PythonとKerasによるディープラーニング

第5章 コンピュータビジョンのためのディープラーニング
5.3　学習済みのCNNを使用する

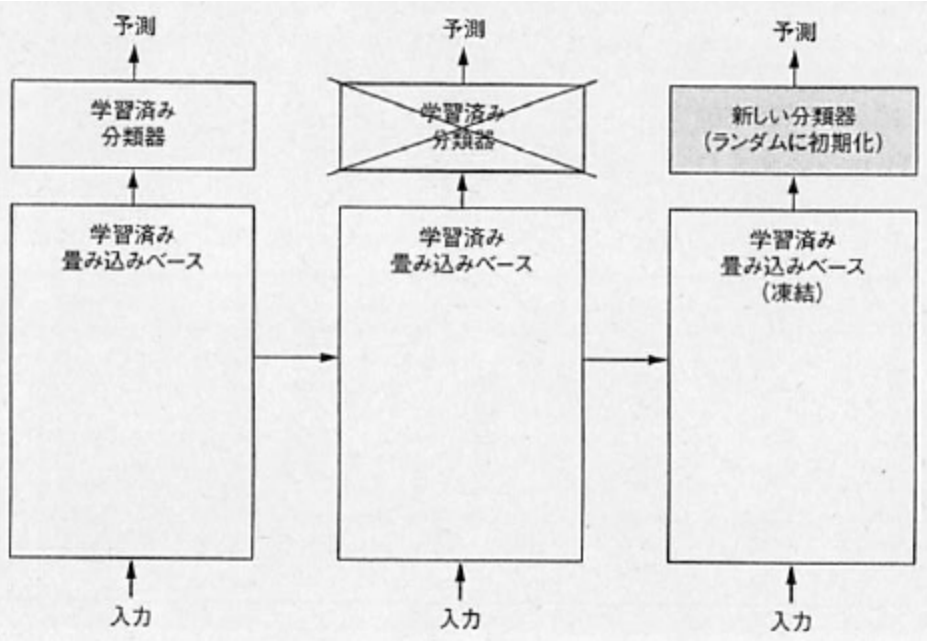
19nm715n ZHANGYIHANG

学習済みのネットワークとは、大規模なデータセットで訓練された後、保存されたネットワークのことであり、一般に大規模な画像分類タスクで作成されます。これは小さな画像データセットを用いたディープラーニングに非常に効果的なアプローチです。例えば、ImageNetのデータセットは犬と猫の分類問題に良い性能が得られることが期待できます。

学習済みのネットワークをしようする方法には、特徴抽出とファインチューニングの2つがあります。

5.3.1　特徴抽出

　特徴抽出は、１つ前のネットワークが学習した表現に基づいて、新しいサンプルがら興味深い特徴量を抽出する手法です。CNNの特徴抽出は、学習済みネットワークの畳み込みベースで新しいデータを処理し、その出力に基づいて新しい分類器を訓練する手順で構成されます。



特定の畳み込み層によって抽出された表現の汎用性の度合いは、その層がモデルのどれくらい深いところにあるかに依存します。

• エッジ • 色 • テクスチャ	• 猫の耳 • 犬の目
汎用性高い	抽象的な概念

リスト 5-16：VGG16 モデルの畳み込みベースのインスタンス化

```
from keras.applications import VGG16

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

最終的な特徴マップの形状は(4,4,512)です。ここから先へ進む方法は2つあります。

- 新しいデータセットで畳み込みベースを実行し、その出力をディスク上のNumPy配列に書き込み、このデータをスタンドアロンの全結合分類器の入力として使用
- 最後にDense層を追加することでモデルを拡張し、最初から最後までのすべての処理を入力データで実行

```
conv_base.summary()
```

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
```

データ拡張を行わない高速な特徴抽出

リスト 5-17：学習済みの畳み込みベースを使って特徴量を抽出

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

# 5.2.2項でsmallデータセットを格納したディレクトリへのパスであることに注意
base_dir = '/Users/fchollet/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

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 = 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:
            # ジェネレータはデータを無限ループで生成するため、
            # 画像をひととおり処理したらbreakしなければならない
            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)
```

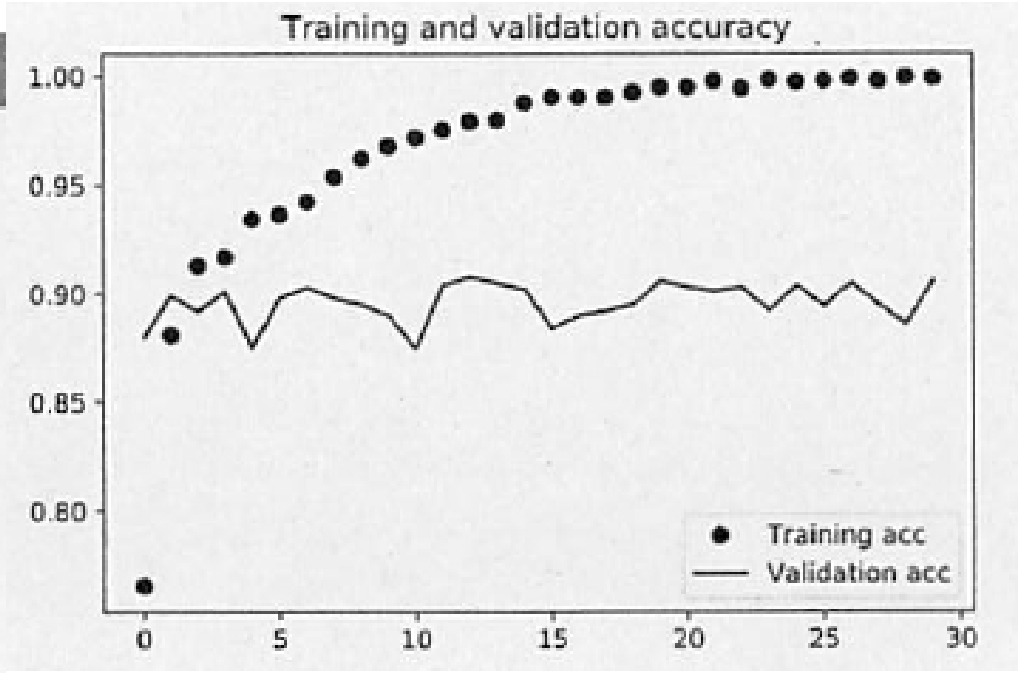
```
train_features = np.reshape(train_features, (2000, 4 * 4 * 512))
validation_features = np.reshape(validation_features, (1000, 4 * 4 * 512))
test_features = np.reshape(test_features, (1000, 4 * 4 * 512))
```

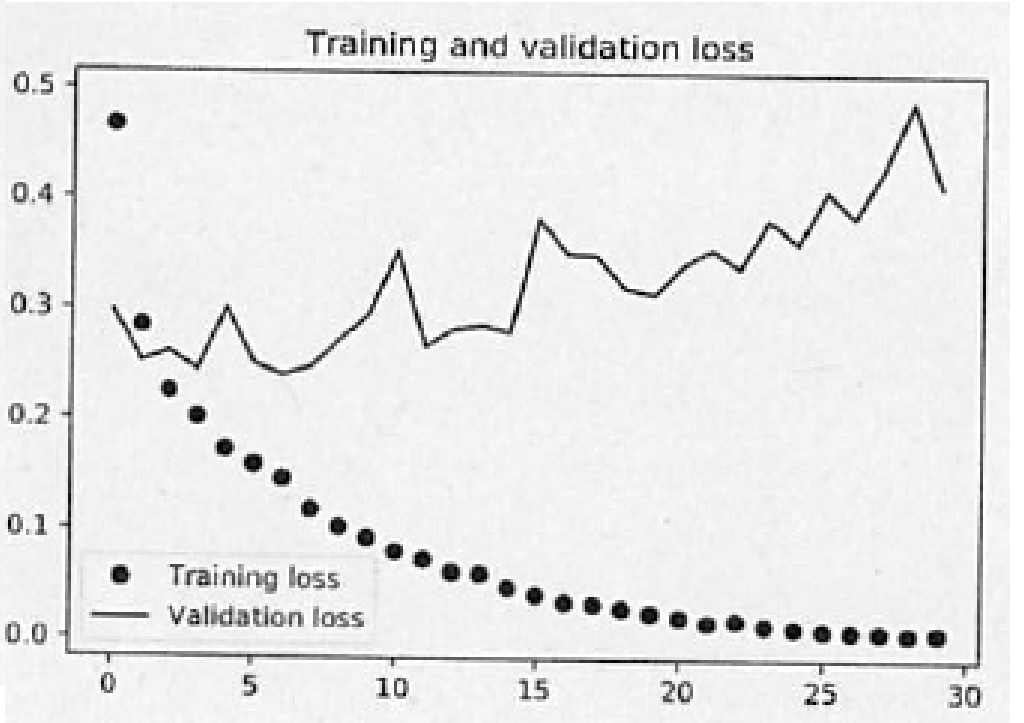
リスト 5-18：全結合分類器の定義と訓練

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

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))
```

リスト 5-19：結果をプロット

```
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))

# 正解率をプロット
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()

# 損失値をプロット
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()
```





データ拡張を行う特徴抽出

リスト 5-20：畳み込みベースに全結合分類器を追加

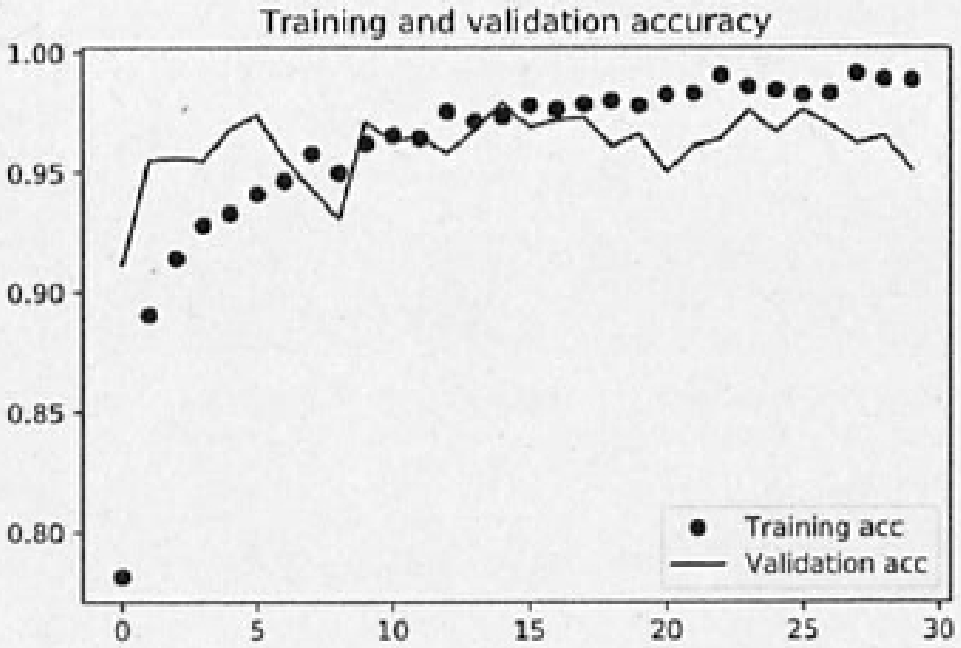
```
from keras import models
from keras import layers

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'))
```

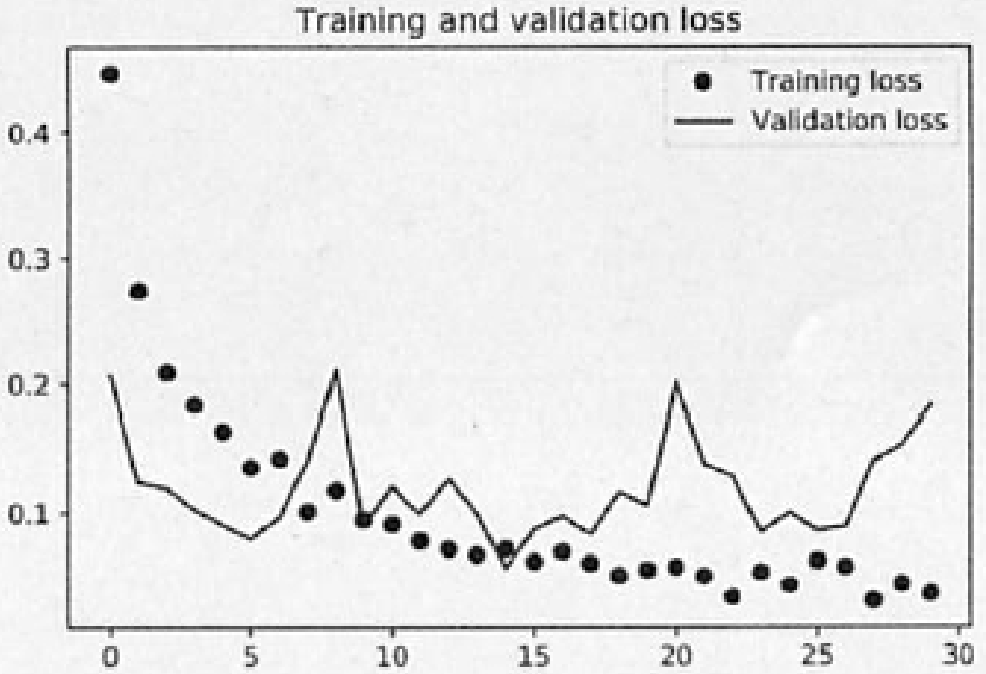
```
>>> model.summary()
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 4, 4, 512)         14714688
_________________________________________________________________
flatten_1 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               2097408
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 257
=================================================================
Total params: 16,812,353
Trainable params: 16,812,353
Non-trainable params: 0
```

　モデルに追加している分類器には、２００万個のパラメータがあります、訓練する前に、畳み込みベースを凍結することが重要となります。凍結とは、それらの層の重みが訓練中に更新されなくなることです。

リスト 5-21：凍結された畳み込みベースを使ってモデル全体を訓練

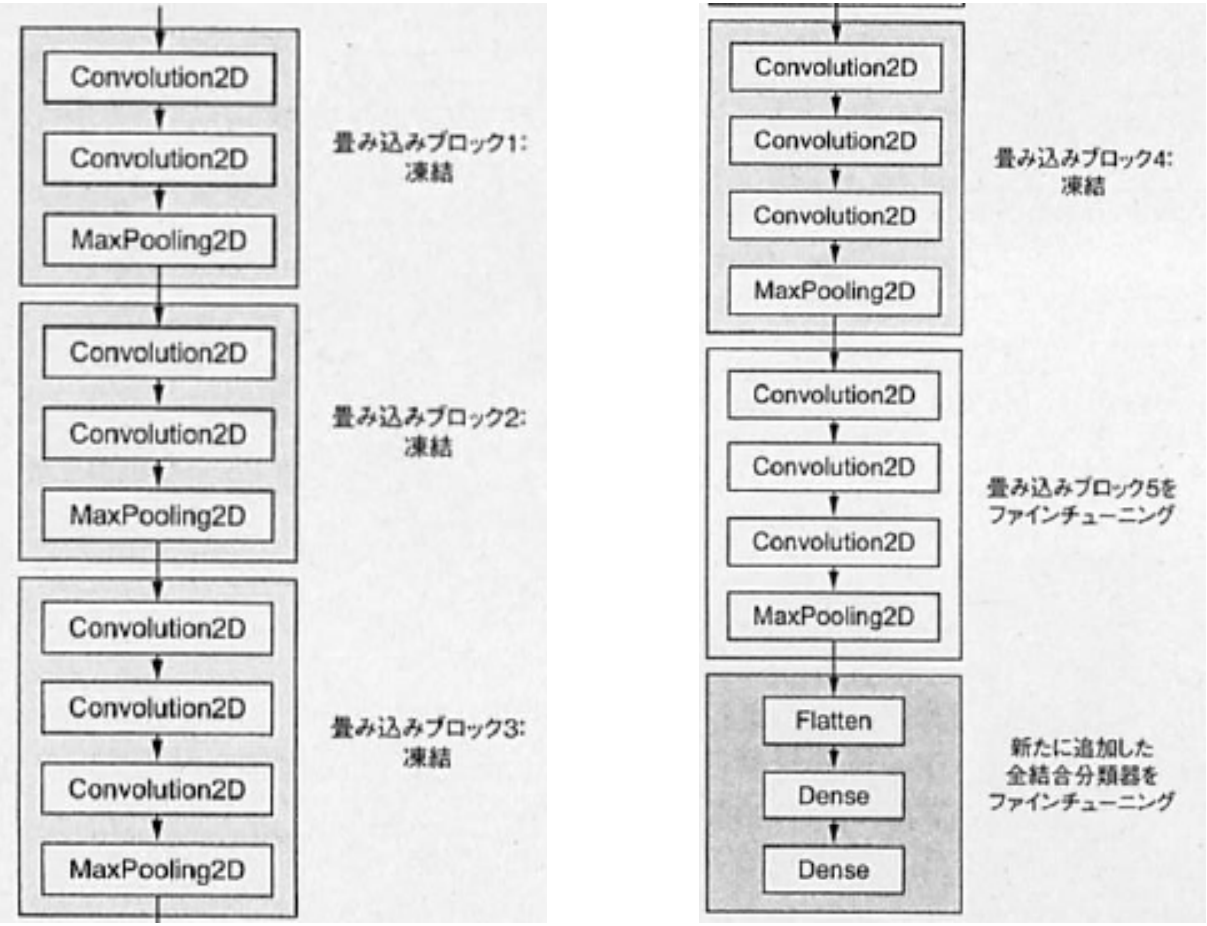
```
from keras.preprocessing.image import ImageDataGenerator

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')

# 検証データは水増しすべきではないことに注意
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
      train_dir,                  # ターゲットディレクトリ
      target_size=(150, 150),     # すべての画像を150×150に変更
      batch_size=20,              # バッチサイズ
      class_mode='binary')        # 損失関数としてbinary_crossentropyを使用するため、
                                  # 二値のラベルが必要

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'])

history = model.fit_generator(train_generator,
                              steps_per_epoch=100,
                              epochs=30,
                              validation_data=validation_generator,
                              validation_steps=50,
                              verbose=2)
```





5.3.2　ファインチューニング

　ファインチューニングとは、特徴抽出に使用される凍結された畳み込みベースの出力側の層をいくつか解凍し、モデルの新しく追加された部分と解凍した層の両方で訓練を行う、という仕組みになっています。



ファインチューニングの手順

1. 訓練済みのベースネットワークの最後にカスタムネットワークを追加する。
2. ベースネットワークを凍結する。
3. 追加した部分のくんれんを行う。
4. ベースネットワークの一部の層を解凍する。
5. 解凍した層と追加した部分の訓練を同時に行う。

畳み込みベースのアーキテクチャ

```
>>> conv_base.summary()
```

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 150, 150, 3)	0
block1_conv1 (Convolution2D)	(None, 150, 150, 64)	1792
block1_conv2 (Convolution2D)	(None, 150, 150, 64)	36928
block1_pool (MaxPooling2D)	(None, 75, 75, 64)	0
block2_conv1 (Convolution2D)	(None, 75, 75, 128)	73856
block2_conv2 (Convolution2D)	(None, 75, 75, 128)	147584
block2_pool (MaxPooling2D)	(None, 37, 37, 128)	0
block3_conv1 (Convolution2D)	(None, 37, 37, 256)	295168
block3_conv2 (Convolution2D)	(None, 37, 37, 256)	590080
block3_conv3 (Convolution2D)	(None, 37, 37, 256)	590080
block3_pool (MaxPooling2D)	(None, 18, 18, 256)	0
block4_conv1 (Convolution2D)	(None, 18, 18, 512)	1180160
block4_conv2 (Convolution2D)	(None, 18, 18, 512)	2359808
block4_conv3 (Convolution2D)	(None, 18, 18, 512)	2359808
block4_pool (MaxPooling2D)	(None, 9, 9, 512)	0
block5_conv1 (Convolution2D)	(None, 9, 9, 512)	2359808
block5_conv2 (Convolution2D)	(None, 9, 9, 512)	2359808
block5_conv3 (Convolution2D)	(None, 9, 9, 512)	2359808
block5_pool (MaxPooling2D)	(None, 4, 4, 512)	0

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

ファインチューニングを行うのは最後の三つの畳み込み層です。

- 畳み込みベースの入力側の層は、より汎用的で再利用可能な特徴量をエンコードしている。これに対し、出力側の層は、より具体的特徴量をエンコードしている。

- 訓練の対象となるパラメータの数が増えれば増えるほど、過学習のリスクは高くなる。

リスト 5-22：最初から特定の層までをすべて凍結

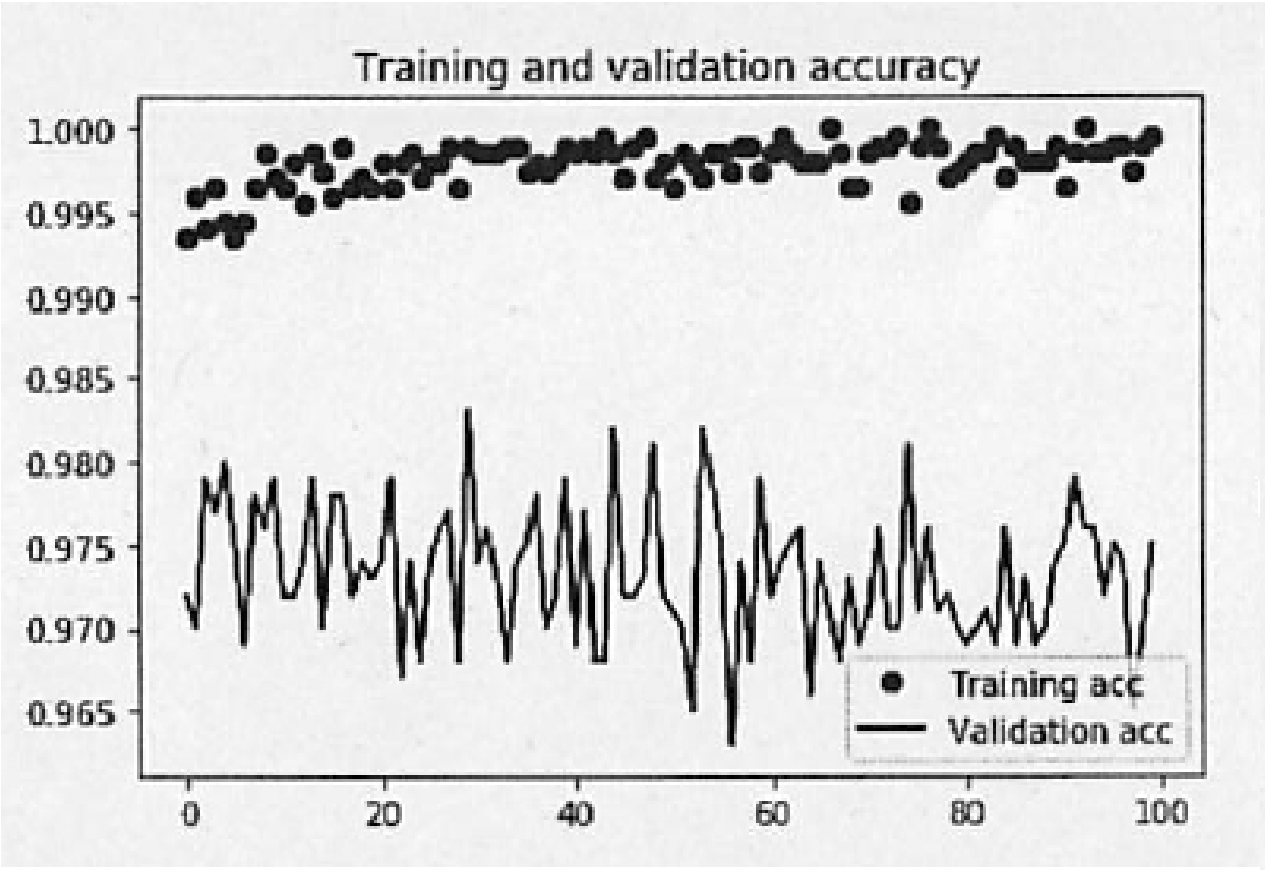
```
conv_base.trainable = True

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

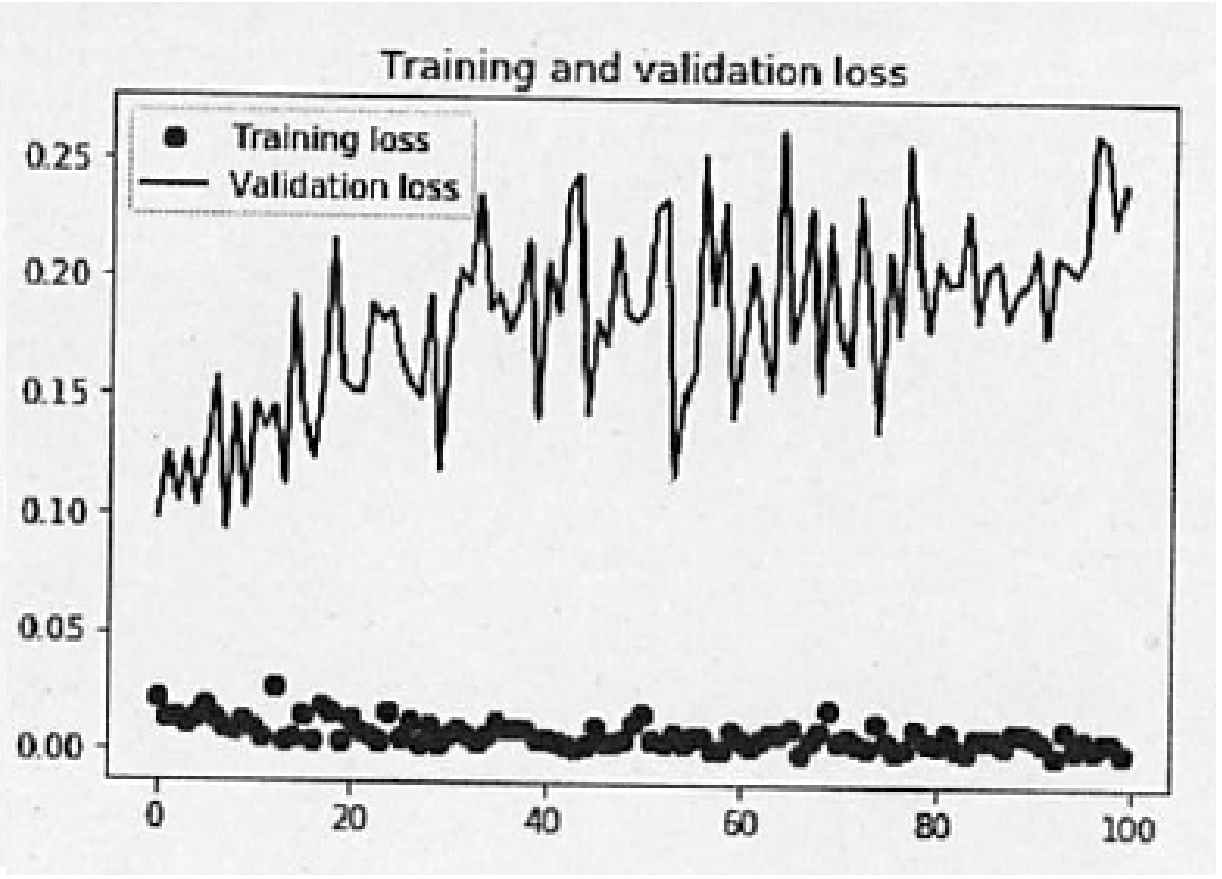
RMSpropオプティマイザとかなり低い学習率を使用します。
ファインチューニングを行う３つの表現に対する変更の大きさを制限したいからです。

リスト 5-23：モデルのファインチューニング

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

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

ファインチューニングを行った場合の正解率/損失値

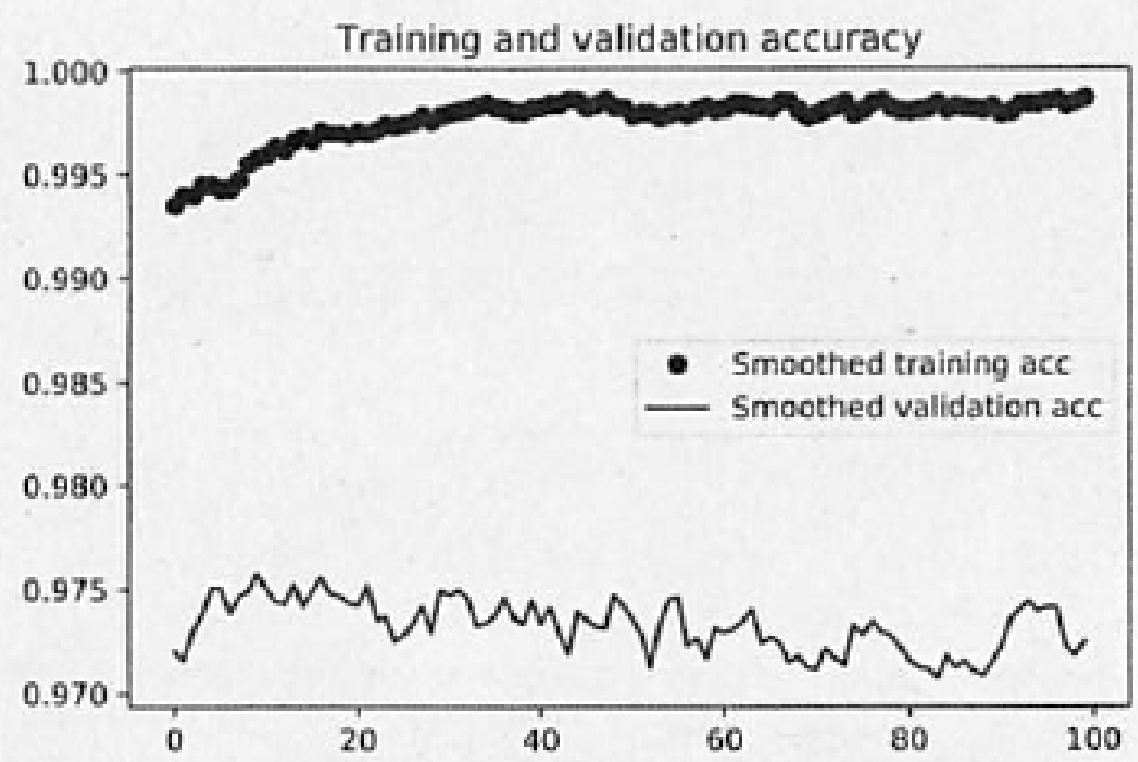


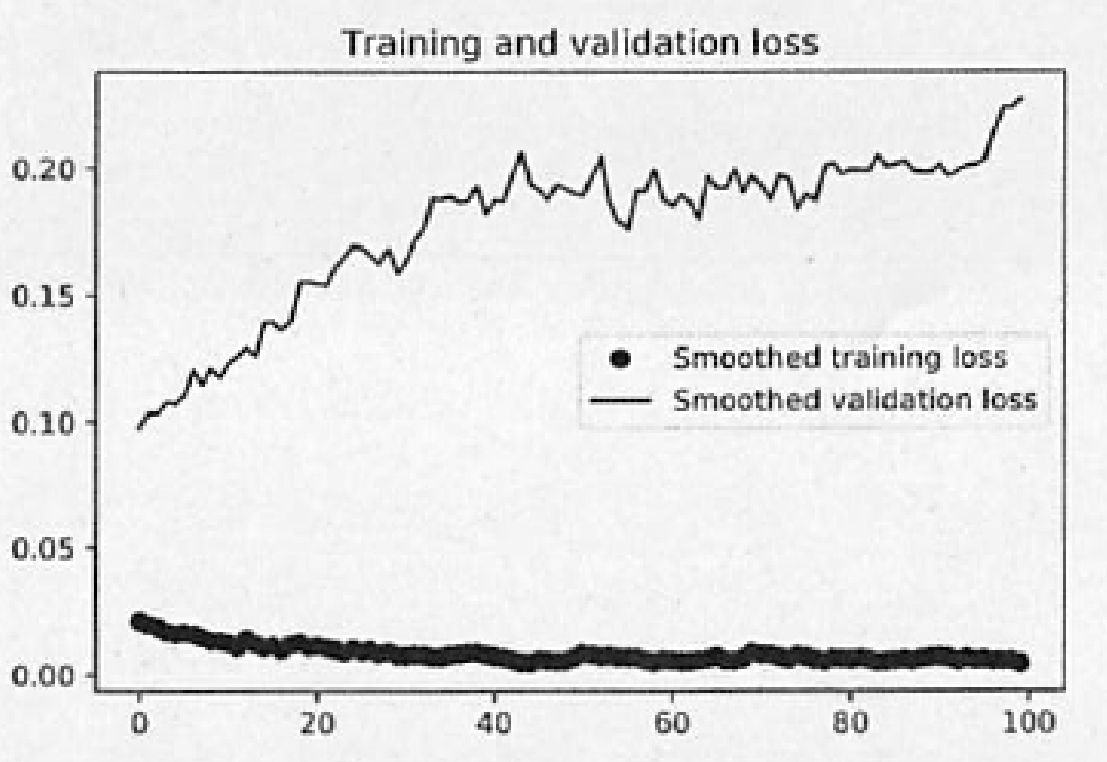


リスト 5-24：プロットのスムージング

```
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()
```

ファインチューニングを行った場合の正解率/損失値(スムージング後)





　ちなみに、この正解率は、このデータセットを使ったKaggleのコンペでは、トップクラスの１つの結果です。

5.3.3　まとめ

- CNNは、コンピュータビジョンのタスクに最適な機械学習モデルである、データセットが非常に小さい場合でも使用できます。
- データセットが小さい場合の主な課題は過学習です。
- 特徴抽出を利用すれば、既存のCNNを新しいデータセットで簡単に再利用できます。
- 特徴抽出の補完にはファインチューニングを利用できます。