

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 (MaxPooling2D)	(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

```
1 import numpy as np
2 from PIL import Image
3 import os
4 import sys
5 from glob import glob
6 import cv2
7 import re
8
9 import keras
10 from keras.models import Sequential, Input, Model
11 from keras.layers import Dense, Dropout, Flatten, Conv2D,
    MaxPooling2D, Lambda
12 from keras.optimizers import Adam
13 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

```
1 # 画像のnumpy形式と対応する正解ラベルを返す関数
2 def readImg(paths):
3
4     N = len(paths)
5
6     # 画像読み込み準備
7     imgs = [[] for i in range(N)]
8     # 正解データ作成
9     imgs_targets = []
10
11     for k, path in enumerate(paths):
12
13         #label = 画像が入ってるフォルダ名
14         label = os.path.basename(os.path.dirname(path))
15
16         if label == "0":
17             imgs_targets.append(0)
18         else:
19             imgs_targets.append(1)
20
21         imgs[k] = np.array(Image.open(path))
22
23         sys.stderr.write('{}枚目\r'.format(k))
24         sys.stderr.flush()
25
26     sys.stderr.write('\n')
27
28     imgs = np.array(imgs, dtype = "float32")
29     imgs_targets = np.array(imgs_targets, dtype = "int32")
30
31     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

```
1 # データセットのあるパス
2 main_path = "/content/drive/My_Drive/Colab_Notebooks/Dataset/"
3 train_path = main_path + "train/"
4 val_path = main_path + "val/"
5 test_path = main_path + "test/"
6
7 # 全画像のパス読み込み
8 train_paths = np.array(sorted(glob(train_path + "**/*.png"),
9                               key=lambda s: int(re.findall(r'\d+', s)[len(
10                                re.findall(r'\d+', s))-1]))))
11 val_paths = np.array(sorted(glob(val_path + "**/*.png"),
12                             key=lambda s: int(re.findall(r'\d+', s)[len(re
13                              .findall(r'\d+', s))-1]))))
14 test_paths = np.array(sorted(glob(test_path + "**/*.png"),
15                              key=lambda s: int(re.findall(r'\d+', s)[len(
16                               re.findall(r'\d+', s))-1]))))
17
18 print(len(train_paths), len(val_paths), len(test_paths))
19 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)

```
1 # 画像読み込み
2 x_train, y_train = readImg(train_paths)
3 x_val, y_val = readImg(val_paths)
4 x_test, y_test = readImg(test_paths)
5
6 print(x_train.shape, y_train.shape, x_val.shape, y_val.shape, x_test.
7       shape, y_test.shape)
8
9 # npy 形式で保存
10 np.save(main_path + "x_train", x_train)
11 np.save(main_path + "y_train", y_train)
12 np.save(main_path + "x_val", x_val)
13 np.save(main_path + "y_val", y_val)
14 np.save(main_path + "x_test", x_test)
15 np.save(main_path + "y_test", y_test)
```

ソースコード 5: npy 形式から読み込み

```
1 # npy 形式から読み込み
2 x_train, y_train = np.load(main_path + "x_train.npy"), np.load(
3     main_path + "y_train.npy")
4 x_val, y_val = np.load(main_path + "x_val.npy"), np.load(main_path +
5     "y_val.npy")
6 x_test, y_test = np.load(main_path + "x_test.npy"), np.load(main_path
7     + "y_test.npy")
8
9 print(x_train.shape, x_val.shape, x_test.shape)
```

ソースコード 6: 画像の前処理

```
1 # 正規化
2 x_train /= 255.0
```

```

3 x_val /= 255.0
4 x_test /= 255.0
5
6 # 訓練データをシャッフル
7 perm = np.random.permutation(x_train.shape[0])
8 x_train, y_train = x_train[perm], y_train[perm]

```

ソースコード 7: データ整形

```

1 # パラメータ
2 num_classes = 2
3 img_rows, img_cols, channel = 224, 224, 1
4
5 # 検証用の正解ラベルを保存しておく
6 true_labels = y_test[:]
7
8 # 正解ラベルをone-hotに変換
9 y_train = keras.utils.to_categorical(y_train, num_classes)
10 y_val = keras.utils.to_categorical(y_val, num_classes)
11 y_test = keras.utils.to_categorical(y_test, num_classes)
12
13 # CNN用に次元を追加
14 if K.image_data_format() == 'channels_first':
15     x_train = x_train.reshape(x_train.shape[0], channel, img_rows,
16                               img_cols)
17     x_val = x_val.reshape(x_val.shape[0], channel, img_rows, img_cols)
18     x_test = x_test.reshape(x_test.shape[0], channel, img_rows,
19                              img_cols)
20     input_shape = (channel, img_rows, img_cols)
21 else:
22     x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols,
23                               channel)
24     x_val = x_val.reshape(x_val.shape[0], img_rows, img_cols, channel)
25     x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols,
26                              channel)
27     input_shape = (img_rows, img_cols, channel)

```

ソースコード 8: モデルの定義

```

1 # モデル定義の関数
2 def model_net():
3
4     model = Sequential()
5     model.add(Conv2D(32, kernel_size=(3, 3),
6                       activation='relu',
7                       input_shape=input_shape))
8     model.add(Conv2D(64, (3, 3), activation='relu'))
9     model.add(MaxPooling2D(pool_size=(2, 2)))
10    model.add(Dropout(0.25))
11    model.add(Flatten())
12    model.add(Dense(128, activation='relu'))
13    model.add(Dropout(0.5))
14    model.add(Dense(num_classes, activation='softmax'))
15
16    return model
17

```

```

18 # 学習済みモデルがあったら読み込み
19 if os.path.exists(main_path + 'models/model_net.h5'):
20     model = keras.models.load_model(main_path + 'models/model_net.h5',
21                                     compile=False)
22     print("モデル読み込み")
23 # なかったら新しく定義
24 else:
25     model = model_net()
26     print("モデル新規作成")
27
28 # モデル出力
29 print(model.summary())
30
31 # モデルの設定
32 model.compile(loss=keras.losses.binary_crossentropy,
33               optimizer=keras.optimizers.Adadelta(),
34               metrics=['accuracy'])

```

ソースコード 9: モデルの学習、保存

```

1 # パラメータ
2 batch_size, epochs = 128, 50
3
4 # 学習
5 stack = model.fit(x_train, y_train,
6                  batch_size=batch_size,
7                  epochs=epochs,
8                  verbose=1,
9                  validation_data=(x_val, y_val))
10
11 # モデルの保存
12 model.save(main_path + 'models/model_net.h5', include_optimizer=False)

```

ソースコード 10: モデルの評価

```

1 # 評価
2 score = model.evaluate(x_test, y_test, verbose=1)
3 print(list(zip(model.metrics_names, score)))
4
5 # 検証データから推定したラベルを出力
6 predict_labels = model.predict_classes(x_test)
7
8 # 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
9 # confusion_m = [TP, TN, FP, FN]
10 # confusion_l = [0, 3, 1, ...] (TP=0, TN=1, FP=2, FN=3)
11 def make_confusion_m(predict_labels, true_labels):
12     confusion_m = np.array([0, 0, 0, 0])
13     confusion_l = []
14
15     for (true, pred) in zip(true_labels, predict_labels):
16         # TP
17         if true==True and pred==True:
18             confusion_m[0] += 1

```

```

19         confusion_l.append(0)
20
21     # TN
22     elif true==False and pred==False:
23         confusion_m[1] += 1
24         confusion_l.append(1)
25
26     # FP
27     elif true==False and pred==True:
28         confusion_m[2] += 1
29         confusion_l.append(2)
30
31     # FN
32     elif true==True and pred==False:
33         confusion_m[3] += 1
34         confusion_l.append(3)
35
36     return confusion_m, np.array(confusion_l)
37
38 # 混合行列
39 confusion_m, confusion_l = make_confusion_m(predict_labels,
40 true_labels)
41 print(confusion_m)
42
43 # 不正解の検証データのインデックスを取り出す
44 negative_indices = np.where(confusion_l >= 2)
45 print(negative_indices)
46
47 # 不正解の検証データ
48 negative_files = test_paths[negative_indices]
49 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。

$$Accuracy = 89.1\%, Precision = 88.1\%, Recall = 91.1\% \quad (1)$$

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)