인공지능_HW3

인공지능학부 215001 서가연

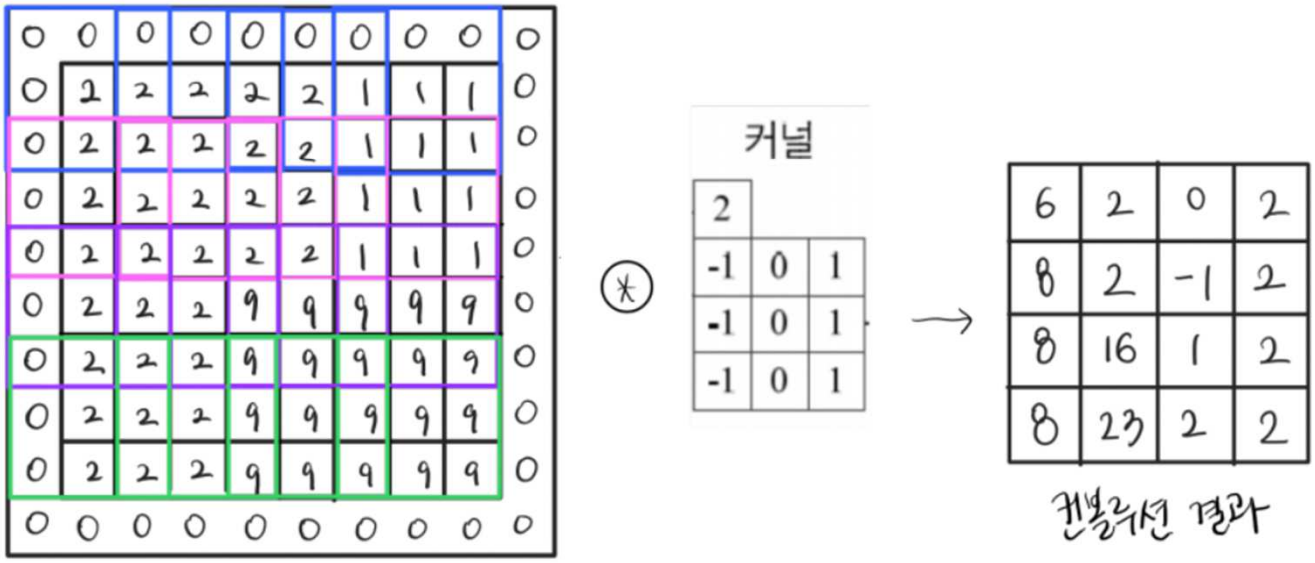
01.

	0	1	2	3	4	5	6	7
0	2	2	2	2	2	1	1	1
1	2	2	2	2	2	1	1	1
2	2	2	2	2	2	1	1	1
3	2	2	2	2	2	1	1	1
4	2	2	2	9	9	9	9	9
5	2	2	2	9	9	9	9	9
6	2	2	2	9	9	9	9	9
7	2	2	2	9	9	9	9	9

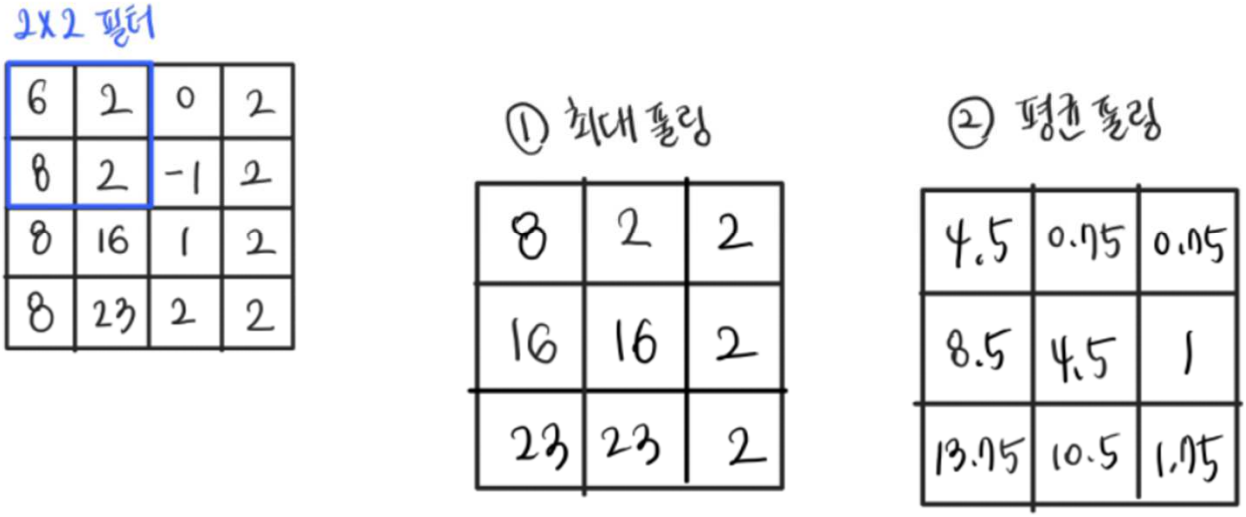
커널

2		
-1	0	1
-1	0	1
-1	0	1

1) 위 입력에 대하여 컨볼루션 결과를 작성하시오. (보폭 : 2, 제로패딩)



2) 위 컨볼루션 결과에서 최대 풀링과 평균 풀링을 적용한 결과를 각각 작성하시오. (필터 : 2x2, 보폭: 1, 패딩없음)



02. 4개의 화소를 다음과 같이 분류하는 심층 신경망 프로그램을 작성하여 훈련 및 테스트를 실시하고 결과를 분석하시오.

데이터셋 준비 4개 화소에 대한 샘플 데이터와 라벨(0, 1, 2, 3) 데이터 준비

```
# install dependencies
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam

# 특징데이터
X = np.array([[1, 1, 1, 1],
              [0, 0, 0, 0],
              [1, 0, 1, 0],
              [0, 1, 0, 1],
              [1, 0, 0, 1],
              [0, 1, 1, 0],
              [0, 0, 1, 1],
              [1, 1, 0, 0]])

# 정답데이터 => 각 클래스를 정수로 표현
y = np.array([0, 0, 1, 1, 2, 2, 3, 3])

# 라벨맵 저장
label = {0: "SOLID", 1:"VERTICAL", 2:"DIAGIONAL", 3:"HORIZONTAL"}
```

모델 생성 3개의 Dense layer를 갖는 심층 신경망 모델 생성

```
model = tf.keras.Sequential([
    Dense(30, activation='relu', input_shape=(4, )),
    Dense(18, activation='relu'),
    Dense(4, activation='softmax') # 4개 클래스로 분류
])
```

모델 학습

```
model.compile(optimizer=Adam(0.01), loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# 모델 학습
history = model.fit(X, y, epochs=20, verbose=0)

# 손실값과 정확도 출력
loss, acc = model.evaluate(X, y)
```

✓ 4.3s

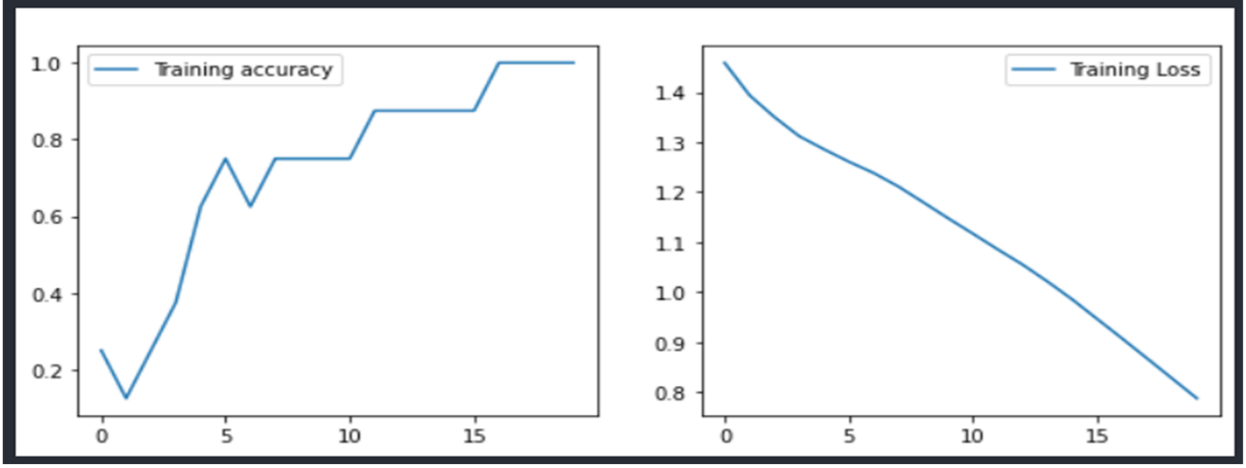
```
1/1 [==============================] - 1s 788ms/step - loss: 0.6341 - accuracy: 1.0000
```

02. 4개의 화소를 다음과 같이 분류하는 심층 신경망 프로그램을 작성하여 훈련 및 테스트를 실시하고 결과를 분석하시오.

ACC & LOSS 20번의 학습 동안, loss와 accuracy의 변화를 시각화

```python
# 학습 과정 시각화
def plot_history():
    plt.figure(figsize=(10, 4))
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Training accuracy')
    plt.title('accuracy plot', color='white')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label = 'Training Loss')
    plt.title('loss plot', color='white')
    plt.legend()

plot_history()
```

✓ 0.2s

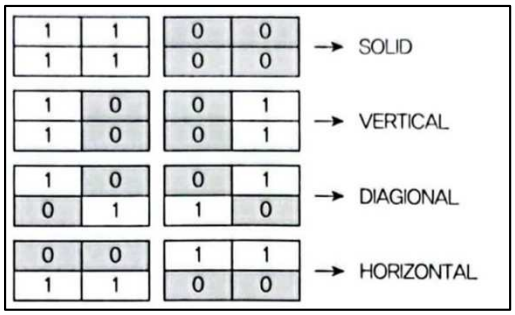


새로운 데이터로 예측 학습된 모델로 새로운 데이터의 예측 결과 확인 => 잘 예측함

```python
# 예측 결과 확인
def predict(data):
    pred = np.argmax(model.predict(data, verbose=3))
    print(f'class: {label[pred]}')

predict([[0.1, 0.1, 0.1, 0.1]])
predict([[0.8, 0.05, 0.7, 0.13]])
predict([[0.7, 0.1, 0.1, 0.54]])
predict([[0.1, 0.1, 0.5, 0.8]])
```

✓ 0.1s

```
class: SOLID
class: VERTICAL
class: DIAGIONAL
class: HORIZONTAL
```



03. 패션 아이템(fashion-MNIST)을 기본 MLP 와 심층신경망을 이용하여 분류하는 프로그램을 각각 작성한 후 훈련 및 테스트를 통해 성능을 비교하시오

데이터 준비 1) 기본 MLP , 2) 심층 신경망 공통

```
from tensorflow.keras import datasets
import tensorflow as tf
import matplotlib.pyplot as plt

from tensorflow.keras.layers import Flatten, Dense
from tensorflow.keras.optimizers import Adam

# 데이터셋 로드하기
fashion_mnist = datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
train_images.shape  # 60000개의 샘플 각각 28*28 크기


# 전처리 : 0~1 사이로 변환
train_images = train_images/255.0
test_images = test_images/255.0
```

모델 생성

1) 기본 MLP => 최소의 hidden layer 사용

```
# 기본 MLP 모델 생성
model = tf.keras.Sequential([
    Flatten(input_shape=(28, 28)),  # input
    Dense(512, activation='relu'),
    Dense(10, activation='softmax')  # output
])
```

2) 심층 신경망 => 1에 비해 hidden layer 수를 늘림

```
# DNN 모델 생성
model = tf.keras.Sequential()
model.add(tf.keras.layers.Flatten(input_shape=(28, 28)))
model.add(tf.keras.layers.Dense(512, activation='relu'))
model.add(tf.keras.layers.Dropout(0.2))
model.add(tf.keras.layers.Dense(256, activation='relu'))
model.add(tf.keras.layers.Dropout(0.2))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dropout(0.2))
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dropout(0.2))
model.add(tf.keras.layers.Dense(10, activation='softmax'))  # 0~9 예측
```

03. 패션 아이템(fashion-MNIST)을 기본 MLP 와 심층신경망을 이용하여 분류하는 프로그램을 각각 작성한 후 훈련 및 테스트를 통해 성능을 비교하시오

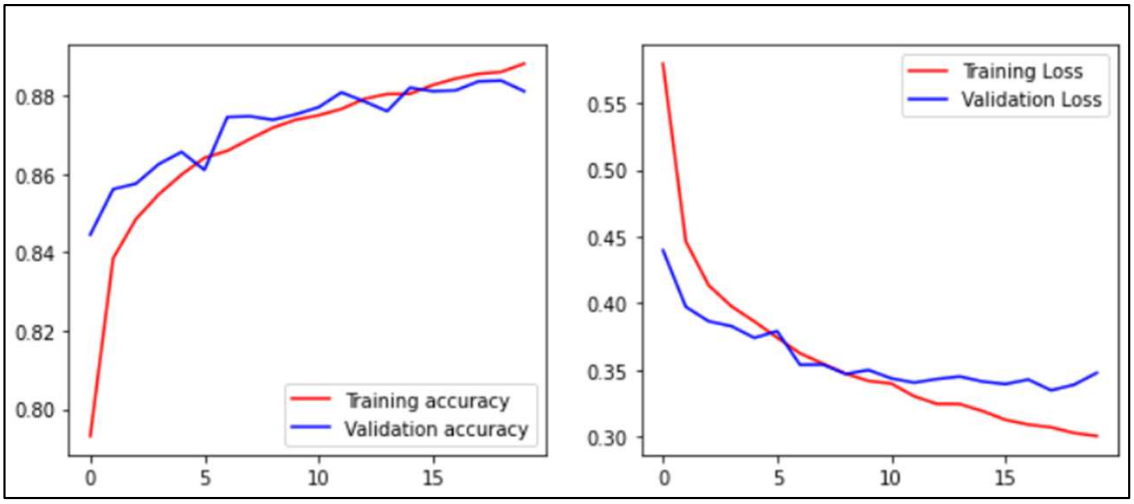
모델 학습 비교를 위해 1), 2) 모두 같은 하이퍼파라미터 설정

```
model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=['accuracy'])
history = model.fit(train_images, train_labels, epochs=20, validation_data=(test_images, test_labels))
```

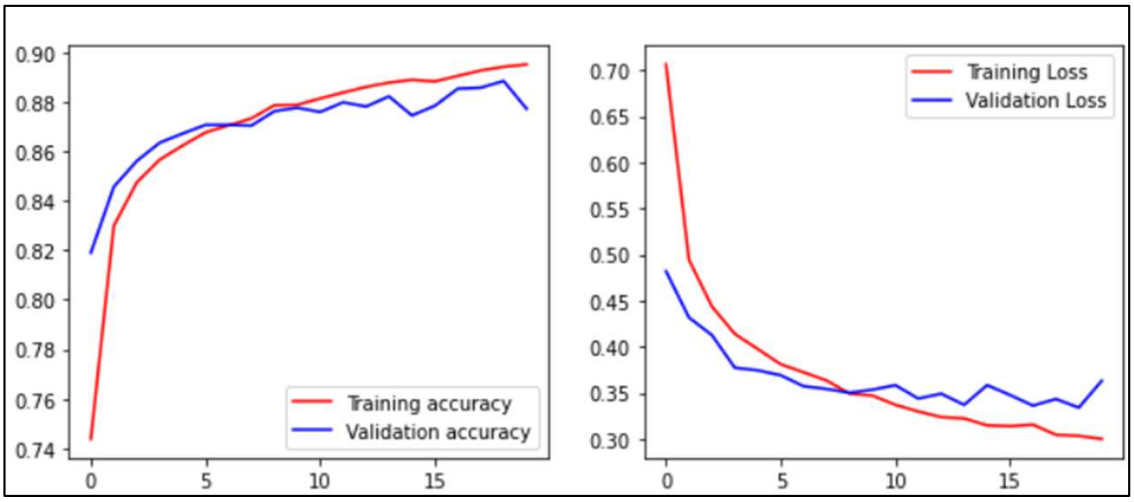
✓ 3m 4.6s

학습 결과 20번의 학습 과정에서 두 모델의 성능은 거의 비슷하게 나옴

1) 기본 MLP => 정확도 : 0.88



2) 심층 신경망 => 정확도 : 0.877



04. 붓꽃 데이터 세트를 이용하여 꽃받침 길이와 너비, 꽃잎의 길이와 너비 등 4 가지 특징을 입력으로 받아서 어떤 붓꽃인지 예측하고자 한다. 케라스로 심층신경망을 구현하고 훈련하여 테스트 하시오.

데이터 준비

```python
# install dependencies
from sklearn import datasets
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt

# data load
iris = datasets.load_iris()
X = iris['data']
y = iris['target']

# data split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print(X_train.shape)
print(X_test.shape)
print(np.unique(y_train))  # 3개의 클래스 존재
```

✓ 1.2s

```
(120, 4)
(30, 4)
[0 1 2]
```

모델링 2개의 Dense layer를 갖는 신경망 생성

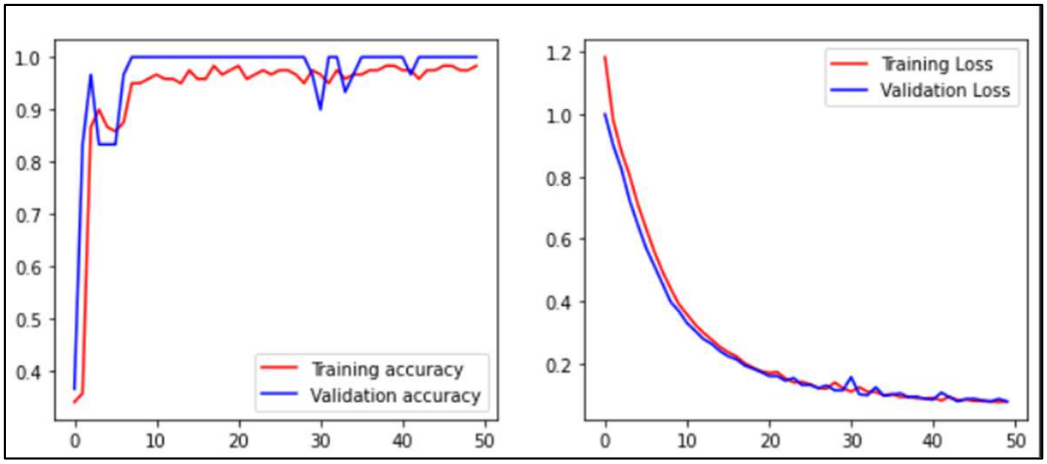
```python
import tensorflow as tf
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam

# 모델생성
model = tf.keras.Sequential([
    Dense(64, input_shape=(4, ), activation='relu'),
    Dense(32, activation='relu'),
    Dense(3, activation='softmax') # 3개의 클래스로 분류
])

model.compile(optimizer=Adam(0.003), loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model.fit(X_train, y_train, epochs=50, validation_data=(X_test, y_test), verbose=0)

print(f"loss, acc : {model.evaluate(X_test, y_test)}")
```

✓ 5.6s

```
1/1 [==============================] - 0s 18ms/step - loss: 0.0777 - accuracy: 1.0000
loss, acc : [0.07769744843244553, 1.0]
```

04. 붓꽃 데이터 세트를 이용하여 꽃받침 길이와 너비, 꽃잎의 길이와 너비 등 4 가지 특징을 입력으로 받아서 어떤 붓꽃인지 예측하고자 한다. 케라스로 심층신경망을 구현하고 훈련하여 테스트 하시오.

학습 곡선 Train, valid set 모두 높은 정확도와 낮은 손실값을 보이며, 과대적합 양상도 없음



테스트 데이터로 예측 학습된 모델로 테스트셋 예측 결과를 확인 => evaluate 결과와 동일

```
# test set으로 예측
predictions = model.predict(X_test, verbose=0)  # 확률값
pred_class = [np.argmax(y, axis=None, out=None) for y in predictions]  # 확률값이 큰 클래스로 예측

print(f'예측값 : {pred_class}')
print(f'정답 : {y_test}')
print()

acc = np.sum(pred_class == y_test) / y_test.shape[0]  # test sample 중에 맞은 샘플 개수 비율
print(f'accuracy : {acc}')
```

✓ 0.2s

```
예측값 : [1, 0, 2, 1, 1, 0, 1, 2, 1, 1, 2, 0, 0, 0, 0, 1, 2, 1, 1, 2, 0, 2, 0, 2, 2, 2, 2, 2, 0, 0]
정답 : [1 0 2 1 1 0 1 2 1 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0]

accuracy : 1.0
```

05. 와인의 화학적 측정(11개의 입력특성)을 고려하여 화이트 와인의 품질(Quailty : 0부터 10사이의 값)을 예측한다. 여기서는 다중 클래스 분류 문제로 접근해보자.

데이터 준비 모듈 임포트는 생략

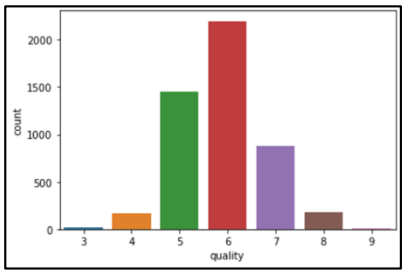
```
# data load
data = pd.read_csv("winequality-white.csv", sep=';')
data.head()
```

✓ 0.6s

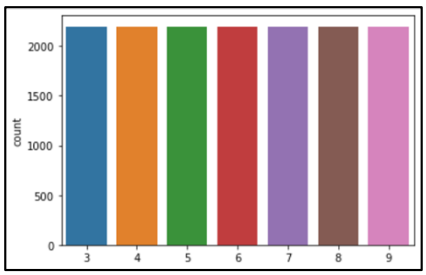
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6

데이터 전처리 표준화 + target 데이터의 불균형 해소를 위한 오버 샘플링 + target value 원핫인코딩

```
# 데이터 전처리
from imblearn.over_sampling import SMOTE
from sklearn.decomposition import PCA
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

data['quality'] -= 3  # 0~6까지의 클래스로 변경

# 특징/정답데이터 분리
X = data.drop('quality', axis=1)
y = data['quality']

# 데이터 표준화
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)


# 오버샘플링
sm = SMOTE(k_neighbors=4)
X_resampled, y_resampled = sm.fit_sample(X, list(y))

X = pd.DataFrame(X_resampled, columns=X.columns)
y = np.array(y_resampled)


# 데이터 분리
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, shuffle=True, stratify=y, random_state=42)


# 범주형 데이터를 원핫인코딩
y_train = tf.keras.utils.to_categorical(y_train, num_classes=7)
y_test = tf.keras.utils.to_categorical(y_test, num_classes=7)
```



오버샘플링

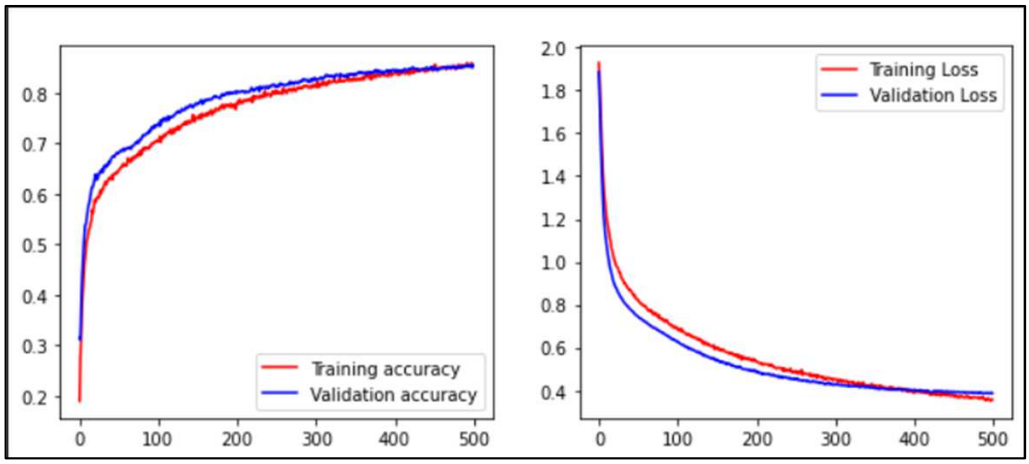


05. 와인의 화학적 측정(11개의 입력특성)을 고려하여 화이트 와인의 품질(Quailty : 0부터 10사이의 값)을 예측한다. 여기서는 다중 클래스 분류 문제로 접근해보자.

모델 생성 과대적합 방지를 위한 Dropout 추가

```
# 케라스 다중 클래스 분류 모델 생성
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(256, activation='relu', input_dim=X_train.shape[1])),
model.add(tf.keras.layers.Dropout(0.2)),
model.add(tf.keras.layers.Dense(128, activation='relu')),
model.add(tf.keras.layers.Dropout(0.2)),
model.add(tf.keras.layers.Dense(64, activation='relu')),
model.add(tf.keras.layers.Dropout(0.2)),
model.add(tf.keras.layers.Dense(32, activation='relu')),
model.add(tf.keras.layers.Dense(32, activation='relu')),
model.add(tf.keras.layers.Dropout(0.2)),
model.add(tf.keras.layers.Dense(7, activation='softmax'))  # 0-6 클래스로 분류


model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])

history = model.fit(X_train, train_Y, epochs=500, batch_size=100, validation_split=0.2, verbose=2)
```

학습 결과 오버샘플링으로 전체 샘플의 개수가 증가해서 정확도가 증가함 => 0.85



예측 결과 Train set과 Test set에 대한 모델 예측의 오차 확인
=> 0은 맞은 샘플 개수, 오차가 3 이상인 샘플의 개수는 매우 적음

```
train sample
0 : 11069
1 : 1145
2 : 91
3 : 3
4 : 0
5 : 0
6 : 0
```

```
test sample
0 : 2658
1 : 371
2 : 46
3 : 3
4 : 0
5 : 0
6 : 0
```

06. 다양한 종류의 밀에서 추출한 종자의 측정값을 기준으로 종 예측을 하는 프로그램을 작성해보자. 이것은 7개의 입력특성과 1개의 출력변수(3가지의 클래스)가 있고 210개의 샘플이 있다.

데이터 탐색

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


# data load
data = pd.read_csv("seeds_dataset.csv")
data.head()
```

✓ 0.2s

	A	P	C	LK	WK	A_Coef	LKG	target
0	15.26	14.84	0.8710	5.763	3.312	2.221	5.220	0
1	14.88	14.57	0.8811	5.554	3.333	1.018	4.956	0
2	14.29	14.09	0.9050	5.291	3.337	2.699	4.825	0
3	13.84	13.94	0.8955	5.324	3.379	2.259	4.805	0
4	16.14	14.99	0.9034	5.658	3.562	1.355	5.175	0

```
data.info()
```

✓ 0.2s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 210 entries, 0 to 209
Data columns (total 8 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   A       210 non-null    float64
 1   P       210 non-null    float64
 2   C       210 non-null    float64
 3   LK      210 non-null    float64
 4   WK      210 non-null    float64
 5   A_Coef  210 non-null    float64
 6   LKG     210 non-null    float64
 7   target  210 non-null    int64
```

```
# 3개 클래스가 고루 분포됨
data.target.value_counts()
```

✓ 0.5s

```
0    70
1    70
2    70
```

3개의 클래스가 고루 분포됨을 확인

데이터 준비 데이터 표준화 및 검증세트 분리

```
# 특징, 정답데이터 분리
X = data.drop("target", axis=1)
y = data['target']

# 특징 데이터 표준화
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
train_df = pd.DataFrame(X_scaled, index=X.index, columns=X.columns)

# 학습, 검증 데이터 분리
X_train, X_test, y_train, y_test = train_test_split(train_df, y, random_state=42, test_size=0.2)

print(X_train.shape)
print(X_test.shape)
```

✓ 0.2s

```
(168, 7)
(42, 7)
```

06. 다양한 종류의 밀에서 추출한 종자의 측정값을 기준으로 종 예측을 하는 프로그램을 작성해보자. 이것은 7개의 입력특성과 1개의 출력변수(3가지의 클래스가 있는 이진 분류 문제)가 있고 210개의 샘플이 있다.

모델링 Train set를 9(train):1(valid)로 나누어 학습 진행 => test set으로 평가 결과, 0.95 정확도

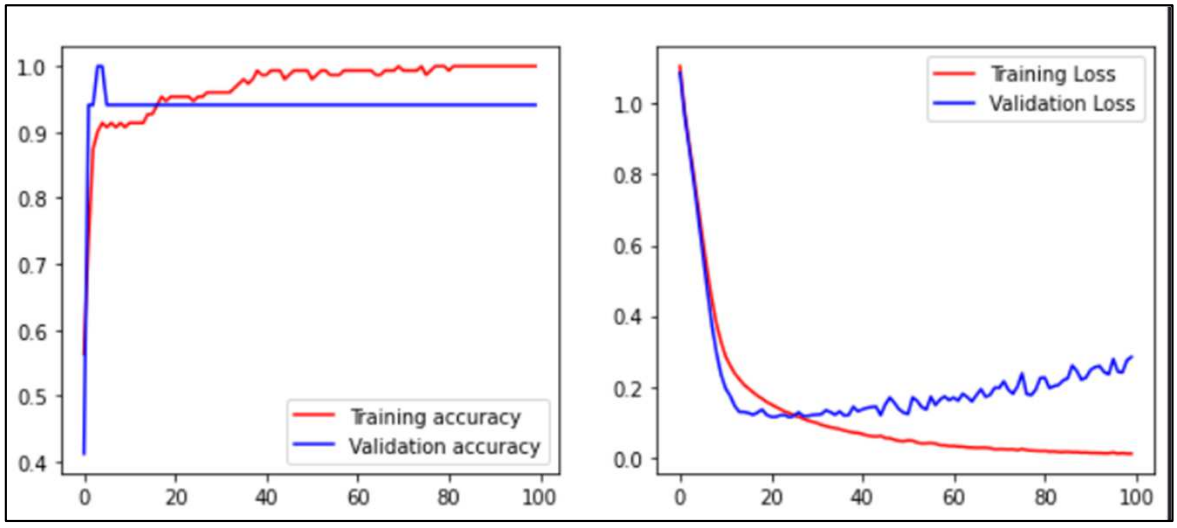
```
# 모델링
model = tf.keras.Sequential([
    tf.keras.layers.Dense(100, input_dim=7, activation='relu'),
    tf.keras.layers.Dense(50, activation='relu'),
    tf.keras.layers.Dense(25, activation='relu'),
    tf.keras.layers.Dense(3, activation='softmax')  # 3개의 클래스로 분류
])

model.compile(optimizer=tf.keras.optimizers.Adam(0.001), loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model.fit(X_train, y_train, validation_split=0.1, epochs=100, verbose=0)

val_loss, val_acc = model.evaluate(X_test, y_test)
print(f'val_loss: {val_loss}, val_acc: {val_acc}')
```

✓ 3.4s

```
2/2 [==============================] - 0s 2ms/step - loss: 0.1196 - accuracy: 0.9524
val_loss: 0.11962871253490448, val_acc: 0.9523809552192688
```

학습결과 적은 수의 학습횟수로도 좋은 성능이 확인됨

=> Loss plot에서 학습이 진행될수록, 과대적합 양상이 확인됨



07. 이전 3 번 문제에서 수행했던 패션 아이템(fashion-MNIST)을 분류하는 동일한 실험을 컨볼루션 신경망을 이용해서 시도하시오. 그리고 MLP, 심층신경망과 컨볼루션 신경망의 성능을 비교하시오.

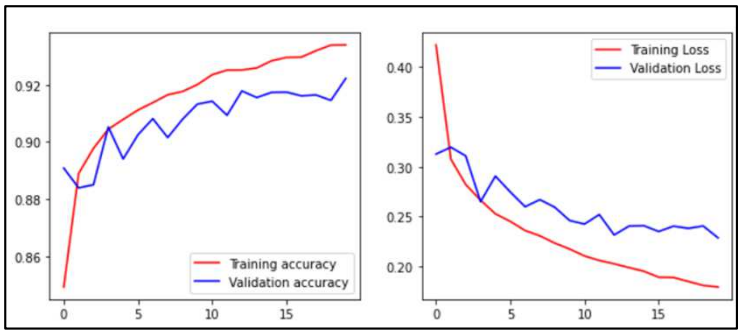
모델링 모델링 전까지 3번 문제와 동일한 작업 수행

```
model = tf.keras.models.Sequential([
    Conv2D(64, kernel_size=(3, 3), padding='Same',
                activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(),
    Dropout(0.25),

    BatchNormalization(),
    Conv2D(64, kernel_size=(3, 3), padding='Same',
                activation='relu'),
    MaxPooling2D(),
    Dropout(0.25),

    Flatten(),
    Dense(10, activation="softmax")  # output
])
```

컴파일 및 학습 과정도 MLP, DNN 모델과 동일하게 설정

```
model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=['accuracy'])
history = model.fit(train_images, train_labels, validation_data=(test_images, test_labels), epochs=20)
```

학습결과 과대적합 양상을 띔, epoch를 늘리면 정확도가 더 증가할 것으로 예상됨



epoch 20 기준

MLP : 0.88(train acc), 0.88(test acc)

DNN : 0.89(train acc), 0.87(test acc)

CNN : 0.93(train acc), 0.92(test acc)

⇒ CNN의 성능이 다른 두 모델보다 높음

⇒ 학습 횟수를 늘리면, 전체적으로 성능 증가 예상