Mnist CNN

```python
import keras
from keras.datasets import mnist
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Dense , Activation , Flatten , Conv2D, MaxPooling2D

import numpy as np
import matplotlib.pyplot as plt

```

mnist : mnist dataset 불러오는 module

np_utils : data 정리를 위한 module

sequential : CNN의 layer들을 순차적으로 실행시키기 위한 module

Dense, Activation, Flatten, Conv2D, MaxPooling2D : CNN 의 layer에 들어가는 module 들

import numpy as np

: numpy를 import 하여 np라는 이름으로 사용.

numpy : 난수 생성, 행렬 / 배열을 처리하기 위한 package

import matplotlib.pyplot as plt

: matplotlib.pyplot을 import 해 plt로 사용

matplotlib.pyplot : 그림이나 그래프 등을 시각적으로 표현하기 위한 package

```python
img_rows = 28  (열)
img_cols = 28  (행)

num_classes = 10  (label 개수 (0 ~ 9))

(x_train , y_train),(x_test , y_test ) = mnist.load_data()

input_shape = (img_rows , img_cols , 1)
x_train = x_train.reshape(x_train.shape[0],img_rows,img_cols,1).astype('float32')/255.0
x_test = x_test.reshape(x_test.shape[0],img_rows,img_cols,1).astype('float32')/255.0

y_train = np_utils.to_categorical(y_train,num_classes)
y_test = np_utils.to_categorical(y_test,num_classes)
```

mnist.load_data() : x_train, y_train, x_test, y_test 네 개의 numpy array를 가져옴.

train sample = 60000 개 , test sample = 10000 개

x_train, x_test : 28 x 28 픽셀의 각 손글씨 이미지 데이터

y_train, y_test : 0 ~ 9 사이의 label

input_shape = (img_rows, img_cols, 1) = (28, 28, 1)

: 크기는 28 X 28 이고 channel = 1 즉, 흑백을 뜻한다.

x_train.reshape(x_train.shape[0], img_rows, img_cols, 1).astype('float32')/255.0

: x_train을 3차원에서 4차원으로 reshape.

→ 실수화 → 0 ~ 1의 값으로 정규화.

실수화 정규화

y_train = np_utils.to_categorical(y_train, num_classes)

: to_categorical() → One_hot 인코딩을 해주는 함수

: 10진 정수 형식을 특수한 2진 binary 형식으로 변경

ex) y = 2 $\xrightarrow[\text{to\_categorical()}]{}$ 0 0 1 0 0 0 0 0 0 0

```python
model = Sequential()
model.add(Conv2D(32,kernel_size = (5,5),
                 activation = 'relu',
                 input_shape = (28,28,1)))
model.add(MaxPooling2D(pool_size =(2,2),
                       strides=(2,2)))
```

model = Sequential()

: Sequential 모델 object를 model 이라는 변수 안에 넣고, 모델 구성을 함.

Conv2D (32. kernel_size = (5,5), activation = 'relu', input_shape = (28.28.1)))

: 32 → 뉴런 개수

(5,5) → kernel 크기 (행,열)

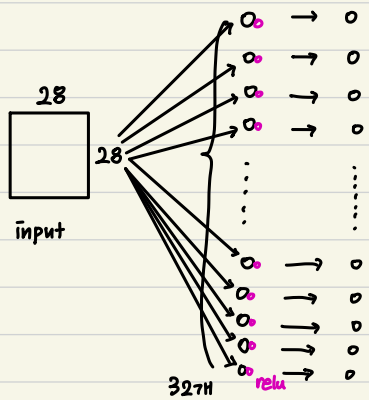
relu → 활성화 함수

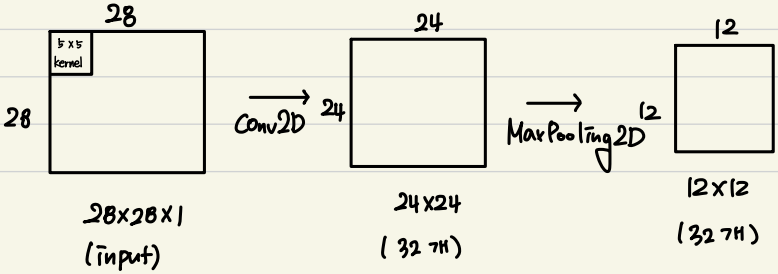
28.28.1 → input_shape, 크기 : 28x28, channel : 1

MaxPooling2D (pool_size = (2,2), strides = (2,2)))

: (2,2) → pool_size = 윈도우 (patch)의 크기 지정

(2,2) → strides = 움직이는 거리





```
model.add(Conv2D(64,kernel_size = (5,5),
                 activation ='relu',
                 padding = 'same'))
model.add(MaxPooling2D(pool_size = (2,2)))
```

Conv2D (64, kernel_size = (5.5), activation ='relu', padding = 'same'))

: 64 → 뉴런 개수

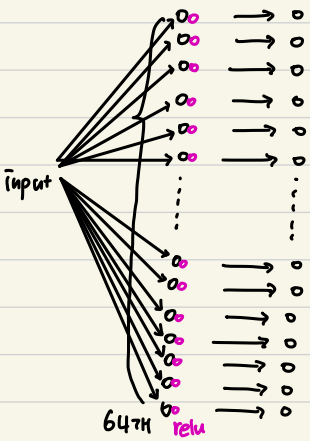
(5.5) → kernel 크기 (행, 열)

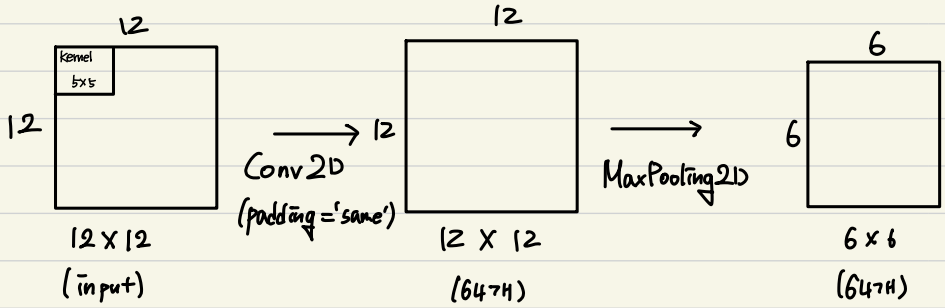
relu → 활성화 함수

'same' → padding = 'same' 즉, input에 zero padding을 더하여 input size와 output size를 같게해줌

MaxPooling2D (pool-size = (2,2)))

: (2.2) → 윈도의 크기





```
model.add(Conv2D(128,kernel_size = (5,5),
                 activation ='relu',
                 padding = 'same'))
model.add(MaxPooling2D(pool_size = (2,2)))
```

Conv2D (128, kernel_size = (5.5), activation ='relu', padding = 'same'))

: 128 → 뉴런 개수

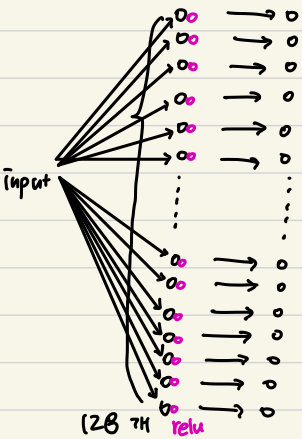
(5.5) → kernel 크기 (행, 열)

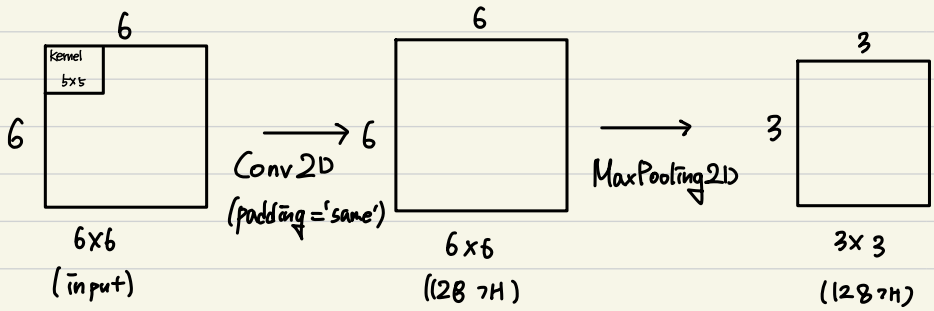
relu → 활성화 함수

'same' → padding = 'same' 즉, input에 zero padding을 더하여 input size와 output size를 같게해줌

MaxPooling2D (pool_size = (2,2)))

: (2.2) → 윈도의 크기





```
model.add(Flatten())
model.add(Dense(512,activation = 'relu'))
model.add(Dense(10,activation = 'softmax'))
model.summary()
```

Flatten()

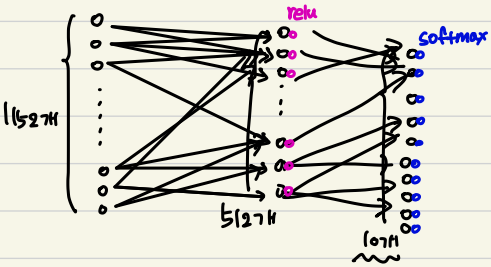
: 128개의 output 펼치기 → 128 x 3x3 = 1152

Dense (512, activation = 'relu')

: 512 → 출력 뉴런 수

Dense (10, activation = 'softmax')

: 10 → 출력 수



↳ label 개수를 맞춰줌

model.summary()

: 모델 구조 확인

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 24, 24, 32)	832
max_pooling2d_4 (MaxPooling 2D)	(None, 12, 12, 32)	0
conv2d_5 (Conv2D)	(None, 12, 12, 64)	51264
max_pooling2d_5 (MaxPooling 2D)	(None, 6, 6, 64)	0
conv2d_6 (Conv2D)	(None, 6, 6, 128)	204928
max_pooling2d_6 (MaxPooling 2D)	(None, 3, 3, 128)	0
flatten_2 (Flatten)	(None, 1152)	0
dense_3 (Dense)	(None, 512)	590336
dense_4 (Dense)	(None, 10)	5130

Total params: 852,490
Trainable params: 852,490
Non-trainable params: 0

```
model.compile(loss = 'categorical_crossentropy',
              optimizer ='sgd', metrics = ['accuracy'])
hist = model.fit(x_train , y_train , epochs = 10 , batch_size = 64)
```

모델 학습 과정 설정

10번 반복

60,000 개 중
64 개 씩

Categorical_crossentropy

: 손실 확인 ⇒ -(실제 x log (예측))

ex) 실제 : 0 0 1 0
예측 : 0 1 0 0

⇒ loss = $-(0 \times \log 0 + 0 \times \log 1 + 1 \times \log 0 + 0 \times \log 0)$
$= \infty$

실제 : 0 0 1 0
예측 : 0 0 1 0

⇒ loss = $-(0 \times \log 0 + 0 \times \log 0 + 1 \times \log 1 + 0 \times \log 0)$
$= 0$

optimizer = 'sgd'

: sgd = Stochastic Gradient Descent (확률 경사 하강법)

→ 하나의 데이터 당 gradient를 구해 업데이트 과정 진행, 데이터가 n 개 라면 n번의 업데이트.

metrics

: 성능 지표. 모니터링 할 지표

model.fit (x_train, y_train, epochs = 10, batch_size = 64)

: 입력 데이터 = x_train
결과 데이터 = y_train
epochs = 10 (학습 데이터 반복 횟수)
batch_size = 64 (한 번 학습할 때 사용하는 데이터 개수)

```python
fig = plt.figure(facecolor = 'white',figsize = (5,3))
loss_ax = fig.add_subplot(111)

loss_ax.plot(hist.history['loss'],'b',label = 'train_loss')
loss_ax.plot(hist.history['accuracy'],'r',label = 'train_accuracy')

loss_ax.set_xlabel('epochs')
loss_ax.legend(loc='center_right')
plt.xlim([0,9])
plt.ylim([0,1])
plt.show()
```

plt.figure(facecolor = 'white', figsize = (5,3))

: facecolor → 배경색

figsize → 크기

⇒

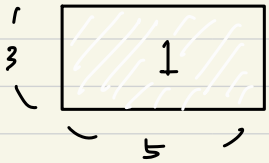


fig.add_subplot(111)

: 1x1 그리드, 첫 번째 서브 플롯

set_xlabel('epochs')

: x축은 epochs

legend(loc='center_right')

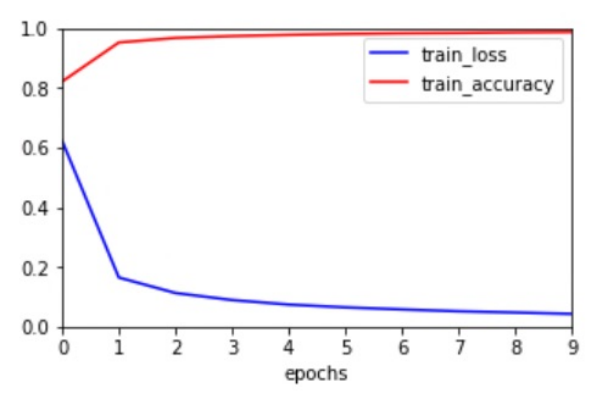
: 범주박스 위치 설정

xlim([0,9])

: x축 범위 0~9

ylim([0,1])

: y축 범위 0~1



```
loss_and_metrics = model.evaluate(x_test,y_test,batch_size = 32)
print('evaluation loss and matrics ##')
print(loss_and_metrics)
```

⇒ 학습에서 얻은 모델을 test 데이터로 평가 (loss 와 accuracy 반환)

x_test . 테스트 데이터

y_test : 테스트 타겟

```
yhat = model.predict(x_test,batch_size = 64)
yhat
```

yhat : x_test에 대한 예측값

⇓

```
array([[1.5676379e-07, 1.1797912e-06, 1.2914016e-05, ..., 9.9990082e-01,
        4.0290789e-07, 4.0810050e-06],
       [2.3864206e-06, 5.0915529e-05, 9.9994564e-01, ..., 1.2417505e-10,
        1.8020704e-07, 4.1186557e-14],
       [1.0451543e-05, 9.9962485e-01, 2.2445070e-05, ..., 1.0723119e-04,
        4.9433598e-05, 3.1653601e-06],
       ...,
       [1.2648989e-10, 1.1861827e-08, 1.0953659e-10, ..., 1.8568605e-06,
        5.6851019e-07, 6.5481850e-06],
       [2.2934555e-07, 8.0020640e-10, 5.6209598e-10, ..., 3.5500769e-09,
        1.6131715e-04, 9.0221599e-09],
       [1.3208206e-06, 7.8799349e-11, 6.3754634e-08, ..., 4.1506718e-13,
        1.4301369e-07, 1.4661121e-11]], dtype=float32)
```

```python
plt_row = 5
plt_col = 5

plt.rcParams["figure.figsize"] = (10,10)
f, axarr = plt.subplots(plt_row , plt_col)
cnt = 0
i = 0
while cnt < (plt_row * plt_col):#25개 까지
  if np.argmax(y_test[i]) != np.argmax(yhat[i]):  # 실제값과 예측값이 다르면 넘어감.
    i += 1
    continue
  sub_plt = axarr[cnt//plt_row , cnt%plt_col]
  sub_plt.axis('off')
  sub_plt.imshow(x_test[i].reshape(img_rows,img_cols))
  sub_plt_title = 'R:' + str(np.argmax(y_test[i])) + 'P: ' + str(np.argmax(yhat[i]))
  sub_plt.set_title(sub_plt_title)
  i += 1
  cnt += 1

plt.show()
```

실제값 예측값