

波形本数判別の機械学習用データ作成プログラム (Data_LineNumber.py) 使用説明書

目次

1. 機能
2. 動作環境
3. 内容
 - 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
 - 3.3.3 データの形式
4. プログラムリスト

1. 機能

このプログラム(Data_LineNumber.py)は、機械学習用の学習データを作成する。作成されたデータは、グラフの波形本数を読み取る機械学習(Learning_LineNumber.py)で用いられる

2. 動作環境

1) このプログラムはWindows10, Ubuntu18-04での動作が確認されている

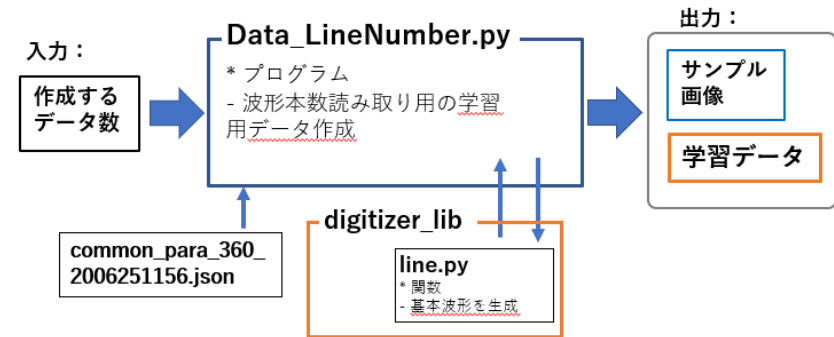
2) このプログラムを実行するには、以下のモジュールがあらかじめ準備された環境が必要である。また、記載されている以外のVersionでの動作確認は行っていないので注意が必要である

- Python 3.6
- Tensorflow-gpu 1.4.0
- Keras 2.2.4
- sklearn 0.19.1, 0.21.2
- matplotlib 2.2.2

3. 内容

3.1 構造

このプログラムは、入力として、作成するデータ数を使用者からのインプットとして受け取り、出力として学習用データとデータのサンプル画像を出力する
このプログラムは、画像データを生成するため、画像パラメータファイル(common_para_360_2006251156.json) から画像パラメータを読み込み、基本波形を生成するため、関数ライブラリdigitizer_lib に収納されている波形生成関数 line.py を呼び出して使用する



3.2 プログラムリストの概要

このプログラムはデータ作成本体(Data_LineNumber.py)と関数ライブラリ(digitizer_lib)に収められている波形生成関数(Line.py)の2つのモジュールに分けられる
全リストは、4. プログラムリストに示す

1. データ作成本体(Data_LineNumber.py)

1-9 行: 開始コメント
11-21 行: モジュールのインポート
29-35 行: 開始準備
37-106 行: データパラメータの設定
108-133 行: パラメータの確認
135-335 行: データの生成
337-353 行: チェック用の画像出力
362-383 行: モデルの保存
385 行: END

2. 波形生成関数(Line.py)

1-7 行: 開始コメント
9-9 行: モジュールのインポート
15-38 行: 開始準備
40-69 行: 波形の生成
71 行: return

3.3 入出力

以下、Jupyter Notebook上で実行された例を示して説明する

3.3.1 入力

このプログラムの入力は、作成するデータ数のみである
プログラムを実行すると、下の図の様にコンソール上でデータ数の入力待ちになるので、使用者は必要なデータ数を入力して実行する

< Data Creating of Machine Learning for LineNumber. 200625-1559 >

Input data number. if <= 99, datas are not saved

=>

次に、下の様な画像パラメータが表示される

このパラメータを確認してOKであれば、'y'もしくは'n'以外を入力して実行するとデータ作成が始まる

< Data Creating of Machine Learning for LineNumber. 200625-1557 >

Input data number. if <= 99, datas are not saved

=> 55

< Data parameter >

common parameter = common_para_360_2006251156.json

dim = 360

dpi = 180

data number = 55

Graph size = 2.0 x 2.0 inch

Line Number = [1, 2, 3, 4, 5, 6]

Line peak number = w/m [1, 2, 3], wo/m [3, 4, 5]

Line width(normal) = 1.0 point(2.5 pixel)

Font size(normal) = 6.0

Marker number = 8 ['+', 'D', '^', 'o', 's', 'v', 'x', '_']

Marker size(normal) = 3

Pitch of marker = 20 - 30

scale factor = [0.8, 0.9, 1.0, 1.13, 1.25]

comment = LineNumber_Data_

Data OK? y or n -->

注) データ数が100個以下が入力された場合は、味見としてデータの保存は実行されず、サンプル画像の表示のみが行われる

3.3.2 出力

データ数が100個以上の場合、データは自動的に保存される

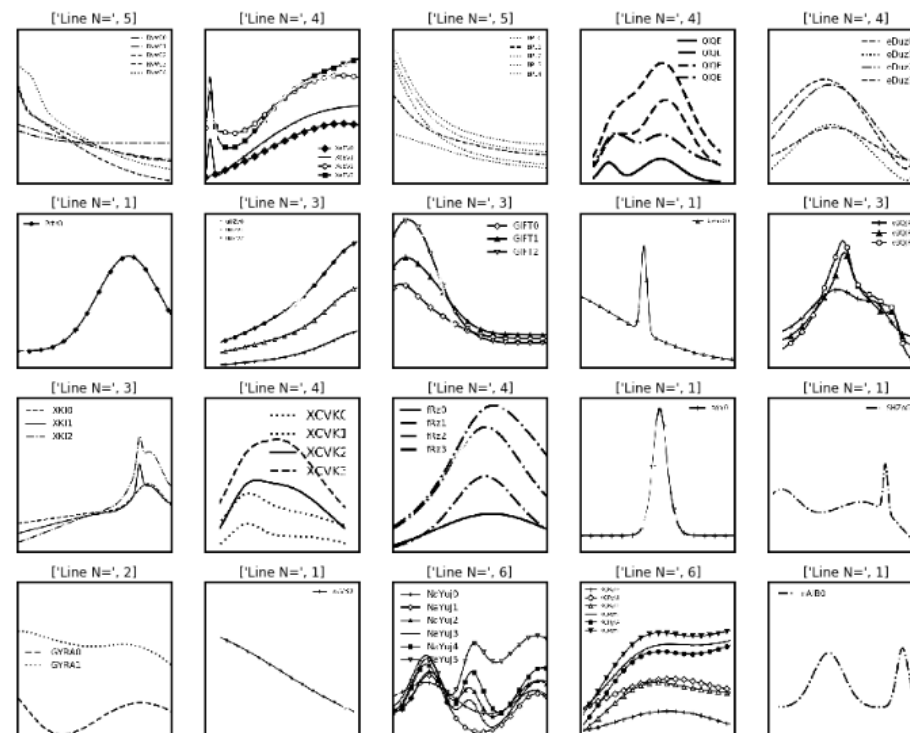
データ数が100個以下の場合は、味見としてデータの保存は実行されず、サンプル画像の表示のみが行われる

以下に500個のデータを作成した実際の出力例をしめす

- まず、20個のサンプルがランダムに選ばれ表示される。各画像の上に波形の本数（教師データ）が表示される
- 次に、最下段に自動的に保存されたデータ名が表示される（この例では LineNumber_Data_360_500_200625-1615.pik）

Data figure check

* sample number = [478 187 119 375 57 172 333 337 149 129 113 291 160 250 270 248 483 41 429 105]



>>> Data was saved : LineNumber Data 360 500 200625-1615.pik

3.3.3 データの形式

データは下記のような形式で出力される

参照:4.1 データ作成本体 (Data_LineNumber.py) プログラムリスト,362-383 行: モデルの保存 及び A2_Learning_LineNumber_Manyual、3.3.1 入力

- pickle形式で保存された辞書型のデータ。<データ名.pik>
- 'data','out','para'の3つのキーを持ち、下表のような要素で構成されている

key	item
-----	------

data	画像データ
------	-------

out	教師データ
-----	-------

key	item
para	画像の大きさ, dpi, コメント, マーカー, 少ない波形ピークの選択数, 大きな波形ピークの選択数, 画像タイプ ラインタイプ

画像データと教師データの形式

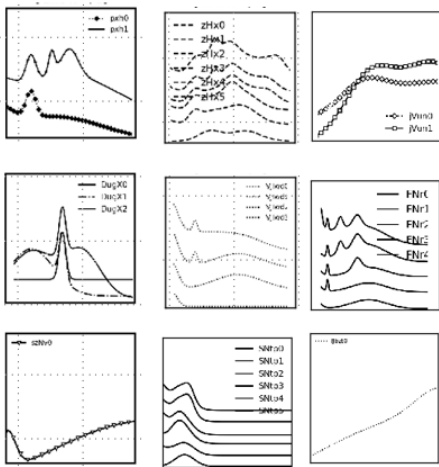
画像データ:

0-255の整数の数値をとり、そのサイズが360x360で1チャンネル(グレイタイプ)の画像配列(360,360,1)

教師データ:

1~6の整数値をとる整数配列(1)

画像データ



教師データ

```
[2] [6] [2]
[3] [4] [5]
[1] [6] [1]
```

4. プログラムリスト

プログラムはデータ作成本体 (Data_LineNumber.py) と関数ライブラリ (digitizer_lib) に収められている波形生成関数 (Line.py) の2つのモジュールに分けられる

4.1 データ作成本体 (Data_LineNumber.py)

1-9 行: 開始コメント
11-21 行: モジュールのインポート
29-35 行: 開始準備
37-106 行: データパラメータの設定
108-133 行: パラメータの確認
135-335 行: データの生成
337-353 行: チェック用の画像出力
362-383 行: モデルの保存
385 行: END


```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Nov.27,2019 / Fixed on Jun.30.2020
4
5 < ライン本数読み取りのための機械学習データ作成 >
6
7 @author: T.Kono
8
9 """
10
11 import cv2
12 import numpy as np
13 import matplotlib.pyplot as plt
14 import sys, string, random
15 import pickle, time
16 from datetime import datetime
17 import json
18
19 from keras.preprocessing.image import img_to_array
20
21 import digitizer_lib.line as line
22
23 # -----
24 #
25 #     Data Creating of Machine Learning for LineNumber
26 #
27 # -----
28
29 # --- Opning -----
30 # date and time
31 today = datetime.datetime.now().strftime('%Y%m%d-%H%M')
32
33 # title
34 print('\n< Data Creating of Machine Learning for LineNumber.'+today+'>')
35 comment = 'LineNumber_Data_'
36
37 # --- Data parameter define -----
38 # data dimension(360x360 fixed)
39 dim =360
40
41 #input data number
42 print('\nInput data number. if <= 99, datas are not saved')
43 loop = int(input(' ==> '))
44
45 # common parameter
46 com_n = 'common_para_360_2006251156.json'
47 with open(com_n, mode='r') as f:
```

```

48 com_para = json.load(f)
49
50 # parameter import
51 ndpi = com_para['para'][1]
52 axis_lw = com_para['para'][2]
53 f_s = com_para['para'][3]
54 weigt = com_para['para'][4]
55 g_val = com_para['para'][5]
56 '''g_val[0][1] はマーカー間隔の最小値と最大値(pixel)
57 g_val[2]はマーカーの大きさ
58 ...
59 pix_poit = round(ndpi/72) # pixel number of point
60 ww = dim/ndpi ; hh = ww # graph width (inchs)
61
62 # image type
63 ''' gray に固定
64 im_type = input(' Input image type, g(gray) or m(mono) => ')
65 '''
66 im_type = 'g'
67 if im_type == 'g':
68     imt = 'L';i_t='gray'
69 elif im_type == 'm':
70     imt = '1';i_t='mono'
71 else :
72     print('Image type is wrong') ;sys.exit()
73
74 # line type
75 ''' cross に固定
76 line_type = input(' Input line type, c(cross) or p(parallel) => ')
77 '''
78 line_type = 'c'
79 if line_type == 'c':
80     of_weight = [0,1,0.2,0.5,0.8];l_t='cross'
81 elif line_type == 'p':
82     of_weight = [1,1.2,1.5,2];l_t='para'
83 else :
84     print('Line type is wrong') ;sys.exit()
85
86 # マーカーの設定
87 marker = ['+', 'D', '^', 'o', 's', 'v', 'x', '_']
88
89 ''' marker sample
90 ['o', 's', 'v']
91 ['+', 'D', '^', 'o', 's', 'v', 'x']
92 ['*', '+', 'x', 'x', 'x', '8', '<', '>', 'D', 'H', \
93 '^', '_', 'd', 'h', 'o', 'p', 's', 'v', 'x', '|']
94 ...
95
96 mak_l=len(marker)
97
98 # p_weightは各ラインのピークの大きさを変化させる
99 p_weight = [0.8, 0.9, 1.1, 1.2, 1.3]
100
101 # Line number
102 cho_l = [1,2,3,4,5,6]
103
104 # Peak number (重ね合わせるガウス関数の数)
105 cho_p_l = [1,2,3]
106 cho_p_h = [3,4,5]
107
108 # --- Parameter confirmation -----

```

```

109 print('\n < Data parameter >')
110 print(' common parameter = ',com_n)
111 print(' dim = ',dim,
112       '\n dpi = ',ndpi,
113       '\n data number = ',loop )
114 print(' Graph size = ', '{:3.3}'.format(ww),
115       'x', '{:3.3}'.format(ww), 'inch')
116 print(' Line Number = ',cho_l)
117 print(' Line peak number = w/m',cho_p_l,
118       ', wo/m ', cho_p_h)
119 print(' Line width(normal) = ',axis_lw,
120       'point(',axis_lw*ndpi/72,'pixel)')
121 print(' Font size(normal) = ',f_s)
122 print(' Marker number = ',len(marker),marker)
123 print(' Marker size(normal) = ',g_val[2],
124       '\n Pitch of marker = ',
125       g_val[0], '-', g_val[1],
126       '\n scale factor = ',weigt)
127 print('\n comment = ',comment)
128
129 yn = input(' Data OK? y or n --> ')
130 if yn == 'n':
131     print('Parameter is wrong') ;sys.exit()
132
133 print('\n Process start ...'); s_time = time.time()
134
135 # --- Data create -----
136 # array initial set
137 ima_data = []
138 out_data = []
139 arrayimage = np.zeros([dim,dim,3],dtype=np.int8)
140 xxyy = np.zeros([2])
141
142 # Data create loop
143 for l in range(0,loop):
144
145     # marker on/off
146     mk_on_off = random.choice([1,0])
147
148     # ラインの本数をランダムに選択する
149     li_n = np.random.choice(cho_l)
150
151     '''マーカーの有無とラインの本数により波形のピーク数(ガンマ関数の重ね合わせ)を変える。
152     マーカーを描く場合もしくはラインの本数が3本以上では少なく、マーカーを描かないか
153     ラインの本数
154     が3本以下の場合は多くする
155     ...
156     cho_p = cho_p_h
157     if mk_on_off == 1 or li_n > 3:
158         cho_p = cho_p_l
159
160     # ピークの数(ガンマ関数の重ね合わせの数)をランダムに選択する
161     p_n = np.random.choice(cho_p)
162
163     plt.close()
164
165     fig = plt.figure(figsize = (ww, hh), dpi=ndpi)
166
167     xy = [0.0, 0.0] # origin position
168     xxyy = [1.0, 1.0] # axis length

```

```

168 # ベースとなる波形の生成
169 ff = line.line_n(dim, p_n)
170 fu = ff[0]
171 yn = ff[1]
172 pf = ff[2]
173 pk = ff[3]
174 g = ff[4]
175 ofs = ff[5]
176
177
178 weigt_ = []
179 y = np.zeros([dim])
180 yy = []
181 for j in range(0,li_n+1):
182
183     ''' Lin_n+1本の波形を形成する。
184     li_n+1 番目のラインは実際のラインに交差することで欠損を生じさせるためのダミー
185     '''
186     for i in range(0,p_n):
187
188         ''' ベースとなる波形から、各パラメータを指定範囲内でランダムに摂動させて
189         新しい波形を生成
190         '''
191         d_f = (np.random.randint(-30,30))/100 # 周波数の摂動量
192         d_k = (np.random.randint(2,20))/10 # 尖度の摂動量
193         d_of = (np.random.randint(-75,50))/100 # オフセットの摂動量
194
195         y_add = d_of + g[i]*np.exp(
196             -((fu-pf[i]+d_f)**2)/
197             ((2*(pk[i]*d_k)**2))
198         )
199
200         y = y + y_add
201
202         yy.append(y)
203
204 yy = np.asarray(yy)
205 ymax = np.max(yy)
206
207 # 最小値がX軸に張り付かないための調整
208 ymin = np.min(yy)-(np.random.randint(5,20))/100
209
210 # レンジをを0-1に入れるための調整
211 yn = (yy-ymin)/((ymax-ymin)*np.random.randint(103,
212 110)/100)
213
214 # legend position
215 loc_n = 'best'
216
217 out = (dim - (yn*(dim-1))).astype(np.int32)
218
219 # line width scale randame choice
220 scale = np.random.choice(weigt,4)/(np.random.randint(500,910)/1000)
221
222 plt.rcParams['axes.linewidth'] = axis_lw*scale[0] # axis linewidth
223 ax = fig.add_axes([xy[0], xy[1], xxy[0], xxy[1]]) # figure position
224
225 # marker style
226 if mk_on_off == 1:
227     mak_ = np.sort(np.random.choice(marker,
228 li_n+1,

```

```

229         replace=False
230     ))
231 else:
232     mak_ =[' ',' ',' ',' ',' ',' ']
233
234 # marker facecolor
235 fc_ = np.random.choice(['w','k'])
236
237 # marker-line plot
238 dif = np.random.randint(g_val[0],g_val[1])
239 idx_d = []
240
241 # maker の間隔を設定範囲内でランダムに変える
242 dif = np.random.randint(g_val[0],g_val[1])
243
244 # 3-5個のラベル文字をランダムに生成する
245 moj = np.random.randint(3,6)
246 d_n="".join(random.choices(string.ascii_letters, k=moj))
247
248 # figure plot
249 for k in range(0,li_n+1):
250     # +1 のラインは実際のラインに交差することで欠損を生じさせるためのダミー
251     # line plot
252     #- ダミーラインの線とマーカを白
253
254     l_col = 'k'
255     lab = d_n+str(k)
256     if k == li_n:
257         l_col = 'w'
258         lab = ''
259         fc_ = 'w'
260         ms = 5
261     else:
262         #マーカーの白抜きと黒塗りをランダムに選択
263         fc_ = np.random.choice(['w','k'])
264
265 # line style
266 if mk_on_off == 1:
267     '''マーカーがある場合は線種は実線、マーカーがない場合は線種はランダムに選択す
268     る
269     '''
270     ls_ = 'solid'
271 else:
272     ls_ = np.random.choice(['-','-.-',':', '--'])
273
274 ax.plot(fu, yn[k,:],
275         l_col,
276         ls = ls_,
277         linewidth = axis_lw*scale[0],
278         label = lab,
279         markevery = dif,
280         marker = mak_[k],
281         mfc = fc_,
282         mec = l_col,
283         ms = g_val[2]*scale[1]
284         )
285
286 # draw legend
287 fr_ch = np.random.choice([1,0])
288 ax.legend(loc = 'best',
289         fontsize = f_s*scale[2]*0.8,

```

```

289     framealpha = 0
290 )
291
292 ax.spines["right" ].set_color('none')
293 ax.spines["top"   ].set_color('none')
294 ax.spines["left"  ].set_color('none')
295 ax.spines["bottom"].set_color('none')
296
297 plt.ylim(0,1) ; plt.xlim(0,1)
298
299 # Buffer から figure データを取得
300 """画像出力は行わずBufferから直接画像の数値データを読み出す
301 ...
302 fig.canvas.draw()
303 buf = np.frombuffer (fig.canvas.tostring_rgb(),
304                     dtype=np.uint8
305                     )
306 imar = np.reshape(buf,(dim,dim,-1))
307
308 flg = l%3
309 if flg == 0 :
310     """3個に1個の割合でデータの前後にランダムな空白を上書きする
311     ...
312     of_x = np.random.randint(0,dim*0.10,2)
313     cv2.rectangle(imar,(0,0),(of_x[0],dim),
314                   (255,255,255),-1) # left
315     cv2.rectangle(imar,(dim-of_x[1],0),(dim,dim),
316                   (255,255,255),-1) # right
317
318 # 画像データ
319 gray_image = cv2.cvtColor(imar,cv2.COLOR_BGR2GRAY)
320 arrayimage = img_to_array(gray_image).astype(np.uint8)
321 ima_data.append(arrayimage)
322
323 # 教師データ(波形の本数)
324 out      = [li_n] # ダミーラインはカウントしない
325 out_data = np.append(out_data,np.array(out),axis=0)
326
327 # 全画像データ
328 ima_data = np.asarray(ima_data)
329
330 # 全教師データ(波形の本数)
331 out_data = np.asarray(out_data).astype(np.uint8)
332
333 # Execution time
334 e_time = time.time()
335 print(' ...Finished. Execution time (s) = ', '{:6.2f}'.format(e_time - s_time) )
336
337 # ---- Data check -----
338 plt.close()
339 print("\n Data figure check ")
340 n      = 20
341 fig    = plt.figure(figsize=(15,12))
342 n_list = np.random.randint(0,len(ima_data),n)
343 print(' * sample number = ',n_list)
344 for i in range(n):
345     nn = n_list[i]
346     ax = plt.subplot(4,5,i+1)
347     plt.imshow(np.reshape(ima_data[nn,:], (dim,-1)))
348     plt.gray()
349     plt.title(["Line N=",out_data[nn]])

```

```

350 ax.get_xaxis().set_visible(False)
351 ax.get_yaxis().set_visible(False)
352
353 plt.show()
354
355 # Special exit
356 """データの数が99個以下の場合はデータの保存を行わないでプログラムを終了する
357 ...
358 if loop <= 99:
359     print('\n /// Data were not saved, because loop was < 100. ///')
360     sys.exit()
361
362 # --- Data save -----
363 axis_test_graf = {'data':ima_data,
364                  'out':out_data,
365                  'para':(dim,
366                          ndpi,
367                          com_n,
368                          marker,
369                          cho_l,
370                          cho_p,
371                          im_type,
372                          line_type
373                          )
374                  }
375
376
377 with open('./'+comment+str(dim)+'_'+str(loop)+'_'+str(today)+'.pik','wb') as f1:
378     pickle.dump(axis_test_graf,
379                 f1,
380                 protocol = 4
381                 )
382
383 print('\n >>> Data was saved : '+comment+str(dim)+'_'+str(loop)+'_'+str(today)+'.pik')
384
385 # _____ End of code _____

```

4.2 波形生成関数（Line.py）

1-7 行: 開始コメント
 9-9 行: モジュールのインポート
 15-38 行: 開始準備
 40-69 行: 波形の生成
 71 行: return

```

1  # -*- coding: utf-8 -*-
2  """
3  Created on Jun.22,2020 / Fixed on Jun.30,2020
4
5  @author: T.Kono
6
7  """
8
9  import numpy as np
10
11 # ..... Line Create Function .....
12
13 def line_n(nn,n,n):

```

```

14 """p_n個のガウス関数を重ね合わせて波形を生成する関数
15
16 入力
17     nn : data number
18     p_n : peak number
19
20 出力
21     x : xの値
22     y : yの値
23     p_f : ピークの中心値
24     p_k : ピークの尖度
25     g_ : 全体のゲイン
26     ofs : オフセット
27
28 """
29
30 # Xの値の生成
31 x = np.linspace(0,1,nn)
32 # yの領域生成
33 y = np.zeros([nn])
34
35 p_f = []
36 p_k = []
37 g_ = []
38 ofs = []
39
40 for i in range(0,p_n):
41
42     """ ガウス関数で波形を発生させる
43     下のパラメータをランダムに変えて、p_n本の波形を生成して重ね合わせる
44
45     fo : ピークの中心周波数
46     pk : ピークの尖度
47     h : 全体のゲイン
48     ofse : オフセット量
49
50     """
51
52     fc = (np.random.randint(-30,130))/100
53     pk = (np.random.randint(10,500))/1000
54     ga = (np.random.randint(100,150))/100
55     ofse = (np.random.randint(10,100))/1000
56
57     # ガウス関数の生成
58     y_add = ofse + ga*np.exp(
59         -((x-fc)**2)/
60         (2*(pk)**2)
61     )
62     # ガウス関数の重ね合わせ
63     y = y + y_add
64
65     p_f.append(fc)
66     p_k.append(pk)
67     g_.append(ga)
68     ofs.append(ofse)
69
70
71 return x, y, p_f, p_k, g_, ofs
72

```

End