

# matplotlibを用いた プログラムの解説

## A.1 はじめに

本書では、グラフ描画用にmatplotlibというライブラリを用いました。付録Aではmatplotlibを用いたプログラムの概略を解説します。必ずしも網羅的な解説にはなっておりませんので、プログラム中のコメント文による解説もあわせてご覧ください。

## A.2 2クラスへの分類における背景の色分け

4.4において、線形サポートベクトルマシンによるアヤマメの分類（2特徴量、2クラス）を2つのプログラムで体験しました。そのうちml-04-02-2feat2class01.pyを**プログラム4-2**（107～110ページ）として解説しました。

**プログラム4-2**の29行目以降では、アヤマメの2つのクラスを背景に異なる色を描画することで区別しています。その部分の概略について解説します。

### ●背景描画によるクラス分け

---

まず背景に色をつけるため、グラフの縦方向と横方向を

それぞれ500ずつ区切って250000個のグリッド（格子）を作ります。そしてその一つ一つのグリッドがどちらのクラスに属するかを予測します。

グリッド上の点からなるNumPy配列Xgに対して、それぞれの点がどちらのクラスに属するかを予測し、その結果をZに格納しているのが下記の行です。

```
47 Z = clf.predict(Xg)
```

データ数250000、特徴量2つのNumPy配列Xgと、要素数250000のNumPy配列Zとの関係は100ページの [図4-6\(A\)](#)のXとyの関係に似ています。

グリッド上の点の属するクラスを格納したZをもとに、背景に色を塗っているのが下記の2行です。

```
54 cmap01 = ListedColormap([(0.5, 1, 1, 1), (1, 0.93, 0.5, 1)])
```

```
57 plt.pcolormesh(XX, YY, Z==0, cmap=cmap01)
```

まず、2つの色を定義しています。0番目の色が(0.5, 1, 1, 1)で水色、1番目の色が(1, 0.93, 0.5, 1)でオレンジ色です。色は0から1の範囲の数字4つで表現し、順に赤、緑、青、不透明度の強さを表します。そして、「Z==0」（Zが0に等しい）という条件が満たされる点には1番目の色のオレンジで、満たされない点には0番目の色の水色で色をつけています。

## ●データ点の描画

次にデータ点の描画です。すべての点はXに格納されています。これらをクラスごとに色分けして表示するため、下記の2行でクラス0 (iris setosa) に属する点のみを集めたXc0とクラス1 (iris versicolor) に属する点のみを集めたXc1とに分けています。

```
66 Xc0 = X[y==0]
```

```
68 Xc1 = X[y==1]
```

そして、Xc0をオレンジ色の点●で描画しているのが下記の命令です。

```
71 plt.scatter(Xc0[:,0], Xc0[:,1], c='#E69F00', linewidths=0.5,  
edgecolors='black')
```

「Xc0[:,0], Xc0[:,1]」はそれぞれXc0の横方向の数値(外花被片の長さ)と縦方向の数値(外花被片の幅)を112ページの図4-9(C)の方法で取り出しています。この点が、塗りつぶしをオレンジ色(#E69F00)で、点の枠線を黒(black)で表示されています。色の指定の方法として、これまで登場したように「4つの数字の組み合わせ((0.5, 1, 1, 1))」、「Webページを作るときなどに使われる表記(#E69F00)」、「色名(black)」など複数の方法が利

用できます。

最後に、サポートベクトルを取り出して赤い枠を表示しているのが下記の命令です。

```
76 SV = clf.support_vectors_  
78 plt.scatter(SV[:,0], SV[:,1], c=(0,0,0,0), linewidths=1.0,  
edgecolors='red')
```

「`clf.support_vectors_`」でサポートベクトル全体を取り出せること、塗りつぶしの色として不透明度が0の透明色を指定していることなどがわかります。サポートベクトルマシンの利用においてサポートベクトルを明示的に表示する必要は必ずしもありませんが、このプログラムでは**4.2**での解説との対応をつけるためにサポートベクトルを表示しました。

### A.3 損失関数の時間変化のグラフ化

5章で解説した多層ニューラルネットワークでは、ネットワークの学習過程における損失関数の時間変化を確認することが重要なのでした。プログラム `ml-05-01-2feat3class-nn.py` において、そのグラフ化を `matplotlib` で実現している部分を解説します。

損失関数の時間変化をグラフ化しているのは `ml-05-01-2feat3class-nn.py` の中の下記の部分です。

```
93 #損失関数のグラフの軸ラベルを設定
94 plt.xlabel('time step')
95 plt.ylabel('loss')
96
97 # グラフ縦軸の範囲を0以上と定める
98 plt.ylim(0, max(clf.loss_curve_))
99
100 # 損失関数の時間変化を描画
101 plt.plot(clf.loss_curve_)
102
103 # 描画したグラフを表示
104 plt.show()
```

縦と横の軸にラベルを設定し、さらに縦軸の最小値を0にするよう設定したうえで、「`clf.loss_curve_`」をグラフ表示しています。「`clf.loss_curve_`」は学習開始時から終了時までの損失関数の値が格納されたNumPy配列です。

#### A.4 多数の手書き数字を描画する

6章で体験した手書き数字の分類では、サンプルに含まれる手書き数字画像を表示することでそのイメージをつかみました。この部分にもmatplotlibが使われています。それを実現するプログラム `ml-06-02-images.py` の概略を説明します。

## ●プログラムの全体

---

ml-06-02-images.py をすべて掲載したのが下記のプログラムA-1です。

```
1 # -*- coding: utf-8 -*-
2 from sklearn import datasets
3 import numpy as np
4 import matplotlib.pyplot as plt
5
6 # 手書き数字のデータをロードし、変数digitsに格納
7 digits = datasets.load_digits()
8
9 # 特徴量のセットを変数Xに、ターゲットを変数yに格納
10 X = digits.data
11 y = digits.target
12
13 # 手書き数字の画像表現を変数imagesに格納
14 images = digits.images
15
16 # 表示エリアの背景をシルバーにセット
17 fig = plt.figure()
18 fig.patch.set_facecolor('silver')
19
20 # 0~9の10枚の画像をそれぞれ3枚ずつ、計30枚描画
21 for i in range(10):
22     for j in range(3):
23         # 数字iの画像のうち、j枚目を取り出す
24         img = images[y==i][j]
```

```

25     #ランダムに取り出したい場合は下記を有効に
26     #img = images[y==i][np.random.randint(0, len(images
[y==i]))]
27     # 縦5x横6の画像表示エリアのうち、3*i+j+1番目に描画
28     plt.subplot(5, 6, 3*i + j + 1)
29     # グラフとしての軸は描画しない
30     plt.axis('off')
31     # 白黒を反転した状態で描画
32     plt.imshow(img, cmap=plt.cm.gray_r, interpolation
='nearest')
33     # 各画像にタイトルを描画
34     plt.title('Data {0}'.format(i))
35
36 # 画像間に余裕をもたせて描画
37 plt.tight_layout()
38
39 # 描画した内容を画面表示
40 plt.show()

```

プログラム A-1 0～9の画像をそれぞれ3枚ずつ表示する ml-06-02-images.py

## ●プログラムの解説

まず、変数 `digits` に格納された 1797 枚の画像を下記の命令で変数 `images` に格納することができます。

```

14 images = digits.images

```

この `images` から、0～9の10個の数字に対応する画像

を3枚ずつ取り出すには、下記のように二重のfor文を  
います。これを入れ子構造とも言います。

```
21 for i in range(10):  
22     for j in range(3):
```

このfor文の内部では、0～9の数字に対応する変数*i*お  
よび3枚の画像の何枚目かを示す変数*j*が利用可能になり  
ます。そして、数字*i*の*j*枚目の画像を取り出して変数に格納しているのが下記の命令です。

```
24         img = images[y==i][j]
```

そしてこの画像表示用の関数で表示しています。

なお、このプログラムでは各数字の画像のうち、先頭の  
3枚しか見ることができません。とはいえ、それぞれ約  
180枚ある数字のすべてを表示することは現実的ではあり  
ません。

そこで、閲覧する3枚の画像がランダムに選ばれるよう  
にもしてあります。プログラム中の下記の部分に着目して  
ください。

```
25         #ランダムに取り出したい場合は下記を有効に  
26         #img = images[y==i][np.random.randint(0, len(images  
[y==i]))]
```



この26行目の先頭の「#」のみを削除して保存すると、3枚の画像がランダムに選ばれるようになります。その際、削除するのは先頭の「#」1文字のみとしてください。その前にある8文字の空白文字のうち1文字でも削除すると、エラーが出てプログラムを実行できなくなるでしょう。これは、プログラミング言語Pythonでは行の先頭の空白文字の個数によりプログラムの構造が決まるためです。

なお、上の命令の「`np.random.randint(0, len(images[y==i]))`」は「0」から「`len(images[y==i])-1`」の範囲のランダムな整数、という意味になります。178枚ある数字0の画像の場合ですと、0から177の整数のどれかがランダムに選ばれます。そのようにしてランダムに選ばれた画像が画面に表示されます。

×ボタンでウインドウを閉じてからプログラムを再実行する、ということを何度か繰り返すと、その度にランダムに選ばれた数字が表示されます。それらを眺めると、さまざまな個性を持った数字を見ることができるよう。

## 付録 B

# OpenCVを用いたプログラムの解説

## B.1 はじめに

本書の8章では、画像処理用ライブラリOpenCVを用いた演習を行いました。付録Bでは、それらを実現したプログラムの概略を解説します。OpenCVはそれだけで1冊の本が書けるほど大規模なライブラリです。最初から全てを理解しようとするのではなく、時間をかけて少しずつ理解することを目指すのが良いと筆者は考えます。

## B.2 手の形の二値化

8.4にて、手の形を白、それ以外を黒とする画像の二値化を行い、その流れを紹介しました。ここではそれをOpenCVで実現するプログラムml-08-02-binary.pyの解説を行います。

### ●HSV形式への変換を行う命令

---

カメラから得られた画像をHSV形式へ変換し、色相 (Hue) と彩度 (Saturation) に対応する画像h\_channelとs\_channelを得ているのが下記の4行です。

```
54 hsv = cv2.cvtColor(stream.array, cv2.COLOR_BGR2HSV)
56 hsv_channels = cv2.split(hsv)
57 h_channel = hsv_channels[0]
58 s_channel = hsv_channels[1]
```

「`hsv = cv2.cvtColor(stream.array, cv2.COLOR_BGR2HSV)`」という命令により、カメラから得た画像 `stream.array` を HSV 形式の画像 `hsv` に変換しています。その次の「`hsv_channels = cv2.split(hsv)`」命令により HSV 形式の画像 `hsv` を色相 (Hue)、彩度 (Saturation)、明度 (Value) の3つに分解し、`hsv_channels` という変数に収めています。このうち0番目の要素を色相に対応する `h_channel` として、1番目の要素を彩度に対応する `s_channel` として利用可能にしているのが57行目、58行目です。明度の情報はこのプログラムでは利用しないのでした。

## ● `h_channel` の二値化を行う命令

`h_channel` の二値化を行っているのが下記の3命令です。

```
61 h_binary = cv2.GaussianBlur(h_channel, (5,5), 0)
65 ret,h_binary = cv2.threshold(h_binary, hmax, 255, cv2.
    THRESH_TOZERO_INV)
```

```
66 ret,h_binary = cv2.threshold(h_binary, hmin, 255, cv2.  
    THRESH_BINARY)
```

まず、61行目はh\_channelのデータのノイズ除去を行っています。そして、65行目はhmax～255の値を持つピクセルを0（黒）にする、という命令を実行しています。そして、66行目ではhmin～255の値を持つピクセル（実際にはhmin～hmaxの値のピクセルしかありません）を255（白）にしそれ以外を0（黒）にする、という命令を実行しています。以上により、264ページの図8-8(A)の処理が実現されます。

## ●s\_channelの二値化を行う命令

s\_channelの二値化は下記の1命令で実現されます。

```
69 ret,s_binary = cv2.threshold(s_channel, smin, 255, cv2.  
    THRESH_BINARY)
```

smin～255の値を持つピクセルを255（白）にしそれ以外を0（黒）にする、という命令です。それにより、図8-8(B)の処理が実現されます。

## ●h\_binaryとs\_binaryの共通部分hs\_andを生成する命令

h\_binaryとs\_binaryの共通部分hs\_andを生成する命令

は下記の1行で実現されます。

```
73 hs_and = h_binary & s_binary
```

すでに述べたようにこの処理はANDという演算で実現され、その記号は「&」です。

## ●白領域のうち最大のもののみ残す命令

白領域のうち最大のものを残す命令はやや複雑で、下記のように複数の命令の組み合わせで実現されます。

```
77 img_dist, img_label = cv2.distanceTransformWithLabels(255
    -hs_and, cv2.cv.CV_DIST_L2, 5)
78 img_label = np.uint8(img_label) & hs_and

80 img_label_not_zero = img_label[img_label != 0]

82 if len(img_label_not_zero) != 0:
83     m = stats.mode(img_label_not_zero)[0]
84 else:
85     m = 0

87 hand = np.uint8(img_label == m)*255
```

「cv2.distanceTransformWithLabels」は、対象となる画像のそれぞれのピクセルから、ゼロが格納されたピクセル

までの距離を計算する関数です。さらに、どのゼロに最も近いかにもとづいて全ピクセルにラベルをつけてグループ分けする機能もあります。

ここでは、ゼロ（黒）までの距離ではなく、255（白）までの距離としたいので、0（黒）と255（白）を入れ替えた「255-hs\_and」を「cv2.distanceTransformWithLabels」への入力としています。

その結果、もともと白であった点への距離を格納した画像

「img\_label\_not\_zero = img\_label[img\_label != 0]」を実行すると、黒い領域である0以外のラベルがNumPy配列として抜き出されます。

「m = stats.mode(img\_label\_not\_zero)[0]」を実行すると、最も個数の多いラベルが変数mに格納されます。これは、統計学の最頻値（モード）を計算する機能をSciPyのstatsパッケージから呼び出すことで実現しています。

あとは、ラベルmに対応する白領域のみに255を格納

し、他の点を0にすれば求めたい画像handが得られます。

### B.3 手の回転と左右反転

8.5では、学習用の手の画像の枚数を増やすため、手の回転と左右反転を行いました。それをOpenCVで実現しているプログラムはml-08-03-learn.pyです。ここではその概略を解説します。

手の回転と左右反転を実現しているのはml-08-03-learn.pyの下記の部分です。コメント文は省略して表示しています。

```
65 for flip in [0, 1]:
66     if flip == 1:
67         img = cv2.flip(img, 1)
68     for angle in [-80, -60, -40, -20, 0, 20, 40, 60, 80]:
69
70         rot_mat = cv2.getRotationMatrix2D((cx, cy), angle, 1.0)
71
72         img_rot = cv2.warpAffine(img, rot_mat, (lx, ly), flags
    =cv2.INTER_CUBIC)
```

プログラムの構造としては左右反転するかしないかの2通りを表すfor文「for flip in [0, 1]:」と、回転角度を9通りに変更するfor文「for angle in [-80, -60, -40, -20, 0, 20, 40, 60, 80]:」の入れ子構造になっています。これにより、 $2 \times 9 = 18$ 倍の枚数の画像を生成するこ

とになります。

変数`flip`が1のとき、すなわち左右反転を行う際は命令「`img = cv2.flip(img, 1)`」によって元画像`img`を左右反転しています。与えている数字の1は左右の反転であることを意味します。0を与えれば上下反転、-1を与えれば左右と上下両方の反転となります。

回転については、数学の概念である行列（マトリックス）がまず必要になります。

「`rot_mat = cv2.getRotationMatrix2D((cx, cy), angle, 1.0)`」により、角度`angle`の回転行列`rot_mat`が用意されます。与えている`(cx, cy)`は回転の中心ピクセルを表します。`(cx, cy)`には`(160, 120)`が格納されており、これは`320 × 240`のサイズの画像の中心位置です。つまり、画像の中心点周りに画像が回転するということです。与えている数字の「1.0」は画像の拡大率を表しており、1.0という数字は画像の拡大縮小は行わず回転のみを行うことを意味します。

最後に、回転行列`rot_mat`を用いて実際に画像の回転を行うのが命令「`img_rot = cv2.warpAffine(img, rot_mat, (lx, ly), flags=cv2.INTER_CUBIC)`」です。`(lx, ly)`は画像のサイズを表しており、ここでは`(320, 240)`が格納されています。回転された画像は`img_rot`に格納されます。



## B.4 手の切り出しと縮小

8.5では学習用の手の画像を810枚まで増やしたあと、それらをOpenCVを用いて縮小して機械学習に入力するベクトルを作成しています。その部分の解説を行います。

### ●手の切り出しと縮小を行うプログラム

手の切り出しと縮小を行い、最終的にベクトル $\mathbf{x}$ の作成までを行う命令は、ml-08-03-learn.pyの下記の部分、すなわち、getImageVectorという関数全体となります。関数とは、繰り返し利用される処理を、部品のようにまとめたものです。この関数には、回転処理を施された手の画像が渡されますので、それをもとにベクトル $\mathbf{x}$ が計算されます。

いつも通り、コメント文を省略して表示しています。

```
19 def getImageVector(img):  
  
21     nonzero = cv2.findNonZero(img)  
  
23     xx, yy, ww, hh = cv2.boundingRect(nonzero)  
  
25     img_nonzero = img[yy:yy+hh, xx:xx+ww]  
  
27     img_small = np.zeros((sh, sw), dtype=np.uint8)
```

```
29     if 4*hh < ww*3 and hh > 0:
30         htmp = int(sw*hh/ww)
31         if htmp>0:
32             img_small_tmp = cv2.resize(img_nonzero, (sw
, htmp), interpolation=cv2.INTER_LINEAR)
33             img_small[(sh-htmp)/2:(sh-htmp)/2+htmp, 0:
sw] = img_small_tmp
34         elif 4*hh >= ww*3 and ww > 0:
35             wtmp = int(sh*ww/hh)
36             if wtmp>0:
37                 img_small_tmp = cv2.resize(img_nonzero, (wtmp
, sh), interpolation=cv2.INTER_LINEAR)
38                 img_small[0:sh, (sw-wtmp)/2:(sw-wtmp)/2+wtmp] = img
_small_tmp
40     return np.array([img_small.ravel()/255.]])
```

このうち、切り出しを行っているのは下記の3行です。

```
21     nonzero = cv2.findNonZero(img)
23
24     xx, yy, ww, hh = cv2.boundingRect(nonzero)
25
26     img_nonzero = img[yy:yy+hh, xx:xx+ww]
```

「nonzero = cv2.findNonZero(img)」により、283ページの図8-15の左端のような画像のうち、ゼロ（黒）ではない点の座標がすべて変数nonzeroに格納されます。

「`xx, yy, ww, hh = cv2.boundingRect(nonzero)`」により、`nonzero`に含まれる座標をすべて包含する長方形の左上点の座標(`xx, yy`)と幅`ww`および高さ`hh`が得られます。この長方形は、283ページの図8-15の左端の画像の赤い長方形のことです

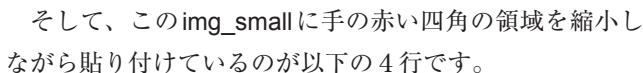
この長方形内の画像を変数に格納しているのが「`img_nonzero = img[yy:yy+hh, xx:xx+ww]`」です。NumPy配列の領域指定により画像を切り出しています。

次に、切り出された画像を縮小して最終的に変数`img_small`に格納します。`img_small`のサイズは幅16×高さ12なのでした。ここでは図8-15(A)および(B)における2つの矢印に相当する処理をまとめて行います。

ただし、プログラムとしては画像のアスペクト比による条件分けが必要なのでやや煩雑になっています。アスペクト比とは、画像の幅と高さの比のことです。最終的な画像`img_small`のアスペクト比は16:12、すなわち4:3です。この4:3よりも手が横長の場合と縦長の場合とがあります。横長の場合が図8-15(A)に該当し、手の上下に黒い余白が生まれます。縦長の場合が図8-15(B)に該当し、手の左右に黒い余白が生まれます。プログラムでは「`4*hh < ww*3`」の場合が横長の場合に対応し、「`4*hh >= ww*3`」の場合が縦長の場合に該当します。

手が横長の場合に絞って解説します。まず以下の命令により幅16×高さ12の黒い画像をあらかじめ用意しておきます。`sw`に16、`sh`に12が格納されています。

```
27 img_small = np.zeros((sh, sw), dtype=np.uint8)
```

そして、このに手の赤い四角の領域を縮小しながら貼り付けているのが以下の4行です。

```
30     htmp = int(sw*hh/ww)
31     if htmp>0:
32         img_small_tmp = cv2.resize(img_nonzero, (sw
, htmp), interpolation=cv2.INTER_LINEAR)
33         img_small[(sh-htmp)/2:(sh-htmp)/2+htmp, 0:
sw] = img_small_tmp
```

htmpは283ページの図8-15(A)右の画像において手の縦方向の範囲を表します。図8-15(A)右ではhtmpは9です。そして、「img\_small\_tmp = cv2.resize(img\_nonzero, (sw, htmp), interpolation=cv2.INTER\_LINEAR)」により、手の領域を幅sw、高さhtmpに縮小した画像img\_small\_tmpを得ます。図8-15(A)ではswは16、htmpは9ですね。

最後に、「img\_small[(sh-htmp)/2:(sh-htmp)/2+htmp, 0:sw] = img\_small\_tmp」という命令により、あらかじめ用意しておいた幅16×高さ12のimg\_smallにimg\_small\_tmpを貼り付けています。配列の領域指定により貼り付けています。

## ●ベクトルの完成

---

最後に、幅16×高さ12の画像

```
40 return np.array([img_small.ravel()/255.])
```

なお、ここで得られたベクトルは、下記のように分類器への入力データXへと加工されます。

まず、あらかじめ入力データXを空のデータとして初期化しておきます。ターゲットyの初期化も一緒に行っています。

```
43 X = np.empty((0,sw*sh), float)
```

```
44 y = np.array([], int)
```

このXとyに下記のようにデータを追加していきます。

```
76 X = np.append(X, img_vector, axis=0)
```

```
77 y = np.append(y, hand_class)
```

`img_vector` は `getImageVector` によって返された  $1 \times 192$  の NumPy 配列、`hand_class` は、0 (グー)、1 (チョキ)、2 (パー) に対応する整数値が格納されています。

「●手の回転と左右反転」(279ページ) で述べたように、手の画像は画像処理により得たものも合わせて810枚ありますから、最終的に  $X$  は  $810 \times 192$  (192次元ベクトルが810個) の NumPy 配列、 $y$  は810個の整数からなる NumPy 配列となります。

# 自分の手の画像を 学習用データとする方法

## C.1 はじめに

8章にて、Raspberry Piに接続したカメラに手をかざし、それがグー、チョキ、パーのいずれかを分類する課題を取り扱いました。その際、学習に用いる画像としては筆者が用意した45枚の画像を使用しました。付録Cでは、手の画像を皆さんで用意する方法を紹介します。

8章で用いたプログラム `ml-08-02-binary.py` を用いますので、8章の内容を十分理解しているという前提で以下の解説を進めます。

## C.2 プログラムを実行して画像を保存する

`ml-08-02-binary.py` を実行するためには、ターミナルを起動して、下記のコマンドを実行するのです。

```
python ml-08-02-binary.py
```

なお、3.3において本章のサンプルファイルを `bluebacks` ディレクトリに展開した方は、上記コマンドを実行する前に「`cd bluebacks`」コマンドを実行し、カレントデ

イレクトリをbluebacksディレクトリに変更することにも注意してください。

このとき、258ページの図8-6のように二値化された手が現れるのでした。手の領域だけを白く表示するための方法は8.4にて解説されています。

このプログラムを終了するためには、図8-6のウィンドウをマウスでクリックし、ウィンドウ上部のバーが水色になった状態でキーボードの「q」をタイプすれば良いことも覚えておきましょう。

このとき、表示されている手の画像をファイルに保存するために必要な知識を列挙すると下記のようになります。

- 画像はml-08-02-binary.pyと同じディレクトリにimg\_guXXX.png、img\_chokiXXX.png、img\_paXXX.pngという名称で保存される。「XXX」の部分はファイルの番号となっており、それぞれの手の画像が増えるたびに「000」→「001」→「002」……と増えていく
- 図8-6のウィンドウ上でキーボードの「g」をタイプすると、グー画像としてimg\_guXXX.pngという名称で保存される。同様に、「c」をタイプするとチョキ画像としてimg\_chokiXXX.pngという名称で保存される。「p」をタイプするとパー画像としてimg\_paXXX.pngという名称で保存される
- 画像を保存する順番は任意。すなわち、最初にグーだけを連続して保存しても構わないし、グー、チョキ、パーと順番に保存しても構わない



- 画像を保存する際は、274ページの図8-12を参考にさまざまな状況を保存すると良い。ただし、8.5で解説したように、回転と左右反転した画像についてはプログラムで生成するので考慮しなくて良い
- グー、チョキ、パーのそれぞれの画像の枚数は同じであることが学習を実行する上で望ましいが、それぞれ異なる枚数でもあとの学習プログラムは問題なく動作する
- 必要な枚数の画像を保存し終えたら、258ページの図8-6のウインドウ上でキーボードの「q」をタイプしてml-08-02-binary.pyを終了する。一般に、画像の枚数が多いほど分類の性能は上がることが期待されるが、そのぶん学習に長い時間がかかる

以上が保存に関する注意点です。ml-08-02-binary.pyと同じディレクトリに画像が保存されているか、ファイルマネージャで確認してみましょう。

### C.3 保存した画像を移動して学習する

保存した画像はml-learnディレクトリに移動しないと、学習時に読み込まれません。ただし、ml-learnディレクトリには筆者が用意した画像が存在します。それをあらかじめ削除するか、あるいはどこか別にディレクトリを作成してその中に移動して退避させておきましょう。そうしてから、皆さんが保存した画像をml-learnディレクトリに移動します。

以上の操作は、すべてファイルマネージャを用いて行うことができます。

あとは8.5の解説に従ってml-08-03-learn.pyを実行すると、皆さんが保存した画像を用いて学習が行われます。

ただし、以下の1点だけ注意してください。

- 画像の番号は連続していないとml-08-03-learn.pyから読み込まれない。たとえば、ゲーの画像としてimg\_gu000.png、img\_gu001.png、img\_gu002.png、img\_gu004.png……のように003番が飛ばされている場合、連続したimg\_gu002.pngまでの画像しか読み込まれず、img\_gu004.png以降の画像は無視される