情報通信プロジェクト実験

マルチメディア情報検索
B1 班 実験レポート

情報通信システムコース 3 年 B1 班
16173009 林田和磨 16173064 伊藤光太郎
18273002 平尾礼央 18273003 伊藤広樹

提出日 : 2019/1/25 (金)

1　実験の目的と内容

「キムタクに一番似ているのは誰か」という疑問や、「顔をパスワードの代わりとして利用したい」と言った要求に答えるシステムを開発する。代表的なパターン識別手法 (K 最近傍法、部分空間法等) を学びながら、高速で認識率のよいアルゴリズムを作成する。

本実験で使用するデータセットはデータベースとクエリから構成され、データベースには 20 人の画像がそれぞれの人物に対し 10 枚ずつの顔画像が収録されている。一方、クエリにはデータベースに登録されていない人物の画像を含め合計 58 枚の顔画像が含まれている。本実験では、クエリから顔画像を任意に選び、その画像がデータベースのどの人物かを識別する顔認識システムを開発する。

2　開発環境および、各手順の担当 (役割分担)

B1 班では MATLAB(R2015a) および Python を使用して実験をおこなった。Python 担当を伊藤広樹、平尾礼央とし、MATLAB 担当を伊藤光太郎、林田和磨とした。各手順において、それぞれの担当分は以下のとおりである。

- 顔のトリミング、正規化を実装 (Python(平尾礼央))
- 学習データの水増し (Python(伊藤広樹))
- 画像データの入力部 (MATLAB(林田和磨),Python(平尾礼央))
- Canny 法によるエッジ検出 (MATLAB(伊藤光太郎),Python(平尾礼央))
- 離散コサイン変換 (DCT) で特徴抽出 (MATLAB(伊藤光太郎),Python(平尾礼央))
- HOG 特徴量の抽出 (MATLAB(伊藤光太郎))
- Dlib を用いた顔の部位検出 (Python(平尾礼央))
- 顔の部位から各部位の長さ等の特徴抽出 (Python(伊藤広樹))
- 単純マッチング (MATLAB(伊藤光太郎),Python(平尾礼央、伊藤広樹))
- k 最近傍法 (MATLAB(伊藤広樹、伊藤光太郎),(Python(伊藤広樹))
- 部分空間法 (MATLAB(林田和磨、伊藤光太郎))
- CNN(Python(平尾礼央、伊藤広樹))
- LightGBM(Python(平尾礼央、伊藤広樹))
- GUI 作成 (Python(伊藤広樹))
- データの分析 (Python(平尾礼央、伊藤広樹))
- スライドの作成 (平尾礼央)
- 中間発表 (伊藤広樹)
- ポスターの作成 (平尾礼央)

- 最終発表 (伊藤広樹)
- レポート作成 (伊藤光太郎、平尾礼央)

3　作成したプログラムの機能の説明

まず、顔認識のシステムの全体像を図 1 に示す。

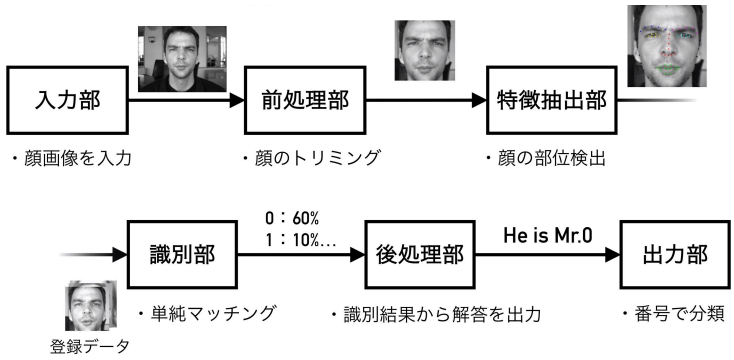

図 1: システムの概要

図 1 の各手順について、採用した手法およびそれぞれの機能をどのように実装したか、及び該当プログラムについて説明する。なお、プログラムは付録として末尾に記載している。

3.1　入力部

入力部では、与えられた画像を読み込む。MATLAB で作成したプログラムでは、使用する特徴量及び識別アルゴリズムを選択する処理も入力部で同時に行う。

- SAISYUU.m … パスを指定し、Python で顔検出および正規化された画像を MATLAB に読み込む。
- preprocessingOpenCV.py … OpenCV を用いて画像の読み込みを行う。画像はフォルダのパスを指定し、フォルダ内の jpg ファイルを読み込むように実装した。

3.2　前処理部

3.2.1　顔検出

与えられたデータセットの画像は人物のみでなく、背景も写ってしまっているため、識別する人物の顔だけを抽出する必要がある。

- preprocessingOpenCV.py … 画像を入力すると、顔を検出し、切り取った画像を出力する。

3.2.2 正規化

顔を検出し、トリミングした場合、画像によって大きさに差異が生まれる。そこで、まず画像サイズの統一化を行なった。その後、画像の明るさによる誤識別を防止するため、画素値のヒストグラムを平坦化することで輝度を調整した。

- preprocessingOpenCV.py … 上記の顔検出を行なった後、輝度の調整を行う。

なお、後述する「3.3 特徴抽出部」中の DCT、HOG の特徴量を使用する場合には Canny 法と呼ばれるエッジ検出をおこなった。なお、Canny 法はエッジ検出の一種で、ガウシアンフィルタを使って平滑化の処理をした後に微分をする。その微分画像の勾配からエッジを検出する方法である。

- SAISYUU.m … Canny 法によるエッジ検出を行った。
- DCT.py … Canny 法によるエッジ検出を行った。

3.3 特徴抽出部

今回の実験では 3 つの特徴量を画像データから抽出した。

3.3.1 二次元離散コサイン変換 (DCT)

画像を余弦波の周波数と係数成分へと変換する方法。画像のエネルギーは低域成分に集中しているため、変換後の周波数成分から低域の部分のみを特徴量として使用する。

- funcDCT.m … 画像を引数として渡す関数。DCT 変換を行い、低周波成分を抜き出したものを特徴量として使用。
- DCT.py … 画像の DCT 変換を行い、低周波成分を抽出する。

3.3.2 HOG(Histograms of Oriented Gradients)

局所領域 (セル) の画素値の勾配方向ヒストグラムを特徴量としたもの。各ピクセルの輝度から勾配を求めた後、セル領域ごとにヒストグラムを求める。それをブロックごとに正規化し、特徴量を抽出する。今回の実験ではセルのサイズを 8 × 8 のもの (HOG8) と 16 × 16 のもの (HOG16) の二種類を使用した。

- funcHOG.m … 画像を引数として渡す関数。HOG 特徴量を求めて特徴量として使用。

3.3.3 Dilb を使った顔パーツの部位情報

Python のライブラリである、Dlib を使用して、顔の部位情報から、各パーツの相対位置や距離を利用し、特徴量とした。MATLAB では Python で書き出された csv ファイルを読み込んで使用した。

- SAISYUU.m … 各識別アルゴリズム内で顔パーツの部位情報を使用する際に Python で作成された csv ファイルを読み込んで使用。

3.4 識別部

今回の実験では、5 つの識別アルゴリズムを用いて入力画像がどの人物なのか識別を行なった。

3.4.1 単純マッチング

特徴量や画像の画素値をピクセルごとに比較し、誤差が最も低いデータが属するクラス (今回は人物) を解答する方法。

- SAISYUU.m … 入力部で指定した、単純マッチングに使用する特徴量や画像をそれぞれ準備する。そして、クエリに対する特徴量を求め、単純マッチングの関数を呼び出し、回答を受け取る。
- matching.m … 画像の画素値や指定された特徴量を用いて、与えられたクエリと、200 個のデータベースそれぞれに対しての距離計算をしている。そして、クエリと最も距離が近くなったデータベースの画像を選び、その画像はどのクラス (人物) だったのかを判別する。
- processes.py … MATLAB と同様に特徴量の比較を行い、誤差が最小となるクラスに分類する。

3.4.2 k 最近傍法 (kNN)

特徴量でデータをプロットし、クエリの特徴量と近い場所にある k 個のデータの中で多数決をしてどのクラスに属するか解答する方法。

- SAISYUU.m … 特徴量を用いて kNN モデルを作成し、作成した kNN モデルを使用して多数決の結果と、その内訳を出力し、多数決の結果を回答として使用。リジェクト機能を使用する場合は、多数決の内訳をチェックし、それが条件を満たさない場合はリジェクトする (リジェクトの条件の詳細については後述)。
- labeling.m … kNN モデルを作成するときに特徴量とラベルを引数として渡す必要があるため、その二つを作成するための関数。
- processes.py … 上記の MATLAB とほぼ同じ仕様である。k 近傍の k 及び特徴量の比較に使う距離の指定はできるように実装した。

3.4.3 部分空間法

特徴量を使ってベクトルを作成し、クエリの特徴量を用いて作成したベクトルと最も類似しているクラスを出力する方法。

- SAISYUU.m … データベースの特徴量から 1 人当たり 10 本のベクトルを作成し、その 10 本のベクトルの平均をとってその人物の代表ベクトルを決定する。これを 20 人分繰り返し、最後にその代表ベクトルとクエリの特徴量から作ったベクトルと、20 人分の代表ベクトルを比較し、最も近似する代表ベクトルを持つ人物を回答する。リジェクト機能を使用する場合は、類似度 (2 つのベクトル間の角度) が閾値よりも大きいかどうか判定し、類似度が小さい (角度が大きい) 場合はリジェクトを行う。

3.4.4 畳み込みニューラルネットワーク (CNN) 及びニューラルネットワーク (NN)

顔を切り出し、前処理を施した画像を入力とし CNN を通し、どのクラスに属するかを出力する方法。ネットワークの構成は、畳み込み層-プーリング層-DropOut 層-畳み込み層-プーリング層-DropOut 層とした。活性化関数は ReLU 関数を使用した。DropOut の確率は 0.5 とし、プーリングのウィンドウサイズは 2 とした。ニューラルネットワークでは、特徴量を入力とし、中間層 3 層でそれぞれ 1024 ニューロンにした。出力は、0〜19 のクラス値とした。

- processes.py … CNN と NN は、pytorch を用いて実装した。processes クラスの CNN クラス及び、eval_net 関数、train_net 関数が実際の処理の内容である。train_net クラスではネットワークの訓練を行い、eval_net 関数では予測結果を算出する。予測された結果を使用し、精度を CNN クラスで計算し、それを返すという構成になっている。

3.4.5 LightGBM

特徴量抽出部で抽出した特徴量を使用し、Gradient Boosting ライブラリである LightGBM を使用し、入力された特徴量はどのクラスに属するかを出力する方法。boosting の概略として、決定木の弱識別機を直列に複数並べ、これらを訓練し、最終的に弱識別器の予測値を線形結合した値を出力として得るというものである。

- preprocess.py … preprocess クラスの lightGBM クラスが実際の処理の内容である。評価指標として multi error を使用し、学習率は 0.1 として、繰り返し数を 200、early stopping を 100 とした。これらの設定は params ディクショナリにまとめてある。最終的に、受け取った特徴量を元に学習を行い、その後訓練データで一番精度が良かった繰り返し数のパラメータで予測を行い正解率を返すという構成になっている。

3.5 出力部

今回の実験では正答率を求める必要があるため、出力した結果が正しいかどうかを判定するプログラムを追加した。

- SAISYUU.m …　正解ラベルを作成し、各手法によって回答された答えと正解ラベルを比較して正答率とリジェクト機能によってはじかれなかった個数を算出し、表示した。

- final_gui.py … PyQt を用いて作成した GUI 上で csv ファイルを読み込むことで精度を表示するように実装した。

4 実験結果

各手法での正解率は表 1 のようになった。

表 1: 各識別手法と特徴量の組み合わせによる正解率 [%]

	単純マッチング	K-NN(5)	部分空間法	Neural Network	LightGBM	ピクセルマッチング	CNN
各部位の大きさと位置	53.44	48.28	43.10	50.00	39.66	-	-
Canny あり DCT15	82.76	74.14	67.24	72.41	48.28	53.44	65.52
Canny なし DCT15	51.72	44.83	50.00	56.90	50.00		
HOG8	72.41	68.97	41.38	62.07	43.1		
HOG16	75.86	75.86	62.07	70.69	48.28		

5 考察

5.1 リジェクト機能

MATLAB で実装された各認識アルゴリズムにおいて、正答率が高かった特徴量を用いたものに対して「データベースに登録されていない人物をはじく機能 (リジェクト機能)」を実装した。リジェクトの方法は「距離や角度に対して閾値を用いてその閾値を上回った場合リジェクトする」方法 (単純マッチング、部分空間法) と「距離が近かった 5 つが同一でない場合にリジェクトする」方法をとった。その時の精度を表 4 に、再現率を表 5 に示す。なお、ここでの精度とは「識別器がリジェクトしなかったものに対して正しく分類できた割合」、再現率とは「データベースに登録されているクエリ (56 枚) に対して正しく分類した割合」とする。また、今回の実験ではシステムが「顔認証」であるため、精度ができるだけ 100%に近づけるようにし、その条件の下で最大となる再現率を表 5 に示した。

表 2: リジェクト機能を使用した際の適合率と再現率 [%]

	単純マッチング	K-NN(5)	部分空間法
Canny あり DCT15 適合率	100	87.88	100
Canny あり DCT15 再現率	46.43	57.14	25.00
HOG16 適合率	100	96.43	100
HOG16 再現率	48.21	50.00	37.93

表 1 と表 2 を比較すると、リジェクト機能を搭載することによって精度は向上しているが、その代わり再現率が下がるというトレードオフの関係が成り立っていることがわかる。また、表 1 の結果では単純マッチングおよび部分空間法では DCT のほうが正答率が高く、良い特徴量であると考えられていた。しかし、表 4、表 5 をみるとその 2 つの識別アルゴリズムではともに適合率は 100%であっても、HOG のほうが再現率は高くなっているため、リジェクト機能を使用する際には HOG 特徴量のほうが有用であると考えられる。また、k 最近傍法では、精度を 100%にすることができなかったため、リジェクト機能を搭載する場合に用いる識別アルゴリズムは単純マッチングか部分空間法が適しているのではないかと考える。

6 付録

6.1 MATLAB ファイル

ソースコード 1: SAISYUU.m

```
clc
clear
%% DB の実装
c = 20; % クラス総数
n = 10; % 1クラス当たりの学習パターン数

% データベースの読み込み
path = 'M:\project\dataset2\DB\jpeg\';
for i=1:c
   for j=1:n
      str = strcat(path, num2str(n*(i-1)+j-1, '%03d'), '.jpg');
      img = imread(str);
      img = edge(img, 'Canny',[],2);
      DB(:,:,n*(i-1)+j) = img;
   end
end

% クエリの読み込み
Qpath = 'M:\project\dataset2\Query\jpeg\';
D = dir('M:\project\dataset2\Query\jpeg\*.jpg');
for i=1:length(D)
   name = strcat(Qpath, D(i).name);
   img = imread(name);
   img = edge(img, 'Canny', [], 2);
   Query(:,:,i) = img;
end
%% 使用する機能選択
fprintf('どのアルゴリズムを使用しますか?\n')
prompt = '0=単純マッチング (輝度値),1=単純マッチング (DCT),2=単純マッチング (HOG),3=
      単純マッチング (Face_Parts),4=KNN(DCT),5=KNN(HOG),6=KNN(Face_Parts),7=部分空
      間 (DCT),8=部分空間 (HOG),9=部分空間 (Face_Parts)\n';
hanbetu = input(prompt);
fprintf('リジェクト機能を使用しますか?\n')
prompt = '1=Yes,0=No\n';
RJ = input(prompt);
answer = zeros(1, 58);
%% 単純マッチング
if (hanbetu >= 0 && hanbetu <= 3)
   %特徴量のDB 作成
   if (hanbetu >= 1 && hanbetu <= 2)
      for i=1:200
          img=DB(:,:,i);
        if hanbetu == 1
          m_feature = funcDCT(img);
        else
          m_feature = funcHOG(img);
        end
        m_DB(i,:) = m_feature;
      end
   end
   if hanbetu == 3
      m_DB = csvread('M:\project\dataset4\DB\csv\FP.csv');
      Query_feature = csvread('M:\project\dataset4\Query\csv\QFP.csv');
   end
   %単純マッチングによる回答
   for i=1:length(D)
      if hanbetu == 0
        Q_feature = double(Query(:,:,i));
      elseif hanbetu == 1
        Q_feature = funcDCT(Query(:,:,i)) ;
      elseif hanbetu == 2
        Q_feature = funcHOG(Query(:,:,i)) ;
```

```
61              elseif hanbetu == 3
62                 Q_feature = Query_feature(i,:) ;
63              end
64              if hanbetu == 0
65                 answer(i) = matching(DB, Q_feature, hanbetu, RJ);
66              else
67                 answer(i) = matching(m_DB, Q_feature, hanbetu, RJ);
68              end
69           end
70        end
71        %% KNN
72        if(hanbetu >= 4 && hanbetu <=6)
73
74           %knn モデルの作成
75           [fDB, C] = labeling(DB, hanbetu); %特徴量のラベリング
76           model = fitcknn(fDB, C);
77           model.NumNeighbors = 5;
78
79           %knn モデルによる回答
80           if hanbetu == 6
81              Query_filename = 'M:\project\dataset4\Query\csv\QFP.csv';
82              Query_feature = csvread(Query_filename);
83           end
84           for i=1:length(D)
85              testImg = Query(:,:,i); %クエリの取得
86              if hanbetu == 4
87                 dctF = funcDCT(testImg);
88                 [answer(i), score(i,:)] = predict(model, dctF);
89              elseif hanbetu == 5
90                 hogF = funcHOG(testImg);
91                 [answer(i), score(i,:)] = predict(model, hogF);
92              elseif hanbetu == 6
93                 faceF = Query_feature(i,:);
94                 [answer(i), score(i,:)] = predict(model, faceF);
95              end
96           end
97           %リジェクト領域
98           if RJ == 1
99              score2 = score.';
100             [MA,I] = max(score2);
101             for i=1:58
102                if MA(i) > 0.9
103                   answer(i) = I(i) − 1;
104                else
105                   answer(i) = 999;
106                end
107                   end
108          end
109       end
110       %% 部分空間 (DCT を特徴量とする場合)
111       if(hanbetu >= 7 && hanbetu <=8)
112          if hanbetu == 7
113             feature_num = 225;
114          else
115             feature_num = 2916;
116          end
117          %DB 部分
118          for i=1:200
119             img = DB(:,:,i);
120             if hanbetu == 7
121                db_feature_list(i,:) = funcDCT(img);
122             else
123                db_feature_list(i,:) = funcHOG(img);
124             end
125          end
126
127          %クエリ部分
128          for i=1:58
129             img = Query(:,:,i);
130             if hanbetu == 7
131                Query_feature(i,:) = funcDCT(img);
```

```
132             else
133                 Query_feature(i,:) = funcHOG(img);
134             end
135         end
136         for i=1:58
137             Query_feature(i, feature_num+1) = sqrt(sumsqr(Query_feature(i,:)));
138         end
139     end
140     %% 部分空間 (顔のパーツを特徴量として扱う場合)
141     if hanbetu == 9
142         %DB 部分
143         DB_filename = 'M:\project\dataset4\DB\csv\FP.csv';
144         db_feature_list = csvread(DB_filename);
145         feature_num = 20;
146 
147         %クエリ部分%
148         Query_filename = 'M:\project\dataset4\Query\csv\QFP.csv';
149         Query_feature = csvread(Query_filename);
150         %クエリのサイズ計算
151         for i=1:58
152             Query_feature(i, feature_num+1) = sqrt(sumsqr(Query_feature(i,:)));
153         end
154     end
155     %% 部分空間回答部
156     if(hanbetu >= 7 && hanbetu <= 9)
157         %平均値計算
158         sub_space = zeros(20, feature_num+1);
159         test = zeros(200, feature_num);
160         for i=1:20 %i=人数
161                 for j=1:10 % i 人目について 10 枚ずつ処理
162             for k=1:feature_num %特徴量が 20個
163                 test(j,k) = db_feature_list((i−1)∗10+j, k);
164             end
165         end
166         %平均値をとる
167         M = mean(test);
168         for k=1:feature_num
169             sub_space(i,k) = M(k);
170         end
171         zettai = sqrt(sumsqr(sub_space(i,:)));
172         sub_space(i, feature_num+1) = zettai;
173     end
174 
175     %部分空間法による回答
176     hairetu = zeros(58,20);
177     for j=1:58
178         for i=1:20
179             for k=1:feature_num
180                 hairetu(j,i) = hairetu(j,i) + Query_feature(j,k) ∗ sub_space(i,k);
181             end
182             hairetu(j,i) = acos(hairetu(j,i) / (Query_feature(j,feature_num+1) ∗ sub_space(i,feature_num
                    +1)) );
183         end
184     end
185     [S, answer] = min(hairetu, [], 2);
186 
187     %リジェクト領域
188     for i=1:58
189         if hanbetu == 8
190             sikiiti = 0.95;
191         elseif hanbetu == 7
192             sikiiti = 0.47;
193         end
194         if(RJ == 1 && S(i) > sikiiti)
195             answer(i) = 999;
196         else
197             answer(i) = answer(i) − 1;
198         end
199     end
200 end
```

```
%% Answer check

% Q ラベルの作成
A = zeros(1, 8);
Qlabels = A;
nums = [3, 3, 4, 3, 3, 4, 2, 3, 3, 3, 3, 3, 3, 2, 1, 1, 1, 1, 2, 2];
for i=1:20
  A = ones(1, nums(i) ) * i;
  Qlabels = horzcat(Qlabels, A);
end

% 正解率の判定
correctNum = 0;
PassNum = 0;
for i=1:length(D)
  if(answer(i) > 19 && RJ == 1)
      PassNum = PassNum + 1;
  end
  if(answer(i) == Qlabels(i))
      correctNum = correctNum + 1;
  end
end

sprintf('正解率: %.4f', correctNum / ( length(D) ) )
sprintf('精度 %.4f',correctNum / ( PassNum ))
```

ソースコード 2: matching.m

```
function number = matching(DB, Query, hanbetu, RJ)
  %単純マッチング関数 (関数M ファイル)
  %クエリ画像
      X と DB の画像をピクセル毎に比較し、二乗誤差が最も小さい人物を出力する
  for i=1:200
    if hanbetu == 0
      A = double(DB(:,:,i));
    elseif (hanbetu >= 1 && hanbetu <= 3)
      A = DB(i,:);
    end
    D = (Query - A).^2;
    distance(i) = sum(sum(D));
  end

  [minimum, index] = min(distance);
  %リジェクト領域
  if RJ == 1
      if hanbetu == 1
          sikiiti = 140;
      elseif hanbetu == 2
          sikiiti = 57;
      end
    if minimum > sikiiti
      index = 10000;
    end
  end

  number = ceil(index / 10) - 1;
end
```

ソースコード 3: funcDCT.m

```
function dctFeature = funcDCT(img)
  % DCT 特徴量
  img4 = dct2(double(img)); %2次元DCT
  imgdctlow = img4(1:15, 1:15); %低周波成分の抜き出し
  dctFeature = reshape(imgdctlow, [1, 15*15]); %1次元化
end
```

ソースコード 4: funcHOG.m

```
function hogFeature = funcHOG(img)
  % DCT 特徴量
  hogFeature = extractHOGFeatures(img, 'CellSize', [16 16]);
end
```

ソースコード 5: labeling.m

```
function [fDB,C] = labeling(DB,hanbetu)
  %特徴量のDB 作成関数

  if hanbetu == 6
    DB_filename = 'M:\project\dataset4\DB\csv\FP.csv';
    feature_list = csvread(DB_filename);
  end

  for i=1:200
    %DB の i 枚目を読み込む
    img = DB(:,:,i);

    %i 枚目の特徴量計算
    if hanbetu == 4
      feature = funcDCT(img); %dct の場合
    elseif hanbetu == 5
      feature = funcHOG(img);
    elseif hanbetu == 6
      feature = feature_list(i,:);
    end

    %特徴量のDB
    fDB(:,i) = feature;
    %正解ラベル
    C(i) = fix( (i-1) / 10);
  end

  C = transpose(C);
  fDB = transpose(fDB);
end
```

6.2 Python ファイル

ソースコード 6: processes.py

```
# coding: utf−8
# 最終的なモデル詰め合わせ (スクリプトバージョン)

import numpy as np
import pandas as pd
from tqdm import tqdm

# kNN
from sklearn.neighbors import KNeighborsClassifier

# NeuralNet, CNN
import torch
from torch import nn, optim
from torch.utils.data import TensorDataset, DataLoader
torch.manual_seed(0)

# LightGBM
import lightgbm as lgb
from sklearn.metrics import accuracy_score

# 画像読み込み
from pathlib import Path
import cv2

# ニューラルネット用
class FlattenLayer(nn.Module):
    def forward(self, x):
        sizes = x.size()
        return x.view(sizes[0], −1)

class processes:
    def __init__(self, dbPath, queryPath, isFolder):
        if(isFolder):
            # 画像の場合target の取得方法がないので強制的に、この csv から取ってくる
            self.targetDBPath = "../input/Dlib/cutface/histFlattening/DB/csv/features_rel_dist.csv"
            self.targetQueryPath = "../input/Dlib/cutface/histFlattening/Query/csv/features_rel_dist.csv"
            self.db_df = pd.read_csv(self.targetDBPath)
            self.query_df = pd.read_csv(self.targetQueryPath)
            self.db_target_df = self.db_df["target"].copy()
            self.query_target_df = self.query_df["target"].copy()

            # DB 画像読み込み
            p = Path(dbPath)
            p = sorted(p.glob("∗.jpg"))
            self.dbImages = []
            for index, filename in enumerate(tqdm(p)):
                # 相対パスだと参照できなかったので絶対パスでやる
                img = cv2.imread(str(filename.resolve()), 0)
                # C, H, W の形式にする(今回はグレースケールなのでC = 1)
                img = img.reshape([1, img.shape[0], img.shape[1]])
                self.dbImages.append((img/225).astype(np.float32))

            # Query 画像読み込み
            p = Path(queryPath)
            p = sorted(p.glob("∗.jpg"))
            self.queryImages = []
            for index, filename in enumerate(tqdm(p)):
                # 相対パスだと参照できなかったので絶対パスでやる
                img = cv2.imread(str(filename.resolve()), 0)
                # C, H, W の形式にする(今回はグレースケールなのでC = 1)
                img = img.reshape([1, img.shape[0], img.shape[1]])
                self.queryImages.append((img/225).astype(np.float32))

        else:
            self.db_df = pd.read_csv(dbPath)
            self.query_df = pd.read_csv(queryPath)
```

```
            self.db_feature_df = self.db_df.drop(["target"], axis=1).copy()
            self.db_target_df = self.db_df["target"].copy()
            self.query_featture_df =self.query_df.drop(["target"], axis=1).copy()
            self.query_target_df = self.query_df["target"].copy()


    def calc_accuracy(self, process_type):
        # {"単純マッチング":0, "kNN":1, "NeuralNet":2, "LightGBM":3, "ピクセルマッチング":4, "
            CNN":5}
        if(process_type == 0):
            accuracy = self.simple_matching()
        elif(process_type == 1):
            accuracy = self.kNN()
        elif(process_type == 2):
            accuracy = self.NeuralNet()
        elif(process_type == 3):
            accuracy = self.lightGBM()
        elif(process_type == 4):
            accuracy = self.pixelMatching()
        elif(process_type == 5):
            accuracy = self.CNN()
        else:
            accuracy = -100
        return accuracy


    def simple_matching(self):
        prediction_df = pd.DataFrame(self.query_target_df)
        for i in range(self.query_featture_df.shape[0]):
            minimum_id = (((self.db_feature_df - self.query_featture_df.iloc[i])**2).sum(axis=1)).idxmin()
            prediction_df.loc[i, "predict"] = self.db_target_df[minimum_id]

        correct_num = (prediction_df["target"] == prediction_df["predict"]).sum()
        accuracy = correct_num / prediction_df.shape[0]
        return accuracy


    def kNN(self):
        DIV_NUM = 3 # k
        DIST_SETTING = 2 # ユークリッド=2, マンハッタン=1
        knn = KNeighborsClassifier(n_neighbors=DIV_NUM, p=DIST_SETTING, metric="minkowski")
        knn.fit(self.db_feature_df.values, self.db_target_df.values)
        prediction = knn.predict(self.query_featture_df.values)
        prediction_df = pd.DataFrame(self.query_target_df)
        prediction_df["predict"] = prediction
        correct_num = (prediction_df["target"] == prediction_df["predict"]).sum()
        accuracy = correct_num / prediction_df.shape[0]
        return accuracy

    def NeuralNet(self):
        input_size = self.db_feature_df.shape[1]
        # define network
        net = nn.Sequential(
                nn.Linear(input_size, 1024),
                nn.ReLU(),
                nn.Linear(1024, 1024),
                nn.ReLU(),
                nn.Linear(1024, 1024),
                nn.ReLU(),
                nn.Linear(1024, 1024),
                nn.ReLU(),
                nn.Linear(1024, 20)
        )

        X_train = torch.tensor(self.db_feature_df.values, dtype=torch.float32)
        y_train = torch.tensor(self.db_target_df.values, dtype=torch.int64)

        X_test = torch.tensor(self.query_featture_df.values, dtype=torch.float32)
        y_test = torch.tensor(self.query_target_df.values, dtype=torch.int64)
```

```
        # 損失関数
        loss_fn = nn.CrossEntropyLoss()
        # adam
        optimizer = optim.Adam(net.parameters())
        # 損失ログ
        losses_train = []
        accuracy_test = []
        accuracy_train = []
        EPOCH = 300

        # 20エポック回す
        # ここだけ繰り返すと再学習しちゃうので注意
        for epoc in range(EPOCH):
            optimizer.zero_grad()

            y_pred = net(X_train)

            loss = loss_fn(y_pred, y_train)
            loss.backward()

            optimizer.step()

            losses_train.append(loss.item())

            _, predicted = torch.max(y_pred, 1)
            corrects_train = 0
            for i in range(len(predicted)):
                if(predicted[i]==y_train[i]):
                    corrects_train += 1
            accuracy_train.append(corrects_train/len(y_train))

            y_test_pred = net(X_test)
            _, predicted = torch.max(y_test_pred, 1)
            corrects_test = 0
            for i in range(len(predicted)):
                if(predicted[i]==y_test[i]):
                    corrects_test += 1
            accuracy_test.append(corrects_test/len(y_test))

            if(epoc%50 == 0 or epoc == (EPOCH-1)):
                print("-"*8+"epoch{}".format(epoc)+"-"*8)
                print("train accuracy:{:.3}".format(accuracy_train[-1]))
                print("train loss:{:.3}".format(losses_train[-1]))
                print("test accuracy:{:.3}".format(accuracy_test[-1]))
                print("-"*20)
        return(max(accuracy_test))


    def lightGBM(self):
        lgb_train = lgb.Dataset(self.db_feature_df.values, self.db_target_df.values)
        lgb_eval = lgb.Dataset(self.query_featture_df.values, self.query_target_df.values, reference=
            lgb_train)

        # LightGBM parameters
        params = {
            'task': 'train',
            'boosting_type': 'gbdt',
            'objective': 'multiclass',
            'metric': 'multi_error',
            'num_class': 20,
            'learning_rate': 0.1,
            'num_leaves': 15,
            'min_data_in_leaf': 10,
            'num_iteration': 200,
            'verbose': -1,
        }
        gbm = lgb.train(params,
                        lgb_train,
                        num_boost_round=300,
                        valid_sets=lgb_eval,
                        early_stopping_rounds=100)
```

```
207             y_pred = gbm.predict(self.query_featture_df.values, num_iteration = gbm.best_iteration)
208             y_pred_max = np.argmax(y_pred, axis=1)
209
210             return (accuracy_score(self.query_target_df, y_pred_max))
211
212         def pixelMatching(self):
213             prediction_df = pd.DataFrame(self.query_target_df)
214             for queryIndex in range(len(self.queryImages)):
215                 distances = np.zeros(len(self.dbImages), dtype=np.float32)
216                 for dbIndex in range(len(self.dbImages)):
217                     distances[dbIndex] = (np.abs(self.dbImages[dbIndex] − self.queryImages[queryIndex])).
                            sum()
218                 minimum_id = np.argmin(distances)
219                 prediction_df.loc[queryIndex, ”predict”] = self.db_target_df[minimum_id]
220             correct_num = (prediction_df[”target”] == prediction_df[”predict”]).sum()
221             accuracy = correct_num / prediction_df.shape[0]
222             return accuracy
223
224
225         def CNN(self):
226             X_dbToech = torch.Tensor(self.dbImages)
227             y_dbTorch = torch.LongTensor(self.db_target_df)
228             X_queryTorch = torch.Tensor(self.queryImages)
229             y_queryTorch = torch.LongTensor(self.query_target_df)
230
231             dbDataset = TensorDataset(X_dbToech, y_dbTorch)
232             queryDataset = TensorDataset(X_queryTorch, y_queryTorch)
233
234             batch_size = 8
235             dbLoader = DataLoader(dbDataset, batch_size=batch_size, shuffle=True)
236             queryLoader = DataLoader(queryDataset, batch_size=batch_size, shuffle=False)
237             conv_net = nn.Sequential(
238                 nn.Conv2d(1, 32, 5),
239                 nn.MaxPool2d(2),
240                 nn.ReLU(),
241                 nn.BatchNorm2d(32),
242                 nn.Dropout2d(0.5),
243                 nn.Conv2d(32, 64, 5),
244                 nn.MaxPool2d(2),
245                 nn.ReLU(),
246                 nn.BatchNorm2d(64),
247                 nn.Dropout2d(0.5),
248                 FlattenLayer()
249             )
250             test_input = torch.ones(1, 1, 128, 128)
251             conv_output_size = conv_net(test_input).size()[−1]
252             mlp = nn.Sequential(
253                 nn.Linear(conv_output_size, 200),
254                 nn.ReLU(),
255                 nn.BatchNorm1d(200),
256                 nn.Dropout(0.25),
257                 nn.Linear(200, 20)
258             )
259             net = nn.Sequential(
260                 conv_net,
261                 mlp
262             )
263             device_name = ”cpu”
264             net.to(device_name)
265             accuracy = train_net(net, dbLoader, queryLoader, n_iter=20, device=device_name)
266             return accuracy
267
268
269 # 評価ヘルパー
270 def eval_net(net, data_loader, device=”cpu”):
271     net.eval()
272     ys = []
273     ypreds = []
274     for x, y in data_loader:
275         x = x.to(device)
276         y = y.to(device)
```

```
            with torch.no_grad():
                _, y_pred = net(x).max(1)
            ys.append(y)
            ypreds.append(y_pred)
        # ミニバッチ毎の結果をまとめる
        ys = torch.cat(ys)
        ypreds = torch.cat(ypreds)
        acc = (ys == ypreds).float().sum() / len(ys)
        return acc.item()

# 訓練ヘルパー
def train_net(net, train_loader, test_loader,
                    optimizer_cls=optim.Adam, loss_fn=nn.CrossEntropyLoss(),
                    n_iter=10, device="cpu"):
        train_losses = []
        train_acc = []
        val_acc = []
        optimizer = optimizer_cls(net.parameters())
        for epoch in range(n_iter):
            running_loss = 0.0
            net.train()
            n = 0
            n_acc = 0
            for i, (xx, yy) in enumerate(tqdm(train_loader)):
                xx = xx.to(device)
                yy = yy.to(device)
                h = net(xx)
                loss = loss_fn(h, yy)
                optimizer.zero_grad()
                loss.backward()
                optimizer.step()
                running_loss += loss.item()
                n += len(xx)
                _, y_pred = h.max(1)
                n_acc += (yy == y_pred).float().sum().item()
            train_losses.append(running_loss / i)
            train_acc.append(n_acc / n)
            val_acc.append(eval_net(net, test_loader, device))
            print(epoch, train_losses[-1], train_acc[-1], val_acc[-1], flush=True)
        return(max(val_acc))
```

ソースコード 7: preprocessingOpenCV.py

```
# -*- coding: utf-8 -*-
import os
import cv2
import numpy as np
from pathlib import Path
from tqdm import tqdm

DBPath = '../input/default/dataset/DB/jpeg/'
QueryPath = '../input/default/dataset/Query/jpeg/'
SIZE = (128, 128)
# face_cascade = cv2.CascadeClassifier('haarcascades/haarcascade_' + type + '.xml')
face_cascade_frontalface_default = cv2.CascadeClassifier('haarcascades/haarcascade_frontalface_default.xml
    ')
face_cascade_frontalface_alt = cv2.CascadeClassifier('haarcascades/haarcascade_frontalface_alt.xml')
face_cascade_frontalface_alt2 = cv2.CascadeClassifier('haarcascades/haarcascade_frontalface_alt2.xml')
face_cascade_frontalface_alt_tree = cv2.CascadeClassifier('haarcascades/haarcascade_frontalface_alt_tree.
    xml')

def cutFace(type):
    DBSavePath = '../input/opencv/' + type + '/DB/jpeg/'
    QuerySavePath = '../input/opencv/' + type + '/Query/jpeg/'

    # Haar-like 特徴分類器の読み込み

    print('Start DB')
    # Database
    p = Path(DBPath)
```

```
    p = sorted(p.glob("*.jpg"))
    count = 0
    for filename in tqdm(p):
        img = cv2.imread(DBPath + filename.name)
        faces = findFace(filename.name, img)
        for (x,y,w,h) in faces:
            face = img[y:y+h, x:x+w]
            face = preProcessing(face)
        cv2.imwrite(DBSavePath + filename.name,face)
        count += 1
    print('Done DB')
    print('Start Query')
    # Query
    p = Path(QueryPath)
    p = sorted(p.glob("*.jpg"))
    count = 0
    for filename in tqdm(p):
        img = cv2.imread(QueryPath + filename.name)
        faces = findFace(filename.name, img)
        for (x,y,w,h) in faces:
            face = img[y:y+h, x:x+w]
            face = preProcessing(face)
        cv2.imwrite(QuerySavePath + filename.name,face)
        count += 1
    print('Done Query')

def findFace(name, img):
    faces = face_cascade_frontalface_alt_tree.detectMultiScale(img)
    if len(faces)==0:
        faces = face_cascade_frontalface_alt.detectMultiScale(img)
        if len(faces)==0:
            faces = face_cascade_frontalface_alt2.detectMultiScale(img)
            if len(faces)==0:
                faces = face_cascade_frontalface_default.detectMultiScale(img)
                if len(faces)==0:
                    print(name + ' not found')
    return faces

def preProcessing(img):
    img = cv2.resize(img, SIZE)
    # img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # #Sobel フィルタで x 方向のエッジ検出
    # gray_sobelx = cv2.Sobel(img,cv2.CV_32F,1,0)
    # #Sobel フィルタで y 方向のエッジ検出
    # gray_sobely = cv2.Sobel(img,cv2.CV_32F,0,1)
    # #8ビット符号なし整数変換
    # gray_abs_sobelx = cv2.convertScaleAbs(gray_sobelx)
    # gray_abs_sobely = cv2.convertScaleAbs(gray_sobely)
    # #重み付き和
    # img = cv2.addWeighted(gray_abs_sobelx,0.5,gray_abs_sobely,0.5,0)
    # img = (img - np.mean(img))/np.std(img)*32+100
    img = cv2.equalizeHist(img)
    # img = cv2.Canny(img,100,200)
    # img = cv2.GaussianBlur(img, (5, 5), 0)
    return img

# cutFace('frontalface_default')
# cutFace('frontalface_alt')
# cutFace('frontalface_alt2')

cutFace('cutface')
```

ソースコード 8: preprocessingDlib.py

```
# -*- coding: utf-8 -*-
import os
import cv2
import csv
import dlib
import imutils
```

```
import numpy as np
import pandas as pd
from pathlib import Path
from imutils import face_utils
from tqdm import tqdm

DBPath = '../input/default/dataset/DB/jpeg/'
QueryPath = '../input/default/dataset/Query/jpeg/'
SIZE = (128, 128)

FILENAME = 'histFlattening'

DBSavePath = '../input/Dlib/cutface/' + FILENAME + '/DB/jpeg/'
QuerySavePath = '../input/Dlib/cutface/' + FILENAME + '/Query/jpeg/'
DBPlotImagePath = '../input/Dlib/cutface/' + FILENAME + '/DBPlot/jpeg/'
QueryPlotImagePath = '../input/Dlib/cutface/' + FILENAME + '/QueryPlot/jpeg/'
DBPlotCSVPath = '../input/Dlib/cutface/' + FILENAME + '/DB/csv/'
QueryPlotCSVPath = '../input/Dlib/cutface/' + FILENAME + '/Query/csv/'

detector = dlib.get_frontal_face_detector()
predictor_path = "./libs/shape_predictor_68_face_landmarks.dat"
predictor = dlib.shape_predictor(predictor_path)

for path in [DBSavePath, QuerySavePath, DBPlotImagePath, \
                QueryPlotImagePath, DBPlotCSVPath, QueryPlotCSVPath]:
    if not os.path.exists(path):
        os.makedirs(path)

def cutFace():
    # Database
    print('Start DB')
    p = Path(DBPath)
    p = sorted(p.glob("*.jpg"))
    df = pd.DataFrame()
    target_df = pd.DataFrame()
    preFace = None
    for index, filename in enumerate(tqdm(p)):
        img = cv2.imread(DBPath + filename.name)
        face, plotImg, featurePoint_df = face_shape_detector_dlib(img, filename.name)
        df = pd.concat([df, featurePoint_df])
        if len(face)==0:
            face = preFace
            print('db error')
        preFace = face
        face = cv2.cvtColor(face, cv2.COLOR_BGR2GRAY)
        #face = cv2.equalizeHist(face)
        face = cv2.resize(face,SIZE)
        cv2.imwrite(DBSavePath + filename.name, face)
        cv2.imwrite(DBPlotImagePath + filename.name, plotImg)

        target = int(index / 10) + 1
        t_df = pd.DataFrame({'target' : target},
                            index = [filename.name])
        target_df = pd.concat([target_df, t_df])
    df = pd.concat([df, target_df], axis=1)
    df.to_csv(DBPlotCSVPath + 'featurePoint.csv')
    print('Done DB')

    print('Start Query')
    # Query
    p = Path(QueryPath)
    p = sorted(p.glob("*.jpg"))
    df = pd.DataFrame()
    target_df = pd.DataFrame()
    preFace = None
    for filename in tqdm(p):
        img = cv2.imread(QueryPath + filename.name)
        face, plotImg, featurePoint_df = face_shape_detector_dlib(img, filename.name)
        df = pd.concat([df, featurePoint_df])
        if len(face)==0:
            face = preFace
```

```
78                  print('query error')
79              preFace = face
80              face = cv2.cvtColor(face, cv2.COLOR_BGR2GRAY)
81              #face = cv2.equalizeHist(face)
82              face = cv2.resize(face,SIZE)
83              cv2.imwrite(QuerySavePath + filename.name, face)
84              cv2.imwrite(QueryPlotImagePath + filename.name, plotImg)
85              target = filename.name[0:2]
86              if target[0] == 'r':
87                  target = 0
88              else:
89                  target = int(target) + 1
90              t_df = pd.DataFrame({'target' : target},
91                              index = [filename.name])
92              target_df = pd.concat([target_df, t_df])
93      df = pd.concat([df, target_df], axis=1)
94      df.to_csv(QueryPlotCSVPath + 'featurePoint.csv')
95      print('Done Query')
96
97  # https://qiita.com/ufoo68/items/b1379b40ae6e63ed3c79
98  def face_shape_detector_dlib(img, name):
99      # presetting
100     img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
101     # frontal_face_detector クラスは矩形, スコア, サブ検出器の結果を返す
102     dets, scores, idx = detector.run(img_rgb, 0)
103
104     if len(dets) > 0:
105         for i, rect in enumerate(dets):
106             if i == 0:
107                 shape = predictor(img_rgb, rect)
108                 shape = face_utils.shape_to_np(shape)
109                 clone = img.copy()
110                 min_x, max_x, min_y, max_y = 1000, 0, 1000, 0
111                 for (x, y) in shape:
112                     if x < min_x:
113                         min_x = x
114                     if x > max_x:
115                         max_x = x
116                     if y < min_y:
117                         min_y = y
118                     if y > max_y:
119                         max_y = y
120                 face = img[min_y:max_y, min_x:max_x]
121                 # cv2.putText(clone, "mouth", (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0,
                        255), 2)
122                 # landmark を画像に書き込む
123                 # TODO
124                 # この辺をcsv に移植する作業(地獄)
125                 for (x, y) in shape[0:17]: # chin
126                     cv2.circle(clone, (x, y), 1, (0, 0, 0), -1)
127                 for (x, y) in shape[17:22]: # right eyebrow
128                     cv2.circle(clone, (x, y), 1, (255, 0, 0), -1)
129                 for (x, y) in shape[22:27]: # left eyebrow
130                     cv2.circle(clone, (x, y), 1, (255, 0, 255), -1)
131                 for (x, y) in shape[27:36]: # nose
132                     cv2.circle(clone, (x, y), 1, (0, 0, 255), -1)
133                 for (x, y) in shape[36:42]: # right eye
134                     cv2.circle(clone, (x, y), 1, (0, 255, 255), -1)
135                 for (x, y) in shape[42:48]: # left eye
136                     cv2.circle(clone, (x, y), 1, (255, 255, 0), -1)
137                 for (x, y) in shape[48:68]: # mouth
138                     cv2.circle(clone, (x, y), 1, (0, 255, 0), -1)
139
140                 df = pd.DataFrame()
141                 for i, (x, y) in enumerate(shape[0:68]): # all
142                     if i == 0:
143                         part = 'chin'
144                         count = 0
145                     elif i == 17:
146                         part = 'right_eyebrow'
147                         count = 0
```

```
                elif i == 22:
                    part = 'left_eyebrow'
                    count = 0
                elif i == 27:
                    part = 'nose'
                    count = 0
                elif i == 36:
                    part = 'right_eye'
                    count = 0
                elif i == 42:
                    part = 'left_eye'
                    count = 0
                elif i == 48:
                    part = 'mouth'
                    count = 0
                xy_df = pd.DataFrame({
                    part + '_x' + str(count) : [x],
                    part + '_y' + str(count) : [y]},
                    index = [name])
                df = pd.concat([df, xy_df], axis=1)
                count += 1
            # shape で指定した個所の切り取り画像(ROI)を取得
            # (x, y, w, h) = cv2.boundingRect(np.array([shape[48:68]])) #口の部位のみ切り出し
            # roi = img[y:y + h, x:x + w]
            # roi = cv2.resize(roi,(100,100))
        return face, clone, df
    else :
        print('error')
        return img, img, None



if __name__ == '__main__':
    cutFace()
    faceStart = 1
    faceEnd = 138

    chinColStart = 1
    chinColEnd = 35

    rightEyebrowStart = 35
    rightEyebrowEnd = 45

    leftEyebrowStart = 45
    leftEyebrowEnd = 55

    noseColStart = 55
    noseColEnd = 73

    rightEyeStart = 73
    rightEyeEnd = 85

    leftEyeStart = 85
    leftEyeEnd = 97

    mouseStart = 97
    mouseEnd = 138

    for mode in ['DB', 'Query']:
        INPUT_DIR = '../input/Dlib/cutface/' + FILENAME + '/' + mode + '/csv/'
        df = pd.read_csv(INPUT_DIR + "featurePoint.csv")

        df_out = pd.DataFrame(df['target'].copy())

        # face(顔)
        df_target = df.iloc[:, faceStart:faceEnd]
        df_target_x = df_target.iloc[:, 0:-1:2]
        df_out['face_width'] = df_target_x.max(axis=1) - df_target_x.min(axis=1)
        df_target_y = df_target.iloc[:, 1:-1:2]
        df_out['face_height'] = df_target_y.max(axis=1) - df_target_y.min(axis=1)
```

```
219
220         # chin(アゴ)
221         df_target = df.iloc[:, chinColStart:chinColEnd]
222         df_target_x = df_target.iloc[:, 0:-1:2]
223         df_out['chin_width'] = df_target_x.max(axis=1) - df_target_x.min(axis=1)
224         df_target_y = df_target.iloc[:, 1:-1:2]
225         df_out['chin_height'] = df_target_y.max(axis=1) - df_target_y.min(axis=1)
226
227         # right eye brow(右マユ)
228         df_target = df.iloc[:, rightEyebrowStart:rightEyebrowEnd]
229         df_target_x = df_target.iloc[:, 0:-1:2]
230         df_out['right_Eyebrow_width'] = df_target_x.max(
231             axis=1) - df_target_x.min(axis=1)
232         df_target_y = df_target.iloc[:, 1:-1:2]
233         df_out['right_Eyebrow_height'] = df_target_y.max(
234             axis=1) - df_target_y.min(axis=1)
235
236         # left eye brow(左マユ)
237         df_target = df.iloc[:, leftEyebrowStart:leftEyebrowEnd]
238         df_target_x = df_target.iloc[:, 0:-1:2]
239         df_out['left_Eyebrow_width'] = df_target_x.max(
240             axis=1) - df_target_x.min(axis=1)
241         df_target_y = df_target.iloc[:, 1:-1:2]
242         df_out['left_Eyebrow_height'] = df_target_y.max(
243             axis=1) - df_target_y.min(axis=1)
244
245         # nose(鼻)
246         df_target = df.iloc[:, noseColStart:noseColEnd]
247         df_target_x = df_target.iloc[:, 0:-1:2]
248         df_out['nose_width'] = df_target_x.max(axis=1) - df_target_x.min(axis=1)
249         df_target_y = df_target.iloc[:, 1:-1:2]
250         df_out['nose_height'] = df_target_y.max(axis=1) - df_target_y.min(axis=1)
251         df['nose_width_center'] = df_target_x.mean(axis=1)
252         df['nose_height_center'] = df_target_y.mean(axis=1)
253
254         # right eye(右目)
255         df_target = df.iloc[:, rightEyeStart:rightEyeEnd]
256         df_target_x = df_target.iloc[:, 0:-1:2]
257         df_out['right_Eye_width'] = df_target_x.max(
258             axis=1) - df_target_x.min(axis=1)
259         df_target_y = df_target.iloc[:, 1:-1:2]
260         df_out['right_Eye_height'] = df_target_y.max(
261             axis=1) - df_target_y.min(axis=1)
262         df['right_Eye_width_center'] = df_target_x.mean(axis=1)
263         df['right_Eye_height_center'] = df_target_y.mean(axis=1)
264
265         # left eye(左目)
266         df_target = df.iloc[:, leftEyeStart:leftEyeEnd]
267         df_target_x = df_target.iloc[:, 0:-1:2]
268         df_out['left_Eye_width'] = df_target_x.max(
269             axis=1) - df_target_x.min(axis=1)
270         df_target_y = df_target.iloc[:, 1:-1:2]
271         df_out['left_Eye_height'] = df_target_y.max(
272             axis=1) - df_target_y.min(axis=1)
273         df['left_Eye_width_center'] = df_target_x.mean(axis=1)
274         df['left_Eye_height_center'] = df_target_y.mean(axis=1)
275
276         # mouse(口)
277         df_target = df.iloc[:, mouseStart:mouseEnd]
278         df_target_x = df_target.iloc[:, 0:-1:2]
279         df_out['mouse_width'] = df_target_x.max(axis=1) - df_target_x.min(axis=1)
280         df_target_y = df_target.iloc[:, 1:-1:2]
281         df_out['mouse_height'] = df_target_y.max(axis=1) - df_target_y.min(axis=1)
282         df['mouse_width_center'] = df_target_x.mean(axis=1)
283         df['mouse_height_center'] = df_target_y.mean(axis=1)
284
285         # relative distance(各部位の相対的な距離)
286         df_out['eye2eye_dist'] = np.sqrt((df['right_Eye_width_center'] - df['left_Eye_width_center'])**2 +
287                                          (df['right_Eye_height_center'] - df['left_Eye_height_center
                                                 '])**2)
288         df_out['Reye2nose_dist'] = np.sqrt((df['right_Eye_width_center'] - df['nose_width_center'])**2 +
```

```
                                                (df['right_Eye_height_center'] - df['nose_height_center'])**2)
        df_out['Leye2nose_dist'] = np.sqrt((df['left_Eye_width_center'] - df['nose_width_center'])**2 +
                                                (df['left_Eye_height_center'] - df['nose_height_center'])**2)
        df_out['nose2mouse_dist'] = np.sqrt((df['nose_width_center'] - df['mouse_width_center'])**2 +
                                                (df['nose_height_center'] - df['mouse_height_center'])**2)
        df_out['nose2mouse_dist'] = np.sqrt((df['nose_width_center'] - df['mouse_width_center'])**2 +
                                                (df['nose_height_center'] - df['mouse_height_center'])**2)
        df_out['Reye2mouse_dist'] = np.sqrt((df['right_Eye_width_center'] - df['mouse_width_center'])**2
            +
                                                (df['right_Eye_height_center'] - df['mouse_height_center
                                                    '])**2)
        df_out['Leye2mouse_dist'] = np.sqrt((df['left_Eye_width_center'] - df['mouse_width_center'])**2
            +
                                                (df['left_Eye_height_center'] - df['mouse_height_center
                                                    '])**2)

        for i in range(df_out.shape[0]):
            df_out.iloc[i, 1:] = df_out.iloc[i, 1:]/df_out['face_width'][i]
        df_out.drop('face_width', axis=1, inplace=True)

        df_out.to_csv(INPUT_DIR + "features_rel_dist.csv")
```

ソースコード 9: DCT.py

```
import numpy as np
import pandas as pd
import cv2
from scipy.fftpack import dct
from pathlib import Path
from time import sleep
from termcolor import colored
from tqdm import tqdm

hist = 'default'
FILENAME = '../input/Dlib/cutface/'

dbPath = FILENAME + hist + '/DB/jpeg/'
queryPath = FILENAME + hist + '/Query/jpeg/'
DBCannySavePath = '../dct/DB/canny/jpeg/'
QueryCannySavePath = '../dct/Query/canny/jpeg/'
DBDctSavePath = '../dct/DB/dctAfterCanny/jpeg/'
QueryDctSavePath = '../dct/Query/dctAfterCanny/jpeg/'
DBDctOriginalSavePath = '../dct/DB/dctOriginal/jpeg/'
QueryDctOriginalSavePath = '../dct/Query/dctOriginal/jpeg/'
best_score = 0

if __name__ == '__main__':
    #with open("dct.csv", mode='w') as f:
    dbList = []
    queryList = []
    distanceList = []
    answerList = []
    count = 0
    bestAcc = 0

    p = Path(dbPath)
    p = sorted(p.glob("*.jpg"))
    feature_name_list = []
    for i in range(225):
        featrue_name = 'dct_' + str(i)
        feature_name_list.append(featrue_name)
    db_df = pd.DataFrame(columns=feature_name_list)
    for index, filename in enumerate(tqdm(p)):
        img = cv2.imread(dbPath + filename.name, 0)

        # img = cv2.GaussianBlur(img, (7, 7), 2)
        # img = cv2.Canny(img, 13, 39)

        imf = np.float32(img)
        dst = cv2.dct(imf)
        dst = dst[:15,:15]
```

```
48              for ind, name in enumerate(feature_name_list):
49                  db_df.loc[index, name] = np.array(dst).flatten()[ind]
50
51              dbList.append(np.float32(dst))
52      p = Path(queryPath)
53      p = sorted(p.glob("*.jpg"))
54      query_df = pd.DataFrame(columns=feature_name_list)
55      for index, filename in enumerate(tqdm(p)):
56              distanceList = []
57              img = cv2.imread(queryPath + filename.name, 0)
58
59              # img = cv2.GaussianBlur(img, (7, 7), 2)
60              # img = cv2.Canny(img, 13, 39)
61
62              imf = np.float32(img)
63              dst = cv2.dct(imf)
64              dst = dst[:15,:15]
65              for ind, name in enumerate(feature_name_list):
66                  query_df.loc[index, name] = np.array(dst).flatten()[ind]
67
68              queryList.append(np.float32(dst))
69              for index in range(len(dbList)):
70                  distance = (dbList[index][:15, :15] - queryList[-1][:15, :15])**2
71                  distanceList.append(distance.sum())
72              distSort = np.argsort(distanceList)
73              answerList.append(np.uint8(np.argmin(distanceList)/10))
74              ranking = np.uint8(distSort[:10]/10)
75              mode = np.argmax(np.bincount(ranking))
76
77              target = filename.name[0:2]
78              if target[0] == 'r':
79                  target = 20
80              else:
81                  target = int(target)
82              # 結果に応じて色を付ける
83              if target == mode :
84                  print(colored(filename.name + ', mode: ' + str(mode) + ', answer: ' + str(answerList[-1]), '
                        blue'))
85              if target == answerList[-1]:
86                  count += 1
87                  print(colored(filename.name + ', mode: ' + str(mode) + ', answer: ' + str(answerList[-1]) + ',
                        correct answer: ' + str(target), 'blue'))
88              else:
89                  print(colored(filename.name + ', mode: ' + str(mode) + ', answer: ' + str(answerList[-1]) + ',
                        correct answer: ' + str(target), 'red'))
90              distanceList = []
91
92      if db_df.min().min() < query_df.min().min():
93              norm_min = db_df.min().min()
94      else:
95              norm_min = query_df.min().min()
96
97      db_df = db_df - norm_min
98      query_df = query_df - norm_min
99
100     if db_df.max().max() > query_df.max().max():
101             norm_max = db_df.max().max()
102     else:
103             norm_max = query_df.max().max()
104
105     db_df = db_df / norm_max
106     query_df = query_df / norm_max
107
108     p = Path(dbPath)
109     p = sorted(p.glob("*.jpg"))
110     for index, filename in enumerate(tqdm(p)):
111             db_df.loc[index, 'target'] = int(index/10)
112     db_df.to_csv("../input/dct/noCanny/" + hist + "/db_list.csv")
113
114     p = Path(queryPath)
115     p = sorted(p.glob("*.jpg"))
```

```
    for index, filename in enumerate(tqdm(p)):
        target = filename.name[0:2]
        if target[0] == 'r':
            target = 0
        else:
            target = int(target)
        query_df.loc[index, 'target'] = target
    query_df.to_csv("../input/dct/noCanny/" + hist + "/query_list.csv")

    accuracy = count / len(queryList) * 100
    print(str(accuracy) + '%')
```

ソースコード 10: final_gui.py

```
# -*- coding: utf-8 -*-
"""
Created on Sun Jan 20 10:47:10 2019

@author: kumac
"""

import sys, os
from PyQt5.QtWidgets import (QApplication, QWidget, QPushButton, QHBoxLayout,
                             QVBoxLayout, QLabel, QLineEdit, QFileDialog,
                             QProgressBar, QComboBox)
from PyQt5.QtGui import QPixmap
from PyQt5.QtCore import Qt, QCoreApplication
from pathlib import Path

import numpy as np
import cv2

from processes import processes

class Example(QWidget):

    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        # 変数
        self.isFolder = False
        self.processNum = 0
        self.tableProcess = ["単純マッチング", "kNN", "NeuralNet", "LightGBM"]
        self.imageProcess = ["ピクセルマッチング", "CNN"]
        self.processDict = {"単純マッチング":0, "kNN":1, "NeuralNet":2, "LightGBM":3, "ピクセルマッチング":4, "CNN":5}
        self.lblEmpty = QLabel("\n", self)
        self.isChanged = False
        self.prefixAns = "<h3>Accuracy :"

        # ウィンドウサイズ
        self.resize(500, 400)
        self.setWindowTitle('face classification')

        # メインの配置
        self.vboxMain = QVBoxLayout(self)

        # 手順 1(データ選択)
        self.vboxOpe1 = QVBoxLayout(self)
        # ラベルの用意
        self.lbl1 = QLabel(self)
        self.lbl1.setText("<h2>1.処理対象のデータを選んでください")
        self.vboxOpe1.addWidget(self.lbl1)
        # combo box でテーブルデータか画像データのフォルダを選ぶか決める
        self.intype_combo = QComboBox(self)
        self.intype_combo.addItems(["csv ファイル", "画像フォルダ (.jpg)"])
        self.intype_combo.activated[str].connect(self.intypeActivated)
        self.vboxOpe1.addWidget(self.intype_combo)
```

```
56          # 入力テキストフォルダとボタンの用意
57          self.hboxDB = QHBoxLayout(self)
58          self.lblDB = QLabel(self)
59          self.lblDB.setText("DB Path:")
60          self.dbPath = QLineEdit(self)
61          self.DBFolder = QPushButton('参照')
62          self.DBFolder.clicked.connect(lambda: self.showFolderDialog(self.dbPath))
63          self.hboxDB.addWidget(self.lblDB)
64          self.hboxDB.addWidget(self.dbPath)
65          self.hboxDB.addWidget(self.DBFolder)
66          self.vboxOpe1.addLayout(self.hboxDB)
67
68          self.hboxQuery = QHBoxLayout(self)
69          self.lblQuery = QLabel(self)
70          self.lblQuery.setText("Query Path:")
71          self.queryPath = QLineEdit(self)
72          self.queryFolder = QPushButton('参照')
73          self.queryFolder.clicked.connect(lambda: self.showFolderDialog(self.queryPath))
74          self.hboxQuery.addWidget(self.lblQuery)
75          self.hboxQuery.addWidget(self.queryPath)
76          self.hboxQuery.addWidget(self.queryFolder)
77          self.vboxOpe1.addLayout(self.hboxQuery)
78
79          # 手順 2(処理方法選択)
80          self.vboxOpe2 = QVBoxLayout(self)
81          # ラベルの用意
82          self.lbl2 = QLabel(self)
83          self.lbl2.setText("<h2>2.処理方法を選んでください")
84          self.vboxOpe2.addWidget(self.lblEmpty)
85          self.vboxOpe2.addWidget(self.lbl2)
86          # bombo box で処理方法を選択
87          self.process_combo = QComboBox(self)
88          self.process_combo.addItems(self.tableProcess)
89          self.process_combo.activated[str].connect(self.processActivated)
90          self.vboxOpe2.addWidget(self.process_combo)
91
92          # 手順 3(実行)
93          self.vboxOpe3 = QVBoxLayout(self)
94          # ラベルの用意
95          self.lbl3 = QLabel(self)
96          self.lbl3.setText("<h2>3.実行")
97          self.vboxOpe3.addWidget(self.lblEmpty)
98          self.vboxOpe3.addWidget(self.lbl3)
99          # 実行ボタン
100         self.btnExec = QPushButton("実行")
101         self.btnExec.clicked.connect(self.btnExecContent)
102         self.vboxOpe3.addWidget(self.btnExec)
103
104         self.lblAccu = QLabel(self)
105         self.lblAccu.setText(self.prefixAns)
106         self.lblAccu.setAlignment(Qt.AlignCenter)
107         self.vboxOpe3.addWidget(self.lblEmpty)
108         self.vboxOpe3.addWidget(self.lblAccu)
109
110
111         self.vboxMain.addLayout(self.vboxOpe1)
112         self.vboxMain.addLayout(self.vboxOpe2)
113         self.vboxMain.addLayout(self.vboxOpe3)
114         self.vboxMain.addWidget(self.lblAccu)
115         self.vboxMain.setAlignment(Qt.AlignTop)
116
117         self.show()
118
119
120     def showFolderDialog(self, path):
121         self.isChanged = True
122         if(self.isFolder):
123             dirname = QFileDialog.getExistingDirectory(self,
124                                                        'open folder',
125                                                        os.path.expanduser('../'),
```

```
                                                        QFileDialog.ShowDirsOnly)
            if dirname:
                self.dirname = dirname.replace('/', os.sep)

        else:
            dirname = QFileDialog.getOpenFileName(self,
                                                  'open folder/file',
                                                  os.path.expanduser('../'))
            if dirname:
                self.dirname = dirname[0].replace('/', os.sep)
        path.setText(self.dirname)


    def intypeActivated(self, text):
        if (text == "画像フォルダ (.jpg)"):
            self.isFolder = True
            self.process_combo.clear()
            self.process_combo.addItems(self.imageProcess)
        else:
            self.isFolder = False
            self.process_combo.clear()
            self.process_combo.addItems(self.tableProcess)


    def processActivated(self, text):
        self.processNum = self.processDict[text]


    def btnExecContent(self):
        if(self.isChanged):
            self.model = processes(self.dbPath.text(), self.queryPath.text(), self.isFolder)
        accu = self.model.calc_accuracy(self.processNum)
        self.lblAccu.setText("{}{:.2%}".format(self.prefixAns, accu))


if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())
```