Contents

In [1]:
```
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
```

1 Listing 5.16 Instantiating the VGG16 convolutional base

- 大神
- 大陸人寫的
- 參考

In [2]:
```
conv_base.summary()# 打印一下模型資訊
```

```
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
_________________________________________________________________
```

2 Listing 5.17 Extracting features using the pretrained convolutional base

- 方法1: 卷積基底:提取特徵 + 串接新的密集分類層:重新訓練

> 我們來看看設置第一種方法所需的程式碼：在我們的數據上記錄“conv_base”的輸出，並使用這些輸出作為新模型的輸入。

> 我們將首先簡單地運行以前介紹的“ImageDataGenerator”的實例，以將圖像提取為Numpy數組及其標籤。我們將通過調用 conv_base模型的predict方法從這些圖像中提取特徵。

In [3]:
```
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')
```

```
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 [4]:

```
# 提取圖像特徵
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
```

```
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
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
```

In [5]:

```
#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))
```

4 Listing 5.19 Plotting the results
5 Listing 5.20 Adding a densely con
6 Listing 5.21 Training the model er

3 Listing 5.18 Defining and training the densely connected classifier

In [6]:

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

# 產生一個新的密集連接層來做為分類器
model = models.Sequential()
model.add(layers.Dense(256, activation='relu', input_dim=4 * 4 * 512))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer=optimizers.RMSprop(lr=2e-5),
              loss='binary_crossentropy',
              metrics=['acc'])

# 把透過預處理的卷積基底所提取的特徵做為input來進行訓練
```

```
history = model.fit(train_features, train_labels,
                    epochs=30,
                    batch_size=20,
                    validation_data=(validation_features, validation_labels))
```

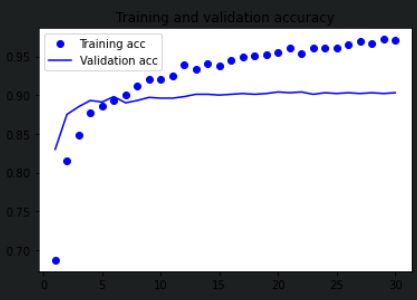
```
Epoch 1/30

C:\Users\Huang\anaconda3\lib\site-packages\keras\optimizers\optimizer_v2\rmsprop.py:143: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super().__init__(name, **kwargs)

100/100 [==============================] - 5s 33ms/step - loss: 0.5872 - acc: 0.6870 - val_loss: 0.4373 - val_acc: 0.8300
Epoch 2/30
100/100 [==============================] - 3s 27ms/step - loss: 0.4236 - acc: 0.8150 - val_loss: 0.3568 - val_acc: 0.8750
Epoch 3/30
100/100 [==============================] - 3s 27ms/step - loss: 0.3666 - acc: 0.8480 - val_loss: 0.3181 - val_acc: 0.8850
Epoch 4/30
100/100 [==============================] - 3s 28ms/step - loss: 0.3138 - acc: 0.8775 - val_loss: 0.2941 - val_acc: 0.8930
Epoch 5/30
100/100 [==============================] - 3s 27ms/step - loss: 0.2850 - acc: 0.8860 - val_loss: 0.2796 - val_acc: 0.8910
Epoch 6/30
100/100 [==============================] - 3s 28ms/step - loss: 0.2664 - acc: 0.8930 - val_loss: 0.2700 - val_acc: 0.8980
Epoch 7/30
100/100 [==============================] - 3s 27ms/step - loss: 0.2506 - acc: 0.9000 - val_loss: 0.2627 - val_acc: 0.8900
Epoch 8/30
100/100 [==============================] - 3s 27ms/step - loss: 0.2345 - acc: 0.9115 - val_loss: 0.2551 - val_acc: 0.8930
Epoch 9/30
100/100 [==============================] - 3s 27ms/step - loss: 0.2170 - acc: 0.9210 - val_loss: 0.2480 - val_acc: 0.8970
Epoch 10/30
100/100 [==============================] - 3s 27ms/step - loss: 0.2148 - acc: 0.9200 - val_loss: 0.2446 - val_acc: 0.8960
Epoch 11/30
100/100 [==============================] - 3s 27ms/step - loss: 0.1998 - acc: 0.9245 - val_loss: 0.2467 - val_acc: 0.8960
Epoch 12/30
100/100 [==============================] - 3s 27ms/step - loss: 0.1812 - acc: 0.9390 - val_loss: 0.2408 - val_acc: 0.8980
Epoch 13/30
100/100 [==============================] - 3s 27ms/step - loss: 0.1809 - acc: 0.9330 - val_loss: 0.2471 - val_acc: 0.9010
Epoch 14/30
100/100 [==============================] - 3s 27ms/step - loss: 0.1691 - acc: 0.9400 - val_loss: 0.2359 - val_acc: 0.9010
Epoch 15/30
100/100 [==============================] - 3s 27ms/step - loss: 0.1675 - acc: 0.9375 - val_loss: 0.2393 - val_acc: 0.9000
Epoch 16/30
100/100 [==============================] - 3s 27ms/step - loss: 0.1582 - acc: 0.9450 - val_loss: 0.2338 - val_acc: 0.9010
Epoch 17/30
100/100 [==============================] - 3s 27ms/step - loss: 0.1486 - acc: 0.9500 - val_loss: 0.2371 - val_acc: 0.9020
Epoch 18/30
100/100 [==============================] - 3s 27ms/step - loss: 0.1444 - acc: 0.9510 - val_loss: 0.2391 - val_acc: 0.9010
Epoch 19/30
100/100 [==============================] - 3s 27ms/step - loss: 0.1384 - acc: 0.9515 - val_loss: 0.2314 - val_acc: 0.9020
Epoch 20/30
100/100 [==============================] - 3s 27ms/step - loss: 0.1324 - acc: 0.9555 - val_loss: 0.2344 - val_acc: 0.9040
Epoch 21/30
100/100 [==============================] - 3s 27ms/step - loss: 0.1275 - acc: 0.9605 - val_loss: 0.2311 - val_acc: 0.9030
Epoch 22/30
100/100 [==============================] - 3s 27ms/step - loss: 0.1295 - acc: 0.9535 - val_loss: 0.2335 - val_acc: 0.9040
Epoch 23/30
100/100 [==============================] - 3s 27ms/step - loss: 0.1191 - acc: 0.9610 - val_loss: 0.2306 - val_acc: 0.9010
Epoch 24/30
100/100 [==============================] - 3s 28ms/step - loss: 0.1131 - acc: 0.9610 - val_loss: 0.2359 - val_acc: 0.9030
Epoch 25/30
100/100 [==============================] - 3s 28ms/step - loss: 0.1157 - acc: 0.9605 - val_loss: 0.2336 - val_acc: 0.9020
Epoch 26/30
100/100 [==============================] - 3s 27ms/step - loss: 0.1103 - acc: 0.9655 - val_loss: 0.2324 - val_acc: 0.9030
Epoch 27/30
100/100 [==============================] - 3s 28ms/step - loss: 0.0984 - acc: 0.9690 - val_loss: 0.2378 - val_acc: 0.9020
Epoch 28/30
100/100 [==============================] - 3s 28ms/step - loss: 0.0992 - acc: 0.9665 - val_loss: 0.2331 - val_acc: 0.9030
Epoch 29/30
100/100 [==============================] - 3s 29ms/step - loss: 0.0907 - acc: 0.9720 - val_loss: 0.2346 - val_acc: 0.9020
Epoch 30/30
100/100 [==============================] - 3s 28ms/step - loss: 0.0929 - acc: 0.9715 - val_loss: 0.2353 - val_acc: 0.9030
```

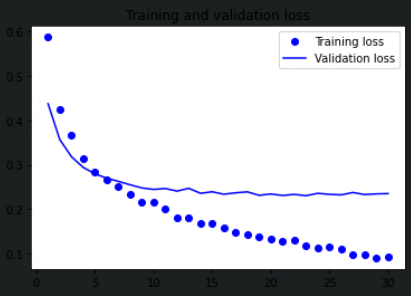
4 Listing 5.19 Plotting the results

In [7]:

```
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(1, len(acc) + 1)

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

```
plt.figure()

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





4 Listing 5.19 Plotting the results
5 Listing 5.20 Adding a densely co
6 Listing 5.21 Training the model er

5 Listing 5.20 Adding a densely connected classifier on top of the convolutional base

- (卷積基底:凍結 + 串接新的密集分類層) >> 重新訓練

> 現在，我們來回顧一下我們提到的第二種特徵提取技術，這種方法要慢得多，而且會花更多的時間與計算資源，但是我們可以在訓練過程中利用數據擴充(data augmentation): 擴展conv_base模型，並進行端(end)對端(end)的訓練。

> 請注意，這種手法真的是非常昂貴的，所以只有在你有GPU時才應該嘗試它：在CPU上是絕對棘手的。如果您無法在GPU上運行代碼，那麼前一個手法就是你要選的路。

> 因為模型的行為就像堆積木，所以你可以添加一個模型（像我們的conv_base）到Sequential模型，就像添加一個圖層一樣。所以你可以執行以下操作：

4 Listing 5.19 Plotting the results
5 Listing 5.20 Adding a densely co
6 Listing 5.21 Training the model er

In [8]:
```
#移花+接木
from keras import models
from keras import layers

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'))
```

In [9]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 4, 4, 512)         14714688

 flatten (Flatten)           (None, 8192)              0

 dense_2 (Dense)             (None, 256)               2097408

 dense_3 (Dense)             (None, 1)                 257

=================================================================
Total params: 16,812,353
```

```
Trainable params: 16,812,353
Non-trainable params: 0
_________________________________________________________________
```

如您所見，VGG16的“卷積基底”有14,714,688個參數，這非常大。我們上面添加的分類器有200萬個參數。

在編譯和訓練我們的模型之前，一個非常重要的事情是凍結“卷積基底”。 “凍結”一層或多層圖層意味著在訓練期間防止其權重被更新。如果我們不這樣做，那麼以前在“卷積基底”上學到的特徵將在訓練期間被修改。由於頂層的Dense層會被隨機初始化，因此非常大的權重更新將通過網絡重新進行傳播， 也會破壞以前學習的結果。

在Keras中，通過將它的trainable屬性設置為False來凍結網絡：

In [10]:
```
# 看一下“凍結前”有多少可以被訓練的權重
print('This is the number of trainable weights '
      'before freezing the conv base:', len(model.trainable_weights))
```

```
This is the number of trainable weights before freezing the conv base: 30
```

In [11]:
```
# “凍結”卷積基底
conv_base.trainable = False
# 再看一下“凍結後”有多少可以被訓練的權重
print('This is the number of trainable weights after freezing the conv base:', len(model.trainable_weig
```



```
This is the number of trainable weights after freezing the conv base: 4
```

通過這種設置，只有來自我們添加的兩個Dense層的權重將被訓練。這是總共有四個權重的張量：每層兩個（主要權重矩陣和偏差向量）。請注意，為了使這些更改生效，我們必須首先編譯模型。

如果您在編譯後修改了權重可訓練性，那麼應該重新編譯模型，否則這些更改將被忽略。

現在我們可以開始訓練我們的模型，使用與我們前面的例子中使用相同的數據增強配置：

6 Listing 5.21 Training the model end to end with a frozen convolutional base

In [12]:
```
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')
# 請注意：驗證用的資料不要進行資料的增強
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    # 圖像資料的目錄
        train_dir,
    # 設定圖像的高(height)與寬(width)
        target_size=(150, 150),
        batch_size=20,
     # 因為我們的目標資料集只有兩類(cat & dog)
        class_mode='binary')

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

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=2e-5),
              metrics=['acc'])
```

```
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
```

```
C:\Users\Huang\anaconda3\lib\site-packages\keras\optimizers\optimizer_v2\rmsprop.py:143: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super().__init__(name, **kwargs)
```

In [13]:

```
history = model.fit_generator(
      train_generator,
      steps_per_epoch=100,
      epochs=30,
      validation_data=validation_generator,
      validation_steps=50)
```

```
C:\Users\Huang\AppData\Local\Temp\ipykernel_11644\3743641980.py:1: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  history = model.fit_generator(
```

```
Epoch 1/30
100/100 [==============================] - 358s 4s/step - loss: 0.5787 - acc: 0.7015 - val_loss: 0.4464 - val_acc: 0.8140
Epoch 2/30
100/100 [==============================] - 351s 4s/step - loss: 0.4698 - acc: 0.7840 - val_loss: 0.3688 - val_acc: 0.8400
Epoch 3/30
100/100 [==============================] - 352s 4s/step - loss: 0.4250 - acc: 0.8210 - val_loss: 0.3304 - val_acc: 0.8690
Epoch 4/30
100/100 [==============================] - 355s 4s/step - loss: 0.4004 - acc: 0.8195 - val_loss: 0.3071 - val_acc: 0.8800
Epoch 5/30
100/100 [==============================] - 333s 3s/step - loss: 0.3782 - acc: 0.8415 - val_loss: 0.2952 - val_acc: 0.8830
Epoch 6/30
100/100 [==============================] - 332s 3s/step - loss: 0.3775 - acc: 0.8260 - val_loss: 0.2840 - val_acc: 0.8810
Epoch 7/30
100/100 [==============================] - 332s 3s/step - loss: 0.3622 - acc: 0.8435 - val_loss: 0.2705 - val_acc: 0.8840
Epoch 8/30
100/100 [==============================] - 2133s 22s/step - loss: 0.3629 - acc: 0.8340 - val_loss: 0.2709 - val_acc: 0.8930
Epoch 9/30
100/100 [==============================] - 364s 4s/step - loss: 0.3466 - acc: 0.8455 - val_loss: 0.2683 - val_acc: 0.8920
Epoch 10/30
100/100 [==============================] - 366s 4s/step - loss: 0.3270 - acc: 0.8475 - val_loss: 0.2663 - val_acc: 0.8970
Epoch 11/30
100/100 [==============================] - 357s 4s/step - loss: 0.3219 - acc: 0.8625 - val_loss: 0.2569 - val_acc: 0.8970
Epoch 12/30
100/100 [==============================] - 356s 4s/step - loss: 0.3303 - acc: 0.8585 - val_loss: 0.2669 - val_acc: 0.8950
Epoch 13/30
100/100 [==============================] - 357s 4s/step - loss: 0.3183 - acc: 0.8620 - val_loss: 0.2508 - val_acc: 0.8900
Epoch 14/30
100/100 [==============================] - 355s 4s/step - loss: 0.3214 - acc: 0.8575 - val_loss: 0.2481 - val_acc: 0.8870
Epoch 15/30
100/100 [==============================] - 354s 4s/step - loss: 0.3130 - acc: 0.8680 - val_loss: 0.2440 - val_acc: 0.8910
Epoch 16/30
100/100 [==============================] - 358s 4s/step - loss: 0.2972 - acc: 0.8735 - val_loss: 0.2587 - val_acc: 0.8930
Epoch 17/30
100/100 [==============================] - 356s 4s/step - loss: 0.3053 - acc: 0.8670 - val_loss: 0.2539 - val_acc: 0.8880
Epoch 18/30
100/100 [==============================] - 360s 4s/step - loss: 0.3099 - acc: 0.8580 - val_loss: 0.2444 - val_acc: 0.8950
Epoch 19/30
100/100 [==============================] - 358s 4s/step - loss: 0.3040 - acc: 0.8655 - val_loss: 0.2455 - val_acc: 0.8990
Epoch 20/30
100/100 [==============================] - 508s 5s/step - loss: 0.3135 - acc: 0.8610 - val_loss: 0.2436 - val_acc: 0.8990
Epoch 21/30
100/100 [==============================] - 331s 3s/step - loss: 0.2996 - acc: 0.8720 - val_loss: 0.2572 - val_acc: 0.8960
Epoch 22/30
100/100 [==============================] - 357s 4s/step - loss: 0.2939 - acc: 0.8730 - val_loss: 0.2442 - val_acc: 0.8960
Epoch 23/30
100/100 [==============================] - 353s 4s/step - loss: 0.2934 - acc: 0.8740 - val_loss: 0.2487 - val_acc: 0.8930
Epoch 24/30
100/100 [==============================] - 352s 4s/step - loss: 0.2902 - acc: 0.8800 - val_loss: 0.2779 - val_acc: 0.8750
Epoch 25/30
100/100 [==============================] - 352s 4s/step - loss: 0.2959 - acc: 0.8745 - val_loss: 0.2463 - val_acc: 0.8950
Epoch 26/30
100/100 [==============================] - 351s 4s/step - loss: 0.2853 - acc: 0.8755 - val_loss: 0.2432 - val_acc: 0.8960
Epoch 27/30
100/100 [==============================] - 351s 4s/step - loss: 0.2737 - acc: 0.8815 - val_loss: 0.2479 - val_acc: 0.9000
Epoch 28/30
100/100 [==============================] - 354s 4s/step - loss: 0.2762 - acc: 0.8810 - val_loss: 0.2441 - val_acc: 0.8970
Epoch 29/30
100/100 [==============================] - 353s 4s/step - loss: 0.2680 - acc: 0.8835 - val_loss: 0.2427 - val_acc: 0.8990
Epoch 30/30
100/100 [==============================] - 351s 4s/step - loss: 0.2812 - acc: 0.8765 - val_loss: 0.2418 - val_acc: 0.8940
```

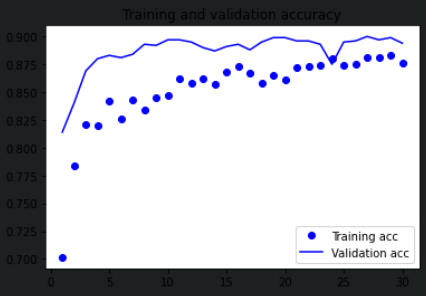
In [16]:

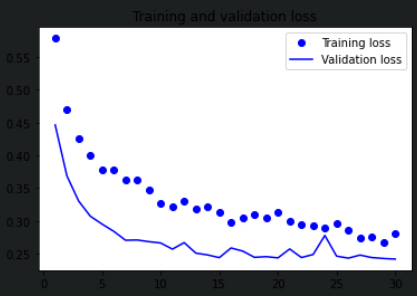
```
model.save('cats_and_dogs_small_3.h5') # 把模型儲存到檔案
```

In [14]:

```
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(1, len(acc) + 1)
```

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

plt.figure()

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



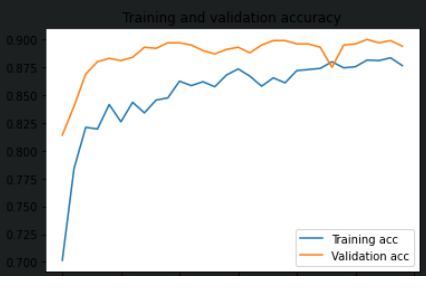


4 Listing 5.19 Plotting the results
5 Listing 5.20 Adding a densely con
6 Listing 5.21 Training the model e

In [15]:

```
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()
```


4 Listing 5.19 Plotting the results
5 Listing 5.20 Adding a densely con
6 Listing 5.21 Training the model e



5 Listing 5.20 Adding a densely con
6 Listing 5.21 Training the model e