4주차 chap08

Chap8. 컨볼루션 신경망

CNN : Convolutional Neural Network 합성곱 신경망

ex. 숫자 이미지를 학습시켜 무슨 숫자인지 맞춘다거나 여러 옷 이미지 중 어떤 카테고리에 속하는 의류인지 맞추는 모델 만들 수 있음

뉴런 = 필터 = 커널

8.1 발상과 전개

(a) 2차원 구조의 영상 (b) 1차원 구조로 펼친 경우

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

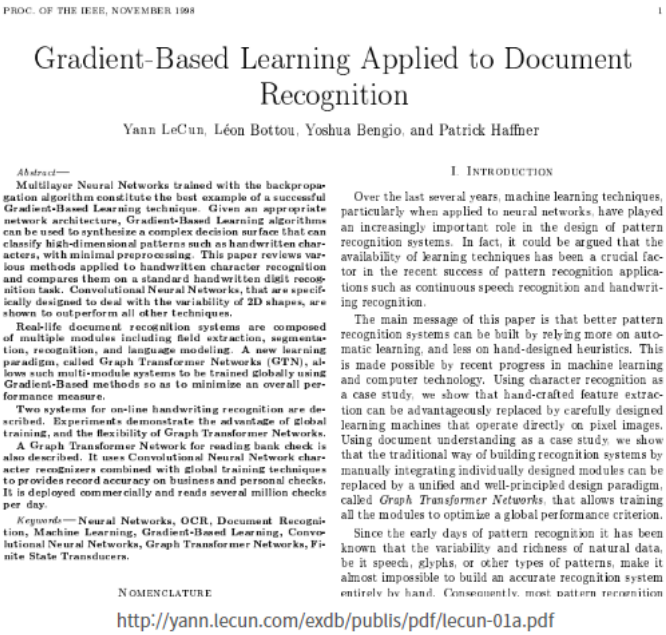
다층 퍼셉트론의 단점 : 화소의 연결성을 쓸 수 없어 개별 화소를 보고 분류할 수밖에 없어 큰 데이터셋일수록 정확도가 낮고 거의 인식하지 못함

=> 완전연결층이기때문

LeNet

https://dotiromoook.tistory.com/20

1998 르쿤이 제안한 컨볼루션 신경망



PROC. OF THE IEEE, NOVEMBER 1998 1

Gradient-Based Learning Applied to Document Recognition

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner

Abstract—
Multilayer Neural Networks trained with the backpropagation algorithm constitute the best example of a successful Gradient-Based Learning technique. Given an appropriate network architecture, Gradient-Based Learning algorithms can be used to synthesize a complex decision surface that can classify high-dimensional patterns such as handwritten characters, with minimal preprocessing. This paper reviews various methods applied to handwritten character recognition and compares them on a standard handwritten digit recognition task. Convolutional Neural Networks, that are specifically designed to deal with the variability of 2D shapes, are shown to outperform all other techniques.

Real-life document recognition systems are composed of multiple modules including field extraction, segmentation, recognition, and language modeling. A new learning paradigm, called Graph Transformer Networks (GTN), allows such multi-module systems to be trained globally using Gradient-Based methods so as to minimize an overall performance measure.

Two systems for on-line handwriting recognition are described. Experiments demonstrate the advantage of global training, and the flexibility of Graph Transformer Networks.

A Graph Transformer Network for reading bank check is also described. It uses Convolutional Neural Network character recognizers combined with global training techniques to provides record accuracy on business and personal checks. It is deployed commercially and reads several million checks per day.

Keywords—Neural Networks, OCR, Document Recognition, Machine Learning, Gradient-Based Learning, Convolutional Neural Networks, Graph Transformer Networks, Finite State Transducers.

I. Introduction

Over the last several years, machine learning techniques, particularly when applied to neural networks, have played an increasingly important role in the design of pattern recognition systems. In fact, it could be argued that the availability of learning techniques has been a crucial factor in the recent success of pattern recognition applications such as continuous speech recognition and handwriting recognition.

The main message of this paper is that better pattern recognition systems can be built by relying more on automatic learning, and less on hand-designed heuristics. This is made possible by recent progress in machine learning and computer technology. Using character recognition as a case study, we show that hand-crafted feature extraction can be advantageously replaced by carefully designed learning machines that operate directly on pixel images. Using document understanding as a case study, we show that the traditional way of building recognition systems by manually integrating individually designed modules can be replaced by a unified and well-principled design paradigm, called *Graph Transformer Networks*, that allows training all the modules to optimize a global performance criterion.

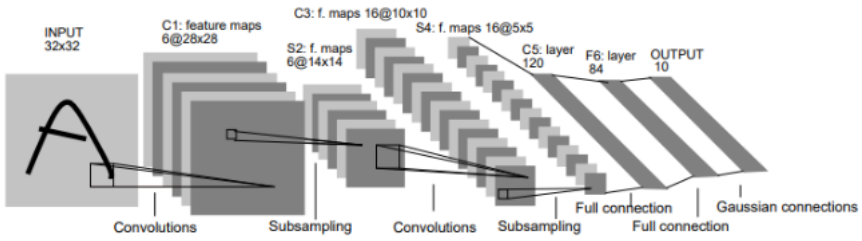
Since the early days of pattern recognition it has been known that the variability and richness of natural data, be it speech, glyphs, or other types of patterns, make it almost impossible to build an accurate recognition system entirely by hand. Consequently, most pattern recognition

Nomenclature

http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

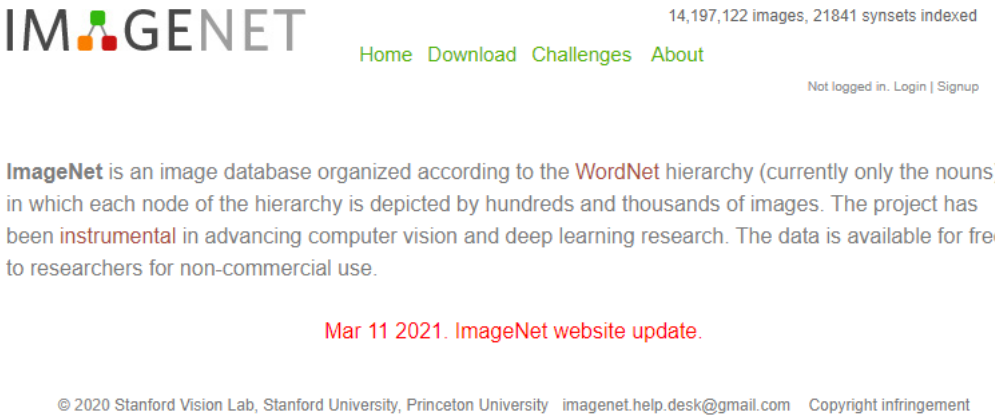
- LeNet-1 ~ LeNet-5까지 존재

- LeNet-5는 앞선 모델보다 입력 이미지의 크기가 커져 이전 대비 디테일한 부분까지 고려해 성능 높아짐
- ImageNet이라는 방대한 데이터셋 구축



LeNet 아키텍쳐

- 위의 구조를 이용해 수표에 적힌 필기문자를 인식하는 시스템 개발



8.2 컨볼루션 신경망의 구조

CNN : Convolution Layer, Pooling Layer(Sub Sampling), Fully Connected Layer 를 사용하여 사람의 시각처리방식을 모방한 딥러닝 학습 모델

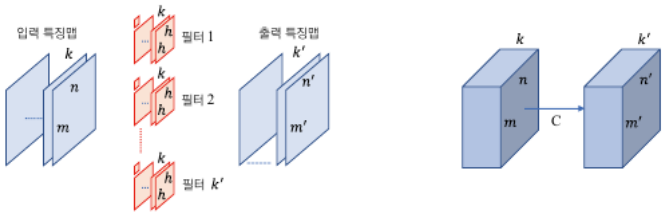
목표 : 데이터에 맞는 최적의 필터를 학습하는 컨볼루션층 설계

- convolution layer : 이미지의 특징점을 찾기 위해 사용
- pooling layer : 이미지 처리에 필요한 가중치와 연산량을 줄이기 위해 사용
- fully connected layer : 이미지 분류를 위해 사용

표준 컨볼루션 연산을 신경망에 적용하기 위해 확장 필요

1. 필터 3차원으로 확장 : 컬러영상은 3차원 구조의 텐서이기 때문
2. 많은 필터 배치 : 풍부한 특징을 추출하기 위함

컨볼루션층



(a) 세부 내용 (b) 간결한 블록 표현

그림 8-6 컨볼루션층

이미지를 분류classification하는 데 필요한 특징점들feteaures 추출
수 많은 필터가 존재하며 이 필터를 이용해 특징점 추출

<특성>

가중치 공유 weight sharing

- 필터의 값 = 가중치
- 입력 특징 맵의 모든 화소가 동일 필터를 사용해 가중치 공유

부분 연결성 partial connection

- 필터가 해당 화소 주위로 국한해 연산을 하기에 만족

=> 두가지 특성으로 학습 알고리즘이 최적화해야 할 가중치 개수를 획기적으로 줄임
=> 필터의 개수만큼 가중치를 추정 x, k'(kh^2+1)

- 컨볼루션층은 입력특징맵(k개 채널, mnk모양의 3차원 텐서)에 컨볼루션을 적용해 얻은 특징feature 맵 출력
- 필터는 하나의 바이어스 값 보유, 필터의 깊이는 입력 특징맵과 동일하게 k, 크기는 h*h

  이때, h = 3 or 5
  -> 팔터는 kh^2+1개의 가중치 보유
- 필터는 깊이 방향으로는 이동하지 않고 좌우 상하 방향으로 이동하며 곱의 합 = logit 을 구하는 컨볼루션 수행
- 필터 여러개 사용하여 풍부한 특징 추출

padding 덧대기

이미지 주변에 있는 중요한 정보를 잃어버리지 않도록 방지
입력 배열 주위를 가상의 원소로 채우는 작업

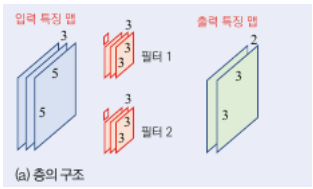
맵의 경계에 필터를 대면 일부 화소가 밖으로 나가기 때문에 적용 불가
맵의 경계를 제외하면 층이 깊어질수록 맵의 크기 점점 작아짐
-> 0덧대기 / 복사 덧대기 적용

- 0 덧대기 : 경계 바깥에 0을 덧댐
- 복사 덧대기 : 경계 화소의 값을 복사해 덧댐

stride 보폭

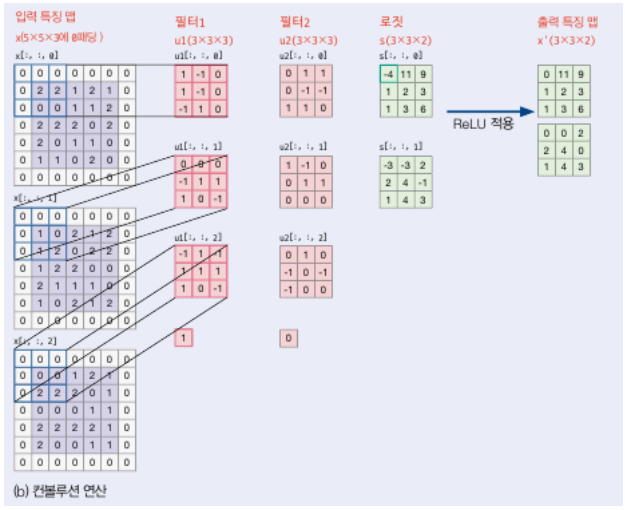
출력 맵의 크기를 줄이는 효과

보폭 s로 설정하면 s 화소씩 건너 필터 적용
보폭이 s면 mXn맵이 (m/s)(n/s)로 감소
ex. stride = 2라면, 2칸씩 이동하게 되면 커널 도장을 찍는 횟수가 줄어드는 만큼 만들어지는 특성 맵의 크기는 더 작아짐



(a) 층의 구조

(a) 0 padding, 2 stride 설정
5x5x3 입력특징 맵에 3x3x3 필터 2개 적용해 3x3x2 출력 특징맵 생성



+++ 제작과정 확인 : https://cs231n.github.io/convolutional-networks/

(b) 각각 필터를 적용해 구한 값(아래 식 확인)에 ReLU함수를 적용하면 출력 특징 맵 도출

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

연산량

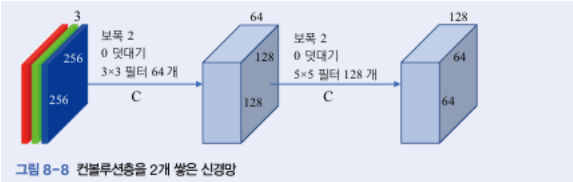


그림 8-8 컨볼루션층을 2개 쌓은 신경망

step1

- 입력 : 256x256크기의 RGB 컬러영상 = 256x256x3(깊이 k=3)
- 지정 : padding 0, stride 2, 3x3(h=3)filter(64)
- 연산량
  - 화소 하나 당 계산 : kh^2 + 1=3x3x3 + 1=28
  - 전체화소에 대한 필터 곱 : 화소개수 256x256, stride 2, 필터 개수 => (256/2) x (256/2) x 28 x 64
- 출력 : (256/2) x (256/2) x 64

step2

- 입력 : (128) x (128) x 64(깊이 k=64)
- 지정 : padding 0, stride 2, 5x5(h=5)filter(128)
- 연산량
  - 화소 하나 당 계산 : kh^2 + 1=5x5x64 + 1=1601

    - 전체화소에 대한 필터 곱 : 화소개수 128x128, stride 2, 필터 개수 => (128/2) x (128/2) x 1601 x 128
- 출력 : (128/2) x (128/2) x 128

풀링층 pooling layer

이미지 처리에 필요한 가중치와 연산량을 줄이기 위해 사용
상세함을 줄이는 효과와 특징 맵의 크기를 줄여 신경망의 효율을 높이는 효과 제공

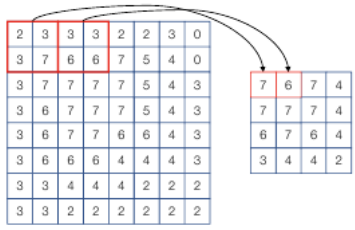


그림 8-9 풀링층(2×2 필터로 최대 풀링 적용, 보폭=2)

stride = 2 인 경우, max pooling(최대풀링 : 필터 안의 화소 중 최댓값을 취하는 연산)을 사용한다면 특징맵 반으로 준다

+) 평균 풀링 average pooling 도 존재

빌딩블록 쌓기 = convolution layer + pooling layer + Fully Connected layer

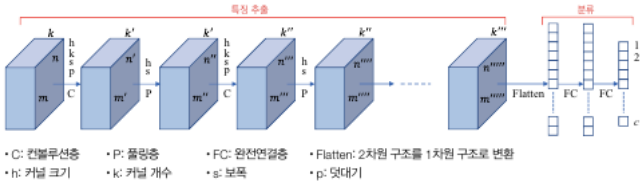
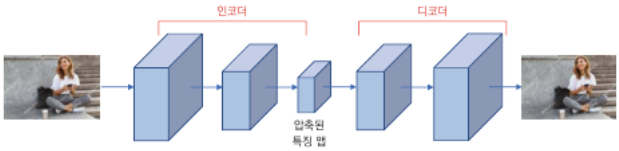


그림 8-10 컨볼루션층과 풀링층을 번갈아 쌓아 만드는 컨볼루션 신경망의 전형적인 구조

- convolution layer + pooling layer : 특징 추출 역할
- Flatten : 다차원 구조를 1차원 구조로 변환해 Fully Connected layer에 입력
- Fully Connected layer : 분류 수행

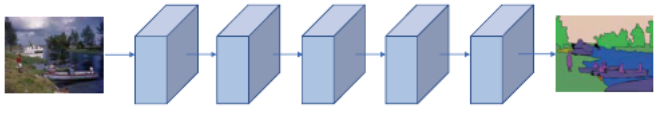
데이터에 따라 or 풀어야 하는 문제에 따라 다양한 모양으로 조립 가능

- case1) [Springenberg2014] 풀링층 모두 제거, only 컨볼루션층으로 쌓음



(a) 오토인코더

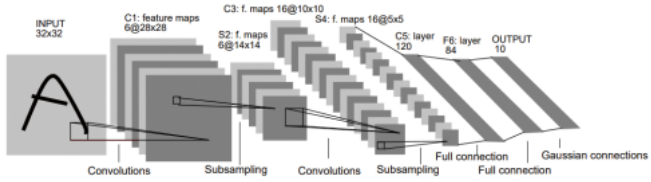
- case2) [Autoencoder] 입력영상을 그대로 출력영상으로 내놓는 컨볼루션 신경망
  encoder와 decoder로 구성되며 이는 대칭을 이루도록 설계
  인코더는 특징 맵을 점점 작게하고 디코더에서 다시 키워 원래 영상 복원



(b) 영상 분할을 위한 컨볼루션 신경망

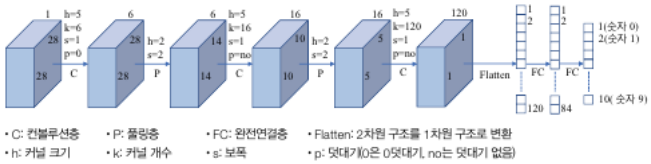
- case3) 완전 연결층 제거해 마지막 컨볼루션층에 레이블된 분할 맵을 배치
  영상분할을 이용한 컨볼루션 신경망

*Example : LeNet-5 **
LeNet-5 : 세계 최초로 실용화에 성공한 컨볼루션 신경망



(a) [LeCun1998]의 그림

- convolution : 컨볼루션층
- sub-sampling : 풀링층
- full-connection : 완전연결층
- output : 필기 숫자를 개발하였기에 출력층에 10개의 노드 존재
- 숫자맵은 명암이기에 입력 특징맵의 깊이는 1



(b) [그림 8-10] 표기에 따른 그림

그림 8-11 LeNet-5의 구조

- 입력 32x32
- C conv1 : padding 0, stride 1, 5x5(h=5)filter 6
- 결과1 : 28x28x6
- P pooling1
- 결과2 : 14x14x6

- 입력 14x14x6
- C conv2 : padding x, stride 1, 5x5(h=5)filter 16
- 결과3 : 10x10x16
- P pooling2
- 결과4 : 5x5x16

- 입력 5x5x16
- C conv3 : padding x, stride 1, 5x5(h=5)filter 120
- 결과5 : 1x1x120

- Flattern : 1차원 구조 변환
- FC 완전연결층 : 84개 노드
- 은닉층
- 출력층 : 10개 노드

8.3 컨볼루션 신경망의 학습

역전파 알고리즘

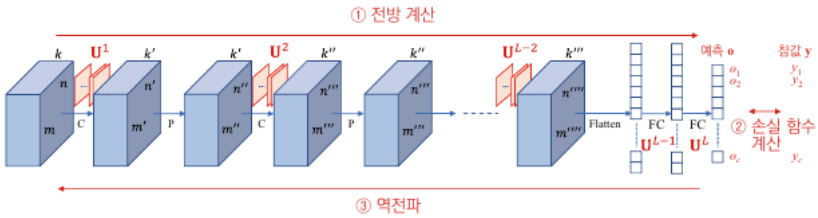


그림 8-13 컨볼루션 신경망을 학습하기 위한 역전파 알고리즘

(1) 전방계산 : 컨볼루션층과 풀링층, 완전연결층을 거쳐 벡터o 출력
(2) 손실계산 : 손실함수를 통해 벡터o와 참값 벡터 y의 오류를 계산
(3) 역전파 : 오류를 줄이는 방향으로 가중치 갱신

- 갱신 대상 : U^1, U^2, ... U^L-2(컨볼루션 층 위치) / U^L-1, U^L(완전연결 층 위치) => 학습대상이 되는 가중치 빨간색

가중치 행렬인 커널이 여러 번 사용되어 손실에 대한 편미분을 계산하기 까다로워 보일 수 있음
-> 컨볼루션 연산 형태로 결과가 나오기에 이해하거나 기억하기 쉬움
-> 단지 완전연결층과 계산이 다르기에 계산식에 따라 미분 다르게 수행

컨볼루션 신경망의 장점은 **특징추출 담당하는 필터 학습**
CNN이 feature learning한다고 표현

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

그림 8-15 딥러닝에 의한 컴퓨터 비전 방법론의 대전환

장점

1. 데이터의 원래 구조 유지
2. 특징학습을 통해 최적의 특징 추출
3. 신경망의 깊이 파악 가능

8.4 컨볼루션 신경망 구현

`cnn.add(Conv2D(6,(5,5),padding='same',activation='relu',input_shape=(28,28,1)))`
add함수와 Conv2D를 이용해 cnn 객체에 컨볼루션층 추가

- (1,2) 5x5 필터 6개
- (3) 0 패딩 <-> `padding='valid'` : 덧대기x
- 활성화함수
- 최초로 입력되는 텐서 모양 지정

`cnn.add(MaxPooling2D(pool_size=(2,2),strides=2))`

- (2,2) 필터
- 보폭2

```
# 8-1
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 # 컨볼루션층, 풀링층, Flattern연산에 사용할 클래스
from tensorflow.keras.optimizers import Adam # Adam 옵티마이저 사용

# 데이터 준비
(x_train,y_train),(x_test,y_test)=ds.mnist.load_data() # mnist 데이터셋 읽기
## 신경망에 입력할 형태로 변환 : 3차원 구조 변환
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)

# 모델선택 (신경망 구조 설계 : LeNet-5 신경망 구조 생성)
cnn=Sequential()
cnn.add(Conv2D(6,(5,5),padding='same',activation='relu',input_shape=(28,28,1))) # conv 층 추가
cnn.add(MaxPooling2D(pool_size=(2,2),strides=2)) # 풀링층 추가
cnn.add(Conv2D(16,(5,5),padding='valid',activation='relu')) # conv 층 추가
cnn.add(MaxPooling2D(pool_size=(2,2),strides=2)) # 풀링층 추가
cnn.add(Conv2D(120,(5,5),padding='valid',activation='relu')) # conv 층 추가
cnn.add(Flatten()) # Flatten연산: 텐서 1차원 구조 변환해 완전연결층 입력
cnn.add(Dense(units=84,activation='relu')) # 완전연결층 추가
cnn.add(Dense(units=10,activation='softmax')) # 안전 연결층추가

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

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

▼ 결과

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
Epoch 1/30
469/469 - 31s - loss: 0.3150 - accuracy: 0.9066 - val_loss: 0.0921 - val_accuracy: 0.9712 - 31s/epoch - 66ms/step
Epoch 2/30
469/469 - 30s - loss: 0.0848 - accuracy: 0.9744 - val_loss: 0.0570 - val_accuracy: 0.9805 - 30s/epoch - 65ms/step
Epoch 3/30
469/469 - 28s - loss: 0.0602 - accuracy: 0.9816 - val_loss: 0.0490 - val_accuracy: 0.9841 - 28s/epoch - 60ms/step
Epoch 4/30
469/469 - 31s - loss: 0.0464 - accuracy: 0.9858 - val_loss: 0.0411 - val_accuracy: 0.9855 - 31s/epoch - 66ms/step
Epoch 5/30
469/469 - 29s - loss: 0.0380 - accuracy: 0.9886 - val_loss: 0.0385 - val_accuracy: 0.9870 - 29s/epoch - 61ms/step
Epoch 6/30
469/469 - 30s - loss: 0.0338 - accuracy: 0.9893 - val_loss: 0.0372 - val_accuracy: 0.9869 - 30s/epoch - 64ms/step
Epoch 7/30
469/469 - 30s - loss: 0.0285 - accuracy: 0.9915 - val_loss: 0.0322 - val_accuracy: 0.9898 - 30s/epoch - 63ms/step
Epoch 8/30
469/469 - 27s - loss: 0.0265 - accuracy: 0.9915 - val_loss: 0.0347 - val_accuracy: 0.9895 - 27s/epoch - 59ms/step
Epoch 9/30
469/469 - 28s - loss: 0.0227 - accuracy: 0.9929 - val_loss: 0.0283 - val_accuracy: 0.9909 - 28s/epoch - 59ms/step
```

```
469/469 - 25s - loss: 0.0058 - accuracy: 0.9979 - val_loss: 0.0414 - val_accuracy: 0.9892 - 25s/epoch - 53ms/step
Epoch 26/30
469/469 - 24s - loss: 0.0059 - accuracy: 0.9983 - val_loss: 0.0338 - val_accuracy: 0.9917 - 24s/epoch - 52ms/step
Epoch 27/30
469/469 - 25s - loss: 0.0042 - accuracy: 0.9985 - val_loss: 0.0599 - val_accuracy: 0.9868 - 25s/epoch - 52ms/step
Epoch 28/30
469/469 - 25s - loss: 0.0069 - accuracy: 0.9976 - val_loss: 0.0367 - val_accuracy: 0.9904 - 25s/epoch - 52ms/step
Epoch 29/30
469/469 - 26s - loss: 0.0015 - accuracy: 0.9995 - val_loss: 0.0390 - val_accuracy: 0.9920 - 26s/epoch - 55ms/step
Epoch 30/30
469/469 - 24s - loss: 0.0047 - accuracy: 0.9985 - val_loss: 0.0456 - val_accuracy: 0.9904 - 24s/epoch - 52ms/step
정확률= 99.04000163078308
```

```
# 8-2
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() # cifar10 : 32*32*3 텐서로 reshape 불필요
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)

# 모델선택 (신경망 구조 설계)
cnn=Sequential()
cnn.add(Conv2D(32,(3,3),activation='relu',input_shape=(32,32,3))) # 컨볼루션층 쌓기
cnn.add(Conv2D(32,(3,3),activation='relu')) # 컨볼루션층 쌓기
cnn.add(MaxPooling2D(pool_size=(2,2))) # 풀링층 쌓기
cnn.add(Dropout(0.25)) # 드롭아웃층 = 규제기법으로 널리 쓰이는 연산
cnn.add(Conv2D(64,(3,3),activation='relu')) # 컨볼루션층 쌓기
cnn.add(Conv2D(64,(3,3),activation='relu')) # 컨볼루션층 쌓기
cnn.add(MaxPooling2D(pool_size=(2,2))) # 풀링층 쌓기
cnn.add(Dropout(0.25)) # 드롭아웃층
cnn.add(Flatten()) # Flatten연산
cnn.add(Dense(units=512,activation='relu')) # 완전연결층
cnn.add(Dropout(0.5)) # 드롭아웃층
cnn.add(Dense(units=10,activation='softmax')) # 완전연결층

# 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)

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

# 정확률과 손실함수 추세 그래프로 그리기
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()

# 과잉적합 방지
```

▼ 결과
과잉적합이 방지된 형태

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

텐서플로 모델 구축을 위해 필요한 모듈

models : 딥러닝 모델 구축을 위해 모델 생성할 때 사용
= 종류

- Sequential : 한 갈래 텐서가 끝까지 흐름
- Functional API : 중간에 여러 갈래로 나뉨

layers : 층 쌓을 때 사용
= 종류

- Dense 완전연결층
- Conv2D 컨볼루션층
- MaxPooling2D 최대풀링
- Flattern 특징맵 1차원 구조로 변경
- Dropout 드롭아웃 규제
  추가로 필터의 방향에 따라 구조가 항상 컨볼루션층을 결정할 필요 없음
  ex. 필터가 3차원 구조여도 conv2d적용 -> 컨볼루션 연산에서 필터가 깊이 방향으로 이동하지 않고 수평 수직 방향으로 이동하기 때문

losses : 손실함수 지정할 때 사용

- MSE Mean Squared Error
- 교차 엔트로피 cross entropy

optimizers : 옵티마이저 지정할 때 사용

- SGD
- Adam

8.5 [비전 에이전트6] 우편번호 인식기 v.2

좋은 하이퍼 매개변수 설정으로 정확률 개선

```
# 8-3
# 목적 : 비전 에이전트 5를 업그레이드하기 위해 높은 정확률과 필기 숫자 인식
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') # 학습 모델 저장해 비전 에이전트5를 업그레이드하는 데 쓸 수 있게 함

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

▼ 결과

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

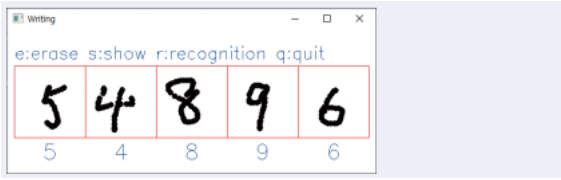
비전 에이전트 버전업

```
# 8-4 우편번호 인식기 v.2(CNN버전)
import numpy as np
import tensorflow as tf
import cv2 as cv
import matplotlib.pyplot as plt
import winsound

model=tf.keras.models.load_model('cnn_v2.h5') # 8-3에 저장해둔 모델 파일 읽기

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 컨볼루션 신경망의 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()
```

▼ 결과

한국인이 전제펜으로 화면에 쓴 데이터셋 수집해 다시 학습하면 정확률 높이기 가능



8.6 딥러닝의 학습 알고리즘 향상

손실함수

모델의 출력값과 사용자가 원하는 출력값의 오차 의미

→ 점수가 낮을수록 모델이 좋음

손실함수의 함수값이 최소화되도록 가중치와 편향을 찾는 것이 목표

교차엔트로피 *cross entropy*

두 확률분포가 얼마나 다른지 측정

신경망이 예측한 벡터 o와 실제 값 y를 확률 분포로 간주하여 둘의 다른 정도 예측

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

- c : 부류의 개수
- 손실함수로 사용할 때 미니 배치에 대해 평균 도출

[예시 8-3]

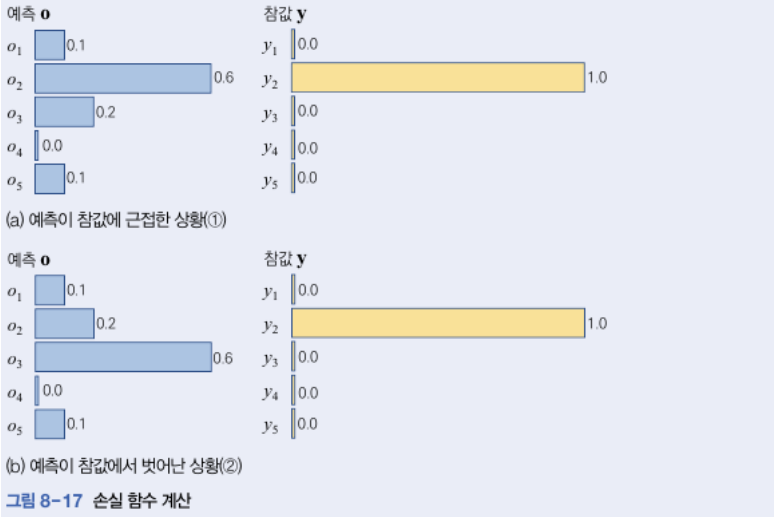


그림 8-17 손실 함수 계산

c=5인 경우이며 미니 배치 샘플을 하나만 가진다고 가정하고 손실함수 계

(a) o가 y에 근접해 신경망을 맞힌 상황

(a) o가 y에서 벗어나 신경망이 틀린 상황

▼ 결과

상황 1의 값이 더 낮게 나옴

이후, 만약 손실함수 값이 크다면 가중치를 갱신하는 양이 커져 학습 원만

평균제곱오차

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

교차 엔트로피

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

Focal 손실함수

교차엔트로피 단점 :

- 영역분할은 보통 물체가 점유한 영역보다 배경영역이 훨씬 넓어 부류 불균형 심함
- 손실함수 계산은 모든 화소의 e를 평균하기에 배경화소만 분류를 잘하면 물체 화소는 못하더라도 교차엔트로피 낮아짐

부류 불균형이 심한 경우에 사용

교차엔트로피에 gamma 와 가중치를 추가해 잘맞추는 클래스에 대한 loss 값을 줄여 갱신속도를 낮춰줌

- 감마 : 하이퍼 매개변수, 클수록 곡선이 에 더욱 빠르게 가까워짐
-

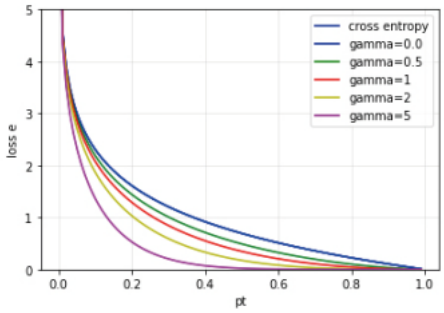


그림 8-18 부류가 2개인 경우의 교차 엔트로피와 Focal 손실 함수

- 1708.02002.pdf (arxiv.org)

$$\mathrm{CE}(p, y) = \begin{cases} -\log(p) & \text{if } y = 1 \\ -\log(1-p) & \text{otherwise.} \end{cases} \quad (1)$$

$$p_t = \begin{cases} p & \text{if } y = 1 \\ 1-p & \text{otherwise,} \end{cases} \quad (2)$$

$$\mathrm{FL}(p_t) = -(1-p_t)^{\gamma}\log(p_t). \quad (4)$$

옵티마이저

손실함수의 최저점인 오류가 작은 점을 찾아가는 최적화 문제를 푸는 표준알고리즘

SGD한계를 모멘텀과 적응적 학습률이라는 아이디어로 구현

모멘텀

신경망의 학습 안정성과 속도를 높여 학습이 잘 하도록 사용

모멘텀을 사용하면 가중치 값이 바뀌지 않고 어느 정도 일정한 방향을 유지하며 움직임

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

- 속도 V(운동량)를 갱신, V로 가중치 W 갱신
- 알파 : 오래된 운동량의 영향력을 줄여주는 인자

```
tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.0, nesterov=False, name='SGD'
```

모멘텀을 추가로 설정하지 않다면, 고적적인 SGD 적용

적응적 학습률

갱신 규칙 : 고정된 학습률 p 사용()

적응적 학습률을 사용해 매개변수마다 자신의 상황에 따라 학습률을 조절해 사용 가능

- AdaGrad(adaptive gradient)

- 각 매개변수 위치에서 그래디언트 도출

  → 매개변수 경사도 정도에 따라 학습률을 나눈 값을 각 매개변수의 학습률로 사용
- 이전 그래디언트를 누적한 정보를 이용해 학습률을 적응적으로 결정

```
tf.keras.optimizers.Adagrad(learning_rate=0.001, initial_accumulator_value=0.1, epsilon=1e-07, name='Adagrad')
```

- RMSProp
  - 이전 그래디언트 누적할 때 오래될수록 영향력을 줄여가는 정책으로 AdaGrad 사용
  - 누적된 값과 새로운 그래디언트 양을 반비례로 구성해 최근 그래디언트로 더 나아갈지 아니면 기존 방향성을 가질지 결정 가능
  - 보통 알파를 0.9 0.99 0.999 로 적용

```
tf.keras.optimizers.RMSProp(learning_rate=0.001, rho=0.9, momentum=0.0, epsilon=1e-07, centered=False, name='RMSProp')
```

  rho 오래된 그래디언트의 영향력 줄이는 매개변수
- Adam(adaptive moment)
  - RMSProp에 모멘텀을 추가로 적용한 알고리즘
  - 속도와 그래디언트 누적을 모두 사용해 식 자체가 누적이 많을수록 속도가 감소하는 구조

```
tf.keras.optimizers.Adam(learning_rate=0.001, beta_1=0.9, beta_1=0.999, epsilon=1e-07, amgrad=False, name='Adam')
```

  beta_1 모멘텀에 관한 매개변수, beta_2 RMSProp의 rho 해당

규제

신경망 모델의 용량을 크게 유지하며 여러 규제 기법을 적용해 과잉 적합을 방지하는 전략 사용

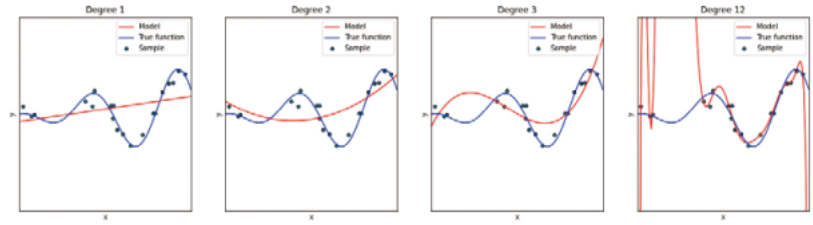
과잉적합

파란색 곡선 : 샘플을 취득한 원래 곡선

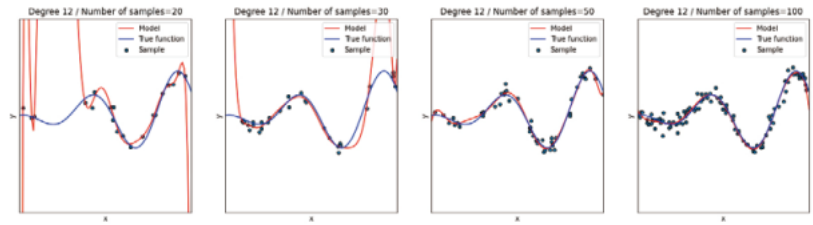
검은색 점 : 샘플

빨간색 곡선 : 샘플을 훈련집합으로 사용해 학습한 모델

우측으로 갈수록 가중치 개수가 많아져 용량이 큰 모델



(a) 과소 적합과 과잉 적합 현상



(b) 데이터 증강을 통한 과잉 적합 방지

그림 8-20 과잉 적합 현상과 데이터 증강을 통한 방지

과소적합 : 데이터에 비해 용량이 작아 모델링이 제대로 되지 않음

과잉적합 : 훈련집합에 대해 거의 완벽 모델링, 실제 함수와는 거리가 멀다, 일반화 능력 낮음

데이터 증강 *Data Agumentation*

주어진 훈련집합을 조금씩 변형해 인위적으로 데이터를 늘리는 방법

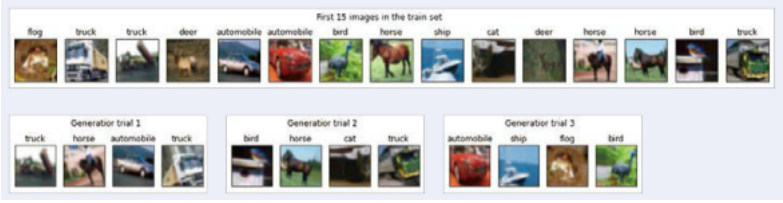
영상에 약간의 이동, 회전, 크기, 명암 변환을 랜덤하게 적용해 데이터 무한대로 증강

```
# 8-5 증강된 영상 확인하기
import tensorflow.keras.datasets as ds
from tensorflow.keras.preprocessing.image import ImageDataGenerator # 데이터 증강 지원
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 # 실수형으로 변환 -> 범위 [0,1]
x_train=x_train[0:15,]; y_train=y_train[0:15,]  # 50,000개 중, 앞 15개에 대해서만 증대 적용
class_names=['airplane','automobile','bird','cat','deer','dog','flog','horse','ship','truck']

# 15개 영상 display
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_siz=4     # 한 번에 생성하는 양(미니 배치) -> 단지 확인할 목적으로 작게 설정
generator=ImageDataGenerator(rotation_range=20.0,width_shift_range=0.2,height_shift_range=0.2,horizontal_flip=True) # 데이터 변환 방식, 범
gen=generator.flow(x_train,y_train,batch_size=batch_siz)

# 세번에 걸쳐 데이터 생성
for a in range(3):
    img,label=gen.next()  # 미니 배치만큼 데이터 증강해 생성
    # 증강된 네 영상 display
    plt.figure(figsize=(8,2.4))
    plt.suptitle("Generatior trial "+str(a+1))
    for i in range(batch_siz):
        plt.subplot(1,batch_siz,i+1)
        plt.imshow(img[i])
        plt.xticks([]); plt.yticks([])
        plt.title(class_names[int(label[i])])
    plt.show()
```

```
generator=ImageDataGenerator(rotation_range=20.0,width_shift_range=0.2,height_shift_range=0.2,horizontal_flip=True)
```

- 회전 : -20~+20
- 가로 방향 이동 20% 이내
- 세로 방향 이동 20% 이내
- 좌우 반전 시도 설정

▼ 결과



드롭아웃 *Drop out*

특징 맵을 구성하는 요소중, 일부를 랜덤 선택해 0으로 설정

과잉 적합 방지에 효과적

r : 0으로 만들 요소의 비율

ex. r=0.3 이라면 전체 요소 중, 30% 가 모두 0

조기멈춤 *EarlyStopping*

훈련집합 성능이 떨어지고 검증집합이 특정 구간부터 증가하면 과잉 적합 발생

→ 이때 조기멈춤 사용

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

- min delta : 그것보다 작은 개선은 개선으로 여기지 않기
- 그 값에 해당하는 세대동안 개선없으면 지정한 세대에 도달하지 않았더라도 조기에 학습 stop
- 멈추는 순간의 가중치 취하기 ↔ 가장 좋았던 epoch 의 가중치 취하기

8.7 전이학습

TL : 개, 고양이뿐만 아니라 곤충, 말 등의 엄청 많은 종류의 사진들을 기반으로 학습한 모델에 예측하고 싶은 데이터를 넣어 input 한 데이터에 맞게 변형

→ 각각 보편적인 특징을 갖고 있는 모델, 너무 많이 학습해서 모든 것의 진리 습득

↔ TL, Classification은 라벨에 따라 데이터를 분리한다면

Few shot : 사람처럼 데이터 본질을 보고 구분, 일부 전이학습으로 간주하나 이를 활용하는 데이터를 이용하는 측면이 다름
⇒ 단순 CNN이 아닌 샴네트워크 등의 거대모델로 활용

데이터의 크기와 같은 문제를 해결하기 위해 전이학습 사용

사전학습모델 *pre-trained model* : 대용량 데이터로 미리 학습되어 있어 전이학습에 활용할 수 있는 모델

→

백본모델 backbone *model* : 사전 학습 모델을 특정 응용에 전이 학습하면 백본모델로 사용했다고 표

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

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

```
# 8-6 ResNet50으로 자연 연상 인식하기
## ResNet50을 백본모델로 사용
import cv2 as cv
import numpy as np
from tensorflow.keras.applications.resnet50 import ResNet50,preprocess_input,decode_predictions

model=ResNet50(weights='imagenet') # 사전 학습 모델읽어 model에 저장, weights='imagenet'인수는 ImageNet으로 학습된 가중치 읽기

# 테스트 영상 읽고 신경망에 입력할 수 있는 형태로 변환
img=cv.imread('rabbit.jpg')
x=np.reshape(cv.resize(img,(224,224)),(1,224,224,3))
x=preprocess_input(x) # 영상을 신경망에 입력하기 전 수행하는 전처리

preds=model.predict(x) # 여러장으로 구성된 미니 배치 단위로 입력받고 예측을 수행하고 결과 저장
top5=decode_predictions(preds,top=5)[0] # 가장 큰 5개 확률 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()
```

▼ 결과

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

웹사이트 접속해 스탠포드 도그 데이터셋 설치...

1. 소스코드가 있는 폴더에 datasets/Stanford_dogs 폴더를 만든다.

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

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

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

[그림 8-25]는 120개 폴더 중에서 앞의 6개 폴더에 있는 첫 번째 영상이다. 대부분 개를 중심에 놓고 찍은 영상이고 배경이 아주 다양함을 확인할 수 있다.

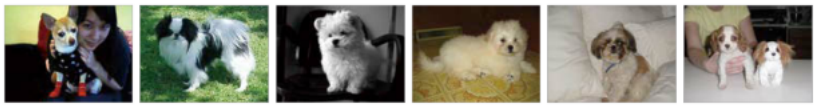


그림 8-25 Stanford dogs 데이터셋(왼쪽부터 Chihuahua, Japanese_spaniel, Maltese_dog, Pekinese, Shih-Tzu, Blenheim_spaniel)

미세조정이 가능한 transfer learning

```
# 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 # 필요할 때마다 사용, 폴더에서 영상을 읽는 데 사용
import pathlib

data_path=pathlib.Path('datasets/stanford_dogs/images/images')

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) #

# 신경망 모델 구성
base_model=DenseNet121(weights='imagenet',include_top=False,input_shape=(224,224,3)) # include_top=False: 모델의 뒷쪽에 있는 완전연결층 포함시
cnn=Sequential()
cnn.add(Rescaling(1.0/255.0)) # 범위 변환
cnn.add(base_model) # densenet121 추가
cnn.add(Flatten()) # 1차원
cnn.add(Dense(1024,activation='relu')) # 완전연결층
cnn.add(Dropout(0.75)) # 드롭아웃
cnn.add(Dense(units=120,activation='softmax'))

# 미세조정 : 사전학습된 모델 뒤에 새로운 층 붙여 신경망 구성, 학습률 낮게 설정해 다시 학습하는 방식
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

import pickle
f=open('dog_species_names.txt','wb')
pickle.dump(train_ds.class_names,f)
f.close()

# 정확률 , 손실함수의 학습곡선
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()
```

8.8 [비전 에이전트7] 견종 인식 프로그램

```
# 8-8 견종 인식 프로그램
import cv2 as cv
import numpy as np
import tensorflow as tf
import winsound
import pickle
import sys
from PyQt5.QtWidgets import *

cnn=tf.keras.models.load_model('cnn_for_stanford_dogs.h5')  # 모델 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) # 윈도우의 위치와 크기 지정

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

        # 버튼의 위치와 크기 지정
        fileButton.setGeometry(10,10,100,30)
        recognitionButton.setGeometry(110,10,100,30)
        quitButton.setGeometry(510,10,100,30)

        # 수행할 콜백함수 생성
        fileButton.clicked.connect(self.pictureOpenFunction)
        recognitionButton.clicked.connect(self.recognitionFunction)
        quitButton.clicked.connect(self.quitFunction)

    # 강아지 사진 열기 : 파일에서 부라우저 읽고 영상 윈도우에 디스플레이
    def pictureOpenFunction(self):
        fname=QFileDialog.getOpenFileName(self,'강아지 사진 읽기','./')
        self.img=cv.imread(fname[0])
        if self.img is None: sys.exit('파일을 찾을 수 없습니다.')

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

    # 품종인식
    def recognitionFunction(self):
        x=np.reshape(cv.resize(self.img,(224,224)),(1,224,224,3))  # img 신경망에 입력할 준비
        res=cnn.predict(x)[0]   # 예측
        top5=np.argsort(-res)[:5]
        top5_dog_species_names=[dog_species[i] for i in top5] # 인덱스를 견종이름으로 변환
        for i in range(5):
            # 품종 이름과 확률 작성
            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)

    # 나가기 : 윈도우 닫고 프로그램 종료
    def quitFunction(self):
        cv.destroyAllWindows()
        self.close()

# 프로그램 메인
app=QApplication(sys.argv)
win=DogSpeciesRecognition()
win.show()
app.exec_()
```