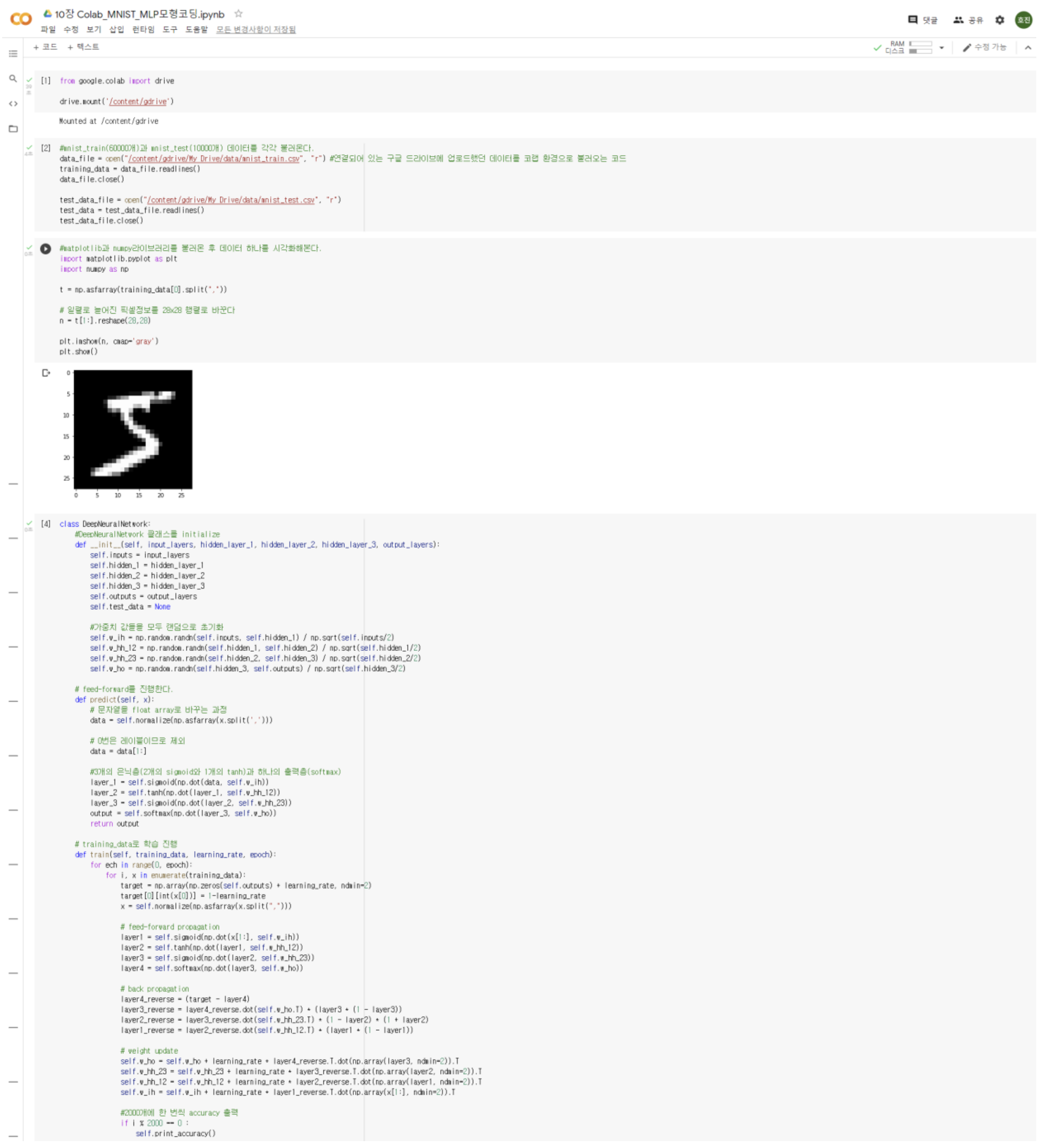
5주차 실습과제

2016314786 김호진

[10장 실습 – Colab MNIST MLP모형코딩]


10장 Colab_MNIST_MLP모형코딩.ipynb
파일 수정 보기 삽입 런타임 도구 도움말 모든 변경사항이 저장됨
+ 코드 + 텍스트
댓글 공유
수정 가능

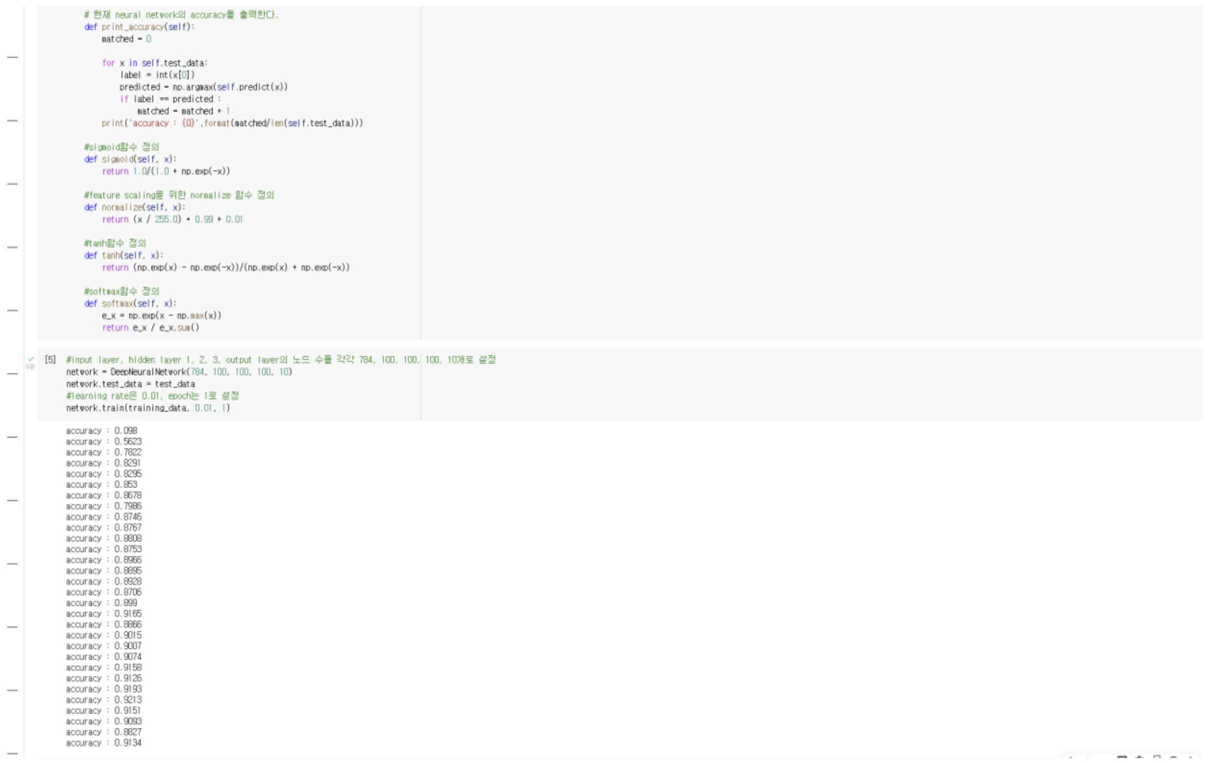
```
[1] from google.colab import drive

    drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

```
[2] #mnist_train(60000개)과 mnist_test(10000개) 데이터를 각각 불러온다.
    data_file = open("/content/gdrive/My Drive/data/mnist_train.csv", "r") #연결되어 있는 구글 드라이브에 업로드했던 데이터를 코랩 환경으로 불러오는 코드
    training_data = data_file.readlines()
    data_file.close()

    test_data_file = open("/content/gdrive/My Drive/data/mnist_test.csv", "r")
    test_data = test_data_file.readlines()
    test_data_file.close()
```

```
#matplotlib과 numpy라이브러리를 불러온 후 데이터 하나를 시각화해본다.
import matplotlib.pyplot as plt
import numpy as np

t = np.asfarray(training_data[0].split(","))

# 일렬로 놓여진 픽셀정보를 28x28 행렬로 바꾼다
n = t[1:].reshape(28,28)

plt.imshow(n, cmap='gray')
plt.show()
```

```
[4] class DeepNeuralNetwork:
        #DeepNeuralNetwork 클래스를 initialize
        def __init__(self, input_layers, hidden_layer_1, hidden_layer_2, hidden_layer_3, output_layers):
            self.inputs = input_layers
            self.hidden_1 = hidden_layer_1
            self.hidden_2 = hidden_layer_2
            self.hidden_3 = hidden_layer_3
            self.outputs = output_layers
            self.test_data = None

            #가중치 값들을 모두 랜덤으로 초기화
            self.w_ih = np.random.randn(self.inputs, self.hidden_1) / np.sqrt(self.inputs/2)
            self.w_hh_12 = np.random.randn(self.hidden_1, self.hidden_2) / np.sqrt(self.hidden_1/2)
            self.w_hh_23 = np.random.randn(self.hidden_2, self.hidden_3) / np.sqrt(self.hidden_2/2)
            self.w_ho = np.random.randn(self.hidden_3, self.outputs) / np.sqrt(self.hidden_3/2)

        # feed-forward를 진행한다.
        def predict(self, x):
            # 문자열을 float array로 바꾸는 과정
            data = self.normalize(np.asfarray(x.split(',')))

            # 0번은 레이블이므로 제외
            data = data[1:]

            #3개의 은닉층(2개의 sigmoid와 1개의 tanh)과 하나의 출력층(softmax)
            layer_1 = self.sigmoid(np.dot(data, self.w_ih))
            layer_2 = self.tanh(np.dot(layer_1, self.w_hh_12))
            layer_3 = self.sigmoid(np.dot(layer_2, self.w_hh_23))
            output = self.softmax(np.dot(layer_3, self.w_ho))
            return output

        # training_data로 학습 진행
        def train(self, training_data, learning_rate, epoch):
            for ech in range(0, epoch):
                for i, x in enumerate(training_data):
                    target = np.array(np.zeros(self.outputs) + learning_rate, ndmin=2)
                    target[0][int(x[0])] = 1-learning_rate
                    x = self.normalize(np.asfarray(x.split(",")))

                    # feed-forward propagation
                    layer1 = self.sigmoid(np.dot(x[1:], self.w_ih))
                    layer2 = self.tanh(np.dot(layer1, self.w_hh_12))
                    layer3 = self.sigmoid(np.dot(layer2, self.w_hh_23))
                    layer4 = self.softmax(np.dot(layer3, self.w_ho))

                    # back propagation
                    layer4_reverse = (target - layer4)
                    layer3_reverse = layer4_reverse.dot(self.w_ho.T) * (layer3 * (1 - layer3))
                    layer2_reverse = layer3_reverse.dot(self.w_hh_23.T) * (1 - layer2) * (1 + layer2)
                    layer1_reverse = layer2_reverse.dot(self.w_hh_12.T) * (layer1 * (1 - layer1))

                    # weight update
                    self.w_ho = self.w_ho + learning_rate * layer4_reverse.T.dot(np.array(layer3, ndmin=2)).T
                    self.w_hh_23 = self.w_hh_23 + learning_rate * layer3_reverse.T.dot(np.array(layer2, ndmin=2)).T
                    self.w_hh_12 = self.w_hh_12 + learning_rate * layer2_reverse.T.dot(np.array(layer1, ndmin=2)).T
                    self.w_ih = self.w_ih + learning_rate * layer1_reverse.T.dot(np.array(x[1:], ndmin=2)).T

                    #2000개에 한 번씩 accuracy 출력
                    if i % 2000 == 0 :
                        self.print_accuracy()
```

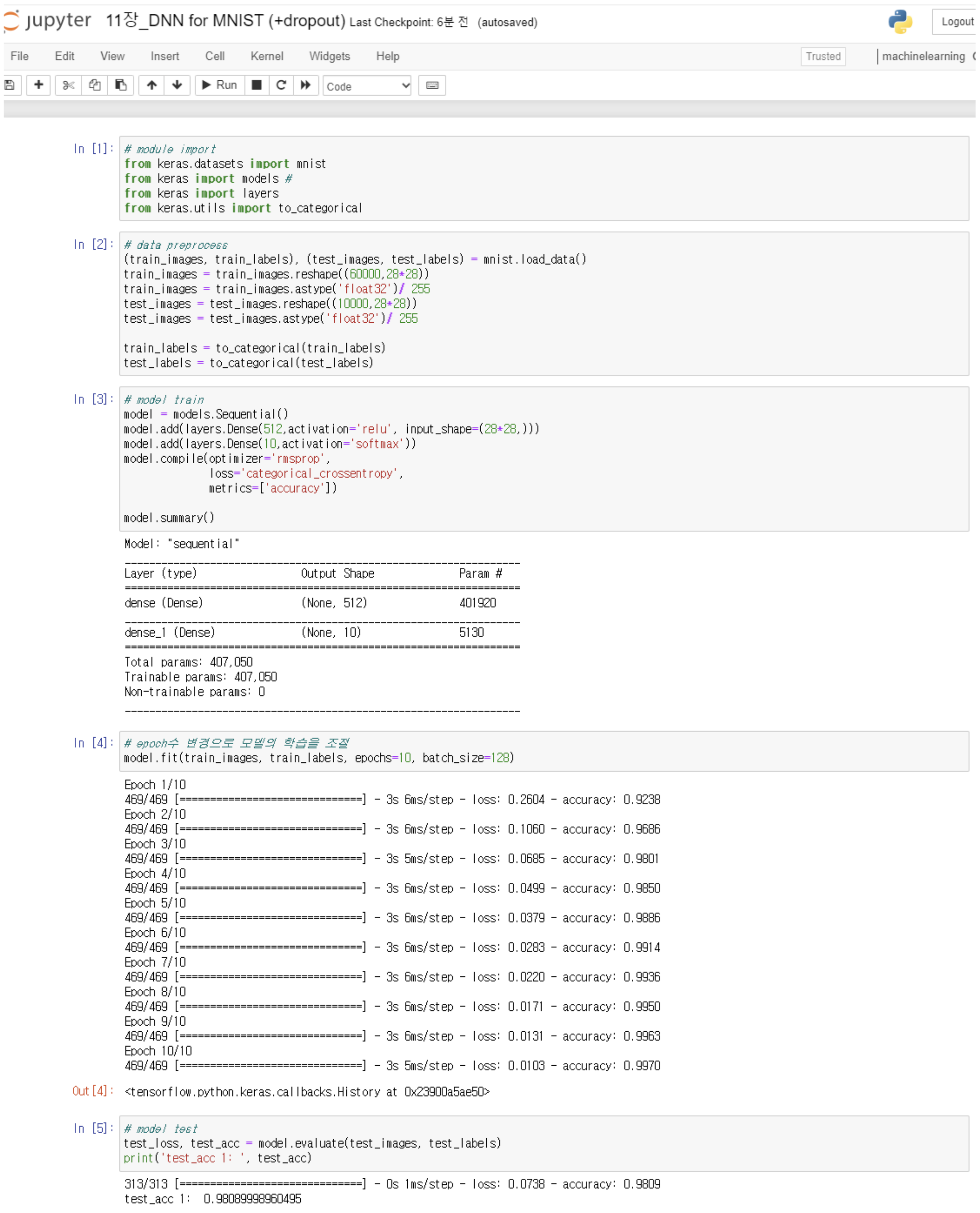


```python
    # 현재 neural network의 accuracy를 출력한다.
    def print_accuracy(self):
        matched = 0

        for x in self.test_data:
            label = int(x[0])
            predicted = np.argmax(self.predict(x))
            if label == predicted :
                matched = matched + 1
        print('accuracy : {0}'.format(matched/len(self.test_data)))

    #sigmoid함수 정의
    def sigmoid(self, x):
        return 1.0/(1.0 + np.exp(-x))

    #feature scaling을 위한 normalize 함수 정의
    def normalize(self, x):
        return (x / 255.0) * 0.99 + 0.01

    #tanh함수 정의
    def tanh(self, x):
        return (np.exp(x) - np.exp(-x))/(np.exp(x) + np.exp(-x))

    #softmax함수 정의
    def softmax(self, x):
        e_x = np.exp(x - np.max(x))
        return e_x / e_x.sum()
```

```python
[5] #input layer, hidden layer 1, 2, 3, output layer의 노드 수를 각각 784, 100, 100, 100, 10개로 설정
    network = DeepNeuralNetwork(784, 100, 100, 100, 10)
    network.test_data = test_data
    #learning rate은 0.01, epoch는 1로 설정
    network.train(training_data, 0.01, 1)
```

```
accuracy : 0.098
accuracy : 0.5623
accuracy : 0.7822
accuracy : 0.8291
accuracy : 0.8295
accuracy : 0.853
accuracy : 0.8678
accuracy : 0.7986
accuracy : 0.8746
accuracy : 0.8767
accuracy : 0.8808
accuracy : 0.8753
accuracy : 0.8966
accuracy : 0.8895
accuracy : 0.8928
accuracy : 0.8706
accuracy : 0.899
accuracy : 0.9165
accuracy : 0.8866
accuracy : 0.9015
accuracy : 0.9007
accuracy : 0.9074
accuracy : 0.9158
accuracy : 0.9126
accuracy : 0.9193
accuracy : 0.9213
accuracy : 0.9151
accuracy : 0.9093
accuracy : 0.8827
accuracy : 0.9134
```

[11장 실습 – DNN for MNIST (+dropout)]


jupyter
11장_DNN for MNIST (+dropout) Last Checkpoint: 6분 전 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
machinelearning
Run
Code

In [1]:
```
# module import
from keras.datasets import mnist
from keras import models #
from keras import layers
from keras.utils import to_categorical
```

In [2]:
```
# data preprocess
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = train_images.reshape((60000,28*28))
train_images = train_images.astype('float32')/ 255
test_images = test_images.reshape((10000,28*28))
test_images = test_images.astype('float32')/ 255

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

In [3]:
```
# model train
model = models.Sequential()
model.add(layers.Dense(512,activation='relu', input_shape=(28*28,)))
model.add(layers.Dense(10,activation='softmax'))
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 512)               401920
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5130
=================================================================
Total params: 407,050
Trainable params: 407,050
Non-trainable params: 0
_________________________________________________________________
```

In [4]:
```
# epoch수 변경으로 모델의 학습을 조절
model.fit(train_images, train_labels, epochs=10, batch_size=128)
```

```
Epoch 1/10
469/469 [==============================] - 3s 6ms/step - loss: 0.2604 - accuracy: 0.9238
Epoch 2/10
469/469 [==============================] - 3s 6ms/step - loss: 0.1060 - accuracy: 0.9686
Epoch 3/10
469/469 [==============================] - 3s 5ms/step - loss: 0.0685 - accuracy: 0.9801
Epoch 4/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0499 - accuracy: 0.9850
Epoch 5/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0379 - accuracy: 0.9886
Epoch 6/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0283 - accuracy: 0.9914
Epoch 7/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0220 - accuracy: 0.9936
Epoch 8/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0171 - accuracy: 0.9950
Epoch 9/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0131 - accuracy: 0.9963
Epoch 10/10
469/469 [==============================] - 3s 5ms/step - loss: 0.0103 - accuracy: 0.9970
```

Out[4]:
```
<tensorflow.python.keras.callbacks.History at 0x23900a5ae50>
```

In [5]:
```
# model test
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('test_acc 1: ', test_acc)
```

```
313/313 [==============================] - 0s 1ms/step - loss: 0.0738 - accuracy: 0.9809
test_acc 1:  0.98089998960495
```

In [6]:
```
model.fit(train_images, train_labels, epochs=20, batch_size=128)
```

```
Epoch 1/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0082 - accuracy: 0.9978
Epoch 2/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0063 - accuracy: 0.9983
Epoch 3/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0048 - accuracy: 0.9987
Epoch 4/20
469/469 [==============================] - 2s 5ms/step - loss: 0.0042 - accuracy: 0.9990
Epoch 5/20
469/469 [==============================] - 3s 5ms/step - loss: 0.0032 - accuracy: 0.9991
Epoch 6/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0025 - accuracy: 0.9995
Epoch 7/20
469/469 [==============================] - 3s 5ms/step - loss: 0.0016 - accuracy: 0.9995
Epoch 8/20
469/469 [==============================] - 3s 5ms/step - loss: 0.0018 - accuracy: 0.9995
Epoch 9/20
469/469 [==============================] - 3s 5ms/step - loss: 0.0013 - accuracy: 0.9997
Epoch 10/20
469/469 [==============================] - 3s 5ms/step - loss: 0.0010 - accuracy: 0.9997
Epoch 11/20
469/469 [==============================] - 3s 5ms/step - loss: 8.0351e-04 - accuracy: 0.9998
Epoch 12/20
469/469 [==============================] - 3s 6ms/step - loss: 6.6165e-04 - accuracy: 0.9998
Epoch 13/20
469/469 [==============================] - 3s 6ms/step - loss: 5.4378e-04 - accuracy: 0.9999
Epoch 14/20
469/469 [==============================] - 3s 6ms/step - loss: 3.7267e-04 - accuracy: 0.9999
Epoch 15/20
469/469 [==============================] - 3s 6ms/step - loss: 3.3445e-04 - accuracy: 0.9998
Epoch 16/20
469/469 [==============================] - 3s 6ms/step - loss: 2.4064e-04 - accuracy: 0.9999
Epoch 17/20
469/469 [==============================] - 3s 6ms/step - loss: 1.9312e-04 - accuracy: 1.0000
Epoch 18/20
469/469 [==============================] - 3s 6ms/step - loss: 1.9193e-04 - accuracy: 1.0000
Epoch 19/20
469/469 [==============================] - 3s 6ms/step - loss: 1.8268e-04 - accuracy: 0.9999
Epoch 20/20
469/469 [==============================] - 3s 6ms/step - loss: 1.6579e-04 - accuracy: 0.9999
```

Out[6]:
```
<tensorflow.python.keras.callbacks.History at 0x23900f462e0>
```

In [7]:
```
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('test_acc 2: ', test_acc)
```

```
313/313 [==============================] - 0s 1ms/step - loss: 0.1175 - accuracy: 0.9837
test_acc 2:  0.9836999773979187
```

In [8]:
```
model2 = models.Sequential()
model2.add(layers.Dense(512,activation='relu', input_shape=(28*28,)))
model2.add(layers.Dropout(0.2,noise_shape=None, seed = None))
model2.add(layers.Dense(10,activation='softmax'))
model2.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

In [9]:
```
model2.fit(train_images, train_labels, epochs=10, batch_size=128)
```

```
Epoch 1/10
469/469 [==============================] - 3s 6ms/step - loss: 0.2729 - accuracy: 0.9218
Epoch 2/10
469/469 [==============================] - 3s 6ms/step - loss: 0.1157 - accuracy: 0.9659
Epoch 3/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0812 - accuracy: 0.9755
Epoch 4/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0632 - accuracy: 0.9805
Epoch 5/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0514 - accuracy: 0.9842
Epoch 6/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0416 - accuracy: 0.9867
Epoch 7/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0373 - accuracy: 0.9886
Epoch 8/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0316 - accuracy: 0.9900
Epoch 9/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0274 - accuracy: 0.9915
Epoch 10/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0231 - accuracy: 0.9930
```

Out[9]:
```
<tensorflow.python.keras.callbacks.History at 0x23900cd7af0>
```

In [10]:
```
test_loss, test_acc = model2.evaluate(test_images, test_labels)
print('test_acc 3: ', test_acc)
```

```
313/313 [==============================] - 0s 1ms/step - loss: 0.0663 - accuracy: 0.9831
test_acc 3:  0.9830999970436096
```

```
In [11]: model2.fit(train_images, train_labels, epochs=20, batch_size=128)
```

```
Epoch 1/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0201 - accuracy: 0.9933
Epoch 2/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0182 - accuracy: 0.9942: 0s - loss: 0
Epoch 3/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0162 - accuracy: 0.9949
Epoch 4/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0142 - accuracy: 0.9954
Epoch 5/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0129 - accuracy: 0.9960
Epoch 6/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0121 - accuracy: 0.9963
Epoch 7/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0105 - accuracy: 0.9965
Epoch 8/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0093 - accuracy: 0.9970
Epoch 9/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0094 - accuracy: 0.9970
Epoch 10/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0083 - accuracy: 0.9974
Epoch 11/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0071 - accuracy: 0.9978
Epoch 12/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0071 - accuracy: 0.9977
Epoch 13/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0068 - accuracy: 0.9978
Epoch 14/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0058 - accuracy: 0.9980
Epoch 15/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0052 - accuracy: 0.9984
Epoch 16/20
469/469 [==============================] - 3s 7ms/step - loss: 0.0051 - accuracy: 0.9983
Epoch 17/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0048 - accuracy: 0.9983
Epoch 18/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0049 - accuracy: 0.9985
Epoch 19/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0046 - accuracy: 0.9985
Epoch 20/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0042 - accuracy: 0.9986
```

```
Out[11]: <tensorflow.python.keras.callbacks.History at 0x239023c5370>
```

```
In [12]: test_loss, test_acc = model2.evaluate(test_images, test_labels)
         print('test_acc 4: ', test_acc)
```

```
313/313 [==============================] - 0s 1ms/step - loss: 0.0963 - accuracy: 0.9838
test_acc 4:  0.9837999939918518
```