12주. Keras DNN			
학번	32183164	이름	이석현

Q1 (7점) 제공된 PimaIndiansDiabetes.csv 파일에 대해 Keras를 이용한 classification 모델을 개발하고 테스트 하시오

- train/test set을 나누되 test set 은 전체 dataset 의 30% 로 한다.
- hidden layer 의 수는 3~4개, layer별 노드수는 각자 정한다.
- hidden layer 의 활성화 함수는 relu, output layer 의 노드수는 softmax 로 한다
- 기타 필요한 매개변수들은 각자 정한다.
- epoch 는 20,40,60,80, 100 으로 변화시켜 가면서 테스트한다.

* 각 epoch별로 training accuracy 와 test accuracy를 제시한다
  (slide 18과 같은 그래프를 함께 제시)

Source code :

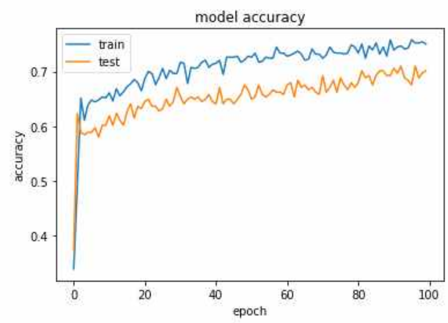
```
# load required modules
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import np_utils
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
import pandas
import matplotlib.pyplot as plt
import numpy as np

# load dataset
dataframe = pandas.read_csv
("D:/data/dataset_0914/PimaIndiansDiabetes.csv")
dataset = dataframe.values
X = dataset[:,0:8].astype(float)
Y = dataset[:,8]

# encode class values as integers
encoder = LabelEncoder()
encoder.fit(Y)
encoded_Y = encoder.transform(Y)

# convert integers to dummy variables (i.e. one hot encoded)
dummy_y = np_utils.to_categorical(encoded_Y)
```

```
# Divide train, test
train_X, test_X , train_y , test_y = train_test_split (X, dummy_y ,
test_size =0.3, random_state =321)

# define model (DNN structure)
epochs = 100
batch_size = 20

model = Sequential()
model.add(Dense(12, input_dim =8 , activation = 'relu'))
model.add(Dense(10, activation = 'relu'))
model.add(Dense(6, activation = 'relu'))
model.add(Dense(2, activation = 'softmax'))
model.summary() # show model structure

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

# model fitting (learning)
disp   =   model.fit(train_X   ,   train_y   ,batch_size   =   batch_size,
epochs=epochs, verbose=1, validation_data = (test_X , test_y))

# Test model
pred = model.predict(test_X)
print(pred)
y_classes = [np.argmax(y, axis=None, out=None) for y in pred]
print(y_classes) # result of prediction

# model performance
score = model.evaluate(test_X , test_y , verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

# summarize history for accuracy
plt.plot(disp.history ['accuracy'])
plt.plot(disp.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

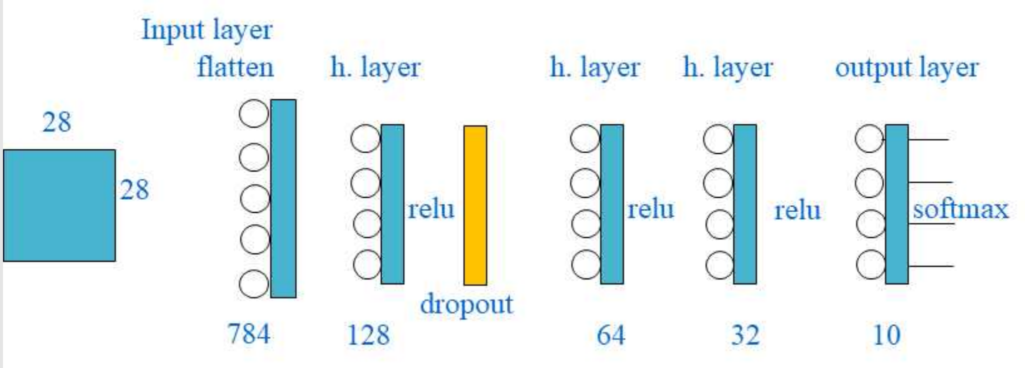
실행화면 캡처:

```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_33 (Dense)             (None, 12)                108
_________________________________________________________________
dense_34 (Dense)             (None, 10)                130
_________________________________________________________________
dense_35 (Dense)             (None, 6)                 66
_________________________________________________________________
dense_36 (Dense)             (None, 2)                 14
=================================================================
Total params: 318
Trainable params: 318
Non-trainable params: 0
```

```
[1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0,
0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0,
1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0,
0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1,
1]
Test loss: 0.5870097129891961
Test accuracy: 0.701298713684082
```



Q2 (3점) chap12_keras_dnn_2.pdf 파일의 source code를 다음과 같은 DNN architecture 로 수정하여 테스트 하시오



Source code :

```
// source code 의 폰트는 Courier10 BT Bold으로 하시오
# load required modules
from keras.datasets import mnist
from keras import optimizers
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout
from keras.utils import np_utils
import matplotlib.pyplot as plt
import numpy as np

# load dataset
(train_X , train_y ), (test_X , test_y ) = mnist.load_data()
train_X, test_X = train_X / 255.0, test_X / 255.0
# one hot encoding
train_y = np_utils.to_categorical(train_y)
test_y = np_utils.to_categorical(test_y)

# define model (DNN structure)
epochs = 20
batch_size = 128
learning_rate = 0.01

model = Sequential()
model.add(Flatten(input_shape=(28, 28)))
model.add(Dense(128, activation='relu'))
model.add(Dropout(rate = 0.4))
model.add(Dense(64,activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(10, activation = 'softmax'))

model.summary()

# Compile model
adam = optimizers.adam(lr=learning_rate)
model.compile(loss='categorical_crossentropy',      optimizer=adam,
metrics=['accuracy'])

# model fitting (learning)
disp   =   model.fit(train_X   ,   train_y   ,batch_size=batch_size,
epochs=epochs, verbose=1, validation_split = 0.2)
```

```
// source code 의 폰트는 Courier10 BT Bold으로 하시오
# Test model
pred = model.predict(test_X)
print(pred)
y_classes = [np.argmax (y, axis=None, out=None) for y in pred]
print(y_classes) # result of prediction
# model performance
score = model.evaluate(test_X , test_y , verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
# summarize history for accuracy
plt.plot(disp.history ['accuracy'])
plt.plot(disp.history ['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('eopch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show
```

실행화면 캡쳐:

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_2 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_4 (Dense)              (None, 128)               100480
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_5 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_6 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_7 (Dense)              (None, 10)                330
=================================================================
Total params: 111,146
Trainable params: 111,146
Non-trainable params: 0
```

```
-----------------------------------------------------------------
Train on 48000 samples, validate on 12000 samples
Epoch 1/20
48000/48000 [==============================] - 2s 43us/step - loss: 0.4131 - accuracy: 0.8721 - val_loss: 0.1941 - val_accuracy: 0.9448
Epoch 2/20
48000/48000 [==============================] - 2s 36us/step - loss: 0.2593 - accuracy: 0.9235 - val_loss: 0.1373 - val_accuracy: 0.9618
Epoch 3/20
48000/48000 [==============================] - 2s 34us/step - loss: 0.2222 - accuracy: 0.9341 - val_loss: 0.1322 - val_accuracy: 0.9644
Epoch 4/20
48000/48000 [==============================] - 2s 35us/step - loss: 0.2029 - accuracy: 0.9411 - val_loss: 0.1348 - val_accuracy: 0.9627
Epoch 5/20
48000/48000 [==============================] - 2s 38us/step - loss: 0.1985 - accuracy: 0.9432 - val_loss: 0.1319 - val_accuracy: 0.9661
Epoch 6/20
48000/48000 [==============================] - 2s 34us/step - loss: 0.1799 - accuracy: 0.9496 - val_loss: 0.1171 - val_accuracy: 0.9682
Epoch 7/20
48000/48000 [==============================] - 2s 35us/step - loss: 0.1762 - accuracy: 0.9499 - val_loss: 0.1127 - val_accuracy: 0.9692
Epoch 8/20
48000/48000 [==============================] - 2s 34us/step - loss: 0.1732 - accuracy: 0.9514 - val_loss: 0.1194 - val_accuracy: 0.9669
Epoch 9/20
48000/48000 [==============================] - 2s 34us/step - loss: 0.1673 - accuracy: 0.9523 - val_loss: 0.1212 - val_accuracy: 0.9682
Epoch 10/20
48000/48000 [==============================] - 2s 34us/step - loss: 0.1674 - accuracy: 0.9524 - val_loss: 0.1086 - val_accuracy: 0.9713
Epoch 11/20
48000/48000 [==============================] - 2s 34us/step - loss: 0.1640 - accuracy: 0.9544 - val_loss: 0.1205 - val_accuracy: 0.9695
Epoch 12/20
48000/48000 [==============================] - 2s 34us/step - loss: 0.1568 - accuracy: 0.9561 - val_loss: 0.1175 - val_accuracy: 0.9696
Epoch 13/20
48000/48000 [==============================] - 2s 35us/step - loss: 0.1509 - accuracy: 0.9585 - val_loss: 0.1140 - val_accuracy: 0.9710
Epoch 14/20
48000/48000 [==============================] - 2s 34us/step - loss: 0.1570 - accuracy: 0.9572 - val_loss: 0.1151 - val_accuracy: 0.9705
Epoch 15/20
48000/48000 [==============================] - 2s 34us/step - loss: 0.1556 - accuracy: 0.9564 - val_loss: 0.1065 - val_accuracy: 0.9728
Epoch 16/20
48000/48000 [==============================] - 2s 34us/step - loss: 0.1468 - accuracy: 0.9594 - val_loss: 0.1106 - val_accuracy: 0.9714
Epoch 17/20
48000/48000 [==============================] - 2s 35us/step - loss: 0.1434 - accuracy: 0.9596 - val_loss: 0.1070 - val_accuracy: 0.9722
Epoch 18/20
48000/48000 [==============================] - 2s 33us/step - loss: 0.1430 - accuracy: 0.9599 - val_loss: 0.1203 - val_accuracy: 0.9696
Epoch 19/20
48000/48000 [==============================] - 2s 34us/step - loss: 0.1374 - accuracy: 0.9625 - val_loss: 0.1058 - val_accuracy: 0.9732
Epoch 20/20
48000/48000 [==============================] - 2s 35us/step - loss: 0.1429 - accuracy: 0.9607 - val_loss: 0.1125 - val_accuracy: 0.9711
<function matplotlib.pyplot.show(*args, **kw)>
```