] ( [0.716, 0.7062, 0.732, 0.7365, 0.7236] / 0.72286 )
```



데이터 증대를 적용하고 드롭아웃과 가중치 감쇠를 적용하지 않은 100 경우가 최대 성능

6.7 전이 학습

전이 학습의 원리

- 인간의 전이 학습
  - C 언어에 능숙한 학생은 파이썬을 금방 배움
  - 채소를 잘 가꾸는 사람은 꽃도 잘 가꿈

  ← 겉모습은 달라도 원천적인 개념과 사용법을 공유하기 때문
- 딥러닝에도 같은 원리가 적용됨
  - 자연 영상으로 학습한 신경망을 새나 개의 종을 분류하거나 자율주행에서 차선과 보행자 인식에 사용

전이 학습

- 어떤 도메인의 데이터로 학습한 모델을 다른 도메인의 데이터를 인식하는데 활용하여 성능 향상을 꾀하는 기법
- 데이터가 부족하여 스크래치 학습으로 높은 성능을 달성하기 어려운 상황에 주로 사용
- 예) cub200-2011은 200종의 새 영상을 가지는데 부류 별로 60장 가량에 불과하여 과잉적합 우려 → ImageNet으로 예비 학습$_{pre_trained}$된 모델을 cub200-2011로 전이하는 전이학습

6.7.1 현대적인 컨볼루션 신경망 모델

- 텐서플로가 제공하는 예비 학습 모델pre-trained model
  - 이들은 ImageNet으로 이미 학습되어 있는 컨볼루션 신경망

표 6-1 텐서플로가 제공하는 예비 학습된 컨볼루션 신경망 모델(출처: https://keras.io/applications)

모델 이름	파일 크기	1순위 정확률	5순위 정확률	매개변수 개수	층의 개수(깊이)
Xception	88MB	0.790	0.945	22,910,480	126
VGG16	528MB	0.713	0.901	138,357,544	23
VGG19	549MB	0.713	0.900	143,667,240	26
ResNet50	98MB	0.749	0.921	25,636,712	–
ResNet101	171MB	0.764	0.928	44,707,176	–
ResNet152	232MB	0.766	0.931	60,419,944	–
ResNet50V2	98MB	0.760	0.930	25,613,800	–
ResNet101V2	171MB	0.772	0.938	44,675,560	–
ResNet152V2	232MB	0.780	0.942	60,380,648	–
InceptionV3	92MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215MB	0.803	0.953	55,873,736	572
MobileNet	16MB	0.704	0.895	4,253,864	88
MobileNetV2	14MB	0.713	0.901	3,538,984	88
DenseNet121	33MB	0.750	0.923	8,062,504	121
DenseNet169	57MB	0.762	0.932	14,307,880	169
DenseNet201	80MB	0.773	0.936	20,242,984	201
NASNetMobile	23MB	0.744	0.919	5,326,716	–
NASNetLarge	343MB	0.825	0.960	88,949,818	–

6.7.1 현대적인 컨볼루션 신경망 모델

- 많이 쓰는 예비 학습 모델

  VGG, Inception, ResNet

  - VGG: 3*3의 작은 마스크 사용
  - Inception(GoogLeNet): 네크워크 속의 네트워크(NIN) 구현
  - ResNet: 잔류 학습

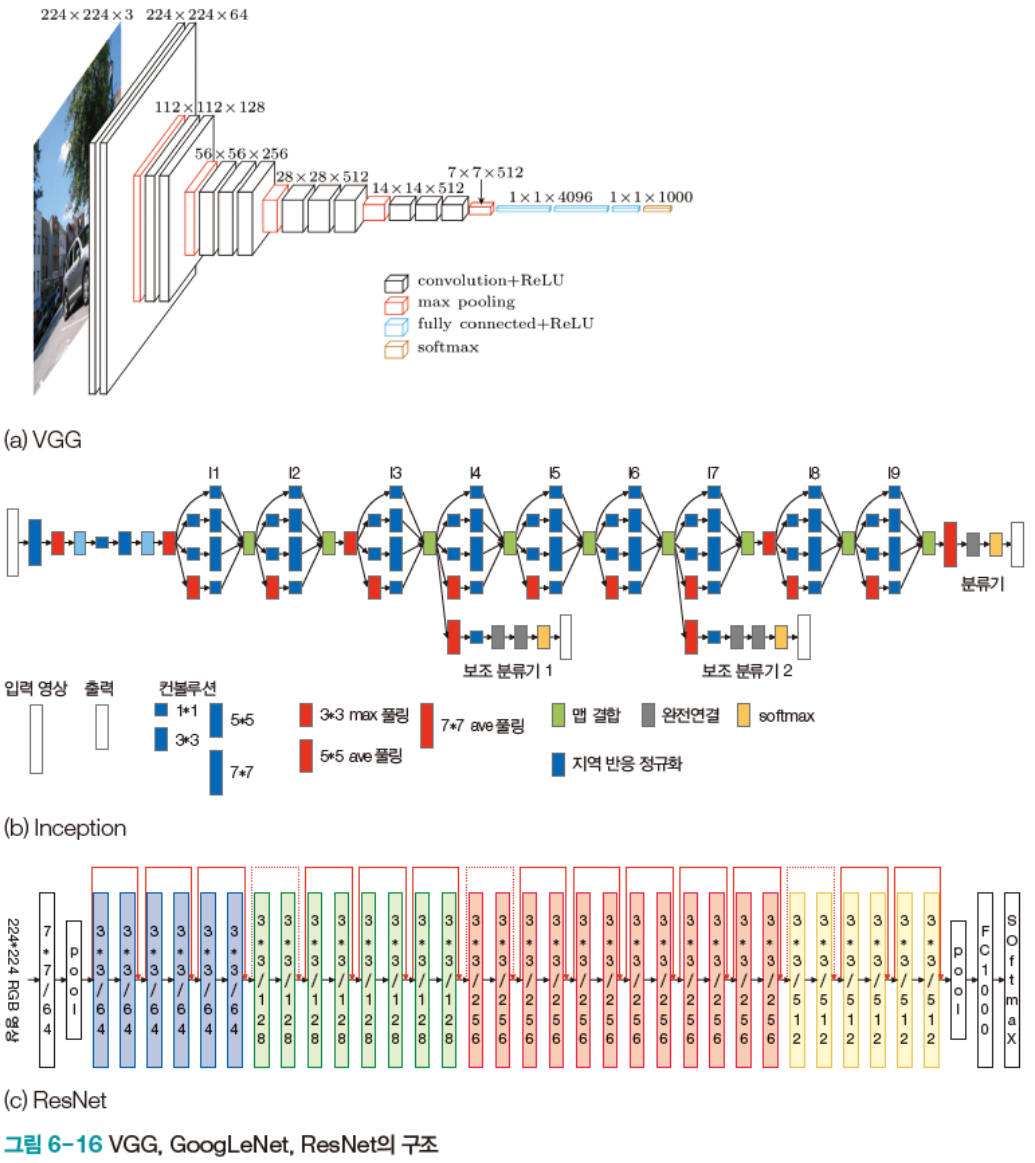


그림 6-16 VGG, GoogLeNet, ResNet의 구조

6.7.2 전이 학습 프로그래밍: 새의 품종 분류

- ImageNet으로 학습한 ResNet 모델을 새 품종 인식 문제로 전이
  - 새 품종 인식 데이터셋으로 cub200-2011을 사용(200종의 새 영상 11788장)
    - 미세 분류fine-grained classification 문제
    - 부류 내 변화within-class variation가 아주 크고 부류 간 유사도between-class similarity가 큼

그림 6-17 cub200-2011 데이터셋의 예제 영상

TIP Cub200-2011 데이터에 대한 성능 경쟁은 https://paperswithcode.com/sota/fine-grained-image-classification-on-cub-200에서 확인할 수 있다. 현재 90%를 약간 넘는 정확률을 확보했다. 이런 성능을 재현해보는 것은 이 책의 범위를 넘는다. 여기서는 스크래치 학습과 전이 학습의 성능을 비교하고 전이 학습의 유용성을 확인하는 데까지만 학습한다.

6.7.2 전이 학습 프로그래밍: 새의 품종 분류

- [프로그램 6-11]

프로그램 6-11 ImageNet으로 학습된 ResNet50을 cub 데이터셋으로 전이 학습

```
01  import numpy as np
02  import tensorflow as tf
03  from tensorflow.keras.models import Sequential
04  from tensorflow.keras.layers import Flatten,Dense
05  from tensorflow.keras.optimizers import Adam
06  from tensorflow.keras.applications.resnet50 import ResNet50,preprocess_input
07  from tensorflow.keras.preprocessing import image
08  import os
09
10  train_folder='CUB200/train'
11  test_folder='CUB200/test'
12
13  class_reduce=0.1      # 부류 수 줄여서 데이터양 줄임(속도와 메모리 효율을 위해)
14  no_class=int(len(os.listdir(train_folder))*class_reduce)  # 부류 개수
15
```

해당 폴더에 데이터를 미리 저장해두어야 함

메모리 문제 때문에 일부 데이터만 사용

6.7.2 전이 학습 프로그래밍: 새의 품종 분류

훈련 집합 읽어 옴

```
16  x_train,y_train=[],[]
17  for i,class_name in enumerate(os.listdir(train_folder)):
18      if i<no_class:   # 13~14행이 지정한 부류만 사용
19          for fname in os.listdir(train_folder+'/'+class_name):
20              img=image.load_img(train_folder+'/'+class_name+'/'+fname,target_
                size=(224,224))
21              if len(img.getbands())!=3:
22                  print("주의: 유효하지 않은 영상 발생",class_name,fname)
23                  continue
24              x=image.img_to_array(img)
25              x=preprocess_input(x)
26              x_train.append(x)
27              y_train.append(i)
28
29  x_test,y_test=[],[]
30  for i,class_name in enumerate(os.listdir(test_folder)):
31      if i<no_class:   # 13~14행이 지정한 부류만 사용
32          for fname in os.listdir(test_folder+'/'+class_name):
33              img=image.load_img(test_folder+'/'+class_name+'/'+fname,target_
                size=(224,224))
34              if len(img.getbands())!=3:
35                  print("주의: 유효하지 않은 영상 발생",class_name,fname)
36                  continue
37              x=image.img_to_array(img)
38              x=preprocess_input(x)
39              x_test.append(x)
40              y_test.append(i)
```

테스트 집합 읽어 옴

6.7.2 전이 학습 프로그래밍: 새의 품종 분류

```
41
42  x_train=np.asarray(x_train)
43  y_train=np.asarray(y_train)
44  x_test=np.asarray(x_test)
45  y_test=np.asarray(y_test)
46  y_train=tf.keras.utils.to_categorical(y_train,no_class)
47  y_test=tf.keras.utils.to_categorical(y_test,no_class)
48
49  base_model=ResNet50(weights='imagenet',include_top=False,input_shape=(224,224,3))
50  cnn=Sequential()
51  cnn.add(base_model)
52  cnn.add(Flatten())
53  cnn.add(Dense(1024,activation='relu'))
54  cnn.add(Dense(no_class,activation='softmax'))
55
56  cnn.compile(loss='categorical_crossentropy',optimizer=Adam(0.00002),metrics=
    ['accuracy'])
57  hist=cnn.fit(x_train,y_train,batch_size=16,epochs=10,validation_data=(x_test,
    y_test),verbose=1)
58
59  res=cnn.evaluate(x_test,y_test,verbose=0)
60  print("정확률은",res[1]*100)
```

49행: ResNet에서 특징 추출 부분 설정
include_top=False는 모델의 뒷부분,
즉 [그림 6-16(c)]에서 FC1000과
softmax 층을 떼내라는 뜻
([그림 6-13]에서 특징 추출 부분만 남겨주는 셈)

뒷부분을 새로 부착

미세 조정 방식의 학습(낮은 학습률 설정)

6.7.2 전이 학습 프로그래밍: 새의 품종 분류

```
Train on 600 samples, validate on 515 samples
Epoch 1/10
600/600 [==============================] - 416s 693ms/sample - loss: 4.4823 - accuracy:
0.4000 - val_loss: 1.1837 - val_accuracy: 0.6757
...
Epoch 10/10
600/600 [==============================] - 361s 601ms/sample - loss: 2.1636e-04 -
accuracy: 1.0000 - val_loss: 0.8966 - val_accuracy: 0.7553

정확률은 75.53398013114929
```

75.53% 정확률을 얻음

6.7.2 전이 학습 프로그래밍: 새의 품종 분류

미세 조정 방식과 동결 방식

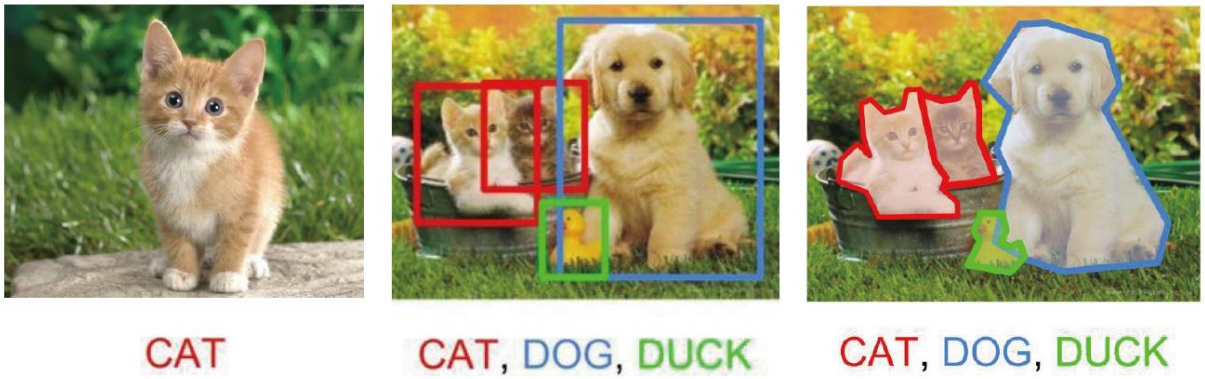
- 미세 조정fine-tuning 방식
  - 컨볼루션층과 완전연결층의 가중치를 동시에 수정
  - 학습률을 낮게 유지하여 조금씩 수정
- 동결 방식
  - 컨볼루션층의 가중치를 동결하여 수정이 일어나지 않게 제한
  - 49행 뒤에 다음 행을 추가하면 됨

```
base_model.trainable=False
```

- CUB 데이터셋에 대해 실험한 결과 미세 조정 방식이 우수함

6.8 물체 검출

- 분류classification, 검출detection, 분할segmentation 문제



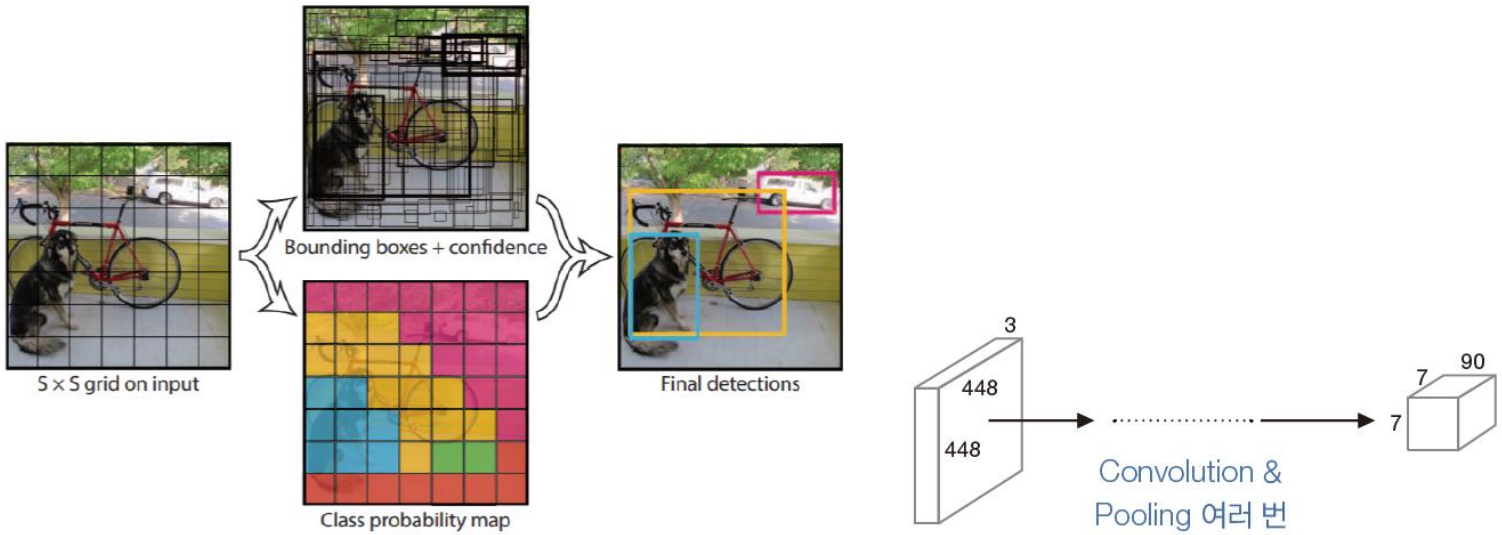
(a) 분류 (b) 검출 (c) 분할

그림 6-18 컴퓨터 비전의 세 가지 문제

- 물체 검출을 위한 딥러닝 모델
  - R-CNN
  - Fast R-CNN
  - Faster R-CNN
  - YOLO(you Only Look Once): 이 책이 사용하는 모델

6.8.1 욜로를 이용한 물체 검출

- [그림 6-19(a)]는 욜로의 동작 원리를 설명
  - 입력 영상을 S*S 격자로 나눈 S^2개의 격자 방 각각에 대해 B개의 바운딩 박스를 생성(바운딩 박스는 (x,y,w,h,o)의 5개 값으로 표현. o는 물체일 가능성)
  - C=80개의 물체 부류
  - 실제 구현에서는 S=7, B=2 → 49개 격자방 각각은 부류 정보를 표현하는 80-차원 벡터와 바운딩 박스 두개를 표현하는 10-차원 벡터를 가짐



(a) 처리 절차

(b) 컨볼루션 신경망의 구조

그림 6-19 욜로의 원리(출처: [Redmon2016])

6.8.1 욜로를 이용한 물체 검출

- [그림 6-20]은 개선된 YOLOv3
  - 14*14, 28*28, 56*56dml 3개 스케일을 사용
  - 각각 yolo_82, yolo_94라는 중간층과 yolo_106이라는 마지막 층에서 추출 가능

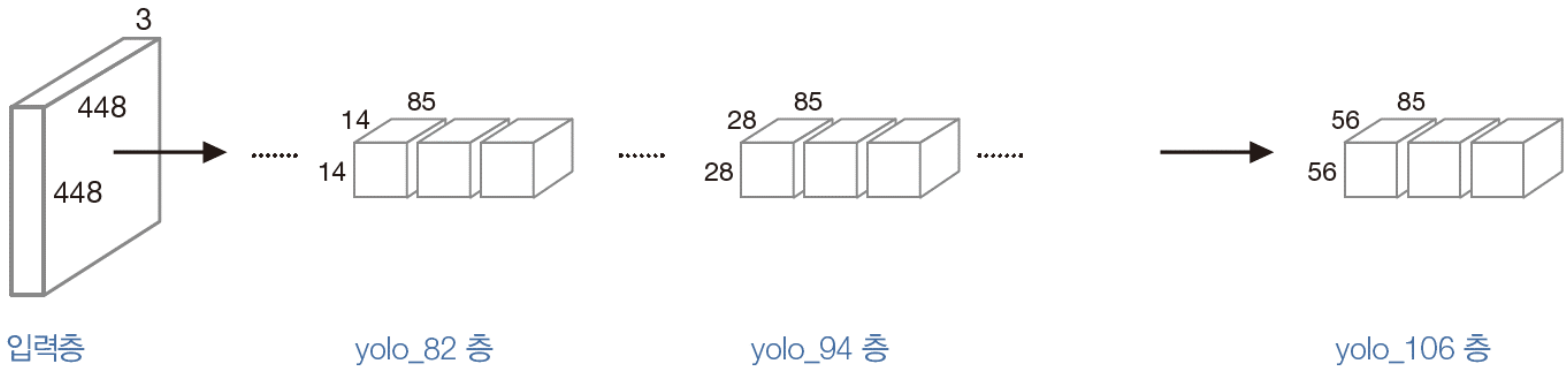


그림 6-20 YOLOv3의 신경망 구조

6.8.2 욜로 프로그래밍

- [프로그램 6-12]

TIP Opencv는 딥러닝 기술이 강세를 보이기 이전부터 널리 활용된 컴퓨터 비전 라이브러리로 2000년에 인텔에서 오픈소스로 공개했다. 주로 빠른 실행을 위해 C와 C++로 코딩되어 있다. Opencv에 Python API를 붙여 파이썬 라이브러리로 제공된다. 공식 홈페이지는 https://opencv.org이다.

프로그램 6-12 YOLOv3을 이용한 물체 검출

```
01  import numpy as np
02  import cv2
03
04  classes = []
05  f=open('coco.names.txt', 'r')
06  classes=[line.strip() for line in f.readlines()]
07  colors=np.random.uniform(0,255,size=(len(classes),3))
08
09  img=cv2.imread('yolo_test.jpg')
10  height,width,channels=img.shape
11  blob=cv2.dnn.blobFromImage(img,1.0/256,(448,448),(0,0,0),swapRB=True,crop=False)
12
13  yolo_model=cv2.dnn.readNet('./yolov3.weights','./yolov3.cfg')
14  layer_names=yolo_model.getLayerNames()
15  out_layers=[layer_names[i[0]-1] for i in yolo_model.getUnconnectedOutLayers()]
16
17  yolo_model.setInput(blob)
18  output3=yolo_model.forward(out_layers)
19
```

욜로는 MS COCO 데이터셋에 대해 학습되었는데, 80개의 부류 이름을 읽어 옴

09~11행: 테스트할 영상 읽고, 전처리 수행

YOLOv3 모델을 불러옴

yolo_82, yolo_94, yolo_106층을 알아냄 ([그림 6-20] 참조)

테스트 영상을 신경망에 입력

출력을 output3 객체에 저장

6.8.2 욜로 프로그래밍

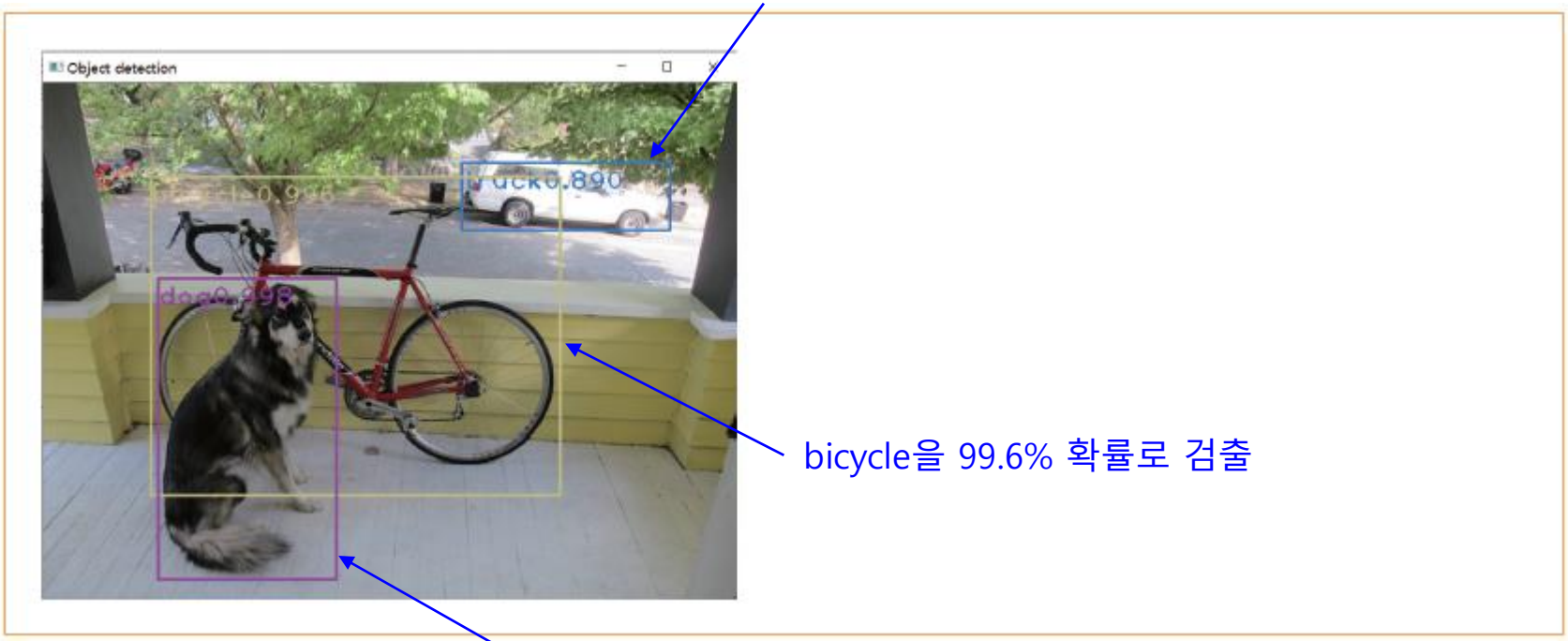
```
20  class_ids,confidences,boxes=[],[],[]
21  for output in output3:
22      for vec85 in output:
23          scores=vec85[5:]
24          class_id=np.argmax(scores)
25          confidence=scores[class_id]
26          if confidence>0.5:          # 신뢰도가 50% 이상인 경우만 취함
27              centerx,centery=int(vec85[0]*width),int(vec85[1]*height)
                                        # [0,1] 표현을 영상 크기로 변환
28              w,h=int(vec85[2]*width),int(vec85[3]*height)
29              x,y=int(centerx-w/2),int(centery-h/2)
30              boxes.append([x,y,w,h])
31              confidences.append(float(confidence))
32              class_ids.append(class_id)
33
34  indexes=cv2.dnn.NMSBoxes(boxes,confidences,0.5,0.4)
35
36  for i in range(len(boxes)):
37      if i in indexes:
38          x,y,w,h=boxes[i]
39          text=str(classes[class_ids[i]])+'%.3f'%confidences[i]
40          cv2.rectangle(img,(x,y),(x+w,y+h),colors[class_ids[i]],2)
41          cv2.putText(img,text,(x,y+30),cv2.FONT_HERSHEY_PLAIN,2,colors[class_
            ids[i]],2)
42
43  cv2.imshow("Object detection", img)
44  cv2.waitKey(0)
45  cv2.destroyAllWindows()
```

20~32행: 최고 부류 확률이 0.5를 넘는 바운딩 박스를 모음

비최대 억제를 적용하여 주위에 비해 최대인 것만 남김

36~41행: 비최대 억제에서 살아남은 바운딩 박스를 영상에 표시

6.8.2 욜로 프로그래밍


truck을 89.0% 확률로 검출(car를 truck으로 틀리게 분류)
Object detection
bicycle을 99.6% 확률로 검출
dog를 99.8% 확률로 검출

6.8.2 욜로 프로그래밍

- 또 다른 영상에 적용(상당히 우수한 성능)

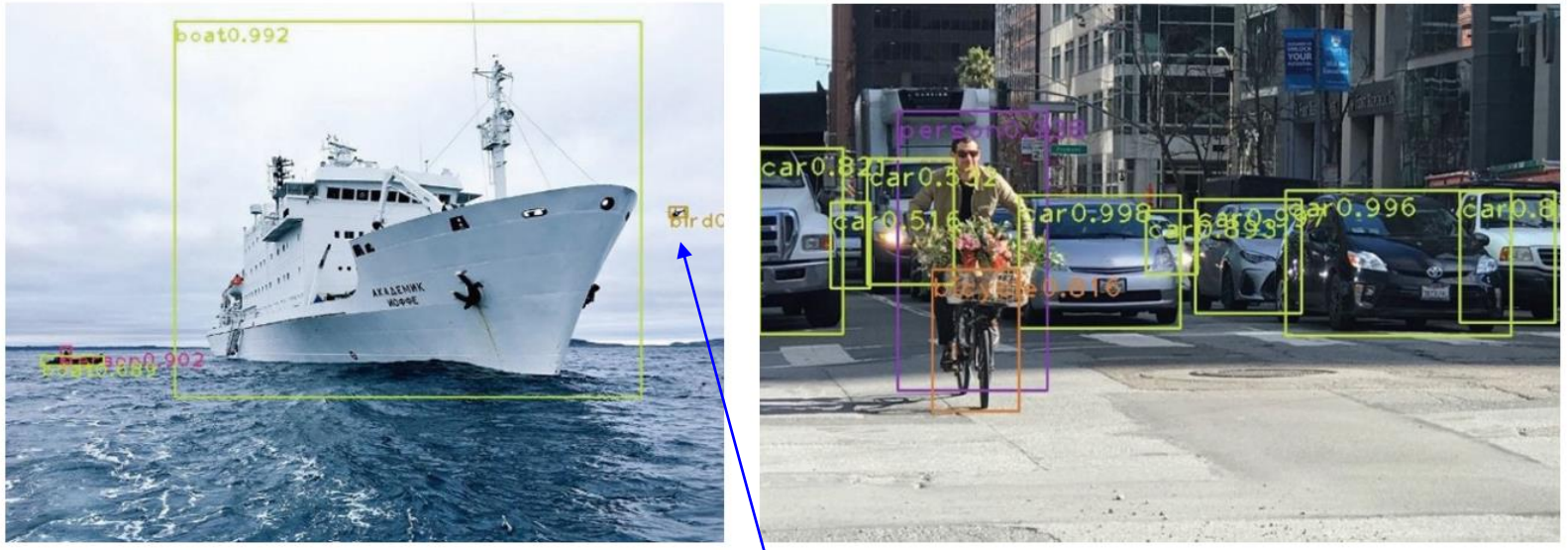


그림 6-21 YOLOv3를 이용한 물체 검출과 분류

helicopter를 bird로 틀리게 분류