顔認識

目次

目的

- Python環境でOSSライブラリを使用した顔認識を実装する。
- 顔認識パラメータのチューニングによる分析結果の差異を確認し、最適な値を抽出する。
- 顔認識結果を画面に表示し、結果を視覚的に理解する。

第一章　背景

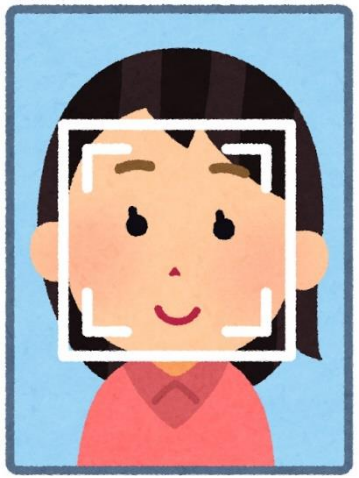
１．「画像認識」って何だろう

- 画像の中に一体何が写っているのか、コンピューターや機械などが識別する技術です。

以下のようなシチュエーションで、知らずに画像認識技術が使用されております。

- 製造業の不良品検知
- 医療の画像診断
- 自動運転
- 空港の顔認証システム

など



第一章　背景

- ２．画像認識に使用されている技術

　画像認識の技術には、「物体認識」、「顔認識」、「文字認識」があります。

今回は、その一つである「顔認識」をテーマに取り上げます。

　「顔認識」とは、画像の中にある顔と思われる部分を抜き出し、顔のデータベースと照合することで特定の人物を識別するシステムのことです。

第一章　背景

- ３．「顔認識」を体験するための環境
　　顔認識を行うためには、画像認識に有効なライブラリが揃っているPythonを使用します。
　　PythonをWebブラウザ上で記述・実行できる総合開発環境ツールとして、「Jupyter Notebook」があります。
　　「Jupyter Notebook」を含む様々なデータサイエンスパッケージが付属したPythonディストリビューションの「Anaconda」を使用しています。

第一章　背景

「Anaconda」を使用するメリットには下記があります。

・無料でインストールし、使用することができる。
・ データ分析や人工知能で使う、ライブラリやツールなどが豊富にある。
・「Python」のバージョン変化が頻繁な機械学習分野において、バージョンの違いを試しながら実装できる環境を提供してくれる。
・インストール時の指定により、「Jupyter Notebook」も同時にインストールできる。

第一章　背景

- ４．「顔認識」を体験するための環境

　顔認識するためには、以下のような処理が必要となってくると考えられます。

・画像ファイルの入出力
・顔認証
・画像の表示や加工
・数値計算

これらの処理を行うためには、どのようなライブラリが必要でしょうか？

第一章　背景

各処理に必要なライブラリについて説明します。

・画像ファイルの入出力ライブラリとして「OpenCV」を使用します。
　コンピューターで画像や動画を処理するための機能がまとめて実装されている、ライブラリのことです。
　その他の入出力ライブラリとして「Matplotlib」があります。
「Matplotlib」はグラフ描画に有効ですが、画像出力にも使用されます。

・顔認証ライブラリとして「face recognition」を使用します。
　詳細については、第五章以降で解説して参ります。

第一章　背景

・高い精度の顔認証を実施するために「Dlib」を使用します。
　「Dlib」は、「face recognition」のコアとなるライブラリです。

第二章　OpenCVの基礎的な使用方法

１．OpenCVの使い方、画像処理後の表示方法

顔認識では、OpenCVの以下の機能を使用します。

・画像の読み込み・表示

・画像の作成・保存

・図形の描画・文字の描画

第二章　OpenCVの基礎的な使用方法

顔を認識したい画像ファイルを読み込みます。
画像ファイルの中に顔と認識する部分を探し、枠で囲みます。
画像ファイルの中で、顔と認識した場所の位置情報を描き込みます。

第三章　顔認証ライブラリ face recognitionの説明

１．face recognitionの概要

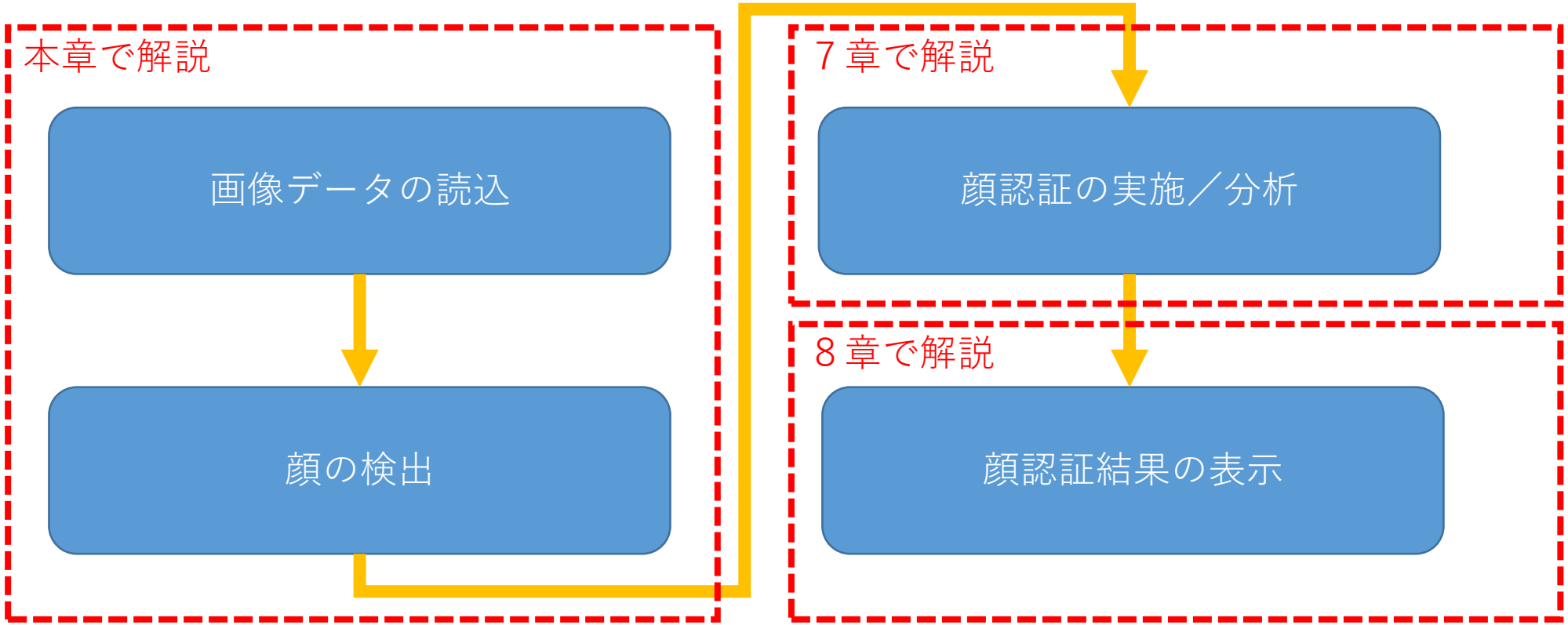
face recognitionとは、Pythonで使用する顔認識ライブラリです。

コマンドラインから顔を認識して操作する世界で最も単純な顔認識ライブラリで、Dlibの最先端の顔認識を使用して深層学習されています。

個人の顔データと認証中の顔の画像を照合し、その人物が誰であるかを識別することができるため、本教材では、face recognitionを使用します。

第四章　各顔認証ライブラリを組み合わせた使用方法

本章から、OpenCVとface_recognitionライブラリを組み合わせた顔認証の方法を紹介します。処理の流れは以下の通りです。



第四章　各顔認証ライブラリを組み合わせた使用方法

実際に画像を使って、動作検証を行います。

１．画像データの読込

画像データの読込が可能なAPIは、さまざまなライブラリで用意されています。

本例では、OpenCVライブラリを活用して画像データを読込ます。

```
# ライブラリのインポート
import cv2                    #ライブラリを使用するときは必ずimportでライブラリのインポートが必要です
# 画像データの読込
Image = cv2.imread(r“画像ファイル(フルパス)”)

# 色情報をBGR形式(青,緑,赤)から、RGB形式(赤,緑,青)に変換
Image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

色情報の形式変換はなぜ必要？

後述する顔の抽出処理では、 face_recognitionのAPIを使用します。このAPIではRGB形式の色情報を持つ画像とみなして処理を行います。このため、本例では色情報の変換処理が必要となります。

第四章　各顔認証ライブラリを組み合わせた使用方法

２．顔データの抽出

顔認証を行うためには、画像データにある顔を認識して抽出する必要があります。

これも様々なライブラリで顔データ抽出用のAPIが用意されておりますが、

本例では、 face_recognitionを使い、顔データを抽出します。

```
# ライブラリのインポート
import face_recognition

# 顔データの抽出
face_locations = face_recognition.face_locations(Image, model=“hog")
```

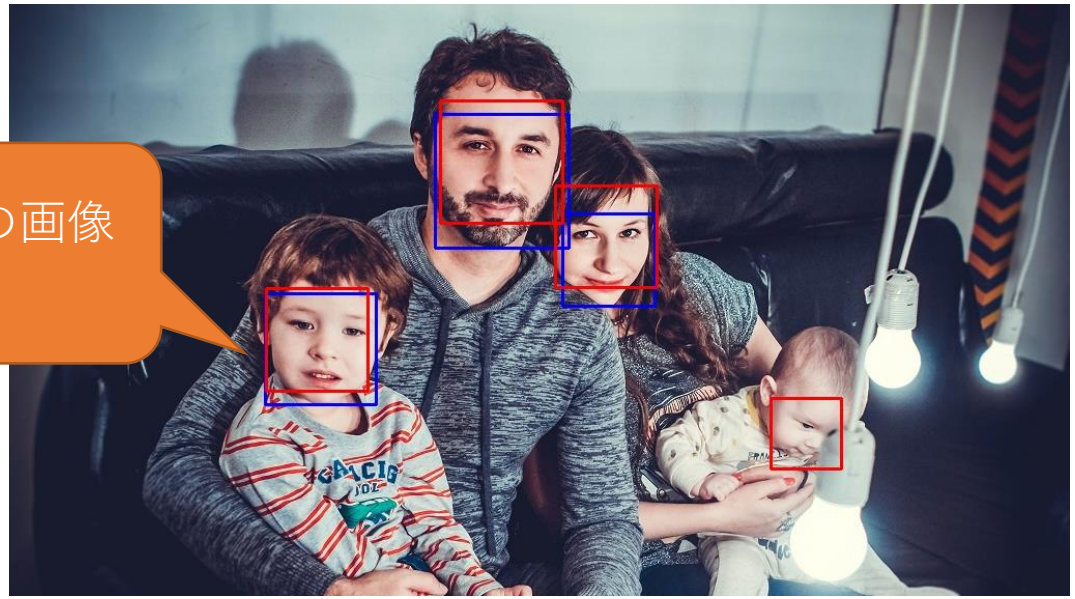
第四章　各顔認証ライブラリを組み合わせた使用方法

３．チューニング方法

以下の画像をHOGモデルで抽出してみましょう。

```
# 顔データの抽出
face_locations = face_recognition.face_locations(Image, model="hog")
```

hogの枠(青)のみの画像と差し替え



■出典

写真のフリー素材サイト - Pexels

https://www.pexels.com/ja-jp/

結果、3人の顔までは検出しますが、1人は検出できていません。

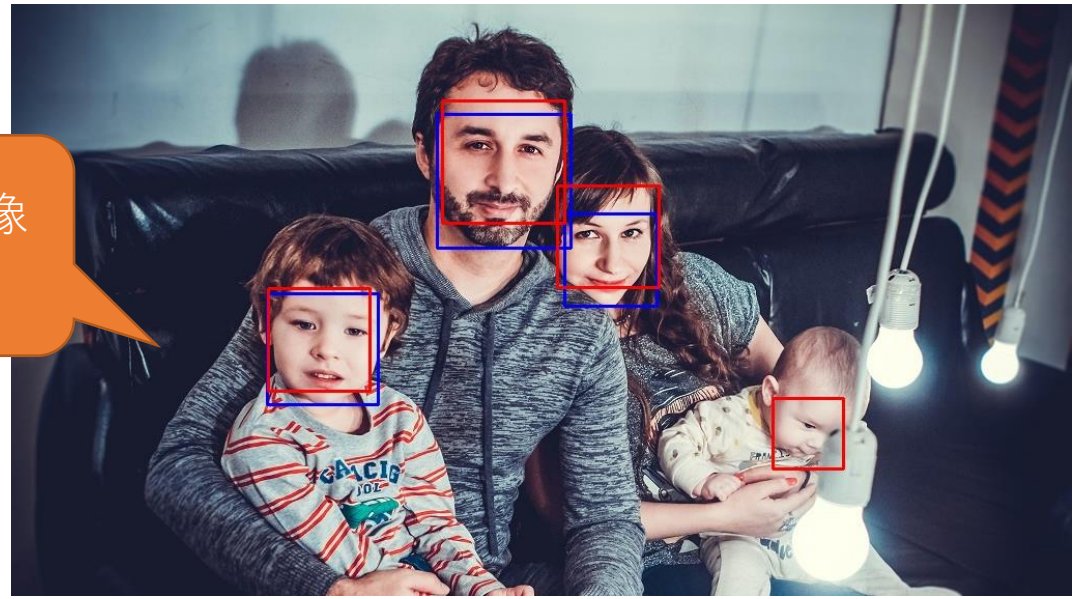
第四章　各顔認証ライブラリを組み合わせた使用方法

３．チューニング方法

次にCNNモデルで抽出してみましょう。

```
# 顔データの抽出
face_locations = face_recognition.face_locations(Image, model=“cnn")
```

cnnの枠(赤)のみの画像と差し替え



結果、すべての人物の顔を検出することができるようになりました。

第四章　各顔認証ライブラリを組み合わせた使用方法

３．チューニング方法

以上の結果から、HOGモデルよりCNNモデルの方が精度が高いことがわかります。
前章で解説した通り、CNNモデルはアップサンプリング回数を増加させることで
さらに高精度で顔認識を行うことができます。

```
# 顔データの抽出
face_locations = face_recognition.face_locations(Image, model="cnn" , number_of_times_to_upsample=1,2,3…)
```

ただし、制度を上げると処理に要する時間が増加し、レスポンスが低下します。
実際にアップサンプリング回数(number_of_times_to_upsample)を調整して
どれくらいレスポンスが低下するか試してみましょう。

第五章　顔認証結果の分析

１．検出した顔の認証方法
まず、検出した顔の特徴をエンコードします。
登録画像と認識させる画像の２つに対して行います。
・エンコード情報 = **face_recognition.face_encodings**(画像, 検出した顔)

エンコードした２つの顔の特徴を比較して、認証結果(TRUE/FALSE)を出力します。
閾値はオプション指定で、デフォルトは 0.6 です。低くすると判定が厳しくなります。
・認証結果 = **face_recognition.compare_faces**
(エンコード情報(認識させる画像), エンコード情報(登録画像), 閾値)

認証値を取得する場合は、以下の関数を使用します。
0 に近いと顔が似ていると判断したことになります。
・認証値 = **face_recognition.face_distance**
(エンコード情報(認識させる画像), エンコード情報(登録画像))

第五章　顔認証結果の分析

2. プログラム作成

では実際に、ここまで解説したライブラリ関数を使って、
顔認証させるPythonプログラムを動かしてみます。

まずは全てデフォルト値のパラメータで顔認証を行い、
複数の顔写真を使い、前ページの認証結果と認証値を確認します。

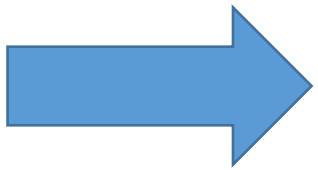
第五章　顔認証結果の分析

3. 結果数値の説明

先述の通り、認証値は0に近いほど顔が似ていると判断したことになります。
今回は1枚の認識させる画像(家族写真)に対し、
登録画像として ①父親 ②母親 ③子供 の
3パターンの顔写真を顔認証させました。

①父親　②母親　③子供





※赤ん坊パターンは、第六章にあった通り、
顔を検出できなかったこと、
後述の子供の顔認証結果からスキップしました。

第五章　顔認証結果の分析

認識結果

	認識させる画像		
登録画像	父親	母親	子供
①父親	**True**	False	False
②母親	False	**True**	False
③子供	False	**True**	**True**

認識率

	認識させる画像		
登録画像	父親	母親	子供
①父親	**0.398**	0.804	0.708
②母親	0.836	**0.414**	0.601
③子供	0.765	**0.591**	**0.359**

父親と母親の認識結果はTrueとなっており、認識率も閾値を大きく下回っています。
しかし、子供は母親もTrueとなっており、認識率が0.591と閾値を微妙に下回ってしまいました・・。

なぜ子供はデフォルトの閾値 0.6 では正しい顔認識ができなかったのでしょうか？

第五章　顔認証結果の分析

その理由は Face Recognition ライブラリの以下の警告にありました。

大人の顔向けに学習されているため、子供の顔をうまく認識させるには、閾値をさらに厳しく 0.6 未満にチューニングすれば良いようです。

Caveats
The face recognition model is trained on adults and does not work very well on children. It tends to mix up children quite easy using the default comparison threshold of 0.6.

第五章　顔認証結果の分析

4．パラメータチューニングによる検討

デフォルトパラメータでは子供の顔を正しく認識できませんでした。
また、第六章の家族写真では一部の顔を検出できませんでした。

そこで、これまでの家族写真を含む下記4枚の「認識させる画像」に対し、以下のパラメータを変えて結果の違いを比較して、チューニングの有効性と、ベストな閾値を分析してみました。

・モデル：hog, cnn
・アップサンプリング回数：1, 2

写真①

写真②

写真③

写真④

第五章　顔認証結果の分析

まず、4枚の家族写真の顔検出の結果は以下の通りです。
モデルが hog では、一部の顔検出に失敗してしまいました。
アップサンプリング回数を上げると若干改善しますが効果は薄いようです。

モデル	アップサンプリング回数	認識させる画像			
		写真①	写真②	写真③	写真④(*1)
hog	1	×(*2)	○	○	×(*3)
	2	×(*2)	○	○	×(*4)
cnn	1	○	○	○	○
	2	○	○	○	○

(*1)

(*2) 赤ん坊の顔を検出できませんでした。
(*3) 母親と子供の顔を検出できませんでした。
(*4) 母親の顔を検出できませんでした。

第五章　顔認証結果の分析

次に、顔認結果は以下の通りです。

登録画像が父親、母親の場合、いずれの写真も正しい認識ができました。

いっぽう子供の場合、モデルやアップサンプリング回数を上げても、写真②③は母親もTrueになってしまいました。

モデル	アップサンプリング回数	登録画像(写真①)	認識させる画像								
			写真②			写真③			写真④		
			父親	母親	子供	父親	母親	子供	父親	母親	子供
hog	1	①父親	True	False	False	True	False	False	True	-	-
		②母親	False	True	False	False	True	False	False	-(*)	-
		③子供	False	True	True	False	True	True	False	-	-(*)
	2	①父親	True	False	False	True	False	False	True	-	False
		②母親	False	True	False	False	True	False	False	-(*)	False
		③子供	False	True	True	False	True	True	False	-	True
cnn	1	①父親	True	False	False	True	False	False	True	False	False
		②母親	False	True	False	False	True	False	False	True	False
		③子供	False	True	True	False	True	True	False	False	True
	2	①父親	True	False	False	True	False	False	True	False	False
		②母親	False	True	False	False	True	False	False	True	False
		③子供	False	True	True	False	True	True	False	False	True

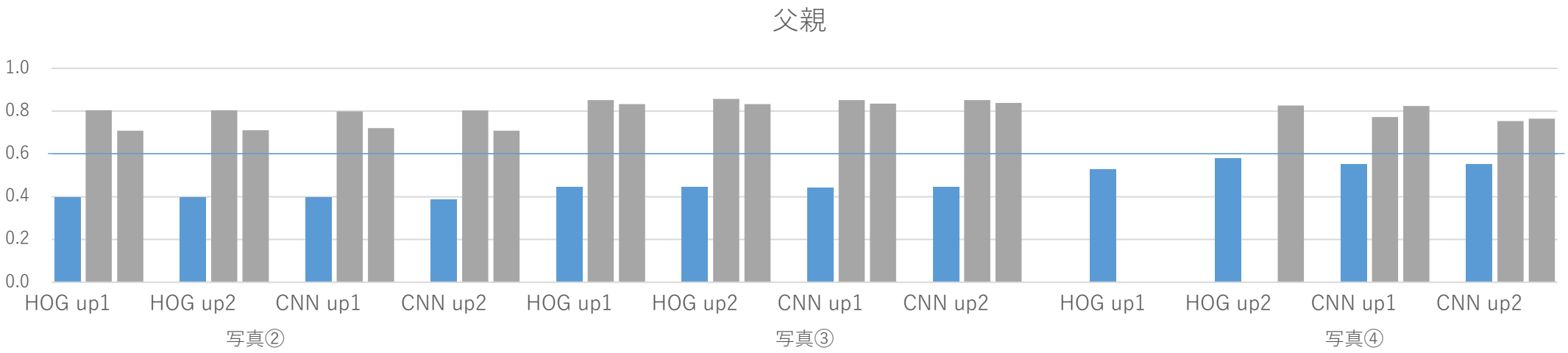
(*) 自分の顔が検出できておらず、写っていない判定に留まりました。

第五章　顔認証結果の分析

各登録画像ごとに認証値をグラフ化してみました。

父親はデフォルトの閾値0.6でちょうど良いようです。

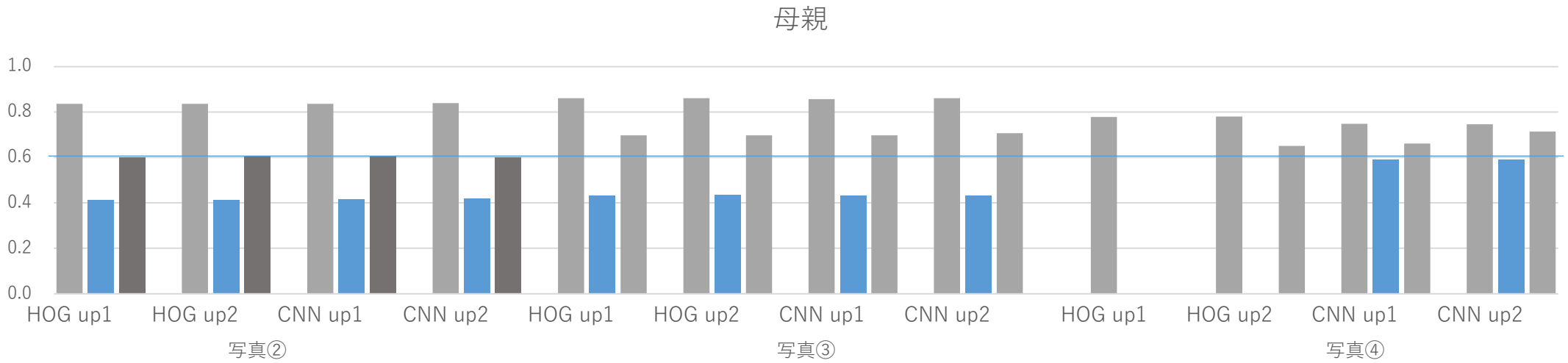
いっぽう、モデルやアップサンプリング回数による違いはあまり見られません。



第五章　顔認証結果の分析

母親のグラフです。写真②の子供の認識値がわずかに0.6より上と、ギリギリのFalse判定でした。いっぽう、写真④は自身の認識値が0.6弱でしたので、結果的には閾値0.6のままで良かったことになります。

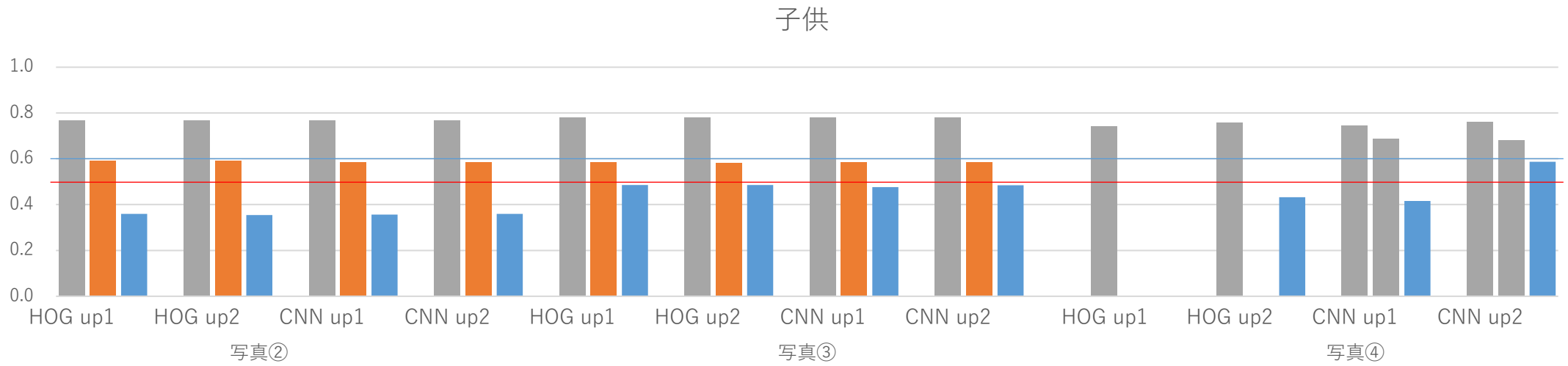
なお、父親同様にモデルやアップサンプリング回数による違いはあまり見られません。



第五章　顔認証結果の分析

子供のグラフです。閾値を0.5(赤線)まで下げていれば、写真②③の母親をFlaseにできたことがわかります。が、今度は写真④の cnn＋アップサンプリング回数2の自身の顔がFalseになってしまます・・。

ここだけアップサンプリング回数による違いが出ていることからも、大人の顔向けに学習されている影響なのかもしれません。



第六章　顔認証結果の画面表示

OpenCVを使って画像を加工(描画)したのち出力します。

・長方形描画関数を使って顔を枠で囲みます。
cv2.rectangle(画像, 左上座標, 右下座標, 枠の色, 枠線の太さ(*))
(*)マイナス値を指定すると内側を塗りつぶします。

・文字描画関数を使って認証結果などを描画します。
cv2.putText(画像, str(文字), 座標, フォント, サイズ, 色, 太さ)

・画像を出力します。

```
cv2.imwrite(“出力パス名”, 画像)        # 画像を保存する場合
cv2.imshow(“image”, 画像)              # ウィンドウ表示
cv2.waitKey(0)                         # キー入力待ち
cv2.destroyAllWindows()                # ウィンドウを閉じる
```

第六章　顔認証結果の画面表示

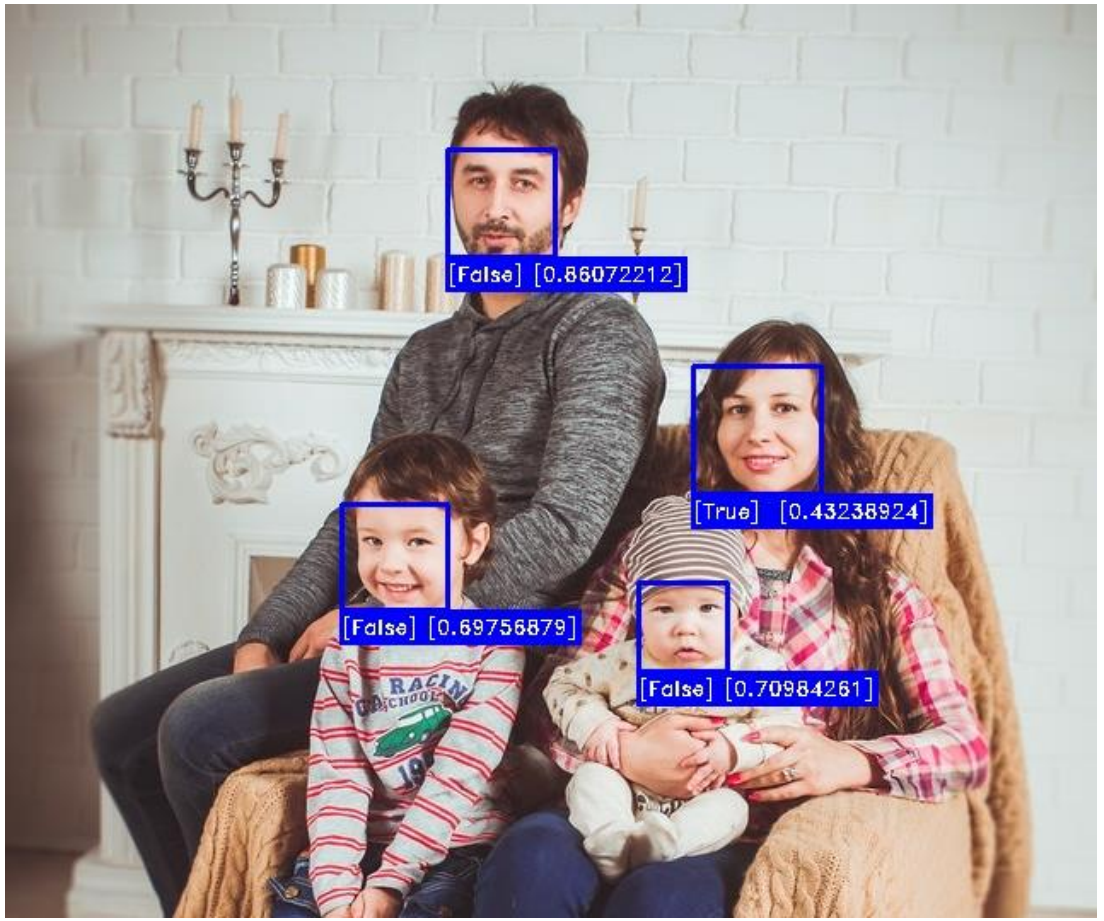
OpenCVの代わりにMatplotlibを使って、
グラフの描画領域に画像を表示するやり方もあります。

```
subplots(figsize=(サイズ))　# グラフのサイズ指定
imshow(画像)　　　　　　　　# グラフの描画領域に画像を表示
set_axis_off()　　　　　　　# グラフの軸や目盛をOFFにする
savefig(“出力パス名”)　　　# 画像を保存する場合
```

第六章　顔認証結果の画面表示

登録画像(母親)

認識させる画像



・検出した顔を枠で囲むことができています。
・顔認識結果と認識値を描画できています。

第七章　課題（カスタマイズ）

【課題①】：難易度低

顔認識結果をTrue／Falseではなく、True→名前※,　False→Nobodyとして名前表記に変更してください。※例：子供→SAM

<対応ヒント>

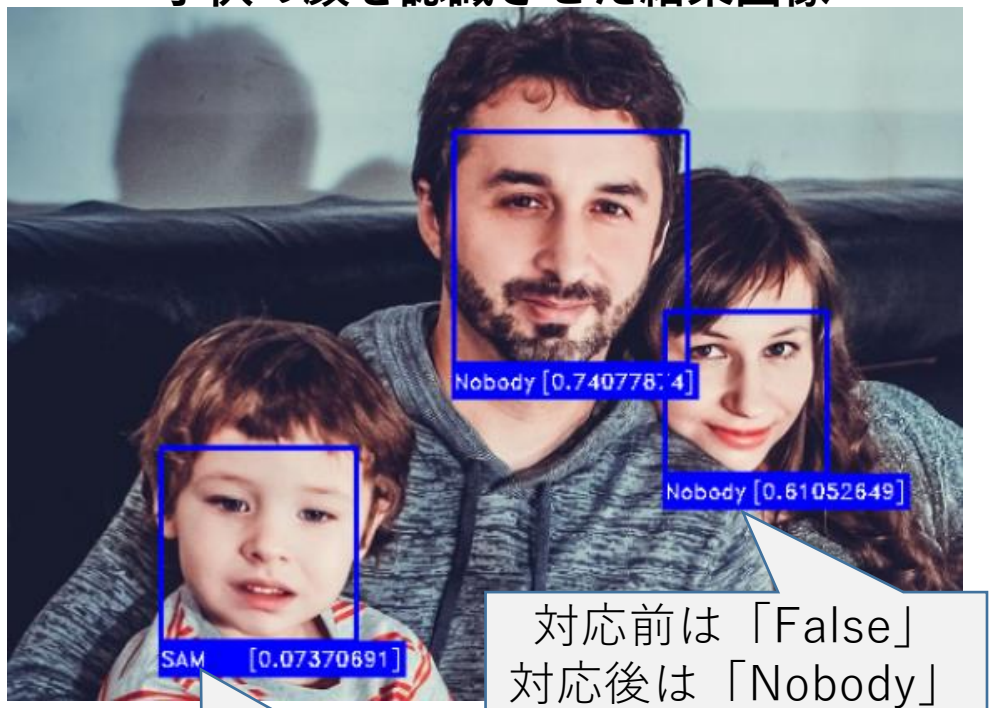
・「結果格納エリア」 に「名前格納エリア」を追加

・「検出した顔の数分、認識処理を実施する」処理で、
「認識結果を出力」が結果True/Falseにより
「名前格納エリア」に名前を格納する処理を追加

・「認識結果と値を書き込む」処理で、
認識結果ではなく、名前を設定して表示
(必要に応じて表示位置の微調整を実施)

・出力するファイル名を「output_name.jpg」に変更

子供の顔を認識させた結果画像



第七章　課題（カスタマイズ）

【課題②】：難易度高

face_recognitionで、顔をどのように分類しているか理解するために、検出した顔に特徴点を追加してください。※課題①を対応した前提

<対応ヒント>

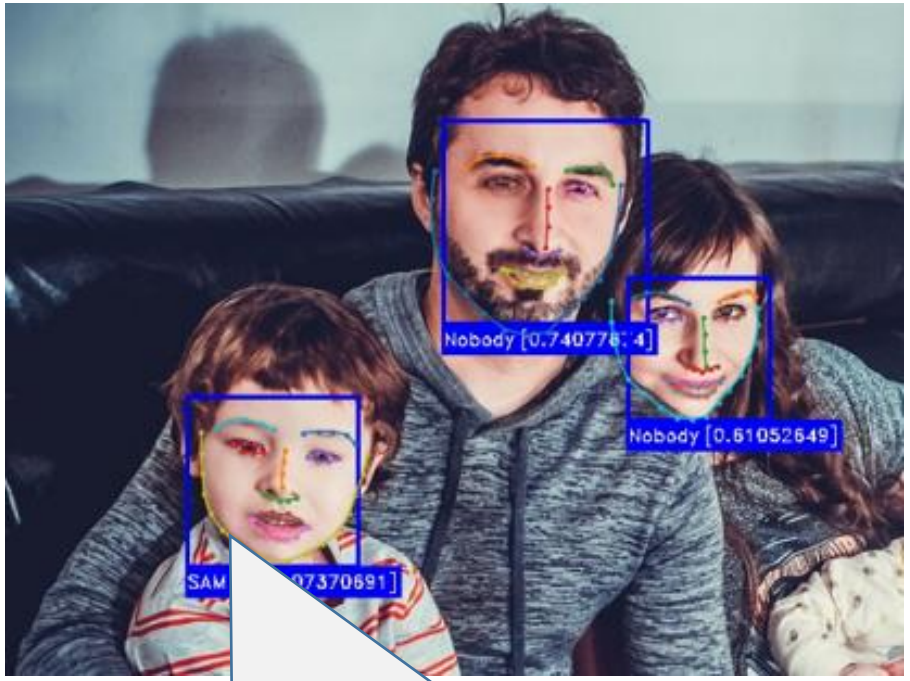
・「検出した顔の数分、認識処理」で face_recognitionメソッド「face_landmarks」を実行

・ face_landmarksの引数は(画像, 検出した顔)

・「認識処理の結果」の合成の後に、Matplotlibを使用して特徴点を合成

・出力ファイル名を「output_landmarks.jpg」として保存

子供の顔を認識させた結果
＋特徴点を追加した画像



各顔に対して特徴点が追加表示される