1.[TensorFlow를 이용한 과제 설명]

> 코드와 주석을 이용한 설명

```
# K2020008: 사이킷런(sklearn)이란?
# K2020008: 사이킷런은 파이썬에서 머신러닝 분석을 할 때 유용하게 사용할 수 있는 라이브러리 입니다

# K2020008: 사이킷런에 내장되어있는 유방암 데이터 import
from sklearn.datasets import load_breast_cancer

# K2020008: sklearn에 내장된 원본 유방암 데이터 불러오기 변수 저장
cancer = load_breast_cancer()
# K2020008: 독립변수 데이터 모음(영향을 주는 변수)
data = cancer.data
# K2020008: 종속변수 데이터 모음(영향을 받는 변수)
labels = cancer.target

# K2020008: [변수 출력]
print("독립변수 -> 암에 영향을 주는 변수₩n%s" % data)
print("종속변수 -> 암진단을 받은 경우 = 1, 암진단을 받지 않은 경우 = 0₩n%s" % labels)
print("행,   열    ₩n",  data.shape)
```

```
독립변수 -> 암에 영향을 주는 변수
[[1.799e+01 1.038e+01 1.228e+02 ... 2.654e-01 4.601e-01 1.189e-01]
 [2.057e+01 1.777e+01 1.329e+02 ... 1.860e-01 2.750e-01 8.902e-02]
 [1.969e+01 2.125e+01 1.300e+02 ... 2.430e-01 3.613e-01 8.758e-02]
 ...
 [1.660e+01 2.808e+01 1.083e+02 ... 1.418e-01 2.218e-01 7.820e-02]
 [2.060e+01 2.933e+01 1.401e+02 ... 2.650e-01 4.087e-01 1.240e-01]
 [7.760e+00 2.454e+01 4.792e+01 ... 0.000e+00 2.871e-01 7.039e-02]]
종속변수 -> 암진단을 받은 경우 = 1, 암진단을 받지 않은 경우 = 0
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 0 1 1 1 1 0 1 0 0 1 1 1 1 0 1 0 0
 1 0 1 0 0 1 1 1 0 0 1 0 0 0 1 1 1 0 1 1 0 0 1 1 1 0 0 1 1 1 1 0 1 1 0 1 1
 1 1 1 1 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 1 1 0 1
 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 1 1 1 1 0 1 1 0 0 0 1 0
 1 0 1 1 1 0 1 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 1 1 0 0 1 1
 1 0 1 1 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 1 1 1 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 1 0 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1
 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 0 0 0 1 1
 1 1 0 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0
 0 1 0 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1
 1 0 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 1 1 1 1 0 1 1
 0 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1
 1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1 0 0 1 0 1 0 1 1 1 1 1 0 1 1 0 1 0 1 0 0
 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 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 0 0 0 0 0 0 1]
행,   열
 (569, 30)
```

```
# K2020008: Split data
# K2020008: train, test 데이터 분할
from sklearn.model_selection import train_test_split
# K2020008: [klearn의 train_test_split() 사용법
    # 머신러닝 모델을 학습하고 그 결과를 검증하기 위해서는 원래의 데이터를 Training, Validation, Testing의 용도로 나누어 다뤄야 한다.
    # 그렇지 않고 Training에 사용한 데이터를 검증용으로 사용하면 시험문제를 알고 있는 상태에서 공부를 하고 그 지식을 바탕으로 시험을 치루는 꼴이
    # 딥러닝을 제외하고도 다양한 기계학습과 데이터 분석 툴을 제공하는 scikit-learn 패키지 중 model_selection에는 데이터 분할을 위한 train_test_s

    # (1) Parameter
    # arrays : 분할시킬 데이터를 입력 (Python list, Numpy array, Pandas dataframe 등..)
    # test_size : 테스트 데이터셋의 비율(float)이나 갯수(int) (default = 0.25)
```

```
    # train_size : 학습 데이터셋의 비율(float)이나 갯수(int) (default = test_size의 나머지)
    # random_state : 데이터 분할시 셔플이 이루어지는데 이를 위한 시드값 (int나 RandomState로 입력)
    # shuffle : 셔플여부설정 (default = True)
    # stratify : 지정한 Data의 비율을 유지한다. 예를 들어, Label Set인 Y가 25%의 0과 75%의 1로 이루어진 Binary Set일 때, stratify=Y로 설정하면 L

    # (2) Return
    # X_train, X_test, Y_train, Y_test : arrays에 데이터와 레이블을 둘 다 넣었을 경우의 반환이며, 데이터와 레이블의 순서쌍은 유지된다.
    # X_train, X_test : arrays에 레이블 없이 데이터만 넣었을 경우의 반환
    # [출처] [Python] sklearn의 train_test_split() 사용법|작성자 Paris Lee


from keras.utils import to_categorical
# K2020008: keras.utils.np_utils.to_categorical(y, num_classes=None) 사용법
    # 클래스 벡터(정수들)를 바이너리 클래스 매트릭스로 변환한다.
    # 클래스 벡터(정수들)를 바이너리 클래스 매트릭스로 변환한다.
    # 예를들어, categorical_crossentroy  와 함께 사용함

    # [파라미터]
    # y: 매트릭스로 변환될 클래스 백터(정수는 0~num_classes)
    # num_classes: 총 클래스 수

    # 출력
    # 입력값에 대한 바이너리 행렬

# K2020008: train, test 데이터 분할하기
# K2020008: 오버피팅을 막기위해 데이터를 train, test로 분할 합시다
X_train, X_test, Y_train, Y_test = train_test_split(data, labels, test_size=0.1)
print("데이터 갯수 : {0:d}, 테스트 데이터 갯수 : {1:d}, 데이터 유형 : {2}".format(len(X_train), len(X_test), type(X_train)))
# K2020008: print("학습데이터 갯수 : {0:d}, 테스트 데이터 갯수 : {1:d}, 데이터 유형 : {2}".format(len(y_train), len(y_test), type(y_train)))
print("학습 데이터[{1}]₩n{0}".format(X_train, len(X_train)))
print("학습 레이블[{1}]₩n{0}".format(Y_train, len(Y_train)))
print("테스트 데이터[{1}]₩n{0}".format(X_test,len(X_test)))
print("테스트 레이블[{1}]₩n{0}".format(Y_test,len(Y_test)))
```

```
데이터 갯수 : 512, 테스트 데이터 갯수 : 57, 데이터 유형 : <class 'numpy.ndarray'>
학습 데이터[512]
[[1.245e+01 1.570e+01 8.257e+01 ... 1.741e-01 3.985e-01 1.244e-01]
 [1.328e+01 2.028e+01 8.732e+01 ... 1.492e-01 3.739e-01 1.027e-01]
 [1.128e+01 1.339e+01 7.300e+01 ... 8.611e-02 2.102e-01 6.784e-02]
 ...
 [1.571e+01 1.393e+01 1.020e+02 ... 1.374e-01 2.723e-01 7.071e-02]
 [1.245e+01 1.641e+01 8.285e+01 ... 1.342e-01 3.231e-01 1.034e-01]
 [1.793e+01 2.448e+01 1.152e+02 ... 1.136e-01 2.504e-01 7.948e-02]]
학습 레이블[512]
[0 0 1 0 1 0 1 1 1 0 0 0 1 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 0 1 0 1 1 1 1 0
 1 0 0 1 1 0 1 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 0 0 0 1 1 0 1 0 0
 1 1 1 1 0 1 1 0 1 0 1 0 1 1 1 0 1 1 1 1 0 1 1 1 0 0 1 1 0 1 1 1 1 1 0 1 1
 0 1 1 0 1 0 1 0 1 1 1 1 1 0 1 1 0 1 0 1 0 0 1 1 1 1 0 1 1 0 1 1 1 0 0 0 1
 1 0 0 0 0 1 1 0 0 1 1 1 0 0 1 1 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 0 1 0 1 0
 1 1 1 0 1 1 1 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 0 0 0 0 0 1 0 1
 1 0 1 0 1 1 0 1 1 1 1 1 0 1 1 1 0 1 0 0 1 1 1 0 1 1 1 1 0 0 0 0 1 1 0 1 0
 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1 1 1 1 1 0 1 1 0 0 0 0 0 0 0 1 1 1 0 1 0
 0 1 1 1 1 0 1 0 1 0 1 1 1 0 1 0 0 0 0 0 1 1 1 1 0 1 1 1 1 0 1 0 0 0 1 0 0
 1 0 1 1 1 1 0 0 0 1 1 1 1 0 1 0 0 1 0 1 1 1 1 0 0 1 1 1 0 1 1 0 1 1 1 1 1
 0 0 1 1 0 0 1 1 1 1 0 0 1 0 1 0 1 1 1 0 1 1 1 1 0 0 1 0 1 1 1 1 0 0 1 1 1
 0 1 1 1 1 0 0 1 1 0 1 1 1 1 1 0 0 1 1 1 0 0 0 0 1 0 1 0 1 1 0 1 1 1 0 0 1
 0 0 0 0 0 0 0 1 0 1 0 1 1 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0
 0 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 0 0 0 1 0 1 0 1 0 1 1 0]
테스트 데이터[57]
[[2.009e+01 2.386e+01 1.347e+02 ... 1.923e-01 3.294e-01 9.469e-02]
 [1.311e+01 2.254e+01 8.702e+01 ... 1.126e-01 4.128e-01 1.076e-01]
 [1.429e+01 1.682e+01 9.030e+01 ... 3.333e-02 2.458e-01 6.120e-02]
 ...
 [1.287e+01 1.621e+01 8.238e+01 ... 5.780e-02 3.604e-01 7.062e-02]
 [1.364e+01 1.634e+01 8.721e+01 ... 8.586e-02 2.346e-01 8.025e-02]
 [1.205e+01 2.272e+01 7.875e+01 ... 1.092e-01 2.191e-01 9.349e-02]]
테스트 레이블[57]
[0 1 1 1 1 1 0 1 1 0 0 1 1 0 0 1 0 1 0 0 1 0 0 1 1 1 1 1 1 0 0 1 0 1 0 0 1
 1 0 0 0 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1]
```

```
# K2020008: Construct model
# K2020008: tf.keras는 케라스 API 명세의 텐서플로 구현입니다.
# K2020008: tf.keras는 머신러닝 모델을 만들고 훈련하기 위한 고수준 API로서 텐서플로의 특수 기능을 모두 지원합니다.
# K2020008: 여기에는 즉시 실행, tf.data 파이프라인(pipeline), Estimators가 포함됩니다.
```

```
# K2020008: tf.keras를 이용하면 유연성과 성능을 손해보지 않고 텐서플로를 쉽게 사용할 수 있습니다.
# K2020008: tf.keras를 임포트하여 텐서플로 프로그램을 시작합니다:
import tensorflow as tf
# K2020008: [간단한 모델 만들기]
# K2020008: Sequential 모델
    # 케라스에서는 층(layer)을 조합하여 모델(model)을 만듭니다.
    # 모델은 (일반적으로) 층의 그래프입니다. 가장 흔한 모델 구조는 층을 차례대로 쌓은 tf.keras.Sequential 모델입니다.
    # 간단한 완전 연결(fully-connected) 네트워크(즉, 다층 퍼셉트론(multi-layer perceptron))를 만들어 만들기
    # Sequential 모델은 레이어를 선형으로 연결하여 구성합니다.
    # 레이어 인스턴스를 생성자에게 넘겨줌으로써 Sequential 모델을 구성할 수 있습니다.
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Activation

n_input = 30
n_hidden_1 = 128
n_hidden_2 = 64
n_hidden_3 = 32
n_hidden_4 = 16
n_classes =1
# K2020008 : 입력층-은닉층-은닉층-출력층의 5층 구조
# K2020008 : 깊은 신경망은 ReLU 적용
# K2020008 : 로지스틱 시그모이드
model=Sequential([
                  Flatten(input_shape=(n_input,)),
                  Dense(n_hidden_1,activation='relu'),
                  Dense(n_hidden_2,activation='relu'),
                  Dense(n_hidden_3,activation='relu'),
                  Dense(n_hidden_4,activation='relu'),
                  Dense(n_classes,activation='sigmoid'),
])
```

```
# K2020008 : Configure optimizer and loss function
# K2020008 : 모델을 학습시키기 이전에,
# K2020008 : compile 메소드를 통해서 학습 방식에 대한 환경설정을 해야 합니다.
# K2020008 : 다음의 세 개의 인자를 입력으로 받습니다.
```

```
    # 정규화기 (optimizer) -> 정규화기에 대한 문자열 식별자 또는 Optimizer(SGD) 클래스의 인스턴스를 사용할 수 있습니다.
    # [참고]: 정규화기손실 함수 (loss function). 모델이 최적화에 사용되는 목적 함수입니다.
    # [binary_crossentropy 같은 기존의 손실 함수의 문자열 식별자 또는 목적 함수를 사용할 수 있습니다.
    # [참고]: 손실 기준(metric) 리스트. 분류 문제에 대해서는 metrics=['accuracy']로 설정합니다.
    # 기준은 문자열 식별자 또는 사용자 정의 기준 함수를 사용할 수 있습니다.

# K2020008 : For a binary classification problem
model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.001),
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

```
# K2020008 : Training (학습)
# K2020008 : 케라스 모델들은 입력 데이터와 라벨로 구성된 Numpy 배열 위에서 이루어집니다.
    # 모델을 학습기키기 위해서는 일반적으로 fit함수를 사용합니다.
# K2020008 : 학습 진행 (200회)
# K2020008 : 배치 Sample 32개
hist=model.fit(X_train,Y_train,epochs=200, batch_size=8, shuffle=True)
# K2020008 : 학습이 잘 되는지 확인하기 위한 내용출력
# K2020008 : 최종 (Cost 계산 / accuracy 계산)
# K2020008 : loss은 0에 가깝고,  accuracy는 1에 근접해야 좋은 학습 결과임
```

```
Epoch 1/200
64/64 [==============================] - 0s 1ms/step - loss: 2.8476 - accuracy: 0.6328
Epoch 2/200
64/64 [==============================] - 0s 1ms/step - loss: 0.7730 - accuracy: 0.6934
Epoch 3/200
64/64 [==============================] - 0s 1ms/step - loss: 0.5173 - accuracy: 0.7891
Epoch 4/200
64/64 [==============================] - 0s 1ms/step - loss: 0.4177 - accuracy: 0.8496
Epoch 5/200
64/64 [==============================] - 0s 1ms/step - loss: 0.5344 - accuracy: 0.7754
Epoch 6/200
64/64 [==============================] - 0s 1ms/step - loss: 0.4656 - accuracy: 0.8184
Epoch 7/200
64/64 [==============================] - 0s 1ms/step - loss: 0.6660 - accuracy: 0.7578
Epoch 8/200
64/64 [==============================] - 0s 2ms/step - loss: 0.4557 - accuracy: 0.8301
Epoch 9/200
64/64 [==============================] - 0s 1ms/step - loss: 0.3934 - accuracy: 0.8438
Epoch 10/200
64/64 [==============================] - 0s 1ms/step - loss: 0.3629 - accuracy: 0.8457
Epoch 11/200
64/64 [==============================] - 0s 1ms/step - loss: 0.3586 - accuracy: 0.8555
Epoch 12/200
64/64 [==============================] - 0s 1ms/step - loss: 0.3292 - accuracy: 0.8750
Epoch 13/200
64/64 [==============================] - 0s 1ms/step - loss: 0.3190 - accuracy: 0.8730
Epoch 14/200
64/64 [==============================] - 0s 1ms/step - loss: 0.3451 - accuracy: 0.8711
Epoch 15/200
64/64 [==============================] - 0s 1ms/step - loss: 0.3290 - accuracy: 0.8691
Epoch 16/200
64/64 [==============================] - 0s 1ms/step - loss: 0.3314 - accuracy: 0.8613
Epoch 17/200
64/64 [==============================] - 0s 1ms/step - loss: 0.3074 - accuracy: 0.8867
Epoch 18/200
64/64 [==============================] - 0s 1ms/step - loss: 0.3149 - accuracy: 0.8848
Epoch 19/200
64/64 [==============================] - 0s 1ms/step - loss: 0.3586 - accuracy: 0.8555
Epoch 20/200
64/64 [==============================] - 0s 1ms/step - loss: 0.3127 - accuracy: 0.8750
Epoch 21/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2871 - accuracy: 0.8984
```

```
Epoch 22/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2842 - accuracy: 0.8809
Epoch 23/200
64/64 [==============================] - 0s 1ms/step - loss: 0.3143 - accuracy: 0.8809
Epoch 24/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2870 - accuracy: 0.8867
Epoch 25/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2798 - accuracy: 0.8887
Epoch 26/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2736 - accuracy: 0.8926
Epoch 27/200
64/64 [==============================] - 0s 1ms/step - loss: 0.3032 - accuracy: 0.8867
Epoch 28/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2947 - accuracy: 0.8809
Epoch 29/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2555 - accuracy: 0.9043
Epoch 30/200
64/64 [==============================] - 0s 2ms/step - loss: 0.2736 - accuracy: 0.8906
Epoch 31/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2836 - accuracy: 0.8828
Epoch 32/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2703 - accuracy: 0.8984
Epoch 33/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2932 - accuracy: 0.8691
Epoch 34/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2452 - accuracy: 0.9082
Epoch 35/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2650 - accuracy: 0.8906
Epoch 36/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2764 - accuracy: 0.8809
Epoch 37/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2567 - accuracy: 0.9023
Epoch 38/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2635 - accuracy: 0.8984
Epoch 39/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2851 - accuracy: 0.8770
Epoch 40/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2485 - accuracy: 0.9062
Epoch 41/200
64/64 [==============================] - 0s 2ms/step - loss: 0.2796 - accuracy: 0.8984
Epoch 42/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2794 - accuracy: 0.8926
```

```
Epoch 43/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2578 - accuracy: 0.8945
Epoch 44/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2531 - accuracy: 0.8984
Epoch 45/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2487 - accuracy: 0.9062
Epoch 46/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2563 - accuracy: 0.9043
Epoch 47/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2526 - accuracy: 0.8926
Epoch 48/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2657 - accuracy: 0.8984
Epoch 49/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2796 - accuracy: 0.8848
Epoch 50/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2532 - accuracy: 0.8887
Epoch 51/200
64/64 [==============================] - 0s 2ms/step - loss: 0.2540 - accuracy: 0.8965
Epoch 52/200
64/64 [==============================] - 0s 2ms/step - loss: 0.2703 - accuracy: 0.9023
Epoch 53/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2335 - accuracy: 0.9082
Epoch 54/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2737 - accuracy: 0.8926
Epoch 55/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2453 - accuracy: 0.9043
Epoch 56/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2344 - accuracy: 0.9082
Epoch 57/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2199 - accuracy: 0.9160
Epoch 58/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2195 - accuracy: 0.9121
Epoch 59/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2308 - accuracy: 0.9141
Epoch 60/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2267 - accuracy: 0.9102
Epoch 61/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2473 - accuracy: 0.8984
Epoch 62/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2472 - accuracy: 0.9004
Epoch 63/200
64/64 [==============================] - 0s 2ms/step - loss: 0.2354 - accuracy: 0.9121
```

```
Epoch 64/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2599 - accuracy: 0.9082
Epoch 65/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2495 - accuracy: 0.8984
Epoch 66/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2414 - accuracy: 0.9062
Epoch 67/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2356 - accuracy: 0.9082
Epoch 68/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2427 - accuracy: 0.9082
Epoch 69/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2357 - accuracy: 0.9121
Epoch 70/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2533 - accuracy: 0.8887
Epoch 71/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2417 - accuracy: 0.9102
Epoch 72/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2511 - accuracy: 0.8906
Epoch 73/200
64/64 [==============================] - 0s 2ms/step - loss: 0.2637 - accuracy: 0.8906
Epoch 74/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2475 - accuracy: 0.8984
Epoch 75/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2384 - accuracy: 0.9082
Epoch 76/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2228 - accuracy: 0.9023
Epoch 77/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2398 - accuracy: 0.9023
Epoch 78/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2289 - accuracy: 0.9082
Epoch 79/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2479 - accuracy: 0.8945
Epoch 80/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2360 - accuracy: 0.9121
Epoch 81/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2443 - accuracy: 0.9043
Epoch 82/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2356 - accuracy: 0.9102
Epoch 83/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2367 - accuracy: 0.9004
Epoch 84/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2327 - accuracy: 0.9043
```

```
Epoch 85/200
64/64 [==============================] - 0s 2ms/step - loss: 0.2338 - accuracy: 0.9121
Epoch 86/200
64/64 [==============================] - 0s 2ms/step - loss: 0.2501 - accuracy: 0.8984
Epoch 87/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2196 - accuracy: 0.9141
Epoch 88/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2289 - accuracy: 0.9102
Epoch 89/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2283 - accuracy: 0.8965
Epoch 90/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2384 - accuracy: 0.9004
Epoch 91/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2247 - accuracy: 0.9121
Epoch 92/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2215 - accuracy: 0.9102
Epoch 93/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2283 - accuracy: 0.9043
Epoch 94/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2410 - accuracy: 0.9023
Epoch 95/200
64/64 [==============================] - 0s 2ms/step - loss: 0.2314 - accuracy: 0.9102
Epoch 96/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2268 - accuracy: 0.9121
Epoch 97/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2340 - accuracy: 0.9102
Epoch 98/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2483 - accuracy: 0.9062
Epoch 99/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2264 - accuracy: 0.9102
Epoch 100/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2295 - accuracy: 0.9141
Epoch 101/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2432 - accuracy: 0.9102
Epoch 102/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2307 - accuracy: 0.9082
Epoch 103/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2449 - accuracy: 0.9004
Epoch 104/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2440 - accuracy: 0.9102
Epoch 105/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2272 - accuracy: 0.8984
```

```
Epoch 106/200
64/64 [==============================] - 0s 2ms/step - loss: 0.2273 - accuracy: 0.9082
Epoch 107/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2270 - accuracy: 0.9082
Epoch 108/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2230 - accuracy: 0.9141
Epoch 109/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2212 - accuracy: 0.9121
Epoch 110/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2341 - accuracy: 0.9141
Epoch 111/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2307 - accuracy: 0.9043
Epoch 112/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2260 - accuracy: 0.9062
Epoch 113/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2396 - accuracy: 0.9004
Epoch 114/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2181 - accuracy: 0.9121
Epoch 115/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2445 - accuracy: 0.9180
Epoch 116/200
64/64 [==============================] - 0s 2ms/step - loss: 0.2563 - accuracy: 0.9023
Epoch 117/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2209 - accuracy: 0.9238
Epoch 118/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2282 - accuracy: 0.9062
Epoch 119/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2310 - accuracy: 0.9023
Epoch 120/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2180 - accuracy: 0.9082
Epoch 121/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2248 - accuracy: 0.9141
Epoch 122/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2280 - accuracy: 0.9082
Epoch 123/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2094 - accuracy: 0.9219
Epoch 124/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2262 - accuracy: 0.9102
Epoch 125/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2252 - accuracy: 0.9141
Epoch 126/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2094 - accuracy: 0.9141
```

```
Epoch 127/200
64/64 [==============================] - 0s 2ms/step - loss: 0.2343 - accuracy: 0.9023
Epoch 128/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2336 - accuracy: 0.9160
Epoch 129/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2204 - accuracy: 0.9180
Epoch 130/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2272 - accuracy: 0.9160
Epoch 131/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2208 - accuracy: 0.9141
Epoch 132/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2226 - accuracy: 0.9160
Epoch 133/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2255 - accuracy: 0.9004
Epoch 134/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2139 - accuracy: 0.9199
Epoch 135/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2011 - accuracy: 0.9219
Epoch 136/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2279 - accuracy: 0.9062
Epoch 137/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2109 - accuracy: 0.9160
Epoch 138/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2172 - accuracy: 0.9102
Epoch 139/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2386 - accuracy: 0.9043
Epoch 140/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2193 - accuracy: 0.9160
Epoch 141/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2272 - accuracy: 0.9121
Epoch 142/200
64/64 [==============================] - 0s 2ms/step - loss: 0.2274 - accuracy: 0.9062
Epoch 143/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2345 - accuracy: 0.9121
Epoch 144/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2172 - accuracy: 0.9180
Epoch 145/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2126 - accuracy: 0.9180
Epoch 146/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2326 - accuracy: 0.9082
Epoch 147/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2178 - accuracy: 0.9160
Epoch 148/200
```

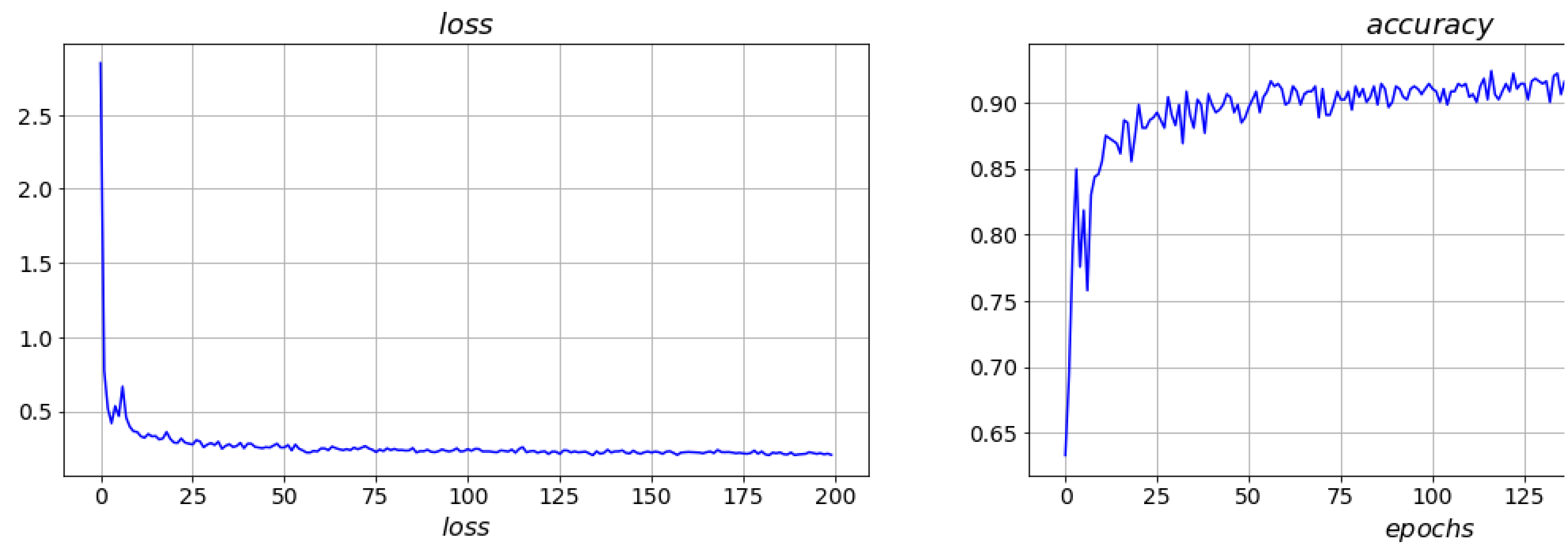
```
Epoch 148/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2105 - accuracy: 0.9297
Epoch 149/200
64/64 [==============================] - 0s 2ms/step - loss: 0.2213 - accuracy: 0.9219
Epoch 150/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2262 - accuracy: 0.9102
Epoch 151/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2194 - accuracy: 0.9082
Epoch 152/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2259 - accuracy: 0.9238
Epoch 153/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2218 - accuracy: 0.9121
Epoch 154/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2100 - accuracy: 0.9102
Epoch 155/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2256 - accuracy: 0.9043
Epoch 156/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2282 - accuracy: 0.9082
Epoch 157/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2168 - accuracy: 0.9102
Epoch 158/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2031 - accuracy: 0.9121
Epoch 159/200
64/64 [==============================] - 0s 2ms/step - loss: 0.2190 - accuracy: 0.9102
Epoch 160/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2207 - accuracy: 0.9062
Epoch 161/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2234 - accuracy: 0.9043
Epoch 162/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2217 - accuracy: 0.9141
Epoch 163/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2199 - accuracy: 0.9160
Epoch 164/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2192 - accuracy: 0.9121
Epoch 165/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2139 - accuracy: 0.9082
Epoch 166/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2222 - accuracy: 0.9141
Epoch 167/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2261 - accuracy: 0.9102
Epoch 168/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2158 - accuracy: 0.9180
Epoch 169/200
```

```
Epoch 169/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2374 - accuracy: 0.9141
Epoch 170/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2228 - accuracy: 0.9141
Epoch 171/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2218 - accuracy: 0.9082
Epoch 172/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2229 - accuracy: 0.9160
Epoch 173/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2195 - accuracy: 0.9062
Epoch 174/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2148 - accuracy: 0.9199
Epoch 175/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2177 - accuracy: 0.9180
Epoch 176/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2135 - accuracy: 0.9023
Epoch 177/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2111 - accuracy: 0.9180
Epoch 178/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2160 - accuracy: 0.9141
Epoch 179/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2332 - accuracy: 0.9141
Epoch 180/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2111 - accuracy: 0.9180
Epoch 181/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2271 - accuracy: 0.9121
Epoch 182/200
64/64 [==============================] - 0s 2ms/step - loss: 0.2074 - accuracy: 0.9121
Epoch 183/200
64/64 [==============================] - 0s 2ms/step - loss: 0.2011 - accuracy: 0.9238
Epoch 184/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2192 - accuracy: 0.9160
Epoch 185/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2144 - accuracy: 0.9160
Epoch 186/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2207 - accuracy: 0.9141
Epoch 187/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2084 - accuracy: 0.9160
Epoch 188/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2062 - accuracy: 0.9258
Epoch 189/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2198 - accuracy: 0.9258
Epoch 190/200
```

```
64/64 [==============================] - 0s 1ms/step - loss: 0.2021 - accuracy: 0.9219
Epoch 191/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2067 - accuracy: 0.9238
Epoch 192/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2082 - accuracy: 0.9180
Epoch 193/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2109 - accuracy: 0.9180
Epoch 194/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2223 - accuracy: 0.9180
Epoch 195/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2162 - accuracy: 0.9062
Epoch 196/200
64/64 [==============================] - 0s 2ms/step - loss: 0.2105 - accuracy: 0.9199
Epoch 197/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2152 - accuracy: 0.9199
Epoch 198/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2075 - accuracy: 0.9180
Epoch 199/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2123 - accuracy: 0.9160
Epoch 200/200
64/64 [==============================] - 0s 1ms/step - loss: 0.2038 - accuracy: 0.9160
```

```
# K2020008 : Matplotlib으로 결과 시각화
import matplotlib.pyplot as plt

# K2020008 : Matplotlib는 파이썬에서 데이타를 차트나 플롯(Plot)으로 그려주는 라이브러리
# K2020008 : 최초 창의 크기 -> 가로20 세로 5인치로 설정, wspace의 경우는 subplot간의 간격 0.2
plt.figure(figsize=(20,5))
plt.subplots_adjust(wspace=0.2)

# K2020008 : plt.subplot(nrow,ncol,pos) _> 여러개의 그래프를 그리고 싶을때 (1행, 2열, 위치)
# K2020008 : 손실률 그래프 추이
# K2020008 : 타이틀,라벨 달기 및 폰트 크기 설정
# K2020008 : fontsize 설정
plt.subplot(1,2,1)
plt.title("$loss$",fontsize = 18)
plt.plot(hist.history['loss'], 'b', label='train loss')
plt.grid()
```

```
plt.xlabel("$epochs$", fontsize = 16)
plt.xlabel("$loss$", fontsize = 16)
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)

# K2020008 : 정확도 그래프 추이
# K2020008 : 타이틀,라벨 달기 및 폰트 크기 설정
plt.subplot(1,2,2)
plt.title("$accuracy$", fontsize = 18)
plt.plot(hist.history['accuracy'], 'b', label='train accuracy')
plt.grid()
plt.xlabel("$epochs$", fontsize = 16)
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)

# K2020008 : 그래프 출력
plt.show()
```



2.[PyTorch를 이용한 과제 설명]

> 코드와 주석을 이용한 설명

```
# K2020008: 사이킷런(sklearn)이란?
# K2020008: 사이킷런은 파이썬에서 머신러닝 분석을 할 때 유용하게 사용할 수 있는 라이브러리 입니다

# K2020008: 사이킷런에 내장되어있는 유방암 데이터 import
from sklearn.datasets import load_breast_cancer

# K2020008: sklearn에 내장된 원본 유방암 데이터 불러오기 변수 저장
cancer = load_breast_cancer()
# K2020008: 독립변수 데이터 모음(영향을 주는 변수)
data = cancer.data
# K2020008: 종속변수 데이터 모음(영향을 받는 변수)
labels = cancer.target

# print('{0:2d} {1:3d} {2:4d}'.format(x, x*x, x*x*x))
# print('0:2d 1:3d 2:4d' % (x, x*x, x*x*x))

print("독립변수 -> 암에 영향을 주는 변수₩n%s" % data)
print("종속변수 -> 암진단을 받은 경우 = 1, 암진단을 받지 않은 경우 = 0₩n%s" % labels)
print("행,   열    ₩n",  data.shape)
```

```
독립변수 -> 암에 영향을 주는 변수
[[1.799e+01 1.038e+01 1.228e+02 ... 2.654e-01 4.601e-01 1.189e-01]
 [2.057e+01 1.777e+01 1.329e+02 ... 1.860e-01 2.750e-01 8.902e-02]
 [1.969e+01 2.125e+01 1.300e+02 ... 2.430e-01 3.613e-01 8.758e-02]
 ...
 [1.660e+01 2.808e+01 1.083e+02 ... 1.418e-01 2.218e-01 7.820e-02]
 [2.060e+01 2.933e+01 1.401e+02 ... 2.650e-01 4.087e-01 1.240e-01]
 [7.760e+00 2.454e+01 4.792e+01 ... 0.000e+00 2.871e-01 7.039e-02]]
종속변수 -> 암진단을 받은 경우 = 1, 암진단을 받지 않은 경우 = 0
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 0 1 1 1 1 0 1 0 0 1 1 1 1 0 1 0 0
 1 0 1 0 0 1 1 1 0 0 1 0 0 0 1 1 1 0 1 1 0 0 1 1 1 0 0 1 1 1 1 0 1 1 0 1 1
 1 1 1 1 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 1 1 0 1
 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 1 1 1 1 0 1 1 0 0 0 1 0
 1 0 1 1 1 0 1 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 1 1 0 0 1 1
 1 0 1 1 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 1 1 1 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 1 0 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1
 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 0 0 0 1 1
 1 1 0 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0
 0 1 0 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1
 1 0 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 1 1 1 1 0 1 1
 0 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1
 1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1 0 0 1 0 1 0 1 1 1 1 1 0 1 1 0 1 0 1 0 0
 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 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 0 0 0 0 0 0 1]
행,   열
 (569, 30)
```

```
# K2020008: Split data
# K2020008: train, test 데이터 분할
from sklearn.model_selection import train_test_split
# K2020008: [klearn의 train_test_split() 사용법
    # 머신러닝 모델을 학습하고 그 결과를 검증하기 위해서는 원래의 데이터를 Training, Validation, Testing의 용도로 나누어 다뤄야 한다.
    # 그렇지 않고 Training에 사용한 데이터를 검증용으로 사용하면 시험문제를 알고 있는 상태에서 공부를 하고 그 지식을 바탕으로 시험을 치루는 꼴이
    # 딥러닝을 제외하고도 다양한 기계학습과 데이터 분석 툴을 제공하는 scikit-learn 패키지 중 model_selection에는 데이터 분할을 위한 train_test_s

    # (1) Parameter
    # arrays : 분할시킬 데이터를 입력 (Python list, Numpy array, Pandas dataframe 등..)
    # test_size : 테스트 데이터셋의 비율(float)이나 갯수(int) (default = 0.25)
```

```
    # train_size : 학습 데이터셋의 비율(float)이나 갯수(int) (default = test_size의 나머지)
    # random_state : 데이터 분할시 셔플이 이루어지는데 이를 위한 시드값 (int나 RandomState로 입력)
    # shuffle : 셔플여부설정 (default = True)
    # stratify : 지정한 Data의 비율을 유지한다. 예를 들어, Label Set인 Y가 25%의 0과 75%의 1로 이루어진 Binary Set일 때, stratify=Y로 설정하면 L

    # (2) Return
    # X_train, X_test, Y_train, Y_test : arrays에 데이터와 레이블을 둘 다 넣었을 경우의 반환이며, 데이터와 레이블의 순서쌍은 유지된다.
    # X_train, X_test : arrays에 레이블 없이 데이터만 넣었을 경우의 반환
    # [출처] [Python] sklearn의 train_test_split() 사용법|작성자 Paris Lee

# K2020008: train, test 데이터 분할하기
# K2020008: 오버피팅을 막기위해 데이터를 train, test로 분할 합시다
x_train, x_test, y_train, y_test = train_test_split(data, labels, test_size=0.1)
print("데이터 갯수 : {0:d}, 테스트 데이터 갯수 : {1:d}, 데이터 유형 : {2}".format(len(x_train), len(x_test), type(x_train)))
# K2020008: print("학습데이터 갯수 : {0:d}, 테스트 데이터 갯수 : {1:d}, 데이터 유형 : {2}".format(len(y_train), len(y_test), type(y_train)))
print("학습 데이터[{1}]₩n{0}".format(x_train, len(x_train)))
print("학습 레이블[{1}]₩n{0}".format(y_train, len(y_train)))
print("테스트 데이터[{1}]₩n{0}".format(x_test,len(x_test)))
print("테스트 레이블[{1}]₩n{0}".format(y_test,len(y_test)))
```

```
데이터 갯수 : 512, 테스트 데이터 갯수 : 57, 데이터 유형 : <class 'numpy.ndarray'>
학습 데이터[512]
[[1.510e+01 2.202e+01 9.726e+01 ... 1.530e-01 2.675e-01 7.873e-02]
 [1.530e+01 2.527e+01 1.024e+02 ... 2.024e-01 4.027e-01 9.876e-02]
 [1.359e+01 1.784e+01 8.624e+01 ... 5.185e-02 2.335e-01 6.263e-02]
 ...
 [1.163e+01 2.929e+01 7.487e+01 ... 6.835e-02 2.884e-01 7.220e-02]
 [1.377e+01 1.327e+01 8.806e+01 ... 5.802e-02 2.823e-01 6.794e-02]
 [1.180e+01 1.658e+01 7.899e+01 ... 1.865e-01 5.774e-01 1.030e-01]]
학습 레이블[512]
[0 0 1 0 1 0 0 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 0 1 0 0
 1 0 1 1 1 0 1 0 1 0 1 0 1 1 1 1 0 1 1 0 0 1 1 0 1 1 1 0 0 1 0 0 1 1 1 0 1
 1 1 0 0 1 1 1 1 1 0 0 0 0 0 1 1 1 1 0 1 1 1 1 0 1 0 1 0 1 0 1 1 1 1 1 0 0
 1 0 1 1 1 0 1 0 1 1 1 1 0 1 0 1 0 0 1 1 1 0 1 1 0 1 1 0 0 1 1 0 0 0 0 0 0
 1 0 1 1 0 1 1 1 1 1 1 1 0 1 0 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 0
 0 1 1 0 0 1 0 0 1 0 1 1 1 1 0 1 1 0 1 0 1 0 0 0 1 1 1 1 1 0 0 0 0 0 1 1 1
 1 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 1 0 0 1 1 0 1 1 1 1 0 0 1 1
 0 1 0 1 1 1 0 1 0 1 1 1 1 1 0 1 1 0 0 0 0 0 1 0 0 1 1 0 1 1 1 1 0 0 0 0 1
 1 1 1 1 1 0 1 1 0 1 1 1 0 1 0 0 0 0 1 0 1 1 1 1 1 1 0 1 1 1 1 0 1 0 1 1 1
 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 0 0 1 1 1 1 1 0 1 0 1 1 1 0 1 1 0 0 1 1 0
 0 0 1 0 1 1 0 1 1 0 0 0 1 0 0 1 1 0 0 1 0 0 0 0 1 1 0 1 1 1 0 1 0 1 1 1 1
 1 0 1 1 1 0 1 0 1 1 1 1 1 1 0 1 1 1 0 0 1 1 1 1 0 1 1 0 1 1 1 1 1 1 0 0 1
 1 1 0 1 0 1 0 1 1 1 0 1 0 1 0 1 0 0 0 1 1 0 1 1 0 1 0 0 0 1 1 1 0 1 1 1 1
 0 0 0 1 1 1 0 1 1 1 0 0 1 1 0 0 1 0 0 1 1 0 0 0 1 0 1 1 1 1 0]
테스트 데이터[57]
[[9.173e+00 1.386e+01 5.920e+01 ... 5.087e-02 3.282e-01 8.490e-02]
 [1.230e+01 1.590e+01 7.883e+01 ... 4.815e-02 2.482e-01 6.306e-02]
 [1.546e+01 2.395e+01 1.038e+02 ... 2.163e-01 3.013e-01 1.067e-01]
 ...
 [1.578e+01 2.291e+01 1.057e+02 ... 2.034e-01 3.274e-01 1.252e-01]
 [1.267e+01 1.730e+01 8.125e+01 ... 5.602e-02 2.688e-01 6.888e-02]
 [2.029e+01 1.434e+01 1.351e+02 ... 1.625e-01 2.364e-01 7.678e-02]]
테스트 레이블[57]
[1 1 0 0 1 1 1 0 1 1 0 1 0 1 1 1 0 0 0 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 0
 0 0 1 0 0 1 0 1 0 0 1 1 1 0 1 0 1 0 1 0]
```

```
# K2020008 : Convert to tensor
import torch
import torch.nn as nn
import torch.nn.functional as F
```

```
import torch.nn.functional as F
from torch.utils.data import DataLoader, TensorDataset
from torch.autograd import Variable

# K2020008 : 자동 계산을 위해서 사용하는 변수는 torch.autograd에 있는 Variable 입니다.

# K2020008 : Variable은 아래와 같은 속성들을 갖고 있습니다.
    # .backward() 가 호출되면 미분이 시작되고 그 정보가 담기게 됩니다.

    # data
    # Tensor 형태의 데이터
    # grad
    # Data가 거쳐온 layer에 대한 미분 값
    # grad_fn
    # 미분 값을 계산한 함수에 대한 정보

    # 출처: https://dororongju.tistory.com/142 [웹 개발 메모장]

# K2020008 : 자동 계산을 위해 변수의 변환
x_train = Variable(torch.from_numpy(x_train).float())
y_train = Variable(torch.from_numpy(y_train).float())

x_test = Variable(torch.from_numpy(x_test).float())
y_test = Variable(torch.from_numpy(y_test).float())

print("="*100)
print("x_train.data:", x_train.data)
print("x_train.grad:", x_train.grad)
print("x_train.grad_fn:", x_train.grad_fn)
print("데이터 유형 :{0}".format(type(x_train)))
print("="*100)
```

```
==========================================================================================
x_train.data: tensor([[1.5100e+01, 2.2020e+01, 9.7260e+01,  ..., 1.5300e-01, 2.6750e-01,
         7.8730e-02],
        [1.5300e+01, 2.5270e+01, 1.0240e+02,  ..., 2.0240e-01, 4.0270e-01,
         9.8760e-02],
        [1.3590e+01, 1.7840e+01, 8.6240e+01,  ..., 5.1850e-02, 2.3350e-01,
         6.2630e-02],
        ...,
        [1.1630e+01, 2.9290e+01, 7.4870e+01,  ..., 6.8350e-02, 2.8840e-01,
         7.2200e-02],
        [1.3770e+01, 1.3270e+01, 8.8060e+01,  ..., 5.8020e-02, 2.8230e-01,
         6.7940e-02],
        [1.1800e+01, 1.6580e+01, 7.8990e+01,  ..., 1.8650e-01, 5.7740e-01,
         1.0300e-01]])
x_train.grad: None
x_train.grad_fn: None
데이터 유형 :<class 'torch.Tensor'>
==========================================================================================
```

```
# K2020008 : Generating dataset
    # 파이토치에서는 데이터를 좀 더 쉽게 다룰 수 있도록 유용한 도구로서 데이터셋(Dataset)과 데이터로더(DataLoader)를 제공합니다
    # 기본적인 사용 방법은 Dataset을 정의하고, 이를 DataLoader에 전달하는 것입니다.
    # TensorDataset은 기본적으로 텐서를 입력으로 받습니다. 텐서 형태로 데이터를 정의합니다(파라메터는 텐서형)

# K2020008 : TensorDataset의 입력으로 사용하고 train_set/test_set에 저장합니다
train_set = TensorDataset(x_train, y_train)
test_set = TensorDataset(x_test, y_test)

# K2020008 : DataLoader
    # 파이토치의 데이터셋을 만들었다면 데이터로더를 사용 가능합니다.
    # 데이터로더는 기본적으로 2개의 인자를 입력받는다. 하나는 데이터셋, 미니 배치의 크기입니다.
    # 이때 미니 배치의 크기는 8의 배수를 사용합니다. 그리고 추가적으로 많이 사용되는 인자로 shuffle이 있습니다.
    # shuffle=True를 선택하면 Epoch마다 데이터셋을 섞어서 데이터가 학습되는 순서를 바꿉니다.


train_loader = DataLoader(train_set, batch_size = 8, shuffle=True)
```

```
# K2020008 : 모델과 설계, Construct model
class Model(nn.Module):
  def __init__(self):
    super().__init__()
    # K2020008 :  입력층-은닉층-은닉층-출력층의 5층 구조
    self.layer1 = nn.Linear(30, 128)
    self.layer2 = nn.Linear(128, 64)
    self.layer3 = nn.Linear(64, 32)
    self.layer4 = nn.Linear(32, 16)
    self.layer5 = nn.Linear(16, 1)
    # self.layer6 = nn.Linear(16, 1)
    # K2020008 : 깊은 신경망은 ReLU 적용
    self.act = nn.ReLU()

  def forward(self,x):
    x = self.act(self.layer1(x))
    x = self.act(self.layer2(x))
    x = self.act(self.layer3(x))
    x = self.act(self.layer4(x))
    # x = self.act(self.layer5(x))
    x = self.layer5(x)
    # K2020008 : 로지스틱 시그모이드
    x = torch.sigmoid(x)

    return x

model = Model()
print(model)
```

```
Model(
  (layer1): Linear(in_features=30, out_features=128, bias=True)
  (layer2): Linear(in_features=128, out_features=64, bias=True)
  (layer3): Linear(in_features=64, out_features=32, bias=True)
  (layer4): Linear(in_features=32, out_features=16, bias=True)
  (layer5): Linear(in_features=16, out_features=1, bias=True)
  (act): ReLU()
)
```

```
# K2020008 : 옵티마이저 설계, Configure optimizer
# optimizer = torch.optim.SGD(model.parameters(), lr=0.001)
# optimizer = torch.optim.Adam(model.parameters(), lr=0.00001)
optimizer = torch.optim.Adam(model.parameters(), lr=0.00001)
# optimizer = torch.optim.Adam(model.parameters(), lr=0.0001, weight_decay=0.001)
```

```
# K2020008 : Training[훈련]

epochs = 8001
losses = list()
accuracies = list()

# K2020008 : 학습 진행 (8000회)
for epoch in range(epochs):
  epoch_loss = 0
  epoch_accuracy = 0

  # K2020008 : 8개씩 훈련
  for x, y in train_loader:
    # print(len(x))
    # print(len(y))
    optimizer.zero_grad()

    # K2020008 : 학습 모델 적용 -> H(x) 계산
    output = model(x)

    # K2020008 : cost 계산
    loss = F.binary_cross_entropy(output, y)

    # K2020008 : cost로 H(x) 개선
    # K2020008 : loss를 x로 미분
    # K2020008 : 경사하강법(Gradient descent 구현)
    # optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

```
      # K2020008 : 테이터의 정규화 (0, 1)
      # K2020008 : output>=0.5 -> 1, output < 0.5 -> 0

      output[output>=0.5] = 1
      output[output<0.5] = 0

      # K2020008 : 예측값이(output.data.T) 같으면 True
      accuracy = sum(sum(y.data.numpy() == output.data.T.numpy()))

      # K2020008 : Cost 계산 / accuracy 계산
      epoch_loss += loss.item()
      epoch_accuracy += accuracy


  # K2020008 : 학습이 잘 되는지 확인하기 위한 내용출력
  # K2020008 : 최종 (Cost 계산 / accuracy 계산)
  # K2020008 : loss은 0에 가깝고,  accuracy는 1에 근접해야 좋은 학습 결과임
    epoch_loss /= len(train_loader)
    epoch_accuracy /= len(x_train)
    if epoch % 10 == 0:
      print(str(epoch).zfill(3), "loss :", round(epoch_loss,4),"accuracy :", round(epoch_accuracy,4))

    losses.append(epoch_loss)
    accuracies.append(epoch_accuracy)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:21: UserWarning: Using a target size (torch.Size([8])) that is different to the
000 loss : 0.9297 accuracy : 0.6289
010 loss : 0.5638 accuracy : 0.8984
020 loss : 0.4548 accuracy : 0.9004
030 loss : 0.3892 accuracy : 0.9023
040 loss : 0.3422 accuracy : 0.9102
050 loss : 0.305 accuracy : 0.9141
060 loss : 0.2808 accuracy : 0.9082
070 loss : 0.2591 accuracy : 0.916
080 loss : 0.243 accuracy : 0.9199
090 loss : 0.2319 accuracy : 0.9141
100 loss : 0.2212 accuracy : 0.9238
110 loss : 0.213 accuracy : 0.9238
120 loss : 0.2088 accuracy : 0.9258
130 loss : 0.2058 accuracy : 0.9277
140 loss : 0.2022 accuracy : 0.9238
150 loss : 0.1984 accuracy : 0.9258
160 loss : 0.1964 accuracy : 0.9277
170 loss : 0.1942 accuracy : 0.9238
180 loss : 0.1916 accuracy : 0.9238
190 loss : 0.1908 accuracy : 0.9219
200 loss : 0.1865 accuracy : 0.9238
210 loss : 0.1866 accuracy : 0.9277
220 loss : 0.1867 accuracy : 0.9277
230 loss : 0.1829 accuracy : 0.9316
240 loss : 0.1844 accuracy : 0.9258
250 loss : 0.1831 accuracy : 0.9297
260 loss : 0.1805 accuracy : 0.9277
270 loss : 0.1795 accuracy : 0.9277
280 loss : 0.1777 accuracy : 0.9316
290 loss : 0.1779 accuracy : 0.9258
300 loss : 0.1785 accuracy : 0.9336
310 loss : 0.1761 accuracy : 0.9316
320 loss : 0.1755 accuracy : 0.9297
330 loss : 0.1743 accuracy : 0.9316
340 loss : 0.1767 accuracy : 0.9238
350 loss : 0.1731 accuracy : 0.9316
360 loss : 0.1719 accuracy : 0.9277
370 loss : 0.1739 accuracy : 0.9336
380 loss : 0.1704 accuracy : 0.9336
390 loss : 0.1694 accuracy : 0.9297
400 loss : 0.1696 accuracy : 0.9297
```

```
410 loss : 0.1695 accuracy : 0.9277
420 loss : 0.1705 accuracy : 0.9316
430 loss : 0.1691 accuracy : 0.9336
440 loss : 0.1672 accuracy : 0.9355
450 loss : 0.1672 accuracy : 0.9316
460 loss : 0.1647 accuracy : 0.9316
470 loss : 0.1657 accuracy : 0.9277
480 loss : 0.1732 accuracy : 0.9336
490 loss : 0.1652 accuracy : 0.9297
500 loss : 0.1622 accuracy : 0.9258
510 loss : 0.1634 accuracy : 0.9355
520 loss : 0.1627 accuracy : 0.9336
530 loss : 0.1623 accuracy : 0.9316
540 loss : 0.1663 accuracy : 0.9375
550 loss : 0.1605 accuracy : 0.9336
560 loss : 0.1621 accuracy : 0.9297
570 loss : 0.1593 accuracy : 0.9316
580 loss : 0.1616 accuracy : 0.9355
590 loss : 0.1596 accuracy : 0.9336
600 loss : 0.1591 accuracy : 0.9395
610 loss : 0.1604 accuracy : 0.9336
620 loss : 0.1568 accuracy : 0.9336
630 loss : 0.1558 accuracy : 0.9395
640 loss : 0.1561 accuracy : 0.9375
650 loss : 0.1559 accuracy : 0.9355
660 loss : 0.1575 accuracy : 0.9355
670 loss : 0.1539 accuracy : 0.9316
680 loss : 0.1537 accuracy : 0.9414
690 loss : 0.1551 accuracy : 0.9375
700 loss : 0.1542 accuracy : 0.9395
710 loss : 0.154 accuracy : 0.9375
720 loss : 0.1549 accuracy : 0.9355
730 loss : 0.1534 accuracy : 0.9395
740 loss : 0.1537 accuracy : 0.9375
750 loss : 0.1501 accuracy : 0.9453
760 loss : 0.1523 accuracy : 0.9355
770 loss : 0.1511 accuracy : 0.9395
780 loss : 0.1534 accuracy : 0.9395
790 loss : 0.1497 accuracy : 0.9375
800 loss : 0.1492 accuracy : 0.9375
810 loss : 0.15 accuracy : 0.9434
820 loss : 0.1512 accuracy : 0.9375
```

```
830 loss : 0.1523 accuracy : 0.9375
840 loss : 0.1471 accuracy : 0.9414
850 loss : 0.1472 accuracy : 0.9395
860 loss : 0.1475 accuracy : 0.9375
870 loss : 0.1462 accuracy : 0.9375
880 loss : 0.1455 accuracy : 0.9453
890 loss : 0.1458 accuracy : 0.9473
900 loss : 0.1484 accuracy : 0.9395
910 loss : 0.1469 accuracy : 0.9414
920 loss : 0.1507 accuracy : 0.9395
930 loss : 0.1464 accuracy : 0.9453
940 loss : 0.1442 accuracy : 0.9434
950 loss : 0.1438 accuracy : 0.9434
960 loss : 0.1456 accuracy : 0.9395
970 loss : 0.1434 accuracy : 0.9414
980 loss : 0.1453 accuracy : 0.9434
990 loss : 0.1425 accuracy : 0.9453
1000 loss : 0.1428 accuracy : 0.9434
1010 loss : 0.1416 accuracy : 0.9434
1020 loss : 0.1419 accuracy : 0.9375
1030 loss : 0.1433 accuracy : 0.9473
1040 loss : 0.1417 accuracy : 0.9453
1050 loss : 0.141 accuracy : 0.9414
1060 loss : 0.1403 accuracy : 0.9395
1070 loss : 0.1424 accuracy : 0.9414
1080 loss : 0.1388 accuracy : 0.9434
1090 loss : 0.1416 accuracy : 0.9434
1100 loss : 0.1385 accuracy : 0.9395
1110 loss : 0.1399 accuracy : 0.9375
1120 loss : 0.1368 accuracy : 0.9414
1130 loss : 0.1379 accuracy : 0.9473
1140 loss : 0.1431 accuracy : 0.9375
1150 loss : 0.1379 accuracy : 0.9434
1160 loss : 0.1387 accuracy : 0.9492
1170 loss : 0.1354 accuracy : 0.9512
1180 loss : 0.1365 accuracy : 0.9453
1190 loss : 0.1359 accuracy : 0.9453
1200 loss : 0.1347 accuracy : 0.9531
1210 loss : 0.1362 accuracy : 0.9453
1220 loss : 0.1352 accuracy : 0.9492
1230 loss : 0.1343 accuracy : 0.9492
1240 loss : 0.1362 accuracy : 0.9473
```

```
1250 loss : 0.1325 accuracy : 0.9492
1260 loss : 0.1376 accuracy : 0.9473
1270 loss : 0.1353 accuracy : 0.9492
1280 loss : 0.1387 accuracy : 0.9434
1290 loss : 0.1341 accuracy : 0.9453
1300 loss : 0.1336 accuracy : 0.9453
1310 loss : 0.1315 accuracy : 0.9473
1320 loss : 0.1332 accuracy : 0.9473
1330 loss : 0.1304 accuracy : 0.9512
1340 loss : 0.1312 accuracy : 0.9512
1350 loss : 0.1337 accuracy : 0.9414
1360 loss : 0.134 accuracy : 0.9453
1370 loss : 0.1302 accuracy : 0.9531
1380 loss : 0.1308 accuracy : 0.9492
1390 loss : 0.1324 accuracy : 0.9492
1400 loss : 0.13 accuracy : 0.9512
1410 loss : 0.1382 accuracy : 0.9434
1420 loss : 0.1298 accuracy : 0.9453
1430 loss : 0.1288 accuracy : 0.9551
1440 loss : 0.1345 accuracy : 0.9395
1450 loss : 0.1301 accuracy : 0.9512
1460 loss : 0.1285 accuracy : 0.9531
1470 loss : 0.1282 accuracy : 0.9512
1480 loss : 0.127 accuracy : 0.9551
1490 loss : 0.1269 accuracy : 0.9473
1500 loss : 0.1284 accuracy : 0.9492
1510 loss : 0.1293 accuracy : 0.9531
1520 loss : 0.1249 accuracy : 0.9512
1530 loss : 0.1262 accuracy : 0.9492
1540 loss : 0.1256 accuracy : 0.9492
1550 loss : 0.124 accuracy : 0.959
1560 loss : 0.1269 accuracy : 0.9512
1570 loss : 0.126 accuracy : 0.959
1580 loss : 0.1239 accuracy : 0.957
1590 loss : 0.1241 accuracy : 0.9492
1600 loss : 0.1246 accuracy : 0.9512
1610 loss : 0.1234 accuracy : 0.957
1620 loss : 0.1216 accuracy : 0.9492
1630 loss : 0.1214 accuracy : 0.957
1640 loss : 0.1241 accuracy : 0.9551
1650 loss : 0.1239 accuracy : 0.9512
1660 loss : 0.1218 accuracy : 0.957
```

```
1670 loss : 0.1257 accuracy : 0.959
1680 loss : 0.1222 accuracy : 0.959
1690 loss : 0.123 accuracy : 0.9531
1700 loss : 0.1232 accuracy : 0.9512
1710 loss : 0.1223 accuracy : 0.9512
1720 loss : 0.1233 accuracy : 0.9512
1730 loss : 0.1196 accuracy : 0.9531
1740 loss : 0.1233 accuracy : 0.9531
1750 loss : 0.1212 accuracy : 0.957
1760 loss : 0.1198 accuracy : 0.9531
1770 loss : 0.1177 accuracy : 0.9609
1780 loss : 0.1184 accuracy : 0.9492
1790 loss : 0.1194 accuracy : 0.957
1800 loss : 0.12 accuracy : 0.9492
1810 loss : 0.116 accuracy : 0.959
1820 loss : 0.1198 accuracy : 0.959
1830 loss : 0.1195 accuracy : 0.957
1840 loss : 0.1176 accuracy : 0.9551
1850 loss : 0.1211 accuracy : 0.957
1860 loss : 0.1193 accuracy : 0.9531
1870 loss : 0.1169 accuracy : 0.9512
1880 loss : 0.117 accuracy : 0.9531
1890 loss : 0.1169 accuracy : 0.9531
1900 loss : 0.1152 accuracy : 0.9512
1910 loss : 0.1152 accuracy : 0.9551
1920 loss : 0.1187 accuracy : 0.9609
1930 loss : 0.1142 accuracy : 0.9551
1940 loss : 0.1166 accuracy : 0.9531
1950 loss : 0.1125 accuracy : 0.957
1960 loss : 0.1152 accuracy : 0.9551
1970 loss : 0.114 accuracy : 0.9551
1980 loss : 0.1147 accuracy : 0.9512
1990 loss : 0.1153 accuracy : 0.957
2000 loss : 0.1121 accuracy : 0.9551
2010 loss : 0.1124 accuracy : 0.9609
2020 loss : 0.1121 accuracy : 0.959
2030 loss : 0.1145 accuracy : 0.9551
2040 loss : 0.1136 accuracy : 0.9551
2050 loss : 0.1174 accuracy : 0.9492
2060 loss : 0.1108 accuracy : 0.957
2070 loss : 0.1095 accuracy : 0.957
2080 loss : 0.1098 accuracy : 0.959
```

```
2090 loss : 0.1101 accuracy : 0.959
2100 loss : 0.1087 accuracy : 0.959
2110 loss : 0.1093 accuracy : 0.9531
2120 loss : 0.1085 accuracy : 0.959
2130 loss : 0.1098 accuracy : 0.957
2140 loss : 0.1088 accuracy : 0.959
2150 loss : 0.1124 accuracy : 0.9512
2160 loss : 0.1131 accuracy : 0.9492
2170 loss : 0.1075 accuracy : 0.9551
2180 loss : 0.1075 accuracy : 0.959
2190 loss : 0.1122 accuracy : 0.9531
2200 loss : 0.1067 accuracy : 0.957
2210 loss : 0.1068 accuracy : 0.9551
2220 loss : 0.1054 accuracy : 0.9609
2230 loss : 0.1067 accuracy : 0.959
2240 loss : 0.1049 accuracy : 0.959
2250 loss : 0.1074 accuracy : 0.9629
2260 loss : 0.107 accuracy : 0.9551
2270 loss : 0.1071 accuracy : 0.959
2280 loss : 0.1039 accuracy : 0.957
2290 loss : 0.1049 accuracy : 0.959
2300 loss : 0.1034 accuracy : 0.957
2310 loss : 0.1026 accuracy : 0.9551
2320 loss : 0.1049 accuracy : 0.9551
2330 loss : 0.103 accuracy : 0.9551
2340 loss : 0.1031 accuracy : 0.9609
2350 loss : 0.1031 accuracy : 0.9551
2360 loss : 0.1016 accuracy : 0.9629
2370 loss : 0.1016 accuracy : 0.959
2380 loss : 0.1037 accuracy : 0.957
2390 loss : 0.1015 accuracy : 0.9609
2400 loss : 0.1016 accuracy : 0.957
2410 loss : 0.1025 accuracy : 0.957
2420 loss : 0.1029 accuracy : 0.9629
2430 loss : 0.1031 accuracy : 0.9609
2440 loss : 0.1014 accuracy : 0.9609
2450 loss : 0.1031 accuracy : 0.9551
2460 loss : 0.0984 accuracy : 0.959
2470 loss : 0.0994 accuracy : 0.957
2480 loss : 0.1007 accuracy : 0.9609
2490 loss : 0.0989 accuracy : 0.959
2500 loss : 0.0989 accuracy : 0.9551
```

```
2510 loss : 0.1007 accuracy : 0.9531
2520 loss : 0.0977 accuracy : 0.959
2530 loss : 0.1033 accuracy : 0.959
2540 loss : 0.0969 accuracy : 0.9629
2550 loss : 0.0973 accuracy : 0.9629
2560 loss : 0.0976 accuracy : 0.959
2570 loss : 0.0964 accuracy : 0.9609
2580 loss : 0.0957 accuracy : 0.9629
2590 loss : 0.0961 accuracy : 0.9629
2600 loss : 0.0947 accuracy : 0.9688
2610 loss : 0.0937 accuracy : 0.9629
2620 loss : 0.099 accuracy : 0.959
2630 loss : 0.0924 accuracy : 0.9668
2640 loss : 0.0995 accuracy : 0.9551
2650 loss : 0.0941 accuracy : 0.9648
2660 loss : 0.0957 accuracy : 0.9629
2670 loss : 0.0953 accuracy : 0.9551
2680 loss : 0.0917 accuracy : 0.9609
2690 loss : 0.1025 accuracy : 0.957
2700 loss : 0.0954 accuracy : 0.9609
2710 loss : 0.0912 accuracy : 0.9668
2720 loss : 0.0942 accuracy : 0.959
2730 loss : 0.0911 accuracy : 0.9707
2740 loss : 0.0907 accuracy : 0.9609
2750 loss : 0.0937 accuracy : 0.959
2760 loss : 0.0926 accuracy : 0.9648
2770 loss : 0.0915 accuracy : 0.9629
2780 loss : 0.0919 accuracy : 0.9609
2790 loss : 0.09 accuracy : 0.9648
2800 loss : 0.0925 accuracy : 0.9648
2810 loss : 0.0912 accuracy : 0.9629
2820 loss : 0.089 accuracy : 0.959
2830 loss : 0.0944 accuracy : 0.9688
2840 loss : 0.0884 accuracy : 0.9668
2850 loss : 0.0933 accuracy : 0.9551
2860 loss : 0.0925 accuracy : 0.9629
2870 loss : 0.0904 accuracy : 0.9668
2880 loss : 0.092 accuracy : 0.9668
2890 loss : 0.091 accuracy : 0.9668
2900 loss : 0.0887 accuracy : 0.9609
2910 loss : 0.0897 accuracy : 0.9648
2920 loss : 0.0885 accuracy : 0.9668
2930 loss : 0.0868 accuracy : 0.9688
```

```
2930 loss : 0.0868 accuracy : 0.9688
2940 loss : 0.0866 accuracy : 0.9707
2950 loss : 0.0874 accuracy : 0.9629
2960 loss : 0.0869 accuracy : 0.9648
2970 loss : 0.0875 accuracy : 0.9629
2980 loss : 0.0887 accuracy : 0.9648
2990 loss : 0.0906 accuracy : 0.9648
3000 loss : 0.0865 accuracy : 0.9609
3010 loss : 0.0869 accuracy : 0.9629
3020 loss : 0.0848 accuracy : 0.9688
3030 loss : 0.0849 accuracy : 0.9668
3040 loss : 0.0864 accuracy : 0.9629
3050 loss : 0.0856 accuracy : 0.9688
3060 loss : 0.0835 accuracy : 0.9668
3070 loss : 0.085 accuracy : 0.9668
3080 loss : 0.0846 accuracy : 0.9648
3090 loss : 0.0831 accuracy : 0.9668
3100 loss : 0.0822 accuracy : 0.9668
3110 loss : 0.0849 accuracy : 0.9668
3120 loss : 0.083 accuracy : 0.9688
3130 loss : 0.082 accuracy : 0.9688
3140 loss : 0.0833 accuracy : 0.9668
3150 loss : 0.0824 accuracy : 0.9688
3160 loss : 0.0846 accuracy : 0.9648
3170 loss : 0.081 accuracy : 0.9746
3180 loss : 0.0912 accuracy : 0.957
3190 loss : 0.0807 accuracy : 0.9629
3200 loss : 0.0906 accuracy : 0.9609
3210 loss : 0.0815 accuracy : 0.9707
3220 loss : 0.0797 accuracy : 0.9668
3230 loss : 0.0826 accuracy : 0.9629
3240 loss : 0.0816 accuracy : 0.9668
3250 loss : 0.0801 accuracy : 0.9688
3260 loss : 0.0843 accuracy : 0.9707
3270 loss : 0.0815 accuracy : 0.9648
3280 loss : 0.0814 accuracy : 0.9707
3290 loss : 0.0781 accuracy : 0.9668
3300 loss : 0.0801 accuracy : 0.9688
3310 loss : 0.0803 accuracy : 0.9688
3320 loss : 0.0804 accuracy : 0.9727
3330 loss : 0.084 accuracy : 0.9629
3340 loss : 0.0771 accuracy : 0.9746
3350 loss : 0.0791 accuracy : 0.9668
```

```
3350 loss : 0.0791 accuracy : 0.9688
3360 loss : 0.0767 accuracy : 0.9688
3370 loss : 0.0776 accuracy : 0.9648
3380 loss : 0.0774 accuracy : 0.9727
3390 loss : 0.0793 accuracy : 0.9766
3400 loss : 0.0752 accuracy : 0.9648
3410 loss : 0.0751 accuracy : 0.9688
3420 loss : 0.0758 accuracy : 0.9746
3430 loss : 0.0756 accuracy : 0.9688
3440 loss : 0.0752 accuracy : 0.9688
3450 loss : 0.0756 accuracy : 0.9707
3460 loss : 0.076 accuracy : 0.9746
3470 loss : 0.0792 accuracy : 0.9707
3480 loss : 0.0758 accuracy : 0.9727
3490 loss : 0.0778 accuracy : 0.9766
3500 loss : 0.0786 accuracy : 0.9688
3510 loss : 0.0772 accuracy : 0.9766
3520 loss : 0.0747 accuracy : 0.9707
3530 loss : 0.0726 accuracy : 0.9707
3540 loss : 0.078 accuracy : 0.9707
3550 loss : 0.0752 accuracy : 0.9688
3560 loss : 0.0727 accuracy : 0.9727
3570 loss : 0.0721 accuracy : 0.9707
3580 loss : 0.0785 accuracy : 0.9707
3590 loss : 0.0716 accuracy : 0.9766
3600 loss : 0.0727 accuracy : 0.9707
3610 loss : 0.078 accuracy : 0.9707
3620 loss : 0.0721 accuracy : 0.9707
3630 loss : 0.0713 accuracy : 0.9688
3640 loss : 0.0717 accuracy : 0.9727
3650 loss : 0.0733 accuracy : 0.9785
3660 loss : 0.0733 accuracy : 0.9707
3670 loss : 0.0723 accuracy : 0.9727
3680 loss : 0.0716 accuracy : 0.9727
3690 loss : 0.0699 accuracy : 0.9746
3700 loss : 0.0729 accuracy : 0.9707
3710 loss : 0.0732 accuracy : 0.9688
3720 loss : 0.0734 accuracy : 0.9668
3730 loss : 0.0714 accuracy : 0.9688
3740 loss : 0.0714 accuracy : 0.9707
3750 loss : 0.0692 accuracy : 0.9746
3760 loss : 0.0717 accuracy : 0.9688
3770 loss : 0.0673 accuracy : 0.9766
```

```
3780 loss : 0.0696 accuracy : 0.9766
3790 loss : 0.0737 accuracy : 0.9727
3800 loss : 0.0681 accuracy : 0.9688
3810 loss : 0.0691 accuracy : 0.9727
3820 loss : 0.0742 accuracy : 0.9688
3830 loss : 0.0695 accuracy : 0.9727
3840 loss : 0.0765 accuracy : 0.9707
3850 loss : 0.0686 accuracy : 0.9766
3860 loss : 0.0791 accuracy : 0.9727
3870 loss : 0.068 accuracy : 0.9707
3880 loss : 0.0699 accuracy : 0.9727
3890 loss : 0.0705 accuracy : 0.9746
3900 loss : 0.068 accuracy : 0.9707
3910 loss : 0.0682 accuracy : 0.9707
3920 loss : 0.0668 accuracy : 0.9727
3930 loss : 0.0685 accuracy : 0.9707
3940 loss : 0.067 accuracy : 0.9746
3950 loss : 0.0677 accuracy : 0.9707
3960 loss : 0.0687 accuracy : 0.9668
3970 loss : 0.0686 accuracy : 0.9766
3980 loss : 0.0677 accuracy : 0.9766
3990 loss : 0.0696 accuracy : 0.9766
4000 loss : 0.0665 accuracy : 0.9727
4010 loss : 0.0655 accuracy : 0.9746
4020 loss : 0.0672 accuracy : 0.9727
4030 loss : 0.0694 accuracy : 0.9727
4040 loss : 0.0696 accuracy : 0.9727
4050 loss : 0.0686 accuracy : 0.9746
4060 loss : 0.0654 accuracy : 0.9785
4070 loss : 0.0667 accuracy : 0.9727
4080 loss : 0.0649 accuracy : 0.9766
4090 loss : 0.0675 accuracy : 0.9727
4100 loss : 0.0679 accuracy : 0.9746
4110 loss : 0.0706 accuracy : 0.9766
4120 loss : 0.0673 accuracy : 0.9727
4130 loss : 0.065 accuracy : 0.9766
4140 loss : 0.0675 accuracy : 0.9766
4150 loss : 0.0657 accuracy : 0.9805
4160 loss : 0.0664 accuracy : 0.9766
4170 loss : 0.065 accuracy : 0.9727
4180 loss : 0.0642 accuracy : 0.9785
4190 loss : 0.0651 accuracy : 0.9727
```

```
4200 loss : 0.0653 accuracy : 0.9766
4210 loss : 0.0646 accuracy : 0.9707
4220 loss : 0.0648 accuracy : 0.9746
4230 loss : 0.0657 accuracy : 0.9746
4240 loss : 0.0693 accuracy : 0.9785
4250 loss : 0.0647 accuracy : 0.9727
4260 loss : 0.0652 accuracy : 0.9766
4270 loss : 0.0668 accuracy : 0.9707
4280 loss : 0.0676 accuracy : 0.9746
4290 loss : 0.0655 accuracy : 0.9746
4300 loss : 0.0639 accuracy : 0.9746
4310 loss : 0.0714 accuracy : 0.9727
4320 loss : 0.0656 accuracy : 0.9727
4330 loss : 0.0647 accuracy : 0.9844
4340 loss : 0.0683 accuracy : 0.9785
4350 loss : 0.0633 accuracy : 0.9766
4360 loss : 0.0627 accuracy : 0.9727
4370 loss : 0.0662 accuracy : 0.9746
4380 loss : 0.0646 accuracy : 0.9766
4390 loss : 0.0628 accuracy : 0.9746
4400 loss : 0.0617 accuracy : 0.9785
4410 loss : 0.0673 accuracy : 0.9688
4420 loss : 0.0639 accuracy : 0.9824
4430 loss : 0.0626 accuracy : 0.9746
4440 loss : 0.0603 accuracy : 0.9766
4450 loss : 0.062 accuracy : 0.9766
4460 loss : 0.0641 accuracy : 0.9785
4470 loss : 0.0636 accuracy : 0.9766
4480 loss : 0.0621 accuracy : 0.9746
4490 loss : 0.0613 accuracy : 0.9746
4500 loss : 0.0622 accuracy : 0.9766
4510 loss : 0.0646 accuracy : 0.9785
4520 loss : 0.0647 accuracy : 0.9746
4530 loss : 0.064 accuracy : 0.9727
4540 loss : 0.0617 accuracy : 0.9766
4550 loss : 0.0615 accuracy : 0.9785
4560 loss : 0.0615 accuracy : 0.9785
4570 loss : 0.0622 accuracy : 0.9766
4580 loss : 0.0612 accuracy : 0.9766
4590 loss : 0.067 accuracy : 0.9727
4600 loss : 0.0602 accuracy : 0.9785
4610 loss : 0.0616 accuracy : 0.9746
```

```
4620 loss : 0.0624 accuracy : 0.9727
4630 loss : 0.0594 accuracy : 0.9766
4640 loss : 0.0657 accuracy : 0.9727
4650 loss : 0.0629 accuracy : 0.9785
4660 loss : 0.0666 accuracy : 0.9805
4670 loss : 0.0626 accuracy : 0.9766
4680 loss : 0.0618 accuracy : 0.9766
4690 loss : 0.0635 accuracy : 0.9727
4700 loss : 0.0614 accuracy : 0.9785
4710 loss : 0.0579 accuracy : 0.9785
4720 loss : 0.0607 accuracy : 0.9727
4730 loss : 0.0601 accuracy : 0.9766
4740 loss : 0.062 accuracy : 0.9766
4750 loss : 0.058 accuracy : 0.9785
4760 loss : 0.0617 accuracy : 0.9746
4770 loss : 0.0591 accuracy : 0.9805
4780 loss : 0.0646 accuracy : 0.9785
4790 loss : 0.0604 accuracy : 0.9785
4800 loss : 0.0596 accuracy : 0.9727
4810 loss : 0.0588 accuracy : 0.9844
4820 loss : 0.0626 accuracy : 0.9766
4830 loss : 0.062 accuracy : 0.9766
4840 loss : 0.0634 accuracy : 0.9746
4850 loss : 0.0583 accuracy : 0.9766
4860 loss : 0.0595 accuracy : 0.9824
4870 loss : 0.0572 accuracy : 0.9785
4880 loss : 0.0603 accuracy : 0.9746
4890 loss : 0.0569 accuracy : 0.9805
4900 loss : 0.0596 accuracy : 0.9766
4910 loss : 0.0594 accuracy : 0.9805
4920 loss : 0.058 accuracy : 0.9844
4930 loss : 0.0635 accuracy : 0.9727
4940 loss : 0.059 accuracy : 0.9746
4950 loss : 0.0563 accuracy : 0.9785
4960 loss : 0.0646 accuracy : 0.9766
4970 loss : 0.0624 accuracy : 0.9746
4980 loss : 0.0606 accuracy : 0.9805
4990 loss : 0.0586 accuracy : 0.9785
5000 loss : 0.0577 accuracy : 0.9785
5010 loss : 0.0622 accuracy : 0.9746
5020 loss : 0.0565 accuracy : 0.9785
5030 loss : 0.0578 accuracy : 0.9785
```

```
5040 loss : 0.0567 accuracy : 0.9766
5050 loss : 0.0544 accuracy : 0.9805
5060 loss : 0.0594 accuracy : 0.9766
5070 loss : 0.0609 accuracy : 0.9785
5080 loss : 0.0566 accuracy : 0.9766
5090 loss : 0.0571 accuracy : 0.9785
5100 loss : 0.0513 accuracy : 0.9844
5110 loss : 0.0557 accuracy : 0.9766
5120 loss : 0.0528 accuracy : 0.9824
5130 loss : 0.0595 accuracy : 0.9805
5140 loss : 0.0549 accuracy : 0.9785
5150 loss : 0.0549 accuracy : 0.9766
5160 loss : 0.0545 accuracy : 0.9766
5170 loss : 0.0568 accuracy : 0.9805
5180 loss : 0.0553 accuracy : 0.9824
5190 loss : 0.0565 accuracy : 0.9805
5200 loss : 0.0556 accuracy : 0.9785
5210 loss : 0.055 accuracy : 0.9805
5220 loss : 0.0596 accuracy : 0.9727
5230 loss : 0.0536 accuracy : 0.9746
5240 loss : 0.0643 accuracy : 0.9746
5250 loss : 0.0544 accuracy : 0.9785
5260 loss : 0.0554 accuracy : 0.9805
5270 loss : 0.0551 accuracy : 0.9785
5280 loss : 0.055 accuracy : 0.9805
5290 loss : 0.0555 accuracy : 0.9844
5300 loss : 0.0564 accuracy : 0.9785
5310 loss : 0.0573 accuracy : 0.9824
5320 loss : 0.0544 accuracy : 0.9824
5330 loss : 0.0535 accuracy : 0.9805
5340 loss : 0.0576 accuracy : 0.9805
5350 loss : 0.0593 accuracy : 0.9805
5360 loss : 0.0556 accuracy : 0.9805
5370 loss : 0.0544 accuracy : 0.9805
5380 loss : 0.0546 accuracy : 0.9805
5390 loss : 0.057 accuracy : 0.9785
5400 loss : 0.0567 accuracy : 0.9863
5410 loss : 0.0582 accuracy : 0.9766
5420 loss : 0.054 accuracy : 0.9785
5430 loss : 0.0545 accuracy : 0.9805
5440 loss : 0.052 accuracy : 0.9766
5450 loss : 0.0547 accuracy : 0.9824
```

```
5460 loss : 0.0538 accuracy : 0.9824
5470 loss : 0.0586 accuracy : 0.9785
5480 loss : 0.0523 accuracy : 0.9824
5490 loss : 0.0551 accuracy : 0.9805
5500 loss : 0.0566 accuracy : 0.9746
5510 loss : 0.0552 accuracy : 0.9766
5520 loss : 0.0525 accuracy : 0.9746
5530 loss : 0.0521 accuracy : 0.9824
5540 loss : 0.0579 accuracy : 0.9785
5550 loss : 0.0539 accuracy : 0.9805
5560 loss : 0.0539 accuracy : 0.9805
5570 loss : 0.0541 accuracy : 0.9805
5580 loss : 0.052 accuracy : 0.9805
5590 loss : 0.0535 accuracy : 0.9824
5600 loss : 0.0541 accuracy : 0.9805
5610 loss : 0.0544 accuracy : 0.9863
5620 loss : 0.0523 accuracy : 0.9844
5630 loss : 0.0532 accuracy : 0.9805
5640 loss : 0.0522 accuracy : 0.9805
5650 loss : 0.0525 accuracy : 0.9844
5660 loss : 0.0517 accuracy : 0.9785
5670 loss : 0.0513 accuracy : 0.9785
5680 loss : 0.0528 accuracy : 0.9844
5690 loss : 0.0512 accuracy : 0.9805
5700 loss : 0.0522 accuracy : 0.9824
5710 loss : 0.0521 accuracy : 0.9805
5720 loss : 0.0499 accuracy : 0.9824
5730 loss : 0.0525 accuracy : 0.9824
5740 loss : 0.0543 accuracy : 0.9805
5750 loss : 0.0514 accuracy : 0.9863
5760 loss : 0.0501 accuracy : 0.9824
5770 loss : 0.0525 accuracy : 0.9824
5780 loss : 0.0503 accuracy : 0.9863
5790 loss : 0.0552 accuracy : 0.9766
5800 loss : 0.0531 accuracy : 0.9785
5810 loss : 0.0528 accuracy : 0.9824
5820 loss : 0.0533 accuracy : 0.9785
5830 loss : 0.0499 accuracy : 0.9844
5840 loss : 0.0507 accuracy : 0.9824
5850 loss : 0.0526 accuracy : 0.9766
5860 loss : 0.0546 accuracy : 0.9805
5870 loss : 0.0504 accuracy : 0.9863
```

```
5880 loss : 0.0499 accuracy : 0.9805
5890 loss : 0.0512 accuracy : 0.9785
5900 loss : 0.0498 accuracy : 0.9824
5910 loss : 0.0481 accuracy : 0.9844
5920 loss : 0.0521 accuracy : 0.9844
5930 loss : 0.0524 accuracy : 0.9824
5940 loss : 0.0512 accuracy : 0.9785
5950 loss : 0.0542 accuracy : 0.9805
5960 loss : 0.0538 accuracy : 0.9824
5970 loss : 0.0502 accuracy : 0.9863
5980 loss : 0.0494 accuracy : 0.9805
5990 loss : 0.0514 accuracy : 0.9766
6000 loss : 0.048 accuracy : 0.9863
6010 loss : 0.0494 accuracy : 0.9805
6020 loss : 0.0488 accuracy : 0.9863
6030 loss : 0.049 accuracy : 0.9805
6040 loss : 0.0507 accuracy : 0.9805
6050 loss : 0.0543 accuracy : 0.9805
6060 loss : 0.0508 accuracy : 0.9824
6070 loss : 0.0481 accuracy : 0.9844
6080 loss : 0.0497 accuracy : 0.9805
6090 loss : 0.0479 accuracy : 0.9844
6100 loss : 0.051 accuracy : 0.9785
6110 loss : 0.0493 accuracy : 0.9824
6120 loss : 0.0477 accuracy : 0.9785
6130 loss : 0.0485 accuracy : 0.9844
6140 loss : 0.0446 accuracy : 0.9883
6150 loss : 0.0503 accuracy : 0.9844
6160 loss : 0.0471 accuracy : 0.9863
6170 loss : 0.0468 accuracy : 0.9805
6180 loss : 0.0481 accuracy : 0.9844
6190 loss : 0.0485 accuracy : 0.9844
6200 loss : 0.0458 accuracy : 0.9883
6210 loss : 0.0452 accuracy : 0.9824
6220 loss : 0.046 accuracy : 0.9824
6230 loss : 0.0477 accuracy : 0.9844
6240 loss : 0.0486 accuracy : 0.9805
6250 loss : 0.0468 accuracy : 0.9844
6260 loss : 0.0497 accuracy : 0.9805
6270 loss : 0.0545 accuracy : 0.9785
6280 loss : 0.0493 accuracy : 0.9863
6290 loss : 0.0496 accuracy : 0.9844
6300 loss : 0.05 accuracy : 0.9785
```

```
6300 loss : 0.05 accuracy : 0.9785
6310 loss : 0.0501 accuracy : 0.9863
6320 loss : 0.0494 accuracy : 0.9863
6330 loss : 0.0487 accuracy : 0.9883
6340 loss : 0.0459 accuracy : 0.9824
6350 loss : 0.0468 accuracy : 0.9824
6360 loss : 0.0457 accuracy : 0.9863
6370 loss : 0.0468 accuracy : 0.9844
6380 loss : 0.0518 accuracy : 0.9766
6390 loss : 0.0474 accuracy : 0.9824
6400 loss : 0.0474 accuracy : 0.9824
6410 loss : 0.049 accuracy : 0.9805
6420 loss : 0.0454 accuracy : 0.9805
6430 loss : 0.0453 accuracy : 0.9844
6440 loss : 0.0455 accuracy : 0.9844
6450 loss : 0.0473 accuracy : 0.9863
6460 loss : 0.0459 accuracy : 0.9863
6470 loss : 0.0465 accuracy : 0.9824
6480 loss : 0.0458 accuracy : 0.9844
6490 loss : 0.0477 accuracy : 0.9824
6500 loss : 0.0503 accuracy : 0.9785
6510 loss : 0.0451 accuracy : 0.9844
6520 loss : 0.0469 accuracy : 0.9883
6530 loss : 0.0509 accuracy : 0.9805
6540 loss : 0.0471 accuracy : 0.9805
6550 loss : 0.0478 accuracy : 0.9863
6560 loss : 0.0457 accuracy : 0.9844
6570 loss : 0.0447 accuracy : 0.9844
6580 loss : 0.0453 accuracy : 0.9863
6590 loss : 0.0465 accuracy : 0.9844
6600 loss : 0.0487 accuracy : 0.9844
6610 loss : 0.0447 accuracy : 0.9844
6620 loss : 0.042 accuracy : 0.9883
6630 loss : 0.0468 accuracy : 0.9844
6640 loss : 0.0418 accuracy : 0.9824
6650 loss : 0.0453 accuracy : 0.9824
6660 loss : 0.041 accuracy : 0.9863
6670 loss : 0.0461 accuracy : 0.9844
6680 loss : 0.0434 accuracy : 0.9883
6690 loss : 0.0508 accuracy : 0.9824
6700 loss : 0.0479 accuracy : 0.9785
6710 loss : 0.0446 accuracy : 0.9824
6720 loss : 0.0428 accuracy : 0.9805
```

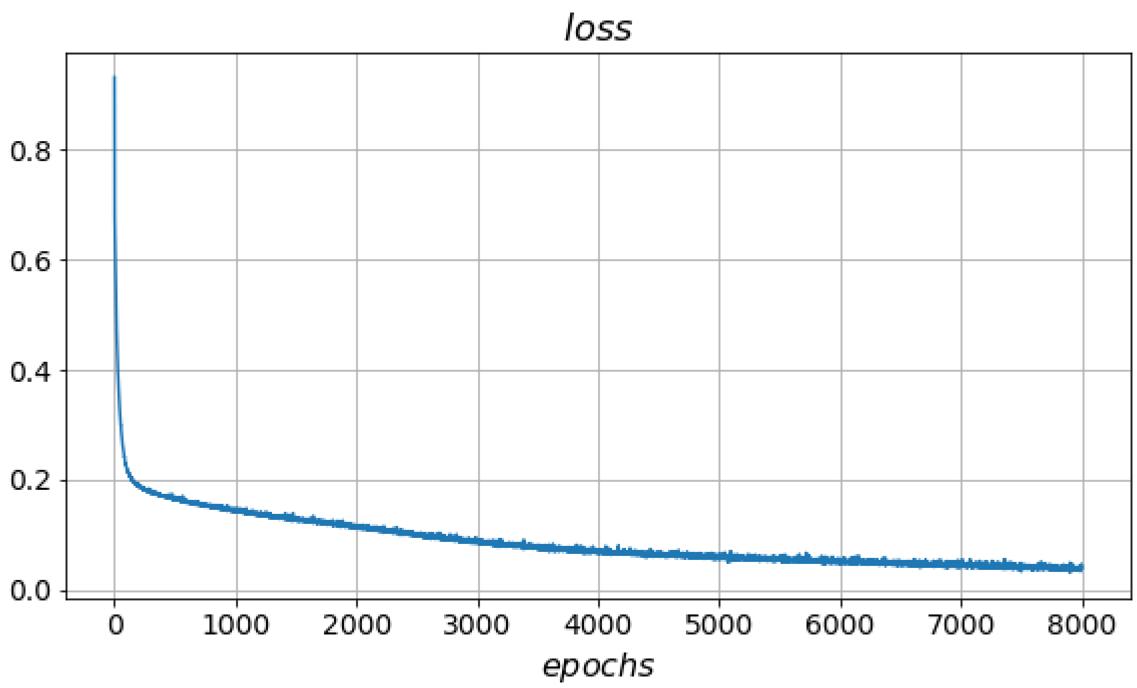
```
6720 loss : 0.0428 accuracy : 0.9805
6730 loss : 0.0441 accuracy : 0.9863
6740 loss : 0.0468 accuracy : 0.9824
6750 loss : 0.0446 accuracy : 0.9844
6760 loss : 0.0445 accuracy : 0.9844
6770 loss : 0.0463 accuracy : 0.9785
6780 loss : 0.0424 accuracy : 0.9844
6790 loss : 0.0429 accuracy : 0.9863
6800 loss : 0.0437 accuracy : 0.9824
6810 loss : 0.043 accuracy : 0.9824
6820 loss : 0.0478 accuracy : 0.9805
6830 loss : 0.0538 accuracy : 0.9824
6840 loss : 0.0425 accuracy : 0.9863
6850 loss : 0.0427 accuracy : 0.9844
6860 loss : 0.042 accuracy : 0.9863
6870 loss : 0.0455 accuracy : 0.9824
6880 loss : 0.047 accuracy : 0.9824
6890 loss : 0.0428 accuracy : 0.9863
6900 loss : 0.0454 accuracy : 0.9766
6910 loss : 0.0504 accuracy : 0.9844
6920 loss : 0.0424 accuracy : 0.9785
6930 loss : 0.0406 accuracy : 0.9844
6940 loss : 0.0475 accuracy : 0.9805
6950 loss : 0.0423 accuracy : 0.9824
6960 loss : 0.0462 accuracy : 0.9824
6970 loss : 0.0524 accuracy : 0.9824
6980 loss : 0.0413 accuracy : 0.9883
6990 loss : 0.0419 accuracy : 0.9863
7000 loss : 0.0472 accuracy : 0.9805
7010 loss : 0.0431 accuracy : 0.9863
7020 loss : 0.041 accuracy : 0.9844
7030 loss : 0.0417 accuracy : 0.9863
7040 loss : 0.0422 accuracy : 0.9863
7050 loss : 0.0432 accuracy : 0.9844
7060 loss : 0.0425 accuracy : 0.9844
7070 loss : 0.0457 accuracy : 0.9863
7080 loss : 0.0447 accuracy : 0.9863
7090 loss : 0.0457 accuracy : 0.9844
7100 loss : 0.0457 accuracy : 0.9824
7110 loss : 0.0445 accuracy : 0.9824
7120 loss : 0.0421 accuracy : 0.9824
7130 loss : 0.0484 accuracy : 0.9824
7140 loss : 0.0443 accuracy : 0.9844
```

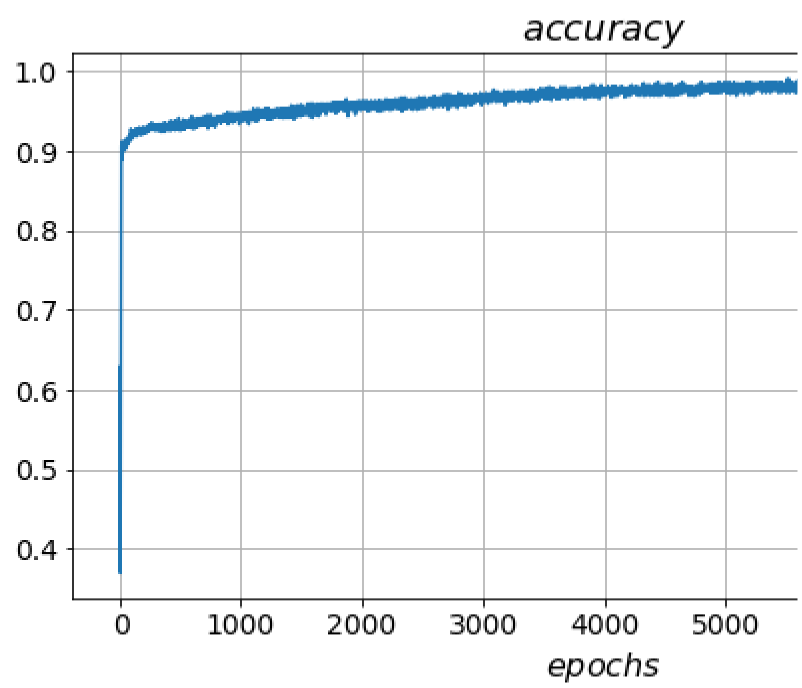
```
7150 loss : 0.0416 accuracy : 0.9824
7160 loss : 0.041 accuracy : 0.9824
7170 loss : 0.0444 accuracy : 0.9805
7180 loss : 0.0405 accuracy : 0.9883
7190 loss : 0.0435 accuracy : 0.9824
7200 loss : 0.0411 accuracy : 0.9883
7210 loss : 0.0396 accuracy : 0.9863
7220 loss : 0.0422 accuracy : 0.9863
7230 loss : 0.0401 accuracy : 0.9863
7240 loss : 0.0409 accuracy : 0.9844
7250 loss : 0.0435 accuracy : 0.9844
7260 loss : 0.0518 accuracy : 0.9766
7270 loss : 0.0391 accuracy : 0.9863
7280 loss : 0.0396 accuracy : 0.9863
7290 loss : 0.0416 accuracy : 0.9863
7300 loss : 0.0428 accuracy : 0.9844
7310 loss : 0.0489 accuracy : 0.9785
7320 loss : 0.0435 accuracy : 0.9883
7330 loss : 0.0446 accuracy : 0.9824
7340 loss : 0.0459 accuracy : 0.9844
7350 loss : 0.0419 accuracy : 0.9824
7360 loss : 0.036 accuracy : 0.9844
7370 loss : 0.0404 accuracy : 0.9844
7380 loss : 0.0487 accuracy : 0.9863
7390 loss : 0.0385 accuracy : 0.9883
7400 loss : 0.0374 accuracy : 0.9922
7410 loss : 0.0501 accuracy : 0.9766
7420 loss : 0.0426 accuracy : 0.9883
7430 loss : 0.0394 accuracy : 0.9844
7440 loss : 0.0403 accuracy : 0.9883
7450 loss : 0.0395 accuracy : 0.9863
7460 loss : 0.0391 accuracy : 0.9902
7470 loss : 0.0406 accuracy : 0.9863
7480 loss : 0.0442 accuracy : 0.9805
7490 loss : 0.0399 accuracy : 0.9863
7500 loss : 0.0428 accuracy : 0.9844
7510 loss : 0.0399 accuracy : 0.9805
7520 loss : 0.0401 accuracy : 0.9883
7530 loss : 0.0377 accuracy : 0.9863
7540 loss : 0.0382 accuracy : 0.9883
7550 loss : 0.0371 accuracy : 0.9863
7560 loss : 0.0417 accuracy : 0.9844
```

```
7570 loss : 0.0404 accuracy : 0.9844
7580 loss : 0.038 accuracy : 0.9863
7590 loss : 0.0373 accuracy : 0.9844
7600 loss : 0.0393 accuracy : 0.9863
7610 loss : 0.0413 accuracy : 0.9863
7620 loss : 0.0395 accuracy : 0.9844
7630 loss : 0.0351 accuracy : 0.9883
7640 loss : 0.04 accuracy : 0.9844
7650 loss : 0.0372 accuracy : 0.9883
7660 loss : 0.0382 accuracy : 0.9863
7670 loss : 0.0413 accuracy : 0.9863
7680 loss : 0.0372 accuracy : 0.9883
7690 loss : 0.0393 accuracy : 0.9883
7700 loss : 0.0391 accuracy : 0.9863
7710 loss : 0.0398 accuracy : 0.9844
7720 loss : 0.0413 accuracy : 0.9844
7730 loss : 0.0425 accuracy : 0.9785
7740 loss : 0.0357 accuracy : 0.9902
7750 loss : 0.0405 accuracy : 0.9844
7760 loss : 0.0429 accuracy : 0.9824
7770 loss : 0.0362 accuracy : 0.9863
7780 loss : 0.0393 accuracy : 0.9805
7790 loss : 0.0364 accuracy : 0.9844
7800 loss : 0.0389 accuracy : 0.9902
7810 loss : 0.0357 accuracy : 0.9824
7820 loss : 0.0359 accuracy : 0.9883
7830 loss : 0.0406 accuracy : 0.9824
7840 loss : 0.0359 accuracy : 0.9902
7850 loss : 0.0373 accuracy : 0.9844
7860 loss : 0.0364 accuracy : 0.9863
7870 loss : 0.0361 accuracy : 0.9883
7880 loss : 0.0399 accuracy : 0.9844
7890 loss : 0.041 accuracy : 0.9863
7900 loss : 0.0376 accuracy : 0.9883
7910 loss : 0.0368 accuracy : 0.9863
7920 loss : 0.036 accuracy : 0.9863
7930 loss : 0.0417 accuracy : 0.9863
7940 loss : 0.0362 accuracy : 0.9863
7950 loss : 0.0379 accuracy : 0.9844
7960 loss : 0.0343 accuracy : 0.9883
7970 loss : 0.0369 accuracy : 0.9844
7980 loss : 0.0371 accuracy : 0.9902
```

```
7990 loss : 0.0385 accuracy : 0.9902
8000 loss : 0.0389 accuracy : 0.9883
```

```
# K2020008 : Matplotlib으로 결과 시각화
import matplotlib.pyplot as plt

# K2020008 : Matplotlib는 파이썬에서 데이타를 차트나 플롯(Plot)으로 그려주는 라이브러리
# K2020008 : 최초 창의 크기 -> 가로20 세로 5인치로 설정, wspace의 경우는 subplot간의 간격 0.1
plt.figure(figsize=(20,5))
plt.subplots_adjust(wspace=0.1)

# K2020008 : plt.subplot(nrow,ncol,pos) _> 여러개의 그래프를 그리고 싶을때 (1행, 2열, 위치)
# K2020008 : 손실률 그래프 추이
# K2020008 : 타이틀,라벨 달기 및 폰트 크기 설정
plt.subplot(1,2,1)
plt.title("$loss$",fontsize = 18)
plt.plot(losses)
plt.grid()
plt.xlabel("$epochs$", fontsize = 16)
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)

# K2020008 : 정확도 그래프 추이
# K2020008 : 타이틀,라벨 달기 및 폰트 크기 설정
plt.subplot(1,2,2)
plt.title("$accuracy$", fontsize = 18)
plt.plot(accuracies)
plt.grid()
plt.xlabel("$epochs$", fontsize = 16)
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)

# K2020008 : 그래프 출력
plt.show()
```






```
# K2020008 : x_test를 입력 했을때 output 결과 정확도를 확인 해 본다

output = model(x_test)
output[output>=0.5] = 1
output[output<0.5] = 0

accuracy = sum(sum(y_test.data.numpy() == output.data.T.numpy())) /len(y_test)

print("test_set accuracy :", round(accuracy,4))
```

```
test_set accuracy : 0.9474
```

3.[학습 성능을 향상시킬 수 있는 방법을 2가지 개선]

> 딥러닝 학습 향상을 위한 고려 사항 (http://www.gisdeveloper.co.kr/?p=8443)

- 다양한 경사하강법(Gradient Descent Variants) 최소의 손실값 찾기 위해 손실함수의 미분으로 구한 기울기를 따라 이동하게 되는데, 이동하는 방식에 대한 선택에 대한 것입니다.

> 1. SGD 방식에서 Adam 방식으로 변경
> 2. lr=0.001 -> 0.00001과 학습 반복횟수 200 -> 8000으로 증가 시킴

[비 교]

1. optimizer = torch.optim.SGD(model.parameters(), lr=0.001), 반복 : 200 회

```
160 loss : 0.195 accuracy : 0.9336
170 loss : 0.1925 accuracy : 0.918
180 loss : 0.1974 accuracy : 0.9102
190 loss : 0.1942 accuracy : 0.9199
200 loss : 0.2015 accuracy : 0.9277

test_set accuracy : 0.9298
```

2. optimizer = torch.optim.Adam(model.parameters(), lr=0.000001), 반복 : 8000 회

```
7950 loss : 0.0379 accuracy : 0.9844
7960 loss : 0.0343 accuracy : 0.9883
7970 loss : 0.0369 accuracy : 0.9844
7980 loss : 0.0371 accuracy : 0.9902
7990 loss : 0.0385 accuracy : 0.9902
8000 loss : 0.0389 accuracy : 0.9883

test_set accuracy : 0.947
```

reCAPTCHA 서비스에 연결할 수 없습니다. 인터넷 연결을 확인한 후 페이지를 새로고침하여 reCAPTCHA 보안문자를 다시 로드하세요.