

# 波形数値化の機械学習プログラム説明書

波形数値化の機械学習プログラムは、波形の数により3つのプログラムが存在する。

- 1波形: Learning\_Digitizer\_1-line.py,
- 2波形: Learning\_Digitizer\_2-lines.py,
- 3波形: Learning\_Digitizer\_3-lines.py

これらの3つのプログラムは多くの部分が共通であり、ネット形状および入出力の一部分が異なるだけである。以下、本説明書では、異なる部分のみ適宜説明を加えるものとし、共通部分については特に断らないで記述する

## 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 入力と実行
    - 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\_Digitizer\_1-line.py, Learning\_Digitizer\_2-lines.py, Learning\_Digitizer\_3-lines.py)は機械学習を行い、グラフ画像内の波形を数値化する学習モデルを生成する。この学習モデルはグラフ数値化プログラム(WaveForm Digitizer.py)で用いる。

複数の波形を持つグラフでは、波形の色やマーカーの違いで各波形を区別している。グラフ数値化プログラム(WaveForm Digitizer.py)では、色分解することで波形を分離し、さらに本数判別を行い、波形本数が1~3本のグラフには本稿のLearning\_Digitizerを用いることにより波形の数値化を実行する。同一色で4本以上の波形が存在するグラフに対しては未対応である

## 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

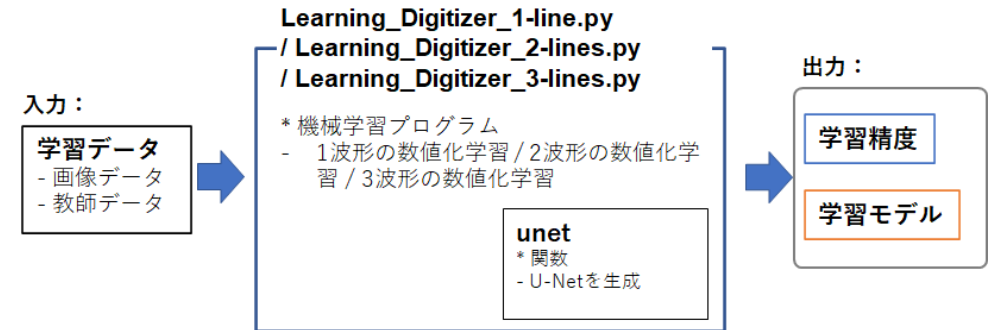
- matplotlib 2.2.2

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resnt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

プログラムは「学習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) Learning\_Digitizer\_1-line.py

- 1-27 行: 開始準備  
2-8 行: コメント  
10-27 行: モジュールのインポート
- 28-210 行: 学習モデルを生成する関数  
40-67 行: モデルパラメータの記述  
69-209行: ResNetの生成
- 211-595 行: 機械学習の実行  
218-231 行: 開始準備  
233-244 行: データの読み込み  
246-270 行: データのインポートと正規化  
272-281 行: パラメータの設定  
283-301 行: モデル構築  
303-358 行: モデルコンパイル  
360-378 行: モデルフィット  
380-419 行: 予測と評価  
421-456 行: 精度の出力  
458-568 行: 結果のプロット

570-593 行: モデルの保存  
595 行: END

## 2) Learning\_Digitizer\_2-lines.py

- 1-27 行: 開始準備  
2-8 行: コメント  
10-26 行: モジュールのインポート
- 28-240 行: 学習モデルを生成する関数**  
38-110 行: モデルパラメータの記述  
112-239行: ResNetの生成
- 241-630 行: 機械学習の実行**  
249-262 行: 開始準備  
264-274 行: データの読み込み  
276-299 行: データのインポートと正規化  
301-310 行: パラメータの設定  
312-330 行: モデル構築  
332-389 行: モデルコンパイル  
391-409 行: モデルフィット  
411-452 行: 予測と評価  
454-493 行: 精度の出力  
495-604 行: 結果のプロット  
606-628 行: モデルの保存  
630 行: END

## 3) Learning\_Digitizer\_3-lines.py

- 1-27 行: 開始準備  
2-8 行: コメント  
10-26 行: モジュールのインポート
- 28-259行: 学習モデルを生成する関数**  
38-116 行: モデルパラメータの記述  
118-258 行: ResNetの生成
- 260-667 行: 機械学習の実行**  
270-283 行: 開始準備  
385-299 行: データの読み込み  
301-326 行: データのインポートと正規化  
328-335 行: パラメータの設定  
337-356 行: モデル構築  
358-415 行: モデルコンパイル  
417-443 行: モデルフィット  
445-489 行: 予測と評価  
491-530 行: 精度の出力  
532-641 行: 結果のプロット  
643-665 行: モデルの保存  
667 行: END

## 3.3 入出力

### 3.3.1 入力と実行

このプログラムを実行するには、まず、下記の例の様に

- 1 波形の場合は、Learning\_Digitizer\_1-line.py の235行
- 2 波形の場合は、Learning\_Digitizer\_2-lines.py の266行
- 3 波形の場合は、Learning\_Digitizer\_3-lines.py の287行

に データ名を打ち込み保存する

例: 2波形の場合  
Digitizer\_Data\_2-lin.pik というデータを読み込む

```
264 # --- Data read -----  
265 # data_n = input('%n Data --> ')  
266 | data_n = './data/Digitizer_Data_2-lin.pik';print(' Data:',data_n)  
267 | with open(data_n, mode='rb') as f:  
268 |     test_graf = pickle.load(f)  
269
```

次に、<2. 動作環境> を満たす環境下のpython上で実行する

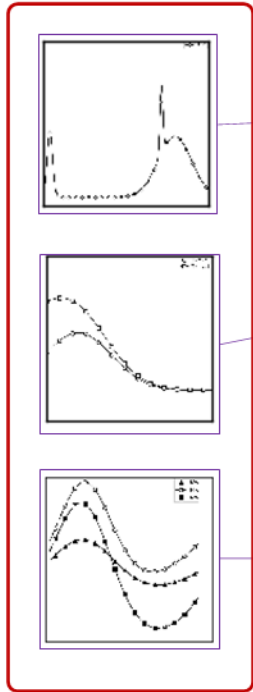
### 学習用データの要件

- 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.pik >
- 2つのキー'in', 'out'を持ち、'in'に画像データ、'out'に画像データと1対1に対応した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1024x1024で1チャンネル(グレイタイプ)の画像配列(1024,1024,1)

教師データ:  
0-1023の整数値をとり、1024 \* 波形数 (1~3) の整数配列

画像データ:  
(1024 x 1024)



教師データ:  
(1024 x 波形の数)

1波形:

[950, 947, 943, ..., 929, 930, 931]

1024個

2波形:

[608, 607, 606, ..., 837, 837, 837],  
[290, 289, 289, ..., 833, 833, 833]

1024個

3波形:

[558, 557, 556, ..., 547, 547, 546],  
[462, 460, 458, ..., 330, 329, 328],  
[523, 521, 519, ..., 617, 615, 613]

1024個

### 3.3.2 出力データ

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（2波形の学習の場合）をしめす

#### (1) 入力パラメータの確認

< RUN: MACHINE LEARNING OF WAVEFORM DIGITIZER\_200317-0700 >  
Data: ./data/Digitizer\_Data\_2-lin\_7m5000+l5000\_1024\_200317-1147.pik

Learning Process Start...

This is ResNet-26 with l2-regularizer and dropout  
L2-Regularizer = 0  
dropout = 0.1

Train\_n = 9000 , Test\_n = 1000  
epoch = 800  
batch = 40  
first\_filter = 16  
patience = 100

Optimizer: Adam  
loss: mean\_squared\_error

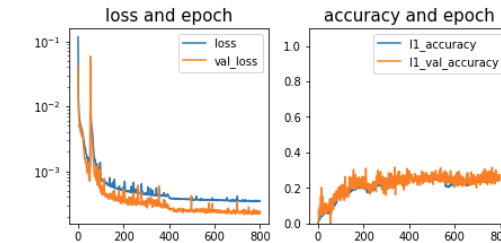
1-gpu running

#### (2) 精度と正答率

lossとaccの履歴と正答率の表示

< Execution time >  
Exe time (hr) = 31.94 ; end\_ep = 800 , exe/ep = 143.7

< loss and accuracy >  
loss = 3.52e-04 , val\_loss = 2.43e-04 ,  
l1-acc = 0.250 , l2-acc = 0.261

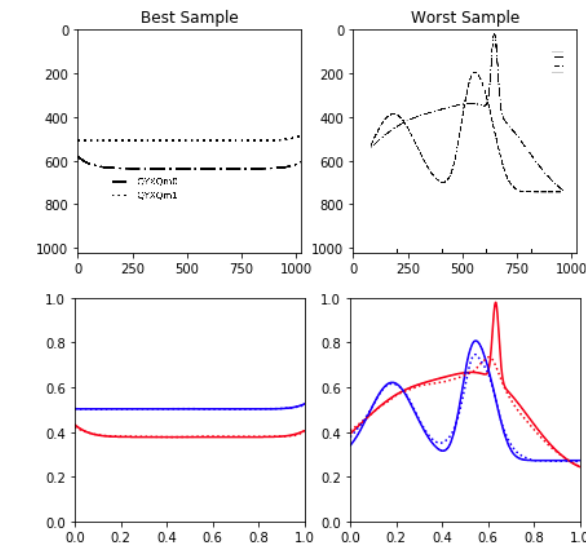


< Correct Rate >  
Correct answer rate = 0.811 , Error count = 189 / 1000 (sl = 0.03 )  
Note : 10% before and after data are not evaluated. Because there may be a blank.

#### (3) 画像での比較

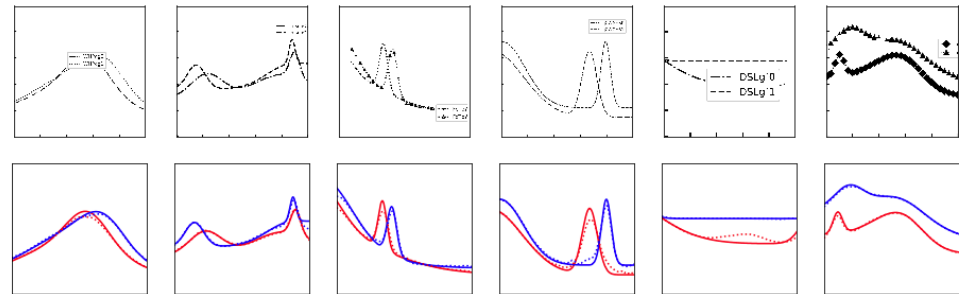
1) 最もうまく数値化できた画像と最も悪い画像の比較

< Best and Worst Sample >  
Best Sample No.= 303 , diff. = 0.00  
Worst Sample No.= 959 , diff. = 0.29  
upper: graph, lower: comparison of value



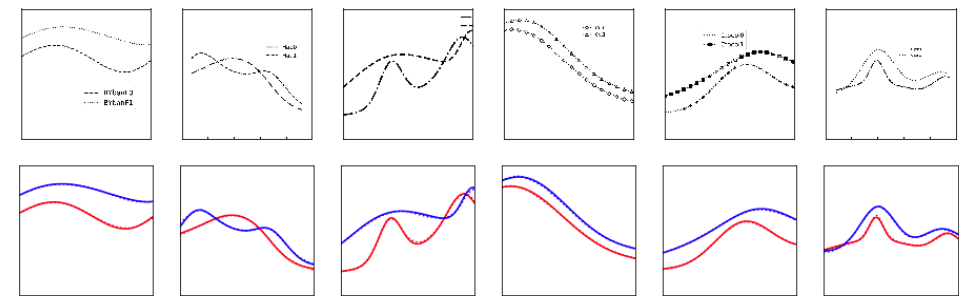
2) うまく数値化できなかった画像の中でランダムに6個を取り出して表示

```
< NG sample >
test number = [323 782 513 482 918 39]
```



3) うまく予測できた画像の中でランダムに6個を取り出して表示

```
< OK sample >
test number = [302 679 880 8 916 12]
```



(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  
\* 下の例は、モデル形状は保存せず、モデルは保存した場合

```
< Model save and print >
Print the model shape? y or n --> n
> Model was NOT printed.

Save the learned model? y or n --> y
> Model was saved: Digitizer_1024_2-BranchOut_RN26_bt40ft16ep800_200317-0700.h5
```

5. パラメータ

このプログラムのパラメータは1024x1024の大きさの数値化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できる。ただし計算環境によってはGPUやCPUのメモリ不足などのため、計算可能な組み合わせには限界が生じる場合がある

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	0.9, 0.1	合計して 1

変数名	説明	デフォルト値	可能な値
dim	画像の大きさ	1024	256, 360, 1024は検証済
m_n	ResNetの種類	RN26	RN18, RN34, RN50, RN26は検証済
reg_f	L2-Regularizer	0	0-1
drop_n	ドロップアウトの割合	0.1	0-1
n_epoch	最大エポック数	400	1-
n_batch	バッチ数	60	1-
first_fil_n	開始フィルタ数	16	8, 16, 24, 32は検証済
n_pat	EarlyStopで計算を打ち切る変動の最大数	100	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

6. プログラムリスト

6.1 1 波形数値化プログラム (Learning\_Digitizer\_1-line.py)

概要

- 1. 1-27 行: 開始準備  
2-8 行: コメント  
10-27 行: モジュールのインポート
- 2. 28-210 行: 学習モデルを生成する関数  
40-67 行: モデルパラメータの記述  
69-209行: ResNetの生成
- 3. 211-595 行: 機械学習の実行  
218-231 行: 開始準備  
233-244 行: データの読み込み  
246-270 行: データのインポートと正規化  
272-281 行: パラメータの設定  
283-301 行: モデル構築  
303-358 行: モデルコンパイル  
360-378 行: モデルフィット  
380-419 行: 予測と評価  
421-456 行: 精度の出力  
458-568 行: 結果のプロット  
570-593 行: モデルの保存  
595 行: END

全リスト

```

1  #-*- coding: utf-8 -*-
2  """"
3  Created on Des.24,2019 / Fixed on Jun.30,2020
4
5  < 1本のラインを持つグラフを数値化するための機械学習 >
6
7  @author: T.KONO
8  """"
9
10 import numpy as np
11 import matplotlib.pyplot as plt
12 from keras.callbacks import EarlyStopping
13 from sklearn.model_selection import train_test_split
14 import pickle, time, sys
15 from datetime import datetime
16 from keras.layers import (Input,
17                             Conv2D,
18                             Add,
19                             Activation,
20                             Dense,
21                             GlobalAveragePooling2D,
22                             MaxPooling2D,
23                             Dropout
24                             )
25 from keras.models import (Model)
26 from keras.regularizers import l2
27
28 # ----- ResNet Model Build Function -----
29 def resnet(input_shape,
30           num_outputs,
31           m_name,
32           n_fil,
33           l_f,
34           drp_n,
35           ln
36           ):
37
38     com = ' with l2-regularizer and dropout'
39
40     if m_name == 'RN18':
41         print('\n This is ResNet-18' + com)
42         nb_blocks = [2,2,2,2]
43         wide = 2
44         block = 'plane'
45
46     elif m_name == 'RN34':
47         print('\n This is ResNet-34' + com)
48         nb_blocks = [3,4,6,3]
49         wide = 2
50         block = 'plane'
51
52     elif m_name == 'RN50':
53         print('\n This is ResNet-50' + com)
54         nb_blocks = [3,4,6,3]
55         wide = 2
56         block = 'bottleneck'
57
58     else:
59         print('\n ???')
60

```

```

61 print(' L2-Regularizer =',l_f)
62 print(' dropout      =',drp_n)
63
64 input = Input(shape=input_shape)
65 n_filter = n_fil
66
67 reg_f = l_f
68
69 # --- 1st Pre convolution -----
70 X = input
71 X = Conv2D(filters=n_fil,
72            kernel_size=(3,3),
73            strides=(2,2),
74            padding="same",
75            kernel_regularizer=l2(reg_f)
76            )(X)
77 X = Activation("relu")(X)
78
79 # pooling
80 X = MaxPooling2D(pool_size=(3,3),
81                  strides=(2, 2),
82                  padding='same'
83                  )(X)
84
85 # --- 2nd Pre convolution -----
86 X = Conv2D(filters=n_fil,
87            kernel_size=(3,3),
88            strides=(2,2),
89            padding="same",
90            kernel_regularizer=l2(reg_f)
91            )(X)
92
93 X = Activation("relu")(X)
94
95 # pooling
96 X = MaxPooling2D(pool_size=(3,3),
97                  strides=(2,2),
98                  padding='same'
99                  )(X)
100
101 # --- Residual Net -----
102 shortcut = X
103
104 if block == 'plane':
105     for i, repete in enumerate(nb_blocks):
106         for j in range(repete):
107             if i>0 and j == 0:
108                 shortcut = Conv2D(n_filter,
109                                   kernel_size=(1, 1),
110                                   strides=(2, 2),
111                                   kernel_regularizer=l2(reg_f)
112                                   )(shortcut)
113
114                 X = Activation("relu")(X)
115                 X = Dropout(drp_n)(X)
116                 X = Conv2D(n_filter,
117                           kernel_size=(3,3),
118                           strides=(2,2),
119                           padding="same",
120                           kernel_regularizer=l2(reg_f)
121                           )(X)

```

```

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

```

```

183                     padding="same",
184                     kernel_regularizer=l2(reg_f)
185                     )(X)
186
187             X = Activation("relu")(X)
188             X = Dropout(drp_n)(X)
189             X = Conv2D(n_filter * 4,
190                       kernel_size=(1,1),
191                       padding="same",
192                       kernel_regularizer=l2(reg_f)
193                       )(X)
194
195             # ショートカットとマージ
196             X = Add()(X, shortcut)
197             shortcut = X
198
199             n_filter *= wide
200
201             X = Activation("relu")(X)
202
203             y = GlobalAveragePooling2D()(X)
204             y = Dropout(drp_n)(y)
205             y = Dense(int(num_outputs))(y)
206
207             model = Model(inputs=[input], outputs=[y])
208
209             return model
210
211 # -----
212 #             Machine Learning for 1-line Waveform Digitizer
213 #             - Mulch GPU
214 #             - ResNet with dropout and l2-regularizer
215 #             - lr_scheduled
216 # -----
217
218 # --- Opening -----
219 # Date and time
220 today = datetime.datetime.now().strftime('%Y-%m-%d-%H%M')
221 print("\n< Machine Learning for 1-line Waveform Digitizer_",
222       today, '>')
223
224 # Start time
225 s_time = time.time()
226
227 # --- GPU Number
228 gpu_n = 2
229
230 # Pass/NG Slice Level, difference between predict and target
231 sl = 0.03
232
233 # --- Data read -----
234 # data_n = input('\n Data --> ')
235 data_n = './data/Digitizer_Data_1-lin_I5000+7m5000_1024_200325-1053.pik'
236 print(' Data:', data_n)
237 with open(data_n, mode='rb') as f:
238     test_graf = pickle.load(f)
239
240 dim = 1024 #test_graf['para'][0]
241 li_n = 1 #test_graf['para'][3]
242 # print(' Dimension:', dim, ', Line-number:', li_n)
243

```

```

244 print('\nLearning Process Start...')
245
246 # --- Data import and normalize -----
247 X = test_graf['data']
248 # X-normalize and black-white reversed
249 if np.max(X[0,:]) == 255:
250     nor = 255 # case of gray data
251 else:
252     nor = 1 # case of monochro data
253 X = 1 - X/nor
254
255 # Y-normalize
256 Y = np.reshape(test_graf['out'],(-1,li_n*dim))/dim
257
258 # input/output shape
259 n_in = len(X[0])
260 n_out = len(Y[0])
261
262 # Data split
263 x_train, x_test, y_train, y_test = train_test_split(
264     X, Y,
265     test_size = 0.1,
266     train_size = 0.9,
267     random_state= 0
268 )
269 # Delet large data
270 del X ; del test_graf
271
272 # --- Iteration parameter -----
273 m_n = 'RN34'
274
275 reg_f = 1e-6
276 n_drop = 0
277 n_epoch = 300
278 n_batch = 40
279 first_fil_n = 16
280 n_pat = 100
281 flag_ver = 0
282
283 # --- Model Build -----
284 input_ = x_train.shape[1:]
285 model = resnet(input_,
286               n_out,
287               m_n,
288               first_fil_n,
289               reg_f,
290               n_drop,
291               li_n
292 )
293
294 # print prameter
295 print('\n Train_n = ',len(x_train),', Test_n =',len(x_test))
296 print(' epoch = ',n_epoch)
297 print(' batch = ',n_batch)
298 print(' first_filter = ',first_fil_n)
299 print(' patience = ',n_pat)
300
301 # print("\n* model_summary") ; model.summary()
302
303 # --- Model compile -----
304 # Optimizer setting

```

```

305 from keras.optimizers import Adam
306 optimizer = Adam()
307 print('\n Opimizer: Adam')
308
309 """## Optimizer Adam
310 from keras.optimizers import Adam
311 optimizer = Adam(lr=0.001)
312 print('\n Opimizer: Adam')
313
314 ## Optimizer SGD + Momentum
315 from keras.optimizers import SGD
316 optimizer = SGD(lr=0.01,decay=1e-6, momentum=0.9, nesterov=True)
317 print('\n Opimizer: SGD + Momentum')
318 """
319 # 学習率をepoch数により切り替える
320 from keras.callbacks import LearningRateScheduler
321 def lr_schedul(epoch):
322     """default: adam:0.001, SDG:0.01
323     """
324     x = 0.001
325     if epoch >= 100:
326         x = 0.00075
327     if epoch >= 200:
328         x = 0.0005
329     if epoch >= 400:
330         x = 0.0005
331     return x
332
333 lr_decay = LearningRateScheduler(lr_schedul)
334
335 # loss setting
336 loss = 'mean_squared_error'
337 print(' loss: ',loss)
338
339 # compile
340 if not gpu_n == 1 :
341     # Multi GPU
342     print('\n',str(gpu_n)+'-gpu running ')
343     from keras.utils import multi_gpu_model
344     parallel_model = multi_gpu_model(model, gpus=gpu_n)
345     parallel_model.compile(loss = loss,
346                          optimizer = optimizer,
347                          metrics = ['accuracy']
348 )
349 else:
350     # 1-GPU
351     print('\n 1-gpu running ')
352     model.compile(loss = 'mean_squared_error',
353                  optimizer = optimizer,
354                  metrics = ['accuracy']
355 )
356
357 early_stopping = EarlyStopping(monitor='val_loss',
358                                patience=n_pat)
359
360 # --- Model fitting -----
361 if not gpu_n == 1 :
362     # Multi GPU
363     hist = parallel_model.fit(x_train, y_train,
364                              epochs = n_epoch,
365                              batch_size = n_batch,

```

```

366         validation_data = (x_test, y_test),
367         verbose         = flag_ver,
368         callbacks       = [early_stopping,lr_decay]
369     )
370 else:
371     # 1-GPU
372     hist = model.fit(x_train, y_train,
373                     epochs      = n_epoch,
374                     batch_size  = n_batch,
375                     validation_data = (x_test,y_test),
376                     verbose     = flag_ver,
377                     callbacks   = [early_stopping,lr_decay]
378                 )
379
380 # --- Predict and evaluation -----
381 predict_ = (model.predict(x_test, batch_size = n_batch))
382
383 dif = abs((y_test - predict_))
384 '''グラフの前後10%は空白の可能性がある
385 この領域では精度の評価をしない
386 '''
387 spc = int(dim*0.1)
388 dif_l = np.reshape(dif,(-1,li_n,dim))
389 dif_ll = np.reshape(dif_l[:,,,:spc:dim-spc],(-1,(dim-2*spc)*li_n))
390
391 n_ep = len(hist.history['loss'])
392
393 # error count
394 pp_list = []
395 pass_list = []; ng_list=[]
396
397 for i in range(0,len(y_test)):
398     # error list
399     pp = np.max(dif_ll[i,:])
400     pp_list.append(pp)
401     if pp > sl:
402         ng_list.append(i) # NG list
403     else:
404         pass_list.append(i) # PASS list
405
406 # number of NG
407 nonzero = len(ng_list)
408 # pass rate
409 cor_pr = 1 - nonzero/(len(dif_ll))
410
411 # Execution time
412 e_time = time.time()
413 print('... Finished. ')
414 print('\n< Execution time > ')
415 print(' Exe time (hr) = ', '{:6.2f}'.format(e_time - s_time),
416       ', end_ep = ',n_ep,
417       ', exe/ep = ',
418       '{:4.1f}'.format((e_time - s_time)/n_ep)
419     )
420
421 # --- Print out of results -----
422 # loss and accuracy
423 print("\n< loss and accuracy > ")
424
425 print(' loss = ', '{:4.2e}'.format(hist.history['loss'][n_ep-1]),
426       ', val_loss = ', '{:4.2e}'.format(hist.history['val_loss'][n_ep-1]),

```

```

427       ',\n l1-acc = ', '{:4.3f}'.format(hist.history['acc'][n_ep-1])
428     )
429
430 # loss history
431 fig = plt.figure(figsize=(7, 7))
432 plt.subplot(2, 2, 1)
433 x_ep = range(len(hist.history['loss']))
434 plt.plot(x_ep,hist.history['loss'],label="loss")
435 plt.plot(x_ep,hist.history['val_loss'],label="val_loss")
436 plt.yscale('log')
437 plt.legend(loc='best')
438 plt.title("loss and epoch",fontsize=15)
439
440 # accuracy history
441 plt.subplot(2, 2, 2)
442 x_ep = range(len(hist.history['acc']))
443 plt.plot(x_ep,hist.history['acc'],label="acc")
444 plt.plot(x_ep,hist.history['val_acc'],label="l1_acc")
445 plt.title("accuracy and epoch",fontsize=15)
446 plt.ylim(0, 1.1)
447 plt.legend(loc='best')
448 plt.show()
449
450 # error
451 print('\n< Correct Rate > ')
452 print(' c_r = ', '{:4.3f}'.format(cor_pr),
453       ', Error count = ',nonzero,'/',len(dif),
454       ' (sl = ', '{:3.2f}'.format(sl),' )'
455     )
456 print(' Note : 10% before and after data are not evaluated. Because there may be a
457       blank. ')
458
459 # --- Result plotting -----
460 # Best & Worst
461 print("\n< Best and Worst Sample > ")
462 fig = plt.figure(figsize=(7,7))
463 # Best sample index
464 pp_min = np.amin(pp_list)
465 kp = pp_list.index(pp_min) # Best sample index
466 print(' Best Sample No.= ', '{:3.0f}'.format(kp),
467       ', diff. = ', '{:4.2f}'.format(pp_min))
468
469 # Worst sample index
470 pp_max = np.amax(pp_list)
471 kg = pp_list.index(pp_max)
472 print(' Worst Sample No.= ', '{:3.0f}'.format(kg),
473       ', diff. = ', '{:4.2f}'.format(pp_max))
474 print(' upper: graph, lower: comparison of value')
475 cor_list = ['r','b','g','c','m','y','w']
476 f = np.linspace(0,1,dim)
477 # Best graph
478 plt.subplot(2,2,1)
479 plt.imshow(1-x_test[kp,:,:,0])
480 plt.title("Best Sample",fontsize=12)
481 plt.gray()
482 # comparison
483 plt.subplot(2,2,3)
484 for ii in range(0,li_n):
485     plt.plot(f,1-y_test[kp,dim*ii:dim*(ii+1)],
486             cor_list[ii],label="Target")
487 for ii in range(0,li_n):

```

```

487 plt.plot(f,1-predict_[kp,dim*ii:dim*(ii+1)],
488          cor_list[ii]+'.',label="Predict")
489 plt.ylim(0,1);plt.xlim(0,1)
490
491 # Worst graph
492 plt.subplot(2,2,2)
493 plt.imshow(1-x_test[kg,::,0])
494 plt.title("Worst Sample",fontsize=12)
495 plt.gray()
496
497 # comparison
498 plt.subplot(2,2,4)
499 for ii in range(0,li_n):
500     plt.plot(f,1-y_test[kg,dim*ii:dim*(ii+1)],
501             cor_list[ii],label="Target")
502 for ii in range(0,li_n):
503     plt.plot(f,1-predict_[kg,dim*ii:dim*(ii+1)],
504             cor_list[ii]+'.',label="Predict")
505 plt.ylim(0,1);plt.xlim(0,1)
506 plt.show()
507
508 # NG sample plot
509 print("\n< NG sample >')
510 fig = plt.figure(figsize=(16, 5))
511 n = 6
512 n_list = np.random.choice(ng_list, n)
513 print(' test number = ', n_list)
514
515 for i in range(n):
516     nn = n_list[i]
517     # graph
518     ax = plt.subplot(2,n,i+1)
519     plt.imshow(1-x_test[nn,::,0])
520     plt.gray()
521     ax.get_xaxis().set_visible(False)
522     ax.get_yaxis().set_visible(False)
523     # comparison
524     ax = plt.subplot(2,n,n+i+1)
525     for ii in range(0,li_n):
526         plt.plot(f,1-y_test[nn,dim*ii:dim*(ii+1)],
527                 cor_list[ii],label="Target")
528     for ii in range(0,li_n):
529         plt.plot(f,1-predict_[nn,dim*ii:dim*(ii+1)],
530                 cor_list[ii]+'.',label="Predict")
531     plt.ylim(0,1);plt.xlim(0,1)
532     ax.get_xaxis().set_visible(False)
533     ax.get_yaxis().set_visible(False)
534 plt.show()
535
536 # OK sample plot
537 print("\n< OK sample >')
538
539 # special exit
540 if len(pass_list) == 0:
541     print("\n *** The Proceess was terminated, because an OK sample does not exist. *** \n')
542     sys.exit()
543
544 # plot
545 fig = plt.figure(figsize=(16, 5))
546 n = 6
547 n_list = np.random.choice(pass_list, n)

```

```

548 print(' test number = ', n_list)
549 for i in range(n):
550     nn = n_list[i]
551     # graph
552     ax = plt.subplot(2,n,i+1)
553     plt.imshow(1-x_test[nn,::,0])
554     plt.gray()
555     ax.get_xaxis().set_visible(False)
556     ax.get_yaxis().set_visible(False)
557     # comparison
558     ax = plt.subplot(2,n,n+i+1)
559     for ii in range(0,li_n):
560         plt.plot(f,1-y_test[nn,dim*ii:dim*(ii+1)],
561                 cor_list[ii],label="Target")
562     for ii in range(0,li_n):
563         plt.plot(f,1-predict_[nn,dim*ii:dim*(ii+1)],
564                 cor_list[ii]+'.',label="Predict")
565     plt.ylim(0,1);plt.xlim(0,1)
566     ax.get_xaxis().set_visible(False)
567     ax.get_yaxis().set_visible(False)
568 plt.show()
569
570 # --- Model print and save-----
571 print("\n< Model save and print >')
572 m_para = m_n+'_bt'+str(n_batch)+'ft'+str(first_fil_n)+'ep'+str(n_ep)
573 model_n = 'Digitizer_'+str(li_n)+'_lin_'+str(dim)+'_'+m_para+'_'+today
574 mp_flag = input(' Print the model shape? y or n --> ')
575
576 # model print
577 if mp_flag == 'y' :
578     from keras.utils import plot_model
579     plot_model(model, to_file = './model/'+model_n+'.png', show_shapes=True)
580     print(' > Model was printed: '+model_n+'.png ')
581 else :
582     print(' > Model was NOT printed.')
583
584 # model save
585 ms_flag = input('\n Save the learned model? y or n --> ')
586 if ms_flag == 'y' :
587     model.save('./model/'+model_n+'.h5')
588     print(' > Model was saved: '+model_n+'.h5')
589 else :
590     print(' > Model was NOT saved.')
591
592 # del large data
593 del x_train ; del x_test
594
595 # _____ END of code _____

```

## 6.2 2 波形数値化プログラム（Learning\_Digitizer\_2-lines.py）

### 概要

- 1-27 行: 開始準備  
2-8 行: コメント  
10-26 行: モジュールのインポート
- 28-240 行: 学習モデルを生成する関数  
38-110 行: モデルパラメータの記述

112-239行: ResNetの生成  
3. **241-630 行: 機械学習の実行**  
249-262 行: 開始準備  
264-274 行: データの読み込み  
276-299 行: データのインポートと正規化  
301-310 行: パラメータの設定  
312-330 行: モデル構築  
332-389 行: モデルコンパイル  
391-409 行: モデルフィット  
411-452 行: 予測と評価  
454-493 行: 精度の出力  
495-604 行: 結果のプロット  
606-628 行: モデルの保存  
630 行: END

## 全リスト

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Des.24,2019 / Fixed on Jun.30,2020
4
5 < 2本のラインを持つグラフを数値化するための機械学習 >
6
7 @author: T.KONO
8 """
9
10 import numpy as np
11 import matplotlib.pyplot as plt
12 from keras.callbacks import EarlyStopping
13 from sklearn.model_selection import train_test_split
14 import pickle, time, sys
15 from datetime import datetime
16 from keras.layers import (Input,
17                             Conv2D,
18                             Add,
19                             Activation,
20                             Dense,
21                             GlobalAveragePooling2D,
22                             MaxPooling2D,
23                             Dropout
24                             )
25 from keras.models import (Model)
26 from keras.regularizers import l2
27
28 # ----- ResNet Model Build Function -----
29 def resnet(input_shape,
30           num_outputs,
31           m_name,
32           n_fil,
33           l_f,
34           drp_n,
35           ln
36           ):
37
38     com = ' with l2-regularizer and dropout'
39
40     if m_name == 'RN18':
```

```
41     print('\n This is ResNet-18'+com)
42     nb_blocks = [2,2,2,2]
43     wide = 2
44     block = 'plane'
45
46     elif m_name == 'RN34':
47         print('\n This is ResNet-34'+com)
48         nb_blocks = [3,4,6,3]
49         wide = 2
50         block = 'plane'
51
52     elif m_name == 'RN50':
53         print('\n This is ResNet-50'+com)
54         nb_blocks = [3,4,6,3]
55         wide = 2
56         block = 'bottleneck'
57
58     elif m_name == 'RN101':
59         print('\n This is ResNet-101'+com)
60         nb_blocks = [3,4,23,3]
61         wide = 2
62         block = 'bottleneck'
63
64     elif m_name == 'RN152':
65         print('\n This is ResNet-152'+com)
66         nb_blocks = [3,4,36,3]
67         wide = 2
68         block = 'plane'
69
70     elif m_name == 'W-RN':
71         print('\n This is Wide-ResNet '+com)
72         nb_blocks = [3,3,3]
73         wide = 3
74         block = 'plane'
75
76     elif m_name == 'RN26':
77         print('\n This is ResNet-26'+com)
78         nb_blocks = [2,2,2,2,2,2]
79         wide = 2
80         block = 'plane'
81
82     elif m_name == 'RN46':
83         print('\n This is ResNet-46'+com)
84         nb_blocks = [3,3,3,4,6,3]
85         wide = 2
86         block = 'plane'
87
88     elif m_name == 'RN22':
89         print('\n This is ResNet-22'+com)
90         nb_blocks = [2,2,2,2,2]
91         wide = 2
92         block = 'plane'
93
94     elif m_name == 'RN40':
95         print('\n This is ResNet-40'+com)
96         nb_blocks = [3,3,4,6,3]
97         wide = 2
98         block = 'plane'
99
100     else:
101         print('\n ???')
```

```

102
103 print(' L2-Regularizer =',l_f)
104 print(' dropout      =',drp_n)
105
106 input = Input(shape=input_shape)
107 X = input
108 n_filter = n_fil
109
110 reg_f = l_f
111
112 # --- Pre convolution -----
113 X = Conv2D(filters=n_fil,
114            kernel_size=(3,3),
115            strides=(2,2),
116            padding="same",
117            kernel_regularizer=l2(reg_f)
118            )(X)
119 X = Activation("relu")(X)
120 # pooling
121 X = MaxPooling2D(pool_size=(3,3),
122                  strides=(2, 2),
123                  padding='same'
124                  )(X)
125
126 # --- Residual Net -----
127 shortcut = X
128
129 if block == 'plane':
130     for i, repete in enumerate(nb_blocks):
131         for j in range(repete):
132             if i>0 and j == 0:
133                 shortcut = Conv2D(n_filter,
134                                   kernel_size=(1, 1),
135                                   strides=(2, 2),
136                                   kernel_regularizer=l2(reg_f)
137                                   )(shortcut)
138
139                 X = Activation("relu")(X)
140                 X = Dropout(drp_n)(X)
141                 X = Conv2D(n_filter,
142                           kernel_size=(3,3),
143                           strides= (2,2),
144                           padding="same",
145                           kernel_regularizer=l2(reg_f)
146                           )(X)
147
148             else:
149                 X = Activation("relu")(X)
150                 X = Dropout(drp_n)(X)
151                 X = Conv2D(n_filter,
152                           kernel_size=(3,3),
153                           padding="same",
154                           kernel_regularizer=l2(reg_f)
155                           )(X)
156
157                 X = Activation("relu")(X)
158                 X = Dropout(drp_n)(X)
159                 X = Conv2D(n_filter,
160                           kernel_size=(3,3),
161                           padding="same",
162                           kernel_regularizer=l2(reg_f)

```

```

163             )(X)
164
165             # ショートカットとマージ
166             X = Add()([X, shortcut])
167             shortcut = X
168
169             n_filter *= wide
170
171         if block == 'bottleneck':
172             shortcut = Conv2D(n_filter * 4,
173                               kernel_size=(1, 1) ,
174                               kernel_regularizer=l2(reg_f)
175                               )(shortcut)
176
177         for i, repete in enumerate(nb_blocks):
178             for j in range(repete):
179                 if i>0 and j == 0:
180                     shortcut = Conv2D(n_filter * 4,
181                                       kernel_size=(1, 1),
182                                       strides=(2, 2),
183                                       kernel_regularizer=l2(reg_f)
184                                       )(shortcut)
185
186                     X = Activation("relu")(X)
187                     X = Dropout(drp_n)(X)
188                     X = Conv2D(n_filter,
189                               kernel_size=(1,1),
190                               strides= (2,2),
191                               padding="same",
192                               kernel_regularizer=l2(reg_f)
193                               )(X)
194
195                 else:
196                     X = Activation("relu")(X)
197                     X = Dropout(drp_n)(X)
198                     X = Conv2D(n_filter,
199                               kernel_size=(1,1),
200                               padding="same",
201                               kernel_regularizer=l2(reg_f)
202                               )(X)
203
204                     X = Activation("relu")(X)
205                     X = Dropout(drp_n)(X)
206                     X = Conv2D(n_filter,
207                               kernel_size=(3,3),
208                               padding="same",
209                               kernel_regularizer=l2(reg_f)
210                               )(X)
211
212                     X = Activation("relu")(X)
213                     X = Dropout(drp_n)(X)
214                     X = Conv2D(n_filter * 4,
215                               kernel_size=(1,1),
216                               padding="same",
217                               kernel_regularizer=l2(reg_f)
218                               )(X)
219
220                     # ショートカットとマージ
221                     X = Add()([X, shortcut])
222                     shortcut = X
223

```

```

224     n_filter *= wide
225
226     X = Activation("relu")(X)
227
228     # 2-Branch Out
229     y1 = GlobalAveragePooling2D()(X)
230     y1 = Dropout(drp_n)(y1)
231     y1 = Dense(int(num_outputs/2),name='l1')(y1)
232
233     y2 = GlobalAveragePooling2D()(X)
234     y2 = Dropout(drp_n)(y2)
235     y2 = Dense(int(num_outputs/2),name='l2')(y2)
236
237     model = Model(inputs=[input], outputs=[y1,y2])
238
239     return model
240
241 # -----
242 #         Machine Learning for 2-lines Waveform Digitizer
243 #         - Mulch GPU
244 #         - ResNet with dorpout and L2-regularizer
245 #         - lr_schedul
246 #         - branch out type
247 # -----
248
249 # --- Opening -----
250 # Date and time
251 today = datetime.today().strftime('%y%m%d-%H%M')
252 print("\n< Machine Learning for 2-lines Waveform Digitizer_",
253       today,'>')
254
255 # Start time
256 s_time = time.time()
257
258 # GPU Number
259 gpu_n = 1
260
261 # Pass/NG Slice Level: difference between predict and target
262 sl = 0.03
263
264 # --- Data read -----
265 # data_n = input('\n Data --> ')
266 data_n = './data/Digitizer_Data_2-lin.pik';print(' Data:',data_n)
267 with open(data_n, mode='rb') as f:
268     test_graf = pickle.load(f)
269
270 dim = 1024# test_graf['para'][0]
271 li_n = 2 # li_n = test_graf['para'][3]
272 # print(' Dimension:',dim,', Line-number:',li_n)
273
274 print("\nLearning Process Start...")
275
276 # --- Data import and normalize -----
277 X = test_graf['data']
278 # X-normalize and black-white reversed
279 if np.max(X[0,:]) == 255:
280     nor = 255     # case of gray data
281 else:
282     nor = 1       # case of monochro data
283 X = 1 - X/nor
284

```

```

285 # Y-normalize
286 Y = np.reshape(test_graf['out'],(-1,li_n*dim))/dim
287
288 # input/output shape
289 n_in = len(X[0])
290 n_out = len(Y[0])
291
292 # Data split
293 x_train, x_test, y_train, y_test = train_test_split(
294     X, Y,
295     test_size = 0.2,
296     train_size = 0.8
297 )
298 # Delet large data
299 del X ; del test_graf
300
301 # --- Iteration parameter -----
302 m_n = 'RN26'
303
304 reg_f = 0
305 n_drop = 0.1
306 n_epoch = 100
307 n_batch = 40
308 first_fil_n = 16
309 n_pat = 100
310 flag_ver = 0
311
312 # --- Model Built -----
313 input_ = x_train.shape[1:]
314 model = resnet(input_,
315               n_out,
316               m_n,
317               first_fil_n,
318               reg_f,
319               n_drop,
320               li_n
321               )
322
323 # print prameter
324 print('\n Train_n = ',len(x_train),', Test_n =',len(x_test))
325 print(' epoch = ',n_epoch)
326 print(' batch = ',n_batch)
327 print(' first filter = ',first_fil_n)
328 print(' patience = ',n_pat)
329
330 # print("\n* model_summary") ; model.summary()
331
332 # --- Model compile -----
333 # Optimizer setting
334 from keras.optimizers import Adam
335 optimizer = Adam()
336 print("\n Opimizer: Adam'")
337
338 ""## Optimizer Adam
339 from keras.optimizers import Adam
340 optimizer = Adam(lr=0.001)
341 print("\n Opimizer: Adam'")
342
343 ## Optimizer SGD + Momentum
344 from keras.optimizers import SGD
345 optimizer = SGD(lr=0.01,decay=1e-6, momentum=0.9, nesterov=True)

```

```

346 print('\n Optimizer: SGD + Momentum')
347 '''
348
349 # 学習率をepoch数により切り替える
350 from keras.callbacks import LearningRateScheduler
351 def lr_schedul(epoch):
352     # default: Adam:0.001, SDG:0.01
353
354     # adam
355     x = 0.001
356     if epoch >= 100:
357         x = 0.00075
358     if epoch >= 200:
359         x = 0.0005
360     if epoch >= 400:
361         x = 0.0005
362     return x
363
364 lr_decay = LearningRateScheduler(lr_schedul)
365
366 # loss setting
367 loss = 'mean_squared_error'
368 print(' loss: ',loss)
369
370 # Compile
371 if not gpu_n == 1:
372     # Multi GPU
373     print('\n',str(gpu_n)+'-gpu running ')
374     from keras.utils import multi_gpu_model
375     parallel_model = multi_gpu_model(model, gpus=gpu_n)
376     parallel_model.compile(loss = loss,
377                           optimizer = optimizer,
378                           metrics = ['accuracy']
379                           )
380 else:
381     # 1-GPU
382     print('\n 1-gpu running ')
383     model.compile(loss = 'mean_squared_error',
384                  optimizer = optimizer,
385                  metrics = ['accuracy']
386                  )
387
388 early_stopping = EarlyStopping(monitor='loss',
389                               patience=n_pat)
390
391 # --- Model fitting -----
392 if not gpu_n == 1:
393     # Multi GPU
394     hist = parallel_model.fit(x_train, [y_train[:,0:dim],y_train[:,dim:dim*2]],
395                             epochs = n_epoch,
396                             batch_size = n_batch,
397                             validation_data = (x_test, [y_train[:,0:dim],y_train[:,dim:dim*2]]),
398                             verbose = flag_ver,
399                             callbacks = [early_stopping,lr_decay]
400                             )
401 else:
402     # 1-GPU
403     hist = model.fit(x_train, [y_train[:,0:dim],y_train[:,dim:dim*2]],
404                     epochs = n_epoch,
405                     batch_size = n_batch,
406                     validation_data = (x_test, [y_test[:,0:dim],y_test[:,dim:dim*2]]),

```

```

407         verbose = flag_ver,
408         callbacks = [early_stopping,lr_decay]
409     )
410
411 # --- Predict and evaluation -----
412 predict = (model.predict(x_test, batch_size = n_batch))
413 predict = np.concatenate([predict[0], predict[1]],1)
414
415 dif = abs((y_test - predict))
416
417 '''グラフの前後10%は空白の可能性がある
418 この領域では精度の評価をしない
419 '''
420 spc = int(dim*0.1)
421 dif_l = np.reshape(dif,(-1,li_n,dim))
422 dif_ll = np.reshape(dif_l[:,::spc,dim:dim-spc],(-1,(dim-2*spc)*li_n))
423
424 n_ep = len(hist.history['loss'])
425
426 # error count
427 pp_list = []
428 pass_list = []; ng_list=[]
429
430 for i in range(0,len(y_test)):
431     # error list
432     pp = np.max(dif_ll[i,:])
433     pp_list.append(pp)
434     if pp > sl:
435         ng_list.append(i) # NG list
436     else:
437         pass_list.append(i) # Pass list
438
439 # number of NG
440 nonzero = len(ng_list)
441 # pass rate
442 cor_pr = 1 - nonzero/(len(dif_ll))
443
444 # --- Execution time
445 e_time = time.time()
446 print('... Finished. ')
447 print('\n< Execution time > ')
448 print(' Exe time (hr) = ', '{:6.2f}'.format((e_time - s_time)/3600),
449       ', end_ep = ',n_ep,
450       ', exe/ep = ',
451       '{:4.1f}'.format((e_time - s_time)/n_ep)
452       )
453
454 # --- Print out of results -----
455 # loss and accuracy
456 print('\n< loss and accuracy > ')
457
458 print(' loss = ', '{:4.2e}'.format(hist.history['loss'][n_ep-1]),
459       ', val_loss = ', '{:4.2e}'.format(hist.history['val_loss'][n_ep-1]),
460       '\n l1-acc = ', '{:4.3f}'.format(hist.history['l1_acc'][n_ep-1]),
461       ', l2-acc = ', '{:4.3f}'.format(hist.history['l2_acc'][n_ep-1])
462       )
463
464 # loss history
465 fig = plt.figure(figsize=(7, 7))
466 plt.subplot(2, 2, 1)
467 x_ep = range(len(hist.history['loss']))

```

```

468 plt.plot(x_ep,hist.history['loss'],label="loss")
469 plt.plot(x_ep,hist.history['val_loss'],label="val_loss")
470 plt.yscale('log')
471 plt.legend(loc='best')
472 plt.title("loss and epoch",fontsize=15)
473
474 # accuracy history
475 plt.subplot(2, 2, 2)
476 x_ep = range(len(hist.history['l1_acc']))
477 plt.plot(x_ep,hist.history['l1_acc'],label="l1_acc")
478 plt.plot(x_ep,hist.history['val_l1_acc'],label="l1_val_acc")
479 # plt.plot(x_ep,hist.history['l2_acc'],label="accuracy")
480 # plt.plot(x_ep,hist.history['val_l2_acc'],label="val_acc")
481 plt.yscale('log')
482 plt.title("accuracy and epoch",fontsize=15)
483 plt.ylim(0, 1.1)
484 plt.legend(loc='best')
485 plt.show()
486
487 # error
488 print("\n< Correct Rate >")
489 print(' Correct answer rate = ', '{:4.3f}'.format(cor_pr),
490       ', Error count = ',nonzero,'/',len(dif),
491       ', (sl = ', '{:3.2f}'.format(sl),')'
492       )
493 print(' Note : 10% before and after data are not evaluated. Because there may be a
blank.')
494
495 # --- Result plotting -----
496 # Best & Worst
497 print("\n< Best and Worst Sample >")
498 fig = plt.figure(figsize=(7,7))
499 # Best sample index
500 pp_min = np.amin(pp_list)
501 kp = pp_list.index(pp_min) # Best sample index
502 print(' Best Sample No.= ', '{:3.0f}'.format(kp),
503       ', diff. = ', '{:4.2f}'.format(pp_min))
504
505 # Worst sample index
506 pp_max = np.amax(pp_list)
507 kg = pp_list.index(pp_max)
508 print(' Worst Sample No.= ', '{:3.0f}'.format(kg),
509       ', diff. = ', '{:4.2f}'.format(pp_max))
510 print(' upper: graph, lower: comparison of value')
511 cor_list = ['r','b','g','c','m','y','w']
512 f = np.linspace(0,1,dim)
513 # Best graph
514 plt.subplot(2,2,1)
515 plt.imshow(1-x_test[kp,:,:,0])
516 plt.title("Best Sample",fontsize=12)
517 plt.gray()
518 # comparison
519 plt.subplot(2,2,3)
520 for ii in range(0,li_n):
521     plt.plot(f,1-y_test[kp,dim*ii:dim*(ii+1)],
522             cor_list[ii],label="Target")
523 for ii in range(0,li_n):
524     plt.plot(f,1-predict[kp,dim*ii:dim*(ii+1)],
525             cor_list[ii]+'.',label="Predict")
526 plt.ylim(0,1);plt.xlim(0,1)
527

```

```

528 # Worst graph
529 plt.subplot(2,2,2)
530 plt.imshow(1-x_test[kg,:,:,0])
531 plt.title("Worst Sample",fontsize=12)
532 plt.gray()
533
534 # comparison
535 plt.subplot(2,2,4)
536 for ii in range(0,li_n):
537     plt.plot(f,1-y_test[kg,dim*ii:dim*(ii+1)],
538             cor_list[ii],label="Target")
539 for ii in range(0,li_n):
540     plt.plot(f,1-predict[kg,dim*ii:dim*(ii+1)],
541             cor_list[ii]+'.',label="Predict")
542 plt.ylim(0,1);plt.xlim(0,1)
543 plt.show()
544
545 # NG sample plot
546 print("\n< NG sample >")
547 fig = plt.figure(figsize=(16, 5))
548 n = 6
549 n_list = np.random.choice(ng_list, n)
550 print(' test number = ', n_list)
551
552 for i in range(n):
553     nn = n_list[i]
554     # graph
555     ax = plt.subplot(2,n,i+1)
556     plt.imshow(1-x_test[nn,:,:,0])
557     plt.gray()
558     ax.get_xaxis().set_visible(False)
559     ax.get_yaxis().set_visible(False)
560     # comparison
561     ax = plt.subplot(2,n,i+1)
562     for ii in range(0,li_n):
563         plt.plot(f,1-y_test[nn,dim*ii:dim*(ii+1)],
564                 cor_list[ii],label="Target")
565     for ii in range(0,li_n):
566         plt.plot(f,1-predict[nn,dim*ii:dim*(ii+1)],
567                 cor_list[ii]+'.',label="Predict")
568     plt.ylim(0,1);plt.xlim(0,1)
569     ax.get_xaxis().set_visible(False)
570     ax.get_yaxis().set_visible(False)
571 plt.show()
572
573 # OK sample plot
574 print("\n< OK sample >")
575
576 # --- Special exit
577 if len(pass_list) == 0:
578     print("\n *** The Proceess was terminated, because an OK sample does not exist. *** \n")
579     sys.exit()
580 # plot
581 fig = plt.figure(figsize=(16, 5))
582 n = 6
583 n_list = np.random.choice(pass_list, n)
584 print(' test number = ', n_list)
585 for i in range(n):
586     nn = n_list[i]
587     # graph
588     ax = plt.subplot(2,n,i+1)

```

```

589 plt.imshow(1-x_test[nn, :, 0])
590 plt.gray()
591 ax.get_xaxis().set_visible(False)
592 ax.get_yaxis().set_visible(False)
593 # comparison
594 ax = plt.subplot(2,n,n+i+1)
595 for ii in range(0,li_n):
596     plt.plot(f,1-y_test[nn,dim*ii:dim*(ii+1)],
597             cor_list[ii],label="Target")
598 for ii in range(0,li_n):
599     plt.plot(f,1-predict[nn,dim*ii:dim*(ii+1)],
600             cor_list[ii]+'.',label="Predict")
601 plt.ylim(0,1);plt.xlim(0,1)
602 ax.get_xaxis().set_visible(False)
603 ax.get_yaxis().set_visible(False)
604 plt.show()
605
606 # --- Model Print and Save -----
607 print('\n< Model save and print >')
608 m_para = m_n+'_bt'+str(n_batch)+'ft'+str(first_fil_n)+'ep'+str(n_ep)
609 model_n = 'Digitizer_'+str(dim)+'_'+str(li_n)+'-lines_BranchOut_'+m_para+'_'+today
610 mp_flag = input(' Print the model shape? y or n --> ')
611 # model print
612 if mp_flag == 'y':
613     from keras.utils import plot_model
614     plot_model(model, to_file = './model/'+model_n+'.png', show_shapes=True)
615     print(' > Model was printed: '+model_n+'.png ')
616 else:
617     print(' > Model was NOT printed.')
618
619 # model save
620 ms_flag = input('\n Save the learned model? y or n --> ')
621 if ms_flag == 'y':
622     model.save('./model/'+model_n+'.h5')
623     print(' > Model was saved: '+model_n+'.h5')
624 else:
625     print(' > Model was NOT saved.')
626
627 # Del large data
628 del x_train ; del x_test
629
630 # _____ END of Code _____

```

### 6.3 3 波形数値化プログラム（Learning\_Digitizer\_3-lines.py）

#### 概要

- 1-27 行: 開始準備  
2-8 行: コメント  
10-26 行: モジュールのインポート
- 28-259 行: 学習モデルを生成する関数  
38-116 行: モデルパラメータの記述  
118-258 行: ResNetの生成
- 260-667 行: 機械学習の実行  
270-283 行: 開始準備  
385-299 行: データの読み込み  
301-326 行: データのインポートと正規化

328-335 行: パラメータの設定  
 337-356 行: モデル構築  
 358-415 行: モデルコンパイル  
 417-443 行: モデルフィット  
 445-489 行: 予測と評価  
 491-530 行: 精度の出力  
 532-641 行: 結果のプロット  
 643-665 行: モデルの保存  
 667 行: END

#### 全リスト

```

1  # -*- coding: utf-8 -*-
2  ""
3  Created on Des.24,2019 / Fixed on Jun.30,2020
4
5  < 3本のラインを持つグラフを数値化するための機械学習 >
6
7  @author: T.KONO
8  ""
9
10 import numpy as np
11 import matplotlib.pyplot as plt
12 from keras.callbacks import EarlyStopping
13 from sklearn.model_selection import train_test_split
14 import pickle, time, sys
15 from datetime import datetime
16 from keras.layers import (Input,
17                             Conv2D,
18                             Add,
19                             Activation,
20                             Dense,
21                             GlobalAveragePooling2D,
22                             MaxPooling2D,
23                             Dropout
24                             )
25 from keras.models import (Model)
26 from keras.regularizers import l2
27
28 # ----- ResNet Model Build Function -----
29 def resnet(input_shape,
30           num_outputs,
31           m_name,
32           n_fil,
33           l_f,
34           drp_n,
35           ln
36           ):
37
38     com = ' with l2-regularizer and dropout'
39
40     if m_name == 'RN18':
41         print('\n This is ResNet-18'+com)
42         nb_blocks = [2,2,2,2]
43         wide = 2
44         bottleneck = False
45
46     elif m_name == 'RN34':

```

```

47     print('\n This is ResNet-34'+com)
48     nb_blocks = [3,4,6,3]
49     wide = 2
50     bottleneck = False
51
52     elif m_name == 'RN50':
53         print('\n This is ResNet-50'+com)
54         nb_blocks = [3,4,6,3]
55         wide = 2
56         bottleneck = True
57
58     elif m_name == 'RN101':
59         print('\n This is ResNet-101'+com)
60         nb_blocks = [3,4,23,3]
61         wide = 2
62         bottleneck = True
63
64     elif m_name == 'RN152':
65         print('\n This is ResNet-152'+com)
66         nb_blocks = [3,4,36,3]
67         wide = 2
68         bottleneck = False
69
70     elif m_name == 'W-RN':
71         print('\n This is Wide-ResNet '+com)
72         nb_blocks = [3,3,3]
73         wide = 3
74         bottleneck = False
75
76     elif m_name == 'RN26':
77         print('\n This is ResNet-26'+com)
78         nb_blocks = [2,2,2,2,2,2]
79         wide = 2
80         bottleneck = False
81
82     elif m_name == 'RN46':
83         print('\n This is ResNet-46'+com)
84         nb_blocks = [3,4,6,3,3,3]
85         wide = 2
86         bottleneck = False
87
88     elif m_name == 'RN54':
89         print('\n This is ResNet-54'+com)
90         nb_blocks = [3,4,6,6,4,3]
91         wide = 2
92         bottleneck = False
93
94     elif m_name == 'RN22':
95         print('\n This is ResNet-22'+com)
96         nb_blocks = [2,2,2,2,2]
97         wide = 2
98         bottleneck = False
99
100    elif m_name == 'RN40':
101        print('\n This is ResNet-40'+com)
102        nb_blocks = [3,4,6,3,3]
103        wide = 2
104        bottleneck = False
105
106    else:
107        print('\n ???')

```

```

108
109    print(' L2-Regularizer =',l_f)
110    print(' dropout      =',drp_n)
111
112    input = Input(shape=input_shape)
113    X = input
114    n_filter = n_fil
115
116    reg_f = l_f
117
118    # --- 1st Pre convolution -----
119    X = Conv2D(filters=n_fil,
120               kernel_size=(3,3),
121               strides=(2,2),
122               padding="same",
123               kernel_regularizer=l2(reg_f)
124               )(X)
125    X = Activation("relu")(X)
126
127    # pooling
128    X = MaxPooling2D(pool_size=(3,3),
129                    strides=(2, 2),
130                    padding='same'
131                    )(X)
132
133    # --- ResNet -----
134    shortcut = X
135
136    if bottleneck == False:
137        for i, repete in enumerate(nb_blocks):
138            for j in range(repete):
139                if i>0 and j == 0:
140                    shortcut = Conv2D(n_filter,
141                                       kernel_size=(1,1),
142                                       strides=(2, 2),
143                                       kernel_regularizer=l2(reg_f)
144                                       )(shortcut)
145
146                    X = Activation("relu")(X)
147                    X = Dropout(drp_n)(X)
148
149                    X = Conv2D(n_filter,
150                               kernel_size=(3,3),
151                               strides= (2,2),
152                               padding="same",
153                               kernel_regularizer=l2(reg_f)
154                               )(X)
155
156                else:
157                    X = Activation("relu")(X)
158                    X = Dropout(drp_n)(X)
159
160                    X = Conv2D(n_filter,
161                               kernel_size=(3,3),
162                               padding="same",
163                               kernel_regularizer=l2(reg_f)
164                               )(X)
165
166                    X = Activation("relu")(X)
167                    X = Dropout(drp_n)(X)
168

```

```

169     X = Conv2D(n_filter,
170               kernel_size=(3,3),
171               padding="same",
172               kernel_regularizer=l2(reg_f)
173             )(X)
174
175     # ショートカットとマージ
176     X = Add()([X, shortcut])
177     shortcut = X
178
179     n_filter *= wide
180
181 if bottleneck == True:
182     shortcut = Conv2D(n_filter * 4,
183                     kernel_size=(1, 1),
184                     kernel_regularizer=l2(reg_f)
185                   )(shortcut)
186
187 for i, repete in enumerate(nb_blocks):
188     for j in range(repete):
189         if i>0 and j == 0:
190             shortcut = Conv2D(n_filter * 4,
191                             kernel_size=(1, 1),
192                             strides=(2, 2),
193                             kernel_regularizer=l2(reg_f)
194                           )(shortcut)
195
196             X = Activation("relu")(X)
197             X = Dropout(drp_n)(X)
198
199             X = Conv2D(n_filter,
200                       kernel_size=(1,1),
201                       strides= (2,2),
202                       padding="same",
203                       kernel_regularizer=l2(reg_f)
204                     )(X)
205
206         else:
207             X = Activation("relu")(X)
208             X = Dropout(drp_n)(X)
209
210             X = Conv2D(n_filter,
211                       kernel_size=(1,1),
212                       padding="same",
213                       kernel_regularizer=l2(reg_f)
214                     )(X)
215
216         X = Activation("relu")(X)
217         X = Dropout(drp_n)(X)
218
219         X = Conv2D(n_filter,
220                   kernel_size=(3,3),
221                   padding="same",
222                   kernel_regularizer=l2(reg_f)
223                 )(X)
224
225
226         X = Activation("relu")(X)
227         X = Dropout(drp_n)(X)
228
229         X = Conv2D(n_filter * 4,

```

```

230             kernel_size=(1,1),
231             padding="same",
232             kernel_regularizer=l2(reg_f)
233           )(X)
234
235     # ショートカットとマージ
236     X = Add()([X, shortcut])
237     shortcut = X
238
239     n_filter *= wide
240
241     X = Activation("relu")(X)
242
243     # Branch Out
244     y1 = GlobalAveragePooling2D()(X)
245     y1 = Dropout(drp_n)(y1)
246     y1 = Dense(int(num_outputs/3),name='l1')(y1)
247
248     y2 = GlobalAveragePooling2D()(X)
249     y2 = Dropout(drp_n)(y2)
250     y2 = Dense(int(num_outputs/3),name='l2')(y2)
251
252     y3 = GlobalAveragePooling2D()(X)
253     y3 = Dropout(drp_n)(y3)
254     y3 = Dense(int(num_outputs/3),name='l3')(y3)
255
256     model = Model(inputs=[input], outputs=[y1,y2,y3])
257
258     return model
259
260 # -----
261 #
262 #           Machine Learning for 3-lines Waveform Digitizer
263 #           - Mulch GPU
264 #           - ResNet with dorpout and L2-regularizer
265 #           - lr_schedul
266 #           - branch out type
267 # -----
268
269 # --- Opening -----
270 # Date and time
271 today = datetime.datetime.now().strftime('%y%m%d-%H%M')
272 print("\n< RUN: Machine Learning for 3-lines Waveform Digitizer",
273       today,'>')
274
275 # Start time
276 s_time = time.time()
277
278 # GPU Number
279 gpu_n = 2
280
281 # Pass/NG Slice Level, difference between predict and target
282 sl = 0.03
283
284 # --- Data read -----
285 # data_n = input(' Data --> ')
286 data_n = './data/Digitizer_Data_3-lin_I5000+m5000_1024_200319-1030.pik';print('
Data:',data_n)
287
288 with open(data_n, mode='rb') as f:
289     test_graf = pickle.load(f)

```

```

290
291 dim = 1024 #test_graf['para'][0]
292 li_n = 3
293 print(' Dimension:',dim,', Line-number:',li_n)
294
295 if not li_n == 3:
296     print('This Learning Model is for 3-Line only.')
297     sys.exit()
298
299 print('\nLearning Process Start...')
300
301 # --- Data import and normalize -----
302 X = test_graf['data']
303 # Y-normalize
304 Y = np.reshape(test_graf['out'],(-1,li_n*dim))/dim
305
306 del test_graf
307
308 # X-normalize and black-white reversed
309 if np.max(X[0,:]) == 255:
310     nor = 255 # case of gray data
311 else:
312     nor = 1 # case of monochro data
313 X = 1 - X/nor
314
315 # input/output shape
316 n_in = len(X[0])
317 n_out = len(Y[0])
318
319 # Data split
320 x_train, x_test, y_train, y_test = train_test_split(
321     X, Y,
322     test_size = 0.1,
323     train_size = 0.9
324 )
325 # Delet large data
326 # del X ;
327
328 # --- Iteration parameter -----
329 reg_f = 0
330 n_drop = 0.1
331 n_epoch = 200
332 n_batch = 40
333 first_fil_n = 18
334 n_pat = 100
335 flag_ver = 0
336
337 # --- Model Built -----
338 input_ = x_train.shape[1:]
339 m_n = 'RN26'
340 model = resnet(input_,
341     n_out,
342     m_n,
343     first_fil_n,
344     reg_f,
345     n_drop,
346     li_n
347 )
348
349 # print prameter
350 print('\n Train_n = ',len(x_train),', Test_n =',len(x_test))

```

```

351 print(' epoch    = ',n_epoch)
352 print(' batch    = ',n_batch)
353 print(' first_filter = ',first_fil_n)
354 print(' patience   = ',n_pat)
355
356 # print("\n* model_summary"); model.summary()
357
358 # --- Model compile -----
359 # Optimizer setting
360 # Optimizer Adam
361 from keras.optimizers import Adam
362 optimizer = Adam()
363 print('\n Opimizer: Adam')
364
365 ""## Optimizer Adam
366 ## Optimizer Adam (lr:default 0.001)
367 from keras.optimizers import Adam
368 optimizer = Adam(lr=0.001)
369 print('\n Opimizer: Adam')
370
371 ## Optimizer SGD + Momentum(lr:default 0.01)
372 from keras.optimizers import SGD
373 optimizer = SGD(lr=0.01,decay=1e-6, momentum=0.9, nesterov=True)
374 print('\n Opimizer: SGD + Momentum')
375
376 # 学習率をepoch数により切り替える
377 from keras.callbacks import LearningRateScheduler
378 def lr_schedul(epoch):
379     ""default: adam=0.001, SDG=0.01
380     ""
381     x = 0.001
382     if epoch >= 100:
383         x = 0.00075
384     if epoch >= 200:
385         x = 0.0005
386     if epoch >= 400:
387         x = 0.0005
388     return x
389
390 lr_decay = LearningRateScheduler(lr_schedul)
391
392 # loss setting
393 loss = 'mean_squared_error'
394 print(' loss: ',loss)
395
396 # Compile
397 if not gpu_n == 1:
398     # Multi GPU
399     print('\n',str(gpu_n)+'-gpu running ')
400     from keras.utils import multi_gpu_model
401     parallel_model = multi_gpu_model(model, gpus=gpu_n)
402     parallel_model.compile(loss = loss,
403         optimizer = optimizer,
404         metrics = ['accuracy']
405     )
406 else:
407     # 1-GPU
408     print('\n 1-gpu running ')
409     model.compile(loss = 'mean_squared_error',
410         optimizer = optimizer,
411         metrics = ['accuracy']

```

```

412     )
413
414     early_stopping = EarlyStopping(monitor='val_loss',
415                                   patience=n_pat)
416
417     # --- Model fit -----
418     if not gpu_n == 1 :
419         # Multi GPU
420         hist = parallel_model.fit(x_train, [y_train[:,0:dim],
421                                           y_train[:,dim:dim*2],
422                                           y_train[:,dim*2:dim*3]],
423                                  epochs = n_epoch,
424                                  batch_size = n_batch,
425                                  validation_data = (x_test, [y_test[:,0:dim],
426                                                                y_test[:,dim:dim*2],
427                                                                y_test[:,dim*2:dim*3]]),
428                                  verbose = flag_ver,
429                                  callbacks = [early_stopping,lr_decay]
430                                )
431     else:
432         # 1-GPU
433         hist = model.fit(x_train, [y_train[:,0:dim],
434                                   y_train[:,dim:dim*2],
435                                   y_train[:,dim*2:dim*3]],
436                          epochs = n_epoch,
437                          batch_size = n_batch,
438                          validation_data = (x_test, [y_test[:,0:dim],
439                                                        y_test[:,dim:dim*2],
440                                                        y_test[:,dim*2:dim*3]]),
441                          verbose = flag_ver,
442                          callbacks = [early_stopping,lr_decay]
443                        )
444
445     # --- Predict and evaluation -----
446     # Predict
447     predict_ = (model.predict(x_test, batch_size = n_batch))
448     """これ以降の処理のためpredictを連結する
449     ...
450     predict = np.concatenate([predict_[0], predict_[1], predict_[2]],1)
451
452     # evaluation
453     dif = abs(y_test - predict)
454     """グラフの前後10%は空白の可能性がある
455     この領域では精度の評価をしない
456     ...
457     spc = int(dim*0.1)
458     dif_l = np.reshape(dif,(-1,li_n,dim))
459     dif_ll = np.reshape(dif_l[:,:,spc:dim-spc],(-1,(dim-2*spc)*li_n))
460
461     n_ep = len(hist.history['loss'])
462
463     # error count
464     pp_list = []
465     pass_list = []; ng_list=[]
466
467     for i in range(0,len(y_test)):
468         # error list
469         pp = np.max(dif_ll[i,:])
470         pp_list.append(pp)
471         if pp > sl:
472             ng_list.append(i) # NG list

```

```

473     else:
474         pass_list.append(i) # Pass list
475
476     # number of NG
477     nonzero = len(ng_list)
478     # pass rate
479     cor_pr = 1 - nonzero/(len(dif_ll))
480
481     # --- Execution time ---
482     e_time = time.time()
483     print('... Finished. ')
484     print('\n< Execution time > ')
485     print(' Exe time (s) = ', '{:6.1f}'.format(e_time - s_time),
486           '; end_ep = ',n_ep,
487           '; exe/ep = ',
488           '{:4.1f}'.format((e_time - s_time)/n_ep)
489         )
490
491     # --- Print out of results -----
492     # loss and accuracy
493     print('\n< loss and accuracy > ')
494
495     print( ' loss =', '{:4.2e}'.format(hist.history['loss'][n_ep-1]),
496           ', val_loss =', '{:4.2e}'.format(hist.history['val_loss'][n_ep-1]),
497           '\n l1-accuracy =', '{:4.3f}'.format(hist.history['l1_acc'][n_ep-1]),
498           ', l2-accuracy =', '{:4.3f}'.format(hist.history['l2_acc'][n_ep-1])
499         )
500
501     # loss history
502     fig = plt.figure(figsize=(7, 7))
503     plt.subplot(2, 2, 1)
504     x_ep = range(len(hist.history['loss']))
505     plt.plot(x_ep,hist.history['loss'],label="loss")
506     plt.plot(x_ep,hist.history['val_loss'],label="val_loss")
507     plt.yscale('log')
508     plt.legend(loc='best')
509     plt.title("loss and epoch",fontsize=15)
510
511     # accuracy history
512     plt.subplot(2, 2, 2)
513     x_ep = range(len(hist.history['l1_acc']))
514     # plt.plot(x_ep,hist.history['l1_acc'],label="l1_accu")
515     # plt.plot(x_ep,hist.history['val_l1_acc'],label="l1_val_acc")
516     plt.plot(x_ep,hist.history['l1_acc'],label="acc")
517     plt.plot(x_ep,hist.history['val_l1_acc'],label="val_acc")
518     #plt.yscale('log')
519     plt.title("accuracy and epoch",fontsize=15)
520     plt.ylim(0, 1.1)
521     plt.legend(loc='best')
522     plt.show()
523
524     # error
525     print('\n< Correct Rate >')
526     print(' Correct answer rate = ', '{:4.3f}'.format(cor_pr),
527           ', Error count = ',nonzero,'/',len(dif),
528           '(sl = ', '{:3.2f}'.format(sl),')'
529         )
530     print(' Note : 10% before and after data are not evaluated. Because there may be a
531           blank.')
532
533     # --- Result plotting -----

```

```

533 # Best & Worst
534 print('\n< Best and Worst Sample >')
535 fig = plt.figure(figsize=(7,7))
536 # Best sample index
537 pp_min = np.amin(pp_list)
538 kp = pp_list.index(pp_min) # Best sample index
539 print(' Best Sample No.= ', '{:3.0f}'.format(kp),
540       ', diff. = ', '{:4.2f}'.format(pp_min))
541
542 # Worst sample index
543 pp_max = np.amax(pp_list)
544 kg = pp_list.index(pp_max)
545 print(' Worst Sample No.= ', '{:3.0f}'.format(kg),
546       ', diff. = ', '{:4.2f}'.format(pp_max))
547 print(' upper: graph, lower: comparison of value')
548 cor_list = ['r','b','g','c','m','y','w']
549 f = np.linspace(0,1,dim)
550 # Best graph
551 plt.subplot(2,2,1)
552 plt.imshow(1-x_test[kp,:;0])
553 plt.title("Best Sample",fontsize=12)
554 plt.gray()
555 ##### comparison
556 plt.subplot(2,2,3)
557 for ii in range(0,li_n):
558     plt.plot(f,1-y_test[kp,dim*ii:dim*(ii+1)],
559             cor_list[ii],label="Target")
560 for ii in range(0,li_n):
561     plt.plot(f,1-predict[kp,dim*ii:dim*(ii+1)],
562             cor_list[ii]+'.',label="Predict")
563 plt.ylim(0,1);plt.xlim(0,1)
564
565 # Worst graph
566 plt.subplot(2,2,2)
567 plt.imshow(1-x_test[kg,:;0])
568 plt.title("Worst Sample",fontsize=12)
569 plt.gray()
570
571 # comparison
572 plt.subplot(2,2,4)
573 for ii in range(0,li_n):
574     plt.plot(f,1-y_test[kg,dim*ii:dim*(ii+1)],
575             cor_list[ii],label="Target")
576 for ii in range(0,li_n):
577     plt.plot(f,1-predict[kg,dim*ii:dim*(ii+1)],
578             cor_list[ii]+'.',label="Predict")
579 plt.ylim(0,1);plt.xlim(0,1)
580 plt.show()
581
582 # NG sample plot
583 print('\n< NG sample >')
584 fig = plt.figure(figsize=(16, 5))
585 n = 6
586 n_list = np.random.choice(ng_list, n)
587 print(' test number = ', n_list)
588
589 for i in range(n):
590     nn = n_list[i]
591     ## graph
592     ax = plt.subplot(2,n,i+1)
593     plt.imshow(1-x_test[nn,:;0])

```

```

594     plt.gray()
595     ax.get_xaxis().set_visible(False)
596     ax.get_yaxis().set_visible(False)
597     ## comparison
598     ax = plt.subplot(2,n,i+1)
599     for ii in range(0,li_n):
600         plt.plot(f,1-y_test[nn,dim*ii:dim*(ii+1)],
601                 cor_list[ii],label="Target")
602     for ii in range(0,li_n):
603         plt.plot(f,1-predict[nn,dim*ii:dim*(ii+1)],
604                 cor_list[ii]+'.',label="Predict")
605     plt.ylim(0,1);plt.xlim(0,1)
606     ax.get_xaxis().set_visible(False)
607     ax.get_yaxis().set_visible(False)
608 plt.show()
609
610 # OK sample plot
611 print('\n< OK sample >')
612
613 # --- Special exit ---
614 if len(pass_list) == 0:
615     print('\n *** The Proceess was terminated, because an OK sample does not exist. *** \n')
616     sys.exit()
617 # plot
618 fig = plt.figure(figsize=(16, 5))
619 n = 6
620 n_list = np.random.choice(pass_list, n)
621 print(' test number = ', n_list)
622 for i in range(n):
623     nn = n_list[i]
624     ## graph
625     ax = plt.subplot(2,n,i+1)
626     plt.imshow(1-x_test[nn,:;0])
627     plt.gray()
628     ax.get_xaxis().set_visible(False)
629     ax.get_yaxis().set_visible(False)
630     ## comparison
631     ax = plt.subplot(2,n,i+1)
632     for ii in range(0,li_n):
633         plt.plot(f,1-y_test[nn,dim*ii:dim*(ii+1)],
634                 cor_list[ii],label="Target")
635     for ii in range(0,li_n):
636         plt.plot(f,1-predict[nn,dim*ii:dim*(ii+1)],
637                 cor_list[ii]+'.',label="Predict")
638     plt.ylim(0,1);plt.xlim(0,1)
639     ax.get_xaxis().set_visible(False)
640     ax.get_yaxis().set_visible(False)
641 plt.show()
642
643 # --- Model print and save -----
644 print('\n< Model save and print >')
645 m_para = m_n+'_bt'+str(n_batch)+'ft'+str(first_fil_n)+'ep'+str(n_ep)
646 model_n = 'Digitizer_'+str(li_n)+'-lin_'+str(dim)+'_'+m_para+'_'+today
647 mp_flag = input(' Print the model shape? y or n --> ')
648 # model print
649 if mp_flag == 'y':
650     from keras.utils import plot_model
651     plot_model(model, to_file = model_n+'.png', show_shapes=True)
652     print(' > Model was printed: '+model_n+'.png ')
653 else:
654     print(' > Model was NOT printed.')

```

```
655
656 # model save
657 ms_flag = input('\n Save the learned model? y or n --> ')
658 if ms_flag == 'y' :
659     model.save('./model/'+model_n+'.h5')
660     print(' > Model was saved: '+model_n+'.h5')
661 else :
662     print(' > Model was NOT saved.')
663
664 # Del large data ---
665 del x_train ; del x_test
666
667 # _____ END of code _____
```

**END**