PythonとKerasによるディープラーニング

5.3 学習済みのCNNを使用する

17T4014Y 伊藤 陽樹

学習済みのネットワーク

- 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. 一般に大規模な画像分類タスクで作成される.

- 小さな画像データセットを用いたDLに関して、効果的なアプローチ.
- この節では、VGG16アーキテクチャを使用.

- 学習済みのネットワークを使用する方法:
  - ✓**特徴抽出**(feature extraction)
  - ✓**ファインチューニング**(fine-tuning)

特徴抽出

- 1つ前のネットワークが学習した表現に基づいて、新しいサンプルから興味深い特徴量を抽出する手法.
- 画像分類に使用されるCNNは、一連のプーリング層と畳み込み層で始まり、全結合分類器で終わる.
  - ✓最初の部分は、モデルの**畳み込みベース**.
- 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.

特徴抽出

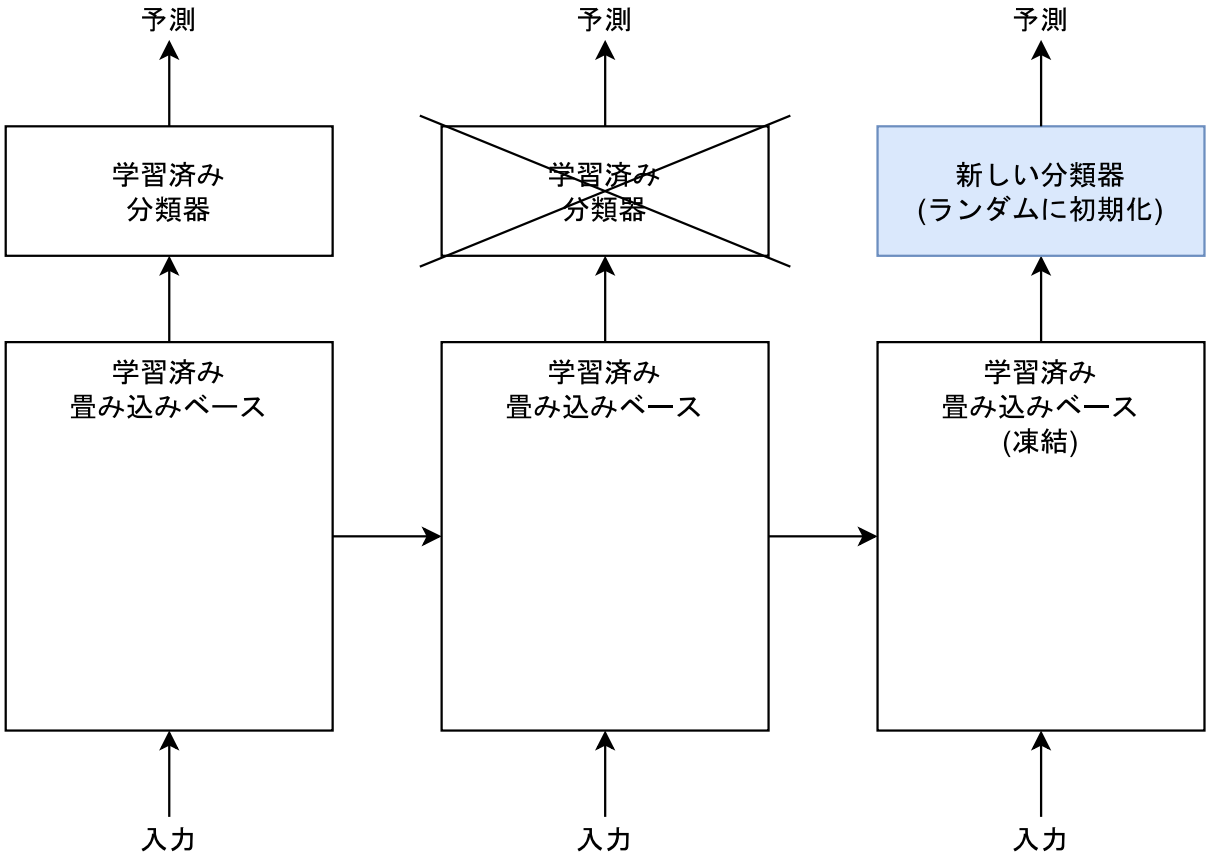

図: 畳み込みベースは同じままで、分類器を入れ替える

特徴抽出

- 畳み込みベースのみ再利用する.
  - ➢こっちの表現の方が汎用的で、再利用できる可能性が高いから.
  - ➢分類器の方の表現は、そのモデルが学習した一連のクラスに特化している.
  - ➢また、入力画像の「どこ」にオブジェクトが位置しているかの情報が含まれていない.

- 特定の畳み込み層によって抽出された表現の汎用性(再利用性)の度合いは、その層がモデルのどれくらいの深さにあるかに依存.
  - ✓元のモデルの訓練に使用されたデータセットと新しいデータセットが大きく異なる場合は、特徴抽出にモデルの最初の方にあるいくつかの層だけを使用する.

特徴抽出

- ImageNetで訓練されたVGG16ネットワークの畳み込みベースを使って犬と猫の画像から特徴量を抽出し、それらの特徴量に基づきDogs vs. Cats分類器の訓練を行う.

- VGG16モデルのインスタンス化

```
import keras
from keras.applications import VGG16

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

特徴抽出

- weights
  このモデルを初期化するための重みのチェックポイントを指定.
- include_top
  ネットワークの出力側にある全結合分類器を含めるかどうか. ここでは新しい分類器(クラスはcatとdogの2つだけ)を使用するため、含めない.
- input_shape
  ネットワークに供給する画像テンソルの形状.

- 最終的な特徴マップの形状は(4, 4, 512).

特徴抽出

- ここから先へ進む方法は以下の2通り.

  ✓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. (データ拡張は不可)

  ✓最後にDense層を追加することでモデル(conv_base)を拡張し、最初から最後まで全ての処理を入力データで実行. (データ拡張は可能)

データ拡張を行わない高速な特徴抽出(1つ目の方法)

- まず、画像とそれらのラベルをNumPy配列として抽出するために、ImageDataGeneratorのインスタンスを実行する. これらの画像から特徴量を抽出するには、conv_baseモデルのpredictメソッドを呼び出す.

```
# データ拡張を行わない特徴抽出
import keras
from keras.applications import VGG16

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

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

base_dir = '/Users/Youki Itoh/Downloads/cats_and_dogs'

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
```

データ拡張を行わない高速な特徴抽出(1つ目の方法)

```
20  def extract_features(directory, sample_count):
21      features = np.zeros(shape=(sample_count, 4, 4, 512))
22      labels = np.zeros(shape=(sample_count))
23      generator = datagen.flow_from_directory(
24          directory,
25          target_size=(150, 150),
26          batch_size=batch_size,
27          class_mode='binary')
28      i = 0
29      for inputs_batch, labels_batch in generator:
30          features_batch = conv_base.predict(inputs_batch)
31          features[i * batch_size : (i + 1) * batch_size] = features_batch
32          labels[i * batch_size : (i + 1) * batch_size] = labels_batch
33          i += 1
34          if i * batch_size >= sample_count:
35              # Note that since generators yield data indefinitely in a loop,
36              # we must `break` after every image has been seen once.
37              break
38      return features, labels
39
40  train_features, train_labels = extract_features(train_dir, 2000)
41  validation_features, validation_labels = extract_features(validation_dir, 1000)
42  test_features, test_labels = extract_features(test_dir, 1000)
43
44  train_features = np.reshape(train_features, (2000, 4 * 4 * 512))
45  validation_features = np.reshape(validation_features, (1000, 4 * 4 * 512))
46  test_features = np.reshape(test_features, (1000, 4 * 4 * 512))
```

- 抽出された特徴量の現時点の形状は(samples, 4, 4, 512).
- 全結合分類器に供給されるため、(samples, 8192)に平坦化.

データ拡張を行わない高速な特徴抽出(1つ目の方法)

- この時点で、新しい全結合分類器を定義し(注: 正規化としてドロップアウトを使用)、記録しておいたデータとラベルを使って訓練を行う.

```
48  from keras import models
49  from keras import layers
50  from keras import optimizers
51
52  model = models.Sequential()
53  model.add(layers.Dense(256, activation='relu', input_dim=4 * 4 * 512))
54  model.add(layers.Dropout(0.5))
55  model.add(layers.Dense(1, activation='sigmoid'))
56
57  model.compile(optimizer=optimizers.RMSprop(lr=2e-5),
58                loss='binary_crossentropy',
59                metrics=['accuracy'])
60
61  history = model.fit(train_features, train_labels,
62                      epochs=30,
63                      batch_size=20,
64                      validation_data=(validation_features, validation_labels))
```

データ拡張を行わない高速な特徴抽出(1つ目の方法)

```
66  import matplotlib.pyplot as plt
67
68  acc = history.history['accuracy']
69  val_acc = history.history['val_accuracy']
70  loss = history.history['loss']
71  val_loss = history.history['val_loss']
72
73  epochs = range(len(acc))
74
75  plt.plot(epochs, acc, 'bo', label='Training acc')
76  plt.plot(epochs, val_acc, 'b', label='Validation acc')
77  plt.title('Training and validation accuracy')
78  plt.legend()
79
80  plt.figure()
81
82  plt.plot(epochs, loss, 'bo', label='Training loss')
83  plt.plot(epochs, val_loss, 'b', label='Validation loss')
84  plt.title('Training and validation loss')
85  plt.legend()
86
87  plt.show()
```

データ拡張を行わない高速な特徴抽出(1つ目の方法)

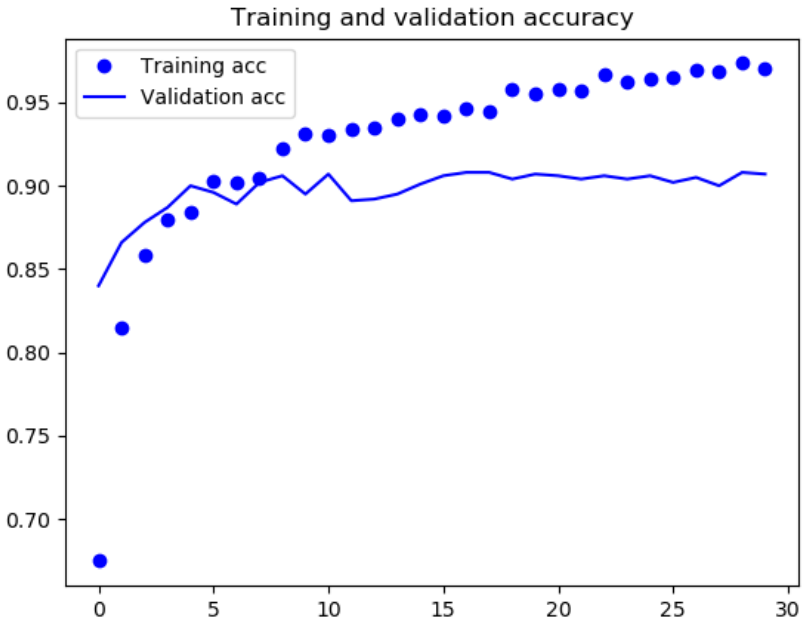

図: 単純な特徴抽出での訓練
データと検証データでの正解率

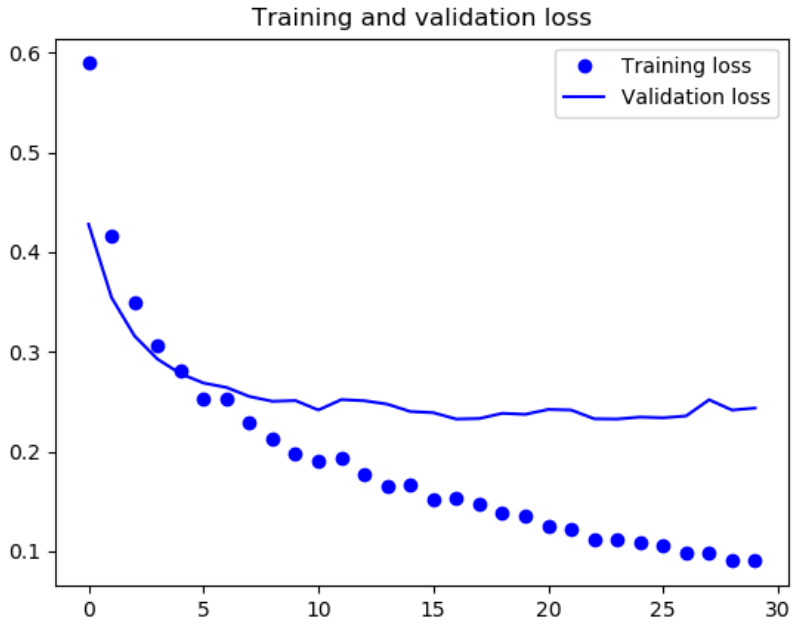

図: 単純な特徴抽出での訓練
データと検証データでの損失値

データ拡張を行う特徴抽出(2つ目の方法)

- conv_baseモデルを拡張し、最初から最後まで全ての処理を入力データで実行する.
- これらのモデルは層のように動作するため、層を追加するときと同じように(conv_baseなどのモデルを)Sequentialモデルに追加できる.

```
89   # データ拡張を行う特徴抽出
90   from keras import models
91   from keras import layers
92
93   model = models.Sequential()
94   model.add(conv_base)
95   model.add(layers.Flatten())
96   model.add(layers.Dense(256, activation='relu'))
97   model.add(layers.Dense(1, activation='sigmoid'))
```

データ拡張を行う特徴抽出(2つ目の方法)

- **凍結**(freezing):
  層の重みが訓練中に更新されなくなること.
- 層を凍結しない場合、畳み込みベースによって学習された表現が訓練中に変更されてしまう.
- モデルに追加されたDense層はランダムに初期化されるため、非常に大きな重みの更新がネットワークに伝播されると、学習済みの表現が実質的に破壊されてしまう.

➢モデルのコンパイルと訓練を行う前に、畳み込みベースを凍結する.

データ拡張を行う特徴抽出(2つ目の方法)

- ネットワークの凍結の後、モデルの訓練を開始.

```
 99  print('This is the number of trainable weights '
100        'before freezing the conv base:', len(model.trainable_weights))
101  conv_base.trainable = False
102  print('This is the number of trainable weights '
103        'after freezing the conv base:', len(model.trainable_weights))
104
105  from keras.preprocessing.image import ImageDataGenerator
106
107  train_datagen = ImageDataGenerator(
108          rescale=1./255,
109          rotation_range=40,
110          width_shift_range=0.2,
111          height_shift_range=0.2,
112          shear_range=0.2,
113          zoom_range=0.2,
114          horizontal_flip=True,
115          fill_mode='nearest')
116
117  # Note that the validation data should not be augmented!
118  test_datagen = ImageDataGenerator(rescale=1./255)
```

データ拡張を行う特徴抽出(2つ目の方法)

```
120  train_generator = train_datagen.flow_from_directory(
121          # This is the target directory
122          train_dir,
123          # All images will be resized to 150x150
124          target_size=(150, 150),
125          batch_size=20,
126          # Since we use binary_crossentropy loss, we need binary labels
127          class_mode='binary')
128
129  validation_generator = test_datagen.flow_from_directory(
130          validation_dir,
131          target_size=(150, 150),
132          batch_size=20,
133          class_mode='binary')
134
135  model.compile(loss='binary_crossentropy',
136                optimizer=optimizers.RMSprop(lr=2e-5),
137                metrics=['accuracy'])
138
139  history = model.fit_generator(
140          train_generator,
141          steps_per_epoch=100,
142          epochs=30,
143          validation_data=validation_generator,
144          validation_steps=50,
145          verbose=2)
```

データ拡張を行う特徴抽出(2つ目の方法)

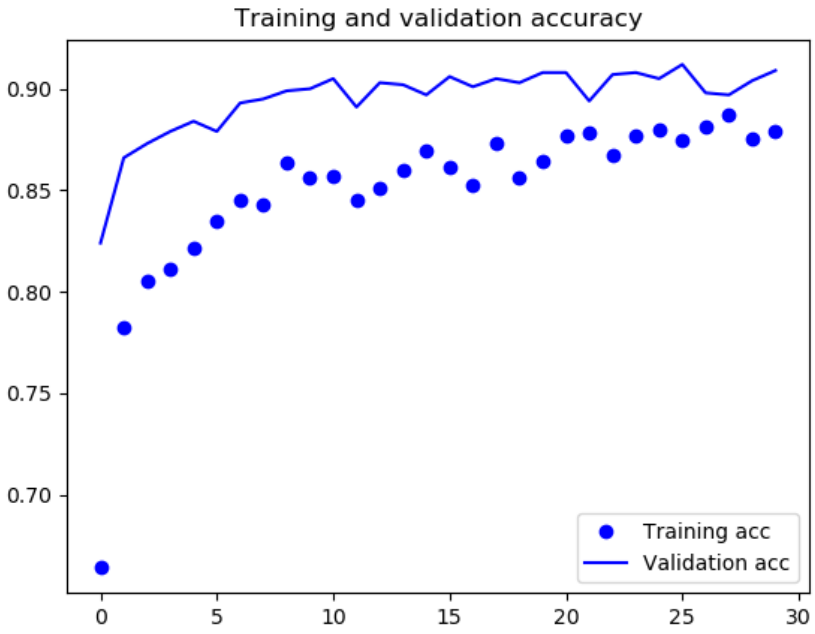

図: データ拡張を使った特徴抽出での訓練と検証の正解率

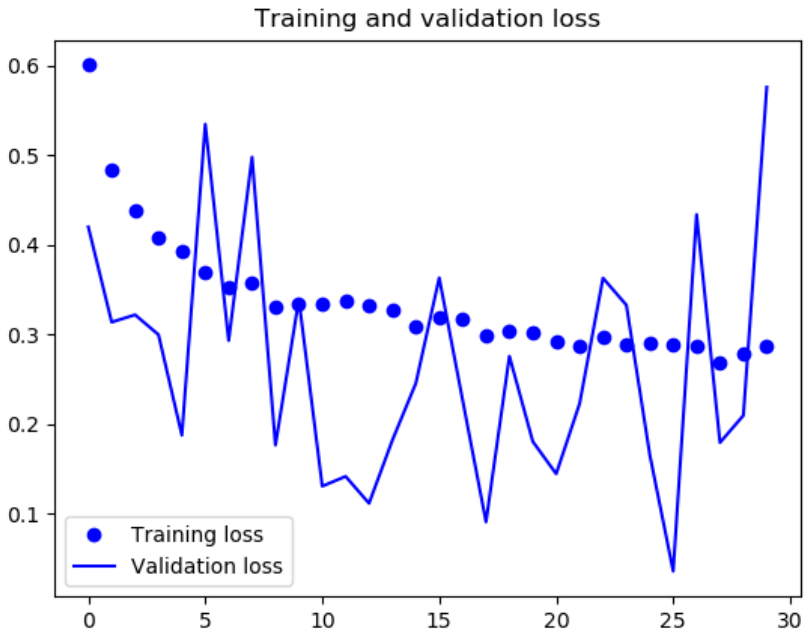

図: データ拡張を使った特徴抽出での訓練と検証の損失値

ファインチューニング

- **ファインチューニング**(fine-tuning):
  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幾つか解凍し、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.

- ランダムに初期化された分類器の訓練を可能にするには、VGG16の畳み込みベースを凍結する必要がある.

- 同じ理由で、畳み込みベースの出力側の層のファインチューニングが可能になるのは、その分類器の訓練が既に完了している場合.

ファインチューニング

- ネットワークの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. 追加した部分の訓練を行う.
  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.

- 最初の3つの手順は、特徴抽出を行ったときに完了している.
- 手順4に進み、畳み込みベース(conv_base)を解凍し、その中に含まれている層を個別に凍結する.

ファインチューニング

- ここでは、畳み込みベースの最後の2つか3つの層だけでファインチューニングを行っている.
  - ✓ 新しい問題での再利用には、より具体的な特徴量をファインチューニングする方が有益なため.
  - ✓ 訓練対象のパラメータの数が増えると、過学習のリスクが高くなるため.

```
170  #ファインチューニング
171  conv_base.trainable = True
172
173  set_trainable = False
174  for layer in conv_base.layers:
175      if layer.name == 'block5_conv1':
176          set_trainable = True
177      if set_trainable:
178          layer.trainable = True
179      else:
180          layer.trainable = False
```

ファインチューニング

- このファインチューニングでは、RMSpropオプティマイザとかなり低い学習率を使用.
  - ✓低い学習率を使用するのは、ファインチューニングを行う3つの層の表現に対する変更の大きさを制限するため.
  - ✓更新値が大きすぎると、これらの表現を傷つけてしまう可能性がある.

```
182    model.compile(loss='binary_crossentropy',
183                  optimizer=optimizers.RMSprop(lr=1e-5),
184                  metrics=['accuracy'])
185
186    history = model.fit_generator(
187          train_generator,
188          steps_per_epoch=100,
189          epochs=100,
190          validation_data=validation_generator,
191          validation_steps=50)
```

ファインチューニング

- そのままプロットするとノイズだらけなので、各データ点をその手前にあるデータ点の指数移動平均に置き換えて、スムージング.

```
216  def smooth_curve(points, factor=0.8):
217      smoothed_points = []
218      for point in points:
219          if smoothed_points:
220              previous = smoothed_points[-1]
221              smoothed_points.append(previous * factor + point * (1 - factor))
222          else:
223              smoothed_points.append(point)
224      return smoothed_points
225
226  plt.plot(epochs,
227           smooth_curve(acc), 'bo', label='Smoothed training acc')
228  plt.plot(epochs,
229           smooth_curve(val_acc), 'b', label='Smoothed validation acc')
230  plt.title('Training and validation accuracy')
231  plt.legend()
232
233  plt.figure()
234
235  plt.plot(epochs,
236           smooth_curve(loss), 'bo', label='Smoothed training loss')
237  plt.plot(epochs,
238           smooth_curve(val_loss), 'b', label='Smoothed validation loss')
239  plt.title('Training and validation loss')
240  plt.legend()
241
242  plt.show()
```

ファインチューニング

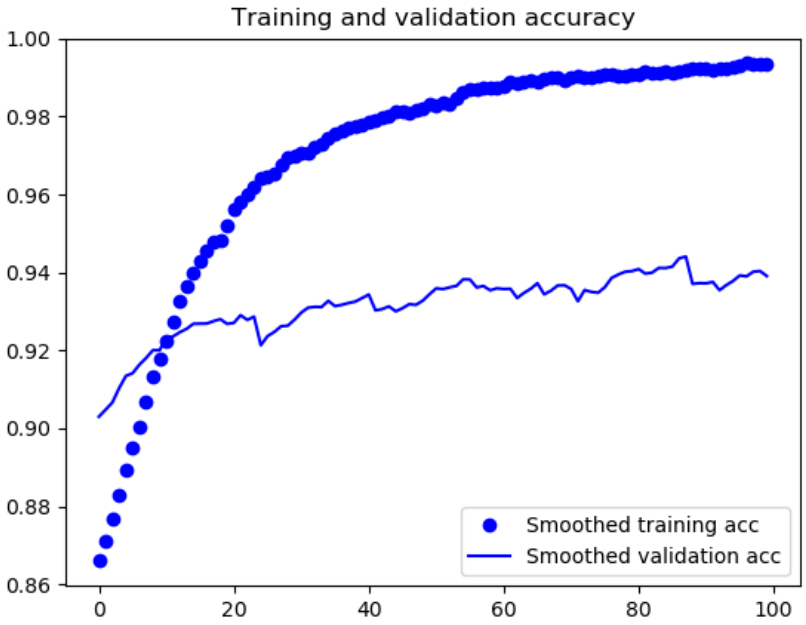

図: ファインチューニングを
行った場合の正解率
(スムージング後)

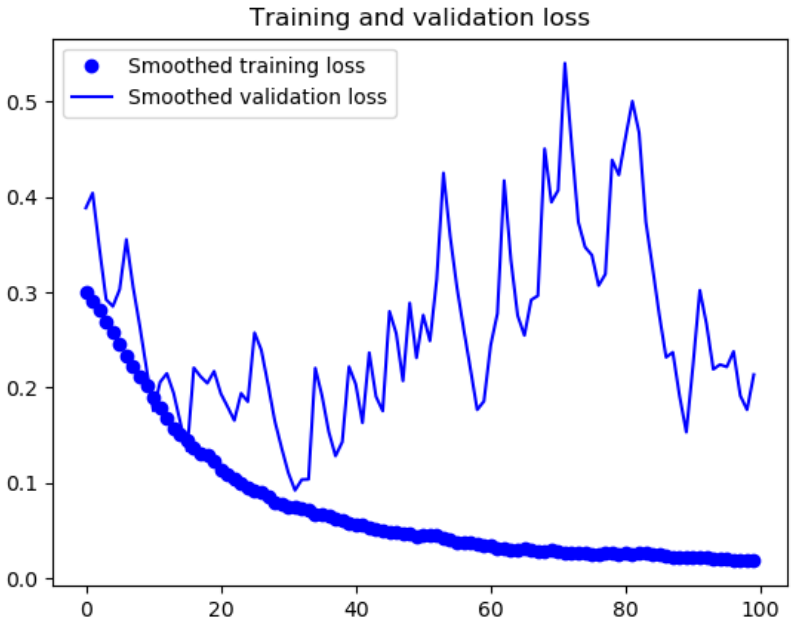

図: ファインチューニングを
行った場合の損失値
(スムージング後)

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モデルである. データセットが非常に小さい場合でもモデルを一から訓練することが可能で、良い結果を得ることが出来る.

- データセットが小さい場合の主な課題は過学習.データの拡張は、画像データを処理するときに過学習に対処するための強力な手段である.

- 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る. 特徴抽出は小さな画像データセットを処理するための有益な手段である.

- 特徴抽出の補完にはファインチューニングを利用できる. ファインチューニングは、既存のモデルによって学習された表現の一部を新しい問題に適合させる. これにより、性能がさらに改善される.