2.1 Keras 학습 분석

이번 실습에서는 모델이 학습하는 동안 변화하는 학습양상을 확인하는 방법을 알아본다

(1) CallBack

(2) History 확인

(3) 틀린 샘플 확인

(4) Confusion Matrix

In [1]:
```
import tensorflow.keras.utils as utils
from tensorflow.keras import datasets
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Input, Dense, Activation

import os
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
(X_train, Y_train),(X_test, Y_test) = datasets.mnist.load_data()
X_train_flat = X_train.reshape(60000, 28*28).astype('float32')/255.0
X_test_flat = X_test.reshape(10000, 28*28).astype('float32')/255.0
Y_train_onehot = utils.to_categorical(Y_train)
Y_test_onehot = utils.to_categorical(Y_test)

n_in = 28*28
n_out = np.shape(Y_test_onehot)[1]
```

In [3]:
```
def DNN_seq(n_in, n_out):
    # Coding Time (5 min) layer : 784 → 128 → 32 → 10, acitvation : relu → relu → softmax
    model = Sequential()
    model.add(Dense(units =128, input_shape=(n_in,), activation='relu'))
    model.add(Dense(units =32, activation='relu'))
    model.add(Dense(units =n_out, activation='softmax'))
    return model

model = DNN_seq(n_in, n_out)
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

(1) Keras Callback

model의 fit() 함수로 학습을 진행하는 동안, 매 epoch마다 지정한 함수를 호출할 수 있음

ModelCheckpoint : 학습 중 모델 저장

EarlyStopping : 학습양상을 보고 학습을 조기에 종료

TensorBoard : tensorboard로 확인할 수 있도록 학습양상을 기록

외에 kereas.callbacks.Callback을 상속받아 원하는 callback 함수를 만들 수 있음

https://keras.io/callbacks/

In [4]:
```
# 학습과정을 저장할 directory 생성
import os
import datetime

def make_dir(path):
    today = str(datetime.date.today())
    path_date = path+'/'+today

    if not os.path.exists(path_date):
        os.makedirs(path_date)
    return path_date
```

In [5]:
```
model_path=make_dir('./model')
tensorboard_path=make_dir('./tensorboard')

modelconfig = str(n_in)+'_'+str(n_out)
model_name_path = model_path+'/'+modelconfig+"_{epoch:02d}-{loss:.4f}_{val_loss:.4f}_{val_accuracy:.4f}.h5"

print(model_name_path)
print(tensorboard_path)
```

```
./model/2022-12-05/784_10_{epoch:02d}-{loss:.4f}_{val_loss:.4f}_{val_accuracy:.4f}.h5
./tensorboard/2022-12-05
```

필요한 callback 함수들을 정의하고 list로 fit() 함수에 전달

In [6]:
```
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, TensorBoard

# Coding Time
# Callback list (checkpointer, earlystopper, tb_saver)
checkpointer = ModelCheckpoint(filepath=model_name_path,
                               monitor='val_accuracy',
                               verbose=0,
                               save_best_only=True)
```

```
earlystopper = EarlyStopping(monitor='val_accuracy',
                             patience=5,
                             verbose=0,
                             mode='auto')
tb_saver = TensorBoard(log_dir=tensorboard_path,
                       write_graph=True)
callback_list=[checkpointer, tb_saver, earlystopper]

# Train (batch : 256, epochs : 50, validation_split : 0.3, verbose : 2, including callback list)
history = model.fit(X_train_flat, Y_train_onehot, batch_size=256, epochs=50, validation_split=0.3, verbose=2, callbacks = callback_
```

```
Epoch 1/50
165/165 - 1s - loss: 0.5591 - accuracy: 0.8434 - val_loss: 0.2536 - val_accuracy: 0.9283 - 1s/epoch - 9ms/step
Epoch 2/50
165/165 - 1s - loss: 0.2084 - accuracy: 0.9408 - val_loss: 0.1835 - val_accuracy: 0.9477 - 517ms/epoch - 3ms/step
Epoch 3/50
165/165 - 0s - loss: 0.1481 - accuracy: 0.9575 - val_loss: 0.1519 - val_accuracy: 0.9563 - 482ms/epoch - 3ms/step
Epoch 4/50
165/165 - 1s - loss: 0.1172 - accuracy: 0.9660 - val_loss: 0.1347 - val_accuracy: 0.9610 - 528ms/epoch - 3ms/step
Epoch 5/50
165/165 - 0s - loss: 0.0931 - accuracy: 0.9732 - val_loss: 0.1247 - val_accuracy: 0.9639 - 486ms/epoch - 3ms/step
Epoch 6/50
165/165 - 0s - loss: 0.0776 - accuracy: 0.9775 - val_loss: 0.1224 - val_accuracy: 0.9631 - 496ms/epoch - 3ms/step
Epoch 7/50
165/165 - 0s - loss: 0.0644 - accuracy: 0.9810 - val_loss: 0.1104 - val_accuracy: 0.9672 - 488ms/epoch - 3ms/step
Epoch 8/50
165/165 - 0s - loss: 0.0540 - accuracy: 0.9847 - val_loss: 0.1068 - val_accuracy: 0.9694 - 496ms/epoch - 3ms/step
Epoch 9/50
165/165 - 0s - loss: 0.0469 - accuracy: 0.9869 - val_loss: 0.1023 - val_accuracy: 0.9697 - 490ms/epoch - 3ms/step
Epoch 10/50
165/165 - 1s - loss: 0.0401 - accuracy: 0.9884 - val_loss: 0.1039 - val_accuracy: 0.9692 - 510ms/epoch - 3ms/step
Epoch 11/50
165/165 - 1s - loss: 0.0358 - accuracy: 0.9904 - val_loss: 0.1073 - val_accuracy: 0.9693 - 544ms/epoch - 3ms/step
Epoch 12/50
165/165 - 1s - loss: 0.0297 - accuracy: 0.9923 - val_loss: 0.1002 - val_accuracy: 0.9716 - 572ms/epoch - 3ms/step
Epoch 13/50
165/165 - 1s - loss: 0.0247 - accuracy: 0.9940 - val_loss: 0.1063 - val_accuracy: 0.9706 - 517ms/epoch - 3ms/step
Epoch 14/50
165/165 - 1s - loss: 0.0228 - accuracy: 0.9941 - val_loss: 0.1086 - val_accuracy: 0.9709 - 542ms/epoch - 3ms/step
Epoch 15/50
165/165 - 1s - loss: 0.0188 - accuracy: 0.9955 - val_loss: 0.1090 - val_accuracy: 0.9700 - 542ms/epoch - 3ms/step
Epoch 16/50
165/165 - 1s - loss: 0.0163 - accuracy: 0.9963 - val_loss: 0.1145 - val_accuracy: 0.9694 - 574ms/epoch - 3ms/step
Epoch 17/50
165/165 - 1s - loss: 0.0142 - accuracy: 0.9968 - val_loss: 0.1094 - val_accuracy: 0.9717 - 576ms/epoch - 3ms/step
Epoch 18/50
165/165 - 1s - loss: 0.0118 - accuracy: 0.9975 - val_loss: 0.1111 - val_accuracy: 0.9718 - 562ms/epoch - 3ms/step
Epoch 19/50
165/165 - 1s - loss: 0.0100 - accuracy: 0.9983 - val_loss: 0.1093 - val_accuracy: 0.9728 - 552ms/epoch - 3ms/step
Epoch 20/50
165/165 - 1s - loss: 0.0077 - accuracy: 0.9989 - val_loss: 0.1121 - val_accuracy: 0.9728 - 544ms/epoch - 3ms/step
Epoch 21/50
165/165 - 1s - loss: 0.0067 - accuracy: 0.9990 - val_loss: 0.1225 - val_accuracy: 0.9714 - 587ms/epoch - 4ms/step
Epoch 22/50
165/165 - 1s - loss: 0.0059 - accuracy: 0.9992 - val_loss: 0.1230 - val_accuracy: 0.9714 - 585ms/epoch - 4ms/step
Epoch 23/50
165/165 - 1s - loss: 0.0053 - accuracy: 0.9994 - val_loss: 0.1172 - val_accuracy: 0.9727 - 585ms/epoch - 4ms/step
Epoch 24/50
165/165 - 1s - loss: 0.0046 - accuracy: 0.9996 - val_loss: 0.1188 - val_accuracy: 0.9724 - 594ms/epoch - 4ms/step
Epoch 25/50
165/165 - 1s - loss: 0.0037 - accuracy: 0.9997 - val_loss: 0.1216 - val_accuracy: 0.9719 - 579ms/epoch - 4ms/step
```

In [7]:
```
os.listdir(model_path)
```

Out[7]:
```
['784_10_01-0.5523_0.2591_0.9251.h5',
 '784_10_01-0.5591_0.2536_0.9283.h5',
 '784_10_02-0.2084_0.1835_0.9477.h5',
 '784_10_02-0.2182_0.2022_0.9431.h5',
 '784_10_03-0.1481_0.1519_0.9563.h5',
 '784_10_03-0.1589_0.1652_0.9509.h5',
 '784_10_04-0.1172_0.1347_0.9610.h5',
 '784_10_04-0.1248_0.1498_0.9553.h5',
 '784_10_05-0.0931_0.1247_0.9639.h5',
 '784_10_05-0.1040_0.1325_0.9611.h5',
 '784_10_06-0.0839_0.1258_0.9617.h5',
 '784_10_07-0.0644_0.1104_0.9672.h5',
 '784_10_07-0.0747_0.1176_0.9644.h5',
 '784_10_08-0.0540_0.1068_0.9694.h5',
 '784_10_08-0.0621_0.1081_0.9679.h5',
 '784_10_09-0.0469_0.1023_0.9697.h5',
 '784_10_09-0.0513_0.1081_0.9693.h5',
 '784_10_12-0.0297_0.1002_0.9716.h5',
 '784_10_12-0.0346_0.1019_0.9707.h5',
 '784_10_17-0.0142_0.1094_0.9717.h5',
 '784_10_18-0.0118_0.1111_0.9718.h5',
 '784_10_19-0.0100_0.1093_0.9728.h5',
 '784_10_20-0.0077_0.1121_0.9728.h5']
```

(2)-1 history를 통한 결과 plotting

fit() 함수는 history 객체를 반환함

history['loss']: epoch 마다 기록되는 train loss

history['accuracy'] : accuracy를 측정할 수 있는 문제이며 compile() 때 metric으로 accuracy를 지정하였다면 기록됨

history['val_loss'] : 검증 데이터가 있다면 기록되는 validation loss history['val_accuracy'] : 검증 데이터가 있고 accuracy를 측정할 수 있다면 기록되는 validation accuracy

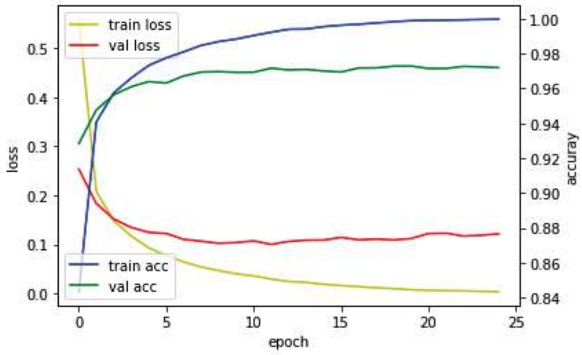
In [8]:

```
print(history.history['loss'])
print(history.history['val_accuracy'])
```

```
[0.5591161847114563, 0.2083957940340042, 0.14808885753154755, 0.11715318262577057, 0.09313711524009705, 0.0775817260146141, 0.06439150869846344, 0.05399508774280548, 0.0468536801636219, 0.040145114064216614, 0.03583592176437378, 0.029653262346982956, 0.02468772418797016, 0.022842351347208023, 0.01882307417690754, 0.016294294968247414, 0.014210488647222519, 0.011814712546765804, 0.009992938488721848, 0.007706913631409407, 0.006666335277725935, 0.005934784654527903, 0.005345095880329609, 0.004550661891698837, 0.003747525159295713]
[0.9283333420753479, 0.9476666450500488, 0.956333339214325, 0.9610000252723694, 0.9639444351196289, 0.9631111025810242, 0.9671666622161865, 0.9693889021873474, 0.9696666598320007, 0.9692222476005554, 0.9693333506584167, 0.9716110825538635, 0.9706110954284668, 0.9709444642066956, 0.9700000286102295, 0.9694444537162781, 0.9717222452163696, 0.9717777967453003, 0.972777783870697, 0.9728333353996277, 0.9714444279670715, 0.9714444279670715, 0.9726666808128357, 0.9723888635635376, 0.9719444513320923]
```

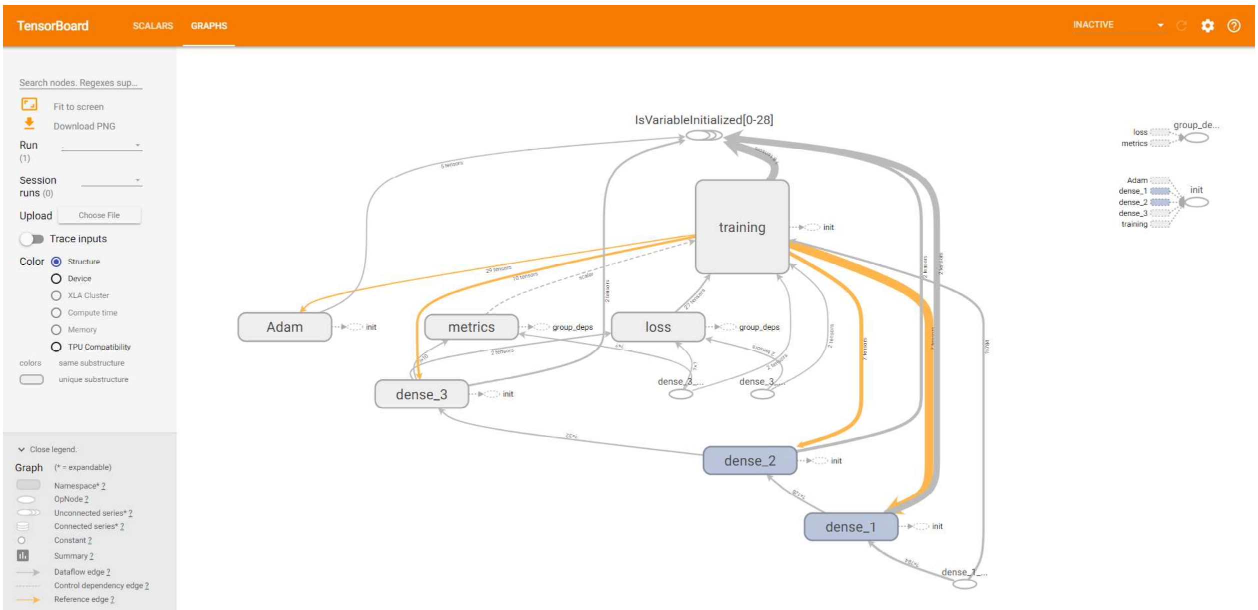
In [9]:

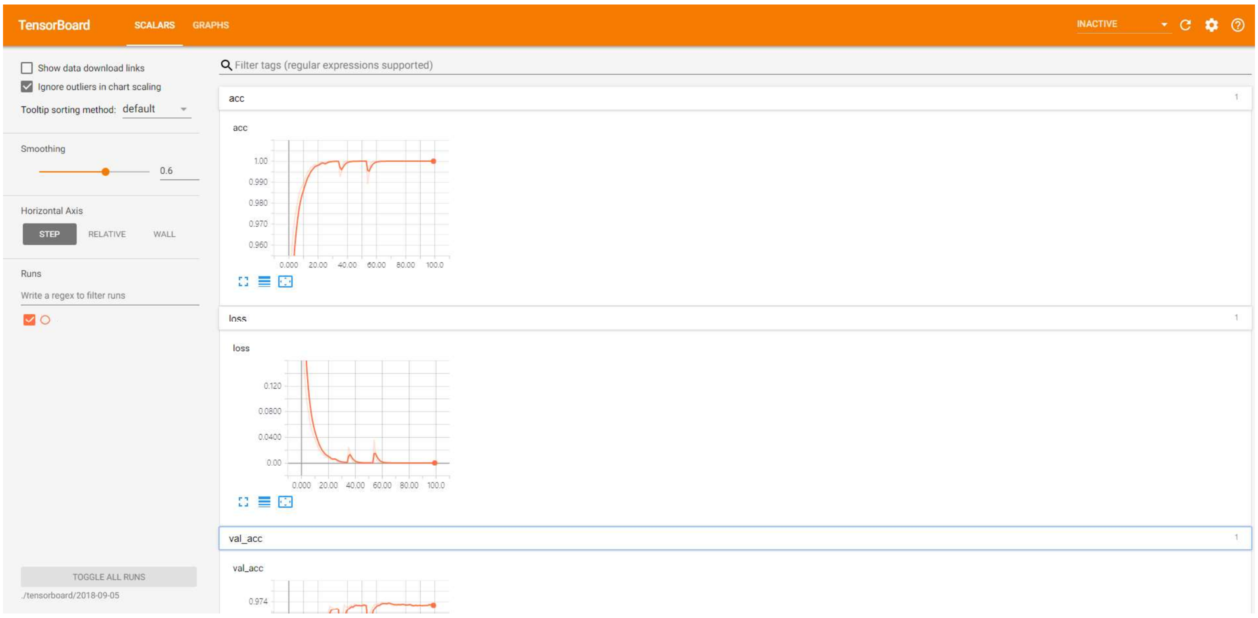
```
# matplotlib를 이용하여 history 객체 내부에 저장된 값들을 graph로 표현
fig, loss_ax = plt.subplots()
acc_ax = loss_ax.twinx()

loss_ax.plot(history.history['loss'], 'y', label='train loss')
loss_ax.plot(history.history['val_loss'], 'r', label='val loss')

acc_ax.plot(history.history['accuracy'], 'b', label='train acc')
acc_ax.plot(history.history['val_accuracy'], 'g', label='val acc')

loss_ax.set_xlabel('epoch')
loss_ax.set_ylabel('loss')
acc_ax.set_ylabel('accuray')

loss_ax.legend(loc='upper left')
acc_ax.legend(loc='lower left')

plt.show()

# 저장하고 싶을 경우
#fig.savefig('final.png')
```



(2)-2 텐서보드를 통한 결과 확인

tensorflow의 가시화 툴인 tensorboard로 학습과정을 확인 터미널에 tensorboard --logdir=./tensorboard/{날짜} 를 타이핑 후, localhost:6006으로 들어감

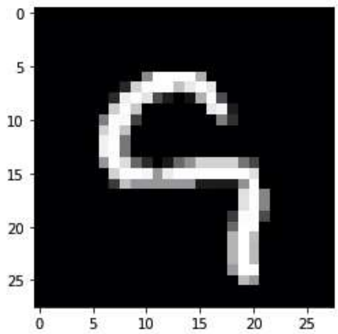




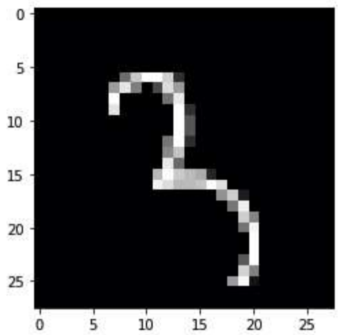
(3) 틀린샘플 찾기

In [10]:
```
def check_error(Number_Of_Error):

    # Coding Time
    cnt = 0
    for i in range(len(Y_test)):
        # 모델 예측 값 도출
        test_data = X_test[i].reshape(1,28*28)
        pred_y = model.predict(test_data, verbose=0)
        pred_y = pred_y.argmax()

        # 예측값과 라벨이 다를 경우, print와 샘플 확인
        if pred_y != Y_test[i]:
            print('real_label : {}, predict_label : {}'.format(Y_test[i], pred_y))
            plt.imshow(X_test[i], cmap='gray')
            plt.show()
            cnt +=1

        if cnt >= Number_Of_Error:
            break

check_error(5)
```

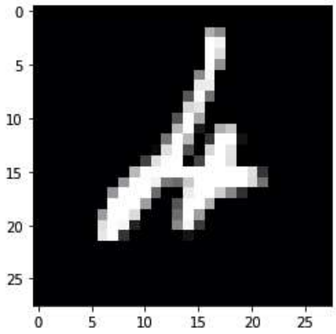
```
real_label : 9, predict_label : 5
```



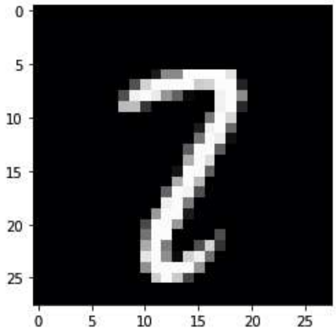

```
real_label : 3, predict_label : 8
```



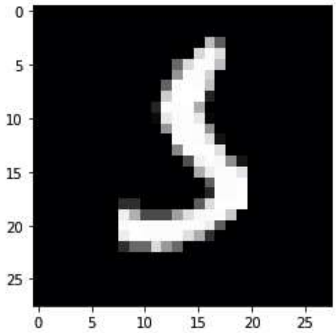

```
real_label : 4, predict_label : 2
```



```
real_label : 2, predict_label : 7
```



```
real_label : 5, predict_label : 3
```



(4) Confusion Matrix 만들기 with pandas

In [11]:
```
import pandas as pd

# Coding Time
#X_test_flat, batch_size 활용하여 predict된 결과 list 만들기
pred_y = model.predict(X_test_flat, batch_size = 10000, verbose=0)
Y_pred = [x.argmax() for x in pred_y]

# Pandas를 활용하여 confusion matrix 만들기
data = {'Real' : Y_test, 'Predict' : Y_pred}
df = pd.DataFrame(data, columns=['Real','Predict'])
conf_mat = pd.crosstab(df['Real'], df['Predict'], rownames=['Real'], colnames=['Predict'])
print(conf_mat)
```

```
Predict    0     1     2    3    4    5    6     7    8    9
Real
0        967     0     3    2    0    1    4     1    1    1
1          0  1123     3    1    0    1    3     1    3    0
2          6     1  1008    2    2    0    2     7    4    0
3          0     1     9  990    0    2    0     3    5    0
4          0     0     3    1  966    0    2     2    2    6
5          2     0     0   11    1  867    4     0    6    1
6          3     3     3    1    4    3  938     0    3    0
7          0     8     8    2    0    0    0  1001    3    6
8          4     0     4    5    6    6    3     3  940    3
9          4     4     1    8   15    3    0     4    3  967
```

2.2 Simple CNN

이번 실습은 classifier 역할을 하는 DNN 앞에, feature extractor 역할을 하는 Covolution layer를 및 Maxplling layer를 덧붙여

CNN 모델을 만들고 학습시켜 볼 것이다.

(1) 데이터셋

In [12]:
```
(X_train, Y_train),(X_test, Y_test) = datasets.mnist.load_data()
print(X_train.shape, Y_train.shape)
```

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

MNIST 데이터는 load했을 때 channel이 없기 때문에 channel을 추가하여 3차원 이미지로 바꿔주어야 함(batch차원 제외)

Tensorflow base에서는 (batch, image row, image column, image channel)으로 이미지를 학습

Teano base에서는 (batch, image channel, image row, image column)으로 이미지를 학습

backend.image_data_format()로 channel의 위치를 확인하고 reshape

In [13]:
```
from tensorflow.keras import backend
backend.image_data_format()
```

Out[13]:
```
'channels_last'
```

In [14]:
```
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], X_train.shape[2], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], X_test.shape[2], 1)

Y_train = utils.to_categorical(Y_train)
Y_test = utils.to_categorical(Y_test)

print(X_train.shape, Y_train.shape)

n_in = X_train.shape[1:]
n_out = Y_train.shape[-1]
```

```
(60000, 28, 28, 1) (60000, 10)
```

(2) 모델링

<사용되는 Layer>

`Conv2D` : 이미지에 필터의 파라미터를 convolution 연산하여 다음 layer로 전달

https://keras.io/layers/convolutional/#conv2d

`MaxPooling2D` : 필터에 겹치는 값들 중 가장 큰 값만 다음 layer로 전달

https://keras.io/layers/pooling/#maxpooling2d

`Flatten` : 다차원 tensor를 1차원 벡터로 변환

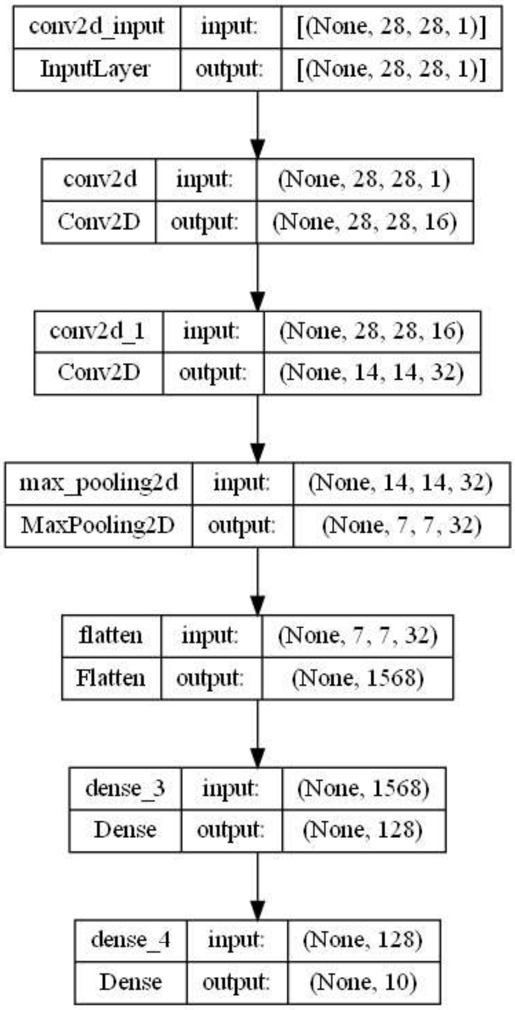
https://keras.io/layers/core/#flatten

In [15]:
```
from tensorflow.keras.layers import Flatten, BatchNormalization, Dropout, ReLU
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
```

In [16]:
```
def CNN(n_in, n_out):
    # Coding Time
    # Feature Extraction
    model = Sequential()
    model.add(Conv2D(16, kernel_size=(3, 3), padding='same', activation='relu', input_shape=n_in))
    model.add(Conv2D(32, (3, 3), padding='same', strides=(2, 2), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    # Classifier
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dense(n_out, activation='softmax'))
    return model

model=CNN(n_in, n_out)
model.summary()

from tensorflow.keras.utils import plot_model
%matplotlib inline
plot_model(model, show_shapes=True)
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 16)        160

 conv2d_1 (Conv2D)           (None, 14, 14, 32)        4640

 max_pooling2d (MaxPooling2D  (None, 7, 7, 32)         0
 )

 flatten (Flatten)           (None, 1568)              0

 dense_3 (Dense)             (None, 128)               200832

 dense_4 (Dense)             (None, 10)                1290

=================================================================
Total params: 206,922
Trainable params: 206,922
Non-trainable params: 0
_________________________________________________________________
```

Out[16]:



(3) 모델의 학습과정 설정

optimizer에 문자열 대신에 파라미터를 수정한 optimizer를 입력할 수 있음

https://keras.io/optimizers/#adam

In [17]:

```
from tensorflow.keras.optimizers import Adam
adam = Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, decay=1e-6, epsilon=None, amsgrad=False)
model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
```

(4) 모델 학습시키기

In [18]:

```
from tensorflow.keras.callbacks import EarlyStopping

# Coding Time
earlystopper = EarlyStopping(monitor='val_accuracy', patience=5, verbose=0, mode='auto')
history = model.fit(X_train, Y_train, batch_size=128, epochs=20, validation_split=0.2, callbacks = [earlystopper])
```

```
Epoch 1/20
375/375 [==============================] - 4s 3ms/step - loss: 0.7891 - accuracy: 0.8979 - val_loss: 0.1214 - val_accuracy: 0.9671
Epoch 2/20
375/375 [==============================] - 1s 3ms/step - loss: 0.0888 - accuracy: 0.9733 - val_loss: 0.1020 - val_accuracy: 0.9719
Epoch 3/20
375/375 [==============================] - 1s 3ms/step - loss: 0.0507 - accuracy: 0.9843 - val_loss: 0.0875 - val_accuracy: 0.9747
Epoch 4/20
375/375 [==============================] - 1s 3ms/step - loss: 0.0314 - accuracy: 0.9900 - val_loss: 0.0757 - val_accuracy: 0.9809
Epoch 5/20
375/375 [==============================] - 1s 3ms/step - loss: 0.0236 - accuracy: 0.9919 - val_loss: 0.0723 - val_accuracy: 0.9826
Epoch 6/20
375/375 [==============================] - 1s 3ms/step - loss: 0.0184 - accuracy: 0.9934 - val_loss: 0.0817 - val_accuracy: 0.9819
Epoch 7/20
375/375 [==============================] - 1s 3ms/step - loss: 0.0183 - accuracy: 0.9936 - val_loss: 0.0934 - val_accuracy: 0.9811
Epoch 8/20
375/375 [==============================] - 1s 3ms/step - loss: 0.0175 - accuracy: 0.9941 - val_loss: 0.0870 - val_accuracy: 0.9815
Epoch 9/20
375/375 [==============================] - 1s 3ms/step - loss: 0.0199 - accuracy: 0.9933 - val_loss: 0.0842 - val_accuracy: 0.9820
Epoch 10/20
375/375 [==============================] - 1s 3ms/step - loss: 0.0100 - accuracy: 0.9968 - val_loss: 0.0842 - val_accuracy: 0.9835
Epoch 11/20
375/375 [==============================] - 1s 3ms/step - loss: 0.0137 - accuracy: 0.9958 - val_loss: 0.0824 - val_accuracy: 0.9830
Epoch 12/20
375/375 [==============================] - 1s 3ms/step - loss: 0.0144 - accuracy: 0.9952 - val_loss: 0.0797 - val_accuracy: 0.9853
Epoch 13/20
375/375 [==============================] - 1s 3ms/step - loss: 0.0152 - accuracy: 0.9948 - val_loss: 0.1004 - val_accuracy: 0.9827
Epoch 14/20
375/375 [==============================] - 1s 3ms/step - loss: 0.0109 - accuracy: 0.9967 - val_loss: 0.0846 - val_accuracy: 0.9845
Epoch 15/20
375/375 [==============================] - 1s 3ms/step - loss: 0.0097 - accuracy: 0.9969 - val_loss: 0.0907 - val_accuracy: 0.9851
Epoch 16/20
375/375 [==============================] - 1s 3ms/step - loss: 0.0104 - accuracy: 0.9966 - val_loss: 0.1130 - val_accuracy: 0.9806
Epoch 17/20
375/375 [==============================] - 1s 3ms/step - loss: 0.0147 - accuracy: 0.9953 - val_loss: 0.0933 - val_accuracy: 0.9835
```

(5) 모델 평가하기

In [19]:
```
loss_and_accuracy = model.evaluate(X_test, Y_test, batch_size=128)
print('loss : %.4f, accruracy : %.4f'%(loss_and_accuracy[0],loss_and_accuracy[1]))
```

```
79/79 [==============================] - 0s 2ms/step - loss: 0.0767 - accuracy: 0.9842
loss : 0.0767, accruracy : 0.9842
```

2.3 BatchNormalization & DropOut

(2) 모델링

<사용되는 Layer>
`Dropout` : 일부 뉴런을 drop하여 overfitting을 방지

https://keras.io/layers/core/#dropout

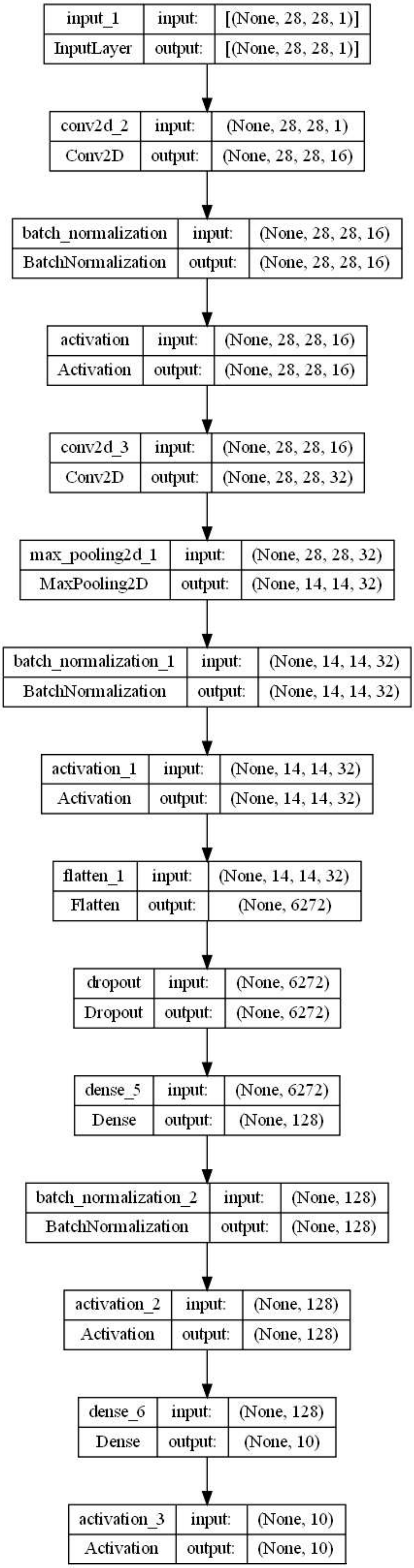
보통 batchnormalization, dropout은 동시에 사용하지 않음

In [20]:
```
def CNN_Dropout(n_in, n_out):
    # Feature Extraction
    model = Sequential()
    model.add(Conv2D(16, kernel_size=(3, 3), padding='same', input_shape=n_in))
    model.add(BatchNormalization())
    model.add(ReLU())
    model.add(Conv2D(32, (3, 3), padding='same',strides=(2, 2)))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(BatchNormalization())
    model.add(ReLU())

    # Classifier
    model.add(Flatten())
    model.add(Dropout(0.5))
    model.add(Dense(128))
    model.add(BatchNormalization())
    model.add(ReLU())
    model.add(Dense(n_out, activation='softmax'))
    return model

def CNN_Dropout_func(n_in, n_out):
    input = Input(shape=(n_in))
    x = Conv2D(16, kernel_size=(3, 3), padding='same', input_shape=n_in)(input)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(32, kernel_size=(3, 3), padding='same', input_shape=n_in)(x)
    x = MaxPooling2D(pool_size=(2, 2))(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Flatten()(x)
    x = Dropout(0.5)(x)
    x = Dense(128)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Dense(n_out)(x)
    y = Activation('softmax')(x)
    model = Model(inputs = input, outputs = y)
    return model

model=CNN_Dropout_func(n_in, n_out)
```

```
model.summary()

from tensorflow.keras.utils import plot_model
%matplotlib inline
plot_model(model, show_shapes=True)
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 28, 28, 1)]       0

 conv2d_2 (Conv2D)           (None, 28, 28, 16)        160

 batch_normalization (BatchN  (None, 28, 28, 16)       64
 ormalization)

 activation (Activation)     (None, 28, 28, 16)        0

 conv2d_3 (Conv2D)           (None, 28, 28, 32)        4640

 max_pooling2d_1 (MaxPooling  (None, 14, 14, 32)       0
 2D)

 batch_normalization_1 (Batc  (None, 14, 14, 32)       128
 hNormalization)

 activation_1 (Activation)   (None, 14, 14, 32)        0

 flatten_1 (Flatten)         (None, 6272)              0

 dropout (Dropout)           (None, 6272)              0

 dense_5 (Dense)             (None, 128)               802944

 batch_normalization_2 (Batc  (None, 128)              512
 hNormalization)

 activation_2 (Activation)   (None, 128)               0

 dense_6 (Dense)             (None, 10)                1290

 activation_3 (Activation)   (None, 10)                0

=================================================================
Total params: 809,738
Trainable params: 809,386
Non-trainable params: 352
_________________________________________________________________
```

Out[20]:


input_1 | input: | [(None, 28, 28, 1)]
InputLayer | output: | [(None, 28, 28, 1)]
conv2d_2 | input: | (None, 28, 28, 1)
Conv2D | output: | (None, 28, 28, 16)
batch_normalization | input: | (None, 28, 28, 16)
BatchNormalization | output: | (None, 28, 28, 16)
activation | input: | (None, 28, 28, 16)
Activation | output: | (None, 28, 28, 16)
conv2d_3 | input: | (None, 28, 28, 16)
Conv2D | output: | (None, 28, 28, 32)
max_pooling2d_1 | input: | (None, 28, 28, 32)
MaxPooling2D | output: | (None, 14, 14, 32)
batch_normalization_1 | input: | (None, 14, 14, 32)
BatchNormalization | output: | (None, 14, 14, 32)
activation_1 | input: | (None, 14, 14, 32)
Activation | output: | (None, 14, 14, 32)
flatten_1 | input: | (None, 14, 14, 32)
Flatten | output: | (None, 6272)
dropout | input: | (None, 6272)
Dropout | output: | (None, 6272)
dense_5 | input: | (None, 6272)
Dense | output: | (None, 128)
batch_normalization_2 | input: | (None, 128)
BatchNormalization | output: | (None, 128)
activation_2 | input: | (None, 128)
Activation | output: | (None, 128)
dense_6 | input: | (None, 128)
Dense | output: | (None, 10)
activation_3 | input: | (None, 10)
Activation | output: | (None, 10)

(3) 모델의 학습과정 설정

In [21]:
```
adam = Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, decay=1e-6, epsilon=None, amsgrad=False)
model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
```

(4) 모델 학습시키기

In [22]:
```
earlystopper = EarlyStopping(monitor='val_accuracy', patience=7, verbose=0, mode='auto')
history = model.fit(X_train, Y_train, batch_size=128, epochs=30, validation_split=0.2, callbacks = [earlystopper])
```

```
Epoch 1/30
375/375 [==============================] - 2s 5ms/step - loss: 0.1685 - accuracy: 0.9525 - val_loss: 0.0533 - val_accuracy: 0.9841
Epoch 2/30
375/375 [==============================] - 2s 4ms/step - loss: 0.0607 - accuracy: 0.9822 - val_loss: 0.0421 - val_accuracy: 0.9875
Epoch 3/30
375/375 [==============================] - 2s 5ms/step - loss: 0.0423 - accuracy: 0.9866 - val_loss: 0.0386 - val_accuracy: 0.9885
Epoch 4/30
375/375 [==============================] - 2s 5ms/step - loss: 0.0349 - accuracy: 0.9892 - val_loss: 0.0379 - val_accuracy: 0.9884
Epoch 5/30
375/375 [==============================] - 2s 4ms/step - loss: 0.0302 - accuracy: 0.9906 - val_loss: 0.0359 - val_accuracy: 0.9884
Epoch 6/30
375/375 [==============================] - 2s 4ms/step - loss: 0.0248 - accuracy: 0.9920 - val_loss: 0.0308 - val_accuracy: 0.9903
Epoch 7/30
375/375 [==============================] - 2s 4ms/step - loss: 0.0204 - accuracy: 0.9937 - val_loss: 0.0326 - val_accuracy: 0.9902
Epoch 8/30
375/375 [==============================] - 2s 4ms/step - loss: 0.0178 - accuracy: 0.9942 - val_loss: 0.0291 - val_accuracy: 0.9911
Epoch 9/30
375/375 [==============================] - 2s 5ms/step - loss: 0.0170 - accuracy: 0.9946 - val_loss: 0.0285 - val_accuracy: 0.9905
Epoch 10/30
375/375 [==============================] - 2s 5ms/step - loss: 0.0145 - accuracy: 0.9953 - val_loss: 0.0308 - val_accuracy: 0.9898
Epoch 11/30
375/375 [==============================] - 2s 5ms/step - loss: 0.0125 - accuracy: 0.9959 - val_loss: 0.0301 - val_accuracy: 0.9912
Epoch 12/30
375/375 [==============================] - 2s 4ms/step - loss: 0.0152 - accuracy: 0.9948 - val_loss: 0.0323 - val_accuracy: 0.9908
Epoch 13/30
375/375 [==============================] - 2s 5ms/step - loss: 0.0112 - accuracy: 0.9965 - val_loss: 0.0309 - val_accuracy: 0.9915
Epoch 14/30
375/375 [==============================] - 2s 5ms/step - loss: 0.0093 - accuracy: 0.9967 - val_loss: 0.0309 - val_accuracy: 0.9914
Epoch 15/30
375/375 [==============================] - 2s 4ms/step - loss: 0.0088 - accuracy: 0.9972 - val_loss: 0.0335 - val_accuracy: 0.9912
Epoch 16/30
375/375 [==============================] - 2s 5ms/step - loss: 0.0091 - accuracy: 0.9972 - val_loss: 0.0340 - val_accuracy: 0.9916
Epoch 17/30
375/375 [==============================] - 2s 4ms/step - loss: 0.0100 - accuracy: 0.9969 - val_loss: 0.0344 - val_accuracy: 0.9901
Epoch 18/30
375/375 [==============================] - 2s 5ms/step - loss: 0.0079 - accuracy: 0.9974 - val_loss: 0.0335 - val_accuracy: 0.9919
Epoch 19/30
375/375 [==============================] - 2s 4ms/step - loss: 0.0083 - accuracy: 0.9973 - val_loss: 0.0338 - val_accuracy: 0.9916
Epoch 20/30
375/375 [==============================] - 2s 5ms/step - loss: 0.0092 - accuracy: 0.9968 - val_loss: 0.0319 - val_accuracy: 0.9918
Epoch 21/30
375/375 [==============================] - 2s 5ms/step - loss: 0.0074 - accuracy: 0.9978 - val_loss: 0.0346 - val_accuracy: 0.9905
Epoch 22/30
375/375 [==============================] - 2s 5ms/step - loss: 0.0075 - accuracy: 0.9977 - val_loss: 0.0300 - val_accuracy: 0.9918
Epoch 23/30
375/375 [==============================] - 2s 5ms/step - loss: 0.0062 - accuracy: 0.9982 - val_loss: 0.0331 - val_accuracy: 0.9918
Epoch 24/30
375/375 [==============================] - 2s 4ms/step - loss: 0.0065 - accuracy: 0.9976 - val_loss: 0.0371 - val_accuracy: 0.9903
Epoch 25/30
375/375 [==============================] - 2s 5ms/step - loss: 0.0055 - accuracy: 0.9984 - val_loss: 0.0326 - val_accuracy: 0.9921
Epoch 26/30
375/375 [==============================] - 2s 5ms/step - loss: 0.0059 - accuracy: 0.9981 - val_loss: 0.0331 - val_accuracy: 0.9918
Epoch 27/30
375/375 [==============================] - 2s 5ms/step - loss: 0.0055 - accuracy: 0.9985 - val_loss: 0.0313 - val_accuracy: 0.9921
Epoch 28/30
375/375 [==============================] - 2s 5ms/step - loss: 0.0061 - accuracy: 0.9979 - val_loss: 0.0325 - val_accuracy: 0.9924
Epoch 29/30
375/375 [==============================] - 2s 5ms/step - loss: 0.0053 - accuracy: 0.9982 - val_loss: 0.0323 - val_accuracy: 0.9922
Epoch 30/30
375/375 [==============================] - 2s 5ms/step - loss: 0.0055 - accuracy: 0.9982 - val_loss: 0.0313 - val_accuracy: 0.9927
```

(5) 모델 평가하기

In [23]:
```
loss_and_accuracy = model.evaluate(X_test, Y_test, batch_size=128)
print('loss : %.4f, accruracy : %.4f'%(loss_and_accuracy[0],loss_and_accuracy[1]))
```

```
79/79 [==============================] - 0s 2ms/step - loss: 0.0297 - accuracy: 0.9929
loss : 0.0297, accruracy : 0.9929
```