20200228_notes

February 28, 2020

1 Introduction

Let's explore the use of transfer learning for the "Cats vs. Dogs" example from Chapter 5 of Chollet. Our data are color photos of cats and dogs, and our goal is to classify a photo according to which kind of animal it has.

I'm basically following the code from Chapter 5 with a few minor modifications.

1.1 Importing VGG16

The following code imports and loads the VGG16 model with its estimated parameters.

[4]:
```
from keras.applications import VGG16

vgg16_full_model = VGG16(
  weights='imagenet',
  include_top=True,
  input_shape=(224, 224, 3))

vgg16_conv_base = VGG16(
  weights='imagenet',
  include_top=False,
  input_shape=(150, 150, 3))
```

Here are summaries of `vgg16_full_model` and `vgg16_conv_base`. Note which layers from the full model are not included in the convolutional base.

[5]:
```
print(vgg16_full_model.summary())
print(vgg16_conv_base.summary())
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
```

```
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 1000)              4097000
=================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
_________________________________________________________________
None
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (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
_________________________________________________________________
None
```

1.2 Approach 1: Feature Extraction

We do three steps: 1. Make "predictions" from the convolutional base. The results are the $4 \times 4 \times 512$ outputs from the last layer of VGG16. 2. Flatten the outputs from VGG16. These are our inputs for a model we will fit, instead of the actual pictures. 3. Fit a model using the image representations from VGG16 as inputs.

Step 1: "predictions" from the convolutional base.

[10]:
```
datagen = ImageDataGenerator(rescale=1./255)
batch_size = 20

def extract_features(directory, sample_count):
    features = np.zeros(shape=(sample_count, 4, 4, 512))
    labels = np.zeros(shape=(sample_count))
    generator = datagen.flow_from_directory(
        directory,
        target_size=(150, 150),
        batch_size=batch_size,
        class_mode='binary')
    i = 0
    for inputs_batch, labels_batch in generator:
        features_batch = vgg16_conv_base.predict(inputs_batch)
        features[i * batch_size : (i + 1) * batch_size] = features_batch
        labels[i * batch_size : (i + 1) * batch_size] = labels_batch
        i += 1
        if i * batch_size >= sample_count:
            break
    return features, labels

train_features, train_labels = extract_features(train_dir, 2000)
validation_features, validation_labels = extract_features(validation_dir, 1000)
test_features, test_labels = extract_features(test_dir, 1000)
```

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

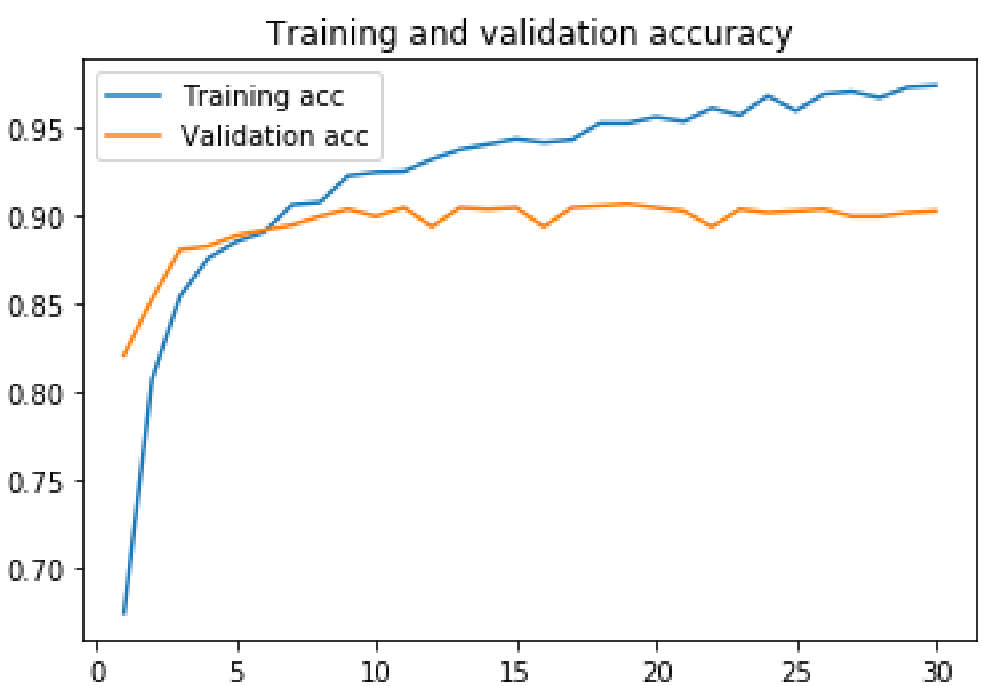
Step 2: Flatten

[0]:
```
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))
```

Step 3: Fit a model that takes activations from last convolutional layer of VGG16 as inputs

[12]:
```
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'])

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

```
Train on 2000 samples, validate on 1000 samples
Epoch 1/30
2000/2000 [==============================] - 1s 501us/step - loss: 0.6048 - acc:
0.6745 - val_loss: 0.4512 - val_acc: 0.8210

... a bunch of epochs...

Epoch 30/30
2000/2000 [==============================] - 1s 302us/step - loss: 0.0873 - acc:
0.9745 - val_loss: 0.2380 - val_acc: 0.9030
```



1.3 Approach 2: Transfer Learning

We do this by directly adding dense layers onto the VGG16 convolutional base. We update weights only for the new dense layers.

[0]:
```
model = models.Sequential()
model.add(vgg16_conv_base)
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

Set parameters in the convolutional base so they are not "trainable"; the weight estimates from VGG16 will not be updated.

[55]:
```
num_trainable = np.sum([np.prod(w.shape.as_list()) for w in model.
 ↪trainable_weights])
print('This is the number of trainable weights '
         'before freezing the conv base:', num_trainable)
vgg16_conv_base.trainable = False
num_trainable = np.sum([np.prod(w.shape.as_list()) for w in model.
 ↪trainable_weights])
print('This is the number of trainable weights '
          'after freezing the conv base:', num_trainable)
```

```
This is the number of trainable weights before freezing the conv base: 16812353
This is the number of trainable weights after freezing the conv base: 2097665
```

[56]:
```
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 4, 4, 512)         14714688
_________________________________________________________________
flatten_4 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_9 (Dense)              (None, 256)               2097408
_________________________________________________________________
dense_10 (Dense)             (None, 1)                 257
=================================================================
Total params: 16,812,353
Trainable params: 2,097,665
Non-trainable params: 14,714,688
_________________________________________________________________
```

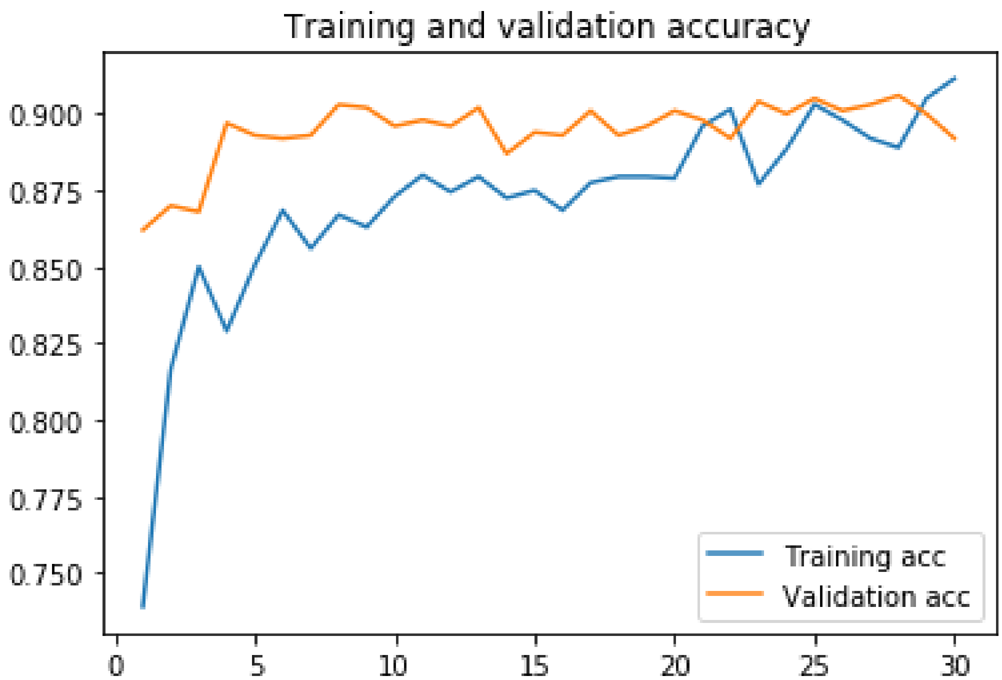
Fit the model using data augmentation.

```
[57]: train_datagen = ImageDataGenerator(
          rescale=1./255,
          rotation_range=40,
          width_shift_range=0.2,
          height_shift_range=0.2,
          shear_range=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=40,
              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.Adam(lr=2e-4),
                    metrics=['acc'])

      tic = time.time()
      history = model.fit_generator(
            train_generator,
            steps_per_epoch=50,
            epochs=30,
            validation_data=validation_generator,
            validation_steps=50)
      toc = time.time()
      toc - tic
```

```
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
Epoch 1/30
50/50 [==============================] - 21s 426ms/step - loss: 0.5106 - acc:
0.7390 - val_loss: 0.3114 - val_acc: 0.8620

... so many epochs ...
```

```
Epoch 30/30
50/50 [==============================] - 17s 347ms/step - loss: 0.2212 - acc:
0.9115 - val_loss: 0.2527 - val_acc: 0.8920
```

[57]: 528.1107420921326



I did a little more training of this model and validation set accuracy improved ever so slightly. Cut for brevity.

1.4 Approach 3: Fine Tuning

We'll now unfreeze the last block of convolutional layers in the VGG16 model, and update those weights as well.

[67]:
```
vgg16_conv_base.summary()
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (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: 7,079,424
Non-trainable params: 7,635,264
_________________________________________________________________
```

We'll set the weights to be trainable starting with the 'block5_conv1' layer.

[0]:
```
vgg16_conv_base.trainable = True

set_trainable = False
for layer in vgg16_conv_base.layers:
    if layer.name == 'block5_conv1':
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False
```

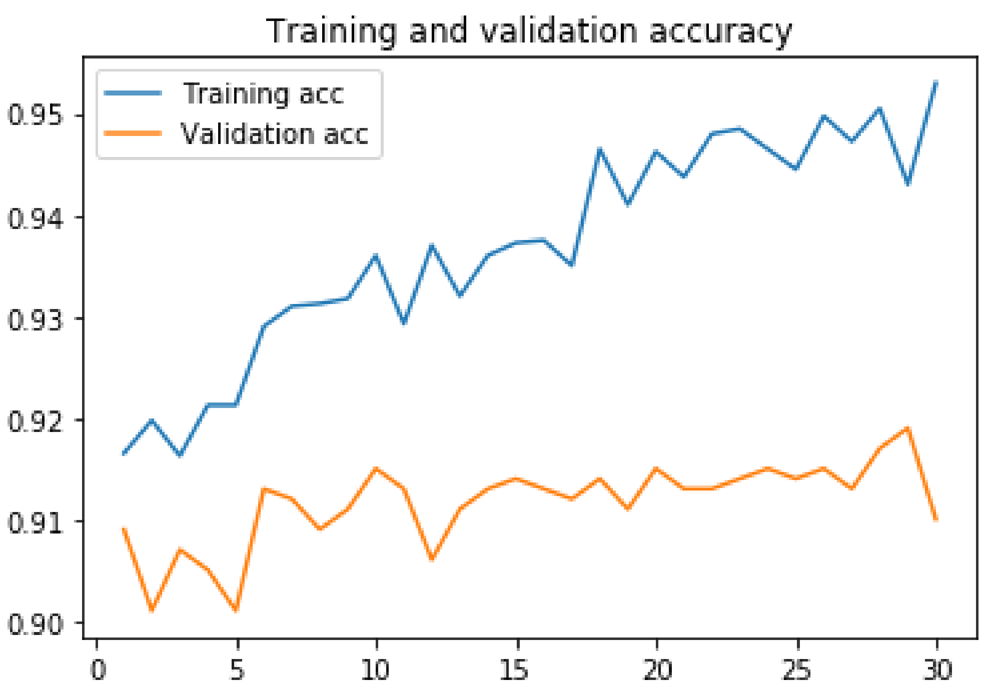
[68]:
```
model.compile(loss='binary_crossentropy',
              optimizer=optimizers.Adam(lr=1e-6),
              metrics=['acc'])
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 4, 4, 512)         14714688
_________________________________________________________________
flatten_4 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_9 (Dense)              (None, 256)               2097408
_________________________________________________________________
dense_10 (Dense)             (None, 1)                 257
=================================================================
Total params: 16,812,353
Trainable params: 9,177,089
Non-trainable params: 7,635,264
_________________________________________________________________
```

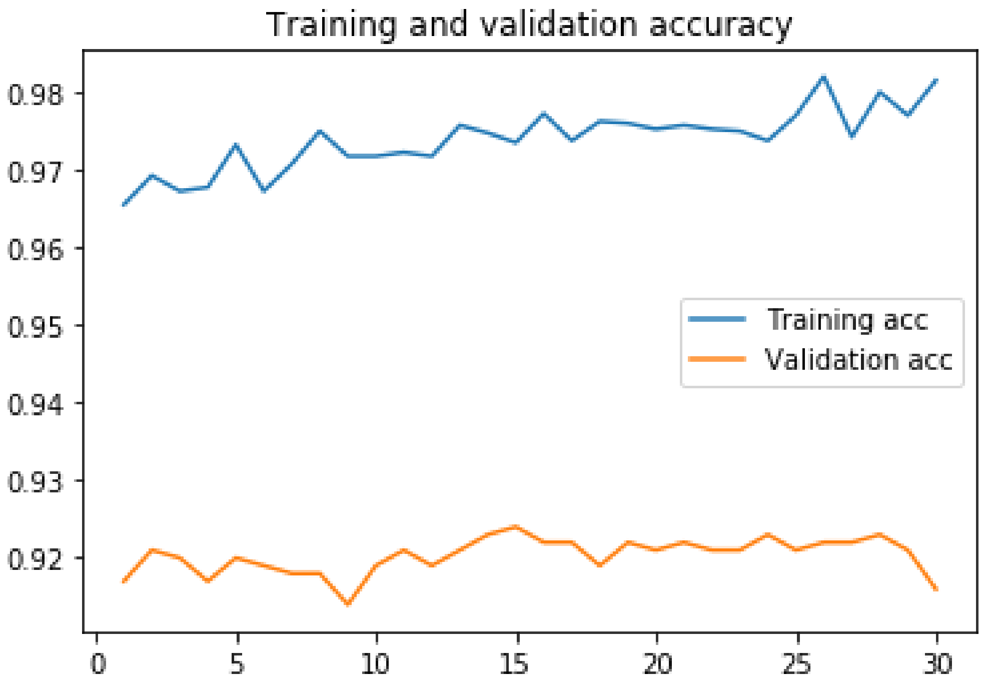
Run estimation. We're continuing from the weight estimates as they were at the end of Approach 2.

[69]:
```
tic = time.time()
even_even_more_history = model.fit_generator(
      train_generator,
      steps_per_epoch=100,
      epochs=30,
      validation_data=validation_generator,
      validation_steps=50)
toc = time.time()
toc - tic
```

```
Epoch 1/30
100/100 [==============================] - 36s 357ms/step - loss: 0.1960 - acc:
0.9165 - val_loss: 0.2336 - val_acc: 0.9090

... all the epochs ...

Epoch 30/30
100/100 [==============================] - 35s 348ms/step - loss: 0.1211 - acc:
0.9530 - val_loss: 0.2168 - val_acc: 0.9100
```



After reducing the learning rate and training for an additional 60 epochs, I eked out another 1% classification accuracy.



[0]:
```
model.save('/content/drive/My Drive/stat344ne_cats_and_dogs_small/
↪cats_and_dogs_from_vgg16_finetuning_final.h5')
```

I've clearly overfit, and possibly could get another percentage point of accuracy out by regularizing more carefully.

However, I'm feeling emotionally ready to look at test set performance.

[76]:
```
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)
```

```
Found 1000 images belonging to 2 classes.
test acc: 0.918999993801117
```

Take aways:

- This was a lot more work, but transfer learning resulted in better performance than what we achieved without transfer learning (around 80%).
- Relative to using VGG16 for just feature extraction, fine tuning helped a little in terms of absolute accuracy (our model with feature extraction was already pretty good), but a lot in terms of percentage unexplained.
- Transfer learning is the norm, not the exception.