2019.09.18

소규모 데이터셋에서 CNN 네트워크 학습하기

Youngtaek Hong, PhD

0. Keras ImageDataGenerator 활용하기

ImageDataGenerator

- keras에서는 이미지데이터 학습을 쉽게 하도록 하기위해 다양한 패키지를 제공합니다.
- ImageDataGenerator 클래스를 통해 객체를 생성할 때 파라미터를 전달해주는 것을 통해 데이터의 전처리를 쉽게 할 수 있습니다.
- *flow_from_directory* 메소드를 활용하면 폴더 형태로 된 데이터 구조를 바로 가져와서 사용할 수 있습니다.
- 이 과정은 매우 직관적이고 코드도 ImageDataGenerator를 사용하지 않는 방법에 비해 상당히 짧아집니다.

ImageDataGenerator

1. 라이브러리 임포트

```
# Part 1 - Building the CNN


# Importing the Keras libraries and packages
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
```

ImageDataGenerator

```
# Initialising the CNN
classifier = Sequential()

# Step 1 - Convolution
classifier.add(Conv2D(32, (3, 3), input_shape = (64, 64, 3), activation = 'relu'))

# Step 2 - Pooling
classifier.add(MaxPooling2D(pool_size = (2, 2)))

# Adding a second convolutional layer
classifier.add(Conv2D(32, (3, 3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))

# Step 3 - Flattening
classifier.add(Flatten())

# Step 4 - Full connection
classifier.add(Dense(units = 128, activation = 'relu'))
classifier.add(Dense(units = 1, activation = 'sigmoid'))

# Compiling the CNN
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

ImageDataGenerator

- ImageDataGenerator를 만들 때 아래와 같이 rescale, shear_range, zoom_range, horizontal_flip을 설정해 간략히 전처리 수행 가능

- **featurewise_center**: Boolean. Set input mean to 0 over the dataset, feature-wise.
- **samplewise_center**: Boolean. Set each sample mean to 0.
- **featurewise_std_normalization**: Boolean. Divide inputs by std of the dataset, feature-wise.
- **samplewise_std_normalization**: Boolean. Divide each input by its std.
- **zca_epsilon**: epsilon for ZCA whitening. Default is 1e-6.
- **zca_whitening**: Boolean. Apply ZCA whitening.
- **rotation_range**: Int. Degree range for random rotations.
- **width_shift_range**: Float, 1-D array-like or int
  - float: fraction of total width, if < 1, or pixels if >= 1.
  - 1-D array-like: random elements from the array.
  - int: integer number of pixels from interval `(-width_shift_range, +width_shift_range)`
  - With `width_shift_range=2` possible values are integers `[-1, 0, +1]`, same as with `width_shift_range=[-1, 0, +1]`, while with `width_shift_range=1.0` possible values are floats in the half-open interval `[-1.0, +1.0[`.

ImageDataGenerator

- **height_shift_range**: Float, 1-D array-like or int
  - float: fraction of total height, if < 1, or pixels if >= 1.
  - 1-D array-like: random elements from the array.
  - int: integer number of pixels from interval `(-height_shift_range, +height_shift_range)`
  - With `height_shift_range=2` possible values are integers `[-1, 0, +1]`, same as with `height_shift_range=[-1, 0, +1]`, while with `height_shift_range=1.0` possible values are floats in the half-open interval `[-1.0, +1.0[`.
- **brightness_range**: Tuple or list of two floats. Range for picking a brightness shift value from.
- **shear_range**: Float. Shear Intensity (Shear angle in counter-clockwise direction in degrees)
- **zoom_range**: Float or [lower, upper]. Range for random zoom. If a float, `[lower, upper] = [1-zoom_range, 1+zoom_range]`.
- **channel_shift_range**: Float. Range for random channel shifts.

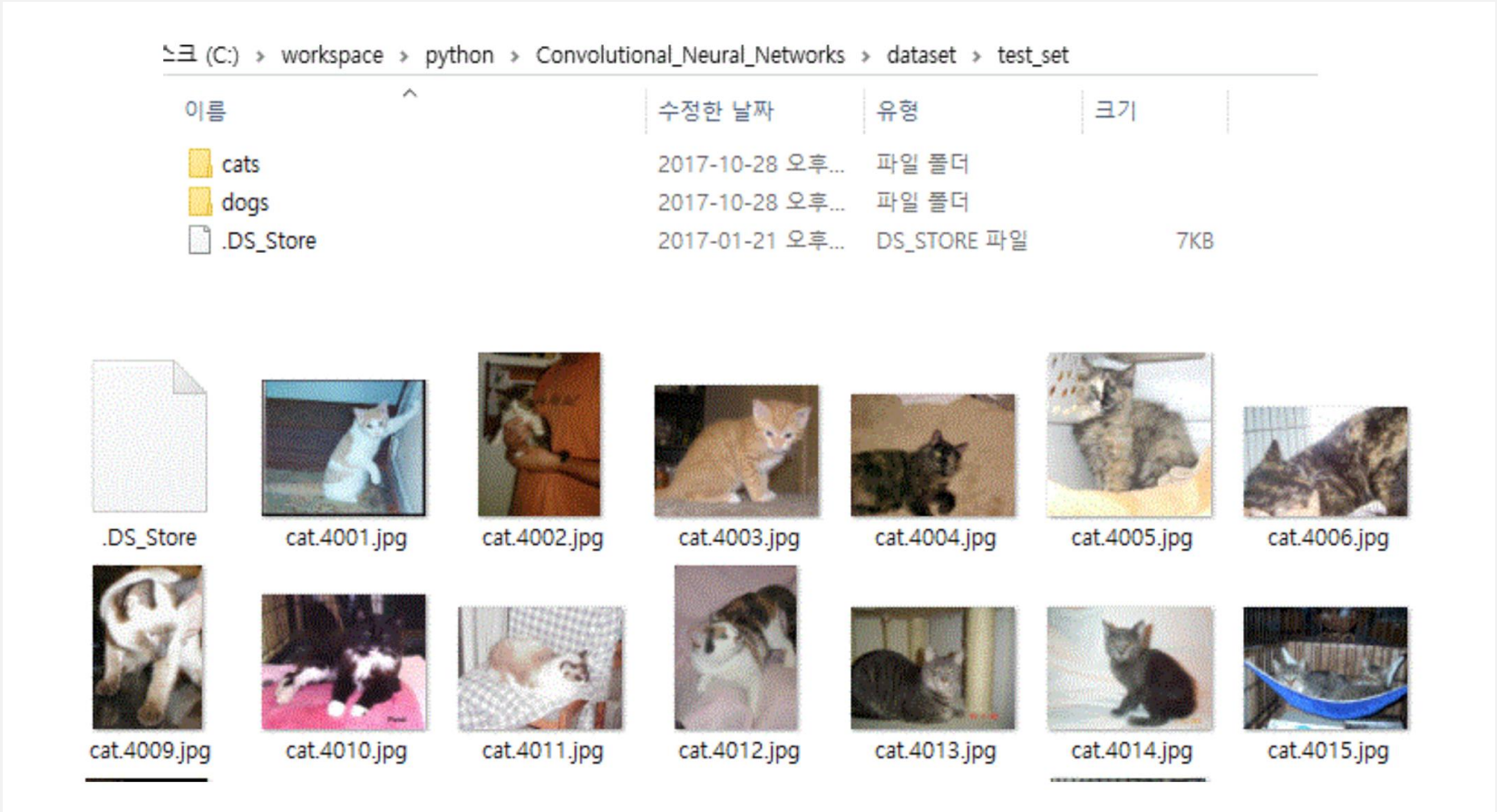
ImageDataGenerator

```
# Part 2 - Fitting the CNN to the images
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)

training_set = train_datagen.flow_from_directory('dataset/training_set',
                                                 target_size = (64, 64),
                                                 batch_size = 32,
                                                 class_mode = 'binary')

test_set = test_datagen.flow_from_directory('dataset/test_set',
                                            target_size = (64, 64),
                                            batch_size = 32,
                                            class_mode = 'binary')

classifier.fit_generator(training_set,
                         steps_per_epoch = 300,
                         epochs = 25,
                         validation_data = test_set,
                         validation_steps = 2000)
```

ImageDataGenerator

- 또한 flow_from_directory 메소드를 사용하면 폴더 구조를 그대로 가져와서 ImageDataGenerator 객체의 실제 데이터를 채워준며, 이 데이터를 불러올 때 앞서 정의한 파라미터로 전처리를 합니다.
- 마지막으로 fit_generator 함수를 실행함으로써 모델 fitting이 이루어진다.
- **validation_steps**는 한 번 epoch 돌 고난 후, validation set을 통해 validation accuracy를 측정할 때 validation set을 몇 번 볼 것인지를 정해준다. **[validation data수/배치사이즈]**
- 즉, ImageDataGenerator를 쓴 경우, fit_generator를 사용하면 됩니다.
- 폴더 구조는 아래와 같이 하면 된다. flow_from_directory에 넣어준 경로(dataset/training_set) 밑에 이런식으로 class(cats, dogs) 별로 폴더를 만들고 폴더 밑에 이미지들을 넣어준다. 그러면 알아서 labeling을 하게 된다.

ImageDataGenerator



ImageDataGenerator

```
output = classifier.predict_generator(test_set, steps=5)
print(test_set.class_indices)
print(output)
```

```
{'cats': 0, 'dogs': 1}
```

1. 사전 훈련된 VGG 네트워크 활용하기

1. VGG on small datasets

```
import os
import numpy as np
from keras.preprocessing.image import ImageDataGenerator

# 모든 이미지를 1/255로 스케일을 조정합니다
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

base_dir = '/content/gdrive/My Drive/datasets/cats_and_dogs_small'

train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')


train_generator = train_datagen.flow_from_directory(
        # 타깃 디렉터리
        train_dir,
        # 모든 이미지를 224 × 224 크기로 바꿉니다
        target_size=(224, 224),
        batch_size=20,
        # binary_crossentropy 손실을 사용하기 때문에 이진 레이블이 필요합니다
        class_mode='binary')

validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(224, 224),
        batch_size=20,
        class_mode='binary')
```

1. VGG on small datasets

```
#  This code cell shows how to utilize VGG16 model by combining Dense layer at the er
from keras.layers import Input, Dense, GlobalAveragePooling2D, Flatten
from keras.models import Model

from keras.applications.vgg16 import VGG16
from keras import layers
from keras import models

VGGNet = VGG16()

# VGGNet.summary()

# We will not update VGG pre-trained model, only added Dense layers will be trained f
for layer in VGGNet.layers:
  layer.trainable = False

vgg_maxpool5 = VGGNet.get_layer('block5_pool').output

Feature_Flatten = Flatten()(vgg_maxpool5)
dense = Dense(10, name='dense', activation='relu')(Feature_Flatten)
predictions = Dense(1, activation='sigmoid')(dense)
New_VGGmodel = Model(inputs=VGGNet.input, outputs=predictions)
New_VGGmodel.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accur

New_VGGmodel.summary()
```

1. VGG on small datasets

```
#  This code cell shows how to utilize VGG16 model by combining Dense layer at the er
from keras.layers import Input, Dense, GlobalAveragePooling2D, Flatten
from keras.models import Model

from keras.applications.vgg16 import VGG16
from keras import layers
from keras import models

VGGNet = VGG16()

# VGGNet.summary()

# We will not update VGG pre-trained model, only added Dense layers will be trained f
for layer in VGGNet.layers:
  layer.trainable = False

vgg_maxpool5 = VGGNet.get_layer('block5_pool').output

Feature_Flatten = Flatten()(vgg_maxpool5)
dense = Dense(10, name='dense', activation='relu')(Feature_Flatten)
predictions = Dense(1, activation='sigmoid')(dense)
New_VGGmodel = Model(inputs=VGGNet.input, outputs=predictions)
New_VGGmodel.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accur

New_VGGmodel.summary()
```

1. VGG on small datasets

```
history = New_VGGmodel.fit_generator(
      train_generator,
      steps_per_epoch=10,
      epochs=30,
      validation_data=validation_generator,
      validation_steps=50)
```

```
Epoch 1/30
10/10 [==============================] - 407s 41s/step - loss: 1.1114 - acc: 0.5750
Epoch 2/30
10/10 [==============================] - 7s 651ms/step - loss: 0.6297 - acc: 0.7000
Epoch 3/30
10/10 [==============================] - 68s 7s/step - loss: 0.5477 - acc: 0.7800 -
Epoch 4/30
10/10 [==============================] - 77s 8s/step - loss: 0.5195 - acc: 0.7650 -
Epoch 5/30
10/10 [==============================] - 80s 8s/step - loss: 0.5840 - acc: 0.7350 -
```

2. 사전 훈련된 Inception 네트워크 활용하기

2.Inception on small datasets

```
from keras.applications.inception_v3 import InceptionV3

Incep= InceptionV3(weights='imagenet',include_top = False,input_shape = (224, 224, 3)
for layer in Incep.layers:
  layer.trainable = False
```

2.Inception on small datasets

```
incep_maxpool4 = Incep.layers[-2].output

Feature_Flatten = Flatten()(incep_maxpool4)
dense = Dense(10, name = 'dense', activation = 'relu')(Feature_Flatten)
predictions = Dense(1, activation='sigmoid')(dense)
New_incep = Model(inputs=Incep.input, outputs = predictions)
New_incep.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy

New_incep.summary()
```

3. Cifar-10 Dataset Challenge

3. CIFAR10 small image classification

Dataset of 50,000 32x32 color training images, labeled over 10 categories, and 10,000 test images.

Usage:

```
from keras.datasets import cifar10

(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

- **Returns:**
  - 2 tuples:
    - **x_train, x_test**: uint8 array of RGB image data with shape (num_samples, 3, 32, 32) or (num_samples, 32, 32, 3) based on the `image_data_format` backend setting of either `channels_first` or `channels_last` respectively.
    - **y_train, y_test**: uint8 array of category labels (integers in range 0-9) with shape (num_samples,).