케라스, 텐서플로우 버전 확인

In [2]:
```
import keras
keras.__version__
```

Using TensorFlow backend.

Out[2]: '2.3.1'

In [3]:
```
import tensorflow as tf
tf.__version__
```

Out[3]: '2.0.0'

사용 라이브러리 및 이미지 불러오기

In [4]:
```
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
```

In [6]:
```
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### 사용 라이브러리 및 이미지 불러오기',3:'Gyarados',4:'Horsea',5:'Mew',6:'Mewtw
```

이미지, 레이블들을 저장

In [7]:

```
#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('₩₩')
        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)
```

```
[[[[  0   0   0]
   [  0   0   0]
   [  0   0   0]
   ...
   [  0   0   0]
   [  0   0   0]
   [  0   0   0]]

  [[  0   0   0]
   [  0   0   0]
   [  0   0   0]
   ...
   [  0   0   0]
   [  0   0   0]
   [  0   0   0]]

  [[  0   0   0]
   [  0   0   0]
   [  0   0   0]
```

train, test 데이터셋 분리

In [8]:

```
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,)
```

정규화 및 원핫인코딩

In [9]:
```
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])
```

```
X_train_shape:  (4169, 32, 32, 3)
X_test_shape:  (1043, 32, 32, 3)
[[[[0.         0.         0.        ]
   [0.         0.         0.        ]
   [0.         0.         0.01960784]
   ...
   [0.         0.         0.        ]
   [0.         0.         0.        ]
   [0.         0.         0.        ]]

  [[0.         0.         0.        ]
   [0.         0.         0.        ]
   [0.00392157 0.         0.05490196]
   ...
   [0.         0.         0.        ]
   [0.         0.         0.        ]
   [0.         0.         0.        ]]

  [[0.16862745 0.16470588 0.29019608]
```

In [10]:
```
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 인공지능 모델 설계

In [11]:
```
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
_________________________________________________________________
```

모델 학습시키기

In [12]:
```
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 [==============================] - 8s 2ms/step - loss: 1.9583 - accuracy: 0.2934
Epoch 2/40
4169/4169 [==============================] - 8s 2ms/step - loss: 1.3218 - accuracy: 0.5443
Epoch 3/40
4169/4169 [==============================] - 8s 2ms/step - loss: 1.0322 - accuracy: 0.6397
Epoch 4/40
4169/4169 [==============================] - 8s 2ms/step - loss: 0.7959 - accuracy: 0.7373
Epoch 5/40
4169/4169 [==============================] - 8s 2ms/step - loss: 0.6260 - accuracy: 0.7961
Epoch 6/40
4169/4169 [==============================] - 8s 2ms/step - loss: 0.5070 - accuracy: 0.8340
Epoch 7/40
4169/4169 [==============================] - 8s 2ms/step - loss: 0.4290 - accuracy: 0.8568
Epoch 8/40
4169/4169 [==============================] - 8s 2ms/step - loss: 0.3308 - accuracy: 0.8897
Epoch 9/40
4169/4169 [==============================] - 8s 2ms/step - loss: 0.2853 - accuracy: 0.9103
Epoch 10/40
4169/4169 [==============================] - 8s 2ms/step - loss: 0.2362 - accuracy: 0.9240
Epoch 11/40
4169/4169 [==============================] - 8s 2ms/step - loss: 0.2287 - accuracy: 0.9252
Epoch 12/40
4169/4169 [==============================] - 8s 2ms/step - loss: 0.1883 - accuracy: 0.9381
Epoch 13/40
4169/4169 [==============================] - 8s 2ms/step - loss: 0.1723 - accuracy: 0.9472
Epoch 14/40
4169/4169 [==============================] - 8s 2ms/step - loss: 0.1279 - accuracy: 0.9590
Epoch 15/40
4169/4169 [==============================] - 8s 2ms/step - loss: 0.1433 - accuracy: 0.9532
Epoch 16/40
4169/4169 [==============================] - 8s 2ms/step - loss: 0.0976 - accuracy: 0.9707
Epoch 17/40
4169/4169 [==============================] - 8s 2ms/step - loss: 0.1159 - accuracy: 0.9633
Epoch 18/40
4169/4169 [==============================] - 8s 2ms/step - loss: 0.0933 - accuracy: 0.9751
Epoch 19/40
4169/4169 [==============================] - 8s 2ms/step - loss: 0.0963 - accuracy: 0.9705
Epoch 20/40
4169/4169 [==============================] - 8s 2ms/step - loss: 0.0968 - accuracy: 0.9715
Epoch 21/40
4169/4169 [==============================] - 8s 2ms/step - loss: 0.0676 - accuracy: 0.9794
Epoch 22/40
4169/4169 [==============================] - 8s 2ms/step - loss: 0.1273 - accuracy: 0.9585
Epoch 23/40
4169/4169 [==============================] - 8s 2ms/step - loss: 0.0709 - accuracy: 0.9784
Epoch 24/40
4169/4169 [==============================] - 8s 2ms/step - loss: 0.0691 - accuracy: 0.9794
Epoch 25/40
4169/4169 [==============================] - 8s 2ms/step - loss: 0.1143 - accuracy: 0.9643
Epoch 26/40
4169/4169 [==============================] - 8s 2ms/step - loss: 0.0597 - accuracy: 0.9813
Epoch 27/40
4169/4169 [==============================] - 8s 2ms/step - loss: 0.0611 - accuracy: 0.9832
Epoch 28/40
4169/4169 [==============================] - 8s 2ms/step - loss: 0.0435 - accuracy: 0.9873
Epoch 29/40
4169/4169 [==============================] - 8s 2ms/step - loss: 0.0637 - accuracy: 0.9794
Epoch 30/40
4169/4169 [==============================] - 8s 2ms/step - loss: 0.0639 - accuracy: 0.9782
Epoch 31/40
4169/4169 [==============================] - 8s 2ms/step - loss: 0.0274 - accuracy: 0.9930
Epoch 32/40
4169/4169 [==============================] - 8s 2ms/step - loss: 0.0395 - accuracy: 0.9875
Epoch 33/40
4169/4169 [==============================] - 8s 2ms/step - loss: 0.0448 - accuracy: 0.9854
Epoch 34/40
4169/4169 [==============================] - 8s 2ms/step - loss: 0.0548 - accuracy: 0.9825
Epoch 35/40
4169/4169 [==============================] - 8s 2ms/step - loss: 0.0479 - accuracy: 0.9868
Epoch 36/40
4169/4169 [==============================] - 8s 2ms/step - loss: 0.0382 - accuracy: 0.9868
Epoch 37/40
4169/4169 [==============================] - 8s 2ms/step - loss: 0.0751 - accuracy: 0.9736
Epoch 38/40
4169/4169 [==============================] - 8s 2ms/step - loss: 0.0461 - accuracy: 0.9832
Epoch 39/40
4169/4169 [==============================] - 8s 2ms/step - loss: 0.0574 - accuracy: 0.9822
Epoch 40/40
4169/4169 [==============================] - 8s 2ms/step - loss: 0.0386 - accuracy: 0.9875
```

Out[12]:
```
<keras.callbacks.callbacks.History at 0x215663647f0>
```

CNN 모델 평가

In [13]:
```
score = model.evaluate(X_test, Y_test)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

```
1043/1043 [==============================] - 1s 605us/step
Test score: 0.35320062333519714
Test accuracy: 0.9290508031845093
```

VGG16 - Transfer Learning

In [14]:
```
from tensorflow.keras import Input, models, layers, optimizers, metrics
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.applications import VGG16
```

VGG 모델 설계

In [44]:

```
transfer_model = VGG16(weights='imagenet', include_top=False, input_shape=(32, 32, 3))
transfer_model.trainable = False
transfer_model.summary()

finetune_model = models.Sequential()
finetune_model.add(transfer_model)
finetune_model.add(Flatten())
finetune_model.add(Dense(64, activation='relu'))
finetune_model.add(Dense(10, activation='softmax'))
finetune_model.summary()
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_5 (InputLayer)         [(None, 32, 32, 3)]       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 32, 32, 64)        1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 32, 32, 64)        36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 16, 16, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 16, 16, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 16, 16, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 8, 8, 128)         0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 8, 8, 256)         295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 8, 8, 256)         590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 8, 8, 256)         590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 4, 4, 256)         0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 4, 4, 512)         1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 2, 2, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 2, 2, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 2, 2, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 2, 2, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 1, 1, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 0
Non-trainable params: 14,714,688
_________________________________________________________________
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 1, 1, 512)         14714688
_________________________________________________________________
flatten_4 (Flatten)          (None, 512)               0
_________________________________________________________________
dense_8 (Dense)              (None, 64)                32832
_________________________________________________________________
dense_9 (Dense)              (None, 10)                650
=================================================================
Total params: 14,748,170
Trainable params: 33,482
Non-trainable params: 14,714,688
_________________________________________________________________
```

모델 학습시키기

In [45]:

```
finetune_model.compile(loss='categorical_crossentropy', optimizer=optimizers.Adam(learning_rate=0.0002), metrics=['accuracy'
t_history = finetune_model.fit(X_train, Y_train, batch_size=50, epochs=30, validation_data=(X_test, Y_test))
```

```
Train on 4169 samples, validate on 1043 samples
Epoch 1/30
4169/4169 [==============================] - 18s 4ms/sample - loss: 2.1986 - accuracy: 0.2231 - val_loss: 2.0290 - val_accuracy:
0.3384
Epoch 2/30
4169/4169 [==============================] - 17s 4ms/sample - loss: 1.8648 - accuracy: 0.4118 - val_loss: 1.7925 - val_accuracy:
0.4343
Epoch 3/30
4169/4169 [==============================] - 17s 4ms/sample - loss: 1.6457 - accuracy: 0.4872 - val_loss: 1.6056 - val_accuracy:
0.4995
Epoch 4/30
4169/4169 [==============================] - 17s 4ms/sample - loss: 1.4880 - accuracy: 0.5395 - val_loss: 1.4860 - val_accuracy:
0.5312
Epoch 5/30
4169/4169 [==============================] - 17s 4ms/sample - loss: 1.3756 - accuracy: 0.5682 - val_loss: 1.3960 - val_accuracy:
0.5618
Epoch 6/30
4169/4169 [==============================] - 17s 4ms/sample - loss: 1.2830 - accuracy: 0.6011 - val_loss: 1.3279 - val_accuracy:
0.5733
Epoch 7/30
4169/4169 [==============================] - 17s 4ms/sample - loss: 1.2057 - accuracy: 0.6222 - val_loss: 1.2510 - val_accuracy:
0.6012
Epoch 8/30
4169/4169 [==============================] - 17s 4ms/sample - loss: 1.1378 - accuracy: 0.6472 - val_loss: 1.1999 - val_accuracy:
0.6203
Epoch 9/30
4169/4169 [==============================] - 17s 4ms/sample - loss: 1.0818 - accuracy: 0.6647 - val_loss: 1.1515 - val_accuracy:
0.6337
Epoch 10/30
4169/4169 [==============================] - 17s 4ms/sample - loss: 1.0303 - accuracy: 0.6803 - val_loss: 1.1131 - val_accuracy:
0.6453 - accu
Epoch 11/30
4169/4169 [==============================] - 17s 4ms/sample - loss: 0.9864 - accuracy: 0.6947 - val_loss: 1.0736 - val_accuracy:
0.6673
Epoch 12/30
4169/4169 [==============================] - 17s 4ms/sample - loss: 0.9435 - accuracy: 0.7141 - val_loss: 1.0369 - val_accuracy:
0.6683
Epoch 13/30
4169/4169 [==============================] - 17s 4ms/sample - loss: 0.9077 - accuracy: 0.7227 - val_loss: 1.0103 - val_accuracy:
0.6807
Epoch 14/30
4169/4169 [==============================] - 17s 4ms/sample - loss: 0.8716 - accuracy: 0.7366 - val_loss: 0.9758 - val_accuracy:
0.6855
Epoch 15/30
4169/4169 [==============================] - 17s 4ms/sample - loss: 0.8395 - accuracy: 0.7477 - val_loss: 0.9539 - val_accuracy:
0.7057
Epoch 16/30
4169/4169 [==============================] - 17s 4ms/sample - loss: 0.8098 - accuracy: 0.7587 - val_loss: 0.9344 - val_accuracy:
0.7114
Epoch 17/30
4169/4169 [==============================] - 17s 4ms/sample - loss: 0.7815 - accuracy: 0.7690 - val_loss: 0.9114 - val_accuracy:
0.7124
Epoch 18/30
4169/4169 [==============================] - 17s 4ms/sample - loss: 0.7559 - accuracy: 0.7757 - val_loss: 0.8889 - val_accuracy:
0.7315
Epoch 19/30
4169/4169 [==============================] - 17s 4ms/sample - loss: 0.7294 - accuracy: 0.7868 - val_loss: 0.8685 - val_accuracy:
0.7258
Epoch 20/30
4169/4169 [==============================] - 17s 4ms/sample - loss: 0.7069 - accuracy: 0.7992 - val_loss: 0.8432 - val_accuracy:
0.7450
Epoch 21/30
4169/4169 [==============================] - 17s 4ms/sample - loss: 0.6846 - accuracy: 0.8071 - val_loss: 0.8302 - val_accuracy:
0.7411
Epoch 22/30
4169/4169 [==============================] - 17s 4ms/sample - loss: 0.6621 - accuracy: 0.8119 - val_loss: 0.8209 - val_accuracy:
0.7450
Epoch 23/30
4169/4169 [==============================] - 17s 4ms/sample - loss: 0.6425 - accuracy: 0.8179 - val_loss: 0.7934 - val_accuracy:
0.7661
Epoch 24/30
4169/4169 [==============================] - 17s 4ms/sample - loss: 0.6242 - accuracy: 0.8261 - val_loss: 0.7932 - val_accuracy:
0.7613
Epoch 25/30
4169/4169 [==============================] - 17s 4ms/sample - loss: 0.6050 - accuracy: 0.8319 - val_loss: 0.7636 - val_accuracy:
0.7593
Epoch 26/30
4169/4169 [==============================] - 17s 4ms/sample - loss: 0.5881 - accuracy: 0.8398 - val_loss: 0.7567 - val_accuracy:
0.7709
Epoch 27/30
4169/4169 [==============================] - 17s 4ms/sample - loss: 0.5712 - accuracy: 0.8462 - val_loss: 0.7499 - val_accuracy:
0.7670
Epoch 28/30
4169/4169 [==============================] - 17s 4ms/sample - loss: 0.5562 - accuracy: 0.8460 - val_loss: 0.7321 - val_accuracy:
0.7785
```

```
Epoch 29/30
4169/4169 [==============================] - 17s 4ms/sample - loss: 0.5381 - accuracy: 0.8554 - val_loss: 0.7243 - val_accuracy: 0.7795
Epoch 30/30
4169/4169 [==============================] - 17s 4ms/sample - loss: 0.5239 - accuracy: 0.8626 - val_loss: 0.7109 - val_accuracy: 0.7919
```

VGG16 모델 평가

In [47]:

```python
score = finetune_model.evaluate(X_test, Y_test)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

```
1043/1 [======================================================================================================================
==============================================================================================================================
==============================================================================================================================
==============================================================================================================================
==============================================================================================================================
==============================================================================================================================
==============================================================================================================================
==============================================================================================================================
==============================================================================================================================
==============================================================================================================================
==============================================================================================================================
==============================================================================================================================
==============================================================================================================================
==============================================================================================================================
==============================================================================================================================
==============================================================================================================================
==============================================================================================================================
==============================================================================================================================
==============================================================================================================================
==============================================================================================================================
```

Autoencoder - Unsupervised Learning

In [55]:

```
autoencoder = Sequential()

# 인코딩 부분입니다.
autoencoder.add(Conv2D(16, kernel_size=3, padding='same', input_shape=(32,32,3), activation='relu'))
autoencoder.add(MaxPooling2D(pool_size=2, padding='same'))
autoencoder.add(Conv2D(8, kernel_size=3, activation='relu', padding='same'))
autoencoder.add(MaxPooling2D(pool_size=2, padding='same'))
autoencoder.add(Conv2D(8, kernel_size=3, strides=2, padding='same', activation='relu'))

# 디코딩 부분이 이어집니다.
autoencoder.add(Conv2D(8, kernel_size=3, padding='same', activation='relu'))
autoencoder.add(UpSampling2D())
autoencoder.add(Conv2D(8, kernel_size=3, padding='same', activation='relu'))
autoencoder.add(UpSampling2D())
#autoencoder.add(Conv2D(16, kernel_size=3, activation='relu'))
autoencoder.add(UpSampling2D())
autoencoder.add(Conv2D(3, kernel_size=3, padding='same', activation='sigmoid'))

# 전체 구조를 확인해 봅니다.
autoencoder.summary()

# 컴파일 및 학습을 하는 부분입니다.
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
autoencoder.fit(X_train, X_train, epochs=50, batch_size=100, validation_data=(X_test, X_test))
```

```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_53 (Conv2D)           (None, 32, 32, 16)        448       
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 16, 16, 16)        0         
_________________________________________________________________
conv2d_54 (Conv2D)           (None, 16, 16, 8)         1160      
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 8, 8, 8)           0         
_________________________________________________________________
conv2d_55 (Conv2D)           (None, 4, 4, 8)           584       
_________________________________________________________________
conv2d_56 (Conv2D)           (None, 4, 4, 8)           584       
_________________________________________________________________
up_sampling2d_23 (UpSampling (None, 8, 8, 8)           0         
_________________________________________________________________
conv2d_57 (Conv2D)           (None, 8, 8, 8)           584       
_________________________________________________________________
up_sampling2d_24 (UpSampling (None, 16, 16, 8)         0         
_________________________________________________________________
up_sampling2d_25 (UpSampling (None, 32, 32, 8)         0         
_________________________________________________________________
conv2d_58 (Conv2D)           (None, 32, 32, 3)         219       
=================================================================
Total params: 3,579
Trainable params: 3,579
Non-trainable params: 0
_________________________________________________________________
Train on 4169 samples, validate on 1043 samples
Epoch 1/50
4169/4169 [==============================] - 4s 952us/sample - loss: 0.6478 - val_loss: 0.5968
Epoch 2/50
4169/4169 [==============================] - 3s 816us/sample - loss: 0.5724 - val_loss: 0.5530
Epoch 3/50
4169/4169 [==============================] - 3s 826us/sample - loss: 0.5383 - val_loss: 0.5132
Epoch 4/50
4169/4169 [==============================] - 3s 829us/sample - loss: 0.5080 - val_loss: 0.4942
Epoch 5/50
4169/4169 [==============================] - 4s 840us/sample - loss: 0.4950 - val_loss: 0.4864
Epoch 6/50
4169/4169 [==============================] - 3s 833us/sample - loss: 0.4868 - val_loss: 0.4787
Epoch 7/50
4169/4169 [==============================] - 3s 832us/sample - loss: 0.4804 - val_loss: 0.4714
Epoch 8/50
4169/4169 [==============================] - 3s 825us/sample - loss: 0.4721 - val_loss: 0.4629
Epoch 9/50
4169/4169 [==============================] - 3s 834us/sample - loss: 0.4655 - val_loss: 0.4592
Epoch 10/50
4169/4169 [==============================] - 4s 844us/sample - loss: 0.4621 - val_loss: 0.4561
Epoch 11/50
4169/4169 [==============================] - 3s 827us/sample - loss: 0.4593 - val_loss: 0.4536
Epoch 12/50
4169/4169 [==============================] - 4s 842us/sample - loss: 0.4573 - val_loss: 0.4519
Epoch 13/50
4169/4169 [==============================] - 3s 834us/sample - loss: 0.4551 - val_loss: 0.4503
Epoch 14/50
4169/4169 [==============================] - 3s 838us/sample - loss: 0.4536 - val_loss: 0.4485
Epoch 15/50
4169/4169 [==============================] - 3s 837us/sample - loss: 0.4520 - val_loss: 0.4474
Epoch 16/50
4169/4169 [==============================] - 3s 838us/sample - loss: 0.4505 - val_loss: 0.4460
```

```
Epoch 17/50
4169/4169 [==============================] - 4s 935us/sample - loss: 0.4494 - val_loss: 0.4450
Epoch 18/50
4169/4169 [==============================] - 4s 858us/sample - loss: 0.4480 - val_loss: 0.4436
Epoch 19/50
4169/4169 [==============================] - 4s 858us/sample - loss: 0.4468 - val_loss: 0.4424
Epoch 20/50
4169/4169 [==============================] - 4s 852us/sample - loss: 0.4455 - val_loss: 0.4414
Epoch 21/50
4169/4169 [==============================] - 4s 852us/sample - loss: 0.4447 - val_loss: 0.4408
Epoch 22/50
4169/4169 [==============================] - 4s 847us/sample - loss: 0.4439 - val_loss: 0.4399
Epoch 23/50
4169/4169 [==============================] - 4s 859us/sample - loss: 0.4426 - val_loss: 0.4390
Epoch 24/50
4169/4169 [==============================] - 4s 843us/sample - loss: 0.4417 - val_loss: 0.4381
Epoch 25/50
4169/4169 [==============================] - 3s 838us/sample - loss: 0.4406 - val_loss: 0.4366
Epoch 26/50
4169/4169 [==============================] - 3s 828us/sample - loss: 0.4398 - val_loss: 0.4357
Epoch 27/50
4169/4169 [==============================] - 3s 837us/sample - loss: 0.4390 - val_loss: 0.4353
Epoch 28/50
4169/4169 [==============================] - 3s 833us/sample - loss: 0.4383 - val_loss: 0.4343
Epoch 29/50
4169/4169 [==============================] - 4s 842us/sample - loss: 0.4379 - val_loss: 0.4336
Epoch 30/50
4169/4169 [==============================] - 3s 837us/sample - loss: 0.4368 - val_loss: 0.4334
Epoch 31/50
4169/4169 [==============================] - 3s 827us/sample - loss: 0.4364 - val_loss: 0.4324
Epoch 32/50
4169/4169 [==============================] - 4s 844us/sample - loss: 0.4358 - val_loss: 0.4326
Epoch 33/50
4169/4169 [==============================] - 3s 834us/sample - loss: 0.4353 - val_loss: 0.4314
Epoch 34/50
4169/4169 [==============================] - 3s 835us/sample - loss: 0.4350 - val_loss: 0.4312
Epoch 35/50
4169/4169 [==============================] - 3s 831us/sample - loss: 0.4344 - val_loss: 0.4316
Epoch 36/50
4169/4169 [==============================] - 3s 826us/sample - loss: 0.4341 - val_loss: 0.4301
Epoch 37/50
4169/4169 [==============================] - 4s 851us/sample - loss: 0.4335 - val_loss: 0.4300
Epoch 38/50
4169/4169 [==============================] - 3s 837us/sample - loss: 0.4331 - val_loss: 0.4294
Epoch 39/50
4169/4169 [==============================] - 4s 848us/sample - loss: 0.4329 - val_loss: 0.4294
Epoch 40/50
4169/4169 [==============================] - 3s 826us/sample - loss: 0.4327 - val_loss: 0.4294
Epoch 41/50
4169/4169 [==============================] - 3s 823us/sample - loss: 0.4323 - val_loss: 0.4285
Epoch 42/50
4169/4169 [==============================] - 3s 825us/sample - loss: 0.4321 - val_loss: 0.4283
Epoch 43/50
4169/4169 [==============================] - 3s 822us/sample - loss: 0.4316 - val_loss: 0.4280
Epoch 44/50
4169/4169 [==============================] - 3s 823us/sample - loss: 0.4319 - val_loss: 0.4280
Epoch 45/50
4169/4169 [==============================] - 3s 820us/sample - loss: 0.4312 - val_loss: 0.4275
Epoch 46/50
4169/4169 [==============================] - 3s 834us/sample - loss: 0.4311 - val_loss: 0.4279
Epoch 47/50
4169/4169 [==============================] - 3s 830us/sample - loss: 0.4306 - val_loss: 0.4270
Epoch 48/50
4169/4169 [==============================] - 3s 834us/sample - loss: 0.4305 - val_loss: 0.4266
Epoch 49/50
4169/4169 [==============================] - 3s 837us/sample - loss: 0.4303 - val_loss: 0.4272
Epoch 50/50
4169/4169 [==============================] - 4s 841us/sample - loss: 0.4300 - val_loss: 0.4263
```

Out[55]:
```
<tensorflow.python.keras.callbacks.History at 0x2150d052710>
```

결과 출력

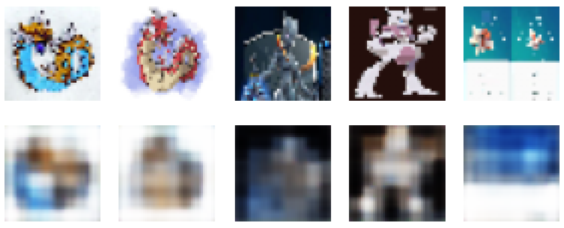
In [56]:

```
#학습된 결과를 출력하는 부분입니다.
random_test = np.random.randint(X_test.shape[0], size=5)  #테스트할 이미지를 랜덤하게 불러옵니다.
ae_imgs = autoencoder.predict(X_test)  #앞서 만든 오토인코더 모델에 집어 넣습니다.

plt.figure(figsize=(7, 2))  #출력될 이미지의 크기

for i, image_idx in enumerate(random_test):    #랜덤하게 뽑은 이미지를 차례로 나열
    ax = plt.subplot(2, 7, i + 1) ### VGG16 - Transfer Learning
    plt.imshow(X_test[image_idx].reshape(32, 32, 3))  #테스트할 이미지
    ax.axis('off')
    ax = plt.subplot(2, 7, 7 + i +1)
    plt.imshow(ae_imgs[image_idx].reshape(32, 32, 3))  #오토인코딩 결과를 다음열에 출력
    ax.axis('off')

plt.show()
```



In [54]:

```
#학습된 결과를 출력하는 부분입니다.
random_test = np.random.randint(X_test.shape[0], size=5)  #테스트할 이미지를 랜덤하게 불러옵니다.
ae_imgs = autoencoder.predict(X_test)  #앞서 만든 오토인코더 모델에 집어 넣습니다.

plt.figure(figsize=(7, 2))  #출력될 이미지의 크기

for i, image_idx in enumerate(random_test):    #랜덤하게 뽑은 이미지를 차례로 나열
    ax = plt.subplot(2, 7, i + 1)
    plt.imshow(X_test[image_idx].reshape(32, 32, 3))  #테스트할 이미지
    ax.axis('off')
    ax = plt.subplot(2, 7, 7 + i +1)
    plt.imshow(ae_imgs[image_idx].reshape(32, 32, 3))  #오토인코딩 결과를 다음열에 출력
    ax.axis('off')

plt.show()
```