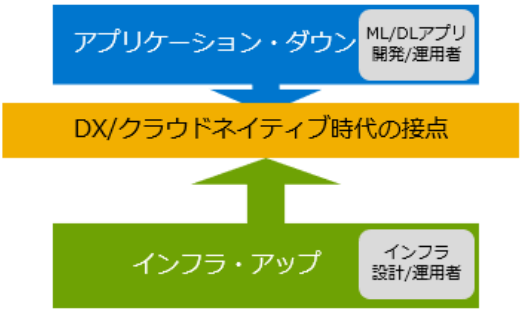
ML/DL ハンズオン (Piper wave X)

・ゴール：APPエンジニアとインフラ接点の会話ができるようになる（DX readyなインフラSE）

・コンセプト：全体把握と時間節約のため、複雑さを極力削ぎ落としたシンプルコード

・キーワード：モデルと学習と推論、GPU効果, Python, Jupyter Notebook, TensorFlow, Keras

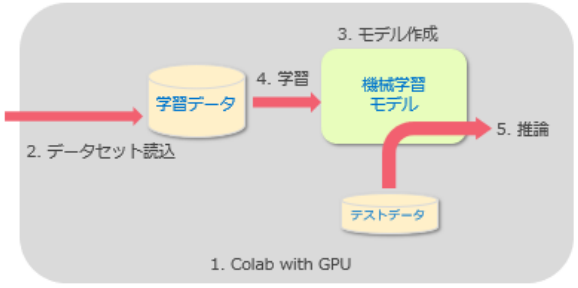


接点はアプリケーション寄りにシフト！

お題：手書き文字（0～9）を認識する学習モデルを作成する

進め方(ステップ)

1. 実行環境確認　　：Google Colaboratory（必要に応じて補足）
2. データセット読込と前処理 : MNIST（必要に応じて補足）
3. モデル作成　　：ニューラルネットワーク定義
4. 学習　　：GPUによる速度改善
5. 推論　　：正解率確認



質問：現状の上図理解度は5段階で？（1:low 5:high）

---

1. 実行環境：Google Colaboratory（必要に応じて補足）

1)Pythonバージョンの確認　　：2つのセル、実行

2)Hello GURU のコンソール出力　：Pythonコード変更、Notebookのsave（ファイル > ドライブにコピーを保存）

3)Hardware / GPU確認　　：Tesla K80等

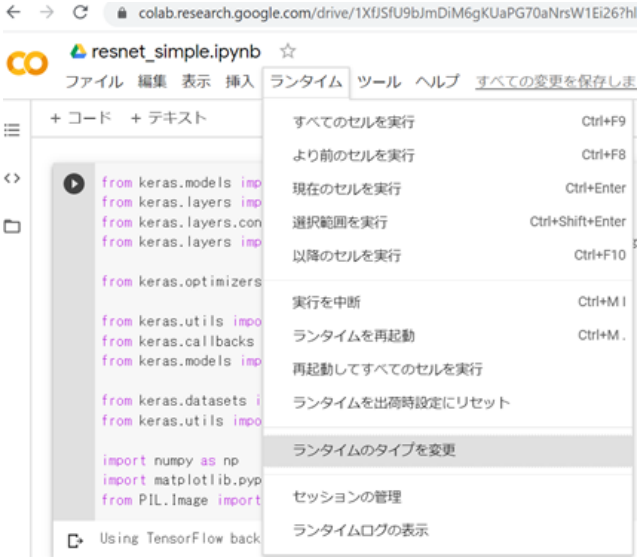
```
!python -V              #3.6.9
```

```
Python 3.7.13
```

```
print ("Hello GURU!")   # Hello GURU
```

```
Hello GURU!
```

GPU使用宣言 ランタイム　-> ランタイムタイプの変更 -> "GPU"を選択




```
### 構成の確認
!uname -a
#!apt-get install lshw  #初回はlshwのInstallの為にコメントアウトを外しましょう
!lshw
```

```
Linux 3a656731b88c 5.4.188+ #1 SMP Sun Apr 24 10:03:06 PDT 2022 x86_64 x86_64 x86_64 GNU/Linux
/bin/bash: lshw: command not found
```

```
# GPUの確認
!nvidia-smi
```

```
Tue May 31 08:05:14 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   51C    P0    28W /  70W |    270MiB / 15109MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
+-----------------------------------------------------------------------------+
```

2. データセット読込と前処理：MNIST（必要に応じて補足）

データセット（教師あり）：統一フォーマットでアノテーション済のデータ集合体
MNISTとは？：「手書き数字のサンプルデータ」MLのHello world的データセット（他にCIFAR10など）

- 7万枚 28*28 8bit (学習用6万,テスト用1万) http://yann.lecun.com/exdb/mnist/

ここまではイントロ、これ以降が学習モデル作成のためのPythonコード

```
##### ライブラリのインポート
import tensorflow as tf
import keras
from keras.datasets import mnist
```

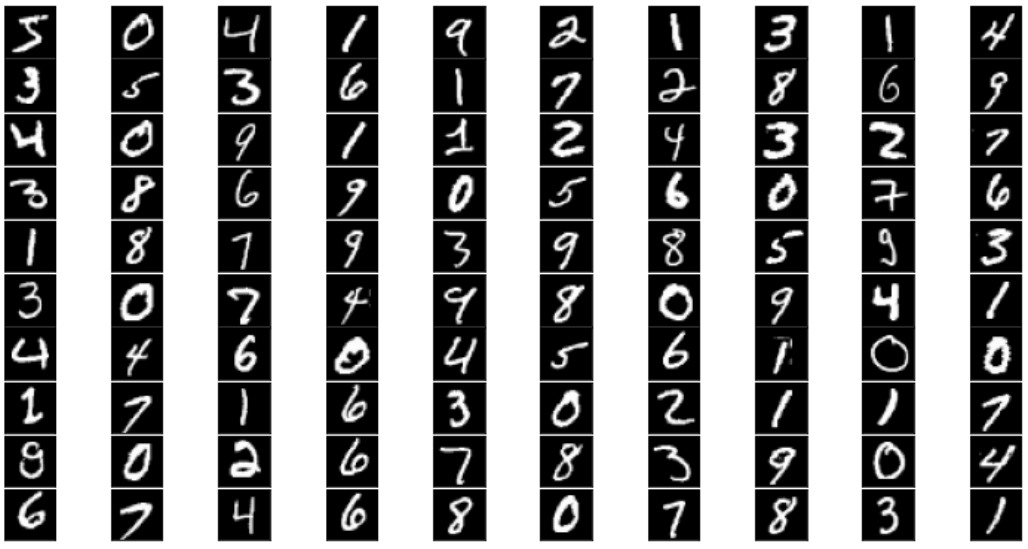
```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K
from __future__ import print_function
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from keras.utils import np_utils    #add 2021/06 miho

%matplotlib inline
print("tensorflow " + tf.__version__)
print("keras " + keras.__version__)
```

```
tensorflow 2.8.0
keras 2.8.0
```

```
##### MNISTデータ読込

# 定数セット
batch_size = 128               #バッチサイズ 128
num_classes = 10               #分類クラス 10
epochs = 12                    #エポック 12
img_rows, img_cols = 28, 28    # input image dimensions

# 学習用、テスト用データのLOAD
(x_train, y_train), (x_test, y_test) = mnist.load_data()

print ("Done load MNIST")
```

```
Done load MNIST
```

```
### debug MNIST画像の100枚表示
fig = plt.figure(figsize=(10,10))
fig.subplots_adjust(left=0, right=1, bottom=0, top=0.5, hspace=0.05, wspace=0.05)
for i in range(100):
    ax = fig.add_subplot(10, 10, i + 1, xticks=[], yticks=[])
    ax.imshow(x_train[i].reshape((28, 28)), cmap='gray')
```



```
### debug  MNIST画像とラベルの10枚表示
fig = plt.figure(figsize=(9, 15))
fig.subplots_adjust(left=0, right=1, bottom=0, top=0.5, hspace=0.05, wspace=0.05)
# 10枚数の画像に対応するラベルを表示
for i in range(10):
    ax = fig.add_subplot(1, 10, i + 1, xticks=[], yticks=[])
    ax.set_title(str(y_train[i]))
    ax.imshow(x_train[i], cmap='gray')
```


```
### debug
p_index = 0                              ### 0(index)番目データを指定
print("Label:", y_train[p_index])         # ラベル表示

np.set_printoptions(edgeitems=28, linewidth=100000) #表示の仕方の設定
np.core.arrayprint._line_width = 200 #表示の仕方の設定
print(x_train[p_index])                   # p_index番目のデータをraw表示(数値)
```

```
Label: 5
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   3  18  18  18 126 136 175  26 166 255 247 127   0   0   0   0]
 [  0   0   0   0   0   0   0   0  30  36  94 154 170 253 253 253 253 253 225 172 253 242 195  64   0   0   0   0]
 [  0   0   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251  93  82  82  56  39   0   0   0   0   0]
 [  0   0   0   0   0   0   0  18 219 253 253 253 253 253 198 182 247 241   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  14   1 154 253  90   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  11 190 253  70   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  35 241 225 160 108   1   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  81 240 253 253 119  25   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  45 186 253 253 150  27   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252 253 187   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 249 253 249  64   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253 253 207   2   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  39 148 229 253 253 253 250 182   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253 253 201  78   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  23  66 213 253 253 253 253 198  81   2   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0  18 171 219 253 253 253 253 195  80   9   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0  55 172 226 253 253 253 253 244 133  11   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0 136 253 253 253 212 135 132  16   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]]
```

```
##### データ前処理（各ビットの0～1化とラベルのOne-hot化=一つのベクトルを示す配列へ変換）
# ※2回実行しないように。2回実行すると多重配列になってしまいます
x_test_bak = x_test
if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255                           #255で除算
x_test /= 255                            #255で除算
#print('x_train shape:', x_train.shape)    # (60000, 28, 28, 1)

# convert class vectors to binary class matrices(One-hot)
y_test_bak = y_test
y_train_bak = y_train

### changed from keras.utils to np_utils 2021/06 miho ###
y_train = np_utils.to_categorical(y_train, num_classes)
y_test = np_utils.to_categorical(y_test, num_classes)
# y_train = keras.utils.to_categorical(y_train, num_classes)
# y_test = keras.utils.to_categorical(y_test, num_classes)

print ("Done pre-process")
```

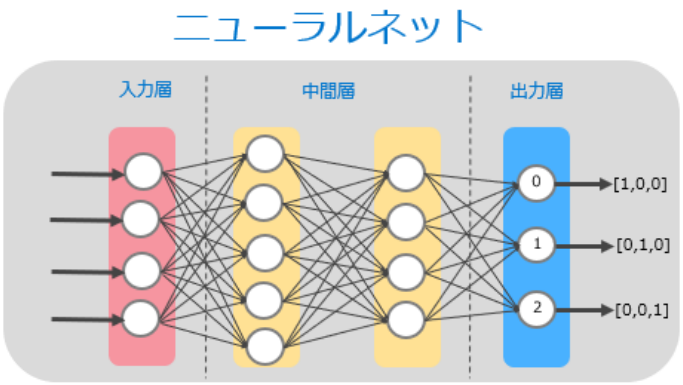
```
Done pre-process
```

```
# debug 表示
print("Label:",y_train_bak[p_index])
```

```
print(y_train[p_index])                 # One-hot エンコード後のラベル表示

#print(p_index)
#print( 'train samples:', x_train.shape[p_index])  # 学習サンプル数 60000
```

```
Label: 5
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

3. モデル作成：ニューラルネットワーク



3つの学習モデルを用意：いづれかを選択する（コメントアウト#を外す）

- MODEL-A
- MODEL-B
- MODEL-C

```
##### 学習モデルの作成
print(input_shape)
model = Sequential()                                    # Sequentialモデルを定義

### MODEL-A :CNN
model.add(Conv2D(32, kernel_size=(3, 3),                #32出力 3*3フィルターで畳み込み
                 activation='relu',
                 input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))               #2*2でMaxプーリング
model.add(Conv2D(64, (3, 3), activation='relu'))        #64出力 3*3フィルターで畳み込み
model.add(MaxPooling2D(pool_size=(2, 2)))               #2*2でMaxプーリング
model.add(Dropout(0.25))                                #過学習防止のDropout; 0.25の割合で入力を0
model.add(Flatten())                                    #1次元配列に整形
model.add(Dense(128, activation='relu'))                #128nodeの中間層
model.add(Dense(num_classes, activation='softmax'))     #出力層;10 classに分類

### MODEL-B :NN
# model.add(Flatten(input_shape=input_shape))
# model.add(Dropout(0.25))
# model.add(Dense(128, activation='relu'))                #128 or 1000にしてみる
# model.add(Dropout(0.5))
# model.add(Dense(num_classes, activation='softmax'))

### MODEL-C : NN
# model.add(Dense(32, activation='relu', input_shape=input_shape))
# model.add(Flatten())                                    #1次元配列に整形
# model.add(Dense(num_classes, activation='softmax'))

### モデルのコンパイル
## changed 2021/09 miho ##
model.compile(loss=keras.losses.categorical_crossentropy, optimizer='adam', metrics=['accuracy'])
# model.compile(loss=keras.losses.categorical_crossentropy,optimizer=keras.optimizers.Adadelta(),metrics=['accuracy'])

### Optimizer 選択肢
# keras.optimizers.SGD()
# keras.optimizers.RMSprop()
# keras.optimizers.Adagrad()
# keras.optimizers.Adam()
```

```
# keras.optimizers.Nadam()
# keras.optimizers.Adadelta()

print("Done modeling")
```

```
(28, 28, 1)
Done modeling
```

```
### debug 作成したモデルの表示
print("layers:", len(model.layers))  # num of layers
model.summary()
```

```
layers: 8
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 26, 26, 32)        320

 max_pooling2d_4 (MaxPooling  (None, 13, 13, 32)       0
 2D)

 conv2d_5 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_5 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 dropout_2 (Dropout)         (None, 5, 5, 64)          0

 flatten_2 (Flatten)         (None, 1600)              0

 dense_4 (Dense)             (None, 128)               204928

 dense_5 (Dense)             (None, 10)                1290

=================================================================
Total params: 225,034
Trainable params: 225,034
Non-trainable params: 0
_________________________________________________________________
```

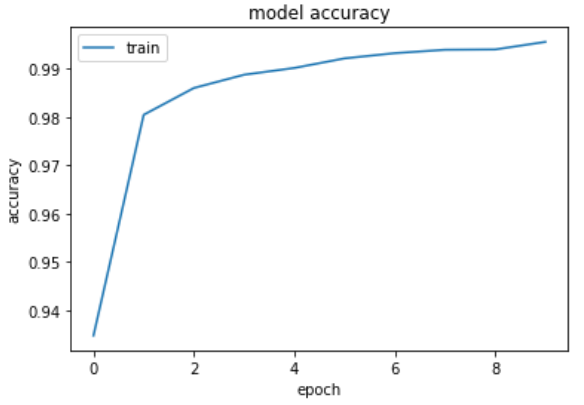
4. 学習: GPUによる演算速度向上

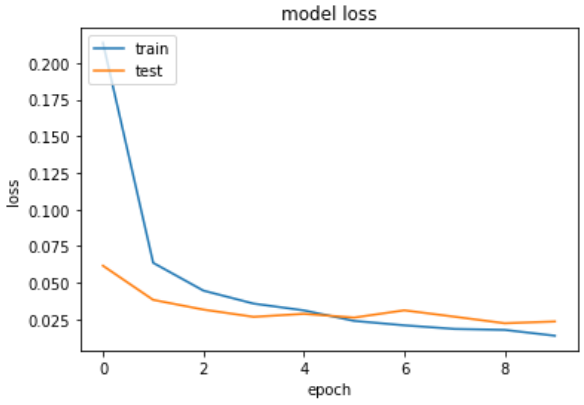
```
##### 学習
history = model.fit(x_train, y_train,
          batch_size=128,
          epochs=10,
          verbose=1,
          validation_data=(x_test, y_test))     # テストデータを指定（画像とラベル）
```

```
Epoch 1/10
469/469 [==============================] - 14s 6ms/step - loss: 0.2139 - accuracy: 0.9347 - val_loss: 0.0615 - val_accuracy: 0.98
Epoch 2/10
469/469 [==============================] - 3s 5ms/step - loss: 0.0635 - accuracy: 0.9804 - val_loss: 0.0382 - val_accuracy: 0.986
Epoch 3/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0445 - accuracy: 0.9860 - val_loss: 0.0316 - val_accuracy: 0.988
Epoch 4/10
469/469 [==============================] - 3s 5ms/step - loss: 0.0357 - accuracy: 0.9887 - val_loss: 0.0266 - val_accuracy: 0.990
Epoch 5/10
469/469 [==============================] - 2s 5ms/step - loss: 0.0310 - accuracy: 0.9901 - val_loss: 0.0287 - val_accuracy: 0.990
Epoch 6/10
469/469 [==============================] - 2s 5ms/step - loss: 0.0239 - accuracy: 0.9921 - val_loss: 0.0261 - val_accuracy: 0.991
Epoch 7/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0208 - accuracy: 0.9932 - val_loss: 0.0311 - val_accuracy: 0.990
Epoch 8/10
469/469 [==============================] - 3s 5ms/step - loss: 0.0184 - accuracy: 0.9939 - val_loss: 0.0267 - val_accuracy: 0.990
Epoch 9/10
469/469 [==============================] - 3s 5ms/step - loss: 0.0177 - accuracy: 0.9940 - val_loss: 0.0222 - val_accuracy: 0.993
Epoch 10/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0137 - accuracy: 0.9955 - val_loss: 0.0234 - val_accuracy: 0.992
```

```
### debug 学習結果の表示
## Accuracy
plt.plot(history.history['accuracy'])
```

```
# plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
# plt.legend(['train', 'test'], loc='upper left')
plt.legend(['train'], loc='upper left')
plt.show()

# loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```





課題1：GPUなしで学習してみる

TensorFlowライブラリがGPU有無を自動判断　　（ラボはモデル作成からやり直しが必要）

課題2：epocの最適値を考える

データセットの繰り返し学習の回数

---

5. 推論: 正解率確認

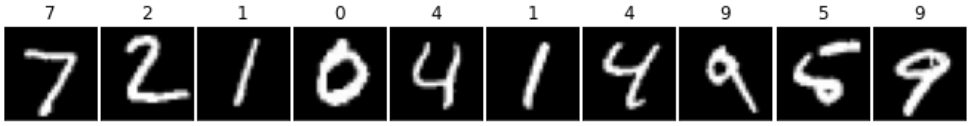
```
##### 推論: x_testを使い推論する
## changed 2021/09 miho ##
predictions = np.argmax(model.predict(x_test, batch_size=batch_size, verbose=1),axis=-1)
# predictions = model.predict_classes(x_test, batch_size=batch_size, verbose=1)    #
```

```
79/79 [==============================] - 0s 2ms/step
```

```
### debug 推論結果の表示-1
#print(type(predictions))
x = list(predictions)
y = list(y_test_bak)
# results = pd.DataFrame({'Actual': y, 'Predictions': x})
```

```
# output = results[0:10]
# print(output)

### 推論結果の表示-2
fig = plt.figure(figsize=(9, 15))
fig.subplots_adjust(left=0, right=1, bottom=0, top=0.5, hspace=0.05, wspace=0.05)
# MNIST画像の上に推論結果を表示
for i in range(10):
    ax = fig.add_subplot(1, 10, i + 1, xticks=[], yticks=[])
    ax.set_title(str(x[i]))
    ax.imshow(x_test_bak[i], cmap='gray')

##### 評価(テストデータでどの程度正解するか？)
score = model.evaluate(x_test, y_test, verbose=0)
#print('Test loss:', score[0])
print('Test data accuracy:', score[1])
#print(score)

### END ###
```

```
Test data accuracy: 0.9922999739646912
```



おまけ：学習モデルの保存

学習済のモデルはファイルとして保存可能（HDF5ファイル形式）

- 再構築可能なモデルの構造
- モデルの重み
- 学習時の設定 (loss，optimizer)
- optimizerの状態．これにより，学習を終えた時点から正確に学習を再開できます

```
### 学習モデルの保存：(再利用、移転学習用)
from google.colab import files
model.save('my_piper_model.h5')             # creates a HDF5 file 'my_model.h5'
!ls -l
files.download('my_piper_model.h5')         # ファイルを自分のパソコンに保存

# del model                                 # deletes the existing model
# model = load_model('my_model.h5')         # returns a compiled model
```

```
total 2688
-rw-r--r-- 1 root root 2748176 May 31 08:07 my_piper_model.h5
drwxr-xr-x 1 root root    4096 May 17 13:39 sample_data
```

＋ コード ＋ テキスト

終わりに

質問：理解度は向上したか？　5段階で（1:low 5:high）

✓ 0 秒　完了時間: 17:07