초보자 딥러닝 입문하기

- keras를 이용한 딥러닝을 이용한 손글씨 인식

학습 내용

- MNIST 데이터 셋을 활용하여 간단한 딥러닝 모델을 구현해 본다.

라이브러리 불러오기

- 대표적인 딥러닝 구현 라이브러리
  - Tensorflow, Keras, Pytorch

환경

- google colab
  - tensorflow 2.15.0
  - keras 2.15.0
  - python 3.10.12

목차

01. 라이브러리 임포트

목차로 이동하기

In [7]:
```
import tensorflow as tf
import keras
import sys

### 라이브러리 확인
import numpy as np  # 선형대수 관련(배열 생성 관련)
import matplotlib   # 시각화
import matplotlib.pyplot as np
```

In [8]:
```
print(tf.__version__)
print(keras.__version__)
print(sys.version)
```

```
2.15.0
2.15.0
3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0]
```

02. 케라스 이해하기

목차로 이동하기

케라스(Keras) 의미 이해

- ONEIROS(Open-ended Neuro-Electronic Intelligent Robot Operating System) 프로젝트 일부
- 오네이로스의 의미는 꿈을 의인화 시킨 신이다.

케라스의(Keras) 주요 특징 4가지

- 개발 및 유지 보수 : 프랑소와 쏠레(Francois Chollet) - 구글에 있음.(tf 2.0 이후 케라스 도입)
- 모듈화(Modularity) : 모듈은 독립적이고 최소한의 제약사항으로 연결
- 최소주의(Minimalism) : 각 모듈은 짧고 간결하다.
- 쉬운 확장성 : 새로운 클래스나 함수로 모듈을 아주 쉽게 추가
- 파이썬 기반 : 파이썬 코드로 모델들이 정의

케라스(Keras) 딥러닝 모델 만들기 절차 이해

- 가. 데이터 셋 준비 및 데이터 형태 맞추기
  - 데이터 준비(훈련셋, 검증셋, 시험셋 등)
  - 딥러닝 모델의 학습 및 평가를 위한 데이터 형태 맞추기(포맷 변환)
- 나. 모델 구축
  - 모델(Sequential)을 선택 후, 레이어를 추가하여 구성
- 다. 모델 학습과정 설정하기(학습에 대한 설정 - compile())
  - 학습에 대한 설정, 손실 함수 및 최적화 방법 정의
- 라. 모델 학습(모델을 훈련셋(train)으로 학습 - fit() 함수)
- 마. 학습과정 살펴보기(훈련셋, 검증셋의 손실 및 정확도 측정)
- 바. 모델 평가(evaluate()) - 준비된 시험셋으로 학습 모델 평가
- 사. 모델 사용하기(predict())

기본 코드 이해

- 십진수 숫자를 0,1로 이루어진 것으로 변환(스칼라를 1차벡터로 변환)
  - from keras.utils import to_categorical
- 데이터 셋(손글씨)
  - from keras.datasets import mnist
- 모델 만들기
  - from keras.models import Sequential
- 딥러닝 층 구성

- (Dense-층세부내용 Activation:활성화함수)
- from keras.layers import Dense, Activation

In [26]:
```
# 사용할 패키지 불러오기
from keras.utils import to_categorical
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Activation
```

In [27]:
```
# 데이터셋 불러오기 (학습, 테스트)
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

03. 손글씨 데이터 시각화

목차로 이동하기

- x_train : 손글씨 그림의 픽셀정보(28x28) 총 784개의 픽셀정보
- y_train : 손글씨 그림의 숫자(0~9) 정보

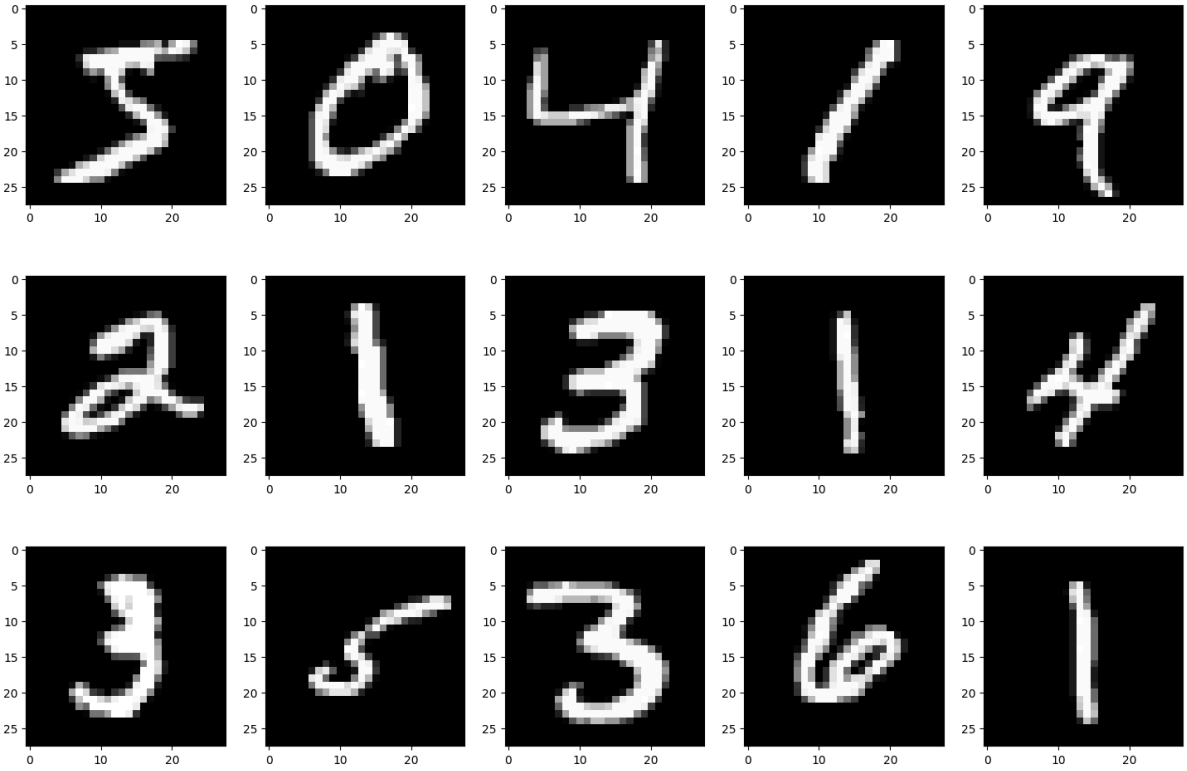
In [28]:
```
import matplotlib.pyplot as plt
```

In [30]:
```
figure,axes = plt.subplots(nrows=3, ncols=5)  # 3행 5열의 구조
figure.set_size_inches(18,12)  # 전체 크기

plt.gray()
print("label={}".format(y_train[0:15]))   # x데이터 0~14개 가져오기

col = 0
for row in range(0,3):
    col = row * 5
    axes[row][0].imshow(X_train[col])  # 0,5,10의 값을 갖는 위치 값 이미지 표시
    axes[row][1].imshow(X_train[col+1])# 1,6,11의 값을 갖는 위치 값 이미지 표시
    axes[row][2].imshow(X_train[col+2])# 2,7,12의 값을 갖는 위치 값 이미지 표시
    axes[row][3].imshow(X_train[col+3])# 3,8,13의 값을 갖는 위치 값 이미지 표시
    axes[row][4].imshow(X_train[col+4])# 4,9,114의 값을 갖는 위치 값 이미지 표시
```

```
label=[5 0 4 1 9 2 1 3 1 4 3 5 3 6 1]
```



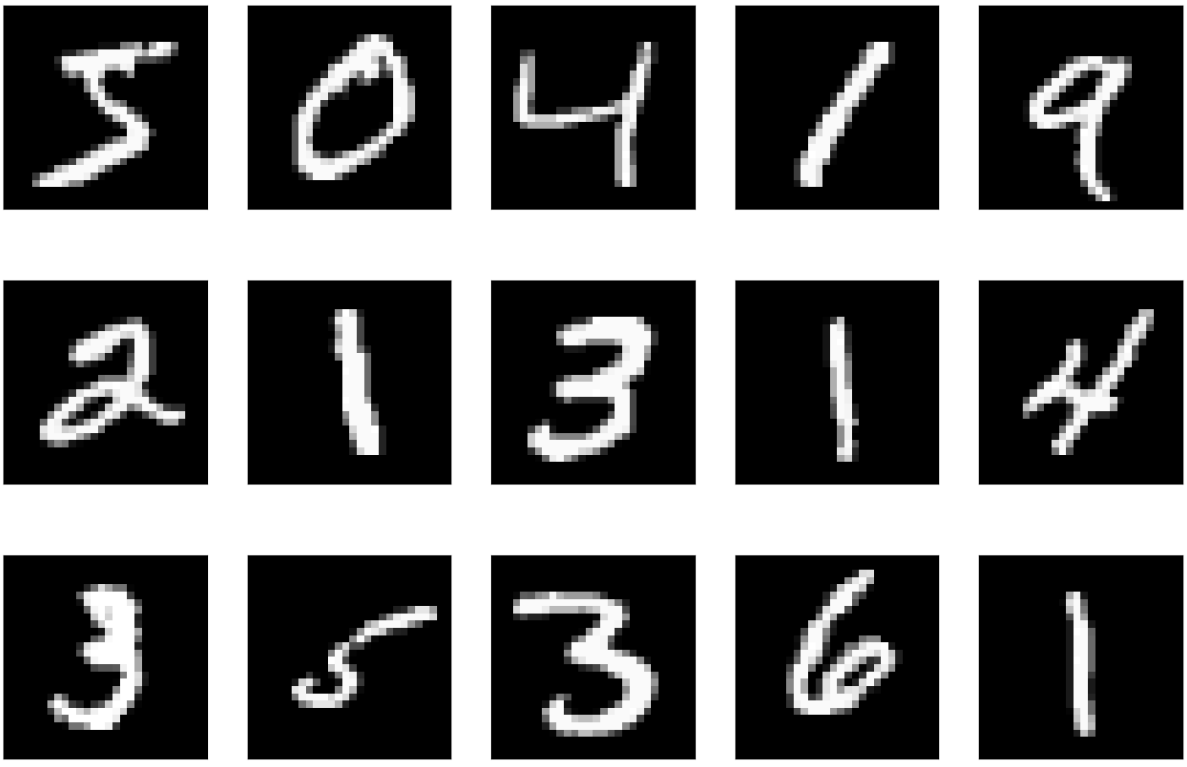
좀 더 간단히 시각화하기

In [32]:
```
fig, axes = plt.subplots(3, 5, figsize=(18,12),
                         subplot_kw= {'xticks':(), 'yticks':() })

print("label={}".format(y_train[0:15]))   # x데이터 0~14개 가져오기

for image, ax in zip( X_train, axes.ravel() ):
    ax.imshow(image) # 이미지 표시
```

```
label=[5 0 4 1 9 2 1 3 1 4 3 5 3 6 1]
```



데이터 처리

- 입력 데이터 : x_train, x_test (60000, 28, 28) -> (60000, 784)
- 레이블 데이터 : y_train, y_test 숫자(0~9)를 2진 벡터형태로 변경(0 0 0 0 0 1 0 0 0 0 0 0)

In [33]:
```
X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

04 모델 구축 및 학습, 평가

목차로 이동하기

- 첫번째 add =>
  - Dense() : 층의 세부 내용 지정
  - input_dim : 입력층의 뉴런 개수, units : 입력층 이후의 은닉층의 뉴런 개수
  - activation : 활성화 함수 지정
- 두번째 add, 마지막층 추가 =>
  - units = 10 : 손글씨의 예측 값이 0~9사이의 값이므로 10개
  - 일반적으로 마지막 층(출력층)이 범주형 여러개 예측일 경우 softmax 함수를 사용

In [34]:
```
# 2. 모델 구성하기
model = Sequential()
model.add(Dense(units=64, input_dim=28*28, activation='relu'))
model.add(Dense(units=10, activation='softmax'))
```

모델의 세부 설정

- 비용함수(loss) : categorical_crossentropy(범주형(다항분류)의 경우)
- 최적화함수(optimizer) : sgd(Stochastic Gradient Descent) 알고리즘 이용
- 최종 평가(metrics) : 정확도로 측정(손글씨를 정답을 맞춰는지 아닌지)

In [35]:
```
# 3. 모델 학습과정 설정하기
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
```

모델 학습시키기

- validation_data : 각 에폭당 한번 사용. 과적합되었는지 확인.
- epochs : 모든 데이터가 훈련에 한번씩 사용되는 횟수.
- batch_size : 적절하게 선택, 배치 크기가 너무 크면 많은 양의 메모리와 계산 능력이 필요.
  - 학습 데이터 크기
  - 모델의 복잡성 고려
  - 사용 가능한 컴퓨팅 리소스

In [36]:
```
# 4. 모델 학습시키기
# validation_data : 모델을 학습할때는 기본 데이터를 이용하고,
#                   평가시에 사용할 데이터를 지정
hist = model.fit(X_train, y_train,
                 validation_data=(X_test, y_test),
                 epochs=10, batch_size=32)
```

```
Epoch 1/10
1875/1875 [==============================] - 6s 3ms/step - loss: 5.7090 - accuracy:
0.1385 - val_loss: 2.1920 - val_accuracy: 0.1624
Epoch 2/10
1875/1875 [==============================] - 4s 2ms/step - loss: 2.2406 - accuracy:
0.1491 - val_loss: 2.1901 - val_accuracy: 0.1681
Epoch 3/10
1875/1875 [==============================] - 5s 3ms/step - loss: 2.2583 - accuracy:
0.1502 - val_loss: 2.3077 - val_accuracy: 0.1135
Epoch 4/10
1875/1875 [==============================] - 5s 2ms/step - loss: 2.2896 - accuracy:
0.1196 - val_loss: 2.2501 - val_accuracy: 0.1352
Epoch 5/10
1875/1875 [==============================] - 4s 2ms/step - loss: 2.2406 - accuracy:
0.1442 - val_loss: 2.1748 - val_accuracy: 0.1662
Epoch 6/10
1875/1875 [==============================] - 6s 3ms/step - loss: 2.2110 - accuracy:
0.1646 - val_loss: 2.1609 - val_accuracy: 0.1744
Epoch 7/10
1875/1875 [==============================] - 4s 2ms/step - loss: 2.2574 - accuracy:
0.1382 - val_loss: 2.1838 - val_accuracy: 0.1631
Epoch 8/10
1875/1875 [==============================] - 4s 2ms/step - loss: 2.1798 - accuracy:
0.1675 - val_loss: 2.1802 - val_accuracy: 0.1614
Epoch 9/10
1875/1875 [==============================] - 6s 3ms/step - loss: 2.1996 - accuracy:
0.1641 - val_loss: 2.2824 - val_accuracy: 0.1315
Epoch 10/10
1875/1875 [==============================] - 4s 2ms/step - loss: 2.2158 - accuracy:
0.1592 - val_loss: 2.1591 - val_accuracy: 0.1730
```

학습을 통해 얻어진 값 살펴보기

- epoch 10번이므로 값이 각각 10개씩

In [37]:
```
# 5. 학습과정 살펴보기
print('## training loss and acc ##')
print(hist.history['loss'])        # 'val_loss' : 평가셋 손실값
print(hist.history['accuracy'])    # 'val_acc'  : 평가셋 정확도
```

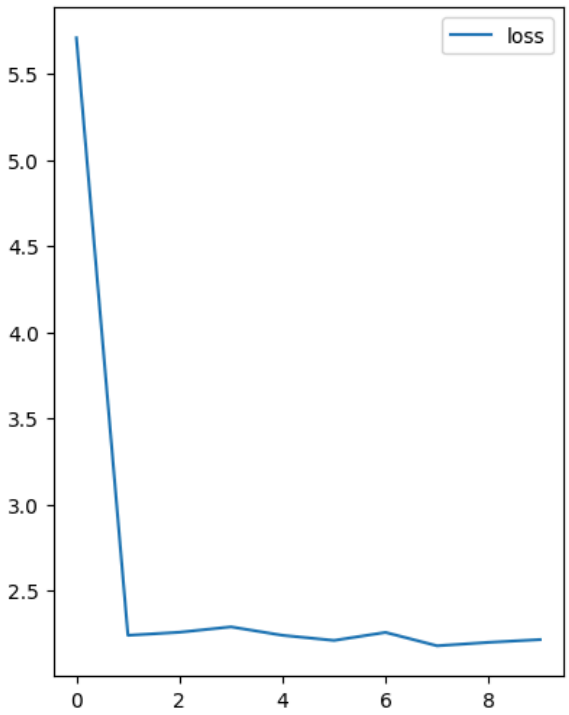
```
## training loss and acc ##
[5.708974838256836, 2.240565061569214, 2.258289337158203, 2.289607524871826, 2.24057
31678009033, 2.2109503746032715, 2.2573599815368652, 2.179774045944214, 2.1995995044
70825, 2.215761661529541]
[0.13854999840259552, 0.14906667172908783, 0.1501999944448471, 0.11964999884366989,
0.14418333768844604, 0.16459999978542328, 0.1381833404302597, 0.1674666702747345, 0.
16410000622272491, 0.15921667218208313]
```

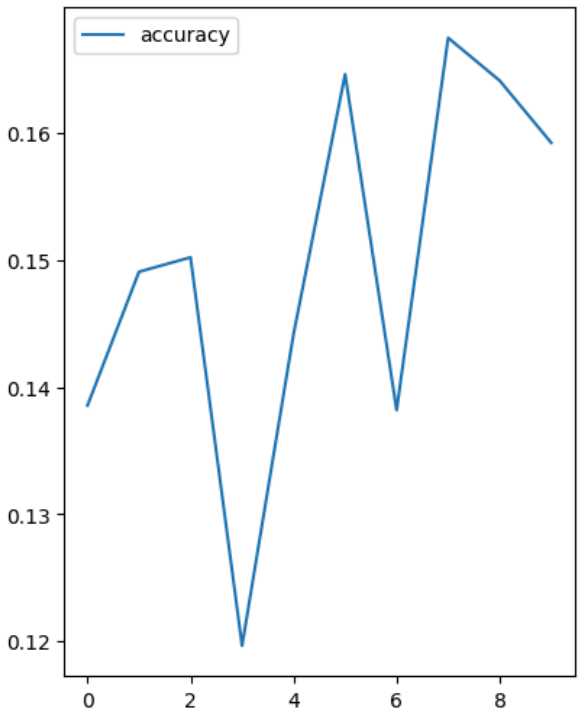
In [38]:
```
plt.figure(figsize=(10, 6))

plt.subplot(1,2,1)
plt.plot(hist.history['loss'], label="loss")
plt.legend()

plt.subplot(1,2,2)
plt.plot(hist.history['accuracy'], label="accuracy")
plt.legend()
```

Out[38]:
```
<matplotlib.legend.Legend at 0x7fc5d9e78040>
```





모델 평가하기

- 테스트 데이터 셋(x_test, y_test) 을 이용
- 첫번째 값이 : loss 손실값.
- 두번째 값이 : 정확도

In [39]:
```
loss_and_metrics = model.evaluate(X_test, y_test, batch_size=32)
print('## evaluation loss and_metrics ##')
print(loss_and_metrics)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 2.1591 - accuracy: 0.
1730
## evaluation loss and_metrics ##
[2.1590678691864014, 0.17299999296665192]
```

실습 과제

- 가. 픽셀값을 0~255를 0~1사이로 변경해 보기
- 나. epoch=10 -> 20를 변경하면 해 보자. 어떤 현상이 발생되는가?
- 다. batch_size=32 -> 16를 변경한 이후에 해 보자.
- 라. 기타 실습

참고 동영상 : https://www.youtube.com/watch?v=aircAruvnKk