



AITCシニア勉強会

Raspberry Piで
画像認識をしてみよう

2021年2月26日

先端IT活用推進コンソーシアム
クラウド・テクノロジー活用部会
リーダー 荒本道隆

今日のスケジュール

各自のラズパイ上で、各サンプルプログラムを実行

- OpenCV の解説&モロモロ：20分
- OpenCV で、顔検出：30分
- posenet で、姿勢検出：20分
- yolov5 で、物体検出：20分
- 3月のハッカソンのご相談：30分

事前準備がまだの人は、今から実行してください

```
cd
```

```
wget http://cloud.aipc.jp/20210226_RaspberryPi4/install.txt
```

```
bash ./install.txt # 約45分
```

インストール作業の解説

- **install.txt の作り方**
 - 手順をテキストファイルに順番に書いた
 - 人力で、最小のステップでできるように最適化
 - できるだけ人手を介さないように「-y」を付けた
 - ラズパイのOSを真っ新にして、何度もテスト実行
 - 今回の写経は、python に集中
- **一番大変だったのは**
 - 姿勢検出を4種類試し、ラズパイで動作したのは1種類
 - 以前は動いた手順が、使えなくなっていた
 - yolov5 は、直前でサンプルの一部が変わった

顔検出とは

- 画像から人間の顔を検出する

- OpenCV では「顔の中の目」も検出できる
- OpenCV には、ネコの顔の検出器も入っている
 - haarcascade_frontalcatface_extended.xml

- 「顔識別」の準備に、顔だけを撮影するのに便利

AITC 先進IT活用推進コンソーシアム

事前準備-1

- 学習させたい人の顔画像を大量に集める
 - OpenCVのサンプル「`facetect.py`」を改造
 - PCで起動しておく、勝手に毎秒保存

```
while True:
    ret, img = cam.read()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    gray = cv2.equalizeHist(gray)

    t = clock()
    rects = detect(gray, cascade)
    vis = img.copy()
    draw_rects(vis, rects, (0, 255, 0))
    for x1, y1, x2, y2 in rects:
        now = datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
        cv2.imwrite("face/" + now + ".jpg", vis[y1:y2, x1:x2])
        roi = gray[y1:y2, x1:x2]
        vis_roi = vis[y1:y2, x1:x2]
        subrects = detect(roi.copy(), nested)
        draw_rects(vis_roi, subrects, (255, 0, 0))
    dt = clock() - t
```



他の人がカメラの前を通らないよう注意

この部分だけをファイルに出力

9

AITC 先進IT活用推進コンソーシアム

事前準備-2

- 家族以外の顔画像を大量に集める
 - Labeled Faces in the Wild から収集
 - 研究用に作成された、著名人がラベル付けされた画像集
 - OpenCVを使って、顔だけを保存

処理済みのファイルを菅井さんからもらった



<http://vis-www.cs.umass.edu/lfw/>

10

姿勢検出とは

- 画像から人のスケルトン（実際の骨じゃない）を検出する
 - 本家OpenPoseでは、
 - GPUパワーがあれば、指や顔のパーツも検出可能



姿勢検出: OpenPose とは

- 概要
 - 画像からPoseの推定に深層学習を用いて、教師データを使用
 - GPUがあれば、顔や手も解析できる
 - ソースコードや学習済モデルが公開されている
<https://github.com/CMU-Perceptual-Computing-Lab/openpose>
 - 非商用であれば無償、商用利用は25,000ドル
 - 無償で使えるものも出てきている



OpenPoseのGithubより

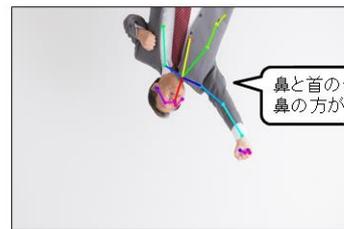


横になっているかを判定

人を検出できなかった場合



人を検出できた場合



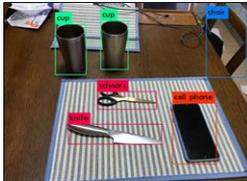
物体検出とは

- 画像から物体を検出する
 - 学習済の物体じゃないと検出できない
 - 学習させるためには、対象の画像データが大量に必要
 - → 様々なオープンデータがある
 - 顔検出や目検出も、物体検出です



Darknet(YOLO)による物体検出精度

高精度 & 低速
yolov3.cfg



IMG_4094.jpg: Predicted in 30.542562 seconds.
scissors: 95%
cell phone: 96%
chair: 80%
knife: 78%
cup: 100%
cup: 98%

低精度 & 高速
yolov3-tiny.cfg



IMG_4094.jpg: Predicted in 1.329064 seconds.
cup: 67%



IMG_4098.jpg: Predicted in 30.556203 seconds.
scissors: 100%
cell phone: 85%
chair: 71%
cup: 99%
cup: 97%



IMG_4098.jpg: Predicted in 1.330256 seconds.
cell phone: 80%
cell phone: 60%
cup: 83%
cup: 77%
cup: 68%

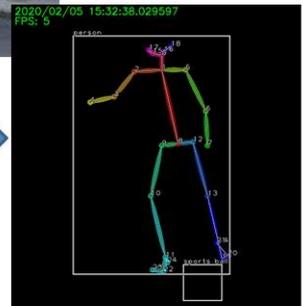


姿勢検出 + 物体検出の実行結果

- 姿勢検出
 - 骨格を描画
- 物体検出
 - 枠で囲む
- 立っているか判定
 - 立っている → 白
 - 横になっている → 赤
- 空間OSにJSON送信
 - 1人分の骨格(25点の座標)
 - 1個の物体(名称と座標)
 - bottleを優先



サンプル画像



画像処理と処理性能

- 精度と速度のバランスが重要
 - 高精度にすると、速度が下がる
 - 速度が下がると、遅延が大きくなる
- 遅延が発生する理由：OpenCVの場合
 - バッファに入らなくなるまで画像を貯める（10枚程度）
 - 1枚を取り出して、画像処理を行う
 - 1枚分の空きができたから、1枚の画像(A)がバッファに入る
 - 1枚分の処理が3秒だと、画像(A)を取り出すのは30秒後
 - → 1枚分の処理を早くするか、取出専用スレッドを作る



性能が足りないのであれば

- ハードウェアを追加 → サンプルコードが少ない
- **別ハードウェアで実行 → 構築手順が違う**
 - マイナーなものじゃなければ、ググれば見つかる
- クラウドを利用 → 超賢い、使った分だけ課金が



準備した機器 - 1

- NVIDIA Jetson Xavier (¥86,832-)
 - GPU 512-core Volta GPU with Tensor Cores
 - CPU 8-core ARM v8.2 64-bit CPU
 - Memory 16 GB 256-Bit LPDDR4x | 137 GB/s
- NVIDIA Jetson Nano (¥12,540-)
 - GPU 128-core Maxwell
 - CPU Quad-core ARM A57 @ 1.43 GHz
 - Memory 4 GB 64-bit LPDDR4 25.6 GB/s
- Raspberry Pi 4 (¥7,700-)
 - CPU 1.5GHz quad-core ARM Cortex-A72 CPU
 - Memory 4 GB LPDDR4



8



姿勢検出と物体検出の処理速度

- 姿勢検出 : OpenPose/tf-pose-estimation

チューニング無し

	高精度&低速	低精度&高速	備考
Jetson Xavier	13FPS	22FPS	どちらも遅延なし
Jetson Nano	2.5FPS	3.6FPS	遅延が3秒、2秒弱ぐらい
ラズパイ4	0.30FPS	0.39 FPS	遅延が20秒、17秒ぐらい
ラズパイ3	0.13FPS	0.17FPS	遅延が40秒、30秒ぐらい
ノートPC	0.8FPS	1.1FPS	遅延が8秒、6秒ぐらい

- 物体検出 : darknet(YOLO)

	高精度&低速	低精度&高速	備考
Jetson Xavier	4.4FPS	12FPS	遅延が2秒、遅延はぼなし
Jetson Nano	0.6FPS	7.4FPS	遅延が12秒、1秒ぐらい
ラズパイ4	応答無し	0.2FPS	応答しない、遅延が34秒ぐらい
ラズパイ3	応答無し	応答無し	応答しない
ノートPC	応答無し	1.0FPS	応答しない、遅延が7秒ぐらい

備考 : v3はbatch=16に修正

16

OpenCVとは

● 概要

画像処理・画像解析および機械学習等の機能を持つC/C++、Java、Python、MATLAB用ライブラリ [1] [2]。プラットフォームとしてMac OS XやFreeBSD等全てのPOSIXに準拠したUnix系OS、Linux、Windows、Android、iOS等をサポートしている。

● できること

Wikipediaより

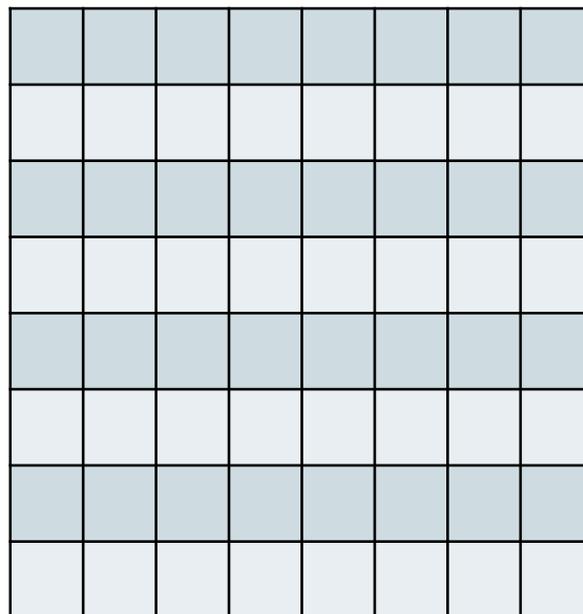
○ たくさんあるのでWikipediaを参照

• <http://ja.wikipedia.org/wiki/OpenCV>

○ Windows/Macで開発して、ラズパイで実行

OpenCVのメリット

- 安心して使用できる
 - BSDライセンス
- 様々な画像操作が簡単にできる
 - 拡大縮小回転、合成、保存読み込み、が1操作
- 画像操作が容易
 - 8x8の画像の例 →
 - 配列として操作できる
 - [0, 0] : 先頭ピクセル



ラズパイでOpenCVを動かすために

- 動作状況を表示するために
 - ラズパイにモニタを接続して、GUIを起動
 - もしくは、パソコンにX-Windowサーバを導入
 - Windows : Xming を導入
 - ・ 接続方法 : TeraTermの『X Forwarding』を有効にする
 - ・ もしくは : 「export DISPLAY=WindowsのIPアドレス:0」を設定
 - Mac : 標準装備
 - ・ 接続方法 : ターミナルから「ssh -Y pi@IPアドレス」
- OSに依存しないやり方
 - PCやスマホのブラウザで結果画像を表示 ← 今回はこっち
 - **ブロードバンドルータのポート転送で、Internet公開も可**
 - **注意 : 操作はできません**
 - **注意 : SDカードとネットワークが高負荷になります**

OpenCVでカメラ映像表示ー1

- PCのブラウザで、結果画面を表示
 - <http://ラズパイのIP/viewx/>



OpenCVでカメラ映像表示ー2

- ちょっとだけプログラムを修正

- `cd ~/opencv/samples/python/`
- `vi video.py`

インデントに注意
スペースで、先頭を合わせる

```
218:  #cv.imshow('capture %d' % i, img)      # この行をコメントアウト
      import os
      cv.imwrite("/tmp/tmp.jpeg", img)      # チラつき防止
      os.rename("/tmp/tmp.jpeg", "/var/www/html/viewx/img/capture.jpeg")
```

- 実行

- `python3 video.py`
- 謎の美女（Lennaさん）が表示された場合
 - → USBカメラを認識していない
 - → ラズパイを再起動「`sudo reboot`」を実行

ラズパイにHDMIモニタを接続し、
X-Windowを起動していれば、
修正は一切不要



OpenCVでカメラ映像表示ー3

- OpenCVを使って、映像を加工してみる

- `cp video.py video2.py` # 改造用にコピー

- `vi video2.py`

赤は、`[0, 0, 0xff]`

```
img[50:100, 50:100] = 0 # 追加
218: #cv.imshow('capture %d' % i, img)
219: import os
220: cv.imwrite("/tmp/tmp.jpeg", img) # チラつき防止
221: os.rename("/tmp/tmp.jpeg", "/var/www/html/viewx/img/capture.jpeg")
```

- 実行

- `python3 video2.py`



(50,50)-(100,100)に、
黒い箱を書く

OpenCVでカメラ映像表示ー4

- OpenCVを使って、Tiktok風ツールを作成
 - vi video2.py

```
fb = None                # 加工済画像入れ
index = 0                # 定義
caps = list(map(create_capture, sources))
shot_idx = 0
while True:
    imgs = []
    for i, cap in enumerate(caps):
        ret, img = cap.read()
        imgs.append(img)
        if fb is None:    # 1回目だけ
            fb = img     # FrameBuffer は img とまったく同じとしたい
        # 縦方向
        fb[index:,] = img[index:,]    # 線より下をfbにコピー
        index = index + 2             # スピード
        fb[index:index+5,] = 0xff     # 白線の太さ
        # 横方向
        #fb[:, index:] = img[:, index:] # 線より右をfbにコピー
        #index = index + 2             # スピード
        #fb[:, index:index+5] = 0xff   # 白線の太さ

    #cv.imshow('capture %d' % i, img)
    import os
    cv.imwrite("/tmp/tmp.jpeg", fb)    # チラつき防止   # 描画をimg -> fb に変更
    os.rename("/tmp/tmp.jpeg", "/var/www/html/viewx/img/capture.jpeg")
```

OpenCVで顔認識

- OpenCVの顔認識を動かしてみる
 - vi facedetect.py

```
68:      #cv.imshow('facedetect', vis)          # この行をコメントアウト
      import os
      cv.imwrite("/tmp/tmp.jpeg", vis)        # チラつき防止
      os.rename("/tmp/tmp.jpeg", "/var/www/html/viewx/img/capture.jpeg")
```

- 実行
 - python3 facedetect.py

ラズパイにHDMIモニタを接続し、
X-Windowを起動していれば、
修正は一切不要

- 他にも色々サンプルが入っている
 - ls

posenetで姿勢検出ー1

- posenetの姿勢検出を動かしてみる
 - `cd ~/posenet-python`
 - `vi webcam_demo.py`

```
57: #cv2.imshow('posenet', overlay_image) # この行をコメントアウト
    cv2.imwrite("/var/www/html/viewx/img/capture.jpeg", overlay_image)
```

- 実行
 - `python3 webcam_demo.py`
 - ※初回だけ学習済データを取得する時間(8分)がかかる
- 精度を下げて、実行速度を上げる
 - `python3 webcam_demo.py --cam_width 80`

posenetで姿勢検出ー2

- posenetの処理速度を計ってみる
 - vi webcam_demo.py

```
59:         frame_count += 1
60:         if cv2.waitKey(1) & 0xFF == ord('q' ):
61:             break
62:
63:             print('FPS: ', 1 / (time.time() - start))           # 追加
64:             start = time.time()                                 # 追加
65:
66:     print(' Average FPS: ', frame_count / (time.time() - start))
```

- 実行
 - python3 webcam_demo.py
 - python3 webcam_demo.py --cam_width 80

yolov5で物体検出ー 1

- サンプル画像を確認
 - ブラウザを見ながら実行
 - `cd ~/yolov5`
 - `cp data/images/bus.jpg /var/www/html/viewx/img/capture.jpeg`
 - `cp data/images/zidane.jpg /var/www/html/viewx/img/capture.jpeg`
- サンプル画像に対して実行
 - `python3 detect.py`
 - ※初回だけ学習済データを取得する時間(1分)がかかる
- 結果を確認
 - `cp runs/detect/exp*/bus.jpg /var/www/html/viewx/img/capture.jpeg`
 - `cp runs/detect/exp*/zidane.jpg /var/www/html/viewx/img/capture.jpeg`

yolov5で物体検出一 2

- yolov5 の物体検出を動かしてみる
 - vi detect.py

```
48:      #view_img = check_imshow()      # この行をコメントアウト
      view_img = True                  # 常にTrue

119:     if view_img:
120:         #cv2.imshow(str(p), im0)     # この行をコメントアウト
      cv2.imwrite("/var/www/html/viewx/img/capture.jpeg", im0) # 追加
121:         cv2.waitKey(1) # 1 millisecond
```

- 実行
 - python3 detect.py --source 0
- 精度を下げて、実行速度を上げる
 - python3 detect.py --source 0 --img-size 100

yolov5で物体検出ー3

- 検出できるモノのリストを出力
 - vi detect.py

```
56: # Get names and colors
57: names = model.module.names if hasattr(model, 'module') else model.names
    print (names)           # 追加
58: colors = [[random.randint(0, 255) for _ in range(3)] for _ in names]
```

- 実行
 - python3 detect.py --source 0 --img-size 100
- 実は
 - vi ~/yolov5/data/coco.yaml
- 次は
 - 検出したいモノの画像を一杯集めて、学習させる

過去3回も含めて、 もっとやってみたいことを検討

2020年11月20日（金）Raspberry Piのセットアップしよう

2020年12月18日（金）Raspberry Piでセンサーを操作する

2021年1月22日（金）Raspberry Piでインターネット連携

2021年2月26日（金）Raspberry Piで画像認識 ← 現在ここ