케라스, 텐서플로우 버전 확인

In [4]:
```
import keras
keras.__version__
```

Out[4]: '2.3.1'

In [5]:
```
import tensorflow as tf
tf.__version__
```

Out[5]: '2.0.0'

사용 라이브러리 및 이미지 불러오기

In [1]:
```
import warnings
warnings.filterwarnings('ignore')

from keras import models, layers
from keras.callbacks import ModelCheckpoint,EarlyStopping
import cv2
from glob import glob
import os
import numpy as np
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot
import tensorflow as tf
from tensorflow import keras

from keras import regularizers
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense, Activation,Dropout, Flatten, Conv2D, MaxPooling2D,BatchNormalization
from keras.callbacks import ModelCheckpoint,EarlyStopping
import matplotlib.pyplot as plt
```

```
Using TensorFlow backend.
```

In [2]:
```
img_data = glob('C:₩₩Users₩₩82106₩₩Desktop₩₩sw_0601₩₩pokemon_g₩₩*.jpg')
class_name = ['Charmander', 'Gastly', 'Goldeen', 'Gyarados', 'Horsea', 'Mew', 'Mewtwo', 'Pikachu', 'Poliwag', 'Squirtle']
dic = {'Charmander':0,'Gastly':1,'Goldeen':2,'Gyarados':3,'Horsea':4,'Mew':5,'Mewtwo':6,'Pikachu':7,'Poliwag':8,'Squirtle':9}
dic2 = {0:'Charmander',1:'Gastly',2:'Goldeen',3:'Gyarados',4:'Horsea',5:'Mew',6:'Mewtwo',7:'Pikachu',8:'Poliwag',9:'Squirtle'
```

In [3]:
```
#데이터들을 담을 리스트 정의
X_all = list()
#레이블들을 담을 리스트 정의
Y_all = list()


for imagename in img_data:
    try:
        img = cv2.imread(imagename)
        img = cv2.resize(img, dsize=(32, 32))
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        image = np.array(img)
        X_all.append(img)

        label = imagename.split('WW')
        label = label[6]
        label = label.split('.')
        label = str(label[0])
        label = dic[label]
        Y_all.append(label)
    except :
        pass # 예외


# X, Y리스트들을 NP형식의 배열로 생성
X_all = np.array(X_all)
Y_all = np.array(Y_all)

print(X_all)
print(Y_all)
print('X_all shape: ', X_all.shape)
print('Y_all shape: ', Y_all.shape)
```

```
  [[255 255 255]
   [255 255 255]
   [255 255 255]
   ...
   [255 255 255]
   [255 255 255]
   [255 255 255]]

  [[255 255 255]
   [255 255 255]
   [255 255 255]
   ...
   [255 255 255]
   [255 255 255]
   [255 255 255]]]]
[0 0 0 ... 9 9 9]
X_all shape:  (5212, 32, 32, 3)
Y_all shape:  (5212,)
```

train, test 데이터셋 분리

In [4]:
```
X_train,X_test,Y_train,Y_test = train_test_split(X_all, Y_all, test_size = 0.2, shuffle=True, random_state=44)
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(4169, 32, 32, 3)
(1043, 32, 32, 3)
(4169,)
(1043,)
```

In [5]:
```
X_train = X_train.reshape(X_train.shape[0], 32, 32, 3)
X_test = X_test.reshape(X_test.shape[0], 32, 32, 3)
X_train = X_train.astype('float') / 255
X_test = X_test.astype('float') / 255

print('X_train_shape: ', X_train.shape)
print('X_test_shape: ', X_test.shape)
print(X_train[:5])
print(X_test[:5])
```

```
 [0.99215686 0.99215686 0.99215686]]

...

[[1.         1.         1.        ]
 [0.99607843 0.99607843 0.99607843]
 [0.99215686 1.         1.        ]
 ...
 [1.         1.         1.        ]
 [1.         1.         1.        ]
 [1.         1.         1.        ]]

[[1.         0.99607843 1.        ]
 [1.         1.         1.        ]
 [1.         1.         1.        ]
 ...
 [0.99607843 0.99607843 0.99607843]
 [1.         1.         0.98431373]
 [1.         1.         1.        ]]
```

In [6]:
```
Y_train = to_categorical(Y_train, 10)
Y_test = to_categorical(Y_test, 10)
print('Y_train_shape:', Y_train.shape)
print('Y_test_shape', Y_test.shape)
```

```
Y_train_shape: (4169, 10)
Y_test_shape (1043, 10)
```

CNN 모델 적용 및 평가

In [7]:
```
model = Sequential()
model.add(Conv2D(64, kernel_size=(5, 5), strides=(1, 1), padding='same',
                 activation='relu',
                 input_shape=(32, 32, 3)))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Conv2D(32, (2, 2), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(1000, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 64)        4864
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 32)        8224
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 32)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 8, 8, 32)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1000)              2049000
_________________________________________________________________
dropout_2 (Dropout)          (None, 1000)              0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                10010
=================================================================
Total params: 2,072,098
Trainable params: 2,072,098
Non-trainable params: 0
_________________________________________________________________
```

In [8]:
```
early_stopping = EarlyStopping(monitor = 'val_loss', patience=5, verbose=1)

model.compile(loss='categorical_crossentropy', optimizer='Adam', metrics=['accuracy'])
model.fit(X_train, Y_train, batch_size=40, epochs=40, verbose=1, callbacks = [early_stopping])
```

```
Epoch 1/40
4169/4169 [==============================] - 4s 942us/step - loss: 1.9449 - accuracy: 0.3032
Epoch 2/40
4169/4169 [==============================] - 4s 929us/step - loss: 1.3887 - accuracy: 0.5068
Epoch 3/40
4169/4169 [==============================] - 4s 905us/step - loss: 1.0884 - accuracy: 0.6311
Epoch 4/40
4169/4169 [==============================] - 4s 901us/step - loss: 0.8656 - accuracy: 0.7040
Epoch 5/40
4169/4169 [==============================] - 4s 920us/step - loss: 0.7402 - accuracy: 0.7553
Epoch 6/40
4169/4169 [==============================] - 4s 927us/step - loss: 0.5837 - accuracy: 0.7992
Epoch 7/40
4169/4169 [==============================] - 4s 946us/step - loss: 0.5284 - accuracy: 0.8285
Epoch 8/40
4169/4169 [==============================] - 4s 919us/step - loss: 0.4531 - accuracy: 0.8477
Epoch 9/40
4169/4169 [==============================] - 4s 917us/step - loss: 0.3578 - accuracy: 0.8834
Epoch 10/40
4169/4169 [==============================] - 4s 940us/step - loss: 0.3067 - accuracy: 0.8978
Epoch 11/40
4169/4169 [==============================] - 4s 923us/step - loss: 0.2527 - accuracy: 0.9194
Epoch 12/40
4169/4169 [==============================] - 4s 913us/step - loss: 0.2239 - accuracy: 0.9285
Epoch 13/40
4169/4169 [==============================] - 4s 912us/step - loss: 0.2131 - accuracy: 0.9295
Epoch 14/40
4169/4169 [==============================] - 4s 936us/step - loss: 0.1737 - accuracy: 0.9415
Epoch 15/40
4169/4169 [==============================] - 4s 921us/step - loss: 0.1543 - accuracy: 0.9523
Epoch 16/40
4169/4169 [==============================] - 4s 970us/step - loss: 0.1702 - accuracy: 0.9429
Epoch 17/40
4169/4169 [==============================] - 4s 999us/step - loss: 0.1342 - accuracy: 0.9539
Epoch 18/40
4169/4169 [==============================] - 4s 1ms/step - loss: 0.1406 - accuracy: 0.9525
Epoch 19/40
4169/4169 [==============================] - 4s 945us/step - loss: 0.1129 - accuracy: 0.9674
Epoch 20/40
4169/4169 [==============================] - 4s 953us/step - loss: 0.1002 - accuracy: 0.9671
Epoch 21/40
4169/4169 [==============================] - 4s 959us/step - loss: 0.0816 - accuracy: 0.9736
Epoch 22/40
4169/4169 [==============================] - 4s 1ms/step - loss: 0.0837 - accuracy: 0.9736
Epoch 23/40
4169/4169 [==============================] - 4s 952us/step - loss: 0.0770 - accuracy: 0.9734
Epoch 24/40
4169/4169 [==============================] - 4s 946us/step - loss: 0.0946 - accuracy: 0.9664
Epoch 25/40
4169/4169 [==============================] - ETA: 0s - loss: 0.0741 - accuracy: 0.97 - 4s 951us/step - loss: 0.0737 - accuracy:
0.9770
Epoch 26/40
4169/4169 [==============================] - 4s 979us/step - loss: 0.0827 - accuracy: 0.9739
Epoch 27/40
4169/4169 [==============================] - 4s 967us/step - loss: 0.0713 - accuracy: 0.9779
Epoch 28/40
4169/4169 [==============================] - 4s 969us/step - loss: 0.1200 - accuracy: 0.9635
Epoch 29/40
4169/4169 [==============================] - 4s 955us/step - loss: 0.0685 - accuracy: 0.9765
Epoch 30/40
4169/4169 [==============================] - 4s 989us/step - loss: 0.0673 - accuracy: 0.9770
Epoch 31/40
4169/4169 [==============================] - 4s 961us/step - loss: 0.0650 - accuracy: 0.9818
Epoch 32/40
4169/4169 [==============================] - 4s 981us/step - loss: 0.0522 - accuracy: 0.9839
Epoch 33/40
4169/4169 [==============================] - 4s 999us/step - loss: 0.0522 - accuracy: 0.9839
Epoch 34/40
4169/4169 [==============================] - 4s 993us/step - loss: 0.0401 - accuracy: 0.9870
Epoch 35/40
4169/4169 [==============================] - 4s 976us/step - loss: 0.0452 - accuracy: 0.9863
Epoch 36/40
4169/4169 [==============================] - 4s 969us/step - loss: 0.0572 - accuracy: 0.9842
Epoch 37/40
4169/4169 [==============================] - 4s 992us/step - loss: 0.0496 - accuracy: 0.9825
Epoch 38/40
4169/4169 [==============================] - 4s 1ms/step - loss: 0.0464 - accuracy: 0.9861
Epoch 39/40
4169/4169 [==============================] - 4s 974us/step - loss: 0.0334 - accuracy: 0.9897
Epoch 40/40
4169/4169 [==============================] - 4s 952us/step - loss: 0.0431 - accuracy: 0.9868
```

Out[8]: <keras.callbacks.callbacks.History at 0x16782f4ae10>

In [9]:
```
score = model.evaluate(X_test, Y_test)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

```
1043/1043 [==============================] - 0s 297us/step
Test score: 0.3161196587151872
Test accuracy: 0.9213806390762329
```

문제: 오토인코더로 생성한 사진을 내 컴퓨터에 저장시킨 후 어떤 캐릭터인지 예측

생성된 사진 개수: test 데이터인 4032개와 동일하게 4032개 생성됨

In [21]:
```
ae_images = glob('C:WWUsersWW82106WWDesktopWWsw_0601WWauto_imageWW*.jpg') # ae로 생성한 이미지의 경로

ae_test = list()
for img in ae_images:
    try:
        img = cv2.imread(img)
        img = cv2.resize(img, dsize=(32, 32))
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        img = np.array(img)
        ae_test.append(img)
    except :
        pass

ae_test = np.array(ae_test)

ae_test = ae_test.astype('float') / 255

predict_classes = np.argmax(model.predict(ae_test), axis = 1)
print(predict_classes### CNN 모델 적용 및 평가)
```

```
[2 8 9 ... 6 3 0]
```

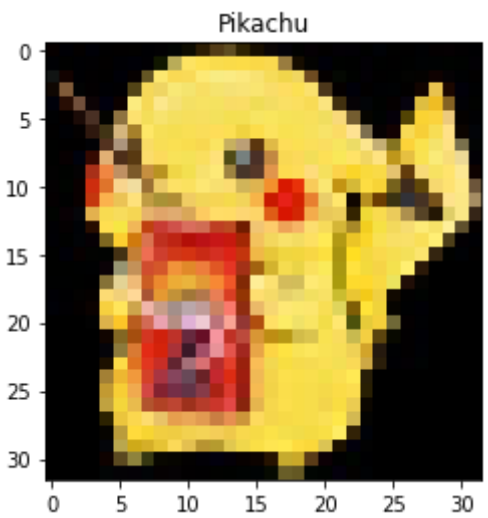
In [18]:
```
print(len(predict_classes))
```

```
4032
```

In [30]:
```
dic_pr = {0:'Charmander',1:'Gastly',2:'Goldeen',3:'Gyarados',4:'Horsea',5:'Mew',6:'Mewtwo',7:'Pikachu',8:'Poliwag',9:'Squirtl

import random
rn = random.randint(1,4032)
rn_predict = predict_classes[rn]
plt.imshow(ae_test[rn])
plt.title('{}'.format(dic_pr[rn_predict]))
```

Out[30]: Text(0.5, 1.0, 'Pikachu')



In [36]:
```
dic_pr = {0:'Charmander',1:'Gastly',2:'Goldeen',3:'Gyarados',4:'Horsea',5:'Mew',6:'Mewtwo',7:'Pikachu',8:'Poliwag',9:'Squirtl

import random
rn = random.randint(1,4032)
rn_predict = predict_classes[rn]
plt.imshow(ae_test[rn])
plt.title('{}'.format(dic_pr[rn_predict]))
```

Out[36]: Text(0.5, 1.0, 'Poliwag')