케라스, 텐서플로우 버전 확인

In [20]:
```
import keras
keras.__version__
```
Out[20]: '2.3.1'

In [21]:
```
import tensorflow as tf
tf.__version__
```
Out[21]: '2.0.0'

사용 라이브러리 및 이미지 불러오기

In [3]:
```
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 [4]:
```
img_data = glob('C:₩₩Users₩₩82106₩₩Desktop₩₩sw_0601₩₩pokemon_l₩₩*.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'
```

이미지, 레이블들을 저장

In [5]:

```
#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 [6]:

```
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)
```

```
(16127, 32, 32, 3)
(4032, 32, 32, 3)
(16127,)
(4032,)
```

정규화 및 원핫인코딩

In [7]:

```
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])
```

```
  ...
  [1.         1.         1.         ]
  [1.         1.         1.         ]
  [1.         1.         1.         ]]

 [[1.         1.         1.         ]
  [1.         1.         1.         ]
  [1.         1.         1.         ]
  ...
  [1.         1.         1.         ]
  [1.         1.         1.         ]
  [1.         1.         1.         ]]

 [[1.         1.         1.         ]
  [1.         1.         1.         ]
  [1.         1.         1.         ]
  ...
  [1.         1.         1.         ]
  [1.         1.         1.         ]
  [1.         1.         1.         ]]]]
```

In [8]:

```
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: (16127, 10)
Y_test_shape (4032, 10)
```

CNN 인공지능 모델 설계

In [9]:

```
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 [10]:
```python
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
16127/16127 [==============================] - 15s 939us/step - loss: 1.3446 - accuracy: 0.5264
Epoch 2/40
16127/16127 [==============================] - 15s 933us/step - loss: 0.6275 - accuracy: 0.7885
Epoch 3/40
16127/16127 [==============================] - 15s 953us/step - loss: 0.3556 - accuracy: 0.8815
Epoch 4/40
16127/16127 [==============================] - 17s 1ms/step - loss: 0.2393 - accuracy: 0.9240
Epoch 5/40
16127/16127 [==============================] - 17s 1ms/step - loss: 0.1640 - accuracy: 0.9470
Epoch 6/40
16127/16127 [==============================] - 16s 1ms/step - loss: 0.1218 - accuracy: 0.9615
Epoch 7/40
16127/16127 [==============================] - 17s 1ms/step - loss: 0.1012 - accuracy: 0.9684
Epoch 8/40
16127/16127 [==============================] - 17s 1ms/step - loss: 0.1023 - accuracy: 0.9670
Epoch 9/40
16127/16127 [==============================] - 16s 1ms/step - loss: 0.0809 - accuracy: 0.9741
Epoch 10/40
16127/16127 [==============================] - 16s 1ms/step - loss: 0.0776 - accuracy: 0.9753
Epoch 11/40
16127/16127 [==============================] - 16s 1ms/step - loss: 0.0759 - accuracy: 0.9751
Epoch 12/40
16127/16127 [==============================] - 17s 1ms/step - loss: 0.0543 - accuracy: 0.9829
Epoch 13/40
16127/16127 [==============================] - 17s 1ms/step - loss: 0.0650 - accuracy: 0.9785
Epoch 14/40
16127/16127 [==============================] - 17s 1ms/step - loss: 0.0474 - accuracy: 0.9857
Epoch 15/40
16127/16127 [==============================] - 17s 1ms/step - loss: 0.0690 - accuracy: 0.9789
Epoch 16/40
16127/16127 [==============================] - 17s 1ms/step - loss: 0.0534 - accuracy: 0.9833
Epoch 17/40
16127/16127 [==============================] - 16s 1ms/step - loss: 0.0652 - accuracy: 0.9788
Epoch 18/40
16127/16127 [==============================] - 16s 1ms/step - loss: 0.0467 - accuracy: 0.9849
Epoch 19/40
16127/16127 [==============================] - 17s 1ms/step - loss: 0.0436 - accuracy: 0.9874
Epoch 20/40
16127/16127 [==============================] - 17s 1ms/step - loss: 0.0454 - accuracy: 0.9858
Epoch 21/40
16127/16127 [==============================] - 17s 1ms/step - loss: 0.0452 - accuracy: 0.9870
Epoch 22/40
16127/16127 [==============================] - 17s 1ms/step - loss: 0.0444 - accuracy: 0.9862
Epoch 23/40
16127/16127 [==============================] - 17s 1ms/step - loss: 0.0460 - accuracy: 0.9860
Epoch 24/40
16127/16127 [==============================] - 17s 1ms/step - loss: 0.0466 - accuracy: 0.9860
Epoch 25/40
16127/16127 [==============================] - 17s 1ms/step - loss: 0.0530 - accuracy: 0.9845
Epoch 26/40
16127/16127 [==============================] - 17s 1ms/step - loss: 0.0426 - accuracy: 0.9877
Epoch 27/40
16127/16127 [==============================] - 17s 1ms/step - loss: 0.0372 - accuracy: 0.9876
Epoch 28/40
16127/16127 [==============================] - 17s 1ms/step - loss: 0.0457 - accuracy: 0.9865
Epoch 29/40
16127/16127 [==============================] - 17s 1ms/step - loss: 0.0493 - accuracy: 0.9852
Epoch 30/40
16127/16127 [==============================] - 17s 1ms/step - loss: 0.0286 - accuracy: 0.9911
Epoch 31/40
16127/16127 [==============================] - 16s 1ms/step - loss: 0.0302 - accuracy: 0.9902
Epoch 32/40
16127/16127 [==============================] - 17s 1ms/step - loss: 0.0339 - accuracy: 0.9904
Epoch 33/40
16127/16127 [==============================] - 17s 1ms/step - loss: 0.0406 - accuracy: 0.9880
Epoch 34/40
16127/16127 [==============================] - 17s 1ms/step - loss: 0.0399 - accuracy: 0.9883
Epoch 35/40
16127/16127 [==============================] - 18s 1ms/step - loss: 0.0409 - accuracy: 0.9888
Epoch 36/40
16127/16127 [==============================] - 17s 1ms/step - loss: 0.0307 - accuracy: 0.9913
Epoch 37/40
16127/16127 [==============================] - 17s 1ms/step - loss: 0.0304 - accuracy: 0.9915
Epoch 38/40
16127/16127 [==============================] - 17s 1ms/step - loss: 0.0279 - accuracy: 0.9918
Epoch 39/40
16127/16127 [==============================] - 17s 1ms/step - loss: 0.0277 - accuracy: 0.9926
Epoch 40/40
16127/16127 [==============================] - 17s 1ms/step - loss: 0.0266 - accuracy: 0.9924
```

Out[10]:
```
<keras.callbacks.callbacks.History at 0x17025eee630>
```

CNN 모델 평가

In [11]:
```
score = model.evaluate(X_test, Y_test)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

```
4032/4032 [==============================] - 1s 269us/step
Test score: 0.043170154354246704
Test accuracy: 0.9908233880996704
```

VGG16 - Transfer Learning

In [12]:
```
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 [13]:
```python
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_1 (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"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
vgg16 (Model)                (None, 1, 1, 512)         14714688  
_________________________________________________________________
flatten (Flatten)            (None, 512)               0         
_________________________________________________________________
dense (Dense)                (None, 64)                32832     
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650       
=================================================================
Total params: 14,748,170
Trainable params: 33,482
Non-trainable params: 14,714,688
_________________________________________________________________
```

모델 학습시키기

In [26]:
```
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=1000, epochs=20, validation_data=(X_test, Y_test))
```

```
Train on 16127 samples, validate on 4032 samples
Epoch 1/20
16127/16127 [==============================] - 42s 3ms/sample - loss: 0.2793 - accuracy: 0.9327 - val_loss: 0.3689 - val_accurac
y: 0.8991
Epoch 2/20
16127/16127 [==============================] - 46s 3ms/sample - loss: 0.2768 - accuracy: 0.9333 - val_loss: 0.3667 - val_accurac
y: 0.8976
Epoch 3/20
16127/16127 [==============================] - 47s 3ms/sample - loss: 0.2745 - accuracy: 0.9347 - val_loss: 0.3644 - val_accurac
y: 0.8988
Epoch 4/20
16127/16127 [==============================] - 48s 3ms/sample - loss: 0.2725 - accuracy: 0.9341 - val_loss: 0.3634 - val_accurac
y: 0.9020
Epoch 5/20
16127/16127 [==============================] - 48s 3ms/sample - loss: 0.2709 - accuracy: 0.9353 - val_loss: 0.3607 - val_accurac
y: 0.9023
Epoch 6/20
16127/16127 [==============================] - 48s 3ms/sample - loss: 0.2691 - accuracy: 0.9360 - val_loss: 0.3598 - val_accurac
y: 0.9010
Epoch 7/20
16127/16127 [==============================] - 49s 3ms/sample - loss: 0.2677 - accuracy: 0.9367 - val_loss: 0.3596 - val_accurac
y: 0.9005
Epoch 8/20
16127/16127 [==============================] - 51s 3ms/sample - loss: 0.2661 - accuracy: 0.9382 - val_loss: 0.3576 - val_accurac
y: 0.9010
Epoch 9/20
16127/16127 [==============================] - 50s 3ms/sample - loss: 0.2644 - accuracy: 0.9371 - val_loss: 0.3562 - val_accurac
y: 0.9030
Epoch 10/20
16127/16127 [==============================] - 50s 3ms/sample - loss: 0.2627 - accuracy: 0.9372 - val_loss: 0.3536 - val_accurac
y: 0.9043
Epoch 11/20
16127/16127 [==============================] - 50s 3ms/sample - loss: 0.2610 - accuracy: 0.9397 - val_loss: 0.3531 - val_accurac
y: 0.9035
Epoch 12/20
16127/16127 [==============================] - 50s 3ms/sample - loss: 0.2593 - accuracy: 0.9392 - val_loss: 0.3518 - val_accurac
y: 0.9040
Epoch 13/20
16127/16127 [==============================] - 50s 3ms/sample - loss: 0.2580 - accuracy: 0.9398 - val_loss: 0.3508 - val_accurac
y: 0.9062
Epoch 14/20
16127/16127 [==============================] - 50s 3ms/sample - loss: 0.2564 - accuracy: 0.9403 - val_loss: 0.3493 - val_accurac
y: 0.9038
Epoch 15/20
16127/16127 [==============================] - 50s 3ms/sample - loss: 0.2549 - accuracy: 0.9406 - val_loss: 0.3467 - val_accurac
y: 0.9075
Epoch 16/20
16127/16127 [==============================] - 50s 3ms/sample - loss: 0.2534 - accuracy: 0.9421 - val_loss: 0.3469 - val_accurac
y: 0.9055
Epoch 17/20
16127/16127 [==============================] - 50s 3ms/sample - loss: 0.2526 - accuracy: 0.9415 - val_loss: 0.3444 - val_accurac
y: 0.9070
Epoch 18/20
16127/16127 [==============================] - 52s 3ms/sample - loss: 0.2510 - accuracy: 0.9425 - val_loss: 0.3434 - val_accurac
y: 0.9065
Epoch 19/20
16127/16127 [==============================] - 51s 3ms/sample - loss: 0.2489 - accuracy: 0.9420 - val_loss: 0.3428 - val_accurac
y: 0.9053
Epoch 20/20
16127/16127 [==============================] - 53s 3ms/sample - loss: 0.2473 - accuracy: 0.9435 - val_loss: 0.3410 - val_accurac
y: 0.9080
```

VGG16 모델 평가

In [27]:

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

```
==========================================================================================================================
==========================================================================================================================
==========================================================================================================================
==========================================================================================================================
==========================================================================================================================
==========================================================================================================================
==========================================================================================================================
==========================================================================================================================
==========================================================================================================================
==========================================================================================================================
==========================================================================================================================

==========================================================================================================================
==========================================================================================================================
==========================================================================================================================
============================================================================================] - 11s 3ms/sample - loss: 0.281
4 - accuracy: 0.9080

Test score: 0.34101119495573495
Test accuracy: 0.9079861
```

Autoencoder - Unsupervised Learning

In [96]:

```
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Input, Dense, Conv2D, MaxPooling2D, UpSampling2D, Flatten, Reshape

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(3, kernel_size=3, padding='same', activation='relu'))
autoencoder.add(UpSampling2D())
#autoencoder.add(Conv2D(3, kernel_size=3, padding='same', 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=100, batch_size=50, validation_data=(X_test, X_test))
```

```
16127/16127 [==============================] - 14s 890us/sample - loss: 0.3778 - val_loss: 0.3745
Epoch 87/100
16127/16127 [==============================] - 14s 890us/sample - loss: 0.3778 - val_loss: 0.3751
Epoch 88/100
16127/16127 [==============================] - 14s 889us/sample - loss: 0.3778 - val_loss: 0.3745
Epoch 89/100
16127/16127 [==============================] - 14s 888us/sample - loss: 0.3776 - val_loss: 0.3743
Epoch 90/100
16127/16127 [==============================] - 14s 889us/sample - loss: 0.3779 - val_loss: 0.3744
Epoch 91/100
16127/16127 [==============================] - 14s 881us/sample - loss: 0.3774 - val_loss: 0.3745
Epoch 92/100
16127/16127 [==============================] - 14s 872us/sample - loss: 0.3776 - val_loss: 0.3743
Epoch 93/100
16127/16127 [==============================] - 14s 890us/sample - loss: 0.3775 - val_loss: 0.3745
Epoch 94/100
16127/16127 [==============================] - 14s 895us/sample - loss: 0.3774 - val_loss: 0.3747
Epoch 95/100
16127/16127 [==============================] - 14s 886us/sample - loss: 0.3775 - val_loss: 0.3742
Epoch 96/100
```

결과 출력 ¶

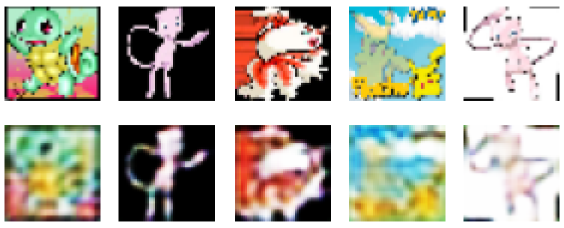
In [97]:

```
#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 [98]:

```
#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()
```