2. 医用画像識別

本田 康祐

2020/04/19

1　モデル

今回の実験では畳み込みニューラルネットワーク (CNN: Convolutional Neural Network) により医用画像を腫瘍あり (1)，なし (0) の 2 値に分類した．

使用したモデルは [1] より参照した．畳み込み層 2 層，max プーリング層 1 層を用いたモデルである．以下にモデルの詳細を示す．

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 222, 222, 32)      320
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 220, 220, 64)      18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 110, 110, 64)      0
_________________________________________________________________
dropout_1 (Dropout)          (None, 110, 110, 64)      0
_________________________________________________________________
flatten_1 (Flatten)          (None, 774400)            0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               99123328
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 258
=================================================================
Total params: 99,142,402
Trainable params: 99,142,402
Non-trainable params: 0
_________________________________________________________________
```

畳み込み層 (Conv2D) の活性化関数は relu 関数，全結合層の入力層は relu 関数，出力層は softmax 関数を使用した．

損失関数は binary crossentropy 関数，最適化アルゴリズムは Adadelta を使用した．

2　ソースコード

今回 Google Colaboratory を用いて実装した．フレームワークは keras を使用した．以下にコードを示していく．

ソースコード 1: インポート

```
import numpy as np
from PIL import Image
import os
import sys
from glob import glob
import cv2
import re

import keras
from keras.models import Sequential, Input, Model
from keras.layers import Dense, Dropout, Flatten, Conv2D,
    MaxPooling2D, Lambda
from keras.optimizers import Adam
from keras import backend as K
```

ソースコード 2: 画像の numpy 形式と対応する正解ラベルを返す関数

```
# 画像のnumpy 形式と対応する正解ラベルを返す関数
def readImg(paths):

  N = len(paths)

  # 画像読み込み準備
  imgs = [[] for i in range(N)]
  # 正解データ作成
  imgs_targets = []

  for k, path in enumerate(paths):

    #label = 画像が入ってるフォルダ名
    label = os.path.basename(os.path.dirname(path))

    if label == "0":
          imgs_targets.append(0)
    else:
      imgs_targets.append(1)

    imgs[k] = np.array(Image.open(path))

    sys.stderr.write('{}枚目\r'.format(k))
    sys.stderr.flush()

  sys.stderr.write('\n')

  imgs = np.array(imgs, dtype = "float32")
  imgs_targets = np.array(imgs_targets, dtype = "int32")

  return imgs, imgs_targets
```

ソースコード 3: データセットのファイルパス読み込み

```
# データセットのあるパス
main_path = "/content/drive/My␣Drive/Colab␣Notebooks/Dataset/"
train_path = main_path + "train/"
val_path = main_path + "val/"
test_path = main_path + "test/"

# 全画像のパス読み込み
train_paths = np.array(sorted(glob(train_path + "**/*.png"),
                          key=lambda s: int(re.findall(r'\d+', s)[len(
                              re.findall(r'\d+', s))-1])))
val_paths = np.array(sorted(glob(val_path + "**/*.png"),
                        key=lambda s: int(re.findall(r'\d+', s)[len(re
                            .findall(r'\d+', s))-1])))
test_paths = np.array(sorted(glob(test_path + "**/*.png"),
                         key=lambda s: int(re.findall(r'\d+', s)[len(
                             re.findall(r'\d+', s))-1])))

print(len(train_paths), len(val_paths), len(test_paths))
print(val_paths)
```

ソースコード 4: 画像の読み込み (初めて画像を読み込むときに使用)

```
# 画像読み込み
x_train, y_train = readImg(train_paths)
x_val, y_val = readImg(val_paths)
x_test, y_test = readImg(test_paths)

print(x_train.shape, y_train.shape, x_val.shape, y_val.shape, x_test.
    shape, y_test.shape)

# npy 形式で保存
np.save(main_path + "x_train", x_train)
np.save(main_path + "y_train", y_train)
np.save(main_path + "x_val", x_val)
np.save(main_path + "y_val", y_val)
np.save(main_path + "x_test", x_test)
np.save(main_path + "y_test", y_test)
```

ソースコード 5: npy 形式から読み込み

```
# npy 形式から読み込み
x_train, y_train = np.load(main_path + "x_train.npy"), np.load(
    main_path + "y_train.npy")
x_val, y_val = np.load(main_path + "x_val.npy"), np.load(main_path +
    "y_val.npy")
x_test, y_test = np.load(main_path + "x_test.npy"), np.load(main_path
     + "y_test.npy")

print(x_train.shape, x_val.shape, x_test.shape)
```

ソースコード 6: 画像の前処理

```
# 正規化
x_train /= 255.0
```

```
x_val /= 255.0
x_test /= 255.0

# 訓練データをシャッフル
perm = np.random.permutation(x_train.shape[0])
x_train, y_train = x_train[perm], y_train[perm]
```

ソースコード 7: データ整形

```
# パラメータ
num_classes = 2
img_rows, img_cols, channel = 224, 224, 1

# 検証用の正解ラベルを保存しておく
true_labels = y_test[:]

# 正解ラベルをone-hot に変換
y_train = keras.utils.to_categorical(y_train, num_classes)
y_val = keras.utils.to_categorical(y_val, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

# CNN 用に次元を追加
if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], channel, img_rows,
        img_cols)
    x_val = x_val.reshape(x_val.shape[0], channel, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], channel, img_rows,
        img_cols)
    input_shape = (channel, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols,
        channel)
    x_val = x_val.reshape(x_val.shape[0], img_rows, img_cols, channel)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols,
        channel)
    input_shape = (img_rows, img_cols, channel)
```

ソースコード 8: モデルの定義

```
# モデル定義の関数
def model_net():

    model = Sequential()
    model.add(Conv2D(32, kernel_size=(3, 3),
                     activation='relu',
                     input_shape=input_shape))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation='softmax'))

    return model

```

```
# 学習済みモデルがあったら読み込み
if os.path.exists(main_path + 'models/model_net.h5'):
    model = keras.models.load_model(main_path + 'models/model_net.h5'
        , compile=False)
    print("モデル読み込み")

# なかったら新しく定義
else:
    model = model_net()
    print("モデル新規作成")

# モデル出力
print(model.summary())

# モデルの設定
model.compile(loss=keras.losses.binary_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])
```

ソースコード 9: モデルの学習，保存

```
# パラメータ
batch_size, epochs = 128, 50

# 学習
stack = model.fit(x_train, y_train,
                  batch_size=batch_size,
                  epochs=epochs,
                  verbose=1,
                  validation_data=(x_val, y_val))

# モデルの保存
model.save(main_path + 'models/model_net.h5', include_optimizer=
    False)
```

ソースコード 10: モデルの評価

```
# 評価
score = model.evaluate(x_test, y_test, verbose=1)
print(list(zip(model.metrics_names, score)))

# 検証データから推定したラベルを出力
predict_labels = model.predict_classes(x_test)

# 推定ラベルと正解ラベルの組を入力し，混合行列
    confusion_m および各組に対するラベル confusion_l を返す
# confusion_m = [TP, TN, FP, FN]
# confusion_l = [0, 3, 1, ...](TP=0, TN=1, FP=2, FN=3)
def make_confusion_m(predict_labels, true_labels):
    confusion_m = np.array([0, 0, 0, 0])
    confusion_l = []

    for (true, pred) in zip(true_labels, predict_labels):
        # TP
        if true==True and pred==True:
            confusion_m[0] += 1
```

```
            confusion_l.append(0)

        # TN
        elif true==False and pred==False:
            confusion_m[1] += 1
            confusion_l.append(1)

        # FP
        elif true==False and pred==True:
            confusion_m[2] += 1
            confusion_l.append(2)

        # FN
        elif true==True and pred==False:
            confusion_m[3] += 1
            confusion_l.append(3)

    return confusion_m, np.array(confusion_l)

# 混合行列
confusion_m, confusion_l = make_confusion_m(predict_labels,
    true_labels)
print(confusion_m)

# 不正解の検証データのインデックスを取り出す
negative_indices = np.where(confusion_l >= 2)
print(negative_indices)

# 不正解の検証データ
negative_files = test_paths[negative_indices]
print(negative_files)
```

3 実験結果

用意されていた訓練データ 8980 枚を用いてモデルを 50 エポック学習させた後，検証データ 2458 枚を分類した．

その結果の混合行列を以下の表 1 に示す．

Table 1: 混合行列

		モデルの予測ラベル		
		Positive	Negative	Total
実際のラベル	Positive	1151	112	1263
	Negative	155	1040	1195
	Total	1306	1152	2458

また，認識率 (Accuracy)，適合率 (Precision)，再現率 (Recall) の値を以下に示す．

$$Accuracy = 89.1\%,\ Precision = 88.1\%,\ Recall = 91.1\% \tag{1}$$

3 つの値のどれも 88%以上を超える結果となったが，実際に画像による腫瘍の有無の分類を実用化するとした場合，Recall を 100%に近づける必要があると考えられる．

References

[1] keras-team (https://github.com/keras-team/keras/blob/master/examples/mnist_cnn.py)