

AI 라이브러리 활용

16장 CNN 익히기

이 찬 우

학습 내용 : 이미지 인식의 꽃, CNN 익히기

1 | 데이터 전처리

2 | 딥러닝 기본 프레임 만들기

3 | 더 깊은 딥러닝

4 | 컨볼루션 신경망(CNN)

5 | 맥스 풀링

6 | 컨볼루션 신경망 실행하기

이미지 인식의 꽃, CNN 익히기

- 실습과제 MNIST 손글씨 인식



- 컴퓨터에게 이 글씨를 읽게 하고 이 글씨가 어떤 의미인지를 알게 하는 과정은 쉽지 않음

이미지 인식의 꽃, CNN 익히기

- MNIST 데이터셋은 미국 국립표준기술원(NIST)이 고등학생과 인구조사국 직원 등이 쓴 손글씨를 이용해 만든 데이터로 구성되어 있음
 - 70,000개의 글자 이미지에 각각 0부터 9까지 이름표를 붙인 데이터셋
 - 머신러닝을 배우는 사람이라면 자신의 알고리즘과 다른 알고리즘의 성과를 비교해 보고자 한 번씩 도전해 보는 가장 유명한 데이터 중 하나임

이미지 인식의 꽃, CNN 익히기



그림 16-1 MNIST 손글씨 데이터 이미지

1 | 데이터 전처리

- MNIST 데이터는 케라스를 이용해 간단히 불러올 수 있음
- `mnist.load_data()` 함수로 사용할 데이터를 불러옴

```
from keras.datasets import mnist
```

1 | 데이터 전처리

- 학습에 사용될 부분: X_train, Y_class_train
- 테스트에 사용될 부분: X_test, Y_class_test

```
(X_train, Y_class_train), (X_test, Y_class_test) = mnist.load_data( )
```

1 | 데이터 전처리

- 케라스의 MNIST 데이터는 총 70,000개의 이미지 중 60,000개를 학습용으로, 10,000개를 테스트용으로 미리 구분해 놓고 있음

```
print("학습셋 이미지 수: %d 개" % (X_train.shape[0]))  
print("테스트셋 이미지 수: %d 개" % (X_test.shape[0]))
```

학습셋 이미지 수: 60000 개

테스트셋 이미지 수: 10000 개

1 | 데이터 전처리

- 불러온 이미지 중 한 개만 다시 불러와 보겠음

```
import matplotlib.pyplot as plt  
plt.imshow(X_train[0], cmap='Greys')  
plt.show()
```

1 | 데이터 전처리

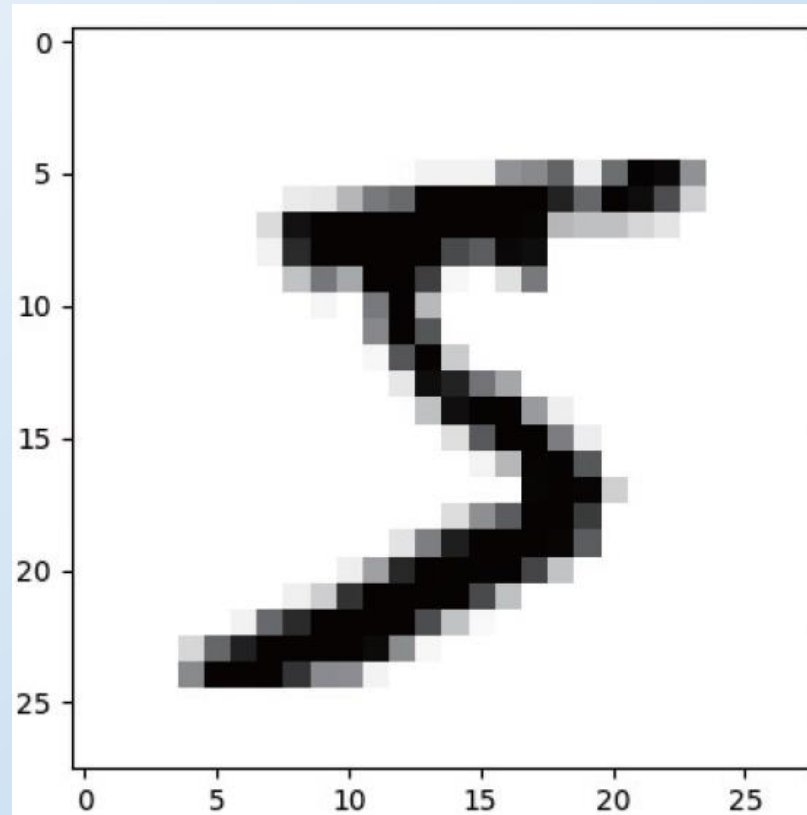


그림 16-2 MNIST 손글씨 데이터의 첫 번째 이미지

1 | 데이터 전처리

- 이 이미지는 가로 28 × 세로 28 = 총 784개의 픽셀로 이루어져 있음
- 각 픽셀은 밝기 정도에 따라 0부터 255까지의 등급을 매김
- 흰색 배경이 0이라면 글씨가 들어간 곳은 1~255까지 숫자 중 하나로 채워져
긴 행렬로 이루어진 하나의 집합으로 변환됨

```
for x in X_train[0]:  
    for i in x:  
        sys.stdout.write('%d\t' % i)  
    sys.stdout.write('\n')
```

1 | 데이터 전처리

[illegible]

그림 16-3 그림의 각 좌표를 숫자로 표현해보기

1 | 데이터 전처리

- 바로 이렇게 이미지는 다시 숫자의 집합으로 바뀌어 학습셋으로 사용됨
- 이제 주어진 가로 28, 세로 28의 2차원 배열을 784개의 1차원 배열로 바꿔 주어야 하는데 이를 위해 reshape() 함수를 사용함
- reshape(총 샘플 수, 1차원 속성의 수) 형식으로 지정함

```
X_train = X_train.reshape(X_train.shape[0], 784)
```

1 | 데이터 전처리

- 정규화(normalization) :
데이터의 폭이 클 때 적절한 값으로 분산의 정도를 바꾸는 과정
- 현재 주어진 데이터의 값은 0부터 255까지의 정수로, 정규화를 위해
255로 나누어 주려면 먼저 이 값을 실수형으로 바꿔야 함

```
X_train = X_train.astype('float64')  
X_train = X_train / 255
```

1 | 데이터 전처리

- X_test에도 마찬가지로 이 작업을 적용함


```
X_test = X_test.reshape(X_test.shape[0], 784).astype('float64') / 255
```

- 실제로 이 숫자의 레이블이 어떤지를 불러오고자
Y_class_train[0]을 다음과 같이 출력해보자

```
print("class : %d " % (Y_class_train[0]))
```

1 | 데이터 전처리

- 이 숫자의 레이블 값인 5가 출력되는 것을 볼 수 있음



```
class : 5
```

- 아이리스 품종을 예측할 때 딥러닝의 분류 문제를 해결하려면 원-핫 인코딩 방식을 적용해야 함
- 즉, 0~9까지의 정수형 값을 갖는 현재 상태에서 0 또는 1로만 이루어진 벡터로 값을 수정해야 함

1 | 데이터 전처리

- 지금 우리가 열어본 이미지의 class는 [5]였음
- 이를 [0,0,0,0,0,1,0,0,0,0]로 바꿔야 함
- 이를 가능하게 해 주는 함수가 바로 `np_utils.to_categorical()` 함수임
- `to_categorical(클래스, 클래스의 개수)`의 형식으로 지정함

```
Y_train = np_utils.to_categorical(Y_class_train,10)
```

```
Y_test = np_utils.to_categorical(Y_class_test,10)
```

1 | 데이터 전처리

- 이제 변환된 값을 출력해보자

```
print(Y_train[0])
```

- 아래와 같이 원-핫 인코딩이 적용된 것을 확인할 수 있음

```
[ 0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.]
```

1 | 데이터 전처리

코드 16-1 MNIST 손글씨 인식하기: 데이터 전처리

- 예제 소스: run_project/14_MNIST_Data.ipynb

```
from keras.datasets import mnist
from keras.utils import np_utils

import numpy
import sys
import tensorflow as tf

# seed 값 설정
seed = 0
numpy.random.seed(seed)
tf.random.set_seed(3)
```

1 | 데이터 전처리

```
# MNIST 데이터셋 불러오기

(X_train, Y_class_train), (X_test, Y_class_test) = mnist.load_
data( )

print("학습셋 이미지 수 : %d 개" % (X_train.shape[0]))
print("테스트셋 이미지 수 : %d 개" % (X_test.shape[0]))

# 그래프로 확인

import matplotlib.pyplot as plt
plt.imshow(X_train[0], cmap='Greys')
plt.show( )

# 코드로 확인

for x in X_train[0]:
```

1 | 데이터 전처리

```
for i in x:
    sys.stdout.write('%d\t' % i)
sys.stdout.write('\n')

# 차원 변환 과정
X_train = X_train.reshape(X_train.shape[0], 784)
X_train = X_train.astype('float64')
X_train = X_train / 255

X_test = X_test.reshape(X_test.shape[0], 784).astype('float64') /
255

# 클래스 값 확인
print("class : %d " % (Y_class_train[0]))
```

1 | 데이터 전처리

바이너리화 과정

```
Y_train = np_utils.to_categorical(Y_class_train, 10)
```

```
Y_test = np_utils.to_categorical(Y_class_test, 10)
```

```
print(Y_train[0])
```

2 | 딥러닝 기본 프레임 만들기

- 총 60,000개의 학습셋과 10,000개의 테스트셋을 불러와 속성 값을 지닌 X, 클래스 값을 지닌 Y로 구분하는 작업

```
from keras.datasets import mnist

(X_train, Y_train), (X_test, Y_test) = mnist.load_data()

X_train = X_train.reshape(X_train.shape[0], 784).
astype('float32') / 255
X_test = X_test.reshape(X_test.shape[0], 784).astype('float32') /
255

Y_train = np_utils.to_categorical(Y_train, 10)
Y_test = np_utils.to_categorical(Y_test, 10)
```

2 | 딥러닝 기본 프레임 만들기

- 이제 딥러닝을 실행하고자 프레임을 설정함

```
model = Sequential()  
model.add(Dense(512, input_dim=784, activation='relu'))  
model.add(Dense(10, activation='softmax'))
```

- 활성화 함수로 은닉층에서는 relu를, 출력층에서는 softmax를 사용함

2 | 딥러닝 기본 프레임 만들기

- 딥러닝 실행 환경을 위해 오차 함수로 `categorical_crossentropy`, 최적화 함수로 `adam`을 사용함

```
model.compile(loss='categorical_crossentropy',  
              optimizer='adam',  
              metrics=['accuracy'])
```

2 | 딥러닝 기본 프레임 만들기

- 모델의 실행에 앞서 모델의 성과를 저장하고 모델의 최적화 단계에서 학습을 자동 중단하게끔 설정함
- 10회 이상 모델의 성과 향상이 없으면 자동으로 학습을 중단함

2 | 딥러닝 기본 프레임 만들기

```
import os
from keras.callbacks import ModelCheckpoint, EarlyStopping

MODEL_DIR = './model/'
if not os.path.exists(MODEL_DIR):
    os.mkdir(MODEL_DIR)

modelpath = "./model/{epoch:02d}-{val_loss:.4f}.hdf5"
checkpointer = ModelCheckpoint(filepath=modelpath, monitor='val_loss',
                               verbose=1, save_best_only=True)
early_stopping_callback = EarlyStopping(monitor='val_loss',
                                         patience=10)
```

2 | 딥러닝 기본 프레임 만들기

- 샘플 200개를 모두 30번 실행하게끔 설정함
- 테스트셋으로 최종 모델의 성과를 측정하여 그 값을 출력함

```
history = model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=30, batch_size=200, verbose=0, callbacks=[early_stopping_callback,checkpointer])
```

```
print("\n Test Accuracy: %.4f" % (model.evaluate(X_test, Y_test)[1]))
```

2 | 딥러닝 기본 프레임 만들기

- 학습셋의 오차는 1에서 학습셋의 정확도를 뺀 값임
- 좀 더 세밀한 변화를 볼 수 있게 학습셋의 오차와 테스트셋의 오차를 그래프 하나로 나타냄

```
import matplotlib.pyplot as plt

y_vloss = history.history['val_loss']

# 학습셋의 오차
y_loss = history.history['loss']
```

2 | 딥러닝 기본 프레임 만들기

그래프로 표현

```
x_len = numpy.arange(len(y_loss))  
plt.plot(x_len, y_vloss, marker='.', c="red", label='Testset_  
loss')  
plt.plot(x_len, y_loss, marker='.', c="blue", label='Trainset_  
loss')
```

그래프에 그리드를 주고 레이블을 표시

```
plt.legend(loc='upper right')  
plt.grid()  
plt.xlabel('epoch')  
plt.ylabel('loss')  
plt.show()
```

2 | 딥러닝 기본 프레임 만들기

코드 16-2 MNIST 손글씨 인식하기: 기본 프레임

- 예제 소스: run_project/15_MNIST_Simple.ipynb

```
from keras.datasets import mnist
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import ModelCheckpoint, EarlyStopping

import matplotlib.pyplot as plt
import numpy
import os
import tensorflow as tf
```

2 | 딥러닝 기본 프레임 만들기

```
# seed 값 설정
```

```
seed = 0
```

```
numpy.random.seed(seed)
```

```
tf.set_random_seed(3)
```

```
# MNIST 데이터 불러오기
```

```
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
```

```
X_train = X_train.reshape(X_train.shape[0], 784).  
astype('float32') / 255
```

```
X_test = X_test.reshape(X_test.shape[0], 784).astype('float32') /  
255
```


2 | 딥러닝 기본 프레임 만들기

```
Y_train = np_utils.to_categorical(Y_train, 10)
Y_test = np_utils.to_categorical(Y_test, 10)

# 모델 프레임 설정
model = Sequential( )
model.add(Dense(512, input_dim=784, activation='relu'))
model.add(Dense(10, activation='softmax'))

# 모델 실행 환경 설정
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

2 | 딥러닝 기본 프레임 만들기

```
# 모델 최적화 설정
MODEL_DIR = './model/'
if not os.path.exists(MODEL_DIR):
    os.mkdir(MODEL_DIR)

modelpath="./model/{epoch:02d}-{val_loss:.4f}.hdf5"
checkpointer = ModelCheckpoint(filepath=modelpath, monitor='val_
loss', verbose=1, save_best_only=True)
early_stopping_callback = EarlyStopping(monitor='val_loss',
patience=10)
```

2 | 딥러닝 기본 프레임 만들기

모델의 실행

```
history = model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=30, batch_size=200, verbose=0, callbacks=[early_stopping_callback,checkpointer])
```

테스트 정확도 출력

```
print("\n Test Accuracy: %.4f" % (model.evaluate(X_test, Y_test)[1]))
```

테스트셋의 오차

```
y_vloss = history.history['val_loss']
```

2 | 딥러닝 기본 프레임 만들기

```
# 학습셋의 오차
```

```
y_loss = history.history['loss']
```

```
# 그래프로 표현
```

```
x_len = numpy.arange(len(y_loss))
```

```
plt.plot(x_len, y_vloss, marker='.', c="red", label='Testset_ loss')
```

```
plt.plot(x_len, y_loss, marker='.', c="blue", label='Trainset_ loss')
```

2 | 딥러닝 기본 프레임 만들기

```
# 그래프에 그리드를 주고 레이블을 표시
plt.legend(loc='upper right')
# plt.axis([0, 20, 0, 0.35])
plt.grid()
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show()
```

2 | 딥러닝 기본 프레임 만들기

실행
결과



Epoch 00000: val_loss improved from inf to 0.15460, saving model to ./model/00-0.1546.hdf5

Epoch 00001: val_loss improved from 0.15460 to 0.10492, saving model to ./model/01-0.1049.hdf5

Epoch 00002: val_loss improved from 0.10492 to 0.08447, saving model to ./model/02-0.0845.hdf5

Epoch 00003: val_loss improved from 0.08447 to 0.07896, saving model to ./model/03-0.0790.hdf5

Epoch 00004: val_loss improved from 0.07896 to 0.06699, saving model to ./model/04-0.0670.hdf5

Epoch 00005: val_loss improved from 0.06699 to 0.06388, saving model to ./model/05-0.0639.hdf5

2 | 딥러닝 기본 프레임 만들기

Epoch 00006: val_loss did not improve

Epoch 00007: val_loss improved from 0.06388 to 0.06291, saving model to ./model/07-0.0629.hdf5

Epoch 00008: val_loss improved from 0.06291 to 0.05828, saving model to ./model/08-0.0583.hdf5

Epoch 00009: val_loss did not improve

Epoch 00010: val_loss did not improve

Epoch 00011: val_loss did not improve

Epoch 00012: val_loss did not improve

Epoch 00013: val_loss did not improve

Epoch 00014: val_loss did not improve

Epoch 00015: val_loss did not improve

Epoch 00016: val_loss did not improve

Epoch 00017: val_loss did not improve

Epoch 00018: val_loss did not improve

Epoch 00019: val_loss did not improve

9920/10000 [=====>.] - ETA: 0s

Test Accuracy: 0.9821

2 | 딥러닝 기본 프레임 만들기

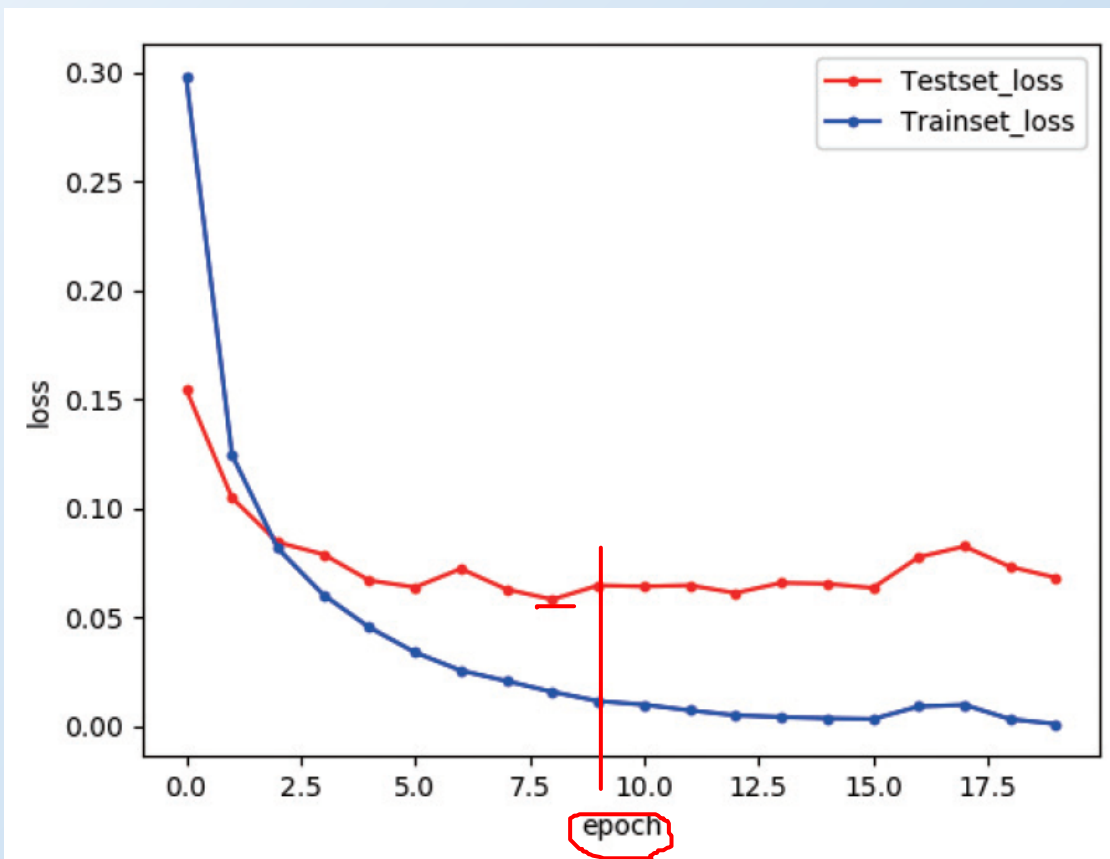


그림 16-4 학습이 진행될 때 학습셋과 테스트셋의 오차 변화

2 | 딥러닝 기본 프레임 만들기

- 학습셋에 대한 오차는 계속해서 줄어듦
- 테스트셋의 과적합이 일어나기 전 학습을 끝낸 모습임

TIP

그림 16-4는 학습셋의 오차(Trainset_loss)와 테스트셋의 오차(Testset_loss)를 그래프로 표현한 것입니다. 앞서 학습셋과 테스트셋을 배울 때는 학습셋의 정확도(Trainset_acc)를 이용했습니다(그림 14-1). 하지만 학습셋의 정확도는 1.00에 가깝고 테스트셋의 오차는 0.00에 가까우므로 두 개를 함께 비교하기가 어렵습니다. 따라서 1에서 학습셋의 정확도를 뺀 값, 즉 학습셋의 오차를 주로 그래프에 적용하여 이와 같이 표현합니다.

정리 학습 : 이미지 인식의 꽃, CNN 익히기

1 | 데이터 전처리

2 | 딥러닝 기본 프레임 만들기

다음 수업

3 | 더 깊은 딥러닝

4 | 컨볼루션 신경망(CNN)

5 | 맥스 풀링

6 | 컨볼루션 신경망 실행하기