CNN개념이해_MNIST

- 본 파일은 GPU 런타임으로 연결됩니다.
- 경우에 따라서는 GPU 연결이 원활하지 않을 수도 있습니다.

1.환경준비

(1) 라이브러리 로딩

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import random as rd
import cv2, os  # cv2 : OpenCV # 이미지 처리 라이브러리

from sklearn.model_selection import train_test_split
from sklearn.metrics import *

from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D # CNN
from keras.backend import clear_session
from keras.optimizers import Adam
from keras.datasets import mnist, fashion_mnist
```

- 함수 만들기

```
# 학습곡선 함수
def dl_history_plot(history):
    plt.figure(figsize=(10,6))
    plt.plot(history['loss'], label='train_err', marker = ',')
    plt.plot(history['val_loss'], label='val_err', marker = ',')

    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend()
    plt.grid()
    plt.show()
```

(2) 데이터로딩

```
# 케라스 데이터셋으로 부터 mnist 불러오기
(x_train, y_train), (x_val, y_val) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

```
x_train.shape, y_train.shape
```

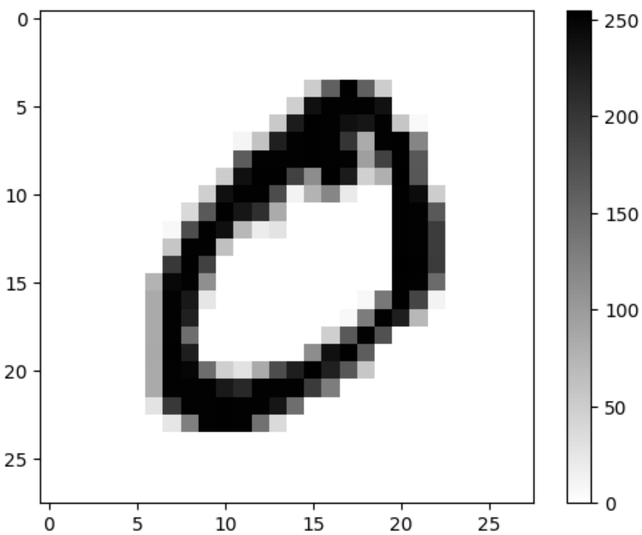
```
((60000, 28, 28), (60000,))
```

```
class_names = ['0','1','2','3','4','5','6','7','8','9']
```

2 데이터 살펴보기

- 이미지 확인하기

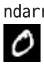
```
# 아래 숫자를 바꿔가며 화면에 그려 봅시다.
n = 1

plt.imshow(x_train[n], cmap=plt.cm.binary)
plt.colorbar()
plt.show()
```



- 이미지를 픽셀 값(배열 값)으로 확인하기

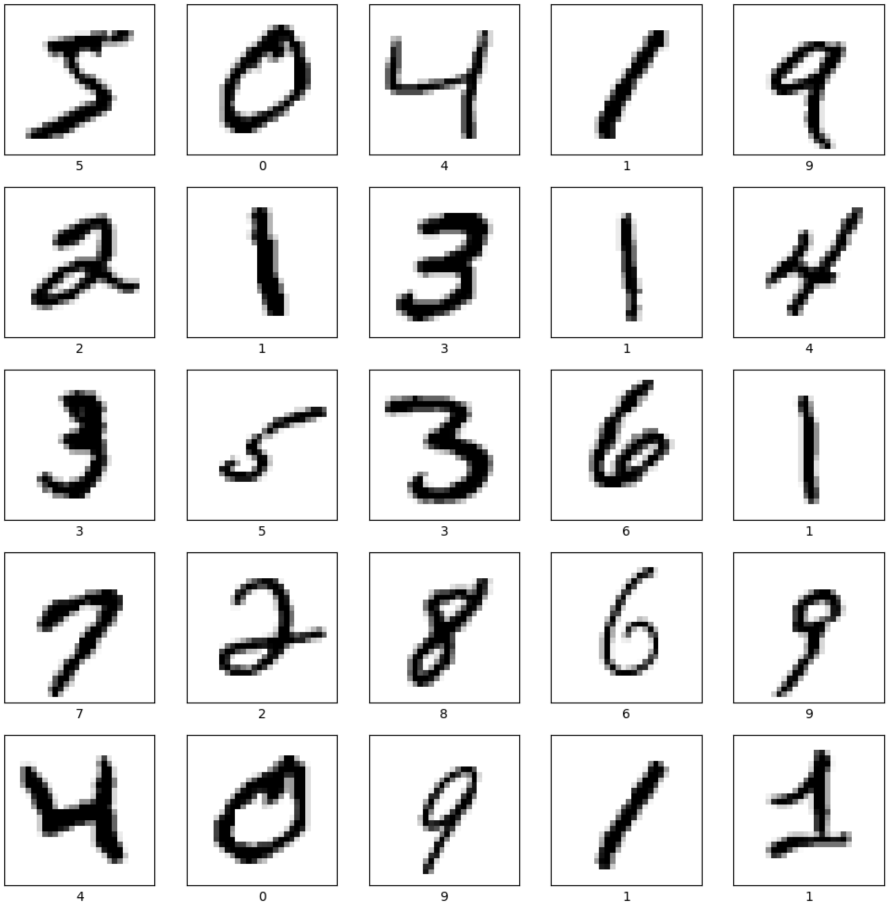
```
# numpy array 화면 출력시 문자열 길이 조정
np.set_printoptions(linewidth=500)
x_train[n]
```

ndarray (28, 28) show data



- 여러 이미지 확인하기

```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(x_train[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[y_train[i]])
plt.tight_layout()
plt.show()
```



3.데이터 준비

- CNN은 3차원 구조의 이미지(데이터셋은 4차원)를 입력해야 합니다.(input_shape)

```
x_train.shape, x_val.shape
```

```
((60000, 28, 28), (10000, 28, 28))
```

- reshape를 이용하여 다음과 같이 변환해 봅시다.
  - x_train.shape : (60000, 28, 28, 1)
  - x_val.shape : (10000, 28, 28, 1)

```
x_train = x_train.reshape(60000,28,28,1)
x_val = x_val.reshape(10000,28,28,1)
```

- Scaling : Min-Max

- 0-255 값으로 되어 있는 데이터를 0-1사이 값으로 변환
- x_train, x_val 그냥 255로 나누면 됨

```
x_train = x_train / 255.
x_val = x_val / 255.
```

4.CNN 기본 모델링

(1) 모델 설계

- CNN 모델의 기본 구조
  - Conv2D : 지역적인 특징 도출
  - MaxPooling : 요약
  - Flatten : 1차원으로 펼치기
  - Dense : Output Layer

```
clear_session()

model = Sequential([Conv2D(16, kernel_size = 3, input_shape=(28, 28, 1), # kernel_size 3x3 크기의 지역에서 특징 도출
                           padding='same', activation='relu'),  # strides = 1(기본값,1)
                    MaxPooling2D(pool_size = 2 ),          # 중요한 것만 요약    # strides = 2(기본값이 pool_size 동일)
                    Flatten(),                               # 1차원으로 펼침
                    Dense(10, activation='softmax')
])

model.summary()

model.compile(optimizer=Adam(learning_rate=0.001), loss='sparse_categorical_crossentropy')
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 16)        160       
                                                                 
 max_pooling2d (MaxPooling2  (None, 14, 14, 16)        0         
 D)                                                              
                                                                 
 flatten (Flatten)           (None, 3136)              0         
                                                                 
 dense (Dense)               (None, 10)                31370     
                                                                 
=================================================================
Total params: 31530 (123.16 KB)
Trainable params: 31530 (123.16 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

(2) 학습

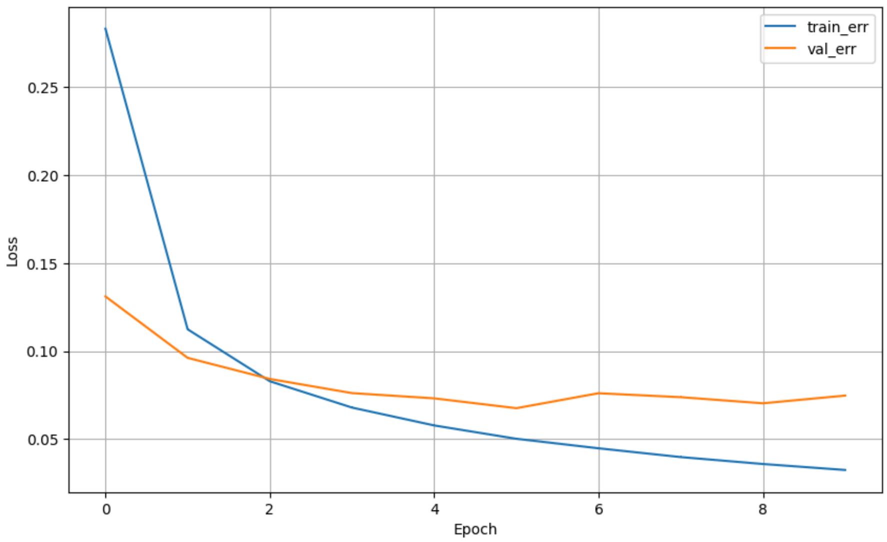
```
history = model.fit(x_train, y_train, epochs = 10, validation_split=0.2).history
```

```
Epoch 1/10
1500/1500 [==============================] - 8s 3ms/step - loss: 0.2833 - val_loss: 0.1311
Epoch 2/10
1500/1500 [==============================] - 4s 3ms/step - loss: 0.1124 - val_loss: 0.0962
Epoch 3/10
1500/1500 [==============================] - 4s 3ms/step - loss: 0.0829 - val_loss: 0.0842
Epoch 4/10
1500/1500 [==============================] - 5s 3ms/step - loss: 0.0679 - val_loss: 0.0761
Epoch 5/10
1500/1500 [==============================] - 5s 4ms/step - loss: 0.0578 - val_loss: 0.0732
Epoch 6/10
1500/1500 [==============================] - 5s 3ms/step - loss: 0.0502 - val_loss: 0.0676
Epoch 7/10
1500/1500 [==============================] - 4s 3ms/step - loss: 0.0448 - val_loss: 0.0761
Epoch 8/10
1500/1500 [==============================] - 4s 3ms/step - loss: 0.0398 - val_loss: 0.0738
Epoch 9/10
1500/1500 [==============================] - 5s 3ms/step - loss: 0.0358 - val_loss: 0.0703
Epoch 10/10
```

```
1500/1500 [==============================] - 4s 3ms/step - loss: 0.0324 - val_loss: 0.0747
```

(3) 학습결과 그래프

```
dl_history_plot(history)
```



(4) 예측 및 평가

```
pred = model.predict(x_val)
```

```
313/313 [==============================] - 1s 2ms/step
```

```
pred_1 = pred.argmax(axis=1)
```

```
print(accuracy_score(y_val,pred_1))
print('-'*60)
print(confusion_matrix(y_val, pred_1))
print('-'*60)
print(classification_report(y_val, pred_1))
```

```
0.979
------------------------------------------------------------
[[ 969    0    3    2    0    4    2    0    0    0]
 [    0 1128    1    2    1    0    1    2    0    0]
 [    3    6  997    6    1    0    2   14    2    1]
 [    0    0    2  988    0   14    0    3    3    0]
 [    0    1    0    0  974    0    1    2    0    4]
 [    2    0    0    4    0  884    2    0    0    0]
 [    7    3    0    0    5    6  935    0    2    0]
 [    0    3    6    3    0    0    0 1014    2    0]
 [    5    1    6    1    2    6    1    8  938    6]
 [    3    5    0    6   10    7    0   14    1  963]]
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.98      0.99      0.98       980
           1       0.98      0.99      0.99      1135
           2       0.98      0.97      0.97      1032
           3       0.98      0.98      0.98      1010
           4       0.98      0.99      0.99       982
           5       0.96      0.99      0.98       892
           6       0.99      0.98      0.98       958
           7       0.96      0.99      0.97      1028
           8       0.99      0.96      0.98       974
           9       0.99      0.95      0.97      1009
```

```
    accuracy                           0.98     10000
   macro avg       0.98      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000
```

5.실습

(1) 모델1

- 기본 모델링에서 다음을 조정해 봅시다.
    - Flatten 이후 Dense 레이어 추가(노드수 128)

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 16)	160
max_pooling2d	(None, 14, 14, 16)	0
flatten (Flatten)	(None, 3136)	0
dense (Dense)	**(None, 128)**	401536
dense_1 (Dense)	(None, 10)	1290

```
clear_session()

model2 = Sequential([Conv2D(16, kernel_size = 3, input_shape=(28, 28, 1), # kernel_size 3x3 크기의 지역에서 특징 도출
                           padding='same', activation='relu'),  # strides = 1(기본값,1)
                    MaxPooling2D(pool_size = 2 ),          # 중요한 것만 요약    # strides = 2(기본값이 pool_size 동일)
                    Flatten(),                               # 1차원으로 펼침
                     Dense(128, activation='relu'),
                    Dense(10, activation='softmax')
])

model2.summary()

model2.compile(optimizer=Adam(learning_rate=0.001), loss='sparse_categorical_crossentropy')
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 16)        160

 max_pooling2d (MaxPooling2  (None, 14, 14, 16)        0
 D)

 flatten (Flatten)           (None, 3136)              0

 dense (Dense)               (None, 128)               401536

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 402986 (1.54 MB)
Trainable params: 402986 (1.54 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

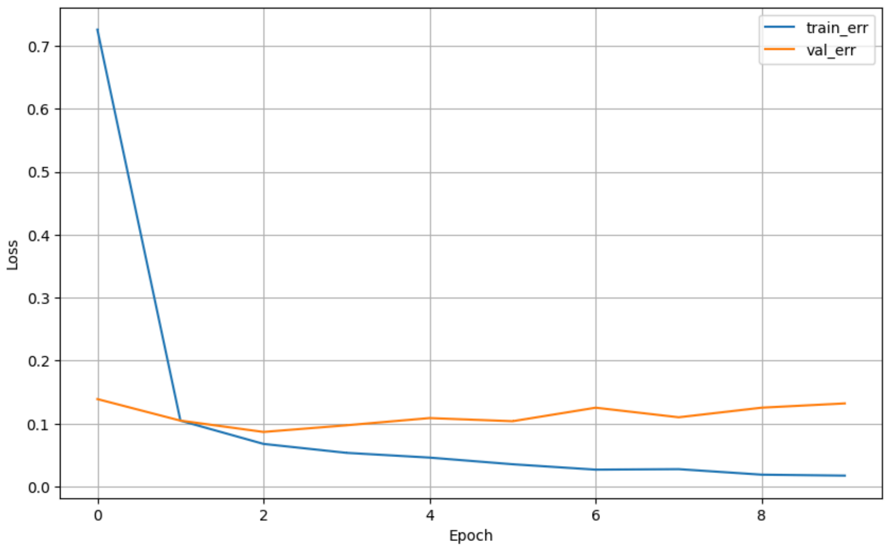
```
hist = model2.fit(x_train, y_train, epochs=10, validation_split=.2).history
```

```
Epoch 1/10
1500/1500 [==============================] - 9s 4ms/step - loss: 0.7259 - val_loss: 0.1390
Epoch 2/10
1500/1500 [==============================] - 5s 3ms/step - loss: 0.1047 - val_loss: 0.1048
Epoch 3/10
1500/1500 [==============================] - 6s 4ms/step - loss: 0.0678 - val_loss: 0.0868
Epoch 4/10
1500/1500 [==============================] - 5s 3ms/step - loss: 0.0536 - val_loss: 0.0973
Epoch 5/10
1500/1500 [==============================] - 6s 4ms/step - loss: 0.0460 - val_loss: 0.1088
Epoch 6/10
1500/1500 [==============================] - 9s 6ms/step - loss: 0.0354 - val_loss: 0.1038
Epoch 7/10
1500/1500 [==============================] - 11s 7ms/step - loss: 0.0269 - val_loss: 0.1253
Epoch 8/10
1500/1500 [==============================] - 10s 7ms/step - loss: 0.0276 - val_loss: 0.1101
```

```
Epoch 9/10
1500/1500 [==============================] - 8s 6ms/step - loss: 0.0189 - val_loss: 0.1253
Epoch 10/10
1500/1500 [==============================] - 9s 6ms/step - loss: 0.0174 - val_loss: 0.1321
```

- 학습결과 그래프

```
dl_history_plot(hist)
```



- 예측 및 평가

```
pred = model2.predict(x_val)
pred = np.argmax(pred, axis=1)
```

```
313/313 [==============================] - 2s 5ms/step
```

```
print(accuracy_score(y_val,pred))
print('-'*60)
print(confusion_matrix(y_val, pred))
print('-'*60)
print(classification_report(y_val, pred))
```

```
0.9762
------------------------------------------------------------
[[ 968    1    2    1    0    0    4    1    3    0]
 [   0 1119    5    2    2    1    0    3    3    0]
 [   1    0 1016    1    1    0    0    7    6    0]
 [   0    0    2  991    0    8    0    2    4    3]
 [   1    0    1    1  960    0    8    2    0    9]
 [   2    0    0   19    0  859    2    0    8    2]
 [   4    2    1    1    1    1  944    0    4    0]
 [   0    1    6    2    2    0    1 1009    4    3]
 [   5    0    4    2    9    6    3    3  938    4]
 [   3    4    5    5   12    3    0   10    9  958]]
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       0.99      0.99      0.99      1135
           2       0.98      0.98      0.98      1032
           3       0.97      0.98      0.97      1010
           4       0.97      0.98      0.98       982
           5       0.98      0.96      0.97       892
           6       0.98      0.99      0.98       958
           7       0.97      0.98      0.98      1028
           8       0.96      0.96      0.96       974
           9       0.98      0.95      0.96      1009
```

```
    accuracy                           0.98     10000
   macro avg       0.98      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000
```

코딩을 시작하거나 AI로 코드를 생성하세요.

(2) 모델2

- 모델1에 이어서 다음을 조정해 봅시다.
  - Convnet의 커널 수를 32로 늘려 봅시다.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, **32**)	320
max_pooling2d	(None, 14, 14, 32)	0
flatten (Flatten)	(None, 6272)	0
dense (Dense)	(None, 128)	802944
dense_1 (Dense)	(None, 10)	1290

```
clear_session()


model2 = Sequential([Conv2D(32, kernel_size = 3, input_shape=(28, 28, 1),  # 필터 정류를 16개, 16개 합성곱 # kernel_size 3x3 크기
                                                                     # 28x28 크기에서 1step씩
                           padding='same', activation='relu'),  # padding: input_shape와 같은 크기로 나오게(same) # strides = 1(기본값,1),
                                                                                                         # 가로,세로 1칸씩 이동
                    MaxPooling2D(pool_size = 2 ),            # 중요한 것만 요약  # strides = 2(기본값이 pool_size 동일)
                    Flatten(),                          # 1차원으로 펼침
                     Dense(128, activation='relu'),
                    Dense(10, activation='softmax')
])

model2.summary()

model2.compile(optimizer=Adam(learning_rate=0.001), loss='sparse_categorical_crossentropy')
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 32)        320

 max_pooling2d (MaxPooling2  (None, 14, 14, 32)        0
 D)

 flatten (Flatten)           (None, 6272)              0

 dense (Dense)               (None, 128)               802944

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 804554 (3.07 MB)
Trainable params: 804554 (3.07 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

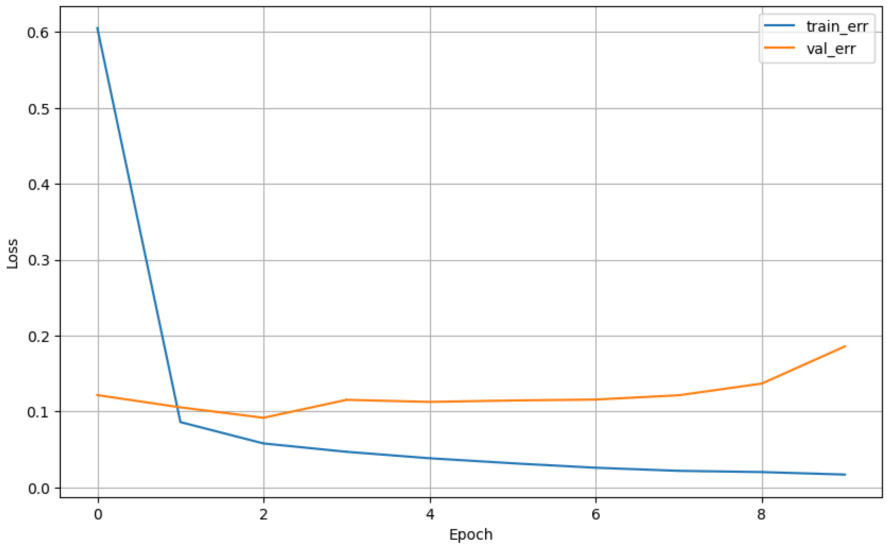
```
hist = model2.fit(x_train, y_train, epochs=10, validation_split=.2).history
```

```
Epoch 1/10
1500/1500 [==============================] - 12s 7ms/step - loss: 0.6052 - val_loss: 0.1216
Epoch 2/10
1500/1500 [==============================] - 9s 6ms/step - loss: 0.0861 - val_loss: 0.1055
Epoch 3/10
1500/1500 [==============================] - 9s 6ms/step - loss: 0.0579 - val_loss: 0.0917
Epoch 4/10
1500/1500 [==============================] - 9s 6ms/step - loss: 0.0469 - val_loss: 0.1154
Epoch 5/10
1500/1500 [==============================] - 13s 8ms/step - loss: 0.0384 - val_loss: 0.1127
Epoch 6/10
1500/1500 [==============================] - 9s 6ms/step - loss: 0.0318 - val_loss: 0.1146
Epoch 7/10
1500/1500 [==============================] - 9s 6ms/step - loss: 0.0259 - val_loss: 0.1158
Epoch 8/10
```

```
1500/1500 [==============================] - 8s 5ms/step - loss: 0.0218 - val_loss: 0.1214
Epoch 9/10
1500/1500 [==============================] - 9s 6ms/step - loss: 0.0202 - val_loss: 0.1368
Epoch 10/10
1500/1500 [==============================] - 11s 7ms/step - loss: 0.0169 - val_loss: 0.1858
```

- 학습결과 그래프

```
dl_history_plot(hist)
```



- 예측 및 평가

```
pred = model2.predict(x_val)
pred = np.argmax(pred, axis=1)
```

```
313/313 [==============================] - 1s 3ms/step
```

```
print(accuracy_score(y_val,pred))
print('-'*60)
print(confusion_matrix(y_val, pred))
print('-'*60)
print(classification_report(y_val, pred))
```

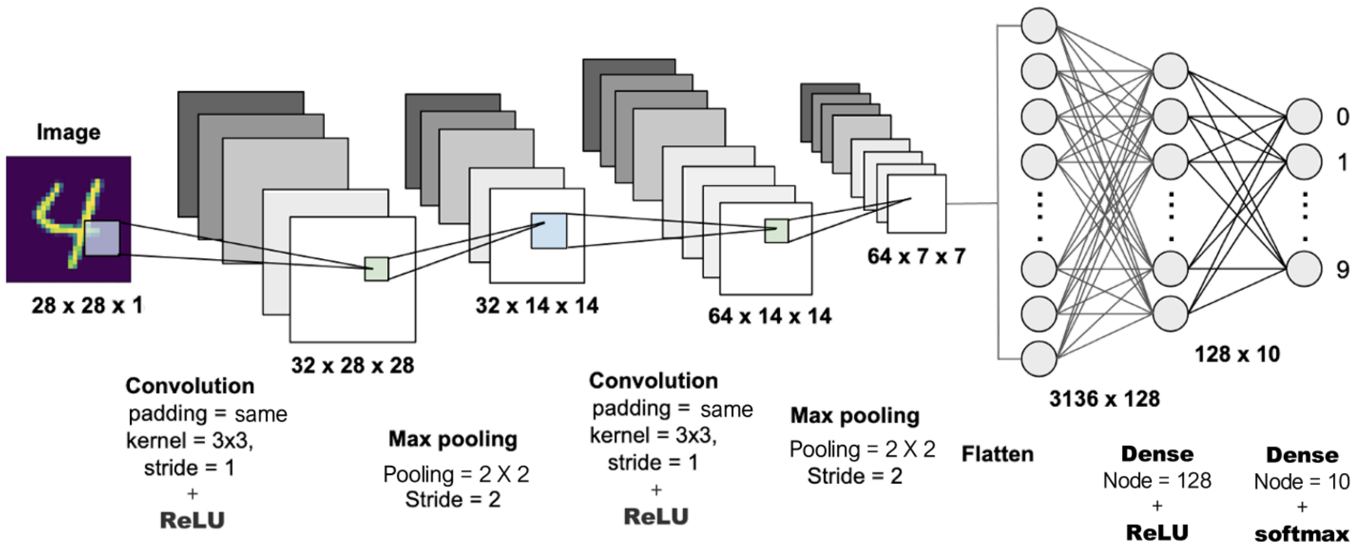
```
0.9768
------------------------------------------------------------
[[ 970    1    2    0    0    0    3    1    3    0]
 [   0 1127    4    0    2    0    1    0    1    0]
 [   0    0 1017    2    1    0    0   12    0    0]
 [   0    0    2  998    0    4    0    4    1    1]
 [   0    0    1    0  973    0    3    2    0    3]
 [   0    0    1   23    0  854    2    1    9    2]
 [   0    2    1    2    6    2  943    0    2    0]
 [   0    3   11    4    5    0    0 1000    1    4]
 [   3    3   15    5    3    1    1    7  930    6]
 [   1    3    3    7   18    9    0    7    5  956]]
------------------------------------------------------------
              precision    recall  f1-score   support

           0       1.00      0.99      0.99       980
           1       0.99      0.99      0.99      1135
           2       0.96      0.99      0.97      1032
           3       0.96      0.99      0.97      1010
           4       0.97      0.99      0.98       982
           5       0.98      0.96      0.97       892
           6       0.99      0.98      0.99       958
           7       0.97      0.97      0.97      1028
           8       0.98      0.95      0.97       974
```

```
           9       0.98      0.95      0.97      1009

    accuracy                           0.98     10000
   macro avg       0.98      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000
```

코딩을 시작하거나 AI로 코드를 생성하세요.

(3) 모델3

- 모델2에 이어서, 아래 그림을 보고, 빠진 부분을 추가하시오.




```
clear_session()


model3 = Sequential([Conv2D(32, kernel_size = 3, input_shape=(28, 28, 1),  # 필터 정류를 16개, 16개 합성곱 # kernel_size 3x3 크기
                                                                    ...            # 28x28 크기에서 1step씩
                           padding='same', activation='relu'),  # padding: input_shape와 같은 크기로 나오게(same) # strides = 1(기본값,1),
                                                                    ...                                      # 가로,세로 1칸씩 이동
                    MaxPooling2D(pool_size = 2 ),            # 중요한 것만 요약  # strides = 2(기본값이 pool_size 동일)

                    Conv2D(64, kernel_size = 3, input_shape=(14, 14, 1),  padding='same', activation='relu'),
                    MaxPooling2D(pool_size = 2 ),

                    Flatten(),                          # 1차원으로 펼침
                    Dense(128, activation='relu'),
                    Dense(10, activation='softmax')
])

model3.summary()

model3.compile(optimizer=Adam(learning_rate=0.001), loss='sparse_categorical_crossentropy')
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 32)        320

 max_pooling2d (MaxPooling2  (None, 14, 14, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 14, 14, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 7, 7, 64)          0
 g2D)

 flatten (Flatten)           (None, 3136)              0

 dense (Dense)               (None, 128)               401536

 dense_1 (Dense)             (None, 10)                1290
```

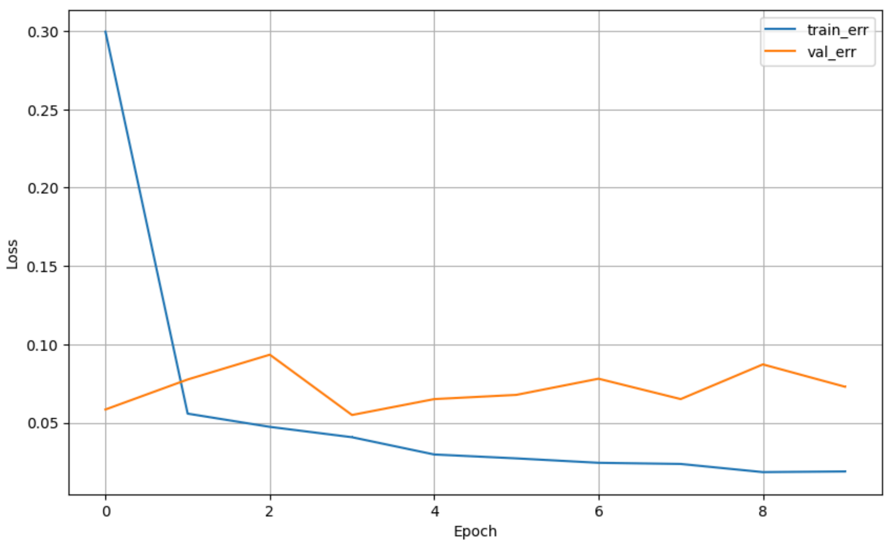
```
=================================================================
Total params: 421642 (1.61 MB)
Trainable params: 421642 (1.61 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
hist = model3.fit(x_train, y_train, epochs=10, validation_split=.2).history
```

```
Epoch 1/10
1500/1500 [==============================] - 12s 6ms/step - loss: 0.2995 - val_loss: 0.0585
Epoch 2/10
1500/1500 [==============================] - 9s 6ms/step - loss: 0.0558 - val_loss: 0.0776
Epoch 3/10
1500/1500 [==============================] - 9s 6ms/step - loss: 0.0474 - val_loss: 0.0934
Epoch 4/10
1500/1500 [==============================] - 9s 6ms/step - loss: 0.0408 - val_loss: 0.0549
Epoch 5/10
1500/1500 [==============================] - 9s 6ms/step - loss: 0.0298 - val_loss: 0.0651
Epoch 6/10
1500/1500 [==============================] - 10s 6ms/step - loss: 0.0273 - val_loss: 0.0678
Epoch 7/10
1500/1500 [==============================] - 9s 6ms/step - loss: 0.0244 - val_loss: 0.0781
Epoch 8/10
1500/1500 [==============================] - 9s 6ms/step - loss: 0.0237 - val_loss: 0.0651
Epoch 9/10
1500/1500 [==============================] - 9s 6ms/step - loss: 0.0185 - val_loss: 0.0872
Epoch 10/10
1500/1500 [==============================] - 9s 6ms/step - loss: 0.0189 - val_loss: 0.0730
```

- 학습결과 그래프

```
dl_history_plot(hist)
```



- 예측 및 평가

```
pred = model3.predict(x_val)
pred = np.argmax(pred, axis=1)
```

```
313/313 [==============================] - 1s 3ms/step
```

```
print(accuracy_score(y_val,pred))
print('-'*60)
print(confusion_matrix(y_val, pred))
print('-'*60)
print(classification_report(y_val, pred))
```

```
0.9865
------------------------------------------------------------
[[ 979    0    0    0    0    0    1    0    0    0]
 [   6 1120    0    1    0    3    1    4    0    0]
 [   2    2 1010    3    1    0    1   10    3    0]
 [   1    0    1  997    0   10    0    0    1    0]
 [   0    0    0    0  953    2    2    2    2   21]
 [   1    0    0    4    0  884    0    0    2    1]
 [  10    2    0    0    2    5  936    0    3    0]
 [   0    1    3    1    2    0    0 1021    0    0]
 [   0    1    0    1    0    1    0    0  967    4]
 [   1    0    1    0    2    2    0    3    2  998]]
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       980
           1       0.99      0.99      0.99      1135
           2       1.00      0.98      0.99      1032
           3       0.99      0.99      0.99      1010
           4       0.99      0.97      0.98       982
           5       0.97      0.99      0.98       892
           6       0.99      0.98      0.99       958
           7       0.98      0.99      0.99      1028
           8       0.99      0.99      0.99       974
           9       0.97      0.99      0.98      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000
```

코딩을 시작하거나 AI로 코드를 생성하세요.

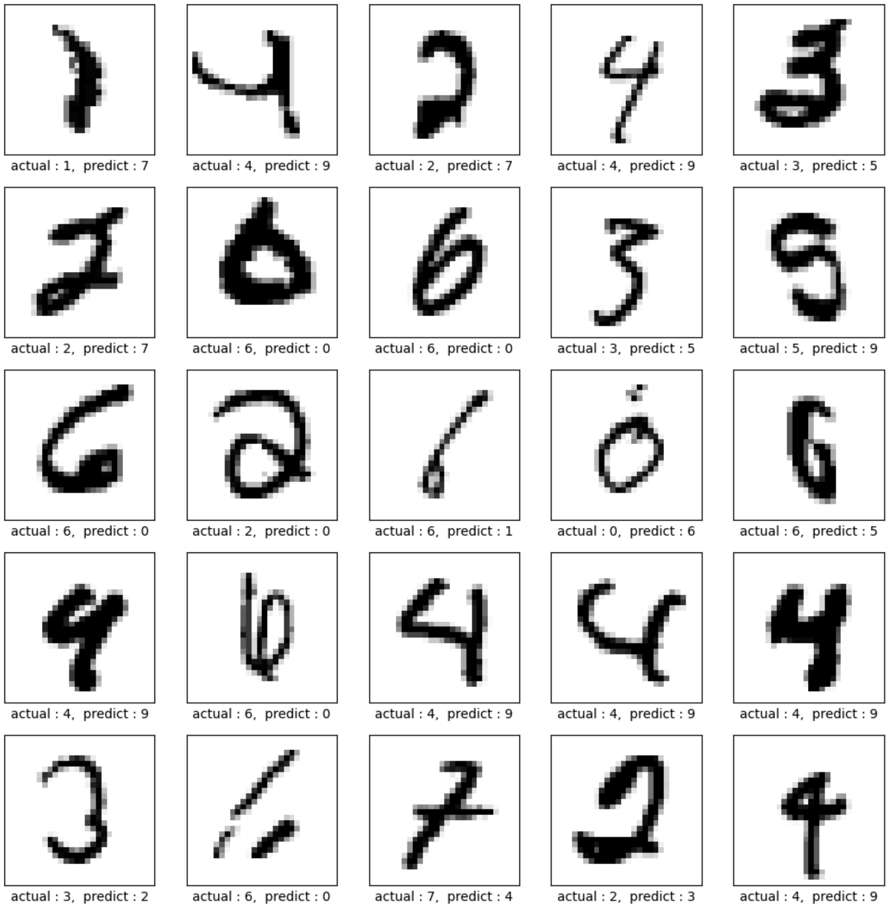
6.틀린그림 찾아보기

- 모델3의 결과에서 틀린 그림을 살펴 봅시다.
- 아래코드는 이해하기보다는 그냥 사용하기 바랍니다.

```
idx = (y_val != pred)
x_val_wr = x_val[idx]
y_val_wr = y_val[idx]
pred_wr = pred[idx]

x_val_wr = x_val_wr.reshape(-1,28,28)
print(x_val_wr.shape)
```

```
(135, 28, 28)
```

```
idx = rd.sample(range(x_val_wr.shape[0]),25)
x_temp = x_val_wr[idx]
y_temp = y_val_wr[idx]
p_temp = pred_wr[idx]

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(x_temp[i], cmap=plt.cm.binary)
    plt.xlabel(f'actual : {y_temp[i]},  predict : {p_temp[i]}')
plt.tight_layout()
plt.show()
```



7.진짜 손글씨로 예측해 봅시다.

- 이미지 처리를 위한 라이브러리와 함수 불러오기

```
import cv2
from google.colab.patches import cv2_imshow
```

- 그림판에서 그린 손글씨를 업로드 합니다.

```
# 파일 열기
img = cv2.imread('33.png', cv2.IMREAD_GRAYSCALE)
cv2_imshow(img)
print(img.shape)
```

```
(140, 140)
```

- 이미지 크기를 28, 28, 1 로 맞추기

```
# 크기 조절하기
img = cv2.resize(255-img, (28, 28))
print(img.shape)
cv2_imshow(img)
```

```
(28, 28)
```

- 예측하기

```
# 입력데이터 형식을 갖추기
test_num = img.reshape(1,28,28,1)

# 예측하기
```