

# A I 基礎セミナー

## 第 7 回 機械学習（1 回目：画像認識モデルの作成例）

### 改訂履歴

日付	担当者	内容
2021/05/08	M. Takeda	Git 公開

### 目次

- (1) はじめに
- (2) 入力データ
  - (2.1) モデル作成で使用する入力データ
  - (2.2) 自作の入力データ
- (3) モデル作成例
  - (3.0) 各モデルでの共通処理
  - (3.1) 「手書き文字認識モデル 1」
  - (3.2) 「手書き文字認識モデル 2」
  - (3.2) 「手書き文字認識モデル 2」
  - (3.3) 「手書き文字認識モデル 3」
  - (3.4) 「手書き文字認識モデル 4」
  - (3.5) 「手書き文字認識モデル 5」
  - (3.6) 「手書き文字認識モデル 6」
  - (3.7) 「手書き文字認識モデル 7」
- (4) まとめ

## (1) はじめに

- ・今回から、いよいよ機械学習に入ります。
- ・1回目は、手書き文字のデータセットとして有名な「MNIST（エムニスト）」を読み込んで、手書き数字の認識モデルを作成する事例を紹介します。  
認識モデルを数種類（7種類ほど）作成して実装し、その学習と評価を「MNIST（エムニスト）」を用いて行い、更に各モデルで自作の手書き数字を認識させることにします。
- ・手書き数字の認識は、機械学習の分類上では「教師あり学習（キョウシアリガクシュ, Supervised learning）」で、「分類（ブンレイ、Classification）」になります。
- ・以下の3種類のニューラルネットワークでパラメータを少しずつ変えて認識モデルを作成して、認識結果を比較検証します。  
各モデルの概要説明は「第2回 機械学習のモデルとアルゴリズム」の「(4) 様々なニューラルネットワーク」にもありますので参照ください。
  - (1) 「順伝播型ニューラルネットワーク (Feed forward neural networks, FFNN)」
  - (2) 「畳み込みニューラルネットワーク (Convolutional neural network, CNN)」
  - (3) 「深層畳み込みニューラルネットワーク (Deep Convolutional neural network, DCN)」

- ・以下の7種類の認識モデルを用意します。

項番とモデル名	層数	モデルタイプ	パラメータ指定など
(3.1) 「手書き文字認識モデル1」	2	FFNN	中間層の活性化関数は sigmoid / relu の2種類
(3.2) 「手書き文字認識モデル2」	2	CNN	プーリング層なし、パディング有り
(3.3) 「手書き文字認識モデル3」	7	DCN	2×2 の最大値プーリング、ドロップアウト率 0.25
(3.4) 「手書き文字認識モデル4」	7	DCN	2×2 の平均値プーリング、ドロップアウト率 0.25
(3.5) 「手書き文字認識モデル5」	9	DCN	2×2 の最大値プーリング、ドロップアウト率 0.25
(3.6) 「手書き文字認識モデル6」	7	DCN	3×3 の最大値プーリング、ドロップアウト率 0.25
(3.7) 「手書き文字認識モデル7」	7	DCN	3×3 の最大値プーリング、ドロップアウト率 0.50

- ・モデルの比較のために、以下は同じとしました：

- (1) 中間層の活性化関数として「relu（正規化線形関数, Rectified Linear Unit）」を用います。（※1）
- (2) 出力層の活性化関数として「ソフトマックス関数（softmax）」を用います。
- (3) 損失関数（誤差を計算する関数）として、「クロスエントピー誤差（cross entropy）」を用います。
- (4) 学習方法（モデルの最適化方法）として、勾配降下法の一種である「Adam」を用います。
- (5) 学習時には「エポック数」として10、「バッチサイズ」として「1000」で学習を行います。

※1 「手書き文字認識モデル1」だけ「sigmoid（シグモイド関数）」でも検証してみます。

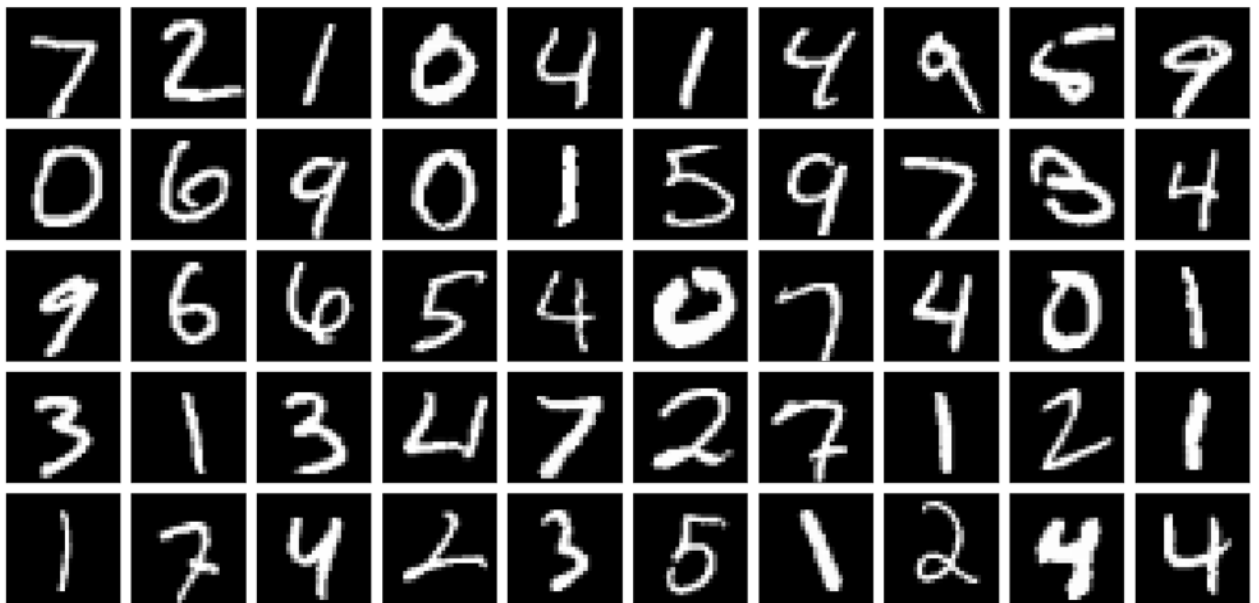
- ・今回は、モデルの作成・実装とその適用という点を中心にニューラルネットワークに馴染むことに目標を置き、理論的な説明と、確認問題は次回「第8回 機械学習（2回目：機械学習の概要と教師あり学習（分類）」にします。

- ・まずは、ニューラルネットワークとディープラーニングの一端を垣間見てみましょう。

## (2) 入力データ

### (2.1) モデル作成で使用する入力データ

- ・今回は、手書き文字のデータセットとして有名な「MNIST (エムニスト, Mixed National Institute of Standards and Technology database)」というデータセットを読み込んで、手書き数字の認識モデルを作成します。
- ・MNIST データセットは、手書きで書かれた数字を、 $28 \times 28$ ピクセルの画像にした画像データと、その画像に書かれた数字 (0~9) を表すラベルデータから構成されています。そのセットが学習用 (60000枚) と検証用 (10000枚) 毎に提供されています。
  - (a) x\_train: 学習用の画像データ (データ型はuint8、60000枚、サイズは  $28 \times 28$ ピクセル)
  - (b) y\_train: 学習用の画像のラベルデータで画像が表す数値 (0~9) (データ型はuint8、60000枚分)
  - (c) x\_test : 検証用の画像データ: (データ型はuint8、10000枚、サイズは  $28 \times 28$ ピクセル)
  - (d) y\_test : 検証用の画像のラベルデータで画像が表す数値 (0~9) (データ型はuint8、10000枚分)
- ・以下に、MNIST データセット検証用画像の先頭の 50枚を掲載します。



#### 【出典・参考】

MNIST⇒ <https://weblabo.oscasierra.net/python/ai-mnist-data-detail.html>

## (2.2) 自作の入力データ

- ・ MNISTを用いて作成した手書き数字認識モデルを用いて、自作の手書き数字を認識させてみることにします。
- ・ 自作の手書き文字の各ファイルは MSペイントで、サイズが 28×28 ピクセルで「256色ビットマップ」形式で作成し、フォルダ「HandwrittenImages」以下に配置しておきます。



- ・ フォルダ「HandwrittenImages」の準備は、「Anaconda Navigator」と「Google Colaboratory」の環境下で異なりますので、以下に解説します。

## (2.2.1) 「Anaconda Navigator」の環境下での配置先

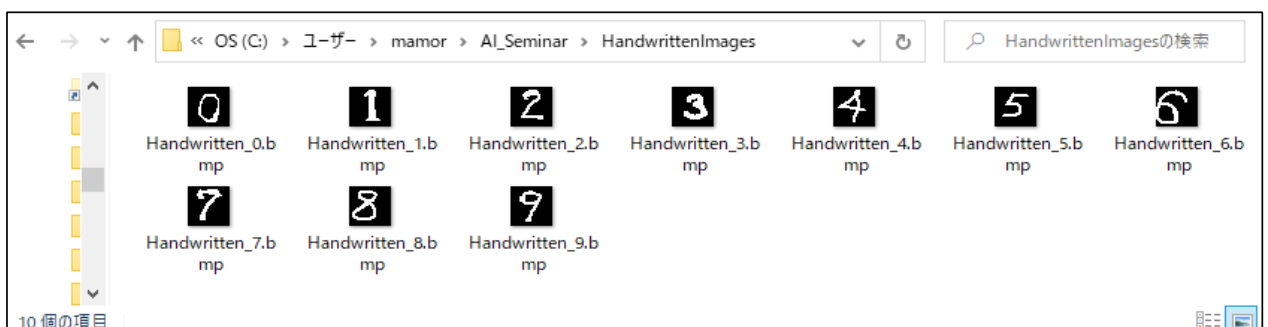
- ・「Anaconda Navigator」の環境下で、Pythonの実行ディレクトリの直下に作成することにします。  
Pythonの実行ディレクトリは「リスト07-(02)-1\_Pythonの実行ディレクトリを取得」を実行して得られます。  
このコードを実行して得られたフォルダの直下に、フォルダ「HandwrittenImages」を作成します。

(リスト07-(02)-1\_Pythonの実行ディレクトリを取得)

```
#####/  
# リスト07-(02)-1_Pythonの実行ディレクトリを取得  
#####/  
import os  
print("\nCurrent working Directory (getcwd) : '%0%'".format(os.getcwd()))
```

(リスト07-(02)-1\_Pythonの実行ディレクトリを取得)の実行結果例

```
Current working Directory (getcwd) : 'C:\Users\mamor\AI_Seminar'
```



- ・「Anaconda Navigator」の場合、上記の配置方法のとおり  
「HandwrittenImages」フォルダの実行ディレクトリからの相対パスを「HandwrittenImages」  
として実装する為に、以下のコードを実行します。

(リスト07-(02)-2\_「HandwrittenImages」フォルダの相対パス(Anaconda Navigator))

```
#####/  
# リスト07-(02)-2_「HandwrittenImages」フォルダの相対パス(Anaconda Navigator)  
#####/  
imageFolderPath = "HandwrittenImages"
```

## (2.2.2) 「Google Colaboratory」の環境下での配置先

- ・「Google Colaboratory」の場合、  
一旦「Google Drive」上にフォルダ「HandwrittenImages」を作成し、そこに各画像ファイルをアップロードします。  
そのうえで「Google Colaboratory」で実行する Python コードから  
「Google Drive」のフォルダが参照できるようにマウントを行い、そのフォルダパスで参照します。

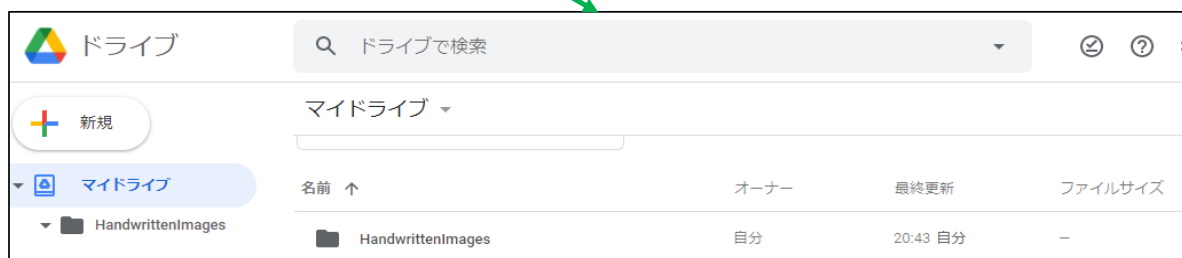
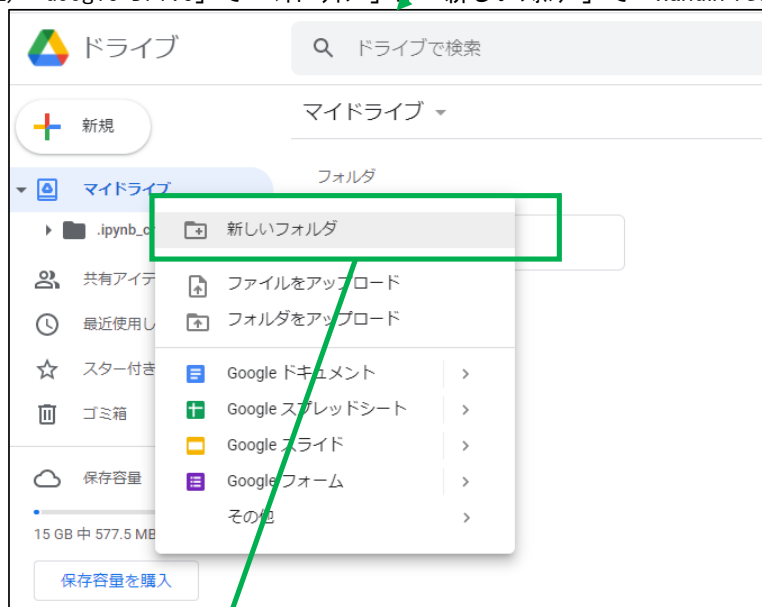
※「Google Colaboratory」や「Google Directory」を利用する場合、  
予め google アカウントを用意しておく必要があります(本セミナー「第1回」を参照)。

(1) 「Google Drive」上にフォルダ「HandwrittenImages」を作成し各画像ファイルをアップロード

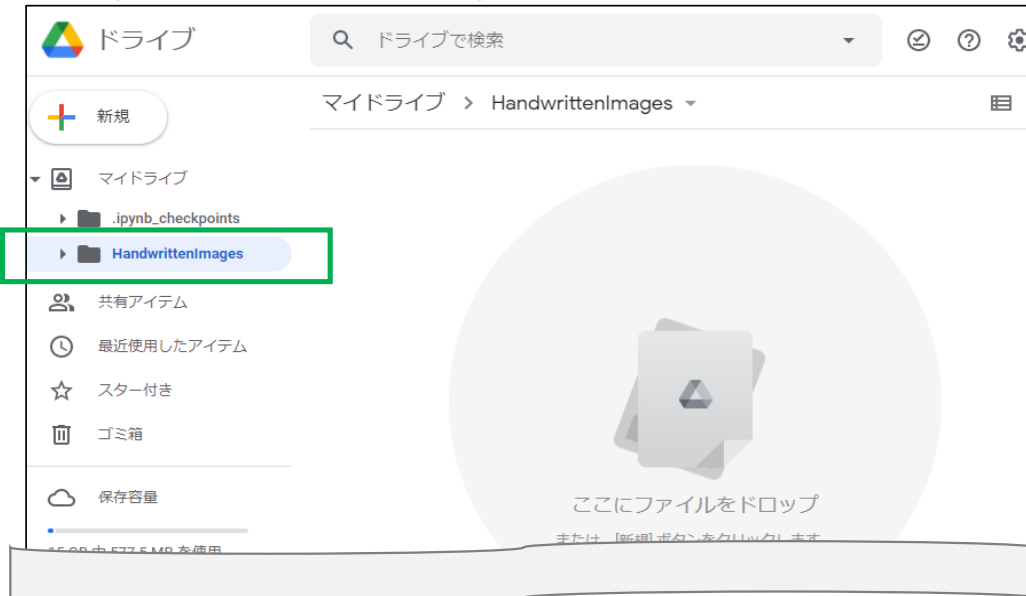
(1.1) Google のサイト から「Google Drive」を開きます



(1.2) 「Google Drive」で「マイドライブ」に「新しいフォルダ」で「HandwrittenImages」を作成します。



(1.3) 「Google Drive」で「HandwrittenImages」に移動します。



ローカルフォルダでファイルを指定して(複数指定可能)ドラッグ & ドロップでファイルをアップロードします。



(2) 「Google Colaboratory」から「Google Drive」のフォルダが参照できるようにマウントを行います。

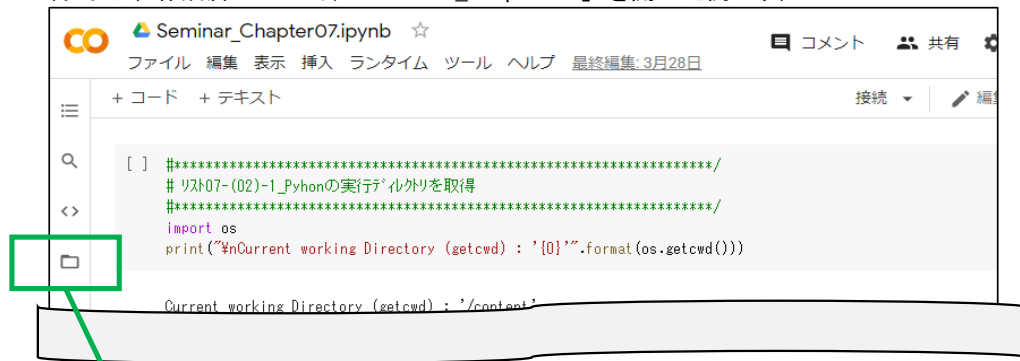
(2.1) 「Google Colaboratory」にログインします。

URL 「<https://colab.research.google.com>」にアクセスして、ログインします。

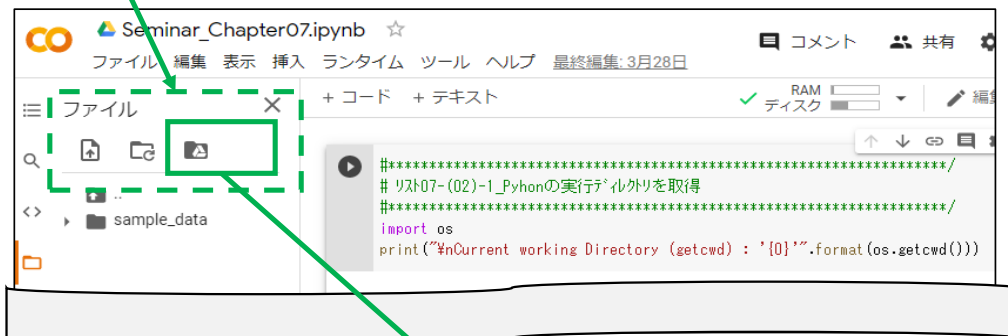
(2.2) ログインすると、以下のような画面が表示されるので、「ノートブックを新規作成」を選択します。  
あるいは、同画面で作成済のノートブックを開きます。



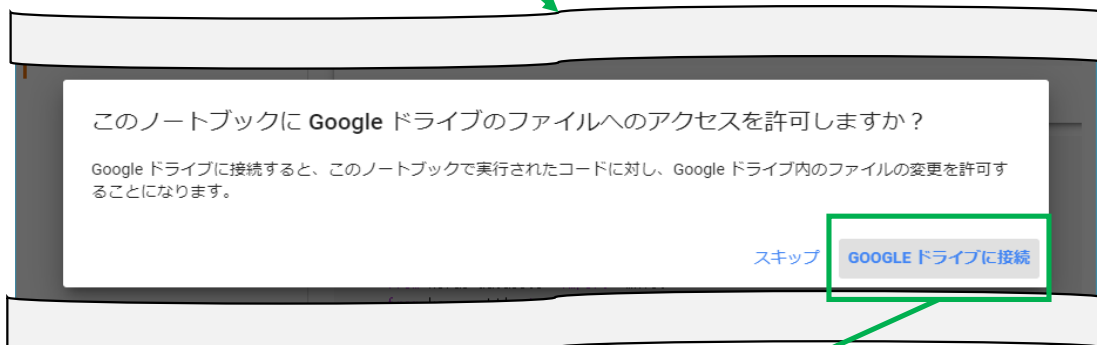
- (2.3) すると「Colab ノートブック」という、Pythonコードを記述して実行できる環境が開きます。  
(以下は、作成済のノートブック「Seminar\_Chapter07」を開いた例です)



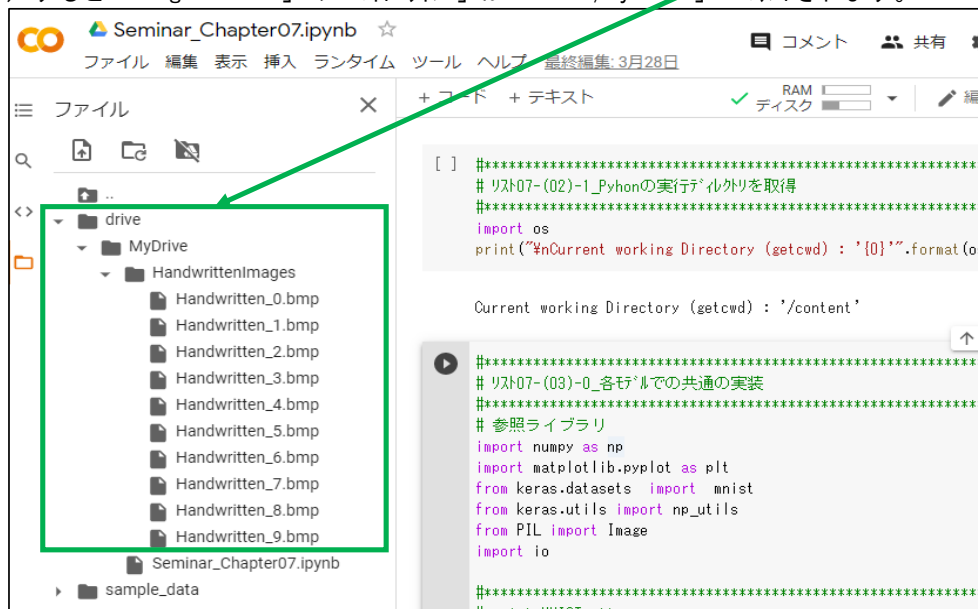
- (2.4) 「Colab ノートブック」の左欄のフォルダマークをクリックすると、ドライブのマウントアイコンが表示されるのでマウントアイコンをクリックします。



- (2.5) 許可が求められるので「Google Drive」のファイルへのアクセスを許可します。



- (2.6) すると「Google Drive」の「マイドライブ」が「drive/MyDrive」へマウントされます。





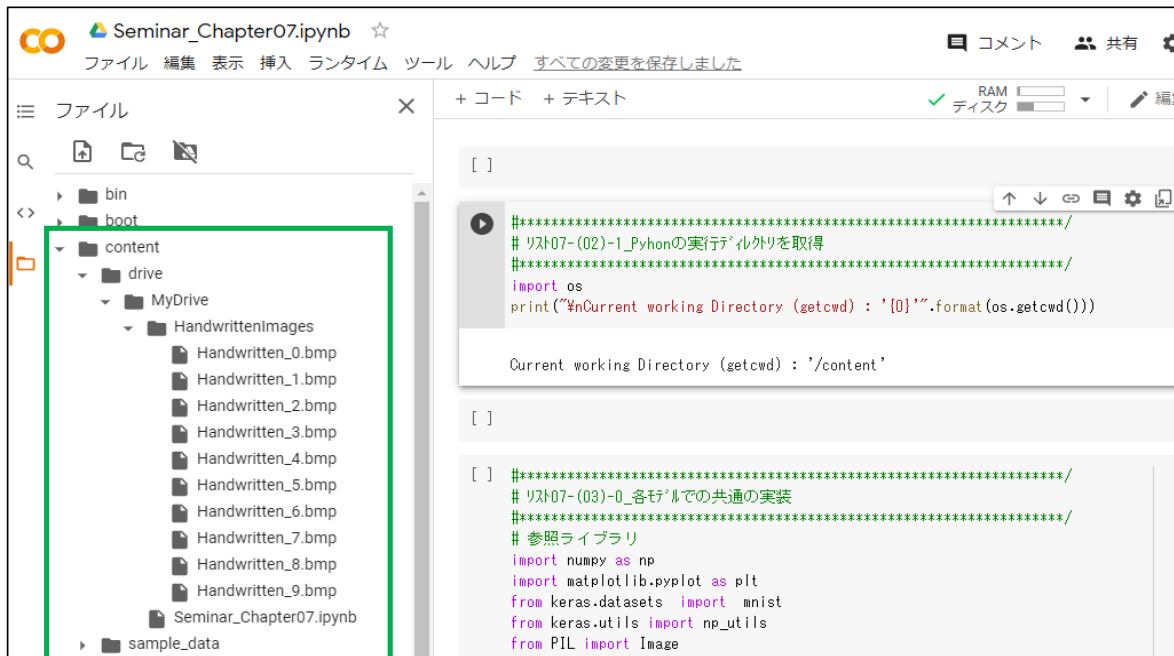
(3) 「Google Colaboratory」での Pythonの実行ディレクトリから見た「HandwrittenImages」フォルダの位置

(3.1) Pythonの実行ディレクトリは「リスト07-(02)-1\_Pythonの実行ディレクトリを取得」を実行して得られます。

(リスト07-(02)-1\_Pythonの実行ディレクトリを取得)の実行結果例

```
Current working Directory (getcwd) : '/content'
```

(3.2) 「Colab ノートブック」の左欄のフォルダ構成で「drive」より上位のフォルダを参照すると、Pythonの実行ディレクトリ「/content」の下に「drive」フォルダがあることがわかります。従って、実行ディレクトリからの相対パスは「drive/MyDrive/HandwrittenImages」となります。



(3.3) 「HandwrittenImages」フォルダの実行ディレクトリからの相対パスの指定

上記を元にして「Google Colaboratory」の場合、

「HandwrittenImages」フォルダの実行ディレクトリからの相対パスを「drive/MyDrive/HandwrittenImages」として実装する為に、以下のコードを実行します。

(リスト07-(02)-3\_「HandwrittenImages」フォルダの相対パス(Google Colaboratory))

```
#####/
# リスト07-(02)-3_「HandwrittenImages」フォルダの相対パス(Google Colaboratory)
#####/
imageFolderPath = "drive/MyDrive/HandwrittenImages"
```

【出典・参考】

ColaboratoryでのGoogle Driveへのマウントが簡単になっていたお話

⇒ [https://qiita.com/kado\\_u/items/45b76f9a6f920bf0f786](https://qiita.com/kado_u/items/45b76f9a6f920bf0f786)

### (3) モデル作成例

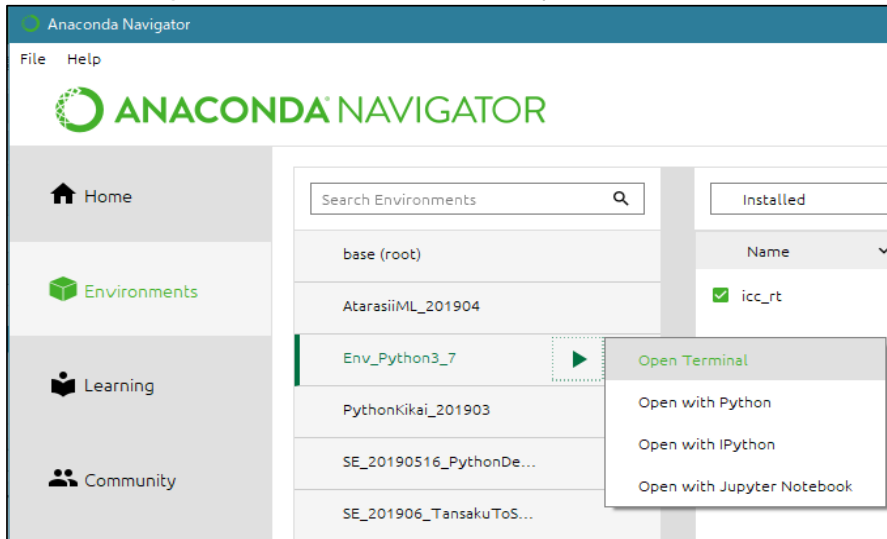
#### (3.0) 各モデルでの共通処理

- ・はじめに、必要なライブラリのインストールと、  
後続のモデルの結果表示や実行で用いる共通の実装を、記しておきます。
- ・主なライブラリとして本セミナー「第1回」で紹介した「Keras (ケラス)」を使用します。  
Keras は「TensorFlow (テンソルフロー)」の使い勝手を良くしたラッパーライブラリでもあり、  
TensorFlow を統合したAPI仕様 となっています。  
インストール方法は本セミナー「第1回」で既述につき、省略します。
- ・これに加えて本セミナー「第1回」で紹介していない画像処理ライブラリの「Pillow (PIL)」  
を使用します。インストール方法は次節に述べます。
- ・共通の実装は、各モデルで使用する予測や表示などに用いる共通関数、および  
MNISTデータセットのロード処理です。

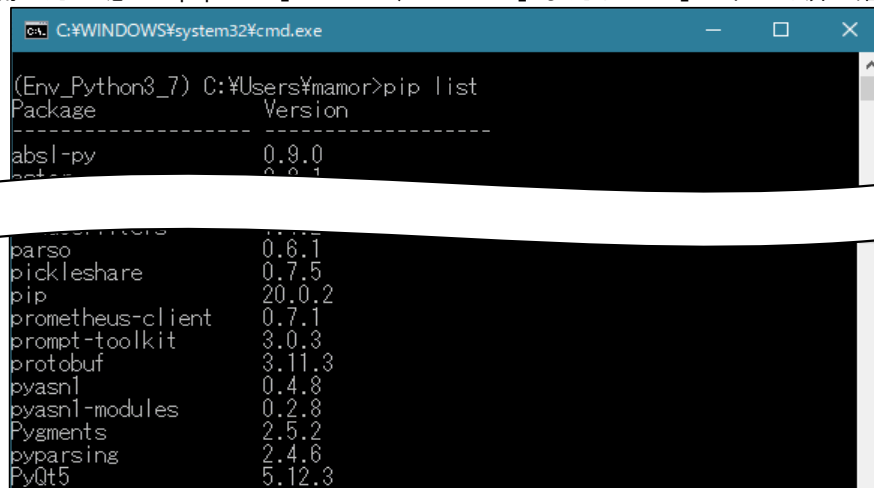
### (3.0.1) 使用するライブラリのインストール

- 自作の画像ファイル読み込みの為に、画像処理ライブラリの「Pillow(PIL)」を使用します。  
「Anaconda Navigator」の環境下では、以下の手順でこのライブラリを事前にインストールしておきます。  
「Google Colaboratory」の環境下では、この準備は不要です。

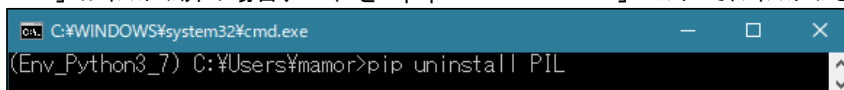
- (1) 「Anaconda Navigator」で使用している環境で「Open Terminal」で、DOS窓を開きます。



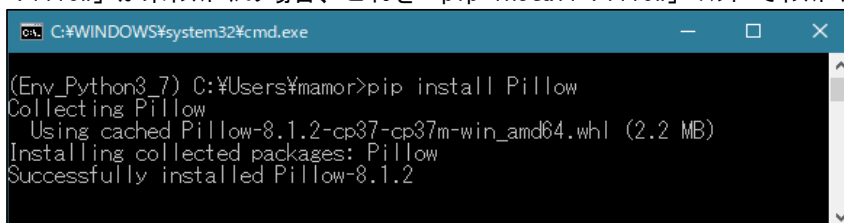
- (2) 開いたDOS窓で「pip list」コマンドで、「Pillow」または「PIL」がインストール済みを確認します。



- (3) 「PIL」がインストール済の場合、これを「pip uninstall PIL」コマンドでアンインストールします。



- (4) 「Pillow」が未インストールの場合、これを「pip install Pillow」コマンドでインストールします。



#### 【出典・参考】

Pythonの画像処理ライブラリPillow(PIL)の使い方⇒ <https://note.nkmk.me/python-pillow-basic/>

### (3.0.2) 各モデルでの共通の実装

- ・共通の実装は、各モデルで使用する予測や表示などに用いる共通関数、およびMNISTデータセットのロード処理です。その実装を以下に示します。

(リスト07-(03)-0\_各モデルでの共通の実装)

```
#####/
# リスト07-(03)-0_各モデルでの共通の実装
#####/
# 参照ライブラリ
import numpy as np
import matplotlib.pyplot as plt
from keras.datasets import mnist
from keras.utils import np_utils
from PIL import Image
import io

#####/
# print_MNIST_attr                                     */
# MNIST データセットの属性を表示する関数             */
#-----*/
# 引数 :                                               */
# titleList : 表示用タイトルのリスト                 */
#             (0:全体用、1~4 : xtrain, ytrain, xtest, ytest用) */
# xtrain    : モデル作成用データ : 説明変数          */
# ytrain    : モデル作成用データ : 出力結果          */
# xtest     : モデル検証用データ : 説明変数          */
# ytest     : モデル検証用データ : 出力結果          */
#-----*/
# 戻り値 : なし                                     */
#####/
def print_MNIST_attr(titleList, xtrain, ytrain, xtest, ytest):
    print(titleList[0])
    print(" {0}.dtype={1}".format(titleList[1], xtrain.dtype))
    print(" {0}.shape={1}".format(titleList[1], xtrain.shape))
    print(" {0}.range={1}~{2}".format(titleList[1], np.amin(xtrain), np.amax(xtrain)))
    print("")
    print(" {0}.dtype={1}".format(titleList[2], ytrain.dtype))
    print(" {0}.shape={1}".format(titleList[2], ytrain.shape))
    print(" {0}.range={1}~{2}".format(titleList[2], np.amin(ytrain), np.amax(ytrain)))
    print("")
    print(" {0}.shape={1}".format(titleList[3], xtest.shape))
    print(" {0}.shape={1}".format(titleList[4], ytest.shape))

#####/
# plot_learning_curve                                 */
# エポック回数に沿ったモデル学習の学習状況を、グラフ表示する関数 */
#-----*/
# 引数 :                                               */
# hist : モデル学習時の履歴情報 (Sequential.fit メソッドの戻り値) */
# epochNo : エポック回数                             */
#-----*/
# 戻り値 : なし                                     */
#####/
def plot_learning_curve(hist, epochNo):
    # print("history.keys={0}\n".format(hist.history.keys()))

    epochVal = range(1, epochNo+1)

    plt.figure(1, figsize=(10, 4))
    plt.subplots_adjust(wspace=0.5)
```

```

plt.subplot(1, 2, 1)
plt.plot(epochVal, hist.history['loss'], label='training', color='black')
plt.plot(epochVal, hist.history['val_loss'], label='test', color='cornflowerblue')
plt.ylim(0, 10)
plt.legend()
plt.grid()
plt.xlabel('epoch')
plt.ylabel('loss')

plt.subplot(1, 2, 2)
plt.plot(epochVal, hist.history['accuracy'], label='training', color='black')
plt.plot(epochVal, hist.history['val_accuracy'], label='test', color='cornflowerblue')
plt.ylim(0, 1)
plt.legend()
plt.grid()
plt.xlabel('epoch')
plt.ylabel('acc')

plt.show()

#####/
# pred_by_model */
# 作成したモデルで入力画像を分類する関数 */
#-----*/
# 引数 : */
#   predModel : 予測モデル (Sequential) */
#   inputData : 予測するデータ (予測モデル用のデータ型にしたもの) */
#-----*/
# 戻り値 : */
#   predData : 予測結果のデータ (float32 (入力画像枚数, 10)) */
#               入力画像の各クラス (0~9) への所属確率 [0~1] を表す */
#   predClass : 予測結果のデータ (uint8 (入力画像枚数)) */
#               入力画像の所属確率が最大のクラス */
#####/
def pred_by_model(predModel, inputData):
    # 入力画像の各クラスへの所属確率
    predData = predModel.predict(inputData)

    # 入力画像の所属確率が最大のクラス
    predClass = np.zeros(predData.shape[0], dtype=np.uint8)
    for i in range(predData.shape[0]):
        predClass[i] = np.argmax(predData[i, :])

    return predData, predClass

#####/
# show_result_images */
# 作成したモデルで入力画像を分類した結果を表示する関数 (その1) */
# 入力画像の所属クラスを画像とともに表示する。 */
# 引数で教師データ「teacherClass」が「None」以外で指定されていて、 */
# 分類結果が教師データと異なる場合、画像の上側に赤でバーを表示する。 */
#-----*/
# 引数 : */
#   inputData : 入力画像 (float32 (入力画像枚数, *)) */
#               (* : モデルによりデータ型は異なる) */
#   predClass : 分類結果のデータ (uint8 (入力画像枚数)) */
#               入力画像の所属確率が最大のクラス */
#   teacherClass: 教師データ (uint8 (入力画像枚数)) */
#               入力画像が表す数値 (0~9)、None指定可能 */
#   imgSizeX : 入力画像のサイズ X */
#   imgSizeY : 入力画像のサイズ Y */
#   n_showMax : 結果表示する入力画像数の最大枚数 */

```

```

#-----*/
# 戻り値：なし */
#-----*/
def show_result_images(inputData, predClass, teacherClass, imgSizeX, imgSizeY, n_showMax):
    dispN = np.amin( [inputData.shape[0], n_showMax] )
    dispX = 10
    dispY = np.int( (dispN + dispX - 1) / dispX )

    plt.figure(1, figsize=(2.8*dispX, 2.8*dispY))
    plt.subplots_adjust(wspace=0.1, hspace=0.1)
    plt.gray()
    for i in range(dispN):
        g = plt.subplot(dispY, dispX, i + 1)
        x = inputData[i, :]
        x = x.reshape(imgSizeY, imgSizeX)
        plt.pcolor(1 - x)

        dispcolor = "black"
        if (teacherClass is not None) and ¥
            (predClass[i] != teacherClass[i]):
            dispcolor = "red"
            plt.text(1.0, 4.5, "NG!", fontsize=25, color=dispcolor)

        plt.text(22, 25.5, "%d" % predClass[i], fontsize=20, color=dispcolor)
        plt.xticks([], "")
        plt.yticks([], "")
        plt.xlim(0, imgSizeX-1)
        plt.ylim(imgSizeY-1, 0)
    plt.show()

#-----*/
# show_result_probability */
# 作成したモデルで入力画像を分類した結果を表示する関数（その2） */
# 入力画像の各クラスへの所属確率[0～1]を表示する。 */
#-----*/
# 引数： */
# inputData : 入力画像（float32（入力画像枚数,*）） */
#             (*: モデルによりデータ型は異なる） */
# predData : 予測結果のデータ（float32（入力画像枚数,10）） */
#             入力画像の各クラス(0～9)への所属確率[0～1]を表す */
# n_showMax : 結果表示する入力画像数の最大枚数 */
#-----*/
# 戻り値：なし */
#-----*/
def show_result_probability(inputData, predData, n_showMax):
    dispN = np.amin( [inputData.shape[0], n_showMax] )
    for i in range(dispN):
        print("(class,probability) ", end="")
        for j in range(predData.shape[1]):
            print("{0}:{1:.2f} ".format(j, predData[i, j]), end="")
        print("")

#-----*/
# pred_test_and_show_result */
# 引数で指定したモデルで入力画像を分類し、 */
# 分類結果を最初の指定枚数分について表示する。 */
#-----*/
# 引数： */
# predModel : 分類モデル */
# inputData : 入力画像（float32（入力画像枚数,*）） */
#             (*: モデルによりデータ型は異なる） */
# teacherClass: 教師データ（uint8（入力画像枚数）） */

```

```

#          入力画像が表す数値 (0~9)          */
#   imgSizeX   : 入力画像のサイズ X          */
#   imgSizeY   : 入力画像のサイズ Y          */
#   n_showMax  : 結果表示する入力画像数の最大枚数          */
#-----*/
# 戻り値 : なし          */
#-----*/
def pred_test_and_show_result(predModel, inputData, teacherClass,
                              imgSizeX, imgSizeY, n_showMax):
    predData, predClass = pred_by_model(predModel, inputData)
    show_result_images(inputData, predClass, teacherClass,
                      imgSizeX, imgSizeY, n_showMax)

#-----*/
# pred_myData_and_show_result          */
# モデルとモデル用のデータ変換用関数を引数で指定し、          */
# 自作データで分類結果を表示する。          */
#-----*/
# 引数 :          */
#   title          : 表示用タイトル          */
#   predModel      : 分類モデル          */
#   reshape_for_model : モデル用のデータ変換用関数          */
#-----*/
# 戻り値 : なし          */
#-----*/
def pred_myData_and_show_result(title, predModel, reshape_for_model):
    # 読み込む画像ファイル情報(ファイルパス、分類クラス)のリスト
    fileInfoList = [['Handwritten_0.bmp', 0],
                    ['Handwritten_1.bmp', 1],
                    ['Handwritten_2.bmp', 2],
                    ['Handwritten_3.bmp', 3],
                    ['Handwritten_4.bmp', 4],
                    ['Handwritten_5.bmp', 5],
                    ['Handwritten_6.bmp', 6],
                    ['Handwritten_7.bmp', 7],
                    ['Handwritten_8.bmp', 8],
                    ['Handwritten_9.bmp', 9]]

    # タイトルを表示
    print(title)

    # 画像ファイル毎に読み込んで分類する
    totalImage = 0
    okImage = 0
    for fileInfo in fileInfoList:
        filePath = imageFolderPath + '/' + fileInfo[0]
        fileLabel = fileInfo[1]
        print("filePath={0}, label={1}".format(filePath, fileLabel))

        # 画像ファイルを読み込み
        with open(filePath, 'rb') as f:
            data = f.read()
            inputImg = Image.open(io.BytesIO(data))

        # 予測モデルに合うようにデータ変換
        inputImgArray = np.asarray(inputImg)
        inputImgArray = inputImgArray.reshape(1, inputImgArray.shape[0], inputImgArray.shape[1])
        inputLabel = [fileLabel]
        inputX, inputY = reshape_for_model(inputImgArray, inputLabel)

        # 予測モデルで分類し、その結果を表示
        predData, predClass = pred_by_model(predModel, inputX)

```

```

print("predData.dtype={0}, shape={1}, range={2} ~ {3}"
      .format(predData.dtype, predData.shape,
              np.amin(predData), np.amax(predData)))
show_result_probability(inputX, predData, 1)
show_result_images(inputX, predClass, inputLabel, IMG_SIZE_X, IMG_SIZE_Y, 1)

# 予測モデルで分類した結果と、期待される結果が等しいかを判定
totalimage = totalimage + 1
if inputLabel[0] == predClass[0]:
    okimage = okimage + 1

# 分類の集計結果
if totalimage > 0 :
    print("Total images={0}, OK images={1}, accuracy={2:.2f}".format(
        totalimage, okimage, okimage/totalimage ))
else:
    print("Total images={0}".format(totalimage))

#*****/
# MNIST データセットをロードする
# (1) x_train: 入力画像 : uint8 (入力画像枚数, 28, 28)
# (2) y_train: 入力画像が表す数値 (0~9) : uint8 (入力画像枚数)
# (3) x_test : 検証用画像 : uint8 (入力画像枚数, 28, 28)
# (4) y_test : 検証用画像が表す数値 (0~9) : uint8 (入力画像枚数)
#*****/
(x_train, y_train), (x_test, y_test) = mnist.load_data()

IMG_SIZE_Y = x_train.shape[1] # 画像サイズ(Y方向)
IMG_SIZE_X = x_train.shape[2] # 画像サイズ(X方向)
IMG_GRAY_LEVEL = 255         # 画像の諧調の最大値(モノクロ256諧調)
IMG_CLASS_NO = 10            # 画像の分類クラス数(0~9 の10クラス)

titleList = ["loaded ...", "x_train", "y_train", "x_test", "y_test"]
print_MNIST_attr(titleList, x_train, y_train, x_test, y_test)

```

「リスト07-(03)-0\_各モデルでの共通の実装」の実行結果は、以下のようになります：

```

loaded ...
x_train.dtype=uint8
x_train.shape=(60000, 28, 28)
x_train.range=0~255

y_train.dtype=uint8
y_train.shape=(60000,)
y_train.range=0~9

x_test.shape=(10000, 28, 28)
y_test.shape=(10000,)

```

#### 【出典・参考】

「SequentialモデルAPI」 <https://keras.io/ja/models/sequential/>



### (3.1) 「手書き文字認識モデル1」

- ・「手書き文字認識モデル1」では、  
2層の「順伝播型ニューラルネットワーク (Feed forward neural networks, FFNN)」を用いてモデルを構成します。
- ・「順伝播型ニューラルネットワーク」は、  
「第2回 機械学習のモデルとアルゴリズム」で概要を述べていますが、  
「第8回 機械学習 (2回目: 機械学習の概要と教師あり学習 (分類))」で詳説します。
- ・モデルとその学習方法は以下のようになります:

- (1) 入力層から出力層に向けて一方通行で流れる (Sequential)。
- (2) 入力層のノード数は、784 ( $28 \times 28$  ピクセルの2次元画像を一次元化したもの) である。  
値は、[0~255] の 256 階調のところを、  
255 で割って [0~1] に正規化したものを用いる。  
入力層の各ノードは中間層の各ノードとの間で、完全に接続させる (全結合)。
- (3) 入力と出力の間に、1つの中間層 (隠れ層) を置く。  
中間層のノード数は、16 である。  
中間層の活性化関数として  
「sigmoid (シグモイド関数, ロジスティック関数, sigmoid function, logistic function)」、  
または「relu (正規化線形関数, Rectified Linear Unit)」を用いる。  
中間層の各ノードは出力層の各ノードとの間で、完全に接続させる (全結合)。
- (4) 出力層のノード数は、10 (入力画像の文字の種類 (0~9) の数) である。  
データ型は「float32 (入力画像枚数, 10)」で、入力画像の各クラスへの所属確率 [0~1] を表す。  
出力層の活性化関数として「ソフトマックス関数 (softmax)」を用いる。
- (5) 損失関数 (誤差を計算する関数) として、「クロスエントロピー誤差 (cross entropy)」を用いる。
- (6) 学習方法 (モデルの最適化方法) として、勾配降下法的一种である「Adam」を用いる。
- (7) 学習時には「エポック数」として10、「バッチサイズ」として「1000」で学習を行う。

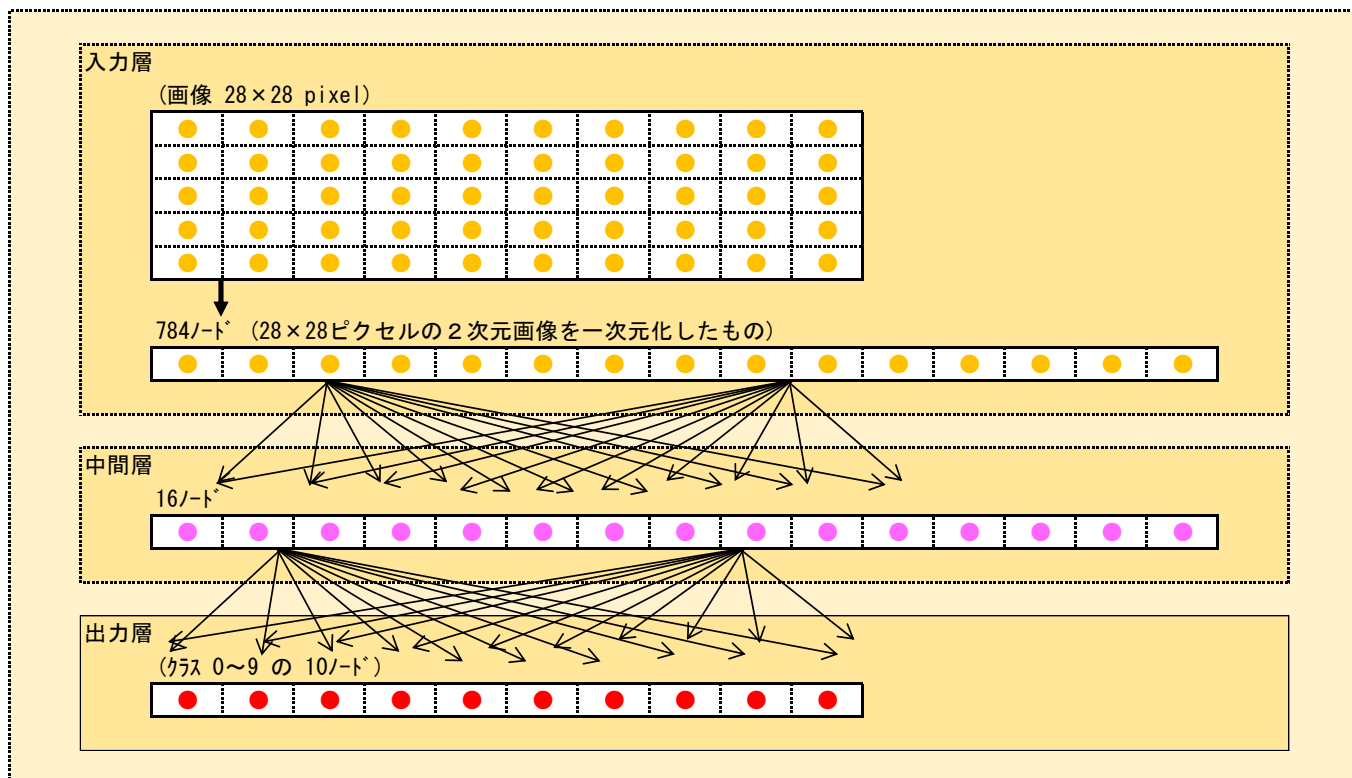
- ・MNIST を用いたモデル作成と評価、及び自作データでの認識精度

- (1) 中間層の活性化関数として「sigmoid」を用いた場合、  
MNIST の検証用画像データの認識精度は90%程度になっていますが、  
自作の画像データの認識精度は、10枚中4枚、つまり40%と低くなっています。
- (2) 中間層の活性化関数として「relu」を用いた場合、  
MNIST の検証用画像データの認識精度は93%程度で、  
「sigmoid」を用いた場合より改善はみられるものの、  
自作の画像データの認識精度は、10枚中5枚、つまり50%と低くなっています。
- (3) 何れの場合も、自作の画像データの認識モデルとして、実用レベルではないことがわかります。

#### 【出典・参考】

- ・「Pythonで動かして学ぶ! あたらしい機械学習の教科書」 (2018年01月 翔泳社 伊藤真著)
- ・クロスエントロピー誤差⇒ <https://qiita.com/celaeno42/items/7efdbb1491406f4bde96>
- ・学習方法⇒ <https://postd.cc/optimizing-gradient-descent/>

- ・「手書き文字認識モデル1」を図解すると以下のように2層になります：  
 (※ 以下の図を見ると3層または4層に見えますが、慣例的に入力層は層の厚さとしては加えないので、2層になります。)



### (3.1.1) 「手書き文字認識モデル1」の実装

- ・以下に、本モデルの実装を示します。

(リスト07-(03)-1\_手書き文字認識モデル1)

```
#####/
# リスト07-(03)-1_手書き文字認識モデル1
#-----
# 「手書き文字認識モデル1」
# 2層のフィードフォワードニューラルネットワーク (FNN)
#####/
# 参照ライブラリ
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import Adam
import matplotlib.pyplot as plt
%matplotlib inline
import time

#####/
# reshape_for_model1 */
# ロットした MNIST データを「手書き文字認識モデル1」 */
# 用に変形したものを返す */
#----- */
# 引数: */
# in_image : 入力画像: uint8 (入力画像枚数, 28, 28) */
# in_label : 入力画像が表す数値 (0~9): uint8 (入力画像枚数) */
#----- */
# 戻り値: */
# (1) in_image を */
# 「3次元(入力画像枚数, 28, 28)、uint8(0~255)」から */
# 「2次元(入力画像枚数, 784)、float32(0.0~1.0)」へ正規化したもの */
# (2) in_label を */
# 「1次元(入力画像枚数)、uint8(0~8: 入力画像が表す数値)」から */
# 「1-of-k 符号化」で 10クラスへ分類し「2次元(入力画像枚数, 10)」 */
# 各クラスへの所属確率を float32(0.0~1.0) で表現したもの */
# へ変換したもの */
#####/
def reshape_for_model1(in_image, in_label):
    out_image = in_image.reshape(in_image.shape[0], ¥
                                in_image.shape[1]*in_image.shape[2])
    out_image = out_image.astype('float32')
    out_image = out_image / IMG_GRAY_LEVEL
    out_label = np_utils.to_categorical(in_label, IMG_CLASS_NO)
    return out_image, out_label

#####/
# make_and_evaluate_model1 */
# 「手書き文字認識モデル1」を作成してそれを評価する */
# 隠れ層の活性化関数は引数で指定可能 */
#----- */
# 「手書き文字認識モデル1」 */
# 2層のフィードフォワードニューラルネットワーク (FNN) */
# (1) 入力層から出力層に向けて一方通行で流れる (Sequential)。 */
# (2) 入力層のノード数は、784 (=28×28, */
# つまり2次元画像を一次元化したもの)である。 */
# 値は、[0~255]の 256 階調のところを、 */
# 255 で割って[0~1] に正規化したものを用いる。 */
# 入力層の各ノードは中間層の各ノードとの間で、 */
# 完全に接続させる (全結合)。 */
# */
```

```

# (3) 入力と出力の間に、1つの中間層（隠れ層）を置く。 */
# 中間層のノード数は、16 である。 */
# 中間層の活性化関数として「sigmoid」または「relu」を用いる。 */
# 中間層の各ノードは出力層の各ノードとの間で、 */
# 完全に接続させる（全結合）。 */
# */
# (4) 出力層のノード数は、10（入力画像の文字の種類(0~9)の数)である。 */
# データ型は「float32（入力画像枚数,10)」で、 */
# 入力画像の各クラスへの所属確率[0~1]を表す。 */
# 出力層の活性化関数は「softmax」 */
# */
# (7) 損失関数(誤差を計算する関数)は「categorical_crossentropy」 */
# (8) 学習方法(モデルの最適化方法)は「Adam」 */
# (9) 学習時には「エポック数」として10、 */
# 「バッチサイズ」として「1000」で学習を行う。 */
#----- */
# 引数： */
# title : 表示用タイトル */
# actvFunc : 隠れ層の活性化関数 */
#----- */
# 戻り値： */
# (1) 作成したモデル */
#----- */
def make_and_evaluate_model1(title, actvFunc):

    print("*****")
    print(title)
    print("*****")

    #=====
    # ロードしたMNIST データを「手書き文字認識モデル1」用に変形する。
    #=====
    inTrain1, outTrain1 = reshape_for_model1(x_train, y_train)
    inTest1, outTest1 = reshape_for_model1(x_test, y_test)

    titlestr = ["modified ...", "inTrain1", "outTrain1", "inTest1", "outTest1"]
    print_MNIST_attr(titlestr, inTrain1, outTrain1, inTest1, outTest1)

    #=====
    # 「手書き文字認識モデル1」を作成
    #=====
    np.random.seed(1)
    predModel = Sequential()
    predModel.add(Dense(16, input_dim=inTrain1.shape[1], activation=actvFunc))
    predModel.add(Dense(IMG_CLASS_NO, activation='softmax'))
    predModel.compile(loss='categorical_crossentropy',
                      optimizer=Adam(), metrics=['accuracy'])
    predModel.summary()

    #=====
    # 「手書き文字認識モデル1」を学習し、
    # 学習状況を、エポック回数の時系列でグラフ表示する
    #=====
    epochNo = 10
    batchSize = 1000

    startTime = time.time()
    history = predModel.fit(inTrain1, outTrain1,
                           epochs=epochNo, batch_size=batchSize,
                           verbose=0, validation_data=(inTest1, outTest1))
    score = predModel.evaluate(inTest1, outTest1, verbose=0)
    endTime = time.time()

```

```

plot_learning_curve( history, epochNo )

print('Test loss:', score[0])
print('Test accuracy:', score[1])
print("Computation time: {0:.3f} sec".format(endTime - startTime))

#=====
# 「手書き文字認識モデル1」で、
# MINTSTの検証用データの最初の50枚で分類結果を表示する。
#=====
pred_test_and_show_result(predModel, inTest1, y_test, IMG_SIZE_X, IMG_SIZE_Y, 50)

return predModel

#-----/
# 「手書き文字認識モデル1」で
# 活性化関数を「sigmoid」でモデルを作成して実行する
#-----/
predModel1_1 = make_and_evaluate_model1(
    "「2層のフィードフォワードニューラルネットワーク（FNN）」で活性化関数は「sigmoid」", 'sigmoid')

# 「手書き文字認識モデル1」で、自作の画像の分類結果を表示する。
pred_myData_and_show_result("手書き文字認識モデル1（FNN, sigmoid）", predModel1_1, reshape_for_model1)

#-----/
# 「手書き文字認識モデル1」で
# 活性化関数を「relu」でモデルを作成して実行する
#-----/
predModel1_2 = make_and_evaluate_model1(
    "「2層のフィードフォワードニューラルネットワーク（FNN）」で活性化関数は「relu」", 'relu')

# 「手書き文字認識モデル1」で、自作の画像の分類結果を表示する。
pred_myData_and_show_result("手書き文字認識モデル1（FNN, relu）", predModel1_2, reshape_for_model1)

```

### (3.1.2) 「手書き文字認識モデル1」の実行結果

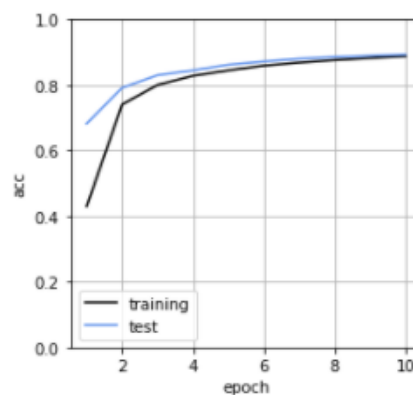
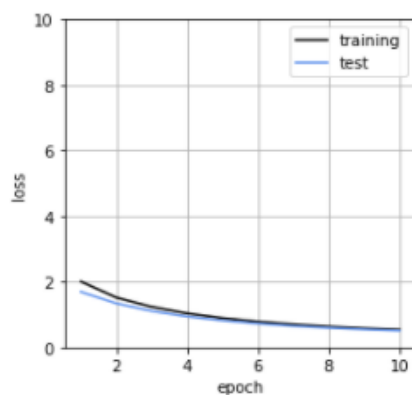
(1) 「2層のフィード・フォワード・ニューラルネットワーク (FNN)」で活性化関数を「sigmoid」で指定

```
*****
「2層のフィード・フォワード・ニューラルネットワーク (FNN)」で活性化関数は「sigmoid」
*****
modified ...
inTrain1.dtype=float32
inTrain1.shape=(60000, 784)
inTrain1.range=0.0~1.0

outTrain1.dtype=float32
outTrain1.shape=(60000, 10)
outTrain1.range=0.0~1.0

inTest1.shape=(10000, 784)
outTest1.shape=(10000, 10)
Model: "sequential_20"
```

Layer (type)	Output Shape	Param #
dense_39 (Dense)	(None, 16)	12560
dense_40 (Dense)	(None, 10)	170
Total params: 12,730		
Trainable params: 12,730		
Non-trainable params: 0		

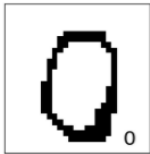


Test loss: 0.5084723676204681  
Test accuracy: 0.892799973487854  
Computation time: 3.275 sec

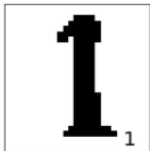


※ 作成したモデルを用いて、自作の手書き文字を認識させてみた結果（活性化関数は「sigmoid」）

```
手書き文字認識モデル (FNN, sigmoid)
filePath=HandwrittenImages/Handwritten_0.bmp, label=0
predData.dtype=float32, shape=(1, 10), range=0.005540387239307165~0.448080
30128479004
(class,probability) 0:0.45 1:0.01 2:0.06 3:0.05 4:0.02 5:0.18 6:0.01
7:0.01 8:0.21 9:0.01
```



```
filePath=HandwrittenImages/Handwritten_1.bmp, label=1
predData.dtype=float32, shape=(1, 10), range=0.0022451761178672314~0.58553
28056480408
(class,probability) 0:0.01 1:0.59 2:0.07 3:0.20 4:0.00 5:0.02 6:0.05
7:0.03 8:0.02 9:0.01
```



```
filePath=HandwrittenImages/Handwritten_2.bmp, label=2
predData.dtype=float32, shape=(1, 10), range=0.00048385222908109426~0.6559
147834777832
(class,probability) 0:0.03 1:0.03 2:0.66 3:0.03 4:0.00 5:0.03 6:0.21
7:0.00 8:0.02 9:0.00
```



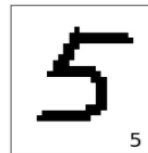
```
filePath=HandwrittenImages/Handwritten_3.bmp, label=3
predData.dtype=float32, shape=(1, 10), range=0.0032647065818309784~0.29544
87236895687
(class,probability) 0:0.30 1:0.08 2:0.20 3:0.22 4:0.00 5:0.02 6:0.11
7:0.04 8:0.02 9:0.00
```



```
filePath=HandwrittenImages/Handwritten_4.bmp, label=4
predData.dtype=float32, shape=(1, 10), range=0.0014404944377020001~0.62081
91514015198
(class,probability) 0:0.01 1:0.00 2:0.00 3:0.01 4:0.18 5:0.04 6:0.03
7:0.10 8:0.01 9:0.62
```



```
filePath=HandwrittenImages/Handwritten_5.bmp, label=5
predData.dtype=float32, shape=(1, 10), range=0.004803676623851061~0.545890
0928497314
(class,probability) 0:0.01 1:0.00 2:0.06 3:0.03 4:0.03 5:0.55 6:0.02
7:0.01 8:0.27 9:0.01
```



```
filePath=HandwrittenImages/Handwritten_6.bmp, label=6
predData.dtype=float32, shape=(1, 10), range=0.0038491764571517706~0.81168
29991340637
(class,probability) 0:0.00 1:0.01 2:0.01 3:0.06 4:0.01 5:0.01 6:0.01
7:0.81 8:0.00 9:0.07
```



```
filePath=HandwrittenImages/Handwritten_7.bmp, label=7
predData.dtype=float32, shape=(1, 10), range=0.0005901095573790371~0.61729
55632209778
(class,probability) 0:0.03 1:0.03 2:0.62 3:0.03 4:0.00 5:0.03 6:0.22
7:0.00 8:0.03 9:0.00
```



```
filePath=HandwrittenImages/Handwritten_8.bmp, label=8
predData.dtype=float32, shape=(1, 10), range=0.005075239645076752~0.883070
8861351013
(class,probability) 0:0.01 1:0.01 2:0.02 3:0.88 4:0.01 5:0.03 6:0.01
7:0.02 8:0.01 9:0.01
```



```
filePath=HandwrittenImages/Handwritten_9.bmp, label=9
predData.dtype=float32, shape=(1, 10), range=0.009878404438495636~0.294469
386391876
(class,probability) 0:0.01 1:0.01 2:0.16 3:0.11 4:0.11 5:0.09 6:0.02
7:0.29 8:0.17 9:0.02
```



Total images=10, OK images=4, accuracy=0.40

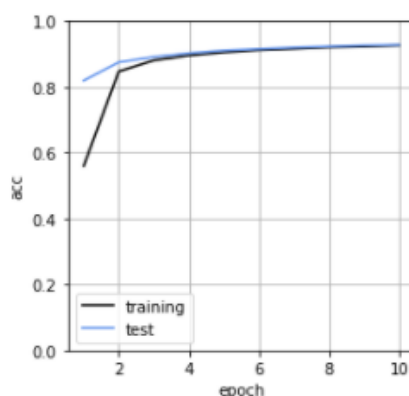
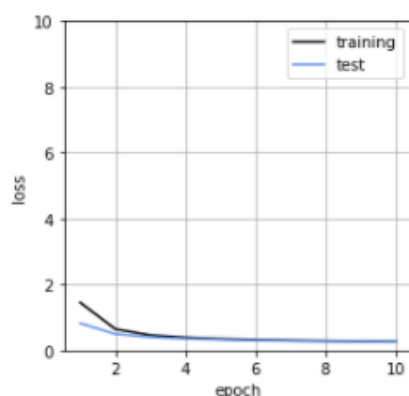
(2) 「2層のフィード・フォワード・ニューラルネットワーク (FNN)」で活性化関数を「relu」で指定

```
*****
「2層のフィード・フォワード・ニューラルネットワーク (FNN)」で活性化関数は「relu」
*****
modified ...
inTrain1.dtype=float32
inTrain1.shape=(60000, 784)
inTrain1.range=0.0~1.0

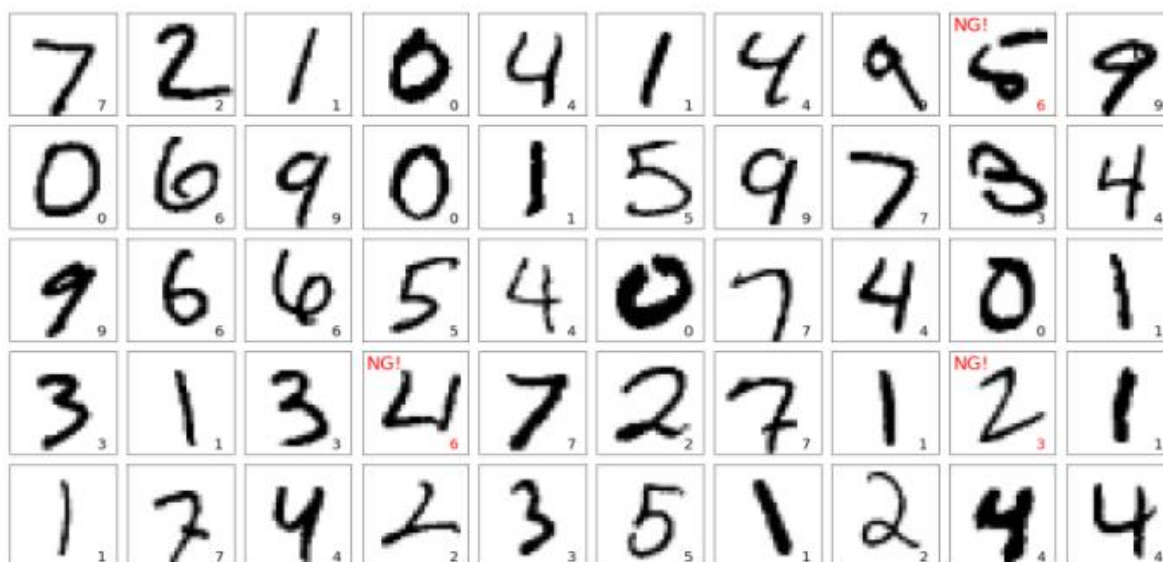
outTrain1.dtype=float32
outTrain1.shape=(60000, 10)
outTrain1.range=0.0~1.0

inTest1.shape=(10000, 784)
outTest1.shape=(10000, 10)
Model: "sequential_21"
```

Layer (type)	Output Shape	Param #
dense_41 (Dense)	(None, 16)	12560
dense_42 (Dense)	(None, 10)	170
Total params: 12,730		
Trainable params: 12,730		
Non-trainable params: 0		



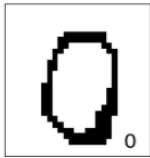
Test loss: 0.2558564089179039  
Test accuracy: 0.9276000261308763  
Computation time: 3.500 sec





※ 作成したモデルを用いて、自作の手書き文字を認識させてみた結果（活性化関数は「relu」）

手書き文字認識モデル1 (FNN, relu)  
 filePath=HandwrittenImages/Handwritten\_0.bmp, label=0  
 predData.dtype=float32, shape=(1, 10), range=1.270803750230698e-06~0.6655200719833374  
 (class,probability) 0:0.67 1:0.00 2:0.03 3:0.03 4:0.00 5:0.17 6:0.00 7:0.00 8:0.10 9:0.00



filePath=HandwrittenImages/Handwritten\_1.bmp, label=1  
 predData.dtype=float32, shape=(1, 10), range=1.3168801604734881e-09~0.678871214399801  
 (class,probability) 0:0.00 1:0.68 2:0.02 3:0.30 4:0.00 5:0.00 6:0.00 7:0.00 8:0.00 9:0.00



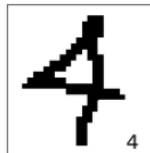
filePath=HandwrittenImages/Handwritten\_2.bmp, label=2  
 predData.dtype=float32, shape=(1, 10), range=1.5499344971532658e-13~0.960720956325531  
 (class,probability) 0:0.00 1:0.00 2:0.96 3:0.02 4:0.00 5:0.00 6:0.02 7:0.00 8:0.00 9:0.00



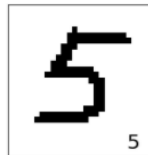
filePath=HandwrittenImages/Handwritten\_3.bmp, label=3  
 predData.dtype=float32, shape=(1, 10), range=8.18581147310482e-10~0.6498020887374878  
 (class,probability) 0:0.18 1:0.00 2:0.65 3:0.17 4:0.00 5:0.00 6:0.00 7:0.00 8:0.00 9:0.00



filePath=HandwrittenImages/Handwritten\_4.bmp, label=4  
 predData.dtype=float32, shape=(1, 10), range=2.364868669246789e-05~0.610945999622345  
 (class,probability) 0:0.00 1:0.00 2:0.00 3:0.00 4:0.61 5:0.06 6:0.01 7:0.05 8:0.00 9:0.27



filePath=HandwrittenImages/Handwritten\_5.bmp, label=5  
 predData.dtype=float32, shape=(1, 10), range=0.0004709748609457165~0.7546668264533997  
 (class,probability) 0:0.00 1:0.00 2:0.00 3:0.00 4:0.00 5:0.75 6:0.02 7:0.00 8:0.22 9:0.00



filePath=HandwrittenImages/Handwritten\_6.bmp, label=6  
 predData.dtype=float32, shape=(1, 10), range=2.407236297585934e-11~0.8505615592002869  
 (class,probability) 0:0.00 1:0.00 2:0.00 3:0.85 4:0.00 5:0.00 6:0.00 7:0.14 8:0.00 9:0.00



filePath=HandwrittenImages/Handwritten\_7.bmp, label=7  
 predData.dtype=float32, shape=(1, 10), range=9.934822062240833e-13~0.6998694539070129  
 (class,probability) 0:0.00 1:0.00 2:0.70 3:0.02 4:0.00 5:0.00 6:0.28 7:0.00 8:0.00 9:0.00



filePath=HandwrittenImages/Handwritten\_8.bmp, label=8  
 predData.dtype=float32, shape=(1, 10), range=5.236202027525394e-10~0.999163031578064  
 (class,probability) 0:0.00 1:0.00 2:0.00 3:1.00 4:0.00 5:0.00 6:0.00 7:0.00 8:0.00 9:0.00



filePath=HandwrittenImages/Handwritten\_9.bmp, label=9  
 predData.dtype=float32, shape=(1, 10), range=3.196702664354234e-06~0.8024590015411377  
 (class,probability) 0:0.00 1:0.00 2:0.01 3:0.15 4:0.00 5:0.02 6:0.00 7:0.02 8:0.80 9:0.00



Total images=10, OK images=5, accuracy=0.50

### (3.2) 「手書き文字認識モデル2」

- ・「手書き文字認識モデル1」では、画像は2次元データなのにわざわざ1次元データに焼きなおしてモデルに入力していたことに違和感を覚えた方もいらっしゃるかと思います。

「手書き文字認識モデル2」では、画像が2次元データであることを意識した

「畳み込みニューラルネットワーク (Convolutional neural network、CNN)」を用いてモデルを構成します。

- ・「畳み込みニューラルネットワーク」は、  
「第2回 機械学習のモデルとアルゴリズム」で概要を述べていますが、  
「第8回 機械学習（2回目：機械学習の概要と教師あり学習（分類）」で詳説します。
- ・「手書き文字認識モデル2」では、「畳み込みニューラルネットワーク」で  
中間層を1段のみとし、中間層を構成する畳み込み層を1段のみ、正規化層とプーリング層は無し、  
という、2層のモデルを作成します。

- ・モデルとその学習方法は以下のようなものです：

- (1) 入力層から出力層に向けて一方通行で流れる (Sequential)。
- (2) 入力層のノード数は、784 (28×28 ピクセルの2次元画像を一次元化したもの) である。  
値は、[0~255] の 256 階調のところを、  
255 で割って [0~1] に正規化したものを用いる。  
入力層の各ノードは中間層の各ノードとの間で、完全に接続させる (全結合)。
- (3) 入力と出力の間に、1つの中間層 (隠れ層) を置く。  
中間層のノード数は、16 である。  
中間層の活性化関数として  
「sigmoid (シグモイド関数, ロジスティック関数, sigmoid function, logistic function)」、  
または「relu (正規化線形関数, Rectified Linear Unit)」を用いる。  
中間層の各ノードは出力層の各ノードとの間で、完全に接続させる (全結合)。
- (4) 出力層のノード数は、10 (入力画像の文字の種類 (0~9) の数) である。  
データ型は「float32 (入力画像枚数, 10)」で、入力画像の各クラスへの所属確率 [0~1] を表す。  
出力層の活性化関数として「ソフトマックス関数 (softmax)」を用いる。
- (5) 損失関数 (誤差を計算する関数) として、「クロスエントロピー誤差 (cross entropy)」を用いる。
- (6) 学習方法 (モデルの最適化方法) として、勾配降下法的一种である「Adam」を用いる。
- (7) 学習時には「エポック数」として10、「バッチサイズ」として「1000」で学習を行う。

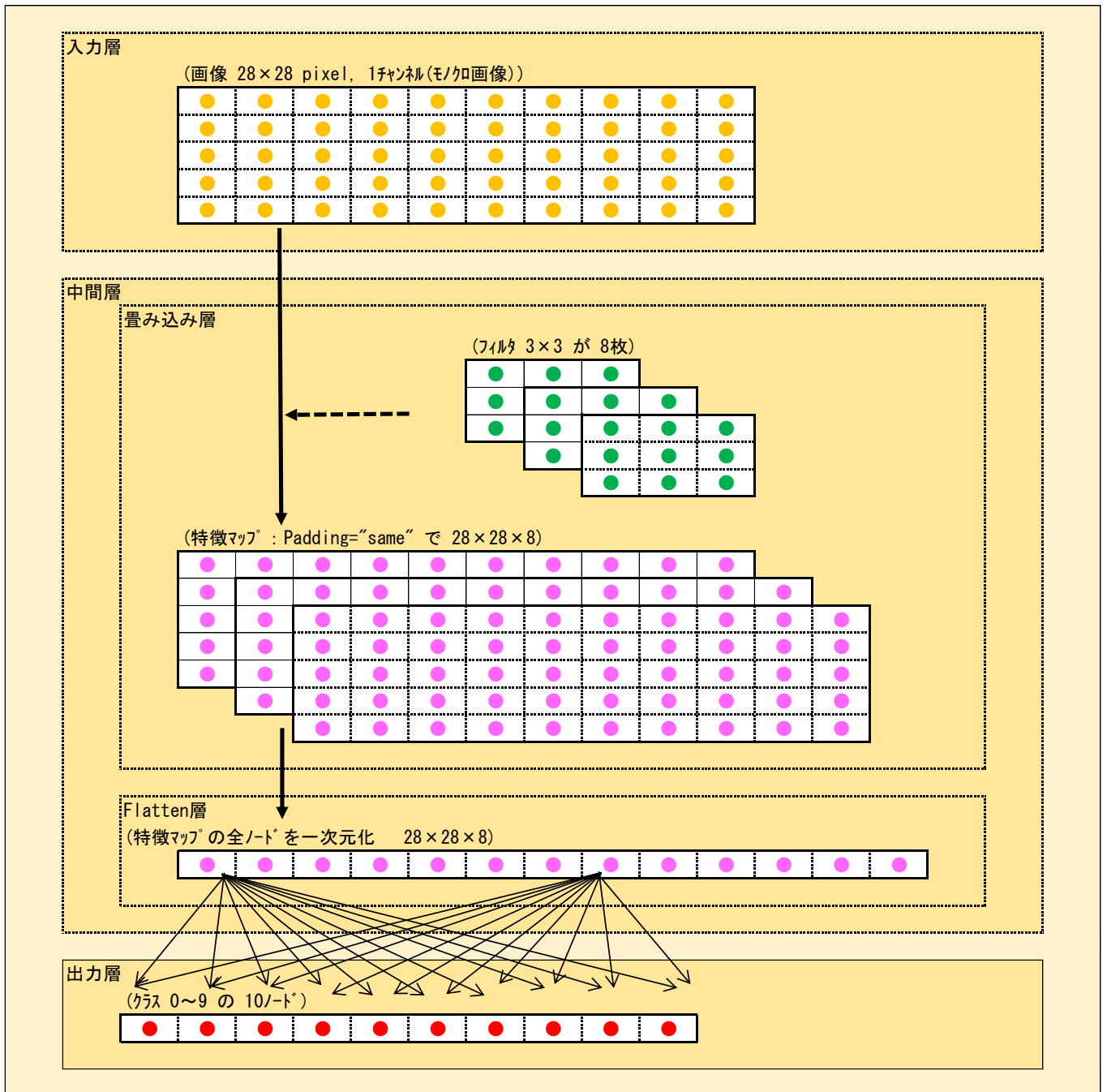
- ・MNIST を用いたモデル作成と評価、及び自作データでの認識精度

- (1) MNIST の検証用画像データの認識精度は96%程度で、  
「順伝播型ニューラルネットワーク (FFNN)」を用いた場合より改善はみられ、  
自作の画像データの認識精度も、10枚中6枚、つまり60%と上昇はしたものの  
それでも、自作の画像データの認識モデルとして、実用レベルではないことがわかります。

#### 【出典・参考】

- ・「Pythonで動かして学ぶ！ あたらしい機械学習の教科書」（2018年01月 翔泳社 伊藤真著）
- ・kerasでCNNモデル構築⇒ <https://qiita.com/sasayabaku/items/9e376ba8e38efe3bcf79>

- ・「手書き文字認識モデル2」を図解すると以下のように2層になります：  
 (※ 以下の図を見ると4層に見えますが、慣例的に入力層とFlatten層は層の厚さとしては加えないので、2層になります。)



### (3.2.1) 「手書き文字認識モデル2」の実装

- ・以下に、本モデルの実装を示します。

(リスト07-(03)-2\_手書き文字認識モデル2)

```
#####/
# リスト07-(03)-2_手書き文字認識モデル2
#-----
# 「手書き文字認識モデル2」
# 2層の畳み込みニューラルネットワーク (CNN)
#####/
# 参照ライブラリ
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.optimizers import Adam
import time

#####/
# reshape_for_model2 */
# ロットした MNIST データを「手書き文字認識モデル2」 */
# 用に変形したものを返す */
#----- */
# 引数: */
# in_image : 入力画像: uint8 (入力画像枚数, 28, 28) */
# in_label : 入力画像が表す数値 (0~9) : uint8 (入力画像枚数) */
#----- */
# 戻り値: */
# (1) in_image を */
# 「3次元(入力画像枚数, 28, 28)、uint8(0~255)」から */
# 「4次元(入力画像枚数, 28, 28, 1)、float32(0.0~1.0)」 */
# へ正規化したもの。 */
# 4次元目は「チャネル数」で本件はモノクロ画像なので1(カラー画像なら3) */
# (2) in_label を */
# 「1次元(入力画像枚数)、uint8(0~8: 入力画像が表す数値)」から */
# 「1-of-k 符号化」で 10クラスへ分類し「2次元(入力画像枚数, 10)」 */
# 各クラスへの所属確率を float32(0.0~1.0) で表現したもの */
# へ変換したもの */
#####/
def reshape_for_model2(in_image, in_label):
    out_image = in_image.reshape(in_image.shape[0], ¥
                                in_image.shape[1], in_image.shape[2], 1)
    out_image = out_image.astype('float32')
    out_image = out_image / IMG_GRAY_LEVEL
    out_label = np_utils.to_categorical(in_label, IMG_CLASS_NO)
    return out_image, out_label

#####/
# make_and_evaluate_model2 */
# 「手書き文字認識モデル2」を作成してそれを評価する */
# 隠れ層の活性化関数は引数で指定可能 */
#----- */
# 「手書き文字認識モデル2」 */
# 2層の畳み込みニューラルネットワーク (CNN) */
# */
# (1) 入力層から出力層に向けて一方通行で流れる (Sequential)。 */
# (2) 入力層の入力データは、 */
# (Y方向サイズ, X方向サイズ, 入力チャネル数) = (28, 28, 1) とする。 */
# 「入力チャネル数」はモノクロ画像なので1(カラー画像なら3)。 */
# 値は、[0~255]の 256 階調のところを、 */
```

```

# 255 で割って[0~1] に正規化したものを用いる。 */
# */
# (3) 入力と出力の間に、1つの中間層（隠れ層）を置く。 */
# 中間層を1段のみとし、中間層を構成する畳み込み層は1段のみ、 */
# 正規化層とプーリング層は無し。 */
# 畳み込み層のフィル数は8で、フィルサイズは3×3とする。 */
# Padding="same" でフィルをかけることにより、 */
# 結果の特徴マップ 28×28 を8枚で出力する。 */
# 中間層の出力を「Flatten」層を通すことで、 */
# 「28×28×8枚×入力チャネル数」という */
# 4次元のデータを一次元化して、出力層との間で全結合させる。 */
# */
# (4) 出力層のノード数は、10（入力画像の文字の種類(0~9)の数)である */
# データ型は「float32（入力画像枚数,10）」で、 */
# 入力画像の各クラスへの所属確率[0~1]を表す。 */
# */
# (5) 中間層の活性化関数は「relu」 */
# (6) 出力層の活性化関数は「softmax」 */
# (7) 損失関数(誤差を計算する関数)は「categorical_crossentropy」 */
# (8) 学習方法(モデルの最適化方法)は「Adam」 */
# (9) 学習時には「エポック数」として10、 */
# 「バッチサイズ」として「1000」で学習を行う。 */
# ----- */
# 引数： */
# title : 表示用タイトル */
# actvFunc : 隠れ層の活性化関数 */
# ----- */
# 戻り値： */
# (1) 作成したモデル */
# ----- */
def make_and_evaluate_model2(title, actvFunc):

    print("*****")
    print(title)
    print("*****")

    #=====
    # ロードしたMNIST データを「手書き文字認識モデル2」用に変形する。
    #=====
    inTrain2, outTrain2 = reshape_for_model2(x_train, y_train)
    inTest2, outTest2 = reshape_for_model2(x_test, y_test)

    titlestr = ["modified ...", "inTrain2", "outTrain2", "inTest2", "outTest2"]
    print_MNIST_attr(titlestr, inTrain2, outTrain2, inTest2, outTest2)

    #=====
    # 「手書き文字認識モデル2」を作成
    #=====
    np.random.seed(1)
    predmodel = Sequential()
    predmodel.add(Conv2D(8, (3, 3), padding='same',
        input_shape=(IMG_SIZE_Y, IMG_SIZE_X, 1), activation=actvFunc))
    predmodel.add(Flatten())
    predmodel.add(Dense(IMG_CLASS_NO, activation='softmax'))
    predmodel.compile(loss='categorical_crossentropy',
        optimizer=Adam(), metrics=['accuracy'])
    predmodel.summary()

    #=====
    # 「手書き文字認識モデル2」を学習し、
    # 学習状況を、エポック回数の時系列でグラフ表示する
    #=====

```

```

epochNo = 10
batchSize = 1000

startTime = time.time()
history = predmodel.fit(inTrain2, outTrain2,
                        epochs=epochNo, batch_size=batchSize,
                        verbose=0, validation_data=(inTest2, outTest2))
score = predmodel.evaluate(inTest2, outTest2, verbose=0)
endTime = time.time()

print('Test loss:', score[0])
print('Test accuracy:', score[1])
print("Computation time: {0:.3f} sec".format(time.time() - startTime))

plot_learning_curve(history, epochNo)

#=====
# 「手書き文字認識モデル2」で、
# MINTSTの検証用データの最初の50枚で分類結果を表示する。
#=====
pred_test_and_show_result(predmodel, inTest2, y_test, IMG_SIZE_X, IMG_SIZE_Y, 50)

return predmodel

#*****/
# 「手書き文字認識モデル2」で
# 活性化関数を「relu」でモデルを作成して評価する
#*****/
predmodel2 = make_and_evaluate_model2(
    "「2層の畳み込みニューラルネットワーク（CNN）」で活性化関数は「relu」", 'relu')

#*****/
# 「手書き文字認識モデル2」で、
# 自作の画像の分類結果を表示する。
#*****/
pred_myData_and_show_result("手書き文字認識モデル2（CNN,relu）", predmodel2, reshape_for_model2)

```

### (3.2.2) 「手書き文字認識モデル2」の実行結果

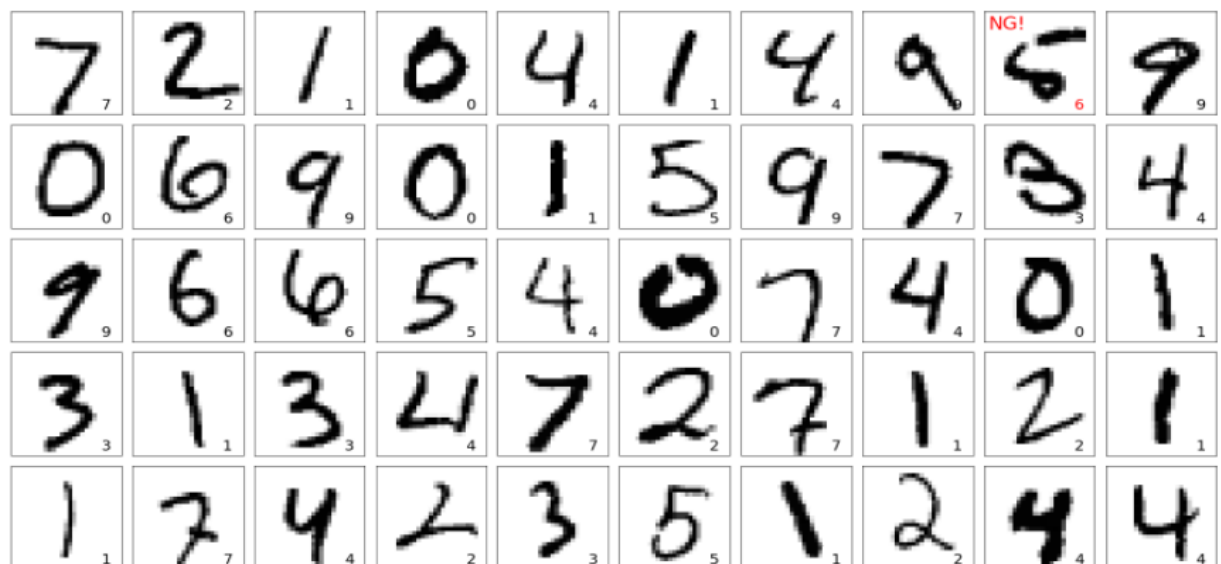
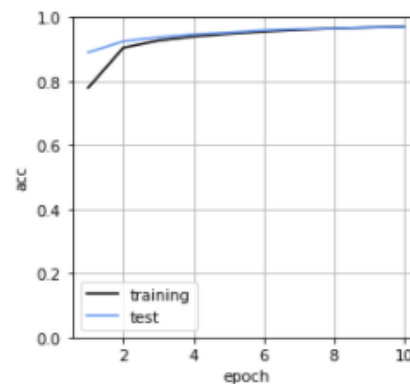
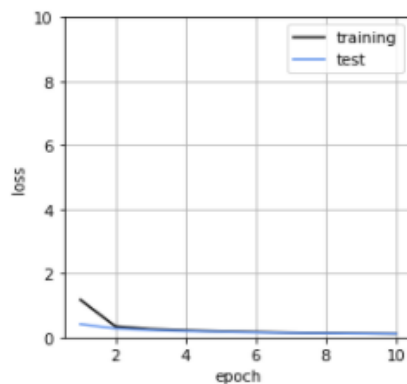
```
*****
「2層の畳み込みニューラルネットワーク(CNN)」で活性化関数は「relu」
*****
modified ...
inTrain2.dtype=float32
inTrain2.shape=(60000, 28, 28, 1)
inTrain2.range=0.0~1.0

outTrain2.dtype=float32
outTrain2.shape=(60000, 10)
outTrain2.range=0.0~1.0

inTest2.shape=(10000, 28, 28, 1)
outTest2.shape=(10000, 10)
Model: "sequential_22"
```

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 28, 28, 8)	80
flatten_1 (Flatten)	(None, 6272)	0
dense_43 (Dense)	(None, 10)	62730
Total params: 62,810		
Trainable params: 62,810		
Non-trainable params: 0		

Test loss: 0.10787517099454999  
Test accuracy: 0.9693999886512756  
Computation time: 36.497 sec



## ※ 作成したモデルを用いて、自作の手書き文字を認識させてみた結果

手書き文字認識モデル2 (CNN,relu)  
 filePath=HandwrittenImages/Handwritten\_0.bmp, label=0  
 predData.dtype=float32, shape=(1, 10), range=5.904797539812989e-09~0.7785465121269226  
 (class,probability) 0:0.78 1:0.00 2:0.10 3:0.00 4:0.00 5:0.04 6:0.00 7:0.00 8:0.08 9:0.00



filePath=HandwrittenImages/Handwritten\_1.bmp, label=1  
 predData.dtype=float32, shape=(1, 10), range=1.2748231803527688e-08~0.9368743300437927  
 (class,probability) 0:0.00 1:0.94 2:0.02 3:0.04 4:0.00 5:0.00 6:0.00 7:0.00 8:0.00 9:0.00



filePath=HandwrittenImages/Handwritten\_2.bmp, label=2  
 predData.dtype=float32, shape=(1, 10), range=1.40542268036066e-13~0.9972469806671143  
 (class,probability) 0:0.00 1:0.00 2:1.00 3:0.00 4:0.00 5:0.00 6:0.00 7:0.00 8:0.00 9:0.00



filePath=HandwrittenImages/Handwritten\_3.bmp, label=3  
 predData.dtype=float32, shape=(1, 10), range=1.3303142587517414e-08~0.9545195698738098  
 (class,probability) 0:0.01 1:0.00 2:0.03 3:0.95 4:0.00 5:0.00 6:0.00 7:0.00 8:0.01 9:0.00



filePath=HandwrittenImages/Handwritten\_4.bmp, label=4  
 predData.dtype=float32, shape=(1, 10), range=1.8414050373394275e-06~0.923552393913269  
 (class,probability) 0:0.00 1:0.00 2:0.00 3:0.00 4:0.92 5:0.00 6:0.00 7:0.03 8:0.01 9:0.03



filePath=HandwrittenImages/Handwritten\_5.bmp, label=5  
 predData.dtype=float32, shape=(1, 10), range=5.767987754090598e-10~0.9942331910133362  
 (class,probability) 0:0.00 1:0.00 2:0.00 3:0.00 4:0.00 5:0.99 6:0.00 7:0.00 8:0.00 9:0.01



filePath=HandwrittenImages/Handwritten\_6.bmp, label=6  
 predData.dtype=float32, shape=(1, 10), range=1.2292263040780682e-10~0.9986959099769592  
 (class,probability) 0:0.00 1:0.00 2:0.00 3:1.00 4:0.00 5:0.00 6:0.00 7:0.00 8:0.00 9:0.00



filePath=HandwrittenImages/Handwritten\_7.bmp, label=7  
 predData.dtype=float32, shape=(1, 10), range=9.84816117544085e-10~0.8856104803085327  
 (class,probability) 0:0.00 1:0.00 2:0.87 3:0.10 4:0.00 5:0.00 6:0.02 7:0.00 8:0.01 9:0.00



filePath=HandwrittenImages/Handwritten\_8.bmp, label=8  
 predData.dtype=float32, shape=(1, 10), range=6.616740488851747e-11~0.9996434450149538  
 (class,probability) 0:0.00 1:0.00 2:0.00 3:1.00 4:0.00 5:0.00 6:0.00 7:0.00 8:0.00 9:0.00



filePath=HandwrittenImages/Handwritten\_9.bmp, label=9  
 predData.dtype=float32, shape=(1, 10), range=4.141669673884962e-09~0.8148922324180603  
 (class,probability) 0:0.00 1:0.00 2:0.00 3:0.18 4:0.00 5:0.00 6:0.00 7:0.00 8:0.81 9:0.00



Total images=10, OK images=6, accuracy=0.60



### (3.3) 「手書き文字認識モデル3」

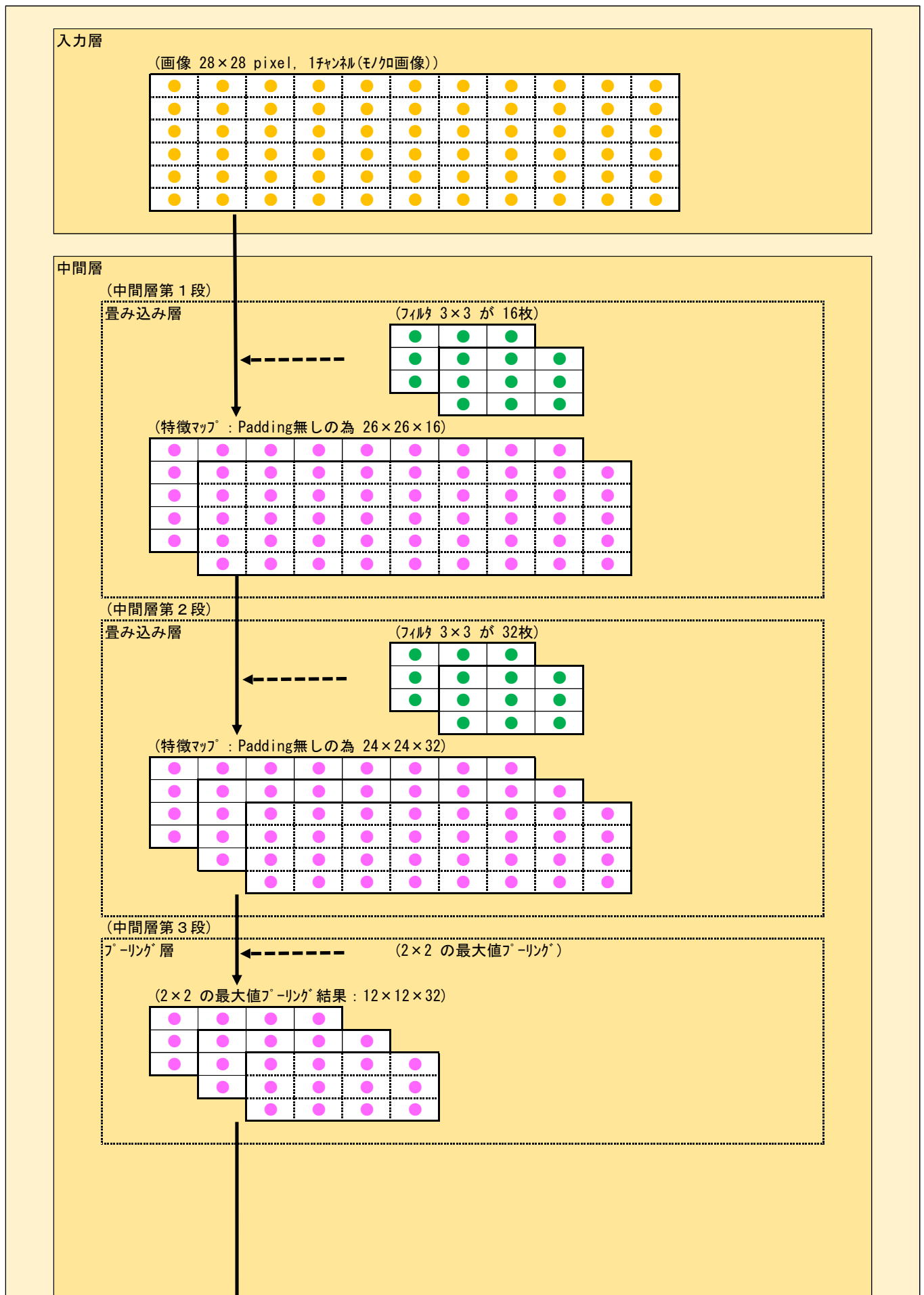
- ・「手書き文字認識モデル2」では、「畳み込みニューラルネットワーク (CNN)」を用いることにより、「順伝播型ニューラルネットワーク (FFNN)」を用いた場合より改善はみられたものの、それでも、自作の画像データの認識モデルとして、実用レベルではありませんでした。
- ・「手書き文字認識モデル3」では、「手書き文字認識モデル2」よりも中間層を厚くすると共に、「プーリング層」や「全結合層」を追加し、一部の層に「ドロップアウト率」を指定します。「畳み込みニューラルネットワーク (CNN)」で、層を厚くしたものなので「深層畳み込みニューラルネットワーク (Deep Convolutional neural network、DCN)」というモデルになります。いよいよディープラーニングですね・・・。
- ・モデルとその学習方法は以下のようなものです：

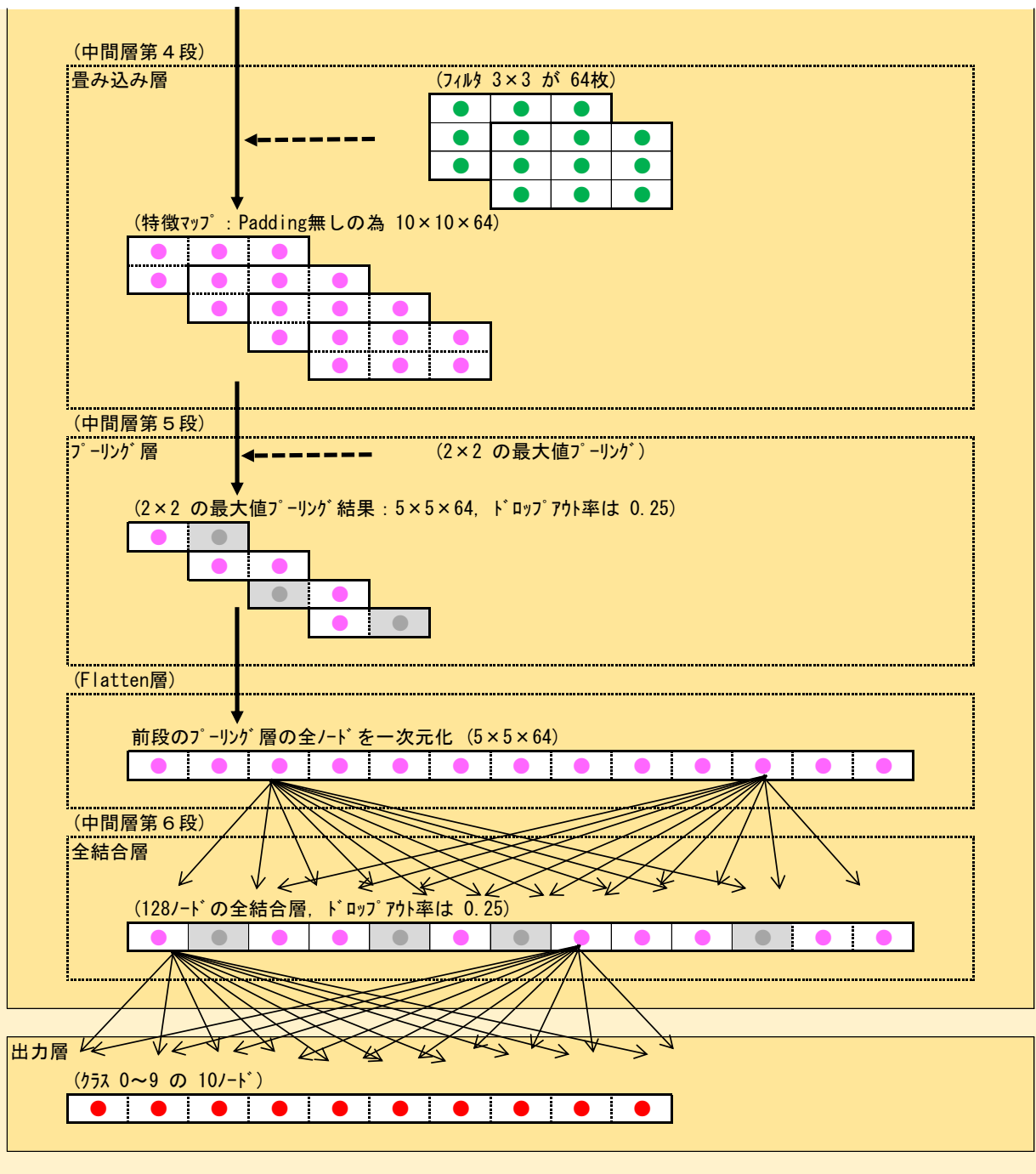
- (1) 入力層から出力層に向けて一方通行で流れる (Sequential)。
- (2) 入力層の入力データは、(Y方向サイズ, X方向サイズ, 入力チャネル数) = (28, 28, 1) とする。  
「入力チャネル数」はモノクロ画像なので1(カラー画像なら3)。  
値は、0~255 の 256 階調のところを、255 で割って 0~1 に正規化したものを用いる。
- (3) 入力と出力の間に、中間層 (隠れ層) を置く。  
中間層を以下の6段で構成する。
  - (3.1) 中間層第1段は、畳み込み層で、フィルタ  $3 \times 3$  を 16枚で、Padding無しで、畳み込む。
  - (3.2) 中間層第2段は、畳み込み層で、フィルタ  $3 \times 3$  を 32枚で、Padding無しで、畳み込む。
  - (3.3) 中間層第3段は、 $2 \times 2$ の最大値プーリング層。
  - (3.4) 中間層第4段は、畳み込み層で、フィルタ  $3 \times 3$  を 64枚で、Padding無しで、畳み込む。
  - (3.5) 中間層第5段は、 $2 \times 2$ の最大値プーリング層で、ドロップアウト率を 0.25 とする。  
この出力を、Flatten層を通して一次元化し、  
一次元化したノードを、中間層第6段の各ノードとの間で全結合させる。
  - (3.6) 中間層第6段は、全結合層で、ドロップアウト率を 0.25 とする。  
各ノードを、出力層の各ノードとの間で全結合させる。
- (4) 出力層のノード数は、10 (入力画像の文字の種類 (0~9) の数) である。  
データ型は「float32 (入力画像枚数, 10)」で、入力画像の各クラスへの所属確率 [0~1] を表す。  
中間層の出力を「Flatten」層を通すことで、「 $28 \times 28 \times 8 \text{枚} \times \text{入力チャネル数}$ 」という4次元のデータを一次元化して、出力層との間で全結合させる。
- (5) 中間層の活性化関数として「relu (正規化線形関数, Rectified Linear Unit)」を用いる。
- (6) 出力層の活性化関数として「ソフトマックス関数 (softmax)」を用いる。
- (7) 損失関数 (誤差を計算する関数) として、「クロスエントロピー誤差 (cross entropy)」を用いる。
- (8) 学習方法 (モデルの最適化方法) として、勾配降下法の一種である「Adam」を用いる。
- (9) 学習時には「エポック数」として10、「バッチサイズ」として「1000」で学習を行う。


- ・MNIST を用いたモデル作成と評価、及び自作データでの認識精度

- (1) MNIST の検証用画像データの認識精度は99%と高く、  
2層の「畳み込みニューラルネットワーク (CNN)」を用いた場合より改善がみられ、  
自作の画像データの認識精度も、10枚中8枚、つまり80%と大分向上しました。  
然しながら、自作の画像データの認識モデルとして、  
実用レベルにはもう少し改善する必要があるでしょう。

- ・「手書き文字認識モデル3」を図解すると以下のように全部で7層になります：  
 (※ 以下の図を見ると9層に見えますが、慣例的に入力層とFlatten層は層の厚さとしては加えないので、7層になります。)





 : ドロップアウト対象のノードを示します

### (3.3.1) 「手書き文字認識モデル3」の実装

- ・以下に、本モデルの実装を示します。

(リスト07-(03)-3\_手書き文字認識モデル3 )

```
#####/
# リスト07-(03)-3_手書き文字認識モデル3
#-----
# 「手書き文字認識モデル3」
# 深層畳み込みニューラルネットワーク (DCNN) ・ ・ ・ 最大値プーリング 使用
#####/
# 参照ライブラリ
import numpy as np
np.random.seed(1)
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.optimizers import Adam
import time

#####/
# reshape_for_model3                                     */
# ロットした MNIST データを「手書き文字認識モデル3」      */
# 用に変形したものを返す                                     */
#-----*/
# 引数 :                                                  */
#   in_image : 入力画像 : uint8 (入力画像枚数, 28, 28)      */
#   in_label : 入力画像が表す数値 (0~9) : uint8 (入力画像枚数) */
#-----*/
# 戻り値 :                                                  */
# (1) in_image を                                           */
#   「3次元(入力画像枚数, 28, 28)、uint8(0~255)」から        */
#   「4次元(入力画像枚数, 28, 28, 1)、float32(0.0~1.0)」      */
#   へ正規化したもの                                         */
# (2) in_label を                                           */
#   「1次元(入力画像枚数)、uint8(0~8: 入力画像が表す数値)」から */
#   「1-of-k 符号化」で 10クラスへ分類し「2次元(入力画像枚数, 10)」 */
#   各クラスへの所属確率を float32(0.0~1.0) で表現したもの   */
#   へ変換したもの                                           */
#####/
def reshape_for_model3(in_image, in_label):
    return reshape_for_model2(in_image, in_label)

#####/
# make_and_evaluate_model3                               */
# 「手書き文字認識モデル3」を作成してそれを評価する        */
#   隠れ層の活性化関数は引数で指定可能                     */
#-----*/
# 「手書き文字認識モデル3」                               */
# 深層畳み込みニューラルネットワーク (DCNN)                */
#-----*/
# (1) 入力層から出力層に向けて一方通行で流れる (Sequential)。 */
# (2) 入力層の入力データは、                                */
#   (Y方向サイズ, X方向サイズ, 入力チャネル数) = (28, 28, 1) とする。 */
#   「入力チャネル数」はモノクロ画像なので1(カラー画像なら3)。 */
#   値は、0~255 の 256 階調のところを、 */
#   255 で割って 0~1 に正規化したものを用いる。 */
#-----*/
# (3) 入力と出力の間に、中間層 (隠れ層) を置く。          */
#   中間層を以下の6段で構成する。 */
# (3.1) 中間層第1段は、畳み込み層で、 */
#   フィルタ 3×3 を 16枚で、Padding無しで、畳み込む。      */
```

```

# (3.2) 中間層第2段は、畳み込み層で、 */
# フィルタ 3×3 を 32枚で、Padding無しで、畳み込む。 */
# (3.3) 中間層第3段は、2×2の最大値プーリング層。 */
# (3.4) 中間層第4段は、畳み込み層で、 */
# フィルタ 3×3 を 64枚で、Padding無しで、畳み込む。 */
# (3.5) 中間層第5段は、2×2の最大値プーリング層で、 */
# ドロップアウト率を 0.25 とする。 */
# この出力を、Flatten層を通して一次元化する。 */
# 一次元化したノードを、中間層第6段の各ノードとの間で全結合する。 */
# (3.6) 中間層第6段は、全結合層で、ドロップアウト率を 0.25 とする。 */
# 各ノードを、出力層の各ノードとの間で全結合させる。 */
# */
# (4) 出力層のノード数は、10（入力画像の文字の種類(0～9)の数)である。 */
# データ型は「float32（入力画像枚数, 10）」で、 */
# 入力画像の各クラスへの所属確率[0～1]を表す。 */
# 中間層の出力を「Flatten」層を通すことで、 */
# 「28×28×8枚×入力チャネル数」という */
# 4次元のデータを一次元化して、出力層との間で全結合させる。 */
# */
# (5) 中間層の活性化関数は「relu」 */
# (6) 出力層の活性化関数は「softmax」 */
# (7) 損失関数(誤差を計算する関数)は「categorical_crossentropy」 */
# (8) 学習方法(モデルの最適化方法)は「Adam」 */
# (9) 学習時には「エポック数」として10、 */
# 「バッチサイズ」として「1000」で学習を行う。 */
# ----- */
# 引数 : */
# title : 表示用タイトル */
# actvFunc : 隠れ層の活性化関数 */
# ----- */
# 戻り値 : */
# (1) 作成したモデル */
# ----- */
def make_and_evaluate_model3(title, actvFunc):

    print("*****")
    print(title)
    print("*****")

    #=====
    # ロットしたMNIST データを「手書き文字認識モデル3」用に変形する。
    #=====
    inTrain3, outTrain3 = reshape_for_model3(x_train, y_train)
    inTest3, outTest3 = reshape_for_model3(x_test, y_test)

    titlestr = ["modified ...", "inTrain3", "outTrain3", "inTest3", "outTest3"]
    print_MNIST_attr(titlestr, inTrain3, outTrain3, inTest3, outTest3)

    #=====
    # 「手書き文字認識モデル3」を作成
    #=====
    np.random.seed(1)

    predModel = Sequential()
    predModel.add(Conv2D(16, (3, 3),
        input_shape=(IMG_SIZE_Y, IMG_SIZE_X, 1), activation=actvFunc))
    predModel.add(Conv2D(32, (3, 3), activation=actvFunc))
    predModel.add(MaxPooling2D(pool_size=(2, 2)))
    predModel.add(Conv2D(64, (3, 3), activation=actvFunc))
    predModel.add(MaxPooling2D(pool_size=(2, 2)))
    predModel.add(Dropout(0.25))
    predModel.add(Flatten())

```

```

predModel.add(Dense(128, activation=actvFunc))
predModel.add(Dropout(0.25))
predModel.add(Dense(IMG_CLASS_NO, activation='softmax'))
predModel.compile(loss='categorical_crossentropy',
    optimizer=Adam(), metrics=['accuracy'])

predModel.summary()

#=====
# 「手書き文字認識モデル3」を学習し、
# 学習状況を、エポック回数の時系列でグラフ表示する
#=====
epochNo = 10
batchSize = 1000

startTime = time.time()
history = predModel.fit(inTrain3, outTrain3,
    epochs=epochNo, batch_size=batchSize,
    verbose=0, validation_data=(inTest3, outTest3))
score = predModel.evaluate(inTest3, outTest3, verbose=0)
endTime = time.time()

print('Test loss:', score[0])
print('Test accuracy:', score[1])
print("Computation time: {0:.3f} sec".format(time.time() - startTime))

plot_learning_curve(history, epochNo)

#=====
# 「手書き文字認識モデル3」で、
# MINTSTの検証用データの最初の50枚で分類結果を表示する。
#=====
pred_test_and_show_result(predModel, inTest3, y_test, IMG_SIZE_X, IMG_SIZE_Y, 50)

return predModel

#####/
# 「手書き文字認識モデル3」で
# 活性化関数を「relu」でモデルを作成して評価する
#####/
predModel3 = make_and_evaluate_model3(
    "「深層畳み込みニューラルネットワーク (DCNN)」で活性化関数は「relu」", 'relu')

#####/
# 「手書き文字認識モデル3」で、
# 自作の画像の分類結果を表示する。
#####/
pred_myData_and_show_result("手書き文字認識モデル3 (DCNN, relu)",
    predModel3, reshape_for_model3)

```

### (3.3.2) 「手書き文字認識モデル3」の実行結果

```
*****
「深層畳み込みニューラルネットワーク (DCNN)」で活性化関数は「relu」
*****
modified ...
inTrain3.dtype=float32
inTrain3.shape=(60000, 28, 28, 1)
inTrain3.range=0.0~1.0

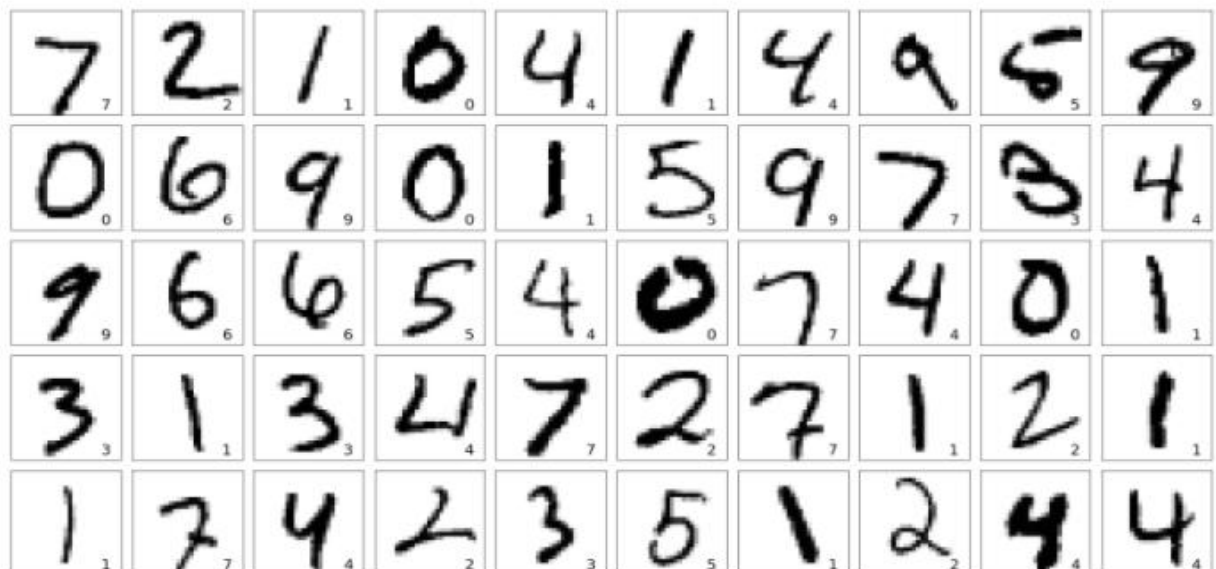
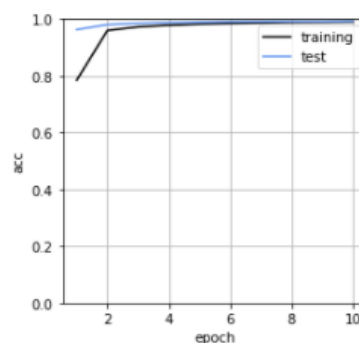
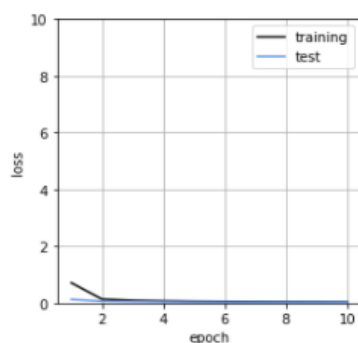
outTrain3.dtype=float32
outTrain3.shape=(60000, 10)
outTrain3.range=0.0~1.0

inTest3.shape=(10000, 28, 28, 1)
outTest3.shape=(10000, 10)
Model: "sequential_4"
```

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 26, 26, 16)	180
conv2d_3 (Conv2D)	(None, 24, 24, 32)	4640
max_pooling2d_1 (MaxPooling2D)	(None, 12, 12, 32)	0
conv2d_4 (Conv2D)	(None, 10, 10, 64)	18496
max_pooling2d_2 (MaxPooling2D)	(None, 5, 5, 64)	0
dropout_1 (Dropout)	(None, 5, 5, 64)	0
flatten_2 (Flatten)	(None, 1600)	0
dense_6 (Dense)	(None, 128)	204928
dropout_2 (Dropout)	(None, 128)	0
dense_7 (Dense)	(None, 10)	1290

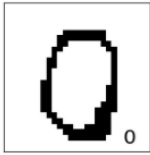
Total params: 229,514  
Trainable params: 229,514  
Non-trainable params: 0

Test loss: 0.023519557415953022  
Test accuracy: 0.9921000003814697  
Computation time:287.099 sec



# ※ 作成したモデルを用いて、自作の手書き文字を認識させてみた結果

手書き文字認識モデル3 (DCNN,relu)  
 filePath=HandwrittenImages/Handwritten\_0.bmp, label=0  
 predData.dtype=float32, shape=(1, 10), range=2.924821842498204e-07~0.993170976638794  
 (class,probability) 0:0.99 1:0.00 2:0.00 3:0.00 4:0.00 5:0.00 6:0.00 7:0.00 8:0.00 9:0.01



filePath=HandwrittenImages/Handwritten\_1.bmp, label=1  
 predData.dtype=float32, shape=(1, 10), range=6.160284442557895e-07~0.964922308921814  
 (class,probability) 0:0.00 1:0.96 2:0.03 3:0.00 4:0.00 5:0.00 6:0.00 7:0.00 8:0.00 9:0.00



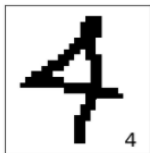
filePath=HandwrittenImages/Handwritten\_2.bmp, label=2  
 predData.dtype=float32, shape=(1, 10), range=3.2497100566075687e-09~0.9999548196792603  
 (class,probability) 0:0.00 1:0.00 2:1.00 3:0.00 4:0.00 5:0.00 6:0.00 7:0.00 8:0.00 9:0.00



filePath=HandwrittenImages/Handwritten\_3.bmp, label=3  
 predData.dtype=float32, shape=(1, 10), range=1.0905981895348305e-08~0.9997572302818298  
 (class,probability) 0:0.00 1:0.00 2:0.00 3:1.00 4:0.00 5:0.00 6:0.00 7:0.00 8:0.00 9:0.00



filePath=HandwrittenImages/Handwritten\_4.bmp, label=4  
 predData.dtype=float32, shape=(1, 10), range=2.0098941955204452e-10~0.9998717308044434  
 (class,probability) 0:0.00 1:0.00 2:0.00 3:0.00 4:1.00 5:0.00 6:0.00 7:0.00 8:0.00 9:0.00



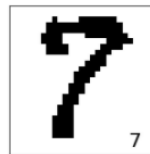
filePath=HandwrittenImages/Handwritten\_5.bmp, label=5  
 predData.dtype=float32, shape=(1, 10), range=9.98547577957254e-10~0.9985882639884949  
 (class,probability) 0:0.00 1:0.00 2:0.00 3:0.00 4:0.00 5:1.00 6:0.00 7:0.00 8:0.00 9:0.00



filePath=HandwrittenImages/Handwritten\_6.bmp, label=6  
 predData.dtype=float32, shape=(1, 10), range=1.8943477897437333e-09~0.9839308857917786  
 (class,probability) 0:0.00 1:0.00 2:0.00 3:0.01 4:0.00 5:0.00 6:0.00 7:0.00 8:0.98 9:0.00



filePath=HandwrittenImages/Handwritten\_7.bmp, label=7  
 predData.dtype=float32, shape=(1, 10), range=8.919333049561828e-05~0.6415036916732788  
 (class,probability) 0:0.00 1:0.03 2:0.28 3:0.00 4:0.00 5:0.00 6:0.00 7:0.64 8:0.05 9:0.00



filePath=HandwrittenImages/Handwritten\_8.bmp, label=8  
 predData.dtype=float32, shape=(1, 10), range=8.081899949274884e-08~0.6487123847007751  
 (class,probability) 0:0.00 1:0.00 2:0.00 3:0.85 4:0.00 5:0.10 6:0.00 7:0.00 8:0.05 9:0.00



filePath=HandwrittenImages/Handwritten\_9.bmp, label=9  
 predData.dtype=float32, shape=(1, 10), range=2.1505359715234817e-09~0.8566951155662537  
 (class,probability) 0:0.00 1:0.00 2:0.00 3:0.01 4:0.00 5:0.00 6:0.00 7:0.09 8:0.04 9:0.86



Total images=10, OK images=8, accuracy=0.80



### (3.4) 「手書き文字認識モデル4」

- ・「手書き文字認識モデル3」では、7層の「深層畳み込みニューラルネットワーク (DCN)」を用いました。  
自作の画像データの認識精度を上げるために、どんな改善を施せばよいのでしょうか？
- ・「手書き文字認識モデル4」は、「手書き文字認識モデル3」とほぼ同じですが、  
「プーリング層」で「最大値プーリング」を用いていた代わりに「平均値プーリング」を用います。
- ・モデルとその学習方法は以下のようなものです：

- (1) 入力層から出力層に向けて一方通行で流れる (Sequential)。
- (2) 入力層の入力データは、(Y方向サイズ, X方向サイズ, 入力チャネル数) = (28, 28, 1) とする。  
「入力チャネル数」はモノクロ画像なので1 (カラー画像なら3)。  
値は、0~255 の 256 階調のところを、255 で割って 0~1 に正規化したものを用いる。
- (3) 入力と出力の間に、中間層 (隠れ層) を置く。  
中間層を以下の6段で構成する。
  - (3.1) 中間層第1段は、畳み込み層で、フィルタ  $3 \times 3$  を 16枚で、Padding無しで、畳み込む。
  - (3.2) 中間層第2段は、畳み込み層で、フィルタ  $3 \times 3$  を 32枚で、Padding無しで、畳み込む。
  - (3.4) 中間層第3段は、 $2 \times 2$ の平均値プーリング層。
  - (3.4) 中間層第4段は、畳み込み層で、フィルタ  $3 \times 3$  を 64枚で、Padding無しで、畳み込む。
  - (3.5) 中間層第5段は、 $2 \times 2$ の平均値プーリング層で、ドロップアウト率を 0.25 とする。  
この出力を、Flatten層を通して一次元化し、  
一次元化したノードを、中間層第6段の各ノードとの間で全結合させる。
  - (3.6) 中間層第6段は、全結合層で、ドロップアウト率を 0.25 とする。  
各ノードを、出力層の各ノードとの間で全結合させる。
- (4) 出力層のノード数は、10 (入力画像の文字の種類 (0~9) の数) である。  
データ型は「float32 (入力画像枚数, 10)」で、入力画像の各クラスへの所属確率 [0~1] を表す。  
中間層の出力を「Flatten」層を通すことで、「 $28 \times 28 \times 8 \text{枚} \times \text{入力チャネル数}$ 」という  
4次元のデータを一次元化して、出力層との間で全結合させる。
- (5) 中間層の活性化関数として「relu (正規化線形関数, Rectified Linear Unit)」を用いる。
- (6) 出力層の活性化関数として「ソフトマックス関数 (softmax)」を用いる。
- (7) 損失関数 (誤差を計算する関数) として、「クロスエントロピー誤差 (cross entropy)」を用いる。
- (8) 学習方法 (モデルの最適化方法) として、勾配降下法の一種である「Adam」を用いる。
- (9) 学習時には「エポック数」として10、「バッチサイズ」として「1000」で学習を行う。

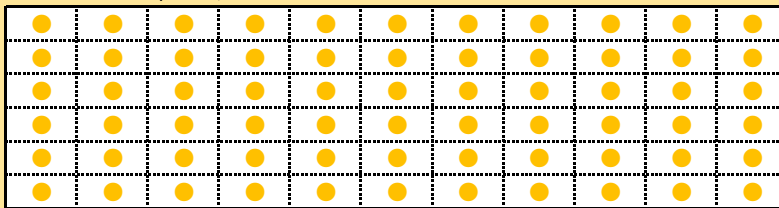
- ・MNIST を用いたモデル作成と評価、及び自作データでの認識精度

- (1) MNIST の検証用画像データの認識精度は99%と高く、  
「最大値プーリング」を用いた「手書き文字認識モデル3」とほぼ同じですが、  
自作の画像データの認識精度も、10枚中6枚、つまり60%と少し下がってしまいました。  
「最大値プーリング」の代わりに「平均値プーリング」を用いるのは、  
モデル改善の方向性として相応しくなさそうです。

- ・「手書き文字認識モデル4」を図解すると以下のように全部で7層になります：  
 (※ 以下の図を見ると9層に見えますが、慣例的に入力層とFlatten層は層の厚さとしては加えないので、7層になります。)

#### 入力層

(画像  $28 \times 28$  pixel, 1チャンネル(モノクロ画像))

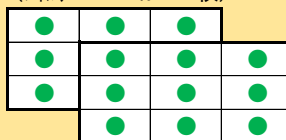


#### 中間層

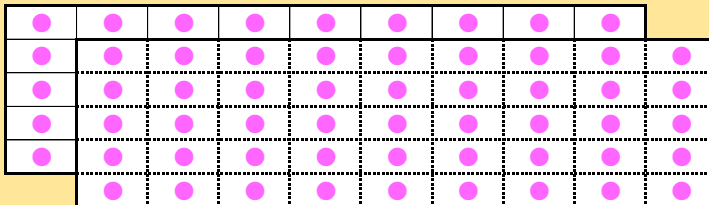
(中間層第1段)

畳み込み層

(フィルタ  $3 \times 3$  が 16枚)



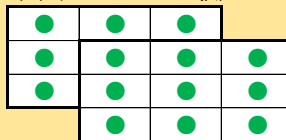
(特徴マップ : Padding無しの為  $26 \times 26 \times 16$ )



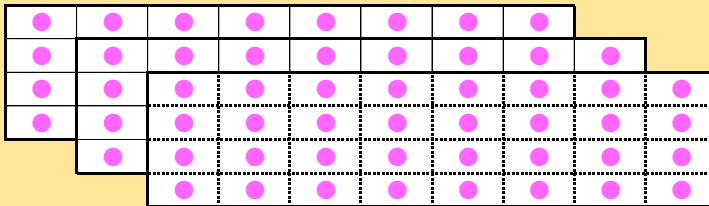
(中間層第2段)

畳み込み層

(フィルタ  $3 \times 3$  が 32枚)



(特徴マップ : Padding無しの為  $24 \times 24 \times 32$ )

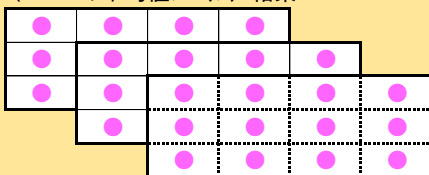


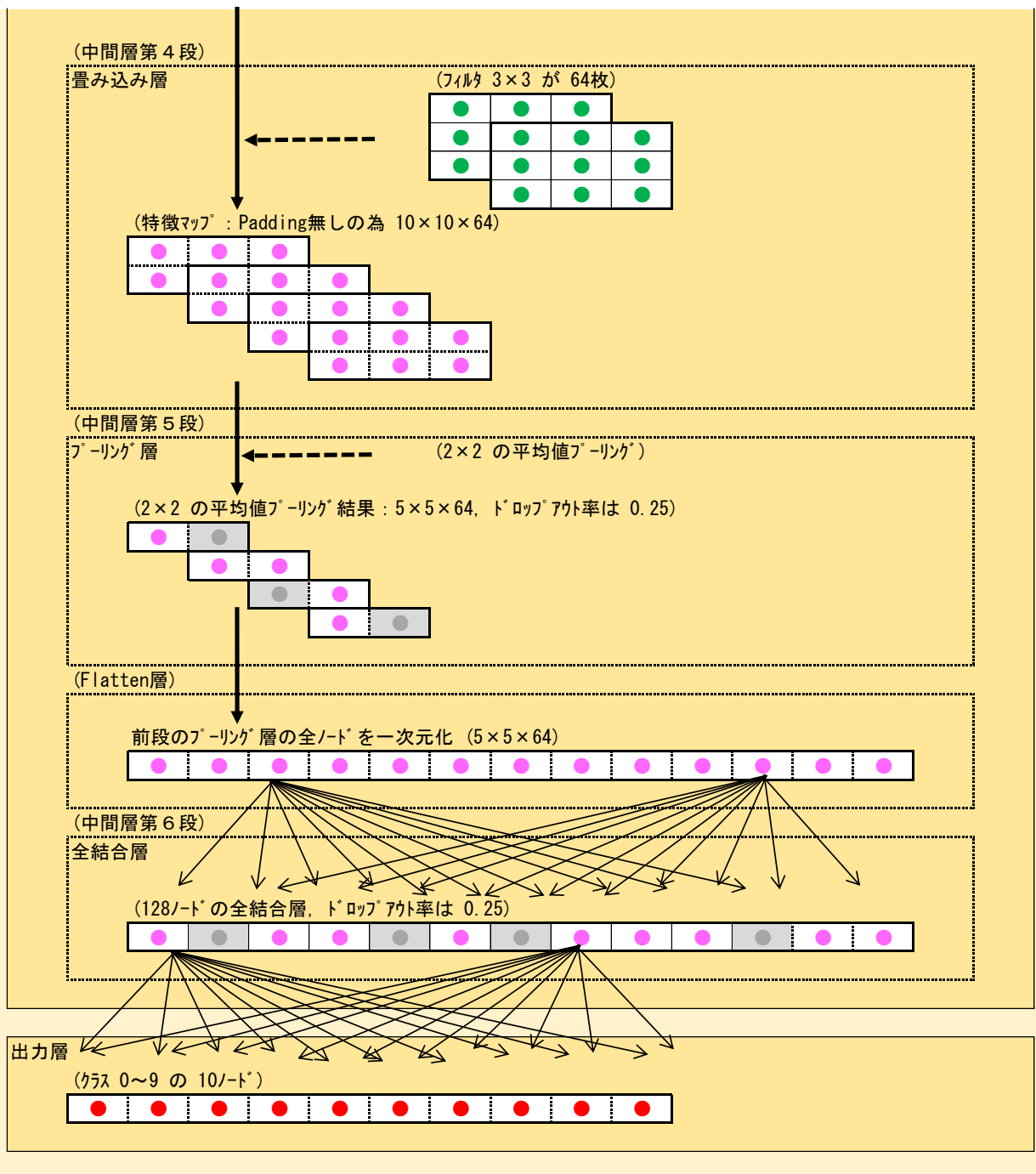
(中間層第3段)


プーリング層

( $2 \times 2$  の平均値プーリング)

( $2 \times 2$  の平均値プーリング結果 :  $12 \times 12 \times 32$ )





 : ドロップアウト対象のノードを示します

### (3.4.1) 「手書き文字認識モデル4」の実装

- ・以下に、本モデルの実装を示します。

(リスト07-(03)-4\_手書き文字認識モデル4)

```
#####/
# リスト07-(03)-4_手書き文字認識モデル4
#-----
# 「手書き文字認識モデル4」
# 深層畳み込みニューラルネットワーク (DCNN)・・・平均値プーリング使用
#####/
# 参照ライブラリ
import numpy as np
np.random.seed(1)
from keras.models import Sequential
from keras.layers import Dense, Conv2D, AveragePooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.optimizers import Adam
import time

#####/
# reshape_for_model4 */
# ロットした MNIST データを「手書き文字認識モデル4」 */
# 用に変形したものを返す */
#----- */
# 引数: */
# in_image : 入力画像: uint8 (入力画像枚数, 28, 28) */
# in_label : 入力画像が表す数値 (0~9) : uint8 (入力画像枚数) */
#----- */
# 戻り値: */
# (1) in_image を */
# 「3次元(入力画像枚数, 28, 28)、uint8(0~255)」から */
# 「4次元(入力画像枚数, 28, 28, 1)、float32(0.0~1.0)」 */
# へ正規化したもの */
# (2) in_label を */
# 「1次元(入力画像枚数)、uint8(0~8: 入力画像が表す数値)」から */
# 「1-of-k 符号化」で 10クラスへ分類し「2次元(入力画像枚数, 10)」 */
# 各クラスへの所属確率を float32(0.0~1.0) で表現したもの */
# へ変換したもの */
#####/
def reshape_for_model4(in_image, in_label):
    return reshape_for_model2(in_image, in_label)

#####/
# make_and_evaluate_model4 */
# 「手書き文字認識モデル4」を作成してそれを評価する */
# 隠れ層の活性化関数は引数で指定可能 */
#----- */
# 「手書き文字認識モデル4」 */
# 深層畳み込みニューラルネットワーク (DCNN) */
#----- */
# (1) 入力層から出力層に向けて一方通行で流れる (Sequential)。 */
# (2) 入力層の入力データは、 */
# (Y方向サイズ, X方向サイズ, 入力チャネル数) = (28, 28, 1) とする。 */
# 「入力チャネル数」はモノクロ画像なので1(カラー画像なら3)。 */
# 値は、0~255 の 256 階調のところを、 */
# 255 で割って 0~1 に正規化したものを用いる。 */
#----- */
# (3) 入力と出力の間に、中間層(隠れ層)を置く。 */
# 中間層を以下の6段で構成する。 */
# (3.1) 中間層第1段は、畳み込み層で、 */
# フィルタ 3×3 を 16枚で、Padding無しで、畳み込む。 */
# (3.2) 中間層第2段は、畳み込み層で、 */
```

```

#       フィルタ 3×3 を 32枚で、Padding無しで、畳み込む。          */
#   (3.3) 中間層第3段は、2×2の平均値プーリング層。                */
#   (3.4) 中間層第4段は、畳み込み層で、                            */
#       フィルタ 3×3 を 64枚で、Padding無しで、畳み込む。          */
#   (3.5) 中間層第5段は、2×2の平均値プーリング層で、                */
#       ドロップアウト率を 0.25 とする。                            */
#       この出力を、Flatten層を通して一次元化する。                */
#       一次元化したノードを、中間層第6段の各ノードとの間で全結合する。*/
#   (3.6) 中間層第6段は、全結合層で、ドロップアウト率を 0.25 とする。 */
#       各ノードを、出力層の各ノードとの間で全結合させる。        */
#                                                                    */
# (4) 出力層のノード数は、10（入力画像の文字の種類(0～9)の数)である。 */
#       データ型は「float32（入力画像枚数,10)」で、                */
#       入力画像の各クラスへの所属確率[0～1]を表す。                */
#       中間層の出力を「Flatten」層を通すことで、                  */
#       「28×28×8枚×入力チャネル数」という                        */
#       4次元のデータを一次元化して、出力層との間で全結合させる。  */
#                                                                    */
# (5) 中間層の活性化関数は「relu」                                */
# (6) 出力層の活性化関数は「softmax」                              */
# (7) 損失関数(誤差を計算する関数)は「categorical_crossentropy」  */
# (8) 学習方法(モデルの最適化方法)は「Adam」                      */
# (9) 学習時には「エポック数」として10、                          */
#       「バッチサイズ」として「1000」で学習を行う。              */
#-----*/
# 引数：                                                            */
#   title      : 表示用タイトル                                    */
#   actvFunc    : 隠れ層の活性化関数                              */
#-----*/
# 戻り値：                                                            */
#   (1) 作成したモデル                                            */
#-----*/
def make_and_evaluate_model4(title, actvFunc):

    print("*****")
    print(title)
    print("*****")

    #-----
    # ロードしたMNIST データを「手書き文字認識モデル4」用に変形する。
    #-----
    inTrain4, outTrain4 = reshape_for_model4(x_train, y_train)
    inTest4, outTest4 = reshape_for_model4(x_test, y_test)

    titlestr = ["modified ...", "inTrain4", "outTrain4", "inTest4", "outTest4"]
    print_MNIST_attr(titlestr, inTrain4, outTrain4, inTest4, outTest4)

    #-----
    # 「手書き文字認識モデル4」を作成
    #-----
    np.random.seed(1)

    predModel = Sequential()
    predModel.add(Conv2D(16, (3, 3),
        input_shape=(IMG_SIZE_Y, IMG_SIZE_X, 1), activation=actvFunc))
    predModel.add(Conv2D(32, (3, 3), activation=actvFunc))
    predModel.add(AveragePooling2D(pool_size=(2, 2)))
    predModel.add(Conv2D(64, (3, 3), activation=actvFunc))
    predModel.add(AveragePooling2D(pool_size=(2, 2)))
    predModel.add(Dropout(0.25))
    predModel.add(Flatten())
    predModel.add(Dense(128, activation=actvFunc))

```

```

predModel.add(Dropout(0.25))
predModel.add(Dense(IMG_CLASS_NO, activation='softmax'))
predModel.compile(loss='categorical_crossentropy',
                  optimizer=Adam(), metrics=['accuracy'])

predModel.summary()

#=====
# 「手書き文字認識モデル4」を学習し、
# 学習状況を、エポック回数の時系列でグラフ表示する
#=====
epochNo = 10
batchSize = 1000

startTime = time.time()
history = predModel.fit(inTrain4, outTrain4,
                       epochs=epochNo, batch_size=batchSize,
                       verbose=0, validation_data=(inTest4, outTest4))
score = predModel.evaluate(inTest4, outTest4, verbose=0)
endTime = time.time()

print('Test loss:', score[0])
print('Test accuracy:', score[1])
print("Computation time: {0:.3f} sec".format(time.time() - startTime))

plot_learning_curve(history, epochNo)

#=====
# 「手書き文字認識モデル4」で、
# MINTSTの検証用データの最初の50枚で分類結果を表示する。
#=====
pred_test_and_show_result(predModel, inTest4, y_test, IMG_SIZE_X, IMG_SIZE_Y, 50)

return predModel

#-----/
# 「手書き文字認識モデル4」で
# 活性化関数を「relu」でモデルを作成して評価する。
#-----/
predModel4 = make_and_evaluate_model4(
    "「深層畳み込みニューラルネットワーク (DCNN)」で活性化関数は「relu」", 'relu')

#-----/
# 「手書き文字認識モデル4」で、
# 自作の画像の分類結果を表示する。
#-----/
pred_myData_and_show_result("手書き文字認識モデル4 (DCNN, relu)", predModel4, reshape_for_model4)

pred_myData_and_show_result("手書き文字認識モデル4 (DCNN, relu)", predModel4, reshape_for_model4)

```

### (3.4.2) 「手書き文字認識モデル4」の実行結果

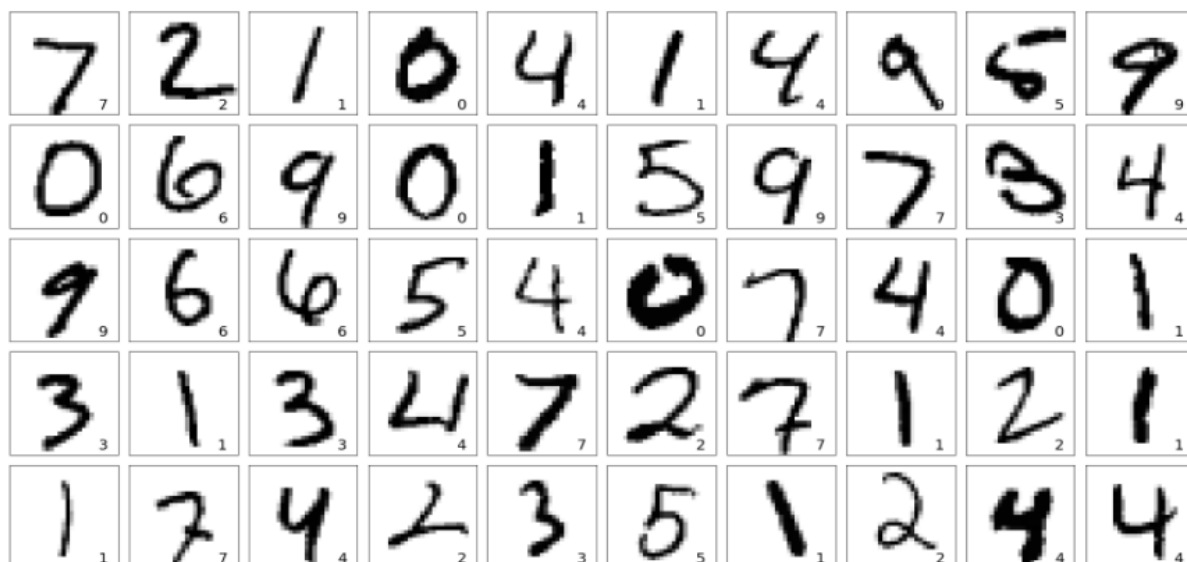
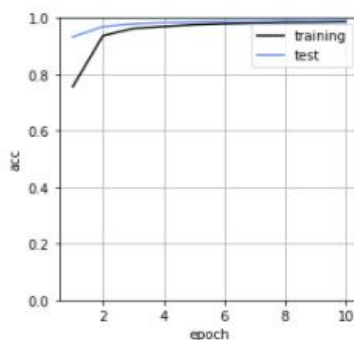
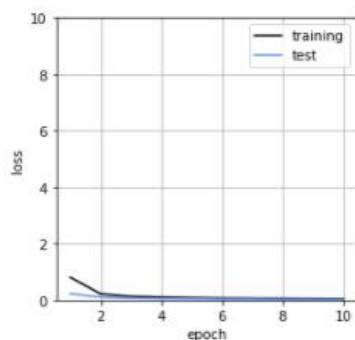
```
*****
「深層畳み込みニューラルネットワーク (DCNN)」で活性化関数は「relu」
*****
modified ...
inTrain4.dtype=float32
inTrain4.shape=(60000, 28, 28, 1)
inTrain4.range=0.0~1.0

outTrain4.dtype=float32
outTrain4.shape=(60000, 10)
outTrain4.range=0.0~1.0

inTest4.shape=(10000, 28, 28, 1)
outTest4.shape=(10000, 10)
Model: "sequential_24"
```

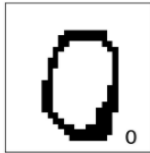
Layer (type)	Output Shape	Param #
conv2d_5 (Conv2D)	(None, 26, 26, 16)	160
conv2d_6 (Conv2D)	(None, 24, 24, 32)	4640
average_pooling2d_1 (Average)	(None, 12, 12, 32)	0
conv2d_7 (Conv2D)	(None, 10, 10, 64)	18496
average_pooling2d_2 (Average)	(None, 5, 5, 64)	0
dropout_3 (Dropout)	(None, 5, 5, 64)	0
flatten_3 (Flatten)	(None, 1600)	0
dense_46 (Dense)	(None, 128)	204928
dropout_4 (Dropout)	(None, 128)	0
dense_47 (Dense)	(None, 10)	1290
Total params: 229,514		
Trainable params: 229,514		
Non-trainable params: 0		

Test loss: 0.029092814877245108  
Test accuracy: 0.9907000064849854  
Computation time:273.368 sec

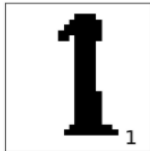


## ※ 作成したモデルを用いて、自作の手書き文字を認識させてみた結果

手書き文字認識モデル4 (DCNN,relu)  
 filePath=HandwrittenImages/Handwritten\_0.bmp, label=0  
 predData.dtype=float32, shape=(1, 10), range=4.017307375647761e-09~0.9990229606828418  
 (class,probability) 0:1.00 1:0.00 2:0.00 3:0.00 4:0.00 5:0.00 6:0.00 7:0.00 8:0.00 9:0.00



filePath=HandwrittenImages/Handwritten\_1.bmp, label=1  
 predData.dtype=float32, shape=(1, 10), range=2.3133696347432675e-10~0.9993512034416199  
 (class,probability) 0:0.00 1:1.00 2:0.00 3:0.00 4:0.00 5:0.00 6:0.00 7:0.00 8:0.00 9:0.00



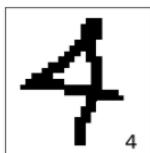
filePath=HandwrittenImages/Handwritten\_2.bmp, label=2  
 predData.dtype=float32, shape=(1, 10), range=4.071908825745152e-13~0.9999837875366211  
 (class,probability) 0:0.00 1:0.00 2:1.00 3:0.00 4:0.00 5:0.00 6:0.00 7:0.00 8:0.00 9:0.00



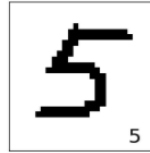
filePath=HandwrittenImages/Handwritten\_3.bmp, label=3  
 predData.dtype=float32, shape=(1, 10), range=3.235424514969054e-07~0.9889208078384399  
 (class,probability) 0:0.00 1:0.00 2:0.00 3:0.99 4:0.00 5:0.00 6:0.00 7:0.00 8:0.00 9:0.00



filePath=HandwrittenImages/Handwritten\_4.bmp, label=4  
 predData.dtype=float32, shape=(1, 10), range=6.94369095552029e-09~0.9995637536048889  
 (class,probability) 0:0.00 1:0.00 2:0.00 3:0.00 4:1.00 5:0.00 6:0.00 7:0.00 8:0.00 9:0.00



filePath=HandwrittenImages/Handwritten\_5.bmp, label=5  
 predData.dtype=float32, shape=(1, 10), range=1.3379673757807353e-10~0.9985055923461914  
 (class,probability) 0:0.00 1:0.00 2:0.00 3:0.00 4:0.00 5:1.00 6:0.00 7:0.00 8:0.00 9:0.00



filePath=HandwrittenImages/Handwritten\_6.bmp, label=6  
 predData.dtype=float32, shape=(1, 10), range=2.8234150573069883e-09~0.9364177584648132  
 (class,probability) 0:0.00 1:0.00 2:0.00 3:0.94 4:0.00 5:0.00 6:0.00 7:0.01 8:0.05 9:0.00



filePath=HandwrittenImages/Handwritten\_7.bmp, label=7  
 predData.dtype=float32, shape=(1, 10), range=2.255153042085567e-08~0.6032914519309998  
 (class,probability) 0:0.00 1:0.02 2:0.60 3:0.00 4:0.00 5:0.00 6:0.00 7:0.02 8:0.35 9:0.00



filePath=HandwrittenImages/Handwritten\_8.bmp, label=8  
 predData.dtype=float32, shape=(1, 10), range=1.4640906542684462e-10~0.999971866607666  
 (class,probability) 0:0.00 1:0.00 2:0.00 3:1.00 4:0.00 5:0.00 6:0.00 7:0.00 8:0.00 9:0.00



filePath=HandwrittenImages/Handwritten\_9.bmp, label=9  
 predData.dtype=float32, shape=(1, 10), range=1.1870400662417069e-08~0.4317936897277832  
 (class,probability) 0:0.00 1:0.00 2:0.00 3:0.01 4:0.00 5:0.00 6:0.00 7:0.43 8:0.41 9:0.15



Total images=10, OK images=6, accuracy=0.60



### (3.5) 「手書き文字認識モデル5」

- ・「手書き文字認識モデル4」では、「手書き文字認識モデル3」で、「プーリング層」で「最大値プーリング」を用いる代わりに「平均値プーリング」を用いてみましたが、改善になっていませんでした。
- ・「手書き文字認識モデル5」では「手書き文字認識モデル3」の改善策として、「中間層第5段」の直後に「畳み込み層」と「プーリング層」を追加して、層を少し深くしてみます。
- ・モデルとその学習方法は以下のようになります：

- (1) 入力層から出力層に向けて一方通行で流れる (Sequential)。
- (2) 入力層の入力データは、(Y方向サイズ, X方向サイズ, 入力チャネル数) = (28, 28, 1) とする。  
「入力チャネル数」はモノクロ画像なので1(カラー画像なら3)。  
値は、0~255 の 256 階調のところを、255 で割って 0~1 に正規化したものを用いる。
- (3) 入力と出力の間に、中間層(隠れ層)を置く。  
中間層を以下の6段で構成する。
  - (3.1) 中間層第1段は、畳み込み層で、フィルタ  $3 \times 3$  を 16枚で、Padding無しで、畳み込む。
  - (3.2) 中間層第2段は、畳み込み層で、フィルタ  $3 \times 3$  を 32枚で、Padding無しで、畳み込む。
  - (3.4) 中間層第3段は、 $2 \times 2$ の最大値プーリング層。
  - (3.4) 中間層第4段は、畳み込み層で、フィルタ  $3 \times 3$  を 64枚で、Padding無しで、畳み込む。
  - (3.5) 中間層第5段は、 $2 \times 2$ の最大値プーリング層で、ドロップアウト率を 0.25 とする。
  - (3.6) 中間層第6段は、畳み込み層で、フィルタ  $3 \times 3$  を 64枚で、Padding無しで、畳み込む。
  - (3.7) 中間層第7段は、 $2 \times 2$ の最大値プーリング層で、ドロップアウト指定はしない。  
この出力を、Flatten層を通して一次元化し、  
一次元化したノードを、中間層第6段の各ノードとの間で全結合させる。
- (3.8) 中間層第8段は、全結合層で、ドロップアウト率を 0.25 とする。  
各ノードを、出力層の各ノードとの間で全結合させる。
- (4) 出力層のノード数は、10 (入力画像の文字の種類(0~9)の数)である。  
データ型は「float32 (入力画像枚数, 10)」で、入力画像の各クラスへの所属確率[0~1]を表す。  
中間層の出力を「Flatten」層を通すことで、「 $28 \times 28 \times 8 \text{枚} \times \text{入力チャネル数}$ 」という  
4次元のデータを一次元化して、出力層との間で全結合させる。
- (5) 中間層の活性化関数として「relu (正規化線形関数, Rectified Linear Unit)」を用いる。
- (6) 出力層の活性化関数として「ソフトマックス関数 (softmax)」を用いる。
- (7) 損失関数(誤差を計算する関数)として、「クロスエントロピー誤差 (cross entropy)」を用いる。
- (8) 学習方法(モデルの最適化方法)として、勾配降下法的一种である「Adam」を用いる。
- (9) 学習時には「エポック数」として10、「バッチサイズ」として「1000」で学習を行う。

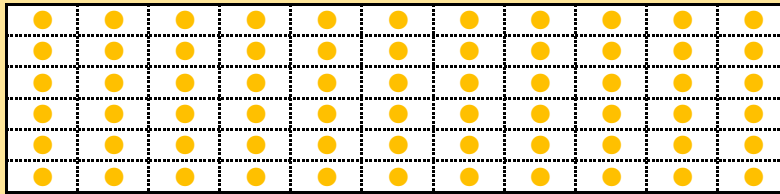
- ・MNIST を用いたモデル作成と評価、及び自作データでの認識精度

- (1) MNIST の検証用画像データの認識精度は99%より少し低くなりましたが、「最大値プーリング」を用いた「手書き文字認識モデル3」とほぼ同じで、自作の画像データの認識精度は、10枚中7枚、つまり70%でした。  
単純に層を深くすれば良いというわけでもないことが分かります。

- ・「手書き文字認識でる5」を図解すると以下のように全部で9層になります：  
 (※ 以下の図を見ると11層に見えますが、慣例的に入力層とFlatten層は層の厚さとしては加えないので、9層になります。)

#### 入力層

(画像 28×28 pixel, 1チャンネル(モノクロ画像))

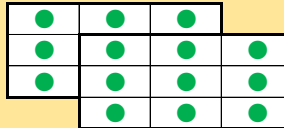


#### 中間層

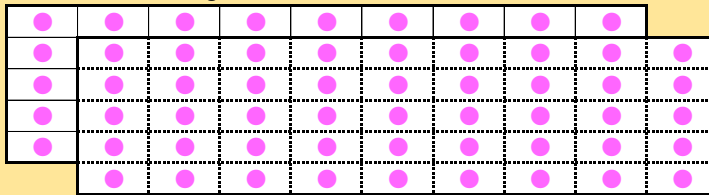
(中間層第1段)

畳み込み層

(フィルタ 3×3 が 16枚)



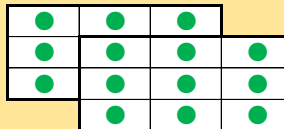
(特徴マップ : Padding無しの為 26×26×16)



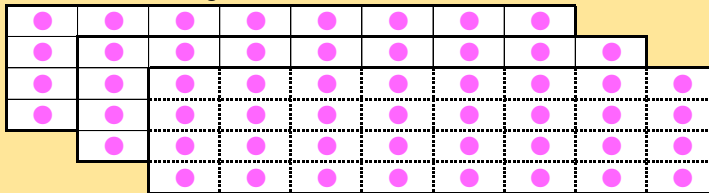
(中間層第2段)

畳み込み層

(フィルタ 3×3 が 32枚)



(特徴マップ : Padding無しの為 24×24×32)

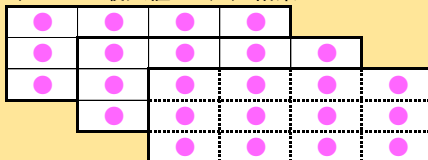


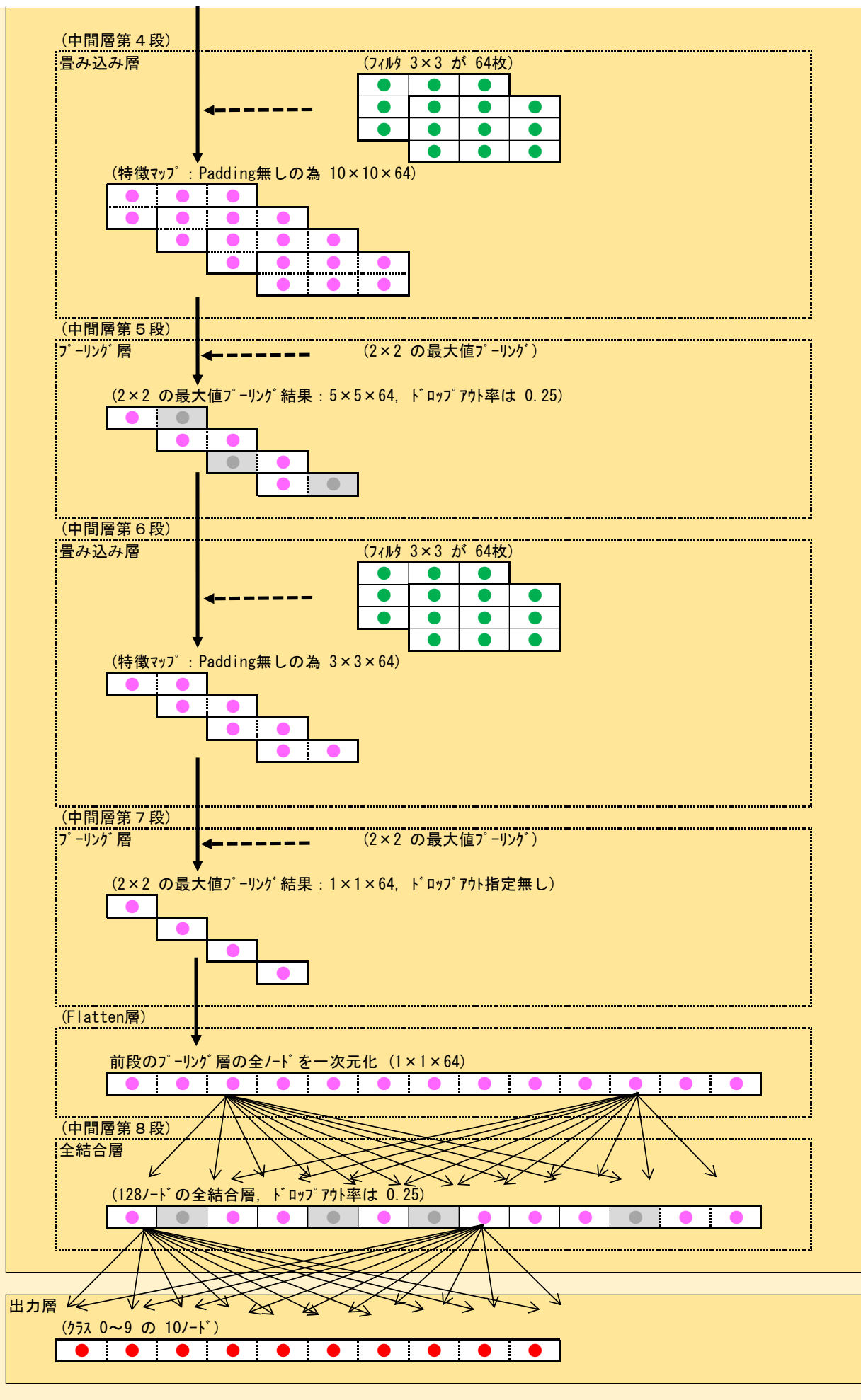
(中間層第3段)

プーリング層

(2×2 の最大値プーリング)

(2×2 の最大値プーリング結果 : 12×12×32)





: ドロップアウト対象のノードを示します

### (3.5.1) 「手書き文字認識モデル5」の実装

- ・以下に、本モデルの実装を示します。

( リスト07-(03)-5\_手書き文字認識モデル5 )

```
#####/
# リスト07-(03)-5_手書き文字認識モデル5
#-----
# 「手書き文字認識モデル5」
# 深層畳み込みニューラルネットワーク (DCNN) . . . 9 層
#####/
# 参照コード
import numpy as np
np.random.seed(1)
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.optimizers import Adam
import time

#####/
# reshape_for_model5 */
# ロットした MNIST データを「手書き文字認識モデル5」 */
# 用に変形したものを返す */
#----- */
# 引数 : */
#   in_image : 入力画像 : uint8 (入力画像枚数, 28, 28) */
#   in_label : 入力画像が表す数値 (0~9) : uint8 (入力画像枚数) */
#----- */
# 戻り値 : */
# (1) in_image を */
#   「3次元(入力画像枚数, 28, 28)、uint8(0~255)」から */
#   「4次元(入力画像枚数, 28, 28, 1)、float32(0.0~1.0)」 */
#   へ正規化したもの */
# (2) in_label を */
#   「1次元(入力画像枚数)、uint8(0~8: 入力画像が表す数値)」から */
#   「1-of-k 符号化」で 10クラスへ分類し「2次元(入力画像枚数, 10)」 */
#   各クラスへの所属確率を float32(0.0~1.0) で表現したもの */
#   へ変換したもの */
#####/
def reshape_for_model5(in_image, in_label):
    return reshape_for_model2(in_image, in_label)

#####/
# make_and_evaluate_model5 */
# 「手書き文字認識モデル5」を作成してそれを評価する */
# 隠れ層の活性化関数は引数で指定可能 */
#----- */
# 「手書き文字認識モデル5」 */
# 深層畳み込みニューラルネットワーク (DCNN) */
#----- */
# (1) 入力層から出力層に向けて一方通行で流れる (Sequential)。 */
# (2) 入力層の入力データは、 */
#   (Y方向サイズ, X方向サイズ, 入力チャネル数) = (28, 28, 1) とする。 */
#   「入力チャネル数」はモノクロ画像なので1(カラー画像なら3)。 */
#   値は、0~255 の 256 階調のところを、 */
#   255 で割って 0~1 に正規化したものを用いる。 */
#----- */
# (3) 入力と出力の間に、中間層 (隠れ層) を置く。 */
#   中間層を以下の6段で構成する。 */
# (3.1) 中間層第1段は、畳み込み層で、 */
```

```

# フィルタ 3×3 を 16枚で、Padding無しで、畳み込む。 */
# (3.2) 中間層第2段は、畳み込み層で、 */
# フィルタ 3×3 を 32枚で、Padding無しで、畳み込む。 */
# (3.3) 中間層第3段は、2×2の平均値プーリング層。 */
# (3.4) 中間層第4段は、畳み込み層で、 */
# フィルタ 3×3 を 64枚で、Padding無しで、畳み込む。 */
# (3.5) 中間層第5段は、2×2の平均値プーリング層で、 */
# ドロップアウト率を 0.25 とする。 */
# (3.6) 中間層第6段は、畳み込み層で、 */
# フィルタ 3×3 を 64枚で、Padding無しで、畳み込む。 */
# (3.7) 中間層第7段は、2×2の最大値プーリング層で、 */
# ドロップアウト指定はしない。 */
# この出力を、Flatten層を通して一次元化する。 */
# 一次元化したノードを、中間層第6段の各ノードとの間で全結合する。 */
# (3.8) 中間層第8段は、全結合層で、ドロップアウト率を 0.25 とする。 */
# 各ノードを、出力層の各ノードとの間で全結合させる。 */
# */
# (4) 出力層のノード数は、10（入力画像の文字の種類(0～9)の数)である。 */
# データ型は「float32（入力画像枚数,10)」で、 */
# 入力画像の各クラスへの所属確率[0～1]を表す。 */
# 中間層の出力を「Flatten」層を通すことで、 */
# 「28×28×8枚×入力チャネル数」という */
# 4次元のデータを一次元化して、出力層との間で全結合させる。 */
# */
# (5) 中間層の活性化関数は「relu」 */
# (6) 出力層の活性化関数は「softmax」 */
# (7) 損失関数(誤差を計算する関数)は「categorical_crossentropy」 */
# (8) 学習方法(モデルの最適化方法)は「Adam」 */
# (9) 学習時には「エポック数」として10、 */
# 「バッチサイズ」として「1000」で学習を行う。 */
#----- */
# 引数： */
# title : 表示用タイトル */
# actvFunc : 隠れ層の活性化関数 */
#----- */
# 戻り値： */
# (1) 作成したモデル */
#----- */
def make_and_evaluate_model5(title, actvFunc):

    print("*****")
    print(title)
    print("*****")

    #-----
    # ロットしたMNIST データを「手書き文字認識モデル5」用に変形する。
    #-----
    inTrain, outTrain = reshape_for_model5(x_train, y_train)
    inTest, outTest = reshape_for_model5(x_test, y_test)

    titlestr = ["modified ...", "inTrain", "outTrain", "inTest", "outTest"]
    print_MNIST_attr(titlestr, inTrain, outTrain, inTest, outTest)

    #-----
    # 「手書き文字認識モデル5」を作成
    #-----
    np.random.seed(1)

    predModel = Sequential()
    predModel.add(Conv2D(16, (3, 3),
        input_shape=(IMG_SIZE_Y, IMG_SIZE_X, 1), activation=actvFunc))
    predModel.add(Conv2D(32, (3, 3), activation=actvFunc))

```

```

predModel.add(MaxPooling2D(pool_size=(2, 2)))
predModel.add(Conv2D(64, (3, 3), activation=actvFunc))
predModel.add(MaxPooling2D(pool_size=(2, 2)))
predModel.add(Dropout(0.25))
predModel.add(Conv2D(64, (3, 3), activation=actvFunc))
predModel.add(MaxPooling2D(pool_size=(2, 2)))
predModel.add(Flatten())
predModel.add(Dense(128, activation=actvFunc))
predModel.add(Dropout(0.25))
predModel.add(Dense(IMG_CLASS_NO, activation='softmax'))
predModel.compile(loss='categorical_crossentropy',
                  optimizer=Adam(), metrics=['accuracy'])

predModel.summary()

#=====
# 「手書き文字認識モデル5」を学習し、
# 学習状況を、エポック回数の時系列でグラフ表示する
#=====
epochNo = 10
batchSize = 1000

startTime = time.time()
history = predModel.fit(inTrain, outTrain,
                        epochs=epochNo, batch_size=batchSize,
                        verbose=0, validation_data=(inTest, outTest))
score = predModel.evaluate(inTest, outTest, verbose=0)
endTime = time.time()

print('Test loss:', score[0])
print('Test accuracy:', score[1])
print("Computation time:{0:.3f} sec".format(time.time() - startTime))

plot_learning_curve(history, epochNo)

#=====
# 「手書き文字認識モデル5」で、
# MINTSTの検証用データの最初の50枚で分類結果を表示する。
#=====
pred_test_and_show_result(predModel, inTest, y_test, IMG_SIZE_X, IMG_SIZE_Y, 50)

return predModel

#*****/
# 「手書き文字認識モデル5」で
# 活性化関数を「relu」でモデルを作成して評価する
#*****/
predModel5 = make_and_evaluate_model5(
    "「深層畳み込みニューラルネットワーク (DCNN)」で活性化関数は「relu」", 'relu')

#*****/
# 「手書き文字認識モデル5」で、
# 自作の画像の分類結果を表示する。
#*****/
pred_myData_and_show_result("手書き文字認識モデル5 (DCNN, relu)",
                             predModel5, reshape_for_model5)

```

### (3.5.2) 「手書き文字認識mnist5」の実行結果

```
*****
「深層畳み込みニューラルネットワーク (DCNN)」で活性化関数は「relu」
*****
modified ...
inTrain.dtype=float32
inTrain.shape=(60000, 28, 28, 1)
inTrain.range=0.0~1.0

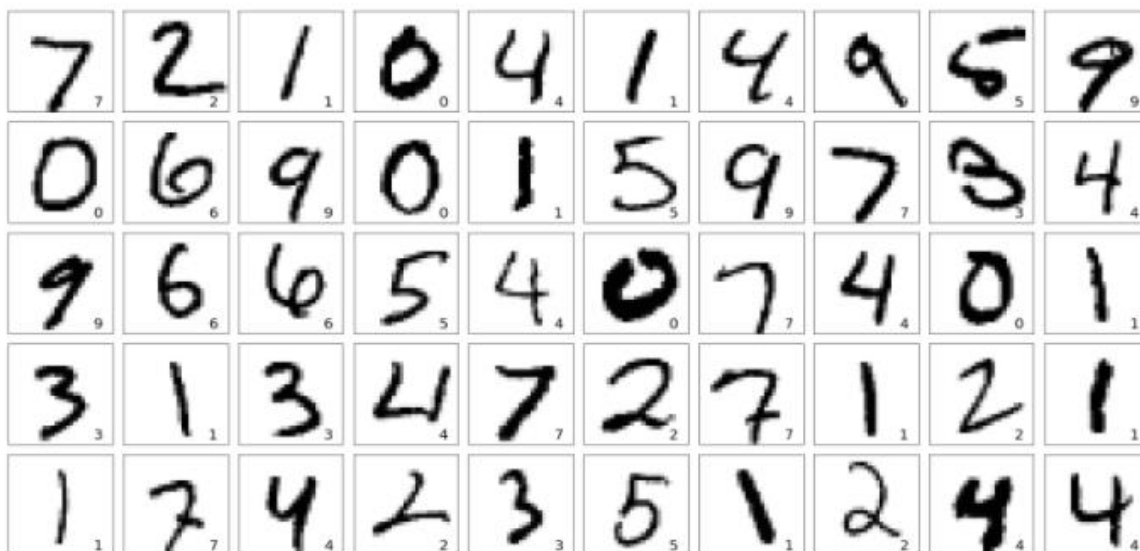
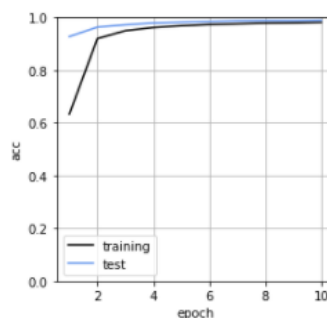
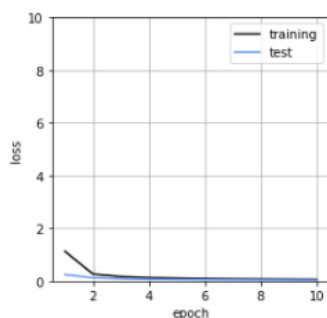
outTrain.dtype=float32
outTrain.shape=(60000, 10)
outTrain.range=0.0~1.0

inTest.shape=(10000, 28, 28, 1)
outTest.shape=(10000, 10)
Model: "sequential_6"
```

Layer (type)	Output Shape	Param #
conv2d_8 (Conv2D)	(None, 26, 26, 16)	160
conv2d_9 (Conv2D)	(None, 24, 24, 32)	4640
max_pooling2d_3 (MaxPooling2D)	(None, 12, 12, 32)	0
conv2d_10 (Conv2D)	(None, 10, 10, 64)	18496
max_pooling2d_4 (MaxPooling2D)	(None, 5, 5, 64)	0
dropout_5 (Dropout)	(None, 5, 5, 64)	0
conv2d_11 (Conv2D)	(None, 3, 3, 64)	36928
max_pooling2d_5 (MaxPooling2D)	(None, 1, 1, 64)	0
flatten_4 (Flatten)	(None, 64)	0
dense_10 (Dense)	(None, 128)	8320
dropout_6 (Dropout)	(None, 128)	0
dense_11 (Dense)	(None, 10)	1290

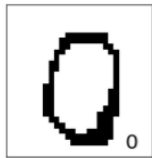
```
=====
Total params: 69,834
Trainable params: 69,834
Non-trainable params: 0

Test loss: 0.037224771117488856
Test accuracy: 0.989300012588501
Computation time:289.468 sec
```

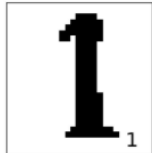


## ※ 作成したモデルを用いて、自作の手書き文字を認識させてみた結果

手書き文字認識モデル5 (DCNN,relu)  
 filePath=HandwrittenImages/Handwritten\_0.bmp, label=0  
 predData.dtype=float32, shape=(1, 10), range=2.802982024263656e-08~0.99992  
 23947525024  
 (class,probability) 0:1.00 1:0.00 2:0.00 3:0.00 4:0.00 5:0.00 6:0.00  
 7:0.00 8:0.00 9:0.00



filePath=HandwrittenImages/Handwritten\_1.bmp, label=1  
 predData.dtype=float32, shape=(1, 10), range=6.628877713410475e-07~0.99884  
 25970077515  
 (class,probability) 0:0.00 1:1.00 2:0.00 3:0.00 4:0.00 5:0.00 6:0.00  
 7:0.00 8:0.00 9:0.00



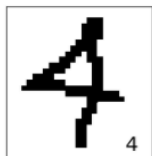
filePath=HandwrittenImages/Handwritten\_2.bmp, label=2  
 predData.dtype=float32, shape=(1, 10), range=1.0927159621587634e-07~0.9993  
 336796760559  
 (class,probability) 0:0.00 1:0.00 2:1.00 3:0.00 4:0.00 5:0.00 6:0.00  
 7:0.00 8:0.00 9:0.00



filePath=HandwrittenImages/Handwritten\_3.bmp, label=3  
 predData.dtype=float32, shape=(1, 10), range=6.057663485409259e-12~0.99993  
 10970306396  
 (class,probability) 0:0.00 1:0.00 2:0.00 3:1.00 4:0.00 5:0.00 6:0.00  
 7:0.00 8:0.00 9:0.00



filePath=HandwrittenImages/Handwritten\_4.bmp, label=4  
 predData.dtype=float32, shape=(1, 10), range=7.16502910336203e-08~0.989067  
 0776367188  
 (class,probability) 0:0.00 1:0.00 2:0.00 3:0.00 4:0.99 5:0.00 6:0.00  
 7:0.00 8:0.01 9:0.00



filePath=HandwrittenImages/Handwritten\_5.bmp, label=5  
 predData.dtype=float32, shape=(1, 10), range=3.738342968517827e-11~0.99999  
 49932098389  
 (class,probability) 0:0.00 1:0.00 2:0.00 3:0.00 4:0.00 5:1.00 6:0.00  
 7:0.00 8:0.00 9:0.00



filePath=HandwrittenImages/Handwritten\_6.bmp, label=6  
 predData.dtype=float32, shape=(1, 10), range=8.903482012101449e-06~0.73891  
 46089553833  
 (class,probability) 0:0.04 1:0.00 2:0.74 3:0.00 4:0.00 5:0.06 6:0.00  
 7:0.00 8:0.15 9:0.00



filePath=HandwrittenImages/Handwritten\_7.bmp, label=7  
 predData.dtype=float32, shape=(1, 10), range=6.161718374642078e-06~0.74904  
 06632423401  
 (class,probability) 0:0.00 1:0.06 2:0.75 3:0.00 4:0.00 5:0.00 6:0.00  
 7:0.00 8:0.18 9:0.00



filePath=HandwrittenImages/Handwritten\_8.bmp, label=8  
 predData.dtype=float32, shape=(1, 10), range=2.5377393519931957e-09~0.9995  
 728135108948  
 (class,probability) 0:0.00 1:0.00 2:0.00 3:0.00 4:0.00 5:0.00 6:0.00  
 7:0.00 8:1.00 9:0.00



filePath=HandwrittenImages/Handwritten\_9.bmp, label=9  
 predData.dtype=float32, shape=(1, 10), range=7.561167247871461e-10~0.89348  
 84071350098  
 (class,probability) 0:0.00 1:0.00 2:0.00 3:0.00 4:0.00 5:0.00 6:0.00  
 7:0.00 8:0.89 9:0.10



Total images=10, OK images=7, accuracy=0.70



### (3.6) 「手書き文字認識モデル6」

- ・「手書き文字認識モデル5」では、「手書き文字認識モデル3」より層を増やしましたが、自作の画像データの認識精度は70%でした。  
一方で、MNIST の検証用画像データの認識精度が、僅かに下がっているのに気づかれた方もいらっしゃるかと思います。
- ・「手書き文字認識モデル6」では「手書き文字認識モデル3」の改善策として、改善策として「プーリング層」のサイズを、 $2 \times 2$  から  $3 \times 3$  へと少し大きくしてみます。
- ・モデルとその学習方法は以下のようなものです：

- (1) 入力層から出力層に向けて一方通行で流れる (Sequential)。
- (2) 入力層の入力データは、(Y方向サイズ, X方向サイズ, 入力チャネル数) = (28, 28, 1) とする。  
「入力チャネル数」はモノカラー画像なので1 (カラー画像なら3)。  
値は、0~255 の 256 階調のところを、255 で割って 0~1 に正規化したものを用いる。
- (3) 入力と出力の間に、中間層 (隠れ層) を置く。  
中間層を以下の6段で構成する。
  - (3.1) 中間層第1段は、畳み込み層で、フィルタ  $3 \times 3$  を 16枚で、Padding無しで、畳み込む。
  - (3.2) 中間層第2段は、畳み込み層で、フィルタ  $3 \times 3$  を 32枚で、Padding無しで、畳み込む。
  - (3.3) 中間層第3段は、 $3 \times 3$ の最大値プーリング層。
  - (3.4) 中間層第4段は、畳み込み層で、フィルタ  $3 \times 3$  を 64枚で、Padding無しで、畳み込む。
  - (3.5) 中間層第5段は、 $3 \times 3$ の最大値プーリング層で、ドロップアウト率を 0.25 とする。  
この出力を、Flatten層を通して一次元化し、  
一次元化したノードを、中間層第6段の各ノードとの間で全結合させる。
  - (3.6) 中間層第6段は、全結合層で、ドロップアウト率を 0.25 とする。  
各ノードを、出力層の各ノードとの間で全結合させる。
- (4) 出力層のノード数は、10 (入力画像の文字の種類 (0~9) の数) である。  
データ型は「float32 (入力画像枚数, 10)」で、入力画像の各クラスへの所属確率 [0~1] を表す。  
中間層の出力を「Flatten」層を通すことで、「 $28 \times 28 \times 8 \text{枚} \times \text{入力チャネル数}$ 」という4次元のデータを一次元化して、出力層との間で全結合させる。
- (5) 中間層の活性化関数として「relu (正規化線形関数, Rectified Linear Unit)」を用いる。
- (6) 出力層の活性化関数として「ソフトマックス関数 (softmax)」を用いる。
- (7) 損失関数 (誤差を計算する関数) として、「クロスエントロピー誤差 (cross entropy)」を用いる。
- (8) 学習方法 (モデルの最適化方法) として、勾配降下法的一种である「Adam」を用いる。
- (9) 学習時には「エポック数」として10、「バッチサイズ」として「1000」で学習を行う。

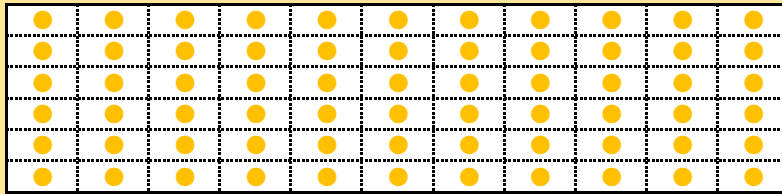
- ・MNIST を用いたモデル作成と評価、及び自作データでの認識精度

- (1) MNIST の検証用画像データの認識精度は99%と高く、  
 $2 \times 2$  の「最大値プーリング」を用いた「手書き文字認識モデル3」とほぼ同じで、  
自作の画像データの認識精度は、10枚中7枚、つまり70%でした。
- (2) 上記の結果は「Anaconda Navigator」の環境下での作成時のものですが、  
同じモデルとデータで、「Google Colaboratory」で実施した時の  
自作の画像データの認識精度は、10枚中8枚、つまり80%でした。  
モデルを毎回作成し直しているので、結果も少しずつ異なります。

- ・「手書き文字認識で°ル6」を図解すると以下のように全部で7層になります：  
 (※ 以下の図を見ると9層に見えますが、慣例的に入力層とFlatten層は層の厚さとしては加えないので、7層になります。)

#### 入力層

(画像 28×28 pixel, 1チャンネル(モノクロ画像))

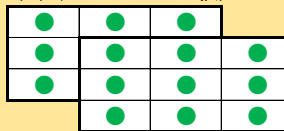


#### 中間層

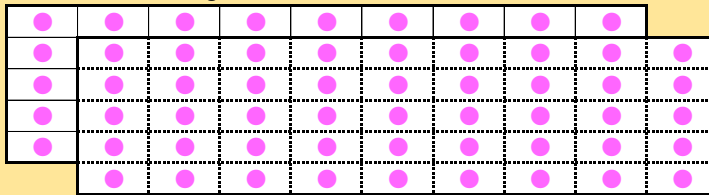
(中間層第1段)

畳み込み層

(フィルタ 3×3 が 16枚)



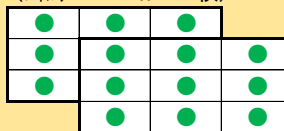
(特徴マップ° : Padding無しの為 26×26×16)



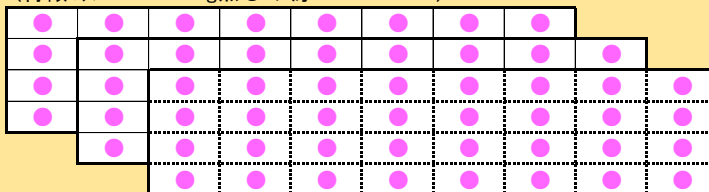
(中間層第2段)

畳み込み層

(フィルタ 3×3 が 32枚)



(特徴マップ° : Padding無しの為 24×24×32)

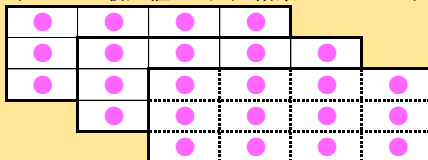


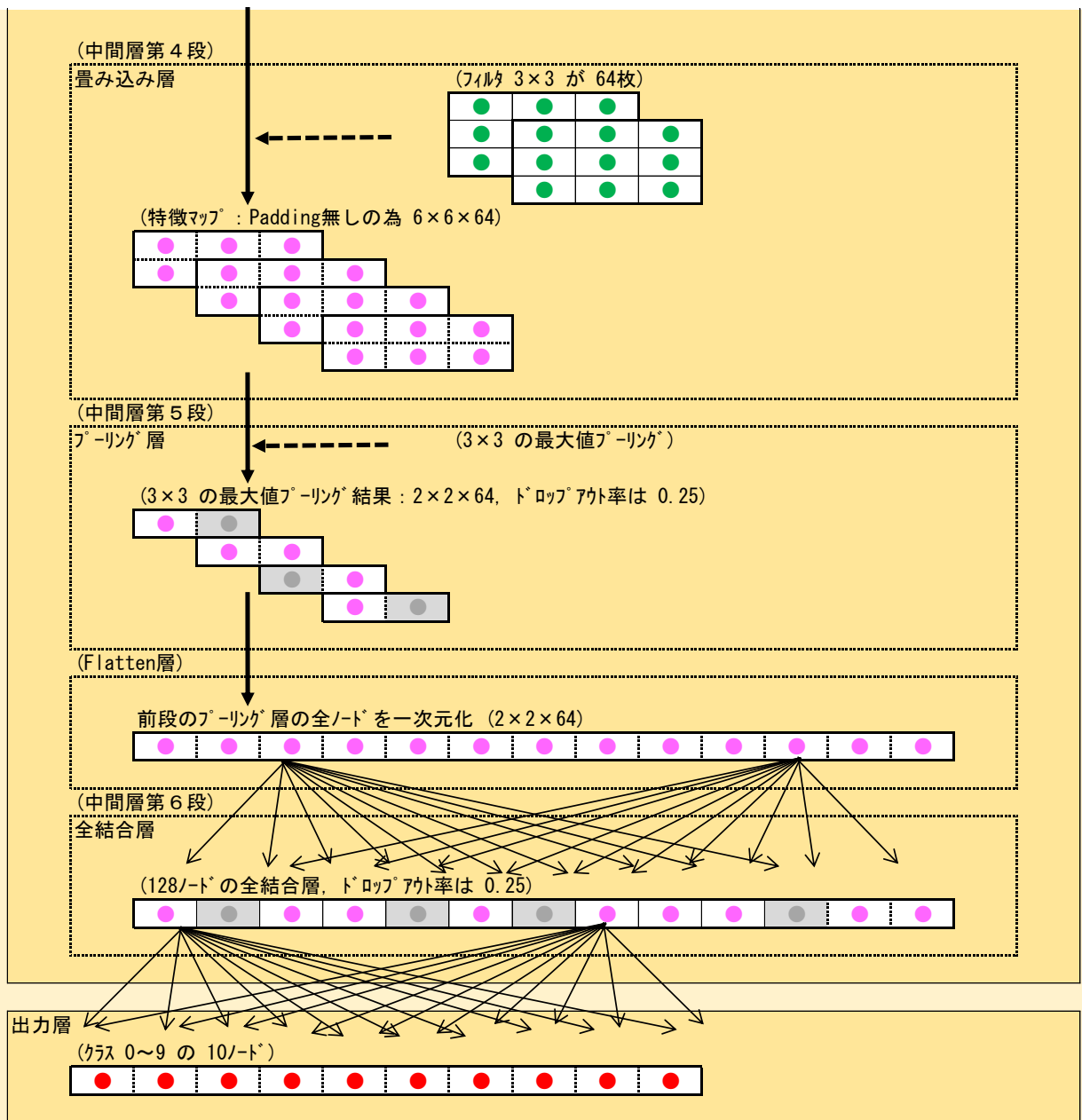
(中間層第3段)


プ°ーリング°層

(3×3 の最大値プ°ーリング°)

(3×3 の最大値プ°ーリング°結果 : 8×8×32)





 : ドロップアウト対象のノードを示します

### (3.6.1) 「手書き文字認識モデル6」の実装

- ・以下に、本モデルの実装を示します。

( リスト07-(03)-6\_手書き文字認識モデル6 )

```
#####/
# リスト07-(03)-6_手書き文字認識モデル6
#-----
# 「手書き文字認識モデル6」
# 深層畳み込みニューラルネットワーク (DCNN) ・ ・ ・ 7 層、3×3の最大値プーリング
#####/
# 参照ライブラリ
import numpy as np
np.random.seed(1)
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.optimizers import Adam
import time

#####/
# reshape_for_model6 */
# ロットした MNIST データを「手書き文字認識モデル6」 */
# 用に変形したものを返す */
#----- */
# 引数 : */
#   in_image : 入力画像 : uint8 (入力画像枚数, 28, 28) */
#   in_label : 入力画像が表す数値 (0~9) : uint8 (入力画像枚数) */
#----- */
# 戻り値 : */
# (1) in_image を */
#   「3次元(入力画像枚数, 28, 28)、uint8(0~255)」から */
#   「4次元(入力画像枚数, 28, 28, 1)、float32(0.0~1.0)」 */
#   へ正規化したもの */
# (2) in_label を */
#   「1次元(入力画像枚数)、uint8(0~8: 入力画像が表す数値)」から */
#   「1-of-k 符号化」で 10クラスへ分類し「2次元(入力画像枚数, 10)」 */
#   各クラスへの所属確率を float32(0.0~1.0) で表現したもの */
#   へ変換したもの */
#####/
def reshape_for_model6(in_image, in_label):
    return reshape_for_model2(in_image, in_label)

#####/
# make_and_evaluate_model6 */
# 「手書き文字認識モデル6」を作成してそれを評価する */
# 隠れ層の活性化関数は引数で指定可能 */
#----- */
# 「手書き文字認識モデル6」 */
# 深層畳み込みニューラルネットワーク (DCNN) */
#----- */
# (1) 入力層から出力層に向けて一方通行で流れる (Sequential)。 */
# (2) 入力層の入力データは、 */
#   (Y方向サイズ, X方向サイズ, 入力チャネル数) = (28, 28, 1) とする。 */
#   「入力チャネル数」はモノクロ画像なので1(カラー画像なら3)。 */
#   値は、0~255 の 256 階調のところを、 */
#   255 で割って 0~1 に正規化したものを用いる。 */
#----- */
# (3) 入力と出力の間に、中間層(隠れ層)を置く。 */
#   中間層を以下の6段で構成する。 */
# (3.1) 中間層第1段は、畳み込み層で、 */
```

```

#       フィルタ 3×3 を 16枚で、Padding無しで、畳み込む。          */
#   (3.2) 中間層第2段は、畳み込み層で、                              */
#       フィルタ 3×3 を 32枚で、Padding無しで、畳み込む。          */
#   (3.3) 中間層第3段は、3×3の最大値プーリング層。                  */
#   (3.4) 中間層第4段は、畳み込み層で、                              */
#       フィルタ 3×3 を 64枚で、Padding無しで、畳み込む。          */
#   (3.5) 中間層第5段は、3×3の最大値プーリング層で、                */
#       ドロップアウト率を 0.25 とする。                             */
#       この出力を、Flatten層を通して一次元化する。                */
#       一次元化したノードを、中間層第6段の各ノードとの間で全結合する。*/
#   (3.6) 中間層第6段は、全結合層で、ドロップアウト率を 0.25 とする。 */
#       各ノードを、出力層の各ノードとの間で全結合させる。        */
#                                                                 */
#   (4) 出力層のノード数は、10（入力画像の文字の種類(0~9)の数)である。 */
#       データ型は「float32（入力画像枚数,10)」で、                  */
#       入力画像の各クラスへの所属確率[0~1]を表す。                  */
#       中間層の出力を「Flatten」層を通すことで、                    */
#       「28×28×8枚×入力チャネル数」という                          */
#       4次元のデータを一次元化して、出力層との間で全結合させる。    */
#                                                                 */
#   (5) 中間層の活性化関数は「relu」                                */
#   (6) 出力層の活性化関数は「softmax」                              */
#   (7) 損失関数(誤差を計算する関数)は「categorical_crossentropy」 */
#   (8) 学習方法(モデルの最適化方法)は「Adam」                      */
#   (9) 学習時には「エポック数」として10、                          */
#       「バッチサイズ」として「1000」で学習を行う。                */
#-----*/
# 引数：                                                              */
#   title      : 表示用タイトル                                     */
#   actvFunc    : 隠れ層の活性化関数                               */
#-----*/
# 戻り値：                                                            */
#   (1) 作成したモデル                                              */
#-----*/
def make_and_evaluate_model6(title, actvFunc):

    print("*****")
    print(title)
    print("*****")

    #=====
    # ロードしたMNIST データを「手書き文字認識モデル6」用に変形する。
    #=====
    inTrain, outTrain = reshape_for_model6(x_train, y_train)
    inTest, outTest = reshape_for_model6(x_test, y_test)

    titlestr = ["modified ...", "inTrain", "outTrain", "inTest", "outTest"]
    print_MNIST_attr(titlestr, inTrain, outTrain, inTest, outTest)

    #=====
    # 「手書き文字認識モデル6」を作成
    #=====
    np.random.seed(1)

    predModel = Sequential()
    predModel.add(Conv2D(16, (3, 3),
        input_shape=(IMG_SIZE_Y, IMG_SIZE_X, 1), activation=actvFunc))
    predModel.add(Conv2D(32, (3, 3), activation=actvFunc))
    predModel.add(MaxPooling2D(pool_size=(3, 3)))
    predModel.add(Conv2D(64, (3, 3), activation=actvFunc))
    predModel.add(MaxPooling2D(pool_size=(3, 3)))
    predModel.add(Dropout(0.25))

```

```

predModel.add(Flatten())
predModel.add(Dense(128, activation=actvFunc))
predModel.add(Dropout(0.25))
predModel.add(Dense(IMG_CLASS_NO, activation='softmax'))
predModel.compile(loss='categorical_crossentropy',
                  optimizer=Adam(), metrics=['accuracy'])

predModel.summary()

#=====
# 「手書き文字認識モデル6」を学習し、
# 学習状況を、エポック回数の時系列でグラフ表示する
#=====
epochNo = 10
batchSize = 1000

startTime = time.time()
history = predModel.fit(inTrain, outTrain,
                        epochs=epochNo, batch_size=batchSize,
                        verbose=0, validation_data=(inTest, outTest))
score = predModel.evaluate(inTest, outTest, verbose=0)
endTime = time.time()

print('Test loss:', score[0])
print('Test accuracy:', score[1])
print("Computation time: {0:.3f} sec".format(time.time() - startTime))

plot_learning_curve(history, epochNo)

#=====
# 「手書き文字認識モデル6」で、
# MINTSTの検証用データの最初の50枚で分類結果を表示する。
#=====
pred_test_and_show_result(predModel, inTest, y_test, IMG_SIZE_X, IMG_SIZE_Y, 50)

return predModel

#-----/
# 「手書き文字認識モデル6」で
# 活性化関数を「relu」でモデルを作成して評価する。
#-----/
predModel = make_and_evaluate_model6(
    "「深層畳み込みニューラルネットワーク (DCNN)」で活性化関数は「relu」", 'relu')

#-----/
# 「手書き文字認識モデル6」で、
# 自作の画像の分類結果を表示する。
#-----/
pred_myData_and_show_result("手書き文字認識モデル6 (DCNN, relu)",
                             predModel, reshape_for_model6)

```

### (3.6.2) 「手書き文字認識モデル6」の実行結果

```
*****
「深層畳み込みニューラルネットワーク (DCNN)」で活性化関数は「relu」
*****
modified ...
inTrain.dtype=float32
inTrain.shape=(60000, 28, 28, 1)
inTrain.range=0.0~1.0

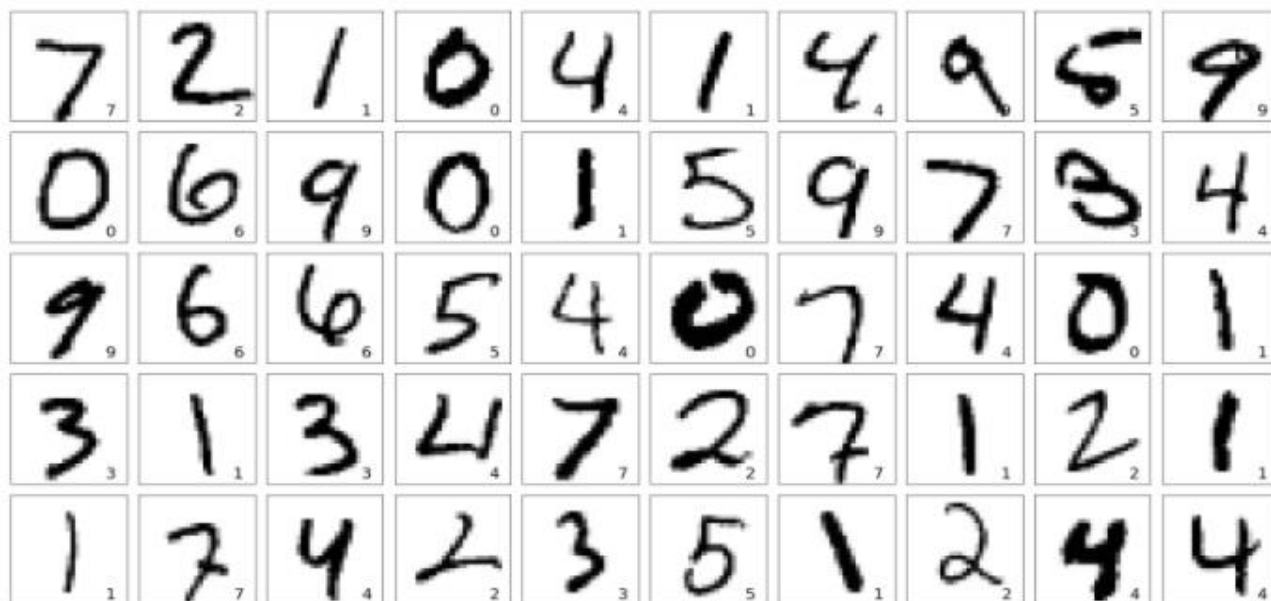
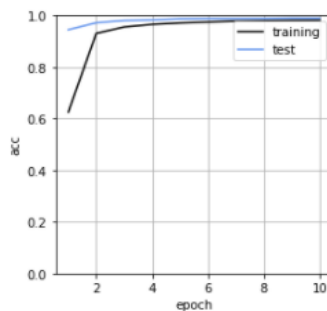
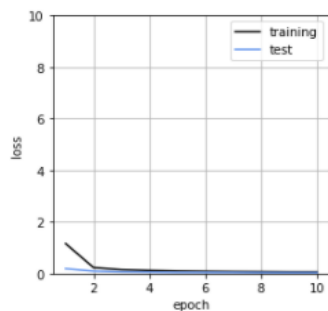
outTrain.dtype=float32
outTrain.shape=(60000, 10)
outTrain.range=0.0~1.0

inTest.shape=(10000, 28, 28, 1)
outTest.shape=(10000, 10)
Model: "sequential_7"
```

Layer (type)	Output Shape	Param #
conv2d_12 (Conv2D)	(None, 26, 26, 16)	160
conv2d_13 (Conv2D)	(None, 24, 24, 32)	4640
max_pooling2d_6 (MaxPooling2D)	(None, 8, 8, 32)	0
conv2d_14 (Conv2D)	(None, 6, 6, 64)	18496
max_pooling2d_7 (MaxPooling2D)	(None, 2, 2, 64)	0
dropout_7 (Dropout)	(None, 2, 2, 64)	0
flatten_5 (Flatten)	(None, 256)	0
dense_12 (Dense)	(None, 128)	32896
dropout_8 (Dropout)	(None, 128)	0
dense_13 (Dense)	(None, 10)	1290

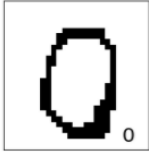
\*\*\*\*\*  
Total params: 57,482  
Trainable params: 57,482  
Non-trainable params: 0

Test loss: 0.028078894618351478  
Test accuracy: 0.9894999861717224  
Computation time:222.972 sec



# ※ 作成したモデルを用いて、自作の手書き文字を認識させてみた結果

手書き文字認識モデル6 (DCNN,relu)  
 filePath=HandwrittenImages/Handwritten\_0.bmp, label=0  
 predData.dtype=float32, shape=(1, 10), range=4.4398171894499683e-07~0.9952666759490967  
 (class,probability) 0:1.00 1:0.00 2:0.00 3:0.00 4:0.00 5:0.00 6:0.00 7:0.00 8:0.00 9:0.00



filePath=HandwrittenImages/Handwritten\_1.bmp, label=1  
 predData.dtype=float32, shape=(1, 10), range=2.500216282896872e-07~0.9982933402061462  
 (class,probability) 0:0.00 1:1.00 2:0.00 3:0.00 4:0.00 5:0.00 6:0.00 7:0.00 8:0.00 9:0.00



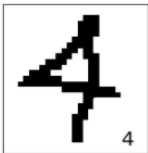
filePath=HandwrittenImages/Handwritten\_2.bmp, label=2  
 predData.dtype=float32, shape=(1, 10), range=5.548336234895146e-10~0.9995998740196228  
 (class,probability) 0:0.00 1:0.00 2:1.00 3:0.00 4:0.00 5:0.00 6:0.00 7:0.00 8:0.00 9:0.00



filePath=HandwrittenImages/Handwritten\_3.bmp, label=3  
 predData.dtype=float32, shape=(1, 10), range=7.769722287775949e-08~0.9990381002426147  
 (class,probability) 0:0.00 1:0.00 2:0.00 3:1.00 4:0.00 5:0.00 6:0.00 7:0.00 8:0.00 9:0.00



filePath=HandwrittenImages/Handwritten\_4.bmp, label=4  
 predData.dtype=float32, shape=(1, 10), range=7.677158464503009e-08~0.999803364276886  
 (class,probability) 0:0.00 1:0.00 2:0.00 3:0.00 4:1.00 5:0.00 6:0.00 7:0.00 8:0.00 9:0.00



filePath=HandwrittenImages/Handwritten\_5.bmp, label=5  
 predData.dtype=float32, shape=(1, 10), range=1.1917759001889294e-08~0.99981290102005  
 (class,probability) 0:0.00 1:0.00 2:0.00 3:0.00 4:0.00 5:1.00 6:0.00 7:0.00 8:0.00 9:0.00



filePath=HandwrittenImages/Handwritten\_6.bmp, label=6  
 predData.dtype=float32, shape=(1, 10), range=1.3384770625179954e-07~0.9697969555854797  
 (class,probability) 0:0.00 1:0.00 2:0.00 3:0.97 4:0.00 5:0.01 6:0.00 7:0.00 8:0.02 9:0.00



filePath=HandwrittenImages/Handwritten\_7.bmp, label=7  
 predData.dtype=float32, shape=(1, 10), range=0.00020815548487007618~0.4666207730770111  
 (class,probability) 0:0.02 1:0.04 2:0.04 3:0.00 4:0.00 5:0.00 6:0.00 7:0.42 8:0.47 9:0.00



filePath=HandwrittenImages/Handwritten\_8.bmp, label=8  
 predData.dtype=float32, shape=(1, 10), range=3.520426616887562e-08~0.7910404205322286  
 (class,probability) 0:0.00 1:0.00 2:0.00 3:0.79 4:0.00 5:0.00 6:0.00 7:0.00 8:0.21 9:0.00



filePath=HandwrittenImages/Handwritten\_9.bmp, label=9  
 predData.dtype=float32, shape=(1, 10), range=3.8959271919658534e-10~0.8083563446998596  
 (class,probability) 0:0.00 1:0.00 2:0.00 3:0.01 4:0.00 5:0.00 6:0.00 7:0.02 8:0.16 9:0.81



Total images=10, OK images=7, accuracy=0.70



### (3.7) 「手書き文字認識モデル7」

- ・「手書き文字認識モデル7」では、「手書き文字認識モデル6」を用いて「ドロップアウト率」を 0.25 から 0.50 に変更して試してみます。

- ・モデルとその学習方法は以下のようなものです：

- (1) 入力層から出力層に向けて一方通行で流れる (Sequential)。
- (2) 入力層の入力データは、(Y方向サイズ, X方向サイズ, 入力チャネル数) = (28, 28, 1) とする。  
「入力チャネル数」はモノクロ画像なので1(カラー画像なら3)。  
値は、0~255 の 256 階調のところを、255 で割って 0~1 に正規化したものを用いる。
- (3) 入力と出力の間に、中間層（隠れ層）を置く。  
中間層を以下の6段で構成する。
  - (3.1) 中間層第1段は、畳み込み層で、フィルタ 3×3 を 16枚で、Padding無しで、畳み込む。
  - (3.2) 中間層第2段は、畳み込み層で、フィルタ 3×3 を 32枚で、Padding無しで、畳み込む。
  - (3.3) 中間層第3段は、3×3の最大値プーリング層。
  - (3.4) 中間層第4段は、畳み込み層で、フィルタ 3×3 を 64枚で、Padding無しで、畳み込む。
  - (3.5) 中間層第5段は、3×3の最大値プーリング層で、ドロップアウト率を 0.50 とする。  
この出力を、Flatten層を通して一次元化し、  
一次元化したノードを、中間層第6段の各ノードとの間で全結合させる。
  - (3.6) 中間層第6段は、全結合層で、ドロップアウト率を 0.50 とする。  
各ノードを、出力層の各ノードとの間で全結合させる。
- (4) 出力層のノード数は、10 (入力画像の文字の種類(0~9)の数)である。  
データ型は「float32 (入力画像枚数, 10)」で、入力画像の各クラスへの所属確率[0~1]を表す。  
中間層の出力を「Flatten」層を通すことで、「28×28×8枚×入力チャネル数」という4次元のデータを一次元化して、出力層との間で全結合させる。
- (5) 中間層の活性化関数として「relu (正規化線形関数, Rectified Linear Unit)」を用いる。
- (6) 出力層の活性化関数として「ソフトマックス関数 (softmax)」を用いる。
- (7) 損失関数(誤差を計算する関数)として、「クロスエントロピー誤差 (cross entropy)」を用いる。
- (8) 学習方法(モデルの最適化方法)として、勾配降下法的一种である「Adam」を用いる。
- (9) 学習時には「エポック数」として10、「バッチサイズ」として「1000」で学習を行う。

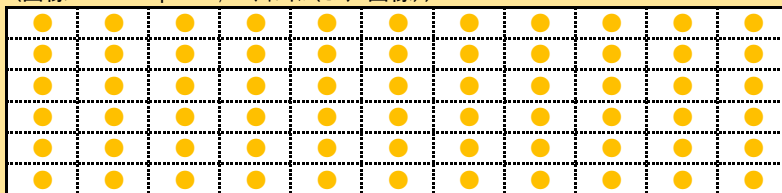
- ・MNIST を用いたモデル作成と評価、及び自作データでの認識精度

- (1) MNIST の検証用画像データの認識精度は99%弱と、  
「手書き文字認識モデル6」より若干落ちましたが、  
自作の画像データの認識精度は、10枚中8枚、つまり80%でした。
- (2) 上記の結果は「Anaconda Navigator」の環境下での作成時のものですが、  
同じモデルとデータで、「Google Colaboratory」で実施した時の  
自作の画像データの認識精度は、10枚中9枚、つまり90%でした。  
モデルを毎回作成し直しているので、結果も少しずつ異なります。
- (3) 自作の画像データ「6」は「8」と誤認識されましたが、もう少し中央寄りに文字を配置するような  
画像の作成あるいは前処理のし方をすれば認識精度が向上する可能性があります。

- ・「手書き文字認識7」を図解すると以下のように全部で7層になります：  
 (※ 以下の図を見ると9層に見えますが、慣例的に入力層とFlatten層は層の厚さとしては加えないので、7層になります。)

#### 入力層

(画像 28×28 pixel, 1チャンネル(モノクロ画像))

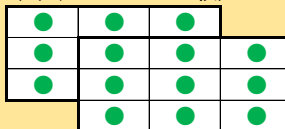


#### 中間層

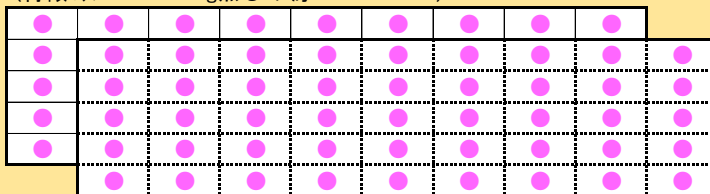
(中間層第1段)

畳み込み層

(フィルタ 3×3 が 16枚)



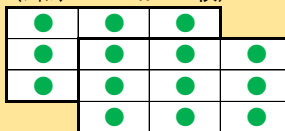
(特徴マップ : Padding無しの為 26×26×16)



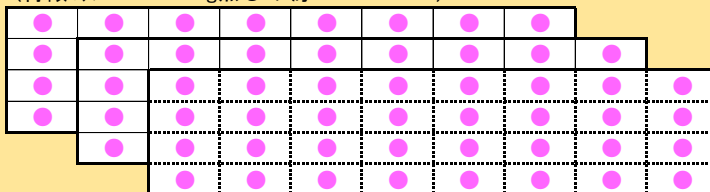
(中間層第2段)

畳み込み層

(フィルタ 3×3 が 32枚)



(特徴マップ : Padding無しの為 24×24×32)

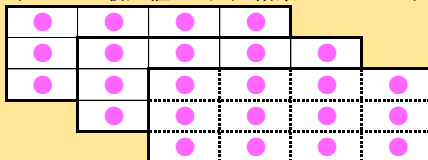


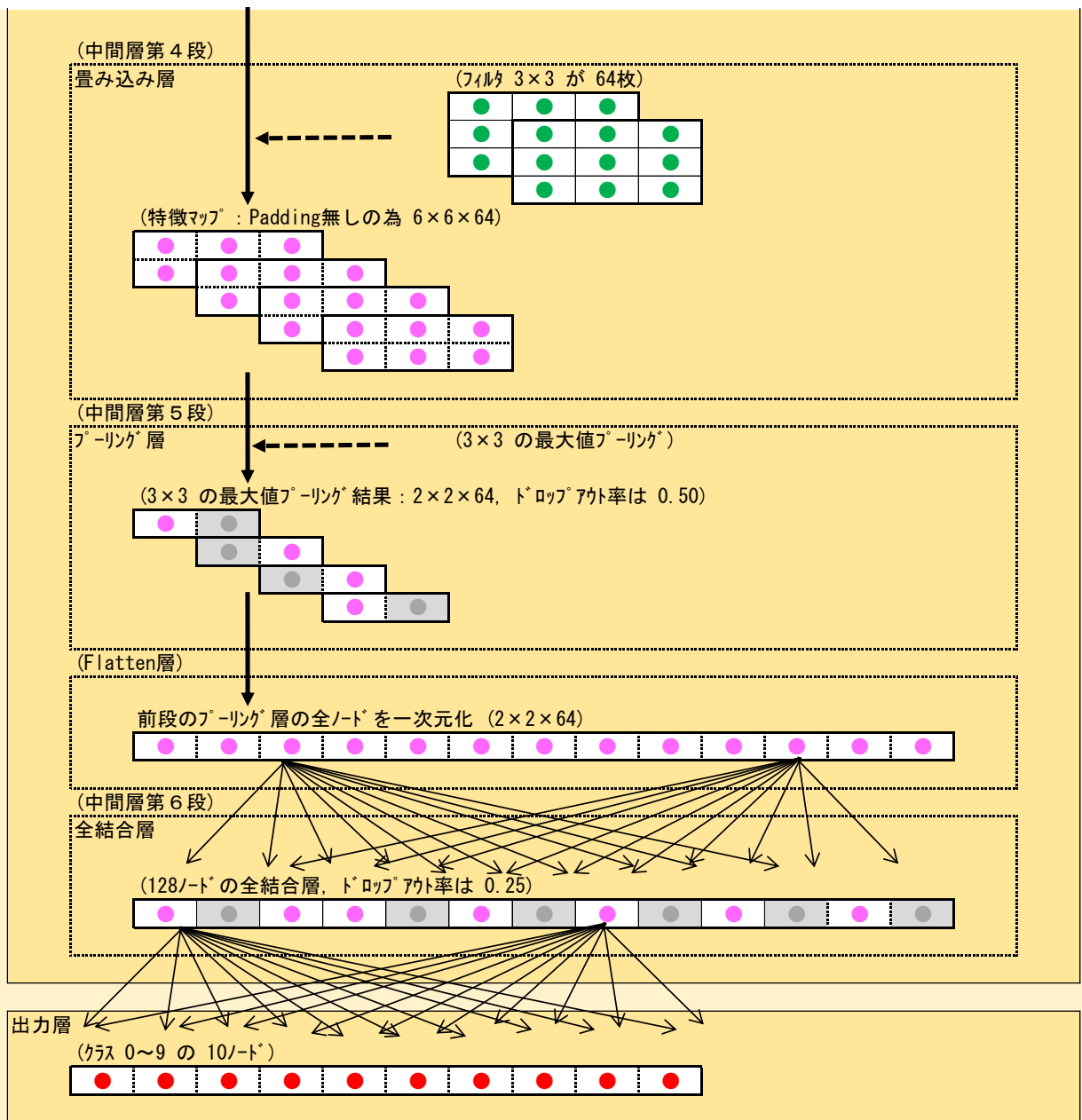
(中間層第3段)


プーリング層

(3×3 の最大値プーリング)

(3×3 の最大値プーリング結果 : 8×8×32)





 : ドロップアウト対象のノードを示します

### (3.7.1) 「手書き文字認識モデル7」の実装

- ・以下に、本モデルの実装を示します。

( リスト07-(03)-7\_手書き文字認識モデル7 )

```
#####/
# リスト07-(03)-7_手書き文字認識モデル7
#-----
# 「手書き文字認識モデル7」
# 深層畳み込みニューラルネットワーク (DCNN)
# . . . 「手書き文字認識モデル6」で、
# 「ドロップアウト率」を 0.25 から 0.50 に変更したもの
#####/
# 参照ライブラリ
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.optimizers import Adam
import time

#####/
# reshape_for_model7 */
# ロットした MNIST データを「手書き文字認識モデル7」 */
# 用に変形したものを返す */
#----- */
# 引数 : */
# in_image : 入力画像 : uint8 (入力画像枚数, 28, 28) */
# in_label : 入力画像が表す数値 (0~9) : uint8 (入力画像枚数) */
#----- */
# 戻り値 : */
# (1) in_image を */
# 「3次元(入力画像枚数, 28, 28)、uint8(0~255)」から */
# 「4次元(入力画像枚数, 28, 28, 1)、float32(0.0~1.0)」 */
# へ正規化したもの */
# (2) in_label を */
# 「1次元(入力画像枚数)、uint8(0~8: 入力画像が表す数値)」から */
# 「1-of-k 符号化」で 10クラスへ分類し「2次元(入力画像枚数, 10)」 */
# 各クラスへの所属確率を float32(0.0~1.0) で表現したもの */
# へ変換したもの */
#####/
def reshape_for_model7(in_image, in_label):
    return reshape_for_model2(in_image, in_label)

#####/
# make_and_evaluate_model7 */
# 「手書き文字認識モデル7」を作成してそれを評価する */
# 隠れ層の活性化関数は引数で指定可能 */
#----- */
# 「手書き文字認識モデル7」 */
# 深層畳み込みニューラルネットワーク (DCNN) */
# */
# (1) 入力層から出力層に向けて一方通行で流れる (Sequential)。 */
# (2) 入力層の入力データは、 */
# (Y方向サイズ, X方向サイズ, 入力チャネル数) = (28, 28, 1) とする。 */
# 「入力チャネル数」はモノクロ画像なので1(カラー画像なら3)。 */
# 値は、0~255 の 256 階調のところを、 */
# 255 で割って 0~1 に正規化したものを用いる。 */
# */
# (3) 入力と出力の間に、中間層 (隠れ層) を置く。 */
# 中間層を以下の6段で構成する。 */
```

```

# (3.1) 中間層第1段は、畳み込み層で、 */
#       フィルタ 3×3 を 16枚で、Padding無しで、畳み込む。 */
# (3.2) 中間層第2段は、畳み込み層で、 */
#       フィルタ 3×3 を 32枚で、Padding無しで、畳み込む。 */
# (3.3) 中間層第3段は、3×3の最大値プーリング層。 */
# (3.4) 中間層第4段は、畳み込み層で、 */
#       フィルタ 3×3 を 64枚で、Padding無しで、畳み込む。 */
# (3.5) 中間層第5段は、3×3の最大値プーリング層で、 */
#       ドロップアウト率を 0.25 とする。 */
#       この出力を、Flatten層を通して一次元化する。 */
#       一次元化したノードを、中間層第6段の各ノードとの間で全結合する。 */
# (3.6) 中間層第6段は、全結合層で、ドロップアウト率を 0.25 とする。 */
#       各ノードを、出力層の各ノードとの間で全結合させる。 */
# */
# (4) 出力層のノード数は、10（入力画像の文字の種類(0～9)の数)である。 */
#       データ型は「float32（入力画像枚数,10）」で、 */
#       入力画像の各クラスへの所属確率[0～1]を表す。 */
#       中間層の出力を「Flatten」層を通すことで、 */
#       「28×28×8枚×入力チャネル数」という */
#       4次元のデータを一次元化して、出力層との間で全結合させる。 */
# */
# (5) 中間層の活性化関数は「relu」 */
# (6) 出力層の活性化関数は「softmax」 */
# (7) 損失関数(誤差を計算する関数)は「categorical_crossentropy」 */
# (8) 学習方法(モデルの最適化方法)は「Adam」 */
# (9) 学習時には「エポック数」として10、 */
#       「バッチサイズ」として「1000」で学習を行う。 */
#----- */
# 引数： */
# title : 表示用タイトル */
# actvFunc : 隠れ層の活性化関数 */
#----- */
# 戻り値： */
# (1) 作成したモデル */
#----- */
def make_and_evaluate_model7(title, actvFunc):

    print("*****")
    print(title)
    print("*****")

    #-----
    # ロードしたMNIST データを「手書き文字認識モデル7」用に変形する。
    #-----
    inTrain, outTrain = reshape_for_model7(x_train, y_train)
    inTest, outTest = reshape_for_model7(x_test, y_test)

    titlestr = ["modified ...", "inTrain", "outTrain", "inTest", "outTest"]
    print_MNIST_attr(titlestr, inTrain, outTrain, inTest, outTest)

    #-----
    # 「手書き文字認識モデル7」を作成
    #-----
    predModel = Sequential()
    predModel.add(Conv2D(16, (3, 3),
        input_shape=(IMG_SIZE_Y, IMG_SIZE_X, 1), activation=actvFunc))
    predModel.add(Conv2D(32, (3, 3), activation=actvFunc))
    predModel.add(MaxPooling2D(pool_size=(3, 3)))
    predModel.add(Conv2D(64, (3, 3), activation=actvFunc))
    predModel.add(MaxPooling2D(pool_size=(3, 3)))
    predModel.add(Dropout(0.50))
    predModel.add(Flatten())

```

```

predModel.add(Dense(128, activation=actvFunc))
predModel.add(Dropout(0.50))
predModel.add(Dense(IMG_CLASS_NO, activation='softmax'))
predModel.compile(loss='categorical_crossentropy',
                   optimizer=Adam(), metrics=['accuracy'])

predModel.summary()

#=====
# 「手書き文字認識モデル」を学習し、
# 学習状況を、エポック回数の時系列でグラフ表示する
#=====
epochNo = 10
batchSize = 1000

startTime = time.time()
np.random.seed(1)
history = predModel.fit(inTrain, outTrain,
                        epochs=epochNo, batch_size=batchSize,
                        verbose=0, validation_data=(inTest, outTest))
score = predModel.evaluate(inTest, outTest, verbose=0)
endTime = time.time()

print('Test loss:', score[0])
print('Test accuracy:', score[1])
print("Computation time: {0:.3f} sec".format(time.time() - startTime))

plot_learning_curve(history, epochNo)

#=====
# 「手書き文字認識モデル」で、
# MINTSTの検証用データの最初の50枚で分類結果を表示する。
#=====
pred_test_and_show_result(predModel,
                           inTest, y_test, IMG_SIZE_X, IMG_SIZE_Y, 50)

return predModel

#####/
# 「手書き文字認識モデル」で
# 活性化関数を「relu」でモデルを作成して評価する。
#####/
predModel7 = make_and_evaluate_model7(
    "「深層畳み込みニューラルネットワーク (DCNN)」で活性化関数は「relu」", 'relu')

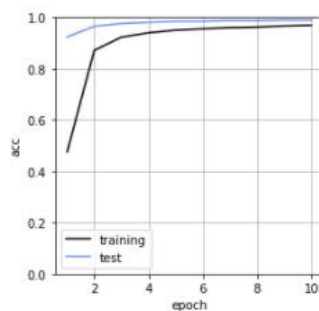
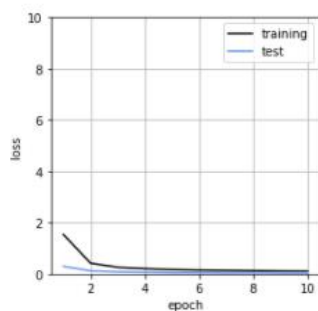
#####/
# 「手書き文字認識モデル」で、
# 自作の画像の分類結果を表示する。
#####/
pred_myData_and_show_result("手書き文字認識モデル (DCNN, relu)",
                             predModel7, reshape_for_model7)

```

### (3.7.2) 「手書き文字認識モデル」の実行結果

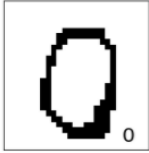
```
*****  
「深層畳み込みニューラルネットワーク (DCNN)」で活性化関数は「relu」  
*****  
modified ...  
inTrain.dtype=float32  
inTrain.shape=(60000, 28, 28, 1)  
inTrain.range=0.0~1.0  
  
outTrain.dtype=float32  
outTrain.shape=(60000, 10)  
outTrain.range=0.0~1.0  
  
inTest.shape=(10000, 28, 28, 1)  
outTest.shape=(10000, 10)  
Model: "sequential_8"
```

Layer (type)	Output Shape	Param #
conv2d_15 (Conv2D)	(None, 26, 26, 16)	160
conv2d_16 (Conv2D)	(None, 24, 24, 32)	4640
max_pooling2d_8 (MaxPooling2D)	(None, 8, 8, 32)	0
conv2d_17 (Conv2D)	(None, 6, 6, 64)	18496
max_pooling2d_9 (MaxPooling2D)	(None, 2, 2, 64)	0
dropout_9 (Dropout)	(None, 2, 2, 64)	0
flatten_6 (Flatten)	(None, 256)	0
dense_14 (Dense)	(None, 128)	32896
dropout_10 (Dropout)	(None, 128)	0
dense_15 (Dense)	(None, 10)	1290
Total params: 57,482		
Trainable params: 57,482		
Non-trainable params: 0		
Test loss: 0.03208133000920527		
Test accuracy: 0.9894000291824341		
Computation time:226.689 sec		



## ※ 作成したモデルを用いて、自作の手書き文字を認識させてみた結果

手書き文字認識モデル (DCNN,relu)  
 filePath=HandwrittenImages/Handwritten\_0.bmp, label=0  
 predData.dtype=float32, shape=(1, 10), range=1.4735069271409884e-06~0.997828623954773  
 (class,probability) 0:1.00 1:0.00 2:0.00 3:0.00 4:0.00 5:0.00 6:0.00 7:0.00 8:0.00 9:0.00



filePath=HandwrittenImages/Handwritten\_1.bmp, label=1  
 predData.dtype=float32, shape=(1, 10), range=1.321142804044939e-07~0.9980498423576355  
 (class,probability) 0:0.00 1:1.00 2:0.00 3:0.00 4:0.00 5:0.00 6:0.00 7:0.00 8:0.00 9:0.00



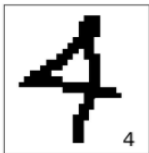
filePath=HandwrittenImages/Handwritten\_2.bmp, label=2  
 predData.dtype=float32, shape=(1, 10), range=1.4515705970552517e-08~0.9980679154396057  
 (class,probability) 0:0.00 1:0.00 2:1.00 3:0.00 4:0.00 5:0.00 6:0.00 7:0.00 8:0.00 9:0.00



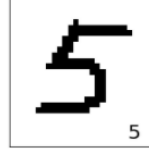
filePath=HandwrittenImages/Handwritten\_3.bmp, label=3  
 predData.dtype=float32, shape=(1, 10), range=4.9916227595758755e-08~0.9982981085777283  
 (class,probability) 0:0.00 1:0.00 2:0.00 3:1.00 4:0.00 5:0.00 6:0.00 7:0.00 8:0.00 9:0.00



filePath=HandwrittenImages/Handwritten\_4.bmp, label=4  
 predData.dtype=float32, shape=(1, 10), range=8.529703878547501e-08~0.9979537725448608  
 (class,probability) 0:0.00 1:0.00 2:0.00 3:0.00 4:1.00 5:0.00 6:0.00 7:0.00 8:0.00 9:0.00



filePath=HandwrittenImages/Handwritten\_5.bmp, label=5  
 predData.dtype=float32, shape=(1, 10), range=6.067101265827546e-10~0.9999589920043945  
 (class,probability) 0:0.00 1:0.00 2:0.00 3:0.00 4:0.00 5:1.00 6:0.00 7:0.00 8:0.00 9:0.00



filePath=HandwrittenImages/Handwritten\_6.bmp, label=6  
 predData.dtype=float32, shape=(1, 10), range=0.0002408088621450588~0.4331049919128418  
 (class,probability) 0:0.03 1:0.00 2:0.01 3:0.01 4:0.00 5:0.06 6:0.00 7:0.06 8:0.43 9:0.39



filePath=HandwrittenImages/Handwritten\_7.bmp, label=7  
 predData.dtype=float32, shape=(1, 10), range=8.211619165696902e-06~0.492817759513855  
 (class,probability) 0:0.00 1:0.21 2:0.26 3:0.00 4:0.00 5:0.00 6:0.00 7:0.49 8:0.03 9:0.00



filePath=HandwrittenImages/Handwritten\_8.bmp, label=8  
 predData.dtype=float32, shape=(1, 10), range=9.078377161131357e-08~0.7528039216995239  
 (class,probability) 0:0.00 1:0.00 2:0.00 3:0.75 4:0.00 5:0.00 6:0.00 7:0.00 8:0.24 9:0.00



filePath=HandwrittenImages/Handwritten\_9.bmp, label=9  
 predData.dtype=float32, shape=(1, 10), range=1.0570614696059355e-11~0.8700175881385803  
 (class,probability) 0:0.00 1:0.00 2:0.00 3:0.00 4:0.00 5:0.00 6:0.00 7:0.12 8:0.01 9:0.87



Total images=10, OK images=8, accuracy=0.80



## (4) まとめ

- ・モデルとそのパラメータを幾つか変えて試してみましたが、その結果は以下の様なものとなりました。

( 項番とモデルの概要 )

項番とモデル名	層数	モデルタイプ	パラメータ指定など
(3.1) 「手書き文字認識モデル1」	2	FFNN	中間層の活性化関数は sigmoid
			中間層の活性化関数は relu
(3.2) 「手書き文字認識モデル2」	2	CNN	プーリング層なし、バッチング有り
(3.3) 「手書き文字認識モデル3」	7	DCN	2×2 の最大値プーリング、ドロップアウト率 0.25
(3.4) 「手書き文字認識モデル4」	7	DCN	2×2 の平均値プーリング、ドロップアウト率 0.25
(3.5) 「手書き文字認識モデル5」	9	DCN	2×2 の最大値プーリング、ドロップアウト率 0.25
(3.6) 「手書き文字認識モデル6」	7	DCN	3×3 の最大値プーリング、ドロップアウト率 0.25
(3.7) 「手書き文字認識モデル7」	7	DCN	3×3 の最大値プーリング、ドロップアウト率 0.50

( モデル毎の実行結果 )

同じモデル構成とデータでも何度か実行すると、毎回モデルを作成し直している為か、その結果に違いがありました。

「Anaconda Navigator」で筆者の環境で1回実施したものを括弧無しで、

「Google Colaboratory」環境で1回実施したものを括弧 ( ) 内に記しておきます。

項番とモデル名	学習に要した時間 [sec]	MNIST 検証用画像の認識精度 [%]	自作の手書き文字の認識精度 [%]
(3.1) 「手書き文字認識モデル1」	3.3 ( 4.9)	89.3 (90.1)	40.0 (40.0)
	3.5 ( 5.1)	92.8 (92.8)	50.0 (30.0)
(3.2) 「手書き文字認識モデル2」	36.5 (100.0)	96.9 (97.3)	60.0 (60.0)
(3.3) 「手書き文字認識モデル3」	287.1 (665.5)	99.2 (99.2)	80.0 (70.0)
(3.4) 「手書き文字認識モデル4」	273.4 (635.8)	99.1 (99.0)	60.0 (70.0)
(3.5) 「手書き文字認識モデル5」	289.5 (680.7)	98.9 (98.9)	70.0 (80.0)
(3.6) 「手書き文字認識モデル6」	223.0 (519.7)	98.9 (99.2)	70.0 (80.0)
(3.7) 「手書き文字認識モデル7」	226.7 (516.9)	98.9 (98.9)	80.0 (90.0)

- ・実際のモデルの実装と認識精度をみて如何でしたでしょうか？

今回は、出来合いの MNIST のデータで学習し、手作りのデータもそれに合わせて、

28×28 ピクセルの「256色ビットマップ」形式で作成したものを入力するという前提で実装していますが、実際の課題に適用するには、

- (1) 画像から各文字を抽出し、モデルの入力とする為に、

抽出した文字を 28×28 ピクセルの「256色ビットマップ」形式の画像に変換する

- (2) 抽出した文字が 28×28 ピクセルのほぼ中央に位置するように移動する

といった前処理や、読み取った文字列を単語や文章として認識する機能などが必要となります。

- ・また、パラメータや階層の深さを変えたモデルの比較のために、

学習時の「エポック数」や「バッチサイズ」を一定にしましたが、

学習しきれていない場合は「エポック数」を多くとることも必要です(※1)。

また、学習方法(モデルの最適化方法)として、「Adam」だけ使用しましたが、

他を用いたらどうなるかも興味のある処です。

- ・ このように、Keras のようなライブラリを使いこなすことができれば、モデルの実装の敷居が 2000年代初頭の頃よりは飛躍的に低くなっている一方で(※2)、開発者には、使いこなす為の基礎知識が求められます。次回のセミナーでは、その基礎知識を扱います。

(※1) 学習曲線(横軸=epoch、縦軸=acc つまり精度)をみて、精度が上昇中で、頭打ちになっていない状態が「学習しきれていない場合」になります。

(※2) 筆者がデータマイニングの作業に携わっていたのは、2000年代初頭で、それは第3次AIブームが始まる10年ほど前でした。当時は AI のエンジン部分は大企業の研究所が開発したものを借用していました。現在では、エンジン部分がパッケージとして公開されていて、精度も各段に向上し、コンピュータリソースも桁違いの現在とは隔世の感があります。

以上。