AI 프로그래밍

- 11주차

인하 공전 컴퓨터 정보과

민 정혜

오늘 수업 순서- 딥러닝 예제

- **케라스 딥러닝 복습**
- **케라스 딥러닝 예제 실습**
  - 타이나닉 생존자 예측
  - 패션 아이템 분류
  - 와인 품종 예측
  - 주택 가격 예측
- **영상 처리기초 : open cv**

Tensorflow 2.8

케라스로 신경망을 작성 하는 절차


```
model = tf.keras.models.Sequential()


model.add(tf.keras.layers.Dense(units=2, input_shape=(2,), activation='sigmoid')) #①
model.add(tf.keras.layers.Dense(units=1, activation='sigmoid')


model.compile(loss='mean_squared_error', optimizer=keras.optimizers.SGD(lr=0.3)
              , metrics='accuracy')


model.fit(X, y, batch_size=1, epochs=10000)


print( model.predict(X) )
```

Sequential 모델을 생성

Add함수를 이용하여 layer 추가

컴파일

학습을 수행

모델 테스트

[Keras API reference](https://keras.io/api/)

https://keras.io/api/

케라스 신경망 파라미터


```
model = tf.keras.models.Sequential()


model.add(tf.keras.layers.Dense(units=2, input_shape=(2,), activation='sigmoid')) #①
model.add(tf.keras.layers.Dense(units=1, activation='sigmoid')



model.compile(loss='mean_squared_error', optimizer=keras.optimizers.SGD(lr=0.3),
, metrics='accuracy'))


model.fit(X, y, batch_size=1, epochs=10000)


print( model.predict(X) )
```

케라스 딥러닝

- **피마 인디언 데이터 분석**
  - 데이터 가공
- **아이리스 품종 예측**
  - 다중 분류, 상관도 그래프, 원-핫 인코딩, 소프트 맥스
- **초음파 광물 예측**
  - 과 적합 (Overfitting) 피하기, 교차 검증, 모델 저장과 재사용

케라스 딥러닝 – 피마 인디언 데이터 분석

	pregnant	plasma	pressure	thickness	insulin	BMI	pedigree	age	class
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

당뇨병 여부 : 0, 1

```
# 모델의 설정
model = Sequential( )
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

입력 층 : 8
은닉층 1 : 12 (relu)
은닉측 2 : 8 (relu)
출력 층 1 (sigmoid)

활성화 함수

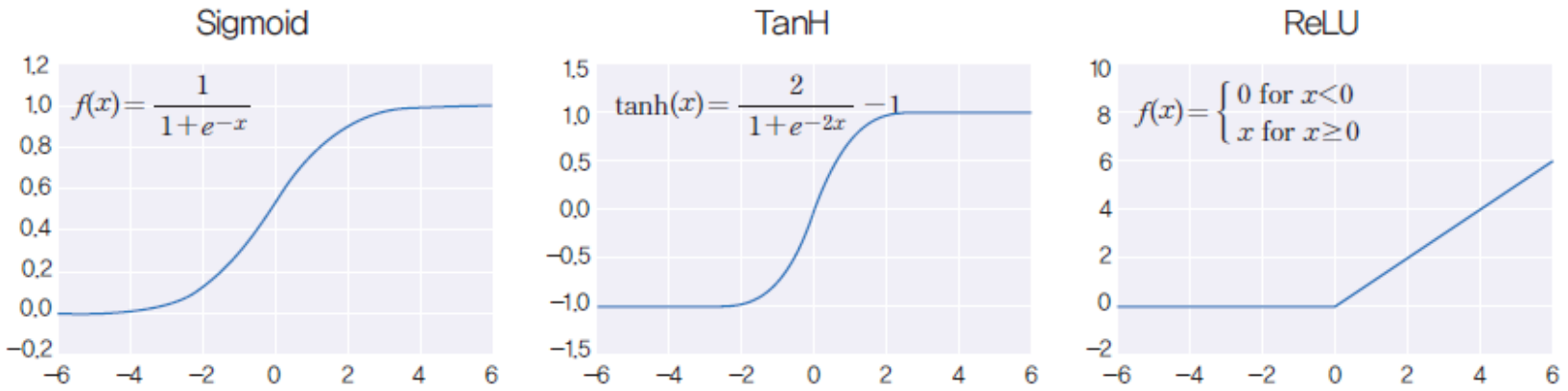


그림 6-3 많이 사용되는 활성화 함수

범위

[0.0, 1.0]

범위

[-1.0, 1.1]

범위

[0.0. ~]

케라스 딥러닝 – 정수 인코딩

- **아이리스 품종 예측**

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5	3.6	1.4	0.2	Iris-setosa

- 문자열을 숫자로 바꿔 주려면 클래스 이름을 숫자 형태로 바꿔 주어야 함

```
from sklearn.preprocessing import LabelEncoder

e = LabelEncoder( )
e.fit(Y_obj)
Y = e.transform(Y_obj)
```

케라스 딥러닝 – 원-핫 인코딩

- **아이리스 품종 예측**

```
from tensorflow.keras.utils import np_utils


Y_encoded = tf.keras.utils.to_categorical(Y)
```

- array([1,2,3])가 다시 array([[1., 0., 0.], [0., 1., 0.],[0., 0., 1.]])로 바뀜
- 원-핫 인코딩(one-hot-encoding) :

  여러 개의 Y 값을 0과 1로만 이루어진 형태로 바꿔 주는 기법

케라스 딥러닝

Y_obj=

```
array(['Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-
setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-
setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-
setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-
setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-
setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-
setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-
setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-
setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-
setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-
setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-
setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-
setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-versicolor',
'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
'Iris-versicolor',
```

```
from sklearn.preprocessing import LabelEncoder

e = LabelEncoder()
e.fit(Y_obj)
Y = e.transform(Y_obj)
```

정수 인코딩

Y=

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2])
```

```
from tensorflow.keras.utils import np_utils

Y_encoded = tf.keras.utils.to_categorical(Y)
```

원 핫 인코딩

Y_encoded

```
array([[1., 0., 0.], [1., 0., 0.], [1., 0., 0.], [1.,
0., 0.], [1., 0., 0.], [1., 0., 0.], [1., 0., 0.],
[1., 0., 0.], [1., 0., 0.],
```

케라스 딥러닝

```
import numpy as np
X = np.array([['Korea', 44, 7200],
              ['Japan', 27, 4800],
              ['China', 30, 6100]])

from sklearn.preprocessing import LabelEncoder
labelencoder = LabelEncoder()
X[:, 0] = labelencoder.fit_transform(X[:, 0])
print(X)
```

```
[['2' '44' '7200']
['1' '27' '4800']
['0' '30' '6100']]
```

정수 인코딩

원 핫 인코딩

원-핫 인코딩은 단 하나의 값만 1이고 나머지는 모두 0인 인코딩을 의미한다.

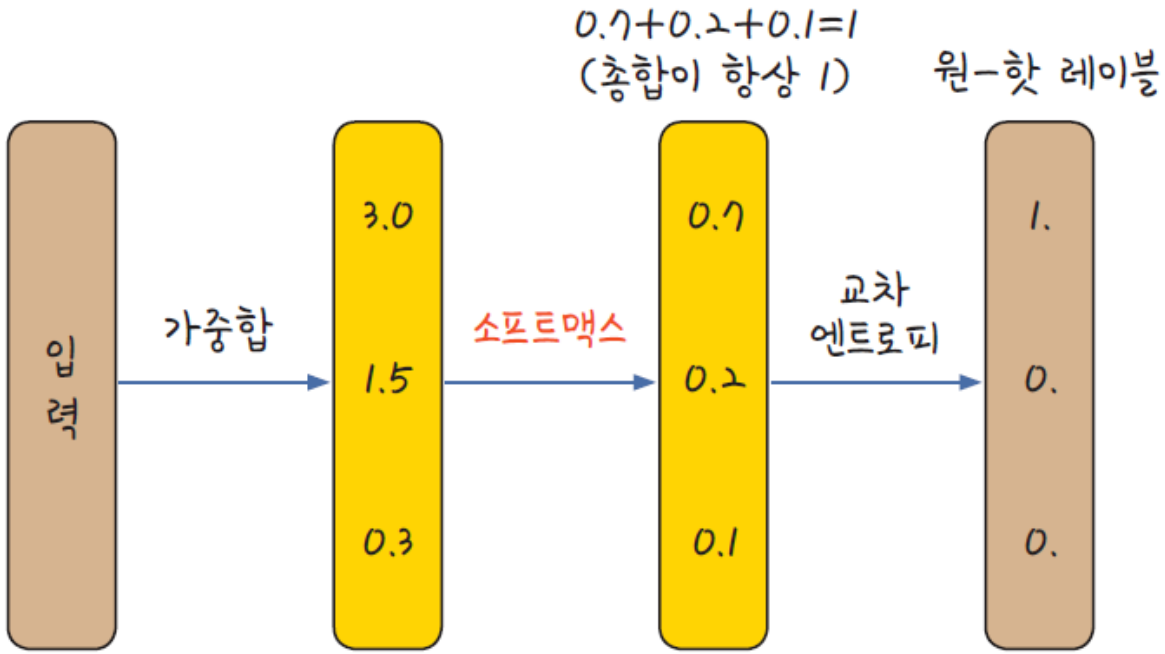
Country	Age	Salary
Korea	38	7200
Japan	27	4800
China	30	3100

Korea	Japan	China	Age	Salary
1	0	0	38	7200
0	1	0	27	4800
0	0	1	30	3100

케라스 딥러닝 – 소프트 맥스

- 소프트맥스 :

  그림 12-3에서와 같이 총합이 1인 형태로 바꿔서 계산해 주는 함수




```
model = Sequential()
model.add(Dense(16, input_dim=4, activation='relu'))
model.add(Dense(3, activation='softmax'))
```

그림 12-3 소프트맥스 함수의 원리

케라스 신경망 - 초음파 광물 예측

- 과적합을 방지하려면 어떻게 해야 할까?

  → 먼저 학습을 하는 데이터셋과 이를 테스트할 데이터셋을 완전히 구분한 다음 학습과 동시에 테스트를 병행하며 진행하는 것이 한 방법

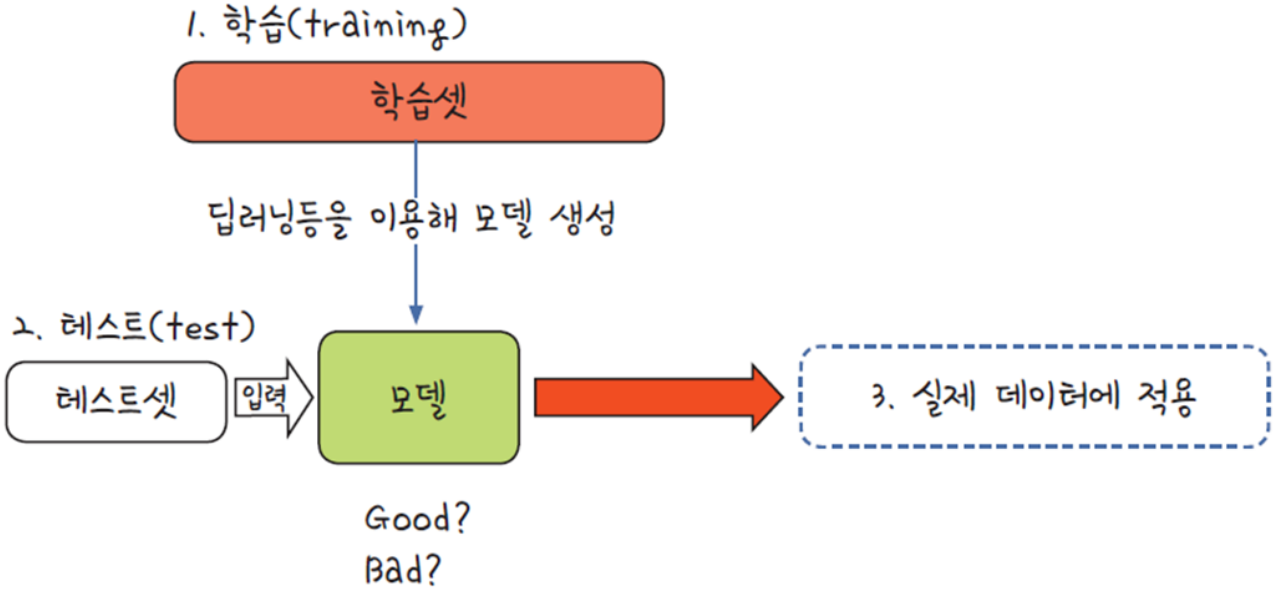
학습셋 테스트 셋 구분

- 데이터셋이 총 100개의 샘플로 이루어져 있다면 다음과 같이 두 개의 셋으로 나눔

70개 샘플은 학습셋으로	30개 샘플은 테스트셋으로

```
from sklearn.model_selection import train_test_split

# 학습셋과 테스트셋의 구분
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_
size=0.3, random_state=seed)
```

케라스 신경망 - 초음파 광물 예측

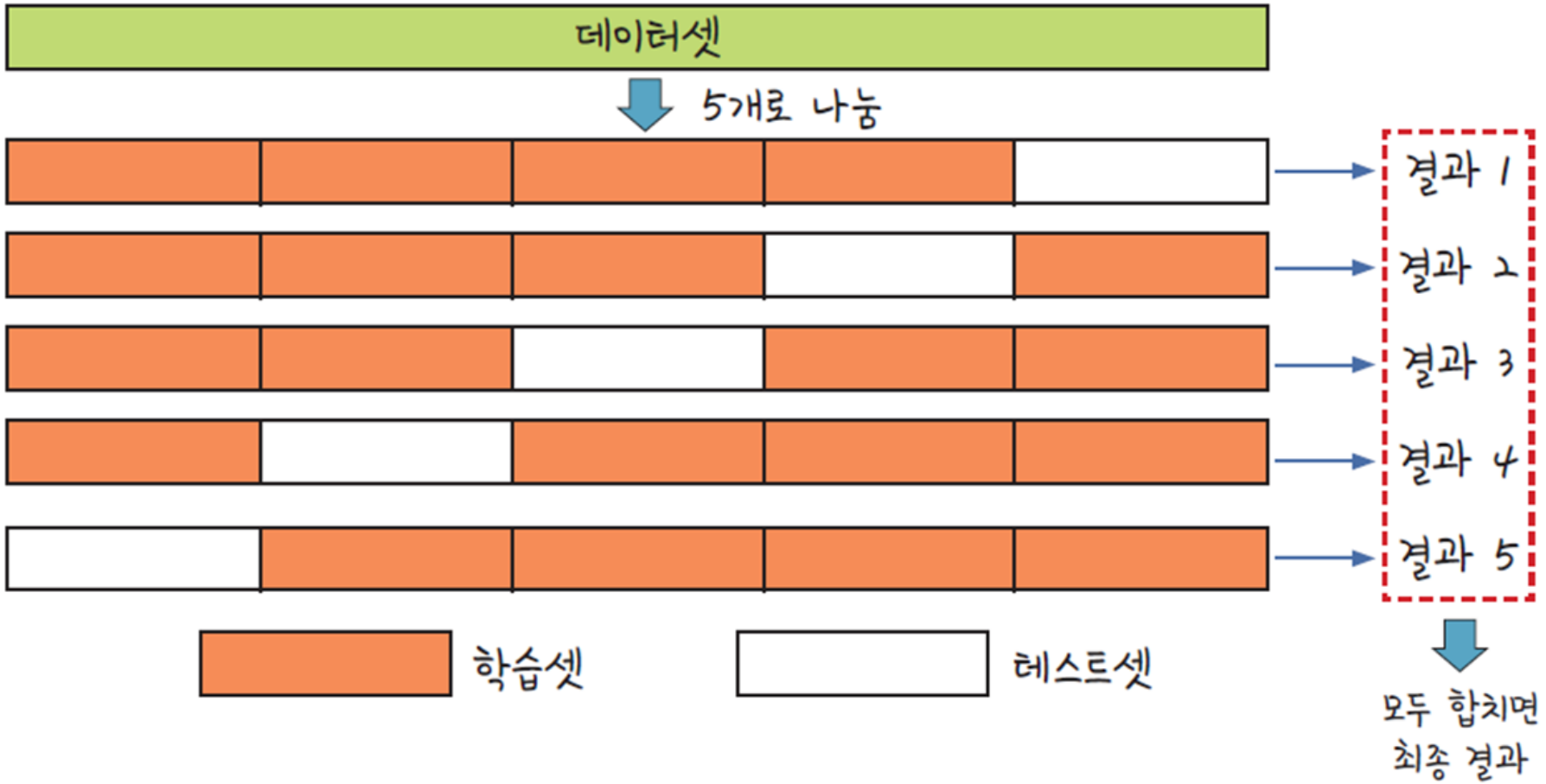


학습셋 테스트 셋 구분

```
model.fit(X_train, Y_train, epochs=130, batch_size=5)

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

케라스 신경망 - 초음파 광물 예측

5 | k겹 교차 검증

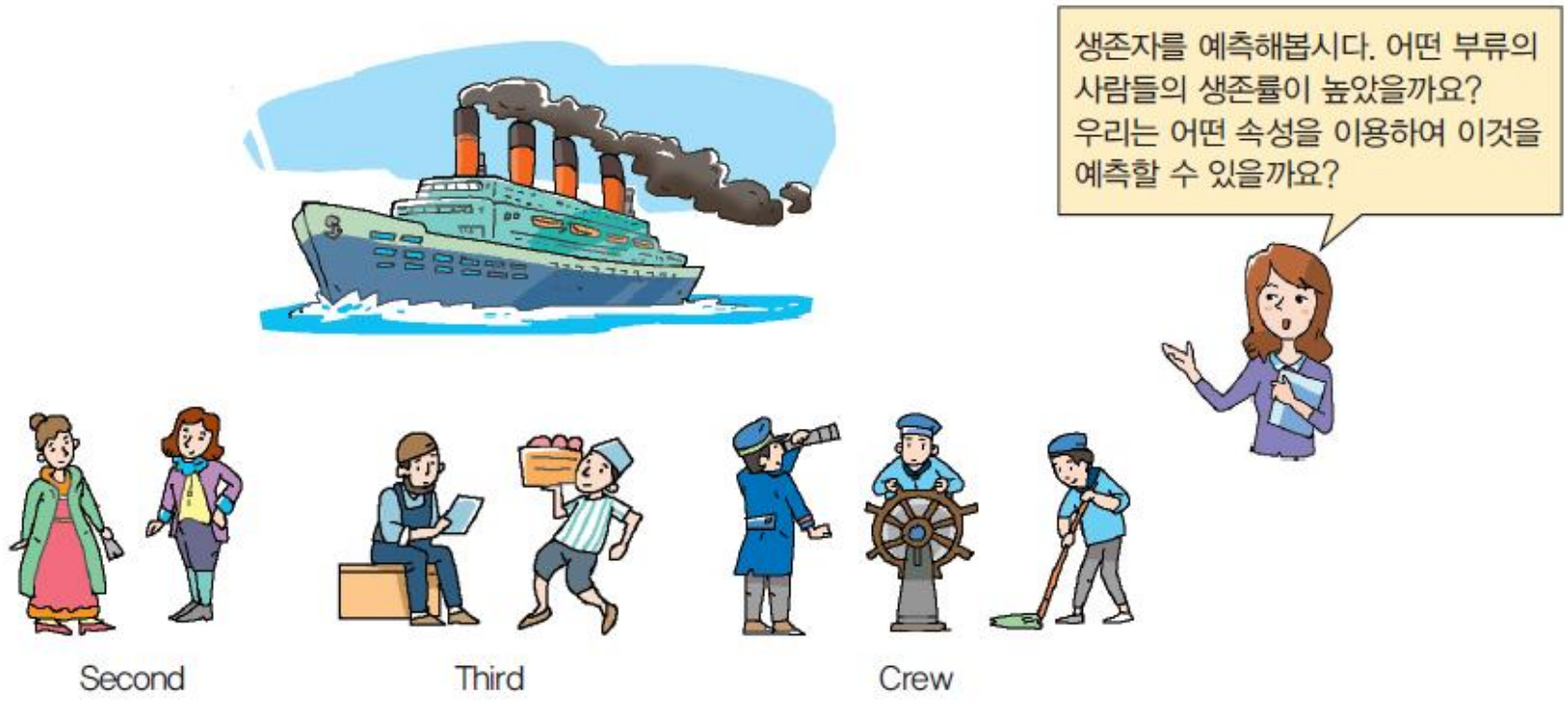


케라스 신경망 - 초음파 광물 예측

모델 저장과 재사용

```
model.fit(X_train, Y_train, epochs=130, batch_size=5)
model.save('my_model.h5')    # 모델을 컴퓨터에 저장


del model   # 테스트를 위해 메모리 내의 모델을 삭제
model = load_model('my_model.h5') # 모델을 새로 불러옴

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

케라스 신경망 실습 – 타이타닉 생존자 예측



케라스 신경망 실습 – 타이타닉 생존자 예측

PassengerId : 각 승객의 고유 번호
Survived : 생존 여부(종속 변수)
- 0 = 사망
- 1 = 생존

Pclass : 객실 등급 - 승객의 사회적, 경제적 지위
- 1st = Upper
- 2nd = Middle
- 3rd = Lower

Name : 이름
Sex : 성별
Age : 나이
SibSp : 동반한 Sibling(형제자매)와 Spouse(배우자)의 수
Parch : 동반한 Parent(부모) Child(자식)의 수
Ticket : 티켓의 고유넘버
Fare : 티켓의 요금
Cabin : 객실 번호
Embarked : 승선한 항
- C = Cherbourg
- Q = Queenstown
- S = Southampton

케라스 신경망 실습 – 타이타닉 생존자 예측


```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf
```

```
# 데이터 세트를 읽어들인다.
train = pd.read_csv("train.csv", sep=',')
test = pd.read_csv("test.csv", sep=',')

# 필요없는 컬럼을 삭제한다.
train.drop(['SibSp', 'Parch', 'Ticket', 'Embarked', 'Name',\
        'Cabin', 'PassengerId', 'Fare', 'Age'], inplace=True, axis=1)

# 결손치가 있는 데이터 행은 삭제한다.
train.dropna(inplace=True)
```

케라스 신경망 실습 – 타이타닉 생존자 예측


```
>>> train.drop(['SibSp', 'Parch', 'Ticket', 'Embarked', 'Name',\
        'Cabin', 'PassengerId', 'Fare', 'Age'], inplace=True, axis=1)


>>> train.head()
   Survived  Pclass     Sex
0         0       3    male
1         1       1  female
2         1       3  female
3         1       1  female
4         0       3    male
```

케라스 신경망 실습 – 타이타닉 생존자 예측


```
# 기호를 수치로 변환한다.
for ix in train.index:
    if train.loc[ix, 'Sex']=="male":
        train.loc[ix, 'Sex']=1
    else:
        train.loc[ix, 'Sex']=0

# 2차원 배열을 1차원 배열로 평탄화한다.
target = np.ravel(train.Survived)

# 생존여부를 학습 데이터에서 삭제한다.
train.drop(['Survived'], inplace=True, axis=1)
train = train.astype(float)     # 최근 소스에서는 float형태로 형변환하여야
```

컴퓨터는 숫자만 처리할 수 있다. 딥러닝은 0부터 1 사이의 실수만 처리 가능

교과서 소스에 추가

케라스 신경망 실습 – 타이타닉 생존자 예측


```
# 케라스 모델을 생성한다.
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(16, activation='relu', input_shape=(2,)))
model.add(tf.keras.layers.Dense(8, activation='relu'))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))

# 케라스 모델을 컴파일한다.
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
# 케라스 모델을 학습시킨다.
model.fit(train, target, epochs=30, batch_size=1, verbose=1)
```

케라스 신경망 실습 – 타이타닉 생존자 예측


```
...
Epoch 29/30
891/891 [==============================] - 1s 753us/sample - loss: 0.4591 - acc:
0.7677
Epoch 30/30
891/891 [==============================] - 1s 753us/sample - loss: 0.4547 - acc:
0.7789
```

약 78% 정확도

케라스 신경망 실습 – 타이타닉 생존자 예측


```
idx=test.shape[0]
pred=model.predict(test)
test_np=test.to_numpy()
for i in range(idx):
    print(test_np[i,:],pred[i])
```

객실 등급. 성별, 생존 확률

```
[3. 1.] [0.11963955]
[3. 0.] [0.42622244]
[2. 1.] [0.20062831]
[3. 1.] [0.11963955]
[3. 0.] [0.42622244]
[3. 1.] [0.11963955]
[3. 0.] [0.42622244]
[2. 1.] [0.20062831]
[3. 0.] [0.42622244]
[3. 1.] [0.11963955]
[3. 1.] [0.11963955]
[1. 1.] [0.35509673]
[1. 0.] [0.9667859]
[2. 1.] [0.20062831]
[1. 0.] [0.9667859]
[2. 0.] [0.84050065]
[2. 1.] [0.20062831]
[3. 1.] [0.11963955]
[3. 0.] [0.42622244]
[3. 0.] [0.42622244]
[1. 1.] [0.35509673]
[3. 1.] [0.11963955]
[1. 0.] [0.9667859]
[1. 1.] [0.35509673]
[1. 0.] [0.9667859]
[3. 1.] [0.11963955]
```

케라스 신경망 실습 – 타이타닉 생존자 예측

과제 titanic_test.ipynb test.csv, train.csv

모델을 다음과 같이 변경 하고 code (모델 생성 부분만)과 정확도(accuracy)를 제출 하시오.

1)

입력 층 : 변동 없음
은닉층 1 : 16 (relu)
은닉측 2 : 8 (relu)
은닉측 3 : 8 (relu)
출력 층 변동 없음.

2)

입력 층 : 변동 없음
은닉층 1 : 16 (relu)
은닉측 2 : 16 (relu)
은닉측 3 : 8 (relu)
출력 층 변동 없음.

```
model.add(tf.keras.layers.Dense(16, activation='relu', input_shape=(2,)))
model.add(tf.keras.layers.Dense(8, activation='relu'))
model.add(tf.keras.layers.Dense(8, activation='relu'))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

입력 층 : 2
은닉층 1 : 16 (relu)
은닉층 2 : 8 (relu)
은닉층 3 : 8 (relu)
출력 층 : 1

Model: "sequential_2"

Layer (type)	Output Shape	Param #
dense_6 (Dense)	(None, 16)	48
dense_7 (Dense)	(None, 8)	136
dense_8 (Dense)	(None, 8)	72
dense_9 (Dense)	(None, 1)	9

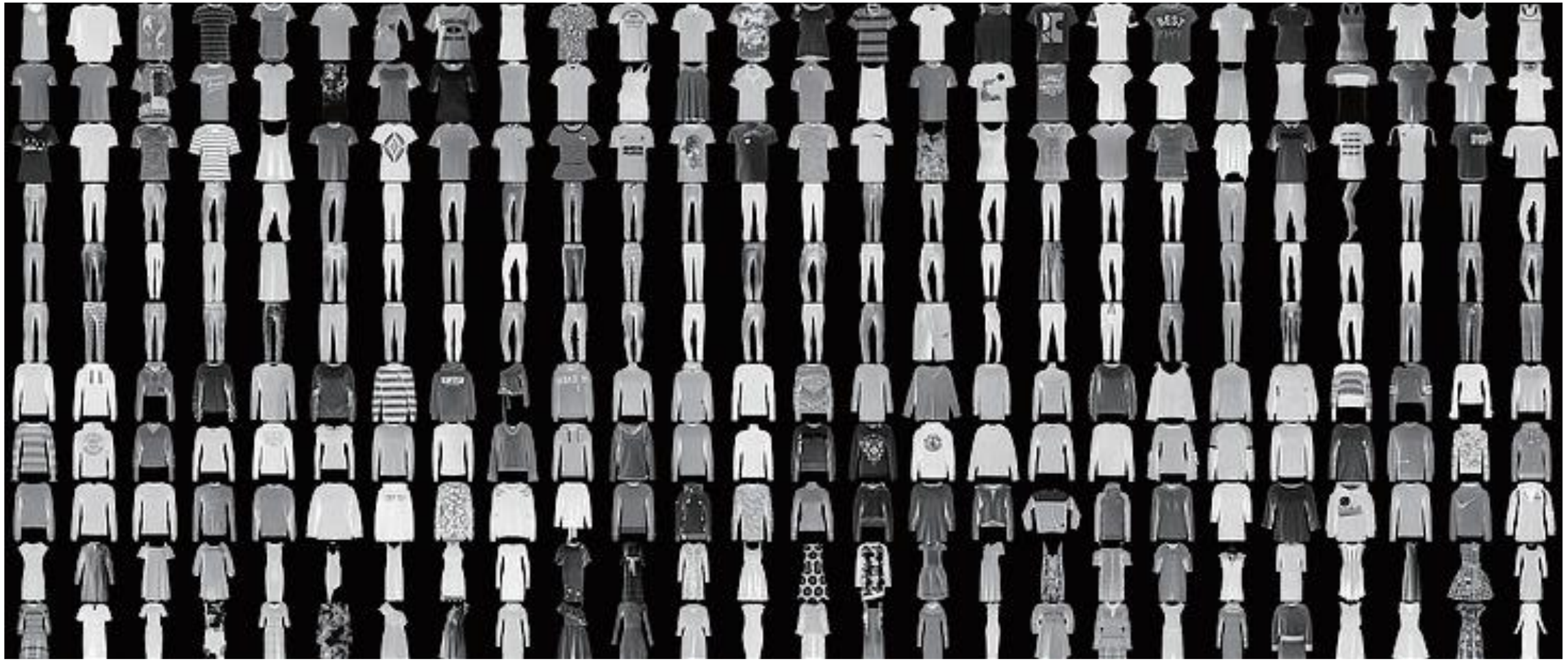
Total params: 265
Trainable params: 265
Non-trainable params: 0

```
model.add(Dense(30,  input_dim=12, activation='relu'))
model.add(Dense(12, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

입력 층 : 12
은닉층 1 : 30 (relu)
은닉층 2 : 12 (relu)
은닉층 3 : 8 (relu)
출력 층 : 1 (sigmoid)

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 30)                390

dense_1 (Dense)              (None, 12)                372

dense_2 (Dense)              (None, 8)                 104

dense_3 (Dense)              (None, 1)                 9

=================================================================
```

케라스 신경망 실습 – 패션 아이템 분류



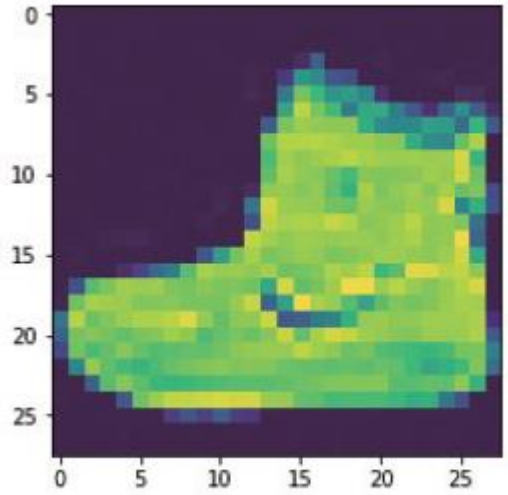
케라스 신경망 실습 – 패션 아이템 분류

- 이미지는 28x28 크기이고 픽셀 값은 0과 255 사이의 값이다. 레이블(label)은 0에서 9까지의 정수로서 패션 아이템의 범주를 나타낸다.

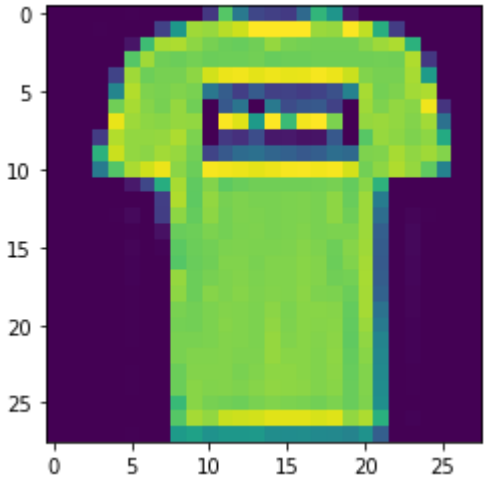
레이블	범주
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

케라스 신경망 실습 – 패션 아이템 분류

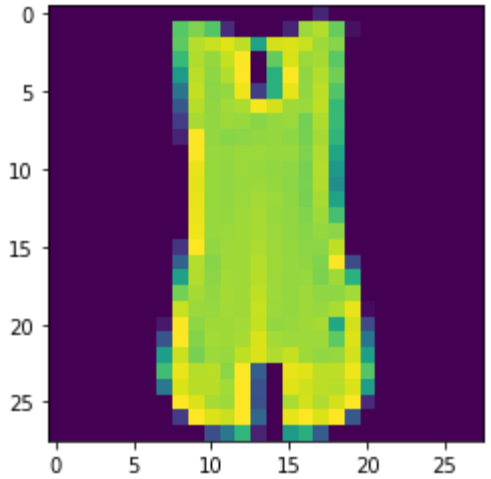
- **plt.imshow(train_images[0])**

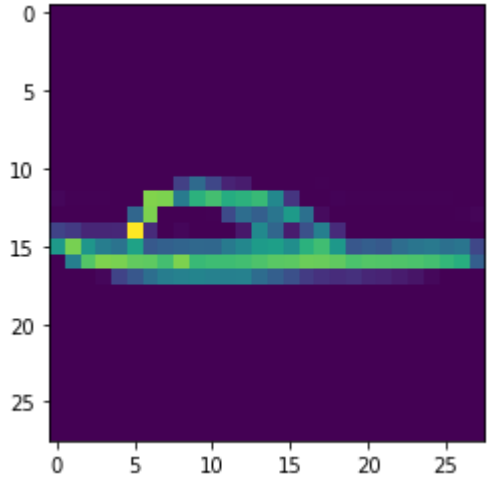


두번째



네번째





28x28 gray image

케라스 신경망 실습 – 패션 아이템 분류


```
import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras import datasets, layers, models

fashion_mnist = keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

plt.imshow(train_images[0])

train_images = train_images / 255.0
test_images = test_images / 255.0
```

케라스 신경망 실습 – 패션 아이템 분류


```
model = models.Sequential()
model.add(layers.Flatten(input_shape=(28, 28)))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

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

model.fit(train_images, train_labels, epochs=5)

test_loss, test_acc = model.evaluate(test_images, test_labels)
print('정확도:', test_acc)
```

케라스 신경망 실습 – 패션 아이템 분류

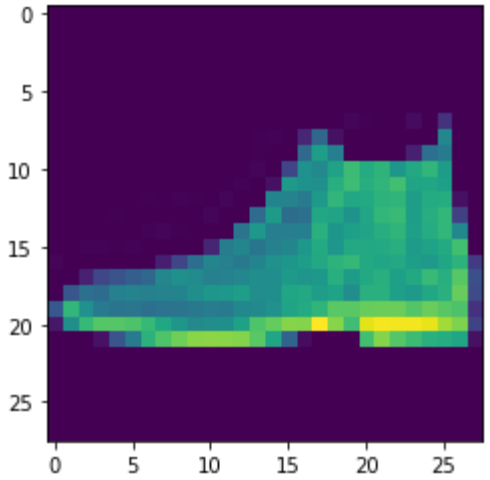

```
model = models.Sequential()
model.add(layers.Flatten(input_shape=(28, 28)))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

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

model.fit(train_images, train_labels, epochs=5)

test_loss, test_acc = model.evaluate(test_images, test_labels)
print('정확도:', test_acc)
```

```
10000/10000 [==============================] - 0s 32us/sample - loss: 0.3560 -
acc: 0.8701
정확도: 0.8701
```

케라스 신경망 실습 – 패션 아이템 분류


```
test_pred=model.predict(test_images)
plt.imshow(test_images[0])
print(np.round(test_pred[0],2))
```

```
[0.  0.  0.  0.  0.  0.  0.  0.04  0.  0.96]
```

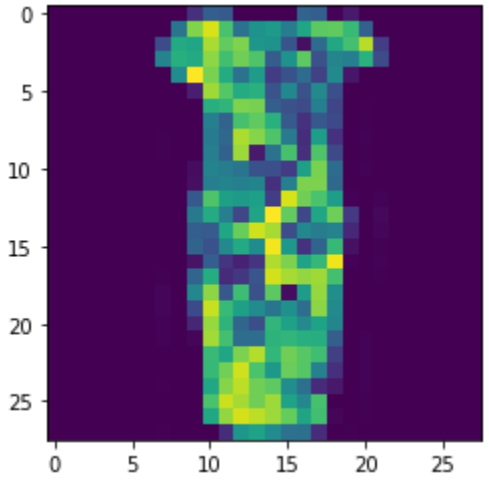


레이블	범주
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

케라스 신경망 실습 – 패션 아이템 분류


```
plt.imshow(test_images[100])
print(np.round(test_pred[100],2))
```

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

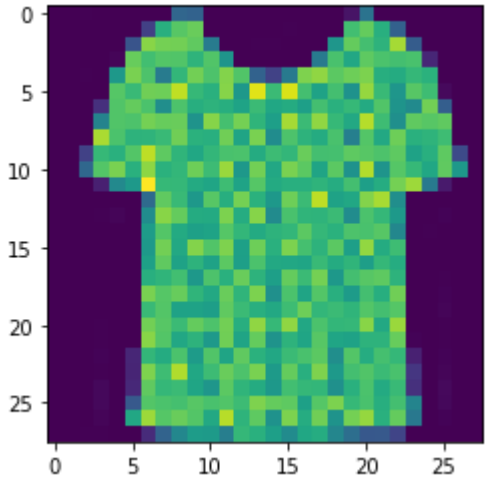


레이블	범주
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

케라스 신경망 실습 – 패션 아이템 분류


```
plt.imshow(test_images[1000])
print(np.round(test_pred[1000],2))
```

```
[0.43 0.   0.05 0.02 0.   0.   0.51 0.   0.   0.  ]
```

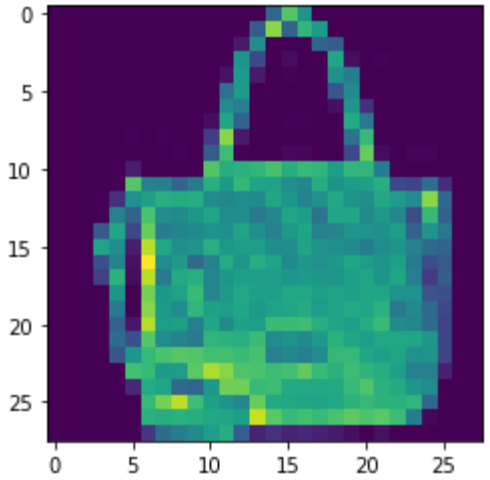


레이블	범주
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

케라스 신경망 실습 – 패션 아이템 분류


```
plt.imshow(test_images[2000])
print(np.round(test_pred[2000],2))
```

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



레이블	범주
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

케라스 신경망 실습 – 패션 아이템 분류

과제 fasion_org_exam.ipynb

모델을 다음과 같이 변경 하고 code (모델 생성 부분만)과 정확도를 제출 하시오.

1)

입력 층 : 변동 없음
은닉층 1 : 32 (relu)
출력 층 변동 없음.

2)

입력 층 : 변동 없음
은닉층 1 : 64 (relu)
은닉측 2 : 16 (relu)
출력 층 변동 없음.

3) 1050번째 test이미지의 사진을 붙이고 카테고리 예측 결과를 제출 하시오
4) 615번째 test 이미지와 사진을 붙이고 카테고리 예측 결과를 제출 하시오.

```
model.add(layers.Flatten(input_shape=(28, 28)))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
model.summary()
```

입력 층 : 28 *28
은닉층 1 : 128 (relu)
출력 층 : 10

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 flatten (Flatten)            (None, 784)               0

 dense (Dense)                (None, 128)               100480

 dense_1 (Dense)              (None, 10)                1290

=================================================================
=================================================================
```

```
model.add(layers.Flatten(input_shape=(28, 28)))
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

입력 층 : 28 *28
은닉층 1 : 32 (relu)
은닉층 2 : 16 (relu)
출력 층 : 10

Layer (type)	Output Shape	Param #
flatten_1 (Flatten)	(None, 784)	0
dense_2 (Dense)	(None, 32)	25120
dense_3 (Dense)	(None, 16)	528
dense_4 (Dense)	(None, 10)	170

케라스 신경망 실습 – 와인 종류 예측 하기

- **데이터의 확인과 검증셋**
  - 먼저 데이터를 불러와 대략적인 구조를 살펴보자

```
import pandas as pd

# 깃허브에 준비된 데이터를 가져옵니다.
!git clone https://github.com/taehojo/data.git

# 와인 데이터를 불러옵니다.
df = pd.read_csv('./data/wine.csv', header=None)

# 데이터를 미리 보겠습니다.
df
```

케라스 신경망 실습 – 와인 종류 예측 하기

실행 결과

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5	1
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5	1
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	5	1
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	6	1
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
6492	6.2	0.21	0.29	1.6	0.039	24.0	92.0	0.99114	3.27	0.50	11.2	6	0
6493	6.6	0.32	0.36	8.0	0.047	57.0	168.0	0.99490	3.15	0.46	9.6	5	0
6494	6.5	0.24	0.19	1.2	0.041	30.0	111.0	0.99254	2.99	0.46	9.4	6	0
6495	5.5	0.29	0.30	1.1	0.022	20.0	110.0	0.98869	3.34	0.38	12.8	7	0
6496	6.0	0.21	0.38	0.8	0.020	22.0	98.0	0.98941	3.26	0.32	11.8	6	0

6497 rows × 13 columns

케라스 신경망 실습 – 와인 종류 예측 하기

- **데이터의 확인과 검증셋**
  - 샘플이 전체 6,497개 있음
  - 모두 속성이 12개 기록되어 있고 13번째 열에 클래스가 준비되어 있음
  - 각 속성에 대한 정보는 다음과 같음

0	주석산 농도	7	밀도
1	아세트산 농도	8	pH
2	구연산 농도	9	황산칼륨 농도
3	잔류 당분 농도	10	알코올 도수
4	염화나트륨 농도	11	와인의 맛(0~10등급)
5	유리 아황산 농도	12	클래스(1: 레드 와인, 0: 화이트 와인)
6	총 아황산 농도		

케라스 신경망 실습 – 와인 종류 예측 하기

- **데이터의 확인과 검증셋**
  - 0~11번째 열에 해당하는 속성 12개를 X로, 13번째 열을 y로 정하겠음

```
X = df.iloc[:,0:12]
y = df.iloc[:,12]
```

- **데이터의 확인과 검증셋**
  - 이제 딥러닝을 실행할 차례
  - 앞서 우리는 학습셋과 테스트셋을 나누는 방법에 대해 알아보았음
  - 이 장에서는 여기에 검증셋을 더해 보자

케라스 신경망 실습 – 와인 종류 예측 하기

▼ 그림 14-1 | 학습셋, 테스트셋, 검증셋

학습 셋 : training set
검증 셋: validation set
테스트 셋: test set

케라스 신경망 실습 – 와인 종류 예측 하기

- **데이터의 확인과 검증셋**
  - 학습이 끝난 모델을 테스트해 보는 것이 테스트셋의 목적이라면, 최적의 학습 파라미터를 찾기 위해 학습 과정에서 사용하는 것이 검증셋
  - 검증셋을 설정하면 검증셋에 테스트한 결과를 추적하면서 최적의 모델을 만들 수 있음
  - 검증셋은 model.fit() 함수 안에 validation_split이라는 옵션을 주면 만들어짐
  - 그림 14-1과 같이 전체의 80%를 학습셋으로 만들고 이 중 25%를 검증셋으로 하면 학습셋:검증셋:테스트셋의 비율이 60:20:20이 됨

케라스 신경망 실습 – 와인 종류 예측 하기

- **데이터의 확인과 검증셋**
  - 전체 코드를 실행하면 다음과 같음

실습 | 와인의 종류 예측하기: 데이터 확인과 실행

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split

# 깃허브에 준비된 데이터를 가져옵니다.
!git clone https://github.com/taehojo/data.git

# 와인 데이터를 불러옵니다.
df = pd.read_csv('./data/wine.csv', header=None)
```

케라스 신경망 실습 – 와인 종류 예측 하기

- **데이터의 확인과 검증셋**

```
# 와인의 속성을 X로, 와인의 분류를 y로 저장합니다.
X = df.iloc[:,0:12]
y = df.iloc[:,12]

# 학습셋과 테스트셋으로 나눕니다.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
shuffle=True)

# 모델 구조를 설정합니다.
model = Sequential()
```

케라스 신경망 실습 – 와인 종류 예측 하기

- **데이터의 확인과 검증셋**

```
model.add(Dense(30, input_dim=12, activation='relu'))
model.add(Dense(12, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()

# 모델을 컴파일합니다.
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])

# 모델을 실행합니다.
history = model.fit(X_train, y_train, epochs=50, batch_size=500,
validation_split=0.25) # 0.8 x 0.25 = 0.2
```

케라스 신경망 실습 – 와인 종류 예측 하기

- **데이터의 확인과 검증셋**

```
# 테스트 결과를 출력합니다.
score = model.evaluate(X_test, y_test)
print('Test accuracy:', score[1])
```

케라스 신경망 실습 – 와인 종류 예측 하기

- **데이터의 확인과 검증셋**

실행 결과

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 30)                390
_________________________________________________________________
dense_1 (Dense)              (None, 12)                372
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 104
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 9
=================================================================
Total params: 875
```

케라스 신경망 실습 – 와인 종류 예측 하기

- **데이터의 확인과 검증셋**

```
Trainable params: 875
Non-trainable params: 0
_________________________________________________________________

Epoch 1/50
8/8 [==============================] - 1s 23ms/step - loss: 2.9423 - accu
racy: 0.7519 - val_loss: 2.2360 - val_accuracy: 0.7562
... (중략) ...
Epoch 50/50
8/8 [==============================] - 0s 6ms/step - loss: 0.1161 - accura
cy: 0.9574 - val_loss: 0.1523 - val_accuracy: 0.9500
```

케라스 신경망 실습 – 와인 종류 예측 하기

- **데이터의 확인과 검증셋**

```
41/41 [==============================] - 0s 1ms/step - loss: 0.1438 - accu
racy: 0.9415
Test accuracy: 0.9415384531021118
```

과적합 판단

- **그래프로 과적합 확인하기**
  - 역전파를 50번 반복하면서 학습을 진행
  - 과연 이 반복 횟수는 적절했을까?
  - 학습의 반복 횟수가 너무 적으면 데이터셋의 패턴을 충분히 파악하지 못함
  - 학습을 너무 많이 반복하는 것도 좋지 않음
  - 너무 과한 학습은 13.2절에서 이야기한 바 있는 과적합 현상을 불러오기 때문임
  - 적절한 학습 횟수를 정하기 위해서는 검증셋과 테스트셋의 결과를 그래프로 보는 것이 가장 좋음
  - 이를 확인하기 위해 학습을 길게 실행해 보고 결과를 알아보자

케라스 신경망 실습 – 와인 종류 예측 하기

실습 | 와인의 종류 예측하기: 그래프 표현

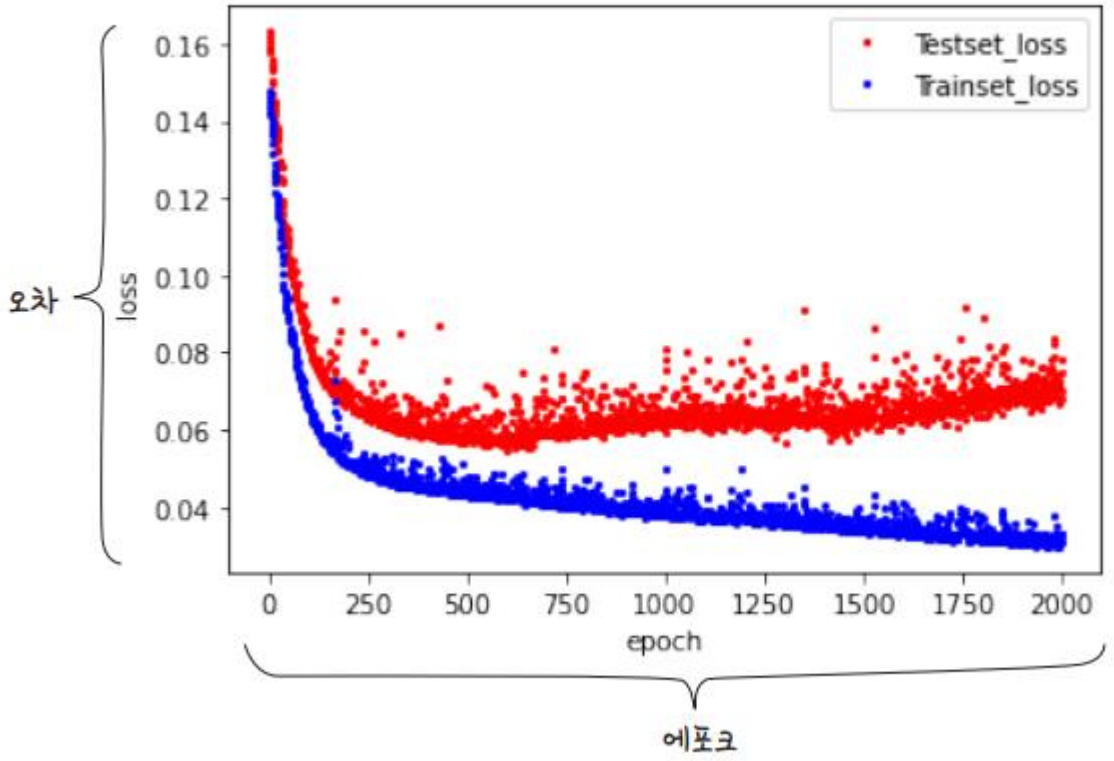
```
# 그래프 확인을 위한 긴 학습(컴퓨터 환경에 따라 시간이 다소 걸릴 수 있습니다)
history = model.fit(X_train, y_train, epochs=2000, batch_size=500,
validation_split=0.25)

# history에 저장된 학습 결과를 확인해 보겠습니다.
hist_df = pd.DataFrame(history.history)
hist_df

# y_vloss에 테스트셋의 오차를 저장합니다.
y_vloss = hist_df['val_loss']
```

케라스 신경망 실습 – 와인 종류 예측 하기


```
# y_loss에 학습셋의 오차를 저장합니다.
y_loss = hist_df['loss']

# x 값을 지정하고 테스트셋의 오차를 빨간색으로, 학습셋의 오차를 파란색으로 표시합니다.
x_len = np.arange(len(y_loss))
plt.plot(x_len, y_vloss, "o", c="red", markersize=2, label='Testset_loss')
plt.plot(x_len, y_loss, "o", c="blue", markersize=2, label='Trainset_loss')

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

케라스 신경망 실습 – 와인 종류 예측 하기

실행 결과



케라스 신경망 실습 – 와인 종류 예측 하기

- **그래프로 과적합 확인하기**
  - 그래프의 형태는 실행에 따라 조금씩 다를 수 있지만 대략 그림 14-2와 같은 그래프가 나옴
  - 우리가 눈여겨보아야 할 부분은 학습이 오래 진행될수록 검증셋의 오차(파란색)는 줄어들지만 테스트셋의 오차(빨간색)는 다시 커진다는 것
  - 이는 과도한 학습으로 과적합이 발생했기 때문임
  - 이러한 사실을 통해 알 수 있는 것은 검증셋 오차가 커지기 직전까지 학습한 모델이 최적의 횟수로 학습한 모델이라는 것
  - 이제 검증셋의 오차가 커지기 전에 학습을 자동으로 중단시키고, 그때의 모델을 저장하는 방법을 알아보자

케라스 신경망 실습 – 와인 종류 예측 하기

- **데이터의 확인과 검증셋**
  - 먼저 세 개의 은닉층을 만들고 각각 30개, 12개, 8개의 노드를 만들었음
  - 50번을 반복했을 때 정확도가 94.15%로 나왔음
  - 꽤 높은 정확도
  - 이것이 과연 최적의 결과일까?
  - 이제 여기에 여러 옵션을 더해 가면서 더 나은 모델을 만들어 가는 방법을 알아보자

케라스 신경망 실습 – 와인 종류 예측 하기

- **학습의 자동 중단**
  - 텐서플로에 포함된 케라스 API는 EarlyStopping() 함수를 제공
  - 학습이 진행되어도 테스트셋 오차가 줄어들지 않으면 학습을 자동으로 멈추게 하는 함수
  - 이를 조금 전 배운 ModelCheckpoint() 함수와 함께 사용해 보면
  -

```
from tensorflow.keras.callbacks import EarlyStopping

early_stopping_callback = EarlyStopping(monitor='val_loss', patience=20)
```

케라스 신경망 실습 – 와인 종류 예측 하기

- **학습의 자동 중단**
  - monitor 옵션은 model.fit()의 실행 결과 중 어떤 것을 이용할지 정함
  - 검증셋의 오차(val_loss)로 지정
  - patience 옵션은 지정된 값이 몇 번 이상 향상되지 않으면 학습을 종료시킬지 정함
  - monitor='val_loss', patience=20이라고 지정하면 검증셋의 오차가 20번 이상 낮아지지 않을 경우 학습을 종료하라는 의미

케라스 신경망 실습 – 와인 종류 예측 하기

- **학습의 자동 중단**
  - 모델 저장에 관한 설정은 앞 절에서 사용한 내용을 그대로 따르겠음
  - 다만 이번에는 최고의 모델 하나만 저장되게끔 해 보자

```
modelpath = "./data/model/Ch14-4-bestmodel.hdf5"

checkpointer = ModelCheckpoint(filepath=modelpath, monitor='val_loss',
verbose = 0, save_best_only=True)
```

4 학습의 자동 중단

- **학습의 자동 중단**
  - 모델을 실행
  - 자동으로 최적의 에포크를 찾아 멈출 예정이므로 epochs는 넉넉하게 설정

```
history = model.fit(X_train, y_train, epochs=2000, batch_size=500,
validation_split=0.25, verbose=1, callbacks=[early_stopping_callback,
checkpointer])
```

4 학습의 자동 중단

- **학습의 자동 중단**
  - 모델을 실행
  - 자동으로 최적의 에포크를 찾아 멈출 예정이므로 epochs는 넉넉하게 설정

```
history = model.fit(X_train, y_train, epochs=2000, batch_size=500,
validation_split=0.25, verbose=1, callbacks=[early_stopping_callback,
checkpointer])
```

케라스 신경망 실습 – 와인 종류 예측 하기

- **학습의 자동 중단**
  - 앞서 만든 기본 코드에 다음과 같이 새로운 코드를 불러와 덧붙

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

# 학습이 언제 자동 중단될지 설정합니다.
early_stopping_callback = EarlyStopping(monitor='val_loss', patience=20)

# 최적화 모델이 저장될 폴더와 모델 이름을 정합니다.
modelpath = "./data/model/Ch14-4-bestmodel.hdf5"

# 최적화 모델을 업데이트하고 저장합니다.
checkpointer = ModelCheckpoint(filepath=modelpath, monitor='val_loss',
verbose=0, save_best_only=True)
```

케라스 신경망 실습 – 와인 종류 예측 하기

- **학습의 자동 중단**

```
# 모델을 실행합니다.
history = model.fit(X_train, y_train, epochs=2000, batch_size=500,
validation_split=0.25, verbose=1, callbacks=[early_stopping_callback,
checkpointer])
```

케라스 신경망 실습 – 와인 종류 예측 하기

- **학습의 자동 중단**

```
# 모델을 실행합니다.
history = model.fit(X_train, y_train, epochs=2000, batch_size=500,
validation_split=0.25, verbose=1, callbacks=[early_stopping_callback,
checkpointer])
```

케라스 신경망 실습 – 와인 종류 예측 하기


```
1/8 [==>.....] - ETA: 0s - loss: 0.1462 - accuracy: 0.9540
Epoch 31: val_loss improved from 0.16765 to 0.16567, saving model to ./data/model/Ch14-4-bestmodel.hdf5
8/8 [==============================] - 0s 9ms/step - loss: 0.1712 - accuracy: 0.9430 - val_loss: 0.1657 - val_accuracy: 0.9392
Epoch 32/2000
1/8 [==>.....] - ETA: 0s - loss: 0.2067 - accuracy: 0.9400
Epoch 32: val_loss did not improve from 0.16567
8/8 [==============================] - 0s 7ms/step - loss: 0.1678 - accuracy: 0.9430 - val_loss: 0.1675 - val_accuracy: 0.9346
Epoch 33/2000
1/8 [==>.....] - ETA: 0s - loss: 0.1330 - accuracy: 0.9560
Epoch 33: val_loss did not improve from 0.16567
8/8 [==============================] - 0s 6ms/step - loss: 0.1661 - accuracy: 0.9438 - val_loss: 0.1691 - val_accuracy: 0.9431
Epoch 34/2000
1/8 [==>.....] - ETA: 0s - loss: 0.1694 - accuracy: 0.9400
Epoch 34: val_loss improved from 0.16567 to 0.15903, saving model to ./data/model/Ch14-4-bestmodel.hdf5
8/8 [==============================] - 0s 9ms/step - loss: 0.1694 - accuracy: 0.9418 - val_loss: 0.1590 - val_accuracy: 0.9362
Epoch 35/2000
1/8 [==>.....] - ETA: 0s - loss: 0.1949 - accuracy: 0.9380
Epoch 35: val_loss improved from 0.15903 to 0.15821, saving model to ./data/model/Ch14-4-bestmodel.hdf5
8/8 [==============================] - 0s 9ms/step - loss: 0.1628 - accuracy: 0.9464 - val_loss: 0.1582 - val_accuracy: 0.9369
Epoch 36/2000
1/8 [==>.....] - ETA: 0s - loss: 0.1485 - accuracy: 0.9360
```

케라스 신경망 실습 – 와인 종류 예측 하기

과제 wine.ipynb, wine.csv

wine.ipynb 를 실행 하고 1) model이 마지막으로 저장 되는 부분과
2) 학습이 종료되는 부분의 출력 결과를 붙이시오
3) test결과의 accuracy를 적으시오.

케라스 신경망 실습 – 주택 가격 예측

- **실제 데이터로 만들어 보는 모델**
  - 지금까지 한 실습은 참 또는 거짓을 맞히거나 여러 개의 보기 중 하나를 예측하는 분류 문제
  - 이번에는 수치를 예측하는 문제
  - 준비된 데이터는 아이오와주 에임스 지역에서 2006년부터 2010년까지 거래된 실제 부동산 판매 기록
  - 주거 유형, 차고, 자재 및 환경에 관한 80개의 서로 다른 속성을 이용해 집의 가격을 예측해 볼 예정
  - 오랜 시간 사람이 일일이 기록하다 보니 빠진 부분도 많고, 집에 따라 어떤 항목은 범위에서 너무 벗어나 있기도 하며, 또 가격과는 관계가 없는 정보가 포함되어 있기도 함
  - 실제 현장에서 만나게 되는 이런 류의 데이터를 어떻게 다루어야 하는지 이 장에서 학습해 보자

케라스 신경망 실습 – 주택 가격 예측

- **데이터 파악하기**
  - 먼저 데이터를 불러와 확인해 보자

```
import pandas as pd

# 깃허브에 준비된 데이터를 가져옵니다.
!git clone https://github.com/taehojo/data.git

# 집 값 데이터를 불러옵니다.
df = pd.read_csv("./data/house_train.csv")
```

1 데이터 파악하기

```
import pandas as pd

# 깃허브에 준비된 데이터를 가져옵니다.
!git clone https://github.com/taehojo/data.git

# 집 값 데이터를 불러옵니다.
df = pd.read_csv("./data/house_train.csv")
```

케라스 신경망 실습 – 주택 가격 예측

실행 결과

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	..
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...
...	...	...	...	...	...	...	...	...	...	...	...
1455	1456	60	RL	62.0	7917	Pave	NaN	Reg	Lvl	AllPub	...
1456	1457	20	RL	85.0	13175	Pave	NaN	Reg	Lvl	AllPub	...
1457	1458	70	RL	66.0	9042	Pave	NaN	Reg	Lvl	AllPub	...
1458	1459	20	RL	68.0	9717	Pave	NaN	Reg	Lvl	AllPub	...
1459	1460	20	RL	75.0	9937	Pave	NaN	Reg	Lvl	AllPub	...

PoolArea	PoolQC	Fence	MiscFeature	MiscVal	MoSold	YrSold	SaleType	SaleCondition	SalePrice
0	NaN	NaN	NaN	0	2	2008	WD	Normal	208500
0	NaN	NaN	NaN	0	5	2007	WD	Normal	181500
0	NaN	NaN	NaN	0	9	2008	WD	Normal	223500
0	NaN	NaN	NaN	0	2	2006	WD	Abnorml	140000
0	NaN	NaN	NaN	0	12	2008	WD	Normal	250000
...	...	...	...	...	...	...	...	...	...
0	NaN	NaN	NaN	0	8	2007	WD	Normal	175000
0	NaN	MnPrv	NaN	0	2	2010	WD	Normal	210000
0	NaN	GdPrv	Shed	2500	5	2010	WD	Normal	266500
0	NaN	NaN	NaN	0	4	2010	WD	Normal	142125
0	NaN	NaN	NaN	0	6	2008	WD	Normal	147500

1460 rows × 81 columns

케라스 신경망 실습 – 주택 가격 예측

- **데이터 파악하기**
  - 총 80개의 속성으로 이루어져 있고 마지막 열이 우리의 타깃인 집 값(SalePrice)
  - 모두 1,460개의 샘플이 들어 있음

케라스 신경망 실습 – 주택 가격 예측

- **주택 가격 예측 모델**
  - 이제 앞서 구한 중요 속성을 이용해 학습셋과 테스트셋을 만들어 보자
  - 집 값을 y로, 나머지 열을 X_train_pre로 저장한 후 전체의 80%를 학습셋으로, 20%를 테스트셋으로 지정

```
cols_train = ['OverallQual','GrLivArea','GarageCars','GarageArea','TotalB
smtSF']
X_train_pre = df[cols_train]
y = df['SalePrice'].values
X_train, X_test, y_train, y_test = train_test_split(X_train_pre, y, test_
size=0.2)
```

케라스 신경망 실습 – 주택 가격 예측

- **주택 가격 예측 모델**
  - 모델의 구조와 실행 옵션을 설정
  - 입력될 속성의 개수를 X_train.shape[1]로 지정해 자동으로 세도록 했음

```
model = Sequential()
model.add(Dense(10, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(30, activation='relu'))
model.add(Dense(40, activation='relu'))
model.add(Dense(1))
model.summary()
```

케라스 신경망 실습 – 주택 가격 예측

- **주택 가격 예측 모델**
  - 실행에서 달라진 점은 손실 함수
  - 선형 회귀이므로 평균 제곱 오차(mean_squared_error)를 적음

```
model.compile(optimizer='adam', loss='mean_squared_error')
```

케라스 신경망 실습 – 주택 가격 예측

- **주택 가격 예측 모델**
  - 20번 이상 결과가 향상되지 않으면 자동으로 중단되게끔 함
  - 저장될 모델 이름을 'Ch15-house.hdf5'로 정함
  - 모델은 차후 '22장. 캐글로 시작하는 새로운 도전'에서 다시 사용(검증셋을 추가하고 싶을 경우 앞서와 마찬가지로 학습셋, 검증셋, 테스트셋의 비율을 각각 60%, 20%, 20%로 정하면 됨)

```
early_stopping_callback = EarlyStopping(monitor='val_loss', patience=20)

modelpath = "./data/model/Ch15-house.hdf5"

checkpointer = ModelCheckpoint(filepath=modelpath, monitor='val_loss',
verbose=0, save_best_only=True)

history = model.fit(X_train, y_train, validation_split=0.25, epochs=2000,
batch_size=32, callbacks=[early_stopping_callback,checkpointer])
```

케라스 신경망 실습 – 주택 가격 예측

- **주택 가격 예측 모델**
  - 모든 코드를 실행하면 다음과 같음

실습 | 주택 가격 예측하기

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split

import matplotlib.pyplot as plt
import seaborn as sns

import pandas as pd
import numpy as np
```

케라스 신경망 실습 – 주택 가격 예측


```
!git clone https://github.com/taehojo/data.git

# 집 값 데이터를 불러옵니다.
df = pd.read_csv("./data/house_train.csv")

# 카테고리형 변수를 0과 1로 이루어진 변수로 바꾸어 줍니다.
df = pd.get_dummies(df)

# 결측치를 전체 칼럼의 평균으로 대체해 채워 줍니다.
df = df.fillna(df.mean())
```

케라스 신경망 실습 – 주택 가격 예측


```
# 데이터 사이의 상관관계를 저장합니다.
df_corr = df.corr()

# 집 값과 관련이 큰 것부터 순서대로 저장합니다.
df_corr_sort = df_corr.sort_values('SalePrice', ascending=False)

# 집 값을 제외한 나머지 열을 저장합니다.
cols_train = ['OverallQual','GrLivArea','GarageCars','GarageArea','TotalB
smtSF']
X_train_pre = df[cols_train]

# 집 값을 저장합니다.
y = df['SalePrice'].values
```

케라스 신경망 실습 – 주택 가격 예측


```
# 전체의 80%를 학습셋으로, 20%를 테스트셋으로 지정합니다.
X_train, X_test, y_train, y_test = train_test_split(X_train_pre, y, test_
size=0.2)

# 모델의 구조를 설정합니다.
model = Sequential()
model.add(Dense(10, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(30, activation='relu'))
model.add(Dense(40, activation='relu'))
model.add(Dense(1))
model.summary()
```

케라스 신경망 실습 – 주택 가격 예측


```
# 모델을 실행합니다.
model.compile(optimizer='adam', loss='mean_squared_error')

# 20번 이상 결과가 향상되지 않으면 자동으로 중단되게끔 합니다.
early_stopping_callback = EarlyStopping(monitor='val_loss', patience=20)

# 모델의 이름을 정합니다.
modelpath = "./data/model/Ch15-house.hdf5"

# 최적화 모델을 업데이트하고 저장합니다.
checkpointer = ModelCheckpoint(filepath=modelpath, monitor='val_loss',
verbose=0, save_best_only=True)

# 실행 관련 설정을 하는 부분입니다. 전체의 20%를 검증셋으로 설정합니다.
history = model.fit(X_train, y_train, validation_split=0.25, epochs=2000,
batch_size=32, callbacks=[early_stopping_callback,checkpointer])
```

케라스 신경망 실습 – 주택 가격 예측

- **주택 가격 예측 모델**

실행 결과

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 10)                60
_________________________________________________________________
dense_1 (Dense)              (None, 30)                330
_________________________________________________________________
dense_2 (Dense)              (None, 40)                1240
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 41
=================================================================
```

케라스 신경망 실습 – 주택 가격 예측

- **주택 가격 예측 모델**

```
Total params: 1,671
Trainable params: 1,671
Non-trainable params: 0
_________________________________________________________________
Epoch 1/2000
28/28 [==============================] - 0s 5ms/step - loss:
39256875008.0000 - val_loss: 38050066432.0000
... (중략) ...
Epoch 145/2000
28/28 [==============================] - 0s 2ms/step - loss:
1962943104.0000 - val_loss: 2011970944.0000
```

145번째에서 학습이 중단

케라스 신경망 실습 – 주택 가격 예측

- **주택 가격 예측 모델**
  - 학습 결과를 시각화하기 위해 예측 값과 실제 값, 실행 번호가 들어갈 빈 리스트를 만들고 25개의 샘플로부터 얻은 결과를 채워 넣겠음

```
real_prices = []
pred_prices = []
X_num = []

n_iter = 0
Y_prediction = model.predict(X_test).flatten()
for i in range(25):
    real = y_test[i]
    prediction = Y_prediction[i]
    print("실제가격: {:.2f}, 예상가격: {:.2f}".format(real, prediction))
    real_prices.append(real)
```

케라스 신경망 실습 – 주택 가격 예측


```
    pred_prices.append(prediction)
    n_iter = n_iter + 1
    X_num.append(n_iter)
```

케라스 신경망 실습 – 주택 가격 예측

- **주택 가격 예측 모델**

실행 결과

```
실제가격: 262500.00, 예상가격: 240051.36
실제가격: 78000.00, 예상가격: 118369.56
... (중략) ...
실제가격: 127000.00, 예상가격: 116693.46
실제가격: 485000.00, 예상가격: 357789.88
```

케라스 신경망 실습 – 주택 가격 예측

실행 결과

케라스 신경망 실습 – 주택 가격 예측

과제 (house.ipynb,house_train.csv)

1) 아래와 같이 모델과 학습 방법을 변경 한 후 모델을 house1.hdf5로 저장 하시오.

입력 층 : 변동 없음
은닉층 1 : 20 (relu)
은닉측 2 : 20 (relu)
은닉측 3 : 40 (relu)
출력 층 변동 없음.

17회 이상 결과가 향상 되지 않으면 자동으로 학습이 중단되게
Batch size 16

2) 아래와 같이 모델과 학습 방법을 변경 한 후 모델을 house2.hdf5로 저장 하시오.

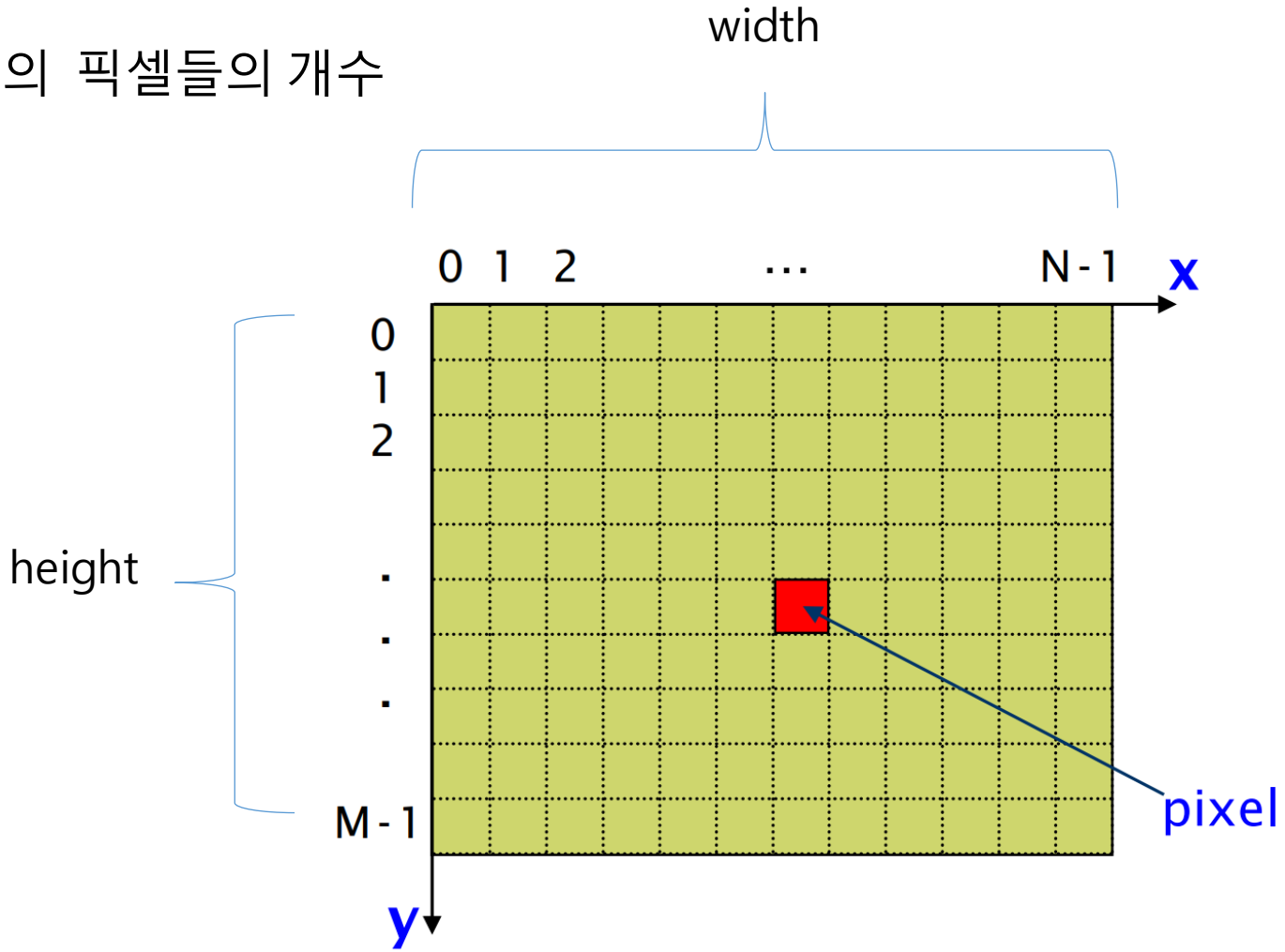
입력 층 : 변동 없음
은닉층 1 : 10 (relu)
은닉측 2 : 20 (relu)
은닉측 3 : 20 (relu)
출력 층 변동 없음.

15회 이상 결과가 향상 되지 않으면 자동으로 학습이 중단되게
Batch size 20

code (.ppt에 직접 붙여도 되고 .py 파일로 제출 가능) 와 model 제출 (house1.hdf5, hoise2.hdf5) 제출

영상 처리 기초

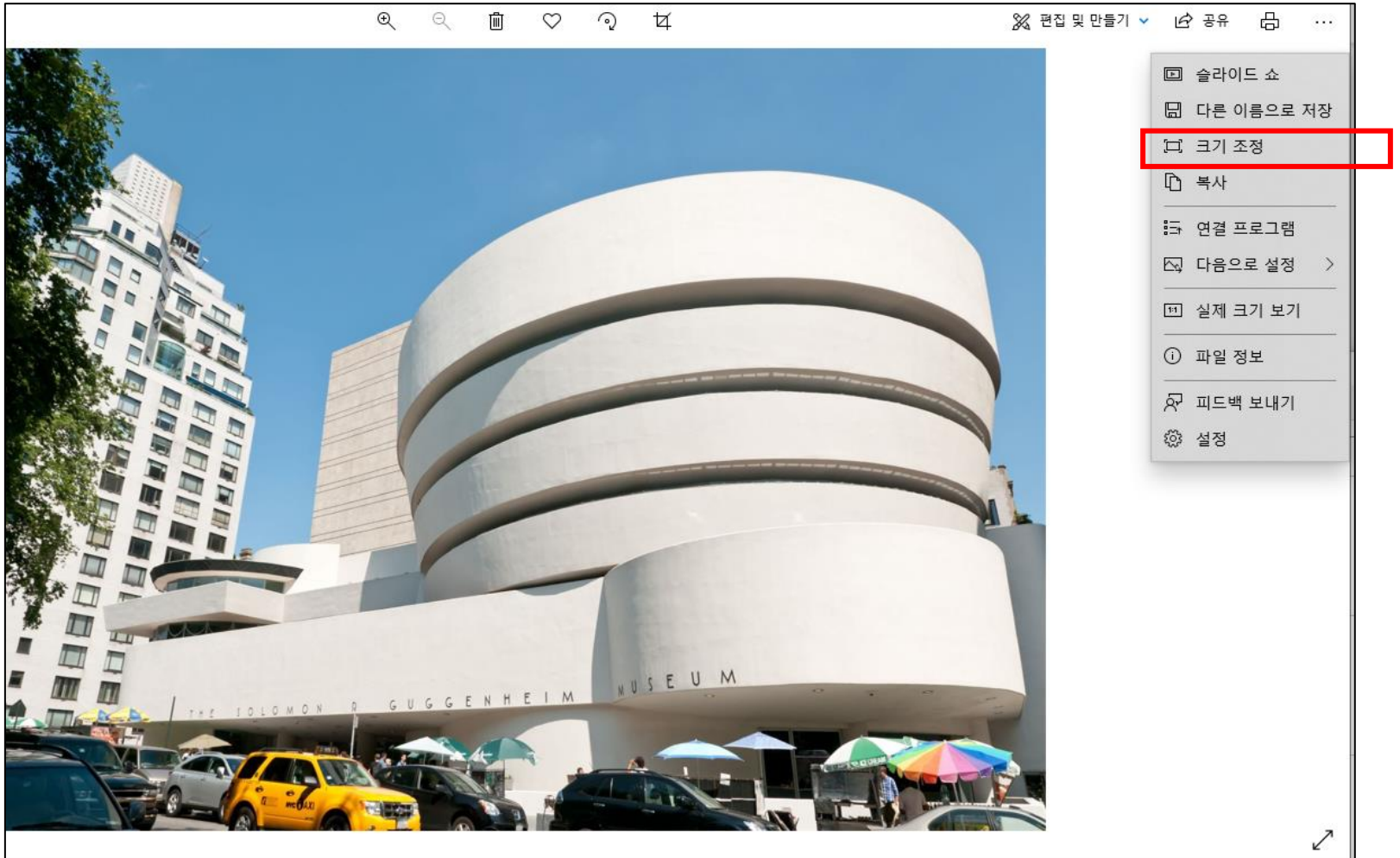
- **영상은 픽셀로 이루어짐**
- **해상도 (Resolution)**
  - 2차원 공간 영역에서, x,y 축 방향의 픽셀들의 개수



영상 처리 기초 – Open CV 실습

본인이 찍어온 사진 or test1_1920.jpg 사용

본인이 찍어온 사진 크기 조정 => 1920x1440 과 비슷한 크기로



영상 처리 기초 – Open CV 실습

Colab에서 실습

```
import cv2
from google.colab.patches import cv2_imshow   ## colab일 경우만 필요
Img=cv2.imread( ' /content/test1_1920.jpg')
print('img.ndim = ',img.ndim)
print('img.shape = ',img.shape)
print('img.dtype = ',img.dtype)
```

영상 처리 기초 – Open CV 실습

Colab에서 실습

```
import cv2
from google.colab.patches import cv2_imshow   ## colab일 경우만 필요
Img=cv2.imread( ' /content/test1_1920.jpg')
print('img.ndim = ',img.ndim)
print('img.shape = ',img.shape)
print('img.dtype = ',img.dtype)
```

```
img.ndim = 3
img.shape = (1440, 1920, 3)
img.dtype = uint8
```

영상 처리 기초 – Open CV 실습

Colab에서 실습

#colab

```
cv2_imshow(img)

cv2.imshow
```

영상 처리 기초 – Open CV 실습

Colab에서 실습

```
img=cv2.resize(img,(0,0),fx=0.5,fy=0.5)
cv2.imwrite('test1_half.jpg',img)
print(img.shape)
```

영상 처리 기초 – Open CV 실습

Colab에서 실습

```
img=cv2.resize(img,(0,0),fx=0.5,fy=0.5)
cv2.imwrite('test1_half.jpg',img)
print(img.shape)
```

```
img.shape = (720, 960, 3)
```

영상 처리 기초 – Open CV 실습

Colab에서 실습

```
img=cv2.resize(img,(0,0),fx=0.5,fy=0.5)
cv2.imwrite('test1_half.jpg',img)
print(img.shape)
```

영상 변형

영상 처리 기초 – Open CV 실습

```
img1=cv2.imread('/content/test1_half.jpg')
gimg = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
cv2.imwrite('test1_halfgray.jpg',gimg)

print(gimg.shape)
imshow(gimg)
```

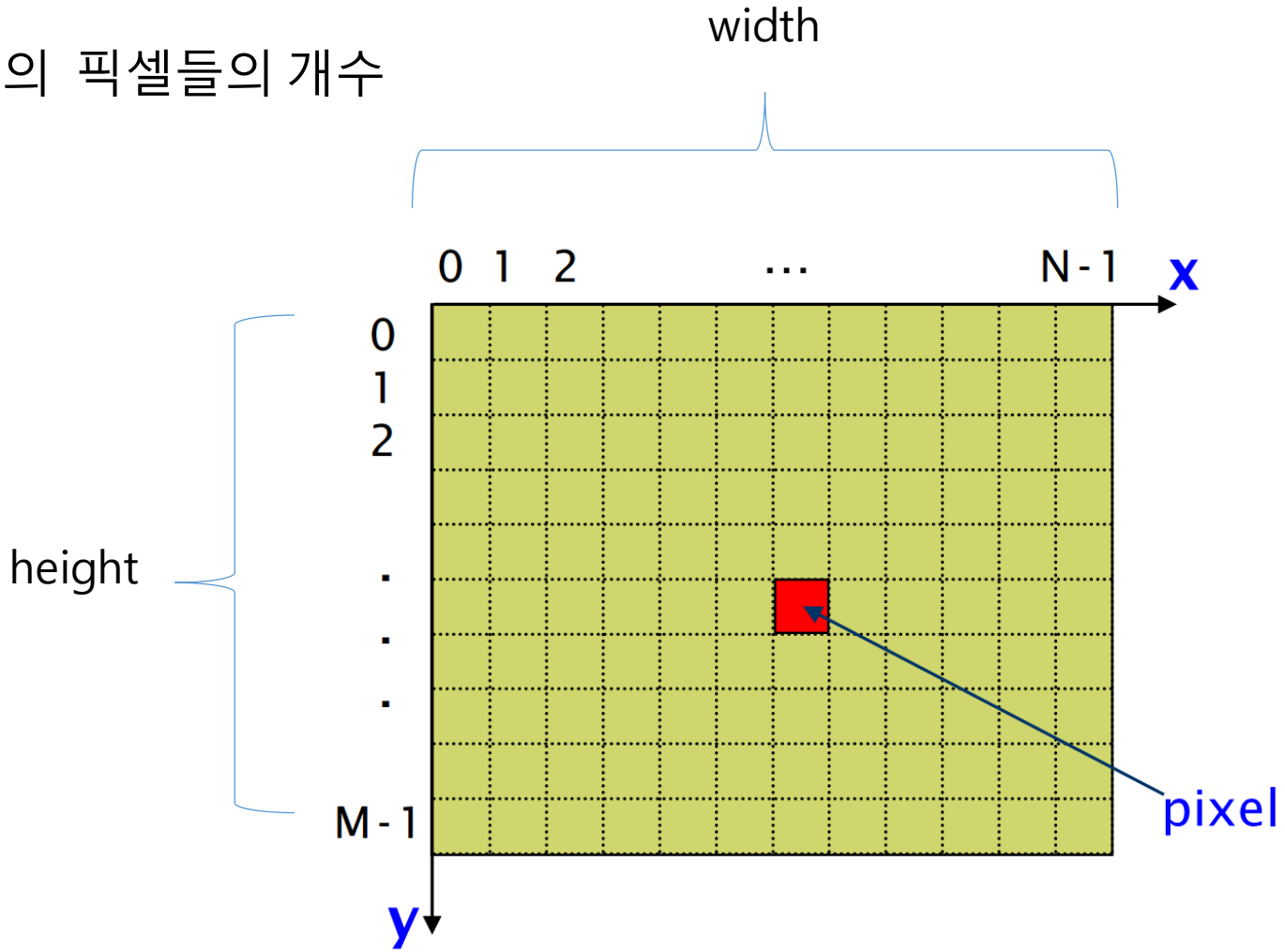
영상 처리 기초 – Open CV 실습

```
img1=cv2.imread('/content/test1_half.jpg')
gimg = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
cv2.imwrite('test1_halfgray.jpg',gimg)

print(gimg.shape)
imshow(gimg)
```

```
(720, 960)
```

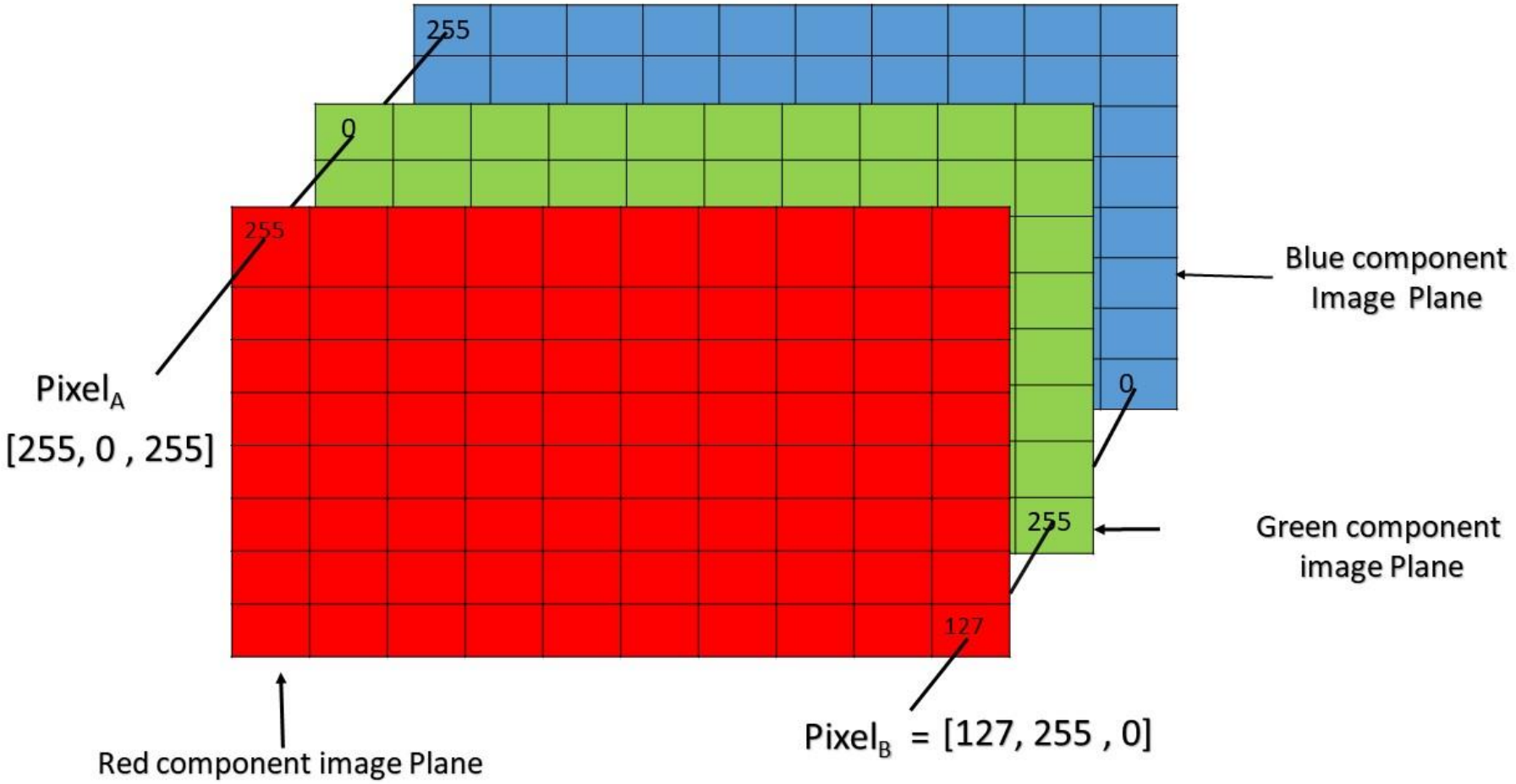
영상 처리 기초

- **영상은 픽셀로 이루어짐**
- **해상도 (Resolution)**
  - 2차원 공간 영역에서, x,y 축 방향의 픽셀들의 개수



영상 처리 기초

- **영상은 픽셀로 이루어짐.**
- **RGB color space**
  - Red, Green, Blue



Pixel of an RGB image are formed from the corresponding pixel of the three component images

Img.shape

=(height, width, rgb==3)

영상 처리 기초

컬러 이미지

Img.shape

=(height, width, rgb==3)

그레이 이미지

Img.shape

=(height, width)

Gray=0.299*R+0.587*G+0.114*R

수고 하셨습니다

jhmin@inhatc.ac.kr