File Edit View Insert Cell Kernel Navigate Widgets Help

Not Trusted | Python 3 (ipykernel)

Run | Code

1 Chapter 5. Deep learning for computer vision

In [1]:
```
import platform
import tensorflow
import keras
```

In [2]:
```
from keras.applications import VGG16

conv_base = VGG16(weights='imagenet',
                  include_top=False, # 在這裡告訴 keras我們只需要卷積基底的權重模型資訊
                  input_shape=(150, 150, 3)) # 宣告我們要處理的圖像大小與顏色通道數
```

In [3]:
```
conv_base.summary()
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 150, 150, 3)]     0

block1_conv1 (Conv2D)        (None, 150, 150, 64)      1792

block1_conv2 (Conv2D)        (None, 150, 150, 64)      36928

block1_pool (MaxPooling2D)   (None, 75, 75, 64)        0

block2_conv1 (Conv2D)        (None, 75, 75, 128)       73856

block2_conv2 (Conv2D)        (None, 75, 75, 128)       147584

block2_pool (MaxPooling2D)   (None, 37, 37, 128)       0

block3_conv1 (Conv2D)        (None, 37, 37, 256)       295168

block3_conv2 (Conv2D)        (None, 37, 37, 256)       590080

block3_conv3 (Conv2D)        (None, 37, 37, 256)       590080

block3_pool (MaxPooling2D)   (None, 18, 18, 256)       0

block4_conv1 (Conv2D)        (None, 18, 18, 512)       1180160

block4_conv2 (Conv2D)        (None, 18, 18, 512)       2359808

block4_conv3 (Conv2D)        (None, 18, 18, 512)       2359808

block4_pool (MaxPooling2D)   (None, 9, 9, 512)         0

block5_conv1 (Conv2D)        (None, 9, 9, 512)         2359808

block5_conv2 (Conv2D)        (None, 9, 9, 512)         2359808

block5_conv3 (Conv2D)        (None, 9, 9, 512)         2359808

block5_pool (MaxPooling2D)   (None, 4, 4, 512)         0

=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

1.1 Listing 5.22. Freezing all layers up to a specific one

In [4]:
```
conv_base.trainable = True#可訓練的 frozen

set_trainable = False#不可訓練 unfrozen
for layer in conv_base.layers:#1:5
    if layer.name == 'block5_conv1':#後面的Layer定義成可訓練
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False
```

1.2 Listing 5.23. Fine-tuning the model

In [5]:
```
import os
```

```
import numpy as np
from keras.preprocessing.image import ImageDataGenerator
base_dir = 'C:/Users/Huang/Downloads/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')

datagen = ImageDataGenerator(rescale=1./255)
batch_size = 20 # 設定每次產生的圖像的數據批量
```

In [6]:

```
def extract_features(directory, sample_count):# 影像的目錄, 要處理的圖像數
    features = np.zeros(shape=(sample_count, 4, 4, 512))# 根據VGG16(卷積基底)的最後一層的輸出張量規格
    labels = np.zeros(shape=(sample_count))# 要處理的圖像數

    # 產生一個"圖像資料產生器"實例(資料是在檔案目錄中), 每呼叫它一次, 它會吐出特定批次數的圖像資料
    generator = datagen.flow_from_directory(
        directory,
        target_size=(150, 150),# 設定圖像的高(height)與寬(width)
        batch_size=batch_size, # 設定每次產生的圖像的數據批量
        class_mode='binary')# 因為我們的目標資料集只有兩類(cat & dog)

    # 讓我們把訓練資料集所有的圖像都跑過一次
    i=0
    for inputs_batch, labels_batch in generator:
        features_batch = conv_base.predict(inputs_batch)# 透過"卷積基底"來萃取圖像特徵
        features[i * batch_size : (i + 1) * batch_size] = features_batch# 把特徵先存放起來
        labels[i * batch_size : (i + 1) * batch_size] = labels_batch#把標籤先存放起來
        i += 1
        if i * batch_size >= sample_count:
            break
    return features, labels

train_features, train_labels = extract_features(train_dir, 2000) # 訓練資料的圖像特徵萃取
validation_features, validation_labels = extract_features(validation_dir, 1000)# 驗證資料的圖像特徵萃取
test_features, test_labels = extract_features(test_dir, 1000)# 測試資料的圖像特徵萃取
```

2.2 Listing 6.2. Character-level o
2.3 Listing 6.3. Using Keras for w
2.4 Listing 6.4. Word-level one-h

```
1/1 [==============================] - 3s 3s/step
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 3s 3s/step
1/1 [==============================] - 3s 3s/step
1/1 [==============================] - 3s 3s/step
1/1 [==============================] - 3s 3s/step
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 3s 3s/step
1/1 [==============================] - 3s 3s/step
```

In [ ]:

```
#提取的特徵當前是(樣本數,4,4,512)的形狀,我們將它們餵給一個密集連接(densely-connected)的分類器,所以首先扌
train_features = np.reshape(train_features, (2000, 4*4* 512))
validation_features = np.reshape(validation_features, (1000, 4*4* 512))
test_features = np.reshape(test_features, (1000, 4*4* 512))
```

2.2 Listing 6.2. Character-level o
2.3 Listing 6.3. Using Keras for w
2.4 Listing 6.4. Word-level one-h

In [ ]:

```
from keras.preprocessing.image import ImageDataGenerator
from keras import optimizers

train_datagen = ImageDataGenerator(
      rescale=1./255,
      rotation_range=40,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')

test_datagen = ImageDataGenerator(rescale=1./255)
```

```
train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')

validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')
```

In [34]:

```
from keras import models
from keras import layers
from keras import optimizers

model = models.Sequential()
model.add(conv_base)
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-5),
              metrics=['acc'])
```

In [35]:

```
model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 4, 4, 512)         14714688

 flatten_1 (Flatten)         (None, 8192)              0

 dense_12 (Dense)            (None, 256)               2097408

 dense_13 (Dense)            (None, 1)                 257

=================================================================
Total params: 16,812,353
Trainable params: 9,177,089
Non-trainable params: 7,635,264
_________________________________________________________________
```

In [ ]:

```
history = model.fit_generator(
      train_generator,
      steps_per_epoch=100,
      epochs=100,
      validation_data=validation_generator,
      validation_steps=50)
```

```
C:\Users\Huang\AppData\Local\Temp\ipykernel_2152\1164194086.py:1: UserWarning: `Model.fit_generator` is deprecated and will be rem
oved in a future version. Please use `Model.fit`, which supports generators.
  history = model.fit_generator(


Epoch 1/100
100/100 [==============================] - 416s 4s/step - loss: 0.2843 - acc: 0.8785 - val_loss: 0.2228 - val_acc: 0.9010
Epoch 2/100
100/100 [==============================] - 391s 4s/step - loss: 0.2530 - acc: 0.8930 - val_loss: 0.2170 - val_acc: 0.9120
Epoch 3/100
100/100 [==============================] - 389s 4s/step - loss: 0.2592 - acc: 0.8900 - val_loss: 0.1888 - val_acc: 0.9190
Epoch 4/100
100/100 [==============================] - 401s 4s/step - loss: 0.2278 - acc: 0.9065 - val_loss: 0.1848 - val_acc: 0.9180
Epoch 5/100
100/100 [==============================] - 406s 4s/step - loss: 0.2177 - acc: 0.9035 - val_loss: 0.1917 - val_acc: 0.9190
Epoch 6/100
100/100 [==============================] - 424s 4s/step - loss: 0.2125 - acc: 0.9175 - val_loss: 0.1929 - val_acc: 0.9220
Epoch 7/100
100/100 [==============================] - ETA: 0s - loss: 0.2018 - acc: 0.9080
```

In [ ]:

```
import matplotlib.pyplot as plt

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
```

```
epochs = range(len(acc))

plt.plot(epochs, acc, label='Training acc')
plt.plot(epochs, val_acc, label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, label='Training loss')
plt.plot(epochs, val_loss, label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

1.3 Listing 5.24. Smoothing the plots

In [37]:

```
def smooth_curve(points, factor=0.8):
  smoothed_points = []
  for point in points:
    if smoothed_points:
      previous = smoothed_points[-1]
      smoothed_points.append(previous * factor + point * (1 - factor))
    else:
      smoothed_points.append(point)
  return smoothed_points

plt.plot(epochs,
         smooth_curve(acc), 'bo', label='Smoothed training acc')
plt.plot(epochs,
         smooth_curve(val_acc), 'b', label='Smoothed validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs,
         smooth_curve(loss), 'bo', label='Smoothed training loss')
plt.plot(epochs,
         smooth_curve(val_loss), 'b', label='Smoothed validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

2.2 Listing 6.2. Character-level or
2.3 Listing 6.3. Using Keras for w
2.4 Listing 6.4. Word-level one-ho

In [ ]:

```
test_generator = test_datagen.flow_from_directory(
        test_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')

test_loss, test_acc = model.evaluate_generator(test_generator, steps=50)
print('test acc:', test_acc)
```

2.2 Listing 6.2. Character-level or
2.3 Listing 6.3. Using Keras for w
2.4 Listing 6.4. Word-level one-ho

2 Chapter 6. Deep learning for text and sequences

2.1 Listing 6.1. Word-level one-hot encoding (toy example)

In [43]:

```
import numpy as np

samples = ['The cat sat on the mat.', 'The dog ate my homework.']

token_index = {}
for sample in samples:
    for word in sample.split():
        if word not in token_index:
```

```
                token_index[word] = len(token_index) + 1

    max_length = 10

    results = np.zeros(shape=(len(samples),
                              max_length,
                              max(token_index.values()) + 1))
    for i, sample in enumerate(samples):
        for j, word in list(enumerate(sample.split()))[:max_length]:
            index = token_index.get(word)
            results[i, j, index] = 1.
    results
```

```
array([[[0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 1., 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., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0., 1., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
        [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
        [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., 0., 0., 0., 0., 0.]]])
```

2.2 Listing 6.2. Character-level one-hot encoding (toy example)

In [44]:

```
import string

samples = ['The cat sat on the mat.', 'The dog ate my homework.']
characters = string.printable
token_index = dict(zip(range(1, len(characters) + 1), characters))

max_length = 50
results = np.zeros((len(samples), max_length, max(token_index.keys()) + 1))
for i, sample in enumerate(samples):
    for j, character in enumerate(sample):
        index = token_index.get(character)
        results[i, j, index] = 1.
results
```

```
array([[[1., 1., 1., ..., 1., 1., 1.],
        [1., 1., 1., ..., 1., 1., 1.],
        [1., 1., 1., ..., 1., 1., 1.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]],

       [[1., 1., 1., ..., 1., 1., 1.],
        [1., 1., 1., ..., 1., 1., 1.],
        [1., 1., 1., ..., 1., 1., 1.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]]])
```

2.3 Listing 6.3. Using Keras for word-level one-hot encoding

In [52]:

```
from keras.preprocessing.text import Tokenizer

samples = ['The cat sat on the mat.', 'The dog ate my homework.']

tokenizer = Tokenizer(num_words=1000)
tokenizer.fit_on_texts(samples)
print('tokenizer',tokenizer,'\n')

sequences = tokenizer.texts_to_sequences(samples)
print('sequences',sequences,'\n')
```

```
one_hot_results = tokenizer.texts_to_matrix(samples, mode='binary')
print('one_hot_results',one_hot_results,'\n')

word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
```

```
tokenizer <keras.preprocessing.text.Tokenizer object at 0x000001973DE9C970>

sequences [[1, 2, 3, 4, 1, 5], [1, 6, 7, 8, 9]]

one_hot_results [[0. 1. 1. ... 0. 0. 0.]
 [0. 1. 0. ... 0. 0. 0.]]

Found 9 unique tokens.
```

2.4 Listing 6.4. Word-level one-hot encoding with hashing trick (toy example)

In [53]:

```
samples = ['The cat sat on the mat.', 'The dog ate my homework.']

dimensionality = 1000
max_length = 10

results = np.zeros((len(samples), max_length, dimensionality))
for i, sample in enumerate(samples):
    for j, word in list(enumerate(sample.split()))[:max_length]:
        index = abs(hash(word)) % dimensionality
        results[i, j, index] = 1.
results
```

2.2 Listing 6.2. Character-level oı
2.3 Listing 6.3. Using Keras for w
2.4 Listing 6.4. Word-level one-h

```
array([[[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., ..., 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.]]])
```