AI 프로그래밍

- 10주차

인하 공전 컴퓨터 정보과

민 정혜

오늘 수업 순서- 딥러닝 예제

- **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
- **영상 처리기초 : open cv**

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

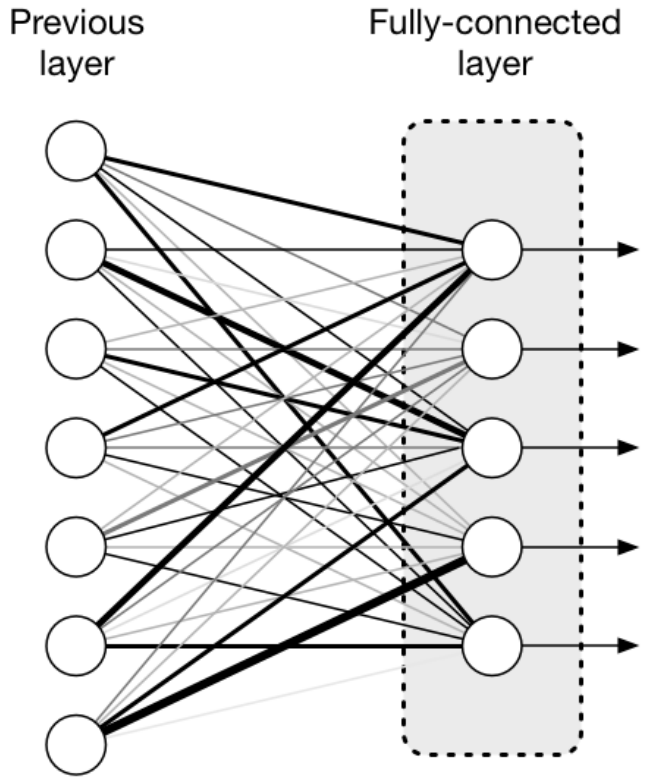

```
model = tf.keras.models.Sequential()


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 신경망 파라미터

tf.keras.layers.**Dense**

완전 연결 계층 : (Fully Connected Layer, Densely connected layer)

한 층 (layer)의 모든 뉴런이 그 다음 층의 모든 뉴런과 연결된 상태



https://keras.io/api/layers/core_layers/

케라스 신경망 파라미터 –loss (손실함수)

평균 제곱 계열	mean_squared_error	평균 제곱 오차 계산: mean(square(yt - yo))
	mean_absolute_error	평균 절대 오차(실제 값과 예측 값 차이의 절댓값 평균) 계산: mean(abs(yt - yo))
	mean_absolute_percentage_error	평균 절대 백분율 오차(절댓값 오차를 절댓값으로 나눈 후 평균) 계산: mean(abs(yt - yo)/abs(yt)) (단, 분모 ≠ 0)
	mean_squared_logarithmic_error	평균 제곱 로그 오차(실제 값과 예측 값에 로그를 적용한 값의 차이를 제곱한 값의 평균) 계산: mean(square((log(yo) + 1) - (log(yt) + 1)))
교차 엔트로피 계열	categorical_crossentropy	범주형 교차 엔트로피(일반적인 분류)
	binary_crossentropy	이항 교차 엔트로피(두 개의 클래스 중에서 예측할 때)

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|x_i - x|$$

케라스 신경망 파라미터 –optimizer

고급 경사 하강법	개요	효과	케라스 사용법
확률적 경사 하강법 (SGD)	랜덤하게 추출한 일부 데이터를 사용해 더 빨리, 자주 업데이트를 하게 하는 것	속도 개선	keras.optimizers.SGD(lr = 0.1) 케라스 최적화 함수를 이용합니다.
모멘텀 (Momentum)	관성의 방향을 고려해 진동과 폭을 줄이는 효과	정확도 개선	keras.optimizers.SGD(lr = 0.1, momentum = 0.9) 모멘텀 계수를 추가합니다.
네스테로프 모멘텀 (NAG)	모멘텀이 이동시킬 방향으로 미리 이동해서 그레이디언트를 계산. 불필요한 이동을 줄이는 효과	정확도 개선	keras.optimizers.SGD(lr = 0.1, momentum = 0.9, nesterov = True) 네스테로프 옵션을 추가합니다.
아다그라드 (Adagrad)	변수의 업데이트가 잦으면 학습률을 적게 하여 이동 보폭을 조절하는 방법	보폭 크기 개선	keras.optimizers.Adagrad(lr = 0.01, epsilon = 1e - 6) 아다그라드 함수를 사용합니다. ※ 참고: 여기서 epsilon, rho, decay 같은 파라미터는 바꾸지 않고 그대로 사용하기를 권장하고 있습니다. 따라서 lr, 즉 learning rate(학습률) 값만 적절히 조절하면 됩니다.

케라스 신경망 파라미터 –optimizer

고급 경사 하강법	개요	효과	케라스 사용법
알엠에스프롭 (RMSProp)	아다그라드의 보폭 민감도를 보완한 방법	보폭 크기 개선	keras.optimizers.RMSprop(lr = 0.001, rho = 0.9, epsilon = 1e - 08, decay = 0.0) 알엠에스프롭 함수를 사용합니다.
아담(Adam)	모멘텀과 알엠에스프롭 방법을 합친 방법	정확도와 보폭 크기 개선	keras.optimizers.Adam(lr = 0.001, beta_1 = 0.9, beta_2 = 0.999, epsilon = 1e - 08, decay = 0.0) 아담 함수를 사용합니다.

케라스 신경망 파라미터 –metrics

모델의 성능을 평가 하는 항목

accuracy :

- **긍정으로 올바르게 예측하면 TP(True Positive)라고 한다.**
- **긍정을 부정으로 잘못 예측하면 FN(False Nagative)라고 한다.**
- **부정을 긍정으로 잘못 예측하면 FP(False Positive)라고 한다.**
- **부정을 부정으로 올바르게 예측하면 TN(True Nagative)라고 한다. 긍정을**

혼동 행렬

$$\text{정확도}(accuarcy) = \frac{\text{올바르게 분류한 샘플 수}}{\text{전체 샘플 수}}$$

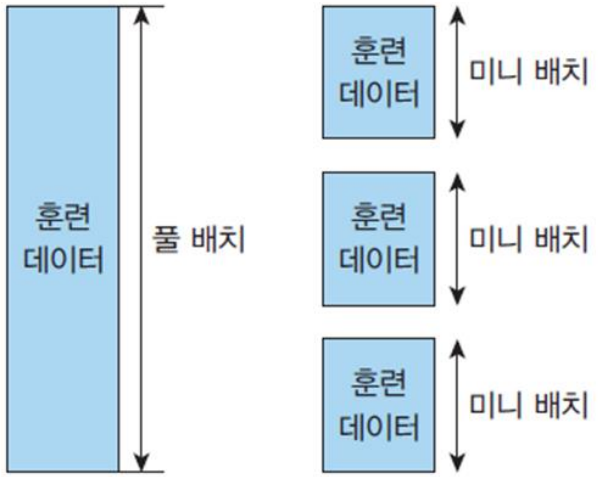
올바르게 분류한 샘플 수=TP + TN

Precision =TP / (TP + FP) True로 예측 한 것 중에서 실제로 True 인 비율

recall = TP /(TP + FN) : 실제로 True인 것 중에서 True로 예측된 비율

케라스 신경망 파라미터 –batch_size

- batch_size
  - 샘플을 한 번에 몇 개씩 처리할지를 정하는 부분으로 batch_size=10은 전체 샘플을 10개씩 끊어서 집어넣으라는 뜻
  - batch_size가 너무 크면 학습 속도가 느려지고, 너무 작으면 각 실행 값의 편차가 생겨서 전체 결과값이 불안정해질 수 있음
  - 변수 (w,b)가 업데이트 되는 단위.
  - 자신의 컴퓨터 메모리가 감당할 만큼의 batch_size를 찾아 설정해 주는 것이 좋음



1 epoch는 전체 샘플이 처리되는 기준

전체 샘플이 이 300 개이고 batch size가 30이면 1 epoch동안 10번 가중치가 update됨.
전체 샘플이 이 300 개이고 batch size가 10이면 1 epoch동안 30번 가중치가 update됨.

과제 리뷰

피마 인디언 데이터 분석

- 비만이 유전 및 환경, 모두의 탓이라는 것을 증명하는 좋은 사례가 바로 미국 남서부에 살고 있는 피마 인디언의 사례

속성 / 클래스

샘플	정보 1	정보 2	정보 3	…	정보 8	당뇨병 여부
1번째 인디언	6	148	72	…	50	1
2번째 인디언	1	85	66	…	31	0
3번째 인디언	8	183	64	…	32	1
…	…	…	…	…	…	…
768번째 인디언	1	93	70	…	23	0

표 11-1 피마 인디언 데이터의 샘플, 속성, 클래스 구분

- 샘플 수: 768
- 속성: 8
  - 정보 1 (pregnant): 과거 임신 횟수
  - 정보 2 (plasma): 포도당 부하 검사 2시간 후 공복 혈당 농도(mm Hg)
  - 정보 3 (pressure): 확장기 혈압(mm Hg)
  - 정보 4 (thickness): 삼두근 피부 주름 두께(mm)
  - 정보 5 (insulin): 혈청 인슐린(2-hour, mu U/ml)
  - 정보 6 (BMI): 체질량 지수(BMI, weight in kg/(height in $m)^2$)
  - 정보 7 (pedigree): 당뇨병 가족력
  - 정보 8 (age): 나이
- 클래스: 당뇨(1), 당뇨 아님(0)

피마 인디언 데이터 분석

속성 / 클래스

	정보 1	정보 2	정보 3	…	정보 8	당뇨병 여부
샘플						
1번째 인디언	6	148	72	…	50	1
2번째 인디언	1	85	66	…	31	0
3번째 인디언	8	183	64	…	32	1
…	…	…	…	…	…	…
768번째 인디언	1	93	70	…	23	0

표 11-1 피마 인디언 데이터의 샘플, 속성, 클래스 구분

- 샘플 수: 768
- 속성: 8
  - 정보 1 (pregnant): 과거 임신 횟수
  - 정보 2 (plasma): 포도당 부하 검사 2시간 후 공복 혈당 농도(mm Hg)
  - 정보 3 (pressure): 확장기 혈압(mm Hg)
  - 정보 4 (thickness): 삼두근 피부 주름 두께(mm)
  - 정보 5 (insulin): 혈청 인슐린(2-hour, mu U/ml)
  - 정보 6 (BMI): 체질량 지수(BMI, weight in kg/(height in $m)^2$)
  - 정보 7 (pedigree): 당뇨병 가족력
  - 정보 8 (age): 나이
- 클래스: 당뇨(1), 당뇨 아님(0)

피마 인디언 데이터 분석

Pandas 라이브러리 : 데이터 처리, 가공

```
import pandas as pd
df = pd.read_csv('../dataset/pima-indians-diabetes.csv',
                 names = ["pregnant", "plasma", "pressure", "thickness",
                 "insulin", "BMI", "pedigree", "age", "class"])
```

- Csv :

  comma separated values file의 약자로, 콤마(,)로 구분된 데이터들의 모음이란 뜻

- 헤더(header) :

  csv 파일에는 데이터를 설명하는 한 줄이 파일 맨 처음에 나옴

- 우리가 가진 csv 파일에는 헤더가 없음
- 이에 names라는 함수를 통해 속성별 키워드를 지정해 줌

피마 인디언 데이터 분석

- 이제 불러온 데이터의 내용을 간단히 확인하고자 head( ) 함수를 이용하여 데이터의 첫 다섯 줄을 불러옴

```
print(df.head(5))
```

	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

피마 인디언 데이터 분석

데이터 전반적인 정보 확인

```
print(df.info( ))
```

Range Index: 768 entries, 0 to 767

Data	Columns (total 9)		
pregnant	768	non-null	int64
plasma	768	non-null	int64
pressure	768	non-null	int64
thickness	768	non-null	int64
insulin	768	non-null	int64
BMI	768	non-null	float64
pedigree	768	non-null	float64
age	768	non-null	int64
class	768	non-null	int64
Dtypes: float64(2) int64(7)			
Memory usage: 54.1 KB			

피마 인디언 데이터 분석

상관 관계 분석

```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
plt.figure(figsize=(12,12))
```

```
sns.heatmap(df.corr( ), linewidths=0.1, vmax=0.5, cmap=plt.cm.gist_
heat, linecolor='white', annot=True)
```

```
plt.show( )
```

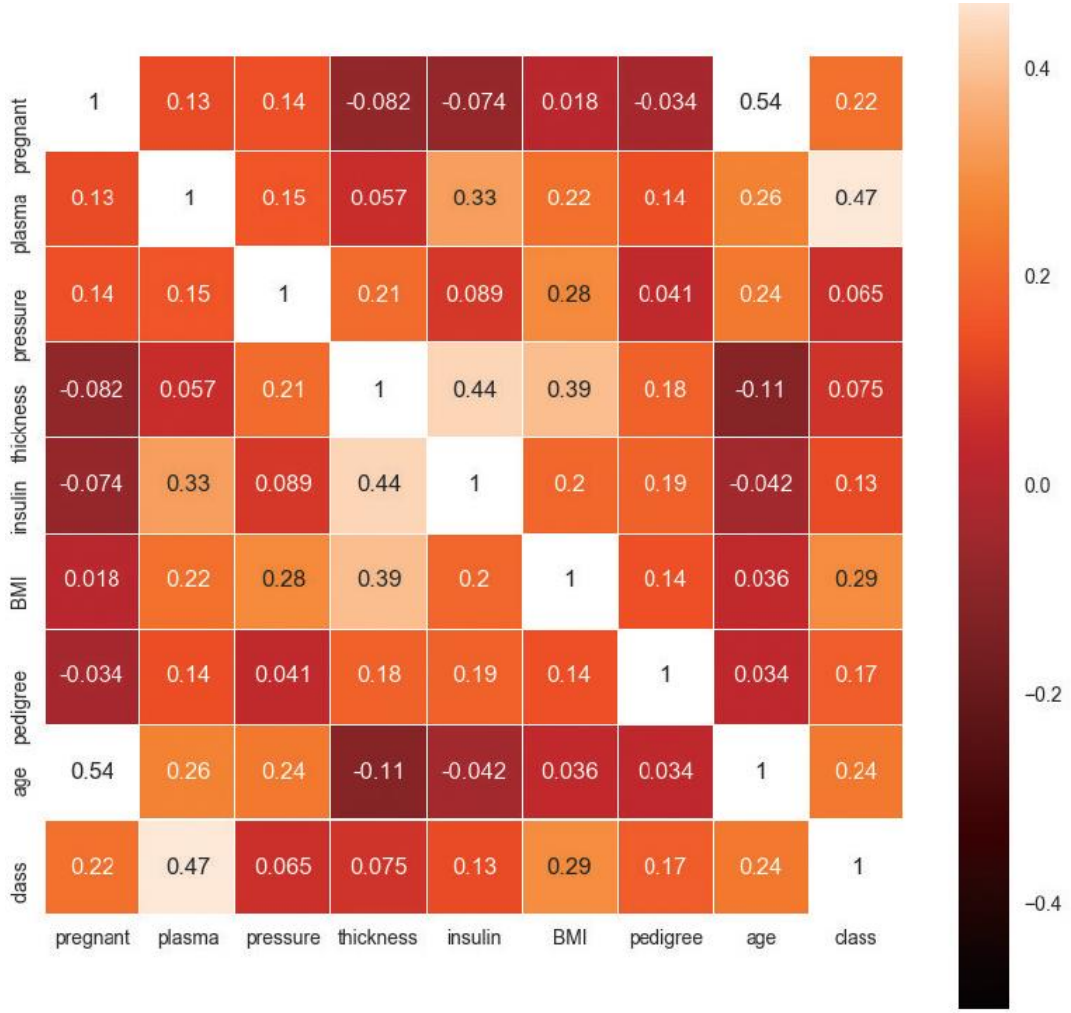
- heatmap( ) 함수 :

  두 항목씩 짝을 지은 뒤 각각 어떤 패턴으로 변화하는지를 관찰하는 힘

- 두 항목이 전혀 다른 패턴으로 변화하고 있으면 0을, 서로 비슷한 패턴으로 변할수록 1에 가까운 값을 출력함

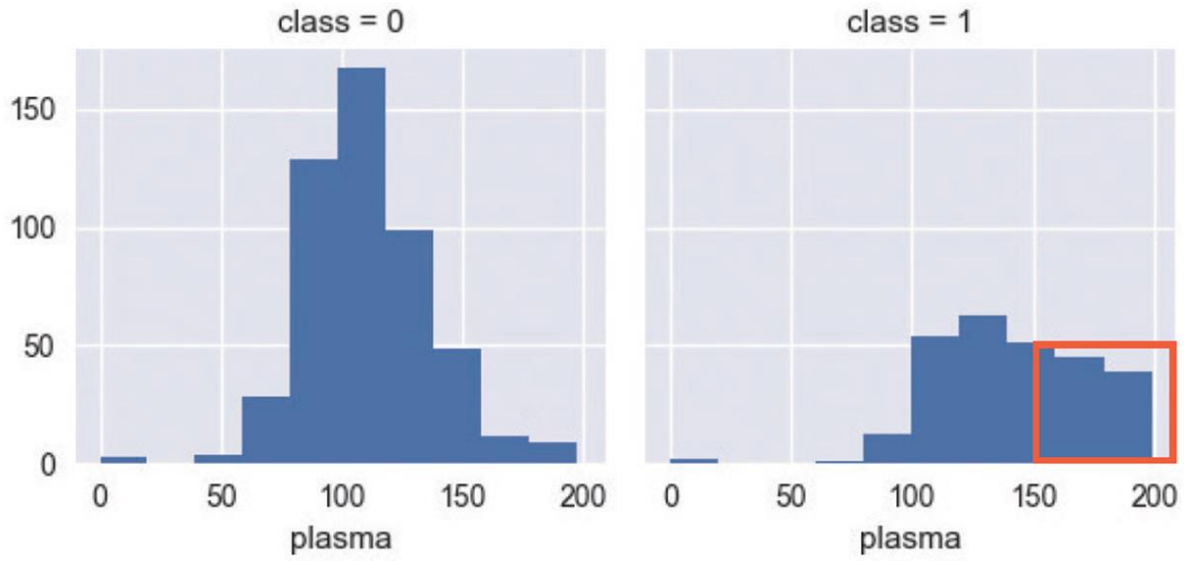
피마 인디언 데이터 분석

```
plt.show()
```



피마 인디언 데이터 분석

- 그래프를 통해 plasma 항목(공복 혈당 농도)이 class 항목과 가장 상관관계가 높다는 것을 알 수 있음
- 즉, 이 항목이 결론을 만드는 데 가장 중요한 역할을 한다는 것을 예측할 수 있음
- 이제 plasma와 class 항목만 따로 떼어 두 항목 간의 관계를 그래프로 다시 한번 확인함

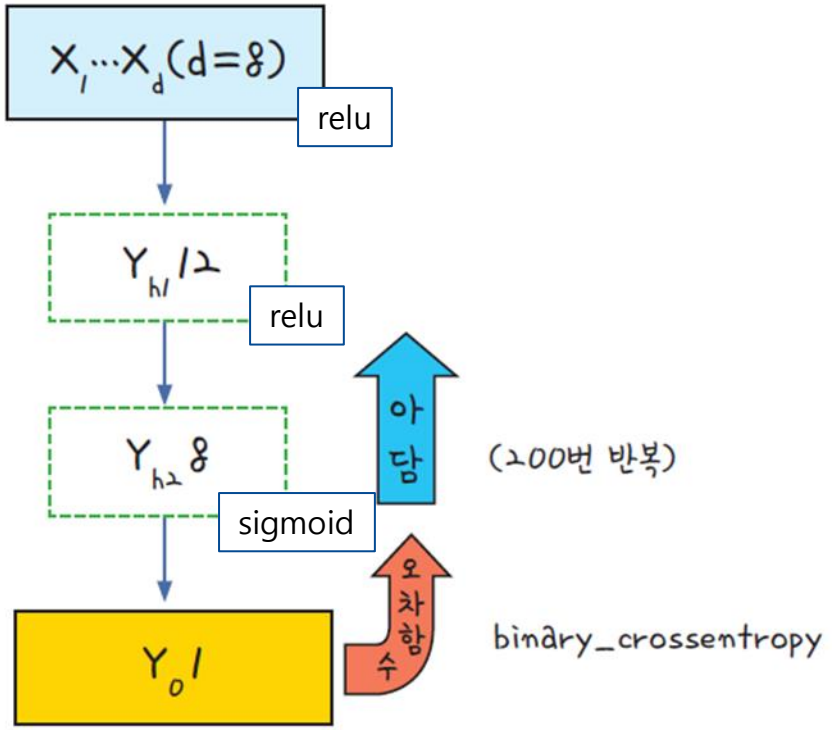


피마 인디언 데이터 분석

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import numpy
import tensorflow as tf

# seed 값 생성
np.random.seed(3)
tf.random.set_seed(3)

# 데이터 로드
dataset = numpy.loadtxt("../dataset/pima-indians-diabetes.csv",
delimiter=",")
X = dataset[:,0:8]
Y = dataset[:,8]
```

피마 인디언 데이터 분석

```
# 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'))

# 모델 컴파일
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
# 모델 실행
model.fit(X, Y, epochs=200, batch_size=10)
# 결과 출력
print("\n Accuracy: %.4f" % (model.evaluate(X, Y)[1]))
```



아이리스 품종 예측

- 아이리스는 꽃잎의 모양과 길이에 따라 여러 가지 품종으로 나뉨
- 사진을 보면 품종마다 비슷해 보이는데 과연 딥러닝을 사용하여 이들을 구별해 낼 수 있을까?

그림 12-1 아이리스의 품종

아이리스 품종 예측

속성 / 클래스

	정보 1	정보 2	정보 3	정보 4	품종
1번째 아이리스	5.1	3.5	4.0	0.2	Iris-setosa
2번째 아이리스	4.9	3.0	1.4	0.2	Iris-setosa
3번째 아이리스	4.7	3.2	1.3	0.3	Iris-setosa
...	...	...	...	...	...
150번째 아이리스	5.9	3.0	5.1	1.8	Iris-virginica

샘플

- 샘플 수: 150
- 속성 수: 4
  - 정보 1: 꽃받침 길이 (sepal length, 단위: cm)
  - 정보 2: 꽃받침 너비 (sepal width, 단위: cm)
  - 정보 3: 꽃잎 길이 (petal length, 단위: cm)
  - 정보 4: 꽃잎 너비 (petal width, 단위: cm)
- 클래스: Iris-setosa, Iris-versicolor, Iris-virginica

다중 분류 : 여러 개의 답 중 하나를 고르는 문제

아이리스 품종 예측

- 먼저 데이터의 일부를 불러와 내용을 보자

```
import pandas as pd
df = pd.read_csv('../dataset/iris.csv', names = ["sepal_length",
"sepal_width", "petal_length", "petal_width", "species"])
print(df.head( ))
```

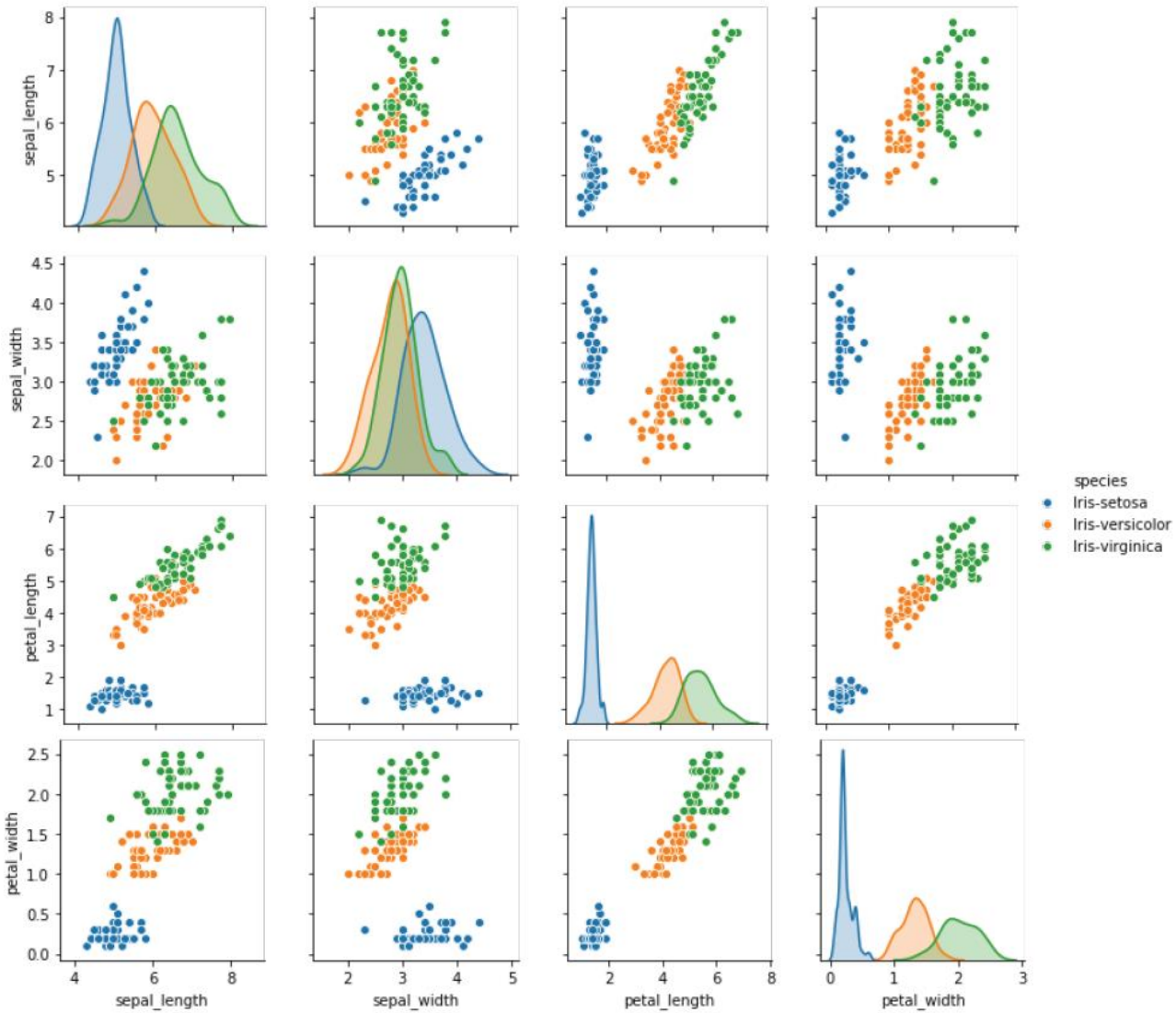
	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5	3.6	1.4	0.2	Iris-setosa

아이리스 품종 예측

- 이번에는 pairplot( ) 함수를 써서 데이터 전체를 한번에 보는 그래프를 다음과 같이 출력

```
import seaborn as sns
import matplotlib.pyplot as plt


sns.pairplot(df, hue='species');
plt.show( )
```

아이리스 품종 예측


sepal_length
sepal_width
petal_length
petal_width
species
Iris-setosa
Iris-versicolor
Iris-virginica

아이리스 품종 예측

원-핫 인코딩

- 이제 케라스를 이용해 아이리스의 품종을 예측해 보자
- 등 데이터 안에 문자열이 포함되어 있음
- numpy보다는 pandas로 데이터를 불러와 X와 Y값을 구분하는 것이 좋음

```
df = pd.read_csv('../dataset/iris.csv', names = ["sepal_length",
"sepal_width", "petal_length", "petal_width", "species"])


dataset = df.values
X = dataset[:,0:4].astype(float)
Y_obj = dataset[:,4]
```

아이리스 품종 예측

- 또한, Y값이 이번에는 숫자가 아니라 문자열임
- 문자열을 숫자로 바꿔 주려면 클래스 이름을 숫자 형태로 바꿔 주어야 함
- 이를 가능하게 하는 함수가 sklearn 라이브러리의 LabelEncoder( ) 함수임

```
from sklearn.preprocessing import LabelEncoder


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

아이리스 품종 예측

- array(['Iris-setosa', 'Iris-versicolor','Iris-virginica'])가 array([1,2,3])로 바뀜
- 활성화 함수를 적용하려면 Y 값이 숫자 0과 1로 이루어져 있어야 함
- 이 조건을 만족시키려면 tf.keras.utils.categorical( ) 함수를 적용해야 함
- 이에 따라 Y 값의 형태는 다음과 같이 변형됨

```
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

아이리스 품종 예측

- 이제 모델을 만들어 보자

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

아이리스 품종 예측

- 소프트맥스 :

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

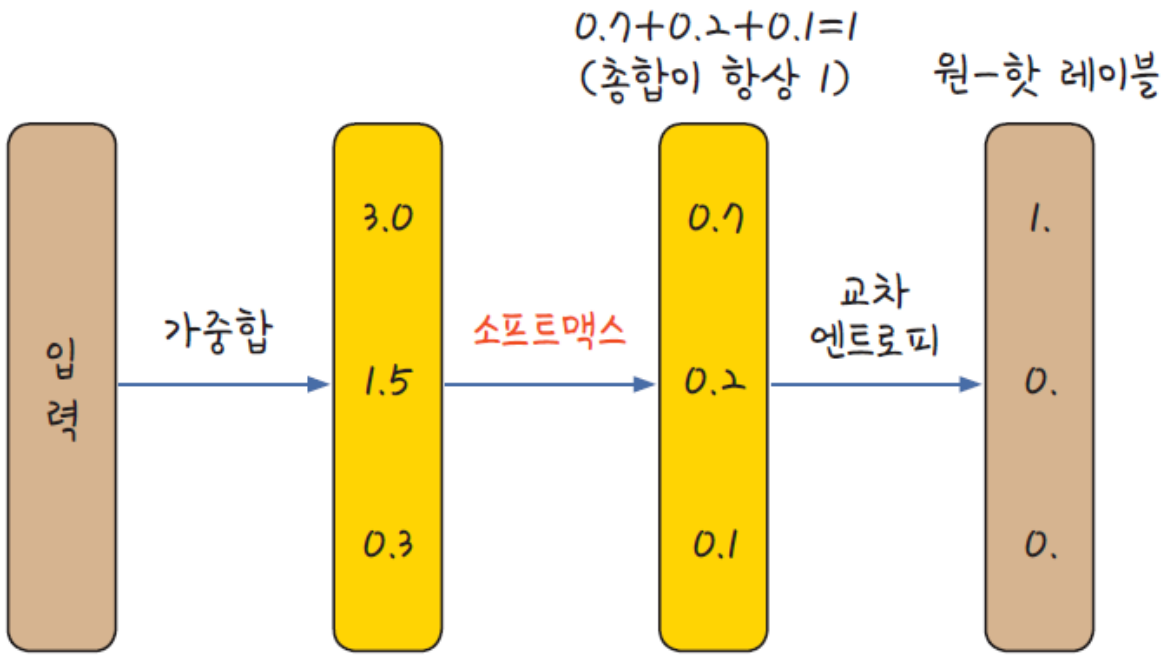


그림 12-3 소프트맥스 함수의 원리

아이리스 품종 예측

- 다중 분류에 적절한 오차 함수인 categorical_crossentropy를 사용하고, 최적화 함수로 adam을 사용함
- 전체 샘플이 50회 반복될 때까지 실험을 진행하되 한 번에 입력되는 값은 1개로 함

코드 12-1 아이리스 품종 예측하기

- 예제 소스: run_project/03_Iris_Multi_Classfication.ipynb

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.preprocessing import LabelEncoder
```

아이리스 품종 예측

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

# seed 값 설정
np.random.seed(3)
tf.random.set_seed(3)

# 데이터 입력
df = pd.read_csv('../dataset/iris.csv', names = ["sepal_length",
"sepal_width", "petal_length", "petal_width", "species"])
```

아이리스 품종 예측

```
# 그래프로 확인
sns.pairplot(df, hue='species');
plt.show( )

# 데이터 분류
dataset = df.values
X = dataset[:,0:4].astype(float)
Y_obj = dataset[:,4]

# 문자열을 숫자로 변환
e = LabelEncoder( )
e.fit(Y_obj)
Y = e.transform(Y_obj)
Y_encoded = tf.keras.utils.to_categorical(Y)
```

아이리스 품종 예측

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

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

# 모델 실행
model.fit(X, Y_encoded, epochs=50, batch_size=1)

# 결과 출력
print("\n Accuracy: %.4f" % (model.evaluate(X, Y_encoded)[1]))
```

초음파 광물 예측

- 1988년 존스홉킨스대학교의 세즈노프스키(Sejnowski) 교수는 2년 전 힌튼 교수가 발표한 역전파 알고리즘에 관심을 가지고 있었음
- 그는 은닉층과 역전파가 얼마나 큰 효과가 있는지를 직접 실험해 보고 싶었음
- 광석과 일반 돌을 가져다 놓고 음파 탐지기를 쏜 후 그 결과를 데이터로 정리함
- 오차 역전파 알고리즘을 사용한 신경망이 과연 얼마나 광석과 돌을 구분하는 데 효과적인지 알아보기 위해서임

초음파 광물 예측

- dataset/sonar.csv 파일을 가져옴

```
import pandas as pd


df = pd.read_csv('../dataset/sonar.csv', header=None)
print(df.info( ))
```

Range Index: 208 entries,0 to 207			
Data columns (total 61 columns):			
0	208	non-null	float64
1	208	non-null	float64
2	208	non-null	float64
3	208	non-null	float64
4	208	non-null	float64

Range Index: 208 entries,0 to 207			
Data columns (total 61 columns):			
5	208	non-null	float64
...	...	...	...
58	208	non-null	float64
59	208	non-null	float64
60	208	non-null	object
Dtypes: float64(60), object(1)			
memory usage: 99.2+ KB			

초음파 광물 예측

- 모든 컬럼이 실수형(float64)인데, 맨 마지막 컬럼만 객체형인 것으로 보아 마지막에 나오는 컬럼은 클래스이며 데이터형 변환이 필요한 것을 알 수 있음
- 실제로 맞는지 일부를 출력해 확인해 보자

```
print(df.head( ))
```

	0	1	2	3	...	59	60
0	0.02	0.0371	0.0428	0.0207	...	0.0032	R
1	0.0453	0.0523	0.0843	0.0689	...	0.0044	R
2	0.0262	0.0582	0.1099	0.1083	...	0.0078	R
3	0.01	0.0171	0.0623	0.0205	...	0.0117	R
4	0.0762	0.0666	0.0481	0.0394	...	0.0094	R

초음파 광물 예측

- 모든 컬럼이 실수형(float64)인데, 맨 마지막 컬럼만 객체형인 것으로 보아 마지막에 나오는 컬럼은 클래스이며 데이터형 변환이 필요한 것을 알 수 있음
- 실제로 맞는지 일부를 출력해 확인해 보자

```
print(df.head( ))
```

	0	1	2	3	...	59	60
0	0.02	0.0371	0.0428	0.0207	...	0.0032	R
1	0.0453	0.0523	0.0843	0.0689	...	0.0044	R
2	0.0262	0.0582	0.1099	0.1083	...	0.0078	R
3	0.01	0.0171	0.0623	0.0205	...	0.0117	R
4	0.0762	0.0666	0.0481	0.0394	...	0.0094	R

초음파 광물 예측

코드 13-1 초음파 광물 예측하기: 데이터 확인과 실행

- 예제 소스: run_project/04_Sonar.ipynb

```
from keras.models import Sequential
from keras.layers.core import Dense
from sklearn.preprocessing import LabelEncoder

import pandas as pd
import numpy
import tensorflow as tf

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

초음파 광물 예측

```
# 데이터 입력
df = pd.read_csv('../dataset/sonar.csv', header=None)

dataset = df.values
X = dataset[:,0:60]
Y_obj = dataset[:,60]

# 문자열 변환
e = LabelEncoder( )
e.fit(Y_obj)
Y = e.transform(Y_obj)
```

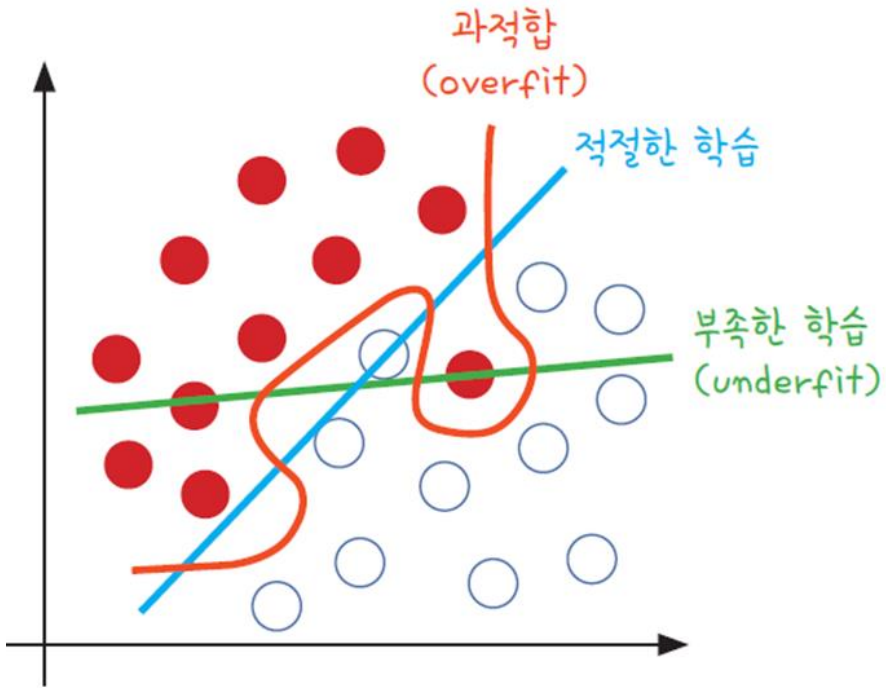
초음파 광물 예측

```
# 모델 설정
model = Sequential()
model.add(Dense(24, input_dim=60, activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# 모델 컴파일
model.compile(loss='mean_squared_error',
              optimizer='adam',
              metrics=['accuracy'])

# 모델 실행
model.fit(X, Y, epochs=200, batch_size=5)

# 결과 출력
print("\n Accuracy: %.4f" % (model.evaluate(X, Y)[1]))
```

초음파 광물 예측

과적합 (Overfitting) 이해

- 완전히 새로운 데이터에 적용하면 이 선을 통해 정확히 두 그룹으로 나누지 못하게 됨



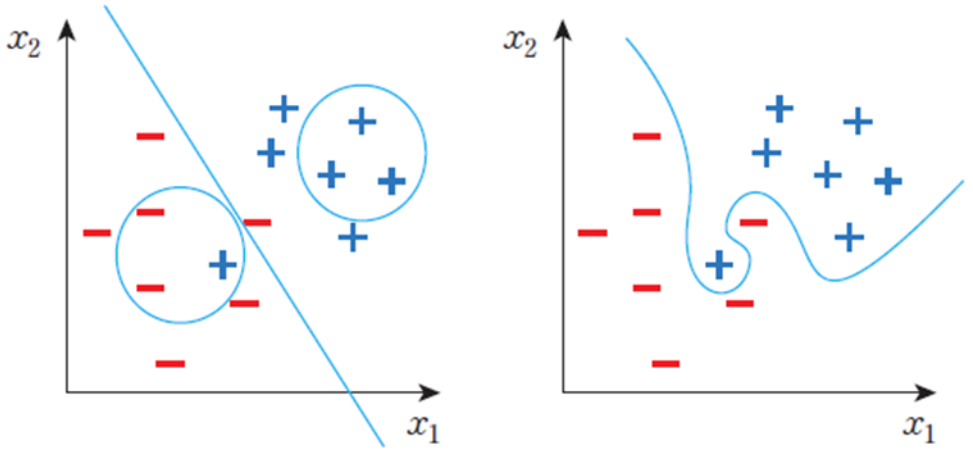


그림 8-13 과잉 적합의 예

초음파 광물 예측

- 과적합은 층이 너무 많거나 변수가 복잡해서 발생하기도 하고 테스트셋과 학습셋이 중복될 때 생기기도 함
- 딥러닝은 학습 단계에서 입력층, 은닉층, 출력층의 노드들에 상당히 많은 변수들이 투입됨
- 딥러닝을 진행하는 동안 과적합에 빠지지 않게 늘 주의해야 함

- 과적합을 방지하려면 어떻게 해야 할까?
  - → 먼저 학습을 하는 데이터셋과 이를 테스트할 데이터셋을 완전히 구분한 다음 학습과 동시에 테스트를 병행하며 진행하는 것이 한 방법
- 데이터셋이 총 100개의 샘플로 이루어져 있다면 다음과 같이 두 개의 셋으로 나눔

70개 샘플은 학습셋으로	30개 샘플은 테스트셋으로

초음파 광물 예측

- 신경망을 만들어 70개의 샘플로 학습을 진행한 후 이 학습의 결과를 저장함

  → 이렇게 저장된 파일을 '모델'이라고 부름
- 모델은 다른 셋에 적용할 경우 학습 단계에서 각인되었던 그대로 다시 수행함
- 나머지 30개의 샘플로 실험해서 정확도를 살펴보면 학습이 얼마나 잘 되었는지를 알 수 있는 것

초음파 광물 예측

학습 셋 (training set) 과 테스트 셋 (test set)

- 딥러닝 같은 알고리즘을 충분히 조절하여 가장 나은 모델이 만들어지면,
  이를 실생활에 대입하여 활용하는 것이 바로 머신러닝의 개발 순서

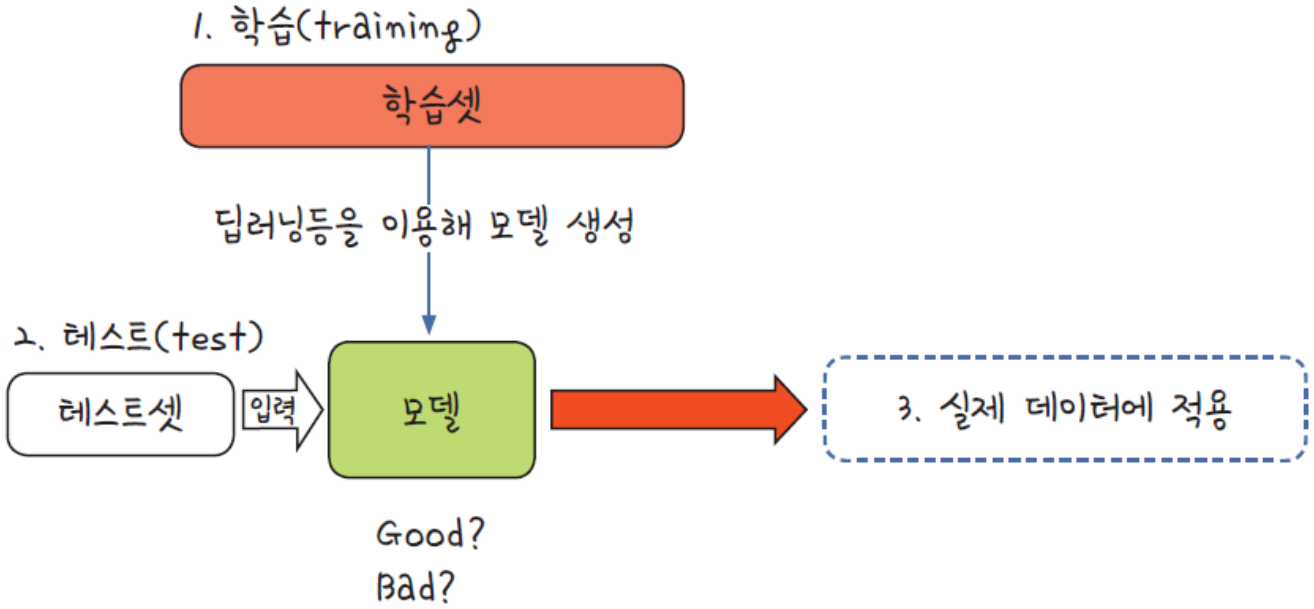
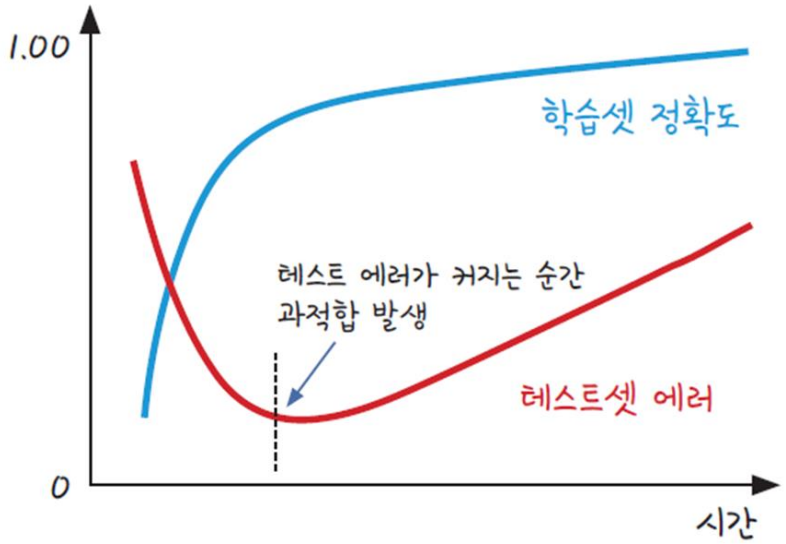


그림 13-2 학습셋과 테스트셋

초음파 광물 예측

- 학습에 사용된 샘플은 테스트에 쓸 수 없으므로 학습 단계에서 테스트할 샘플은 자동으로 빼고, 이를 테스트한 결과를 모아 정확도를 계산하는 것
- 이러한 방법은 빠른 시간에 모델 성능을 파악하고 수정할 수 있도록 도와 줌
- 머신러닝의 최종 목적은 과거의 데이터를 토대로 새로운 데이터를 예측하는 것
- 테스트셋을 만들어 정확한 평가를 병행하는 것이 매우 중요함



초음파 광물 예측

- 불러온 X 데이터와 Y 데이터에서 각각 정해진 비율(%)만큼 구분하여 한 그룹은 학습에 사용함
- 다른 한 그룹은 테스트에 사용하게 하는 함수가 sklearn 라이브러리의 train_test_split( ) 함수임
- 학습셋을 70%, 테스트셋을 30%로 설정했을때의 예

```
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)
```

초음파 광물 예측

- 모델을 실행하는 부분에서 위에서 만들어진 학습셋으로 학습을, 테스트셋으로 테스트를 하게 하려면 다음과 같이 실행함

```
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]))
```

초음파 광물 예측

코드 13-2 초음파 광물 예측하기: 학습셋과 테스트셋 구분

- 예제 소스: run_project/05_Sonar_Train_Test.ipynb

```
from keras.models import Sequential
from keras.layers.core import Dense
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

import pandas as pd
import numpy
import tensorflow as tf
```

초음파 광물 예측

```
# seed 값 설정
seed = 0
numpy.random.seed(seed)
tf.random.set_seed(3)
df = pd.read_csv('../dataset/sonar.csv', header=None)
dataset = df.values
X = dataset[:,0:60]
Y_obj = dataset[:,60]

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

초음파 광물 예측

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

model = Sequential()
model.add(Dense(24, input_dim=60, activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

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

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]))
```

초음파 광물 예측

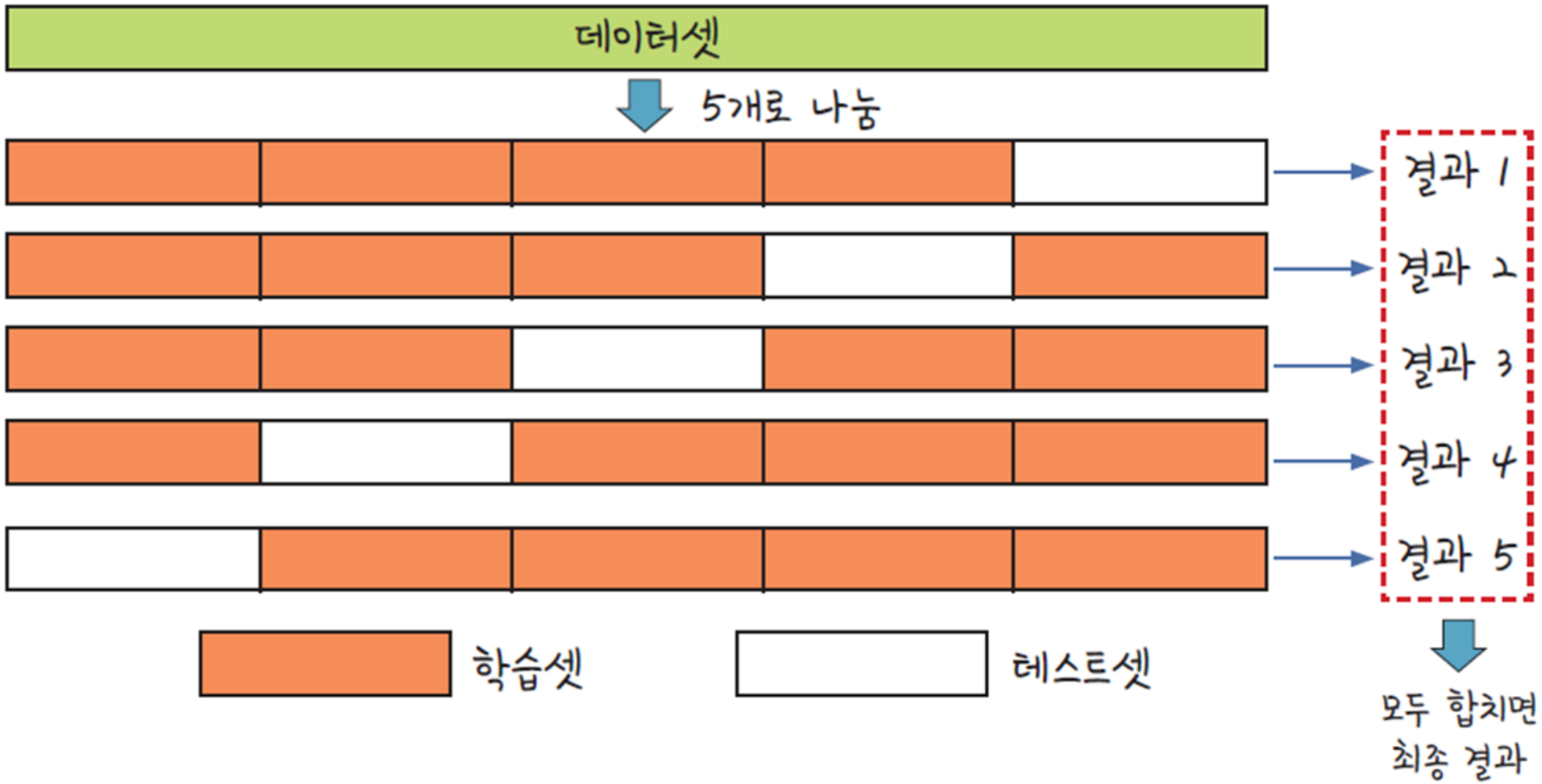
모델 저장과 재사용

- 학습이 끝난 후 테스트해 본 결과가 만족스러울 때 이를 모델로 저장하여 새로운 데이터에 사용할 수 있음
- 앞서 학습한 결과를 모델로 저장하려면 다음과 같이 실행함

```
from keras.models import load_model


model.save('my_model.h5')
```

초음파 광물 예측

5 | k겹 교차 검증



초음파 광물 예측

- 데이터를 원하는 숫자만큼 쪼개 각각 학습셋과 테스트셋으로 사용되게 만드는 함수는 sklearn의 StratifiedKFold( ) 함수

```
from sklearn.model_selection import StratifiedKFold


n_fold = 10
skf = StratifiedKFold(n_splits=n_fold, shuffle=True, random_
state=seed)
```

초음파 광물 예측

- 모델을 만들고 실행하는 부분을 for 구문으로 묶어 n_fold만큼 반복되게 함

```
for train, test in skf.split(X, Y):
    model = Sequential( )
    model.add(Dense(24, input_dim=60, activation='relu'))
    model.add(Dense(10, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='mean_squared_error',
                  optimizer='adam',
                  metrics=['accuracy'])
    model.fit(X[train], Y[train], epochs=100, batch_size=5)
```

초음파 광물 예측

- 정확도(Accuracy)를 매번 저장하여 한 번에 보여줄 수 있게 accuracy 배열을 만듦

```
accuracy = []

for train, test in skf.split(X, Y):
    (중략)
    k_accuracy = "%.4f" % (model.evaluate(X[test], Y[test])[1])
    accuracy.append(k_accuracy)

print("\n %.f fold accuracy:" % n_fold, accuracy)
```

초음파 광물 예측

코드 13-4 초음파 광물 예측하기: k겹 교차 검증

- 예제 소스: run_projec/07_Sonar-K-fold.ipynb

```
from keras.models import Sequential
from keras.layers.core import Dense
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import StratifiedKFold

import numpy
import pandas as pd
import tensorflow as tf
```

초음파 광물 예측

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

df = pd.read_csv('../dataset/sonar.csv', header=None)

dataset = df.values
X = dataset[:,0:60]
Y_obj = dataset[:,60]

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

초음파 광물 예측

```
# 10개의 파일로 쪼갬
n_fold = 10

skf = StratifiedKFold(n_splits=n_fold, shuffle=True, random_
state=seed)


# 빈 accuracy 배열
accuracy = []
```

초음파 광물 예측

```
# 모델의 설정, 컴파일, 실행
for train, test in skf.split(X, Y):
    model = Sequential( )
    model.add(Dense(24, input_dim=60, activation='relu'))
    model.add(Dense(10, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='mean_squared_error',
                  optimizer='adam',
                  metrics=['accuracy'])
    model.fit(X[train], Y[train], epochs=100, batch_size=5)
    k_accuracy = "%.4f" % (model.evaluate(X[test], Y[test])[1])
    accuracy.append(k_accuracy)

# 결과 출력
print("\n %.f fold accuracy:" % n_fold, accuracy)
```

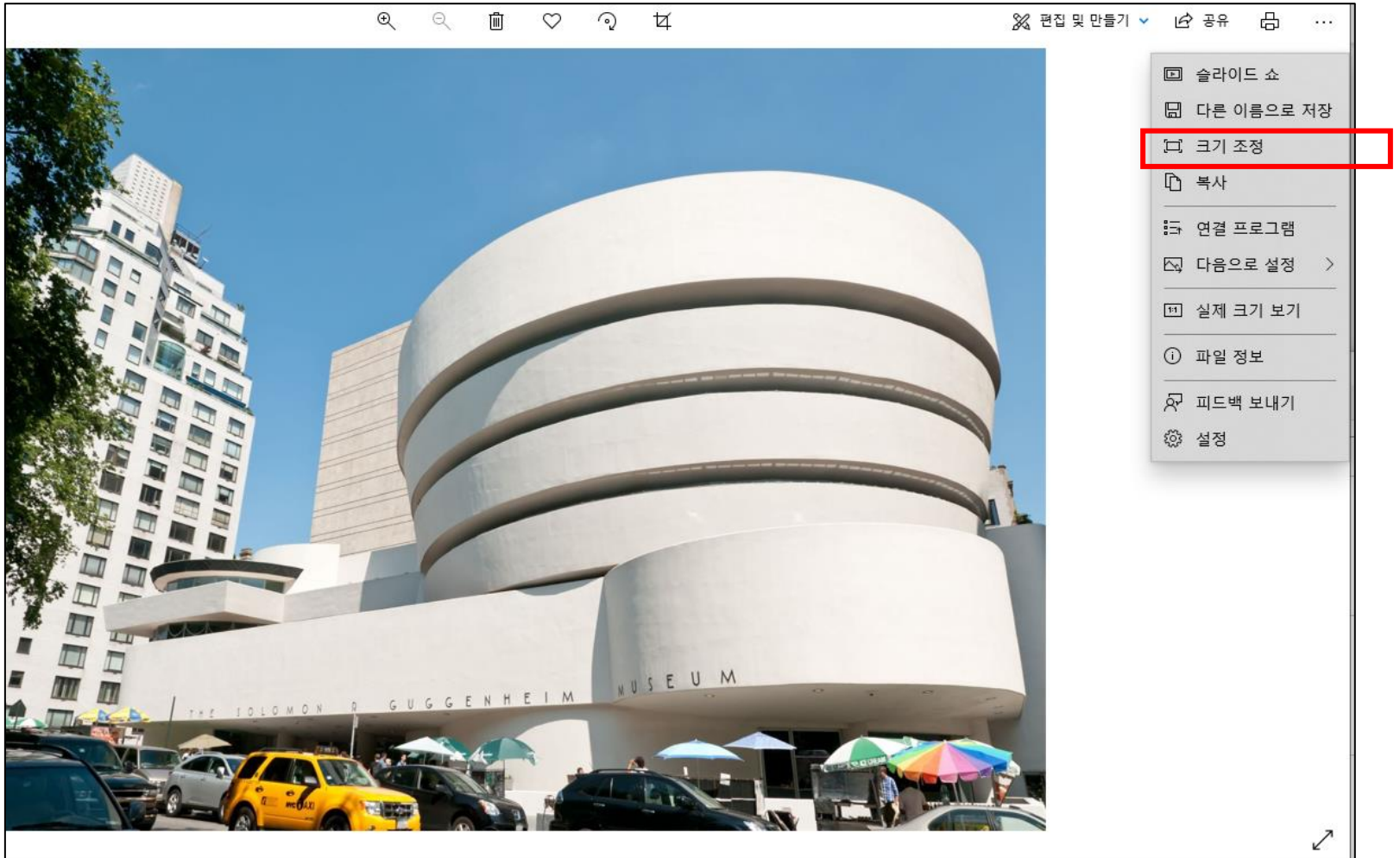
초음파 광물 예측

```
# 결과 출력
print("\n %.f fold accuracy:" % n_fold, accuracy)
```

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)
```

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

```
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)
```

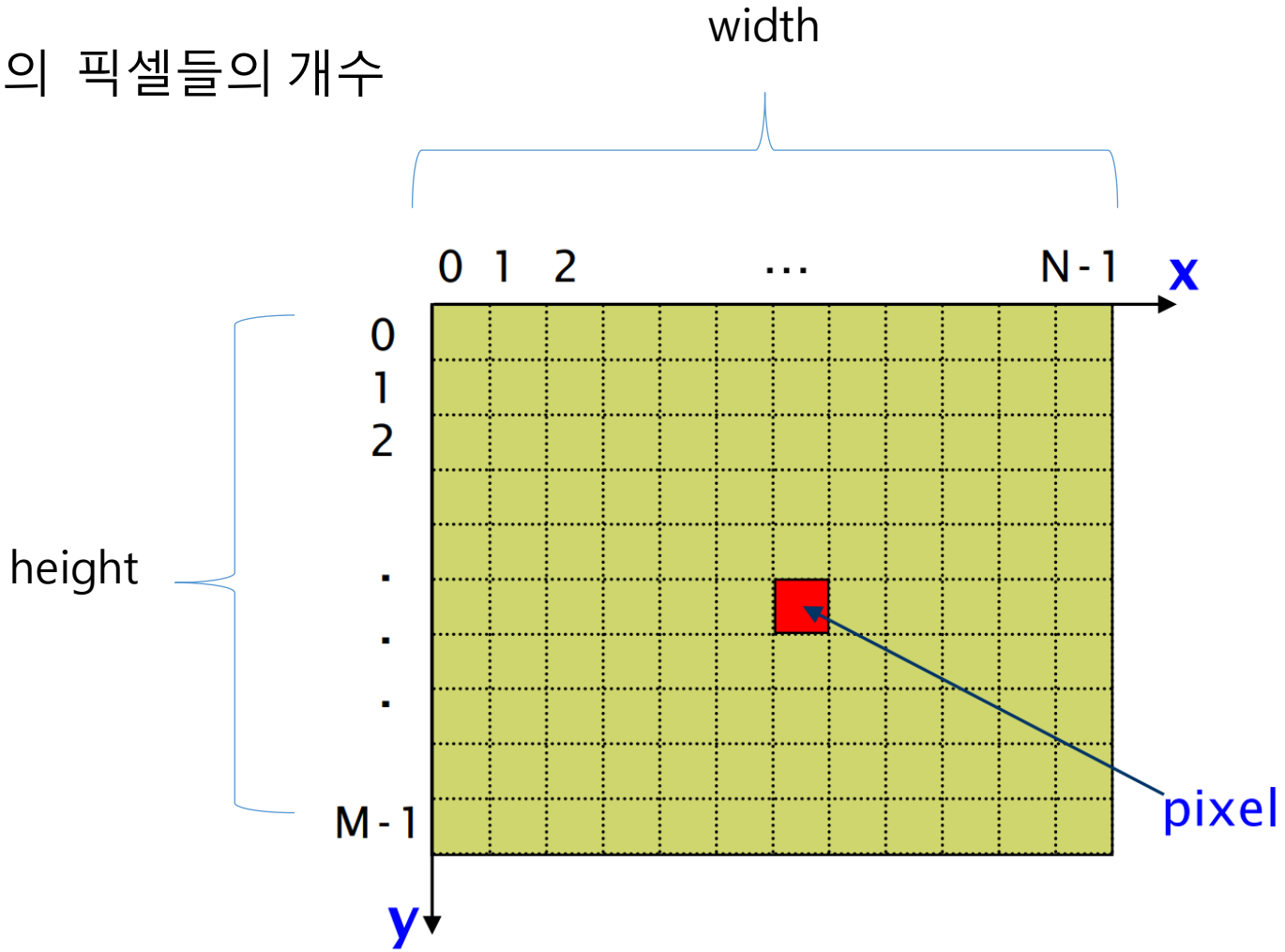
영상 처리 기초 – Open CV 실습

```
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)
```

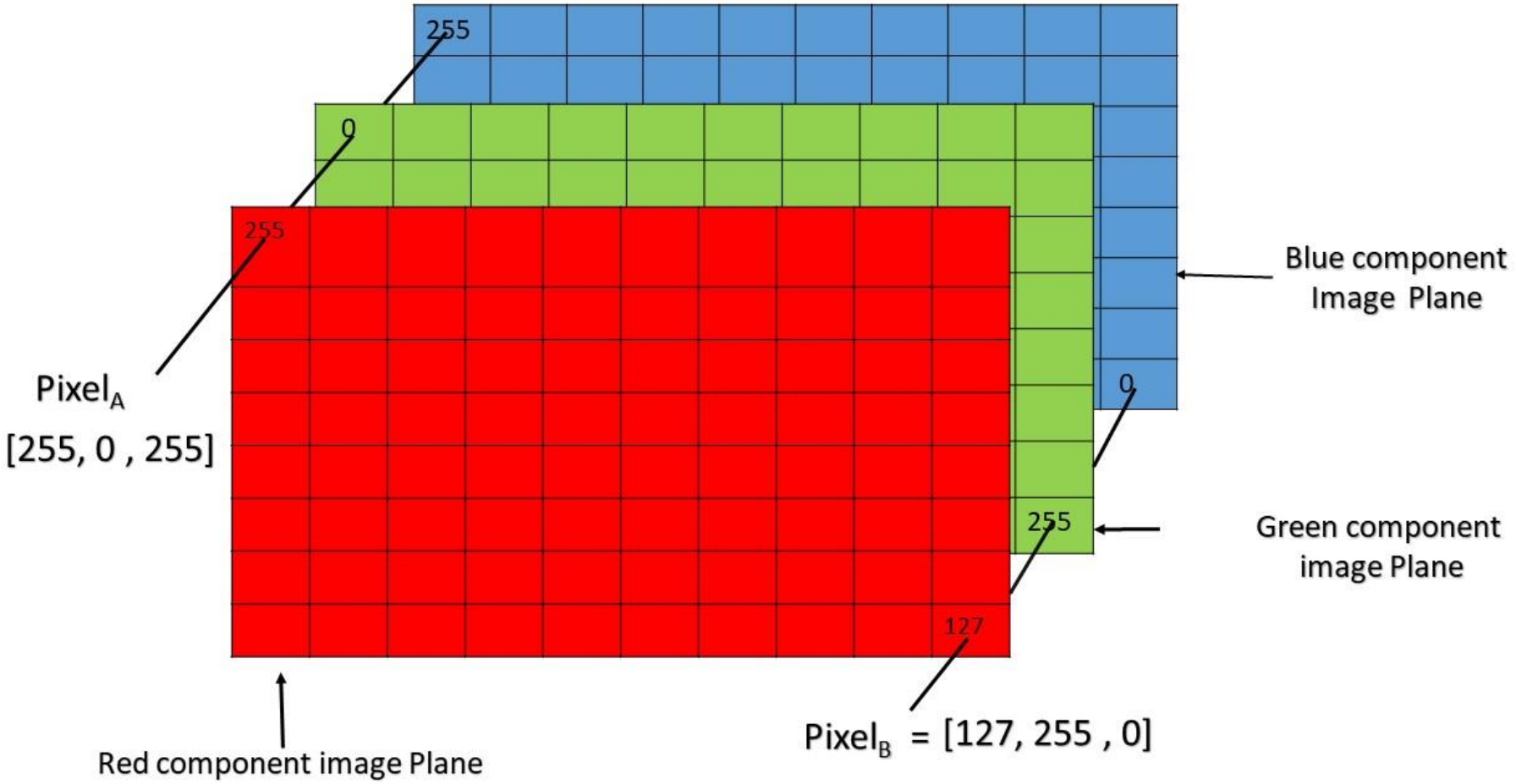
영상 처리 기초

- **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