과목명 : 인공지능
학번: 201601989 이름: 김진섭

과제 설명 AI_HW2 : Deep Feedforward Neural Networks

Requirement

1. Data를 train:test로 분할한다.
2. Keras를 이용해서 CNN을 만든다.
3. 이 때 layer 설정은 예제와 다르게 한다.
4. 성능을 test 한다.

데이터 MNIST.npy, Label.npy

MNIST 데이터는 손으로 쓴 0~9숫자의 데이터이다.
MNIST.npy는 손글씨에 대한 데이터이고, Label.npy는 그 손글씨가 0~9중 어느 글자인지에 대한 Label이다.

```
datay_onehot = to_categorical(datay)
print(datay[0:10])
print(datay_onehot[0:10,:])
```

```
[5 0 4 1 9 2 1 3 1 4]
[[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]]
```

Label은 one hot encoding을 해서 datay를 위와 같이 생성해 주었다.

요구사항 1 Data를 train:test로 분할한다.

```
from sklearn.model_selection import train_test_split
trnx, tstx, trny, tsty = train_test_split(datax_norm, datay_onehot, test_size=0.3, random_state=111)
print(trnx.shape)
print(tstx.shape)
print(trny.shape)
print(trnx[0].shape)
```

```
(42000, 28, 28)
(18000, 28, 28)
(42000, 10)
(28, 28)
```

설명

데이터를 train data와 test data로 분할한 결과이다.
train data를 70% test data를 30%로 분할하였다.

요구사항 2,3 Karas를 이용해서 CNN을 만든다. 이 때 layer 설정은 다르게 한다.

```
input_shape = (28,28,1)

cnn_model = models.Sequential()

cnn_model.add(layers.Conv2D(8, (2,2), padding='same', input_shape=input_shape))
cnn_model.add(layers.BatchNormalization())
cnn_model.add(layers.Activation("relu"))
cnn_model.add(layers.MaxPooling2D((2,2)))

cnn_model.add(layers.Conv2D(16, (2,2), padding='same', input_shape=input_shape))
cnn_model.add(layers.BatchNormalization())
cnn_model.add(layers.Activation("relu"))
cnn_model.add(layers.MaxPooling2D((2,2)))

cnn_model.add(layers.Conv2D(16, (2,2), padding='same'))
cnn_model.add(layers.BatchNormalization())
cnn_model.add(layers.Activation("relu"))
cnn_model.add(layers.Dropout(0.2))
cnn_model.add(layers.MaxPooling2D((2,2)))

cnn_model.add(layers.Flatten())

cnn_model.add(layers.Dense(units = 64, activation = "relu"))
cnn_model.add(layers.Dense(units = 10, activation =  "softmax"))

cnn_model.compile(optimizer='Adam', loss = 'categorical_crossentropy', metrics=['accuracy'])
```

설명

위 처럼 CNN 모델을 생성하였습니다. 맨 처음에 Output 채널 수는 8개 그리고 filter size는 (2,2)이며, MaxPooling을 활용하였습니다.
다음으로는 Output 채널 수는 16개 동일한 filter size와 Maxpooling을 활용하였습니다.
그 다음에는 동일한 작업에 Dropout만 추가적으로 처리해 주었습니다.
다음에는 MLP처럼 처리해 주기 위해서 Flatten을 통해 1차원 벡터로 펴주는 작업을 수행합니다.
마지막에는 Dense를 활용하여 MLP로 처리하여준다. 즉 Dense는 Fully Connect이다.

요구사항 4 성능을 test한 결과

```
Epoch 1/50
42000/42000 [==============================] - 64s 2ms/step - loss: 1.2073 - accuracy: 0.6174 - val_loss: 1.8733 - val_accuracy: 0.2266
Epoch 2/50
42000/42000 [==============================] - 64s 2ms/step - loss: 0.3284 - accuracy: 0.9011 - val_loss: 1.5889 - val_accuracy: 0.3780
Epoch 3/50
42000/42000 [==============================] - 65s 2ms/step - loss: 0.2074 - accuracy: 0.9355 - val_loss: 0.9148 - val_accuracy: 0.6677
Epoch 4/50
42000/42000 [==============================] - 65s 2ms/step - loss: 0.1698 - accuracy: 0.9479 - val_loss: 0.3948 - val_accuracy: 0.8841
Epoch 5/50
42000/42000 [==============================] - 65s 2ms/step - loss: 0.1409 - accuracy: 0.9567 - val_loss: 0.1843 - val_accuracy: 0.9527
Epoch 6/50
42000/42000 [==============================] - 68s 2ms/step - loss: 0.1250 - accuracy: 0.9615 - val_loss: 0.1234 - val_accuracy: 0.9657
Epoch 7/50
42000/42000 [==============================] - 65s 2ms/step - loss: 0.1124 - accuracy: 0.9651 - val_loss: 0.1024 - val_accuracy: 0.9706
Epoch 8/50
42000/42000 [==============================] - 63s 1ms/step - loss: 0.1043 - accuracy: 0.9676 - val_loss: 0.0925 - val_accuracy: 0.9727
Epoch 9/50
42000/42000 [==============================] - 64s 2ms/step - loss: 0.0945 - accuracy: 0.9704 - val_loss: 0.0752 - val_accuracy: 0.9774
Epoch 10/50
42000/42000 [==============================] - 64s 2ms/step - loss: 0.0876 - accuracy: 0.9721 - val_loss: 0.0723 - val_accuracy: 0.9791
                                                                ...
Epoch 40/50
42000/42000 [==============================] - 73s 2ms/step - loss: 0.0369 - accuracy: 0.9874 - val_loss: 0.0474 - val_accuracy: 0.9862
Epoch 41/50
42000/42000 [==============================] - 72s 2ms/step - loss: 0.0357 - accuracy: 0.9881 - val_loss: 0.0465 - val_accuracy: 0.9862
Epoch 42/50
42000/42000 [==============================] - 70s 2ms/step - loss: 0.0359 - accuracy: 0.9883 - val_loss: 0.0409 - val_accuracy: 0.9877
Epoch 43/50
42000/42000 [==============================] - 75s 2ms/step - loss: 0.0349 - accuracy: 0.9882 - val_loss: 0.0399 - val_accuracy: 0.9884
Epoch 44/50
42000/42000 [==============================] - 73s 2ms/step - loss: 0.0360 - accuracy: 0.9881 - val_loss: 0.0391 - val_accuracy: 0.9884
Epoch 45/50
42000/42000 [==============================] - 70s 2ms/step - loss: 0.0352 - accuracy: 0.9878 - val_loss: 0.0404 - val_accuracy: 0.9883
Epoch 46/50
42000/42000 [==============================] - 66s 2ms/step - loss: 0.0334 - accuracy: 0.9887 - val_loss: 0.0381 - val_accuracy: 0.9887
Epoch 47/50
42000/42000 [==============================] - 68s 2ms/step - loss: 0.0331 - accuracy: 0.9889 - val_loss: 0.0376 - val_accuracy: 0.9892
Epoch 48/50
42000/42000 [==============================] - 74s 2ms/step - loss: 0.0324 - accuracy: 0.9888 - val_loss: 0.0376 - val_accuracy: 0.9894
Epoch 49/50
42000/42000 [==============================] - 68s 2ms/step - loss: 0.0308 - accuracy: 0.9898 - val_loss: 0.0394 - val_accuracy: 0.9885
Epoch 50/50
42000/42000 [==============================] - 66s 2ms/step - loss: 0.0409 - accuracy: 0.9863 - val_loss: 0.0371 - val_accuracy: 0.9894
```

설명

성능을 테스트한 결과, 학습을 적게한 경우는 정확도가 0.7 이하로 다소 낮았었는데, 점차 학습을 할수록 올라가고 끝에 가서 정확도가 0.98정도로 매우 높게 형성됨을 알 수 있었다.
학습을 할수록, val_accuracy와 accuracy가 좋아짐을 알 수 있다.