

画像選択の機械学習プログラム (Learning_FigSelection.py) 使用説明書

目次

1. 機能
2. 動作環境
3. 概要
 - 3.1 構造
 - 3.2 プログラムリストの概要
 - 3.3 入出力
 - 3.3.1 入力
 - 3.3.2 出力
4. パラメータ
5. プログラムリスト

1. 機能

このプログラム(Learning_FigSelection.py)は機械学習を行い、グラフ数値化プログラム(WaveForm Digitizer.py)で読み取り可能な画像を選別するための学習モデルを生成する

2. 動作環境

1) このプログラムを実行するには、以下のモジュールがあらかじめ準備された環境が必要である

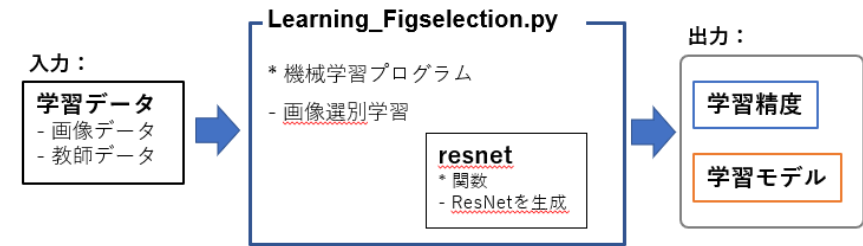
- Python 3.5 or 3.6
- Tensorflow 1.13
- Keras
- sklearn
- matplotlib

2) このプログラムの存在するディレクトリ下に'data'および'model'の2つのサブディレクトリが必要である

3. 構造

3.1 概要

このプログラムは、学習用データ（画像データと教師データからなる）を入力とし、内部の関数 resnetで生成するモデルに従って機械学習を行い、学習精度を出力し学習済モデルを生成する。



3.2 プログラムリストの概要

このプログラムは全441行からなり、「モデルを生成する関数」と「機械学習の実行」の2つのモジュールに分けられる
全リストは、5. プログラムリストに示す

1. 1-9 行: 開始コメント
2. 10-219 行: 学習モデルを生成する関数
 - 15-26 行: モジュールのインポート
 - 33-81 行: モデルパラメータの記述
 - 83-219 行: ResNetの生成
3. 221- 行: 機械学習の実行
 - 225-231 行: モジュールのインポート
 - 233-242 行: 開始準備
 - 244-279 行: データの読み込みと振り分け
 - 281-290 行: パラメータの設定
 - 292-312 行: モデル構築
 - 314-338 行: モデルコンパイル
 - 340-351 行: モデルフィット
 - 353-428 行: 結果の評価と出力
 - 430-453 行: モデルの保存
 - 455 行: END

3.3 入出力

3.3.1 入力

学習用データは以下の要件を満たす必要がある

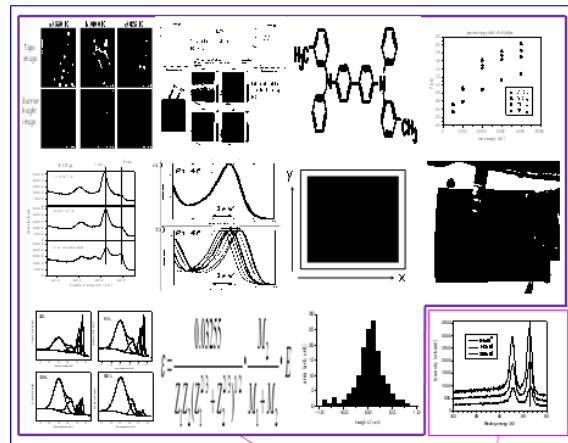
- pickle形式で保存された辞書型のデータであること. <データ名.pkl >
- 2つのキー'in', 'out'を持ち、'in'に画像データ、'out'に教師データが保存されていること

画像データ:
0-255の整数の数値をとり、そのサイズが256x256で1チャンネル(グレイタイ
プ)の画像配列(256,256,1)

教師データ:

0: 適合しない, 1: 適合するをとる1個の整数配列(1)

画像データ (256 x 256)



0 1

教師データ

3.3.2 出力

学習済モデルおよびそのモデル形状は計算終了後、保存の可否を選ぶことができる。

以下に実際の出力例をしめす

(1) 入力パラメータの確認

```
< MACHINE LEARNING OF IMAGE SELECTION__ 20191121-1500 >
```

```
Data = ./data/FigSelection_ad4500-y1-y2_256_11532_20191114-0958.pik
```

```
* Learning Process Start ...
```

```
This is Wide-ResNet  
for Classification  
with Dropout, BatchNormalization and l2-regularizer
```

```
Dropout = 0.3  
L2-Regularizer = 0.0001
```

```
Learn_n      = 9225 , Test_n = 2307  
first_filter = 32  
batch        = 10  
patience     = 30
```

```
Optimizer : adam  
loss      : binary_crossentropy
```

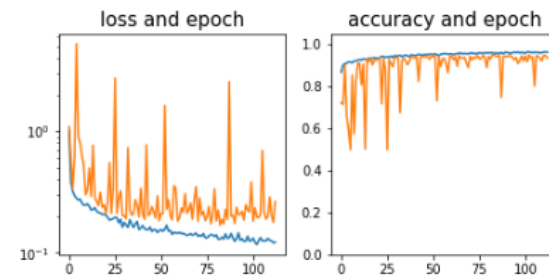
(2) 精度と正答率

lossとaccの履歴と正答率の表示

```
... Finished. Exe time = 6213.89 sec
```

```
< Loss and accuracy >
```

```
loss      = 1.22e-01 , val_loss = 2.64e-01 , delta = 1.42e-01  
accuracy  = 0.962
```



```
< Error and correct rate >
```

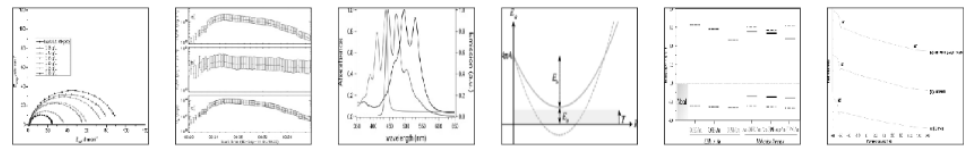
```
error_n = 152 / 2307 , correct predict rate = 0.934
```

(3) エラー

うまく予測できなかったデータの中でランダムに6個を取り出して表示

```
< Error and correct rate >
error_n = 93 / 2307 , correct predict rate = 0.960

< Error sample >
figure number = [ 814 1131 995 1396 2156 384]
814 : 1 --> 0 prd
1131 : 0 --> 1 prd
995 : 1 --> 0 prd
1396 : 1 --> 0 prd
2156 : 0 --> 1 prd
384 : 1 --> 0 prd
```



(4) モデル出力

モデルおよびモデル形状は計算終了後、下の様に入力待ち状態になるので、キーボードから **y** をタイプすることで保存できる。**y** 以外では保存されない

```
< Model Save and Print >
Print the learned model? y or n --> y
Model is printed --> ./model/Shape_FigSelection_W-RN_256_20191121-1500.png
Save the learned model? y or n --> y
Model is saved --> ./model/ FigSelection_W-RN_256_20191121-1500.h5
```

4. パラメータ

このプログラムのパラメータは256 x 256の大きさの画像選択に最適化されている。しかし、学習データおよびパラメータは下記の様に変更可能であり、このプログラムの全体あるいは部分を利用して他の用途に転用することが可能である

変数名	説明	デフォルト値	可能な値
test_size , train_size	データの訓練数とテスト数の割合	0.8,0.2	合計して 1
dim	画像の大きさ	256	128,256,360,512,1024で計算で検討された
m_n	ResNetの種類	RN34	RN18,RN34,RN50,RN101,RN152
drp_n	ドロップアウト	0.3	0-1
reg_f	L2-Regularizer	1e-4	0-1
n_epoch	最大エポック数	200	1-
n_batch	バッチ数	40	1-
first_fil_n	開始フィルタ数	16	4,8,16,32... 2^n

変数名	説明	デフォルト値	可能な値
n_pat	EarlyStopで計算を打ち切る変動の最大数	30	1-
flag_ver	エポック毎の表示の選択	0	0:表示しない、1:表示する

5. プログラムリスト

概要

- 1. 1-9 行: 開始コメント
- 2. 10-219 行: 学習モデルを生成する関数
 - 15-26 行: モジュールのインポート
 - 33-81 行: モデルパラメータの記述
 - 83-219 行: ResNetの生成
- 3. 221- 行: 機械学習の実行
 - 225-231 行: モジュールのインポート
 - 233-242 行: 開始準備
 - 244-279 行: データの読み込みと振り分け
 - 281-290 行: パラメータの設定
 - 292-312 行: モデル構築
 - 314-338 行: モデルコンパイル
 - 340-351 行: モデルフィット
 - 353-428 行: 結果の評価と出力
 - 430-453 行: モデルの保存
 - 455 行: END


```

121         )(X)
122         #X = BatchNormalization()(X)
123
124     else:
125         X = Activation("relu")(X)
126         X = Dropout(drp_n)(X)
127         X = Conv2D(n_filter,
128                   kernel_size=(3,3),
129                   padding="same",
130                   kernel_regularizer=l2(reg_f)
131                   )(X)
132         #X = BatchNormalization()(X)
133
134         X = Activation("relu")(X)
135         X = Dropout(drp_n)(X)
136         X = Conv2D(n_filter,
137                   kernel_size=(3,3),
138                   padding="same",
139                   kernel_regularizer=l2(reg_f)
140                   )(X)
141
142         # Add
143         X = Add()([X, shortcut])
144         shortcut = X
145
146         #X = BatchNormalization()(X)
147
148         n_filter *= wide
149
150     if block == 'bottleneck':
151         shortcut = Conv2D(n_filter*4,
152                           kernel_size=(1,1),
153                           kernel_regularizer=l2(reg_f)
154                           )(shortcut)
155
156     for i, repete in enumerate(nb_blocks):
157         for j in range(repete):
158             if i>0 and j == 0:
159                 shortcut = Conv2D(n_filter*4,
160                                   kernel_size=(1,1),
161                                   strides=(2,2),
162                                   kernel_regularizer=l2(reg_f)
163                                   )(shortcut)
164
165                 X = Activation("relu")(X)
166                 X = Dropout(drp_n)(X)
167                 X = Conv2D(n_filter,
168                           kernel_size=(1,1),
169                           strides=(2,2),
170                           padding="same",
171                           kernel_regularizer=l2(reg_f)
172                           )(X)
173                 #X = BatchNormalization()(X)
174
175             else:
176                 X = Activation("relu")(X)
177                 X = Dropout(drp_n)(X)
178                 X = Conv2D(n_filter,
179                           kernel_size=(1,1),
180                           padding="same",
181                           kernel_regularizer=l2(reg_f)

```

```

182         )(X)
183         #X = BatchNormalization()(X)
184
185         X = Activation("relu")(X)
186         X = Dropout(drp_n)(X)
187         X = Conv2D(n_filter,
188                   kernel_size=(3,3),
189                   padding="same",
190                   kernel_regularizer=l2(reg_f)
191                   )(X)
192         #X = BatchNormalization()(X)
193
194         X = Activation("relu")(X)
195         X = Dropout(drp_n)(X)
196         X = Conv2D(n_filter*4,
197                   kernel_size=(1,1),
198                   padding="same",
199                   kernel_regularizer=l2(reg_f)
200                   )(X)
201
202         # add
203         X = Add()([X, shortcut])
204         shortcut = X
205         #X = BatchNormalization()(X)
206
207         n_filter *= wide
208
209         X = Activation("relu")(X)
210         X = GlobalAveragePooling2D()(X)
211         X = Dropout(drp_n)(X)
212         Out = Dense(num_outputs,
213                     activation="sigmoid"
214                     )(X)
215
216         model = Model(inputs=[input], outputs=[Out])
217
218     return model
219
220 # -----
221 #                                     Machine Learning Process
222 # -----
223
224 import numpy as np
225 import matplotlib.pyplot as plt
226 from keras.callbacks import EarlyStopping
227 from keras.utils import np_utils
228 from sklearn.model_selection import train_test_split
229 import pickle, time
230 from datetime import datetime
231
232 # --- Opening -----
233 # date
234 today = datetime.today().strftime('%y%m%d-%H%M')
235
236 # opening tittle
237 print("\n< MECHINE LEARNINI OF IMAGE SELECTION_',
238       today,>' )
239
240 # start time
241 s_time = time.time()

```

```

243
244 # --- Data import -----
245 # data read
246 # data_n = input(' Input Data Name --> \n ')
247 data_n = './data/FigSerrection_ad4500-y1-y2_256_11532_20191114-0958.pik'
248 # data_n = './data/FigSerrection_256_2678_20191024-1632.pik'
249 print('\n Data = ',data_n)
250
251 with open(data_n, mode='rb') as f:
252     test_graf = pickle.load(f)
253
254 # X-normalize and black-white reverse
255 dim = 256 # dim = int(test_graf['para'][0])
256 X = test_graf['in']
257 if np.max(X[0,:]) == 255:
258     nor = 255 # case of gray data
259 else:
260     nor = 1 # case of monochro data
261 X = 1-X/nor # black-white reverse
262
263 print('\n> Learning Process Start ... ')
264
265 # Y-categorize
266 Yo = test_graf['out']
267 Y = np_utils.to_categorical(Yo) # 教師データをカテゴリー型に変換
268
269 # input/output size
270 n_in = len(X[0])
271 n_out = len(Y[0])
272
273 # --- Data split -----
274 x_train, x_test, y_train, y_test = train_test_split(
275     X, Y,
276     test_size = 0.2,
277     train_size = 0.8,
278     random_state = 0
279 )
280
281 # --- Iteration parameter -----
282 drp_n = 0.5
283 reg_f = 0
284
285 n_epoch = 300
286 n_batch = 30
287 first_fil_n = 32
288
289 n_pat = 100
290 flag_ver = 0
291
292 # --- Model build -----
293 input_ = x_train.shape[1:]
294 m_n = 'RN34'
295
296 model = resnet(input_,
297     n_out,
298     m_n,
299     first_fil_n,
300     drp_n,
301     reg_f
302 )
303

```

```

304 # model.summary()
305
306 # parameter print
307 print('\n Learn_n = ',len(x_train),
308     ',Test_n = ',len(x_test))
309 print(' max_epoch = ',n_epoch)
310 print(' first_filter = ',first_fil_n)
311 print(' batch = ',n_batch)
312 print(' patience = ',n_pat)
313
314 # --- Model compile -----
315 # Optimizer adam
316 optimizer = 'adam'
317 print('\n Opimizer : adam')
318
319 ''' Optimizer selection
320 # Optimizer adam
321 optimizer = 'adam'
322 print('\n Opimizer : adam')
323
324 # Optimizer SGD + Momentum
325 from keras.optimizers import SGD
326 print('\n Opimizer : SGD + Momentum')
327 optimizer = SGD(decay=1e-6, momentum=0.9, nesterov=True)
328 '''
329
330 loss = 'binary_crossentropy'
331 # compile
332 model.compile(loss = loss,
333     optimizer = optimizer,
334     metrics = ['accuracy']
335 )
336
337 # 'categorical_crossentropy', 'binary_crossentropy'
338 print(' loss : ',loss)
339
340 # --- Model fit -----
341 early_stopping = EarlyStopping(monitor='val_loss',
342     patience=n_pat
343 )
344
345 hist = model.fit(x_train, y_train,
346     epochs = n_epoch,
347     batch_size = n_batch,
348     validation_data = (x_test, y_test),
349     verbose = flag_ver,
350     callbacks = [early_stopping]
351 )
352
353 # --- Predict and evaluation -----
354 predict_ = model.predict(x_test, batch_size=n_batch)
355 # dif = np.round(y_test - predict_,0)
356 dif = (y_test - np.round(predict_,0))
357
358 # error list
359 dif_ = abs(dif)
360 pass_list = []; ng_list=[]
361 for i in range(0,len(y_test)):
362     if max(dif_[i]) == 0:
363         pass_list.append(i)
364     else:

```

```

365     ng_list.append(i)
366
367 # correct predict rate
368 cpr = 1 - len(ng_list)/(len(dif)) # correct predict rate
369
370 # Execution time
371 n_ep = len(hist.history['loss'])
372 e_time = time.time() # end time
373 print("\n* Process was finished. Exe time (s) = ",
374       '{:6.2f}'.format(e_time - s_time),
375       ', end_ep = ', n_ep)
376
377 # accuracy print
378 de_loss = hist.history['val_loss'][n_ep-1]-\
379 hist.history['loss'][n_ep-1]
380 print("\n< Loss and accuracy >\n loss    = ',
381       '{:4.2e}'.format(hist.history['loss'][n_ep-1]),
382       ', val_loss = ',
383       '{:4.2e}'.format(hist.history['val_loss'][n_ep-1]),
384       ', delta = ',
385       '{:4.2e}'.format(de_loss),
386       '\n accuracy = ',
387       '{:4.3f}'.format(hist.history['acc'][n_ep-1])
388 )
389
390 # loss and accuracy history
391 fig = plt.figure(figsize=(7, 7))
392 plt.subplot(2, 2, 1)
393 x_ep = range(len(hist.history['loss']))
394 plt.plot(x_ep, hist.history['loss'])
395 plt.plot(x_ep, hist.history['val_loss'])
396 plt.yscale('log')
397 plt.title("loss and epoch", fontsize=15)
398
399 plt.subplot(2, 2, 2)
400 x_ep = range(len(hist.history['acc']))
401 plt.plot(x_ep, hist.history['acc'])
402 plt.plot(x_ep, hist.history['val_acc'])
403 plt.title("accuracy and epoch", fontsize=15)
404 plt.ylim(0, 1.05)
405 plt.show()
406
407 print('< Error and correct rate >')
408 print(' error_n = ', len(ng_list), '/', len(dif),
409       ', correct predict rate = ', '{:4.3f}'.format(cpr))
410 # error images drawing
411 if not len(ng_list) == 0:
412     print("\n< Error sample >")
413     fig = plt.figure(figsize=(20,20))
414     n = 6
415     n_list = np.random.choice(ng_list, n)
416     print(' figure number = ', n_list)
417
418     for i in range(n):
419         nn = n_list[i]
420         print(nn, ': ', np.argmax(y_test[nn]), '-->',
421               np.argmax(np.round(predict_[nn], 0)), 'prd')
422         ax = plt.subplot(4, n, i+1)
423         plt.imshow(np.reshape(1-x_test[nn,:], (dim, -1)))
424         plt.gray()
425         ax.get_xaxis().set_visible(False)

```

```

426     ax.get_yaxis().set_visible(False)
427     plt.ylim(dim, 0); plt.xlim(0, dim)
428     plt.show()
429
430 # --- Model print and save -----
431 mp_flag = 'n'; ms_flag = 'n'
432 model_n = 'FigSelection_'+m_n+'_'+str(dim)+'_'+str(today)
433
434 # model print
435 mp_flag = input(' Print the learned model? y or n --> ')
436 if mp_flag == 'y':
437     from keras.utils import plot_model
438     plot_model(model,
439               to_file='./Shape_'+model_n+'.png',
440               show_shapes=True)
441     print(' Model is printed --> ./model/Shape_'+model_n+'.png')
442 else:
443     print(' Model is NOT printed.')
444
445 # model save
446 ms_flag = input(' Save the learned model? y or n --> ')
447 if ms_flag == 'y':
448     model.save('./model/'+model_n+'.h5')
449     print(' Model is saved --> ./model/'+model_n+'.h5')
450 else:
451     print(' Model is NOT saved.')
452
453 del X
454
455 # _____ END of Code _____
456

```

End