10.5. 姿勢推定

今度は、姿勢推定と体の部位の認識を行ってみましょう。これには BodyPose が使用できます。BodyPose は体の主要な関節の位置を推定することで、画像や動画内の人物の姿勢推定が行えます。例えば、サンプルプログラムの BodyPose を実行すると、BodyPose が検出した部位に丸が表示されます。人の目や鼻、耳などが検出され丸が表示されているのが分かります。BodyPose では 2 つのモデルが使用できますが、以下では、デフォルトで使用できる MoveNet をもとに説明します。

図 10.5 BodyPose の実行結果

10.5.1 プログラムの処理の流れ

このような推定処理は、基本的には、画像分類と同じです。すなわち、(1) 推定器を用意する、(2) 推定器で姿勢や部位を推定する、(3) 推定結果を受け取る、です。

リスト 人工知能を扱う.3 BodyPose の sketch.js

```
1.  // BodyPose.js
2.  let bodyPose; // 姿勢推定器
3.  let img; // 推定する画像
4.
5.  function preload() {
6.    bodyPose = ml5.bodyPose(modelLoaded); // (1) 推定器の用意
7.    img = loadImage("kao.jpg") // 画像の読込み
8.  }
9.
10. function setup() {
11.   createCanvas(800, 800);
12.   image(img, 0, 0);// 画像を表示する
13. }
14.
15. function draw() {
```

[ここに入力]

```
16.
17.  }
18.
19.  function modelLoaded(){
20.    bodyPose.detect(img, onPose);// (2) 推定し結果が得られたら onPose を実行する
21.  }
22.
23.  function onPose(r) {
24.    print(r);// (3) 推定結果の表示
25.  }
```

これを実行すると、以下のように推定結果が格納された[Object]がコンソールに表示されます（結果を丸で表示するのは 10.5.3 節で説明します）。

```
22
23 function onPose(r) {
24   print(r);
25 }
```

コンソール

▶ (1) [Object]

ここでは流れを分かりやすくするために print しかしていません。BodyPose の推定結果は複雑なので説明は後にして、まずプログラムの流れを説明します。

5 行目の preload 関数で ml5.bodyPose 関数で推定器を作成しています。モデルはネットワークからダウンロードされます。ImageClassifier と同様に preload 内で実行していますが、ダウンロードが終わったら modelLoaded 関数を呼び出すようにしています。

ml5.bodyPose([model], [callbck])
　BodyPose の事前学習済みモデルを読み込む。
引数
　model 使用するモデル。”MoveNet”（デフォルト）と”BlazePose”が指定可能
　callback　モデルを読込み終わったら実行される関数
戻り値
　読み込んだモデルを管理するオブジェクト(ml5. bodyPose オブジェクト)

ml5.poseNet オブジェクトは次のメソッドを提供しています。

表 10.1 ml5.poseNet オブジェクトが提供するメソッド

メソッド名	説明
detect(input, callback)	input は画像、または動画。推論が終わったら callback に指定された関数が呼び出される
getSkeleton()	骨格を構成するキーポイントの接続情報を返す
detectStart(input, callback)	input は画像、または動画。推論を開始し、動画の場合は連続的に推論を行う
detectStop()	推論を停止する

[ここに入力]

modelLoaded 関数は、19 行目にありますが、ImageClassifier のとは処理が少し異なります。BodyPose の場合は、推論が終わったときに呼び出される関数（onPose）を detect メソッドで指定しておきます（20 行目）。BodyPose は推論が終わると、detect メソッドで指定した onPose 関数が実行されます。これは 23 行目で定義されており、推定結果を受け取る引数（r）を 1 つとります。

10.5.2 推論結果のデータ構造

基本的な処理の流れが理解できたら、推論結果の内容を見てみましょう。BodyPose は、姿勢と体の部位を識別できるので、推定結果であるキーポイントの信頼度の平均値、キーポイントの位置を持っています（キーポイント間の接続情報である骨格情報は、getSkeleton メソッドで取り出します）。

実際に見てみると、今回のサンプルプログラムでは次のようなデータが得られています。以下は、[Object]の左の▶をクリックして展開したものです。いろいろなデータが格納されていることが分かります。

```
25 function onPose(r) {
26   print(r);// (3) 推定結果の表示
27 }
```

コンソール

```
▼(1) [Object]
 ▼0: Object
  ▶keypoints: Array(17)
  ▶box: Object
   score: 0.4471070170402527
   id: 1
  ▼nose: Object
     x: 260.6850814819336
     y: 257.8308264339671
     confidence: 0.5495452880859375
  ▼left_eye: Object
     x: 277.43330001831055
     y: 197.707344010297
     confidence: 0.7088833451271057
  ▼right_eye: Object
     x: 222.43467330932617
     y: 230.82883617176728
     confidence: 0.757487952709198
  ▶left_ear: Object
  ▶right_ear: Object
  ▶left_shoulder: Object
  ▶right_shoulder: Object
  ▶left_elbow: Object
  ▶right_elbow: Object
  ▶left_wrist: Object
  ▶right_wrist: Object
```

これを、配列とオブジェクトの形で書くと以下のような構造になります。なお、数値は長いので適当に省略してあります。また、番号とコメントは便宜上つけたものです。

```
 1 [
 2 {
 3   0: { // 姿勢の推定結果(キーポイントの情報)
 4     keypoints: [ 0: { x: 257, y: 260, name: "nose", confidence: 0.549}, ...], // キーポイントのリスト
 5     box: { yMin: 54,4, xMin: 158.7, yMax: 376.8, xMax:516.9, width:358, height: 332.3},
 6     score: 0.447,
 7      id: 1},
```

[ここに入力]

```
 8    nose: { x: 261, y: 241, confidence: 0.999 }, // 各キーポイントの位置と信頼度（鼻）
 9    leftEye: { x: 281, y: 192, confidence: 0.998 }, // 左目
 10   rightEye: { x: 221, y: 233, confidence: 0.995 }, // 右目
      ...
24        rightAnkle: { x: 240, y: 532, confidence: 0.033 } // 右の足首
25    }
26 ];
```

これをまとめた 3 行目～25 行目が 1 人分の推定結果です。複数人を推定させると、オブジェクトがその人数分続きます。このため一番外側に[と]があり配列になっているのです。

複雑そうに見えるのは、BodyPose が推定しているのは 17 個のキーポイントの位置だけなのに対して、得られた結果をプログラムで使いやすいように加工して別の形でも提供しているからです。まず基本となるキーポイントから説明していきましょう。

BodyPose では、以下の 17 カ所のキーポイントの位置を推定し、その位置と推定結果に関する確信度（0.0 ～ 1.0）が得られます（なお、BlazePose モデルでは 33 個のキーポイントが取得できます）。

表 10.2 BodyPose の検出するキーポイント

ID	部位	ID	部位	ID	部位	ID	部位
0	鼻	5	左肩	10	右手首	15	左足首
1	左目	6	右肩	11	左腰	16	右足首
2	右目	7	左ひじ	12	右腰		
3	左耳	8	右ひじ	13	左ひざ		
4	右耳	9	左手首	14	右ひざ		

図にすると以下のようになります。

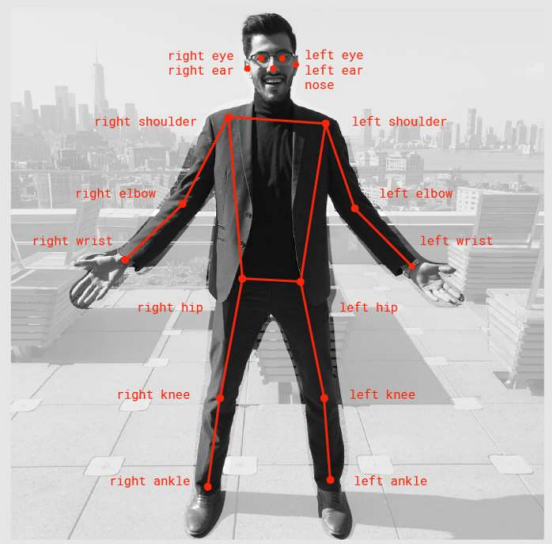

図 10.6 BodyPose のキーポイント

この 17 個のキーボードを格納しているのが、4 行目の keypoints プロパティです。

```
keypoints: [ 0: { x: 257, y: 260, name: "nose", confidence: 0.549}, 1: {x, 197...], // キーポイント
```

keypoints プロパティの配列の中に、オブジェクト形式でそれぞれの部位(part)の位置(x, y)、名前、その推定の信頼度(confidence)が 17 個入っています。これが BodyPose の検出してい

る基本データです。

これ以外のデータは、このデータをもとに作り出されたものです。6 行目の score プロパティは、keypoints の confidence の平均値で、推定全体の信頼度が分かります。その後の 8～24 行目は、先ほど説明した keypoints の各部位名をプロパティとして、その位置と信頼度をオブジェクトとして持っているものが列挙されているだけです。

10.5.3 推定結果（キーポイント）を可視化する

データ構造が分かったので結果を可視化してみましょう。キーポイントの位置に円を描画してみます。BodyPose から結果を受け取る 23 行目の onPose 関数に drawKeypoints 関数を追加します。これは、キーポイントの場所に円を描画する関数で、28 行目で定義されています。

```
23 function onPose(r) {
24   print(r);
25   drawKeypoints(r); // キーポイントに円を描画する
26 }
27
28 function drawKeypoints(r){
29    for (let i = 0; i < r.length; i++) {// 検出された人数分処理する
30      let pose = r[i];// キーポイント情報の取り出し
31      for (let j = 0; j < pose.keypoints.length; j++) {// 個数分処理する
32        let kp = pose.keypoints[j];// 個々のキーポイントの取り出し
33        if (kp.confidence < 0.5) continue; // 確信度のチェック
34        ellipse(kp.x, kp.y, 8, 8);// 円の描画
35      }
36    }
37 }
```

ここでは、先ほどのデータ構造に沿って確信度の高いキーポイントだけを円で描画しているだけです。29 行目で得られた人数分のデータを処理するため、配列 r の長さ（r.length）分、for 文をまわしています。30 行目で、各要素の情報を取り出し pose 変数に代入し、31 行目で keypoints 配列が持つキーポイントの個数分 for 文を回しています。32 行目は keypoints 配列から 1 つずつデータを取り出し kp 変数に代入しています。33 行目では、それぞれのキーポイントで確信度の低いものを読み飛ばし、それ以外を 34 行目で確信度の高いものだけを描画しているだけです。0.5 という値はアプリケーションに応じて適当に調整されて構いません。これを実行すると図 10.5 のような表示が得られます。

10.5.4 推定結果（骨格情報）を可視化する

キーポイントがどのような接続関係を持っているかは骨格情報を取り出すことで得られます。これは、getSkeleton メソッドで得ることができます。以下に、サンプルプログラムの BodyNetSkeleton でこの情報を取り出している部分を示します。

```
20.function modelLoaded(){
21.  bodyPose.detect(img, onPose);
22.  skeleton = bodyPose.getSkeleton(); // 骨格情報の取り出し
23.  print(skeleton)
24.}
```

[ここに入力]

22 行目で取り出している骨格情報（skeleton）は以下のような配列になっています。ここでの数字は表 10.2 に示したキーポイントの ID の組です。最初の[0, 1]は、ID の 0（つまり、鼻）と 1（左目）が接続されていることを示します。

```
[[0, 1],[]0, 2],[1, 3], ...]
```

この接続情報を描画するのは簡単です。以下にプログラムを示します。46、47 行目でキーポイントの ID を取り出し、それをもとに、接続する 2 点を a、b に取り出し、line 関数で線を引いています。

```
31. function drawSkeleton(r) {
32.   for (let i = 0; i < r.length; i += 1) {
33.     let pose = r[i];
34.     for (let j = 0; j < skelton.length; j += 1) {
35.       let id_a = skelton[j][0]
36.       let id_b = skelton[j][1]
37.       let a = pose.keypoints[id_a]
38.       let b = pose.keypoints[id_b]
39.     if (a.confidence < 0.1 || b.confidence < 0.1) continue;
40.     line(a.x, a.y, b.x, b.y);
41.   }
42. }
43.}
```

サンプルプログラムの BodyPoseSkeleton を実行すると以下のように表示されます。

図 10.7 PoseNetSkeleton の実行結果

この写真では、両肩と肘、右手首が線で結ばれていることが分かります。

10.5.5 動画を処理する

BodyPose を動画に対して使ってみましょう。ml5.bodyPose は ml5.imageClassifier とは異なり、モデルを読み込んだ後で、camera を指定して推定を開始します。以下にプログラムを示します。

リスト 10.4 BodyPoseCamera の sketch.js

```
1.   // BodyPoseCamera
2.   let bodyPose; // 姿勢推定器
```

[ここに入力]

```
3.  let camera; // カメラ
4.  let skelton;
5.
6.  function setup() {
7.    createCanvas(640, 480);
8.    camera = createCapture(VIDEO);
9.    pn = ml5.bodyPose(modelLoaded); // (1) 推定器の用意
10.   camera.hide(); // 描画領域に表示するので隠す
11. }
12.
13. function draw() {
14.
15. }
16.
17. function modelLoaded() {
18.   bodyPose.detectStart(camera, onPose);
19.   skelton = bodyPose.getSkeleton(); // 骨格情報の取り出し
20. }
21.
22. function onPose(r) {
23.   image(camera, 0, 0, width, height); // カメラ画像の表示
24.   drawKeypoints(r); // キーポイントの描画
25.   drawSkeleton(r); // 骨格の描画
26. }
…
```

ご覧になって分かるように 18 行目で detect メソッドではなく、detectStart メソッドで camera を指定しています。あとはカメラから得られる画像に対して、順次 onPose 関数が実行されるので、その関数内で画像の表示、キーポイント、骨格を表示するだけです。drawKeypoints と drawSkeleton はこれまでと同じです