

1章 開発環境の構築

パッケージマネージャの更新

必要なソフトウェアの導入には、ソフトウェアやライブラリの導入と削除、それらの依存関係を管理するパッケージマネージャを使います。

ここでは、システムにインストールされるソフトウェアやライブラリを管理するためにAPT(エー・ピー・ティーまたはアプト)、そしてPythonに関するものはpip(ピップ)と呼ばれるパッケージマネージャを使います。それぞれapt、pipコマンドから利用できます。

インストールを行う前に、APT自体を更新しておきます。pipに関してはそのままです。

ラズパイのターミナルを開き、下記のコマンドを実行してください。

```
sudo apt update -y
```

TensorFlow

[TensorFlow](#)は機械学習のためのソフトウェアフレームワークです。テンソルフローと読みます。今回はカメラの映像から人を判別するために使います。

はじめにaptとpipでそれぞれTensorFlowが依存するライブラリをインストールします。インストールには15分ほどの時間を要します。

```
sudo apt install libhdf5-dev \  
                 libc-ares-dev \  
                 libeigen3-dev \  
                 openmpi-bin \  
                 libatlas-base-dev
```

実行途中で「続行しますか？[Y/n]」と表示されるのでYを入力してエンターキーで続行します。こちらもインストールします。

↓1行ですので全てコピーして貼り付けてください。

```
pip3 install --no-deps keras_applications==1.0.8  
keras_preprocessing==1.1.0
```

```
pip3 install h5py==2.9.0 grpcio==1.25.0
```

次に、TensorFlowをインストールします。

↓1行ですので全てコピーして貼り付けてください。

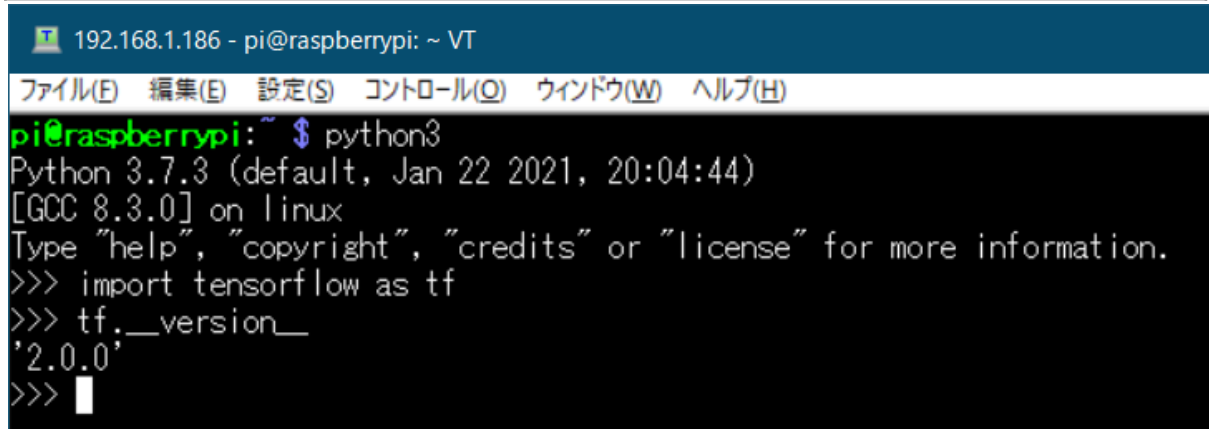
```
wget
https://isaax-public.s3-ap-northeast-1.amazonaws.com/ai-kit/tensorflow-2.0.0-cp37-cp37m-linux_armv7l.whl
```

こちらもインストールします。

```
pip3 install --user tensorflow-2.0.0-cp37-cp37m-linux_armv7l.whl
```

ターミナルで「python3」コマンドを実行し、Pythonの対話モードでTensorFlowがインストールできているかチェックしましょう。

```
>>> import tensorflow as tf
>>> tf.__version__
```



```
192.168.1.186 - pi@raspberrypi: ~ VT
ファイル(E) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
pi@raspberrypi:~ $ python3
Python 3.7.3 (default, Jan 22 2021, 20:04:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow as tf
>>> tf.__version__
'2.0.0'
>>>
```

エラーが発生せず、バージョン番号が表示されればOKです。

実行後はexit()を実行してpython3の対話モードを終了してください。

OpenCV

[OpenCV](#)は画像処理用のライブラリです。オープンシーブイ (または、オープンシーヴィ) と読みます。画像処理に必要な様々な処理をAPIから手軽に扱うことができます。

先ほどと同じく、依存ライブラリのインストールから行います。

```
sudo apt install libjasper1 \
                  libqtcore4 \
                  libqtgui4 \
                  libqt4-test
```

OpenCVのインストールを行います。

```
pip3 install opencv-python==3.4.6.27
```

それでは、Pythonの対話モードで正しくインストールできていることを確認しましょう。

```
>>> import cv2
>>> cv2.__version__
```

```
192.168.1.186 - pi@raspberrypi: ~ VT
ファイル(E) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
pi@raspberrypi:~$ python3
Python 3.7.3 (default, Jan 22 2021, 20:04:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import cv2
>>> cv2.__version__
'3.4.6'
>>> 
```

エラーが発生する場合は、依存ライブラリやOpenCVのインストールをもう一度試してみてください。

必要なソフトウェアは以上です！

2章 人の判別にチャレンジしよう

ディレクトリの用意

AI見守りカメラに関わるすべてのプログラムを一箇所にまとめておくために作業用ディレクトリを作ります。ホームディレクトリに「mimamori」を作り、移動しておきましょう。

```
cd ~/
mkdir mimamori
cd mimamori/
```

```
192.168.1.186 - pi@raspberrypi: ~/mimamori VT
ファイル(E) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
pi@raspberrypi:~$ cd ~
pi@raspberrypi:~$ mkdir mimamori
pi@raspberrypi:~$ cd mimamori/
pi@raspberrypi:~/mimamori$ 
```

モデルのダウンロード

AIによる人の判別は、たくさんの画像を使って学習されたAIモデルを使って行います。あらかじめ用意したモデルを使うため、ダウンロードしましょう。

ターミナルで下記コマンドを実行します。

```
↓1行ですので全てコピーして貼り付けてください。
```

```
wget
https://isaax-public.s3-ap-northeast-1.amazonaws.com/ai-kit/ssd.tflite
```

モジュールの読み込み

まずは、必要なPythonのモジュールをインポートします。Thonny IDEを開き、新しいファイルを作成します。ファイル名は「camera.py」としておきましょう。

それでは、camera.pyに以下のコードを記述します。

```
import cv2
import tensorflow as tf
import numpy as np

import time
```

人の判別を行うために必要なすべてのモジュールをインポートしています。

tensorflowはAIによる人の判別を行うために、cv2はカメラの映像を扱うために使います。これらは前章でインストールしましたね。

[Numpy](#)(ナムパイ)は数値計算を行うためのPythonライブラリです。画像データや判別結果を扱うために必要となります。AIではよく登場するライブラリです。

初期化処理

ダウンロードしたモデル(ssd.tflite)の読み込みと、カメラへの接続を行います。さきほどのインポート文に続けて入力しましょう。

```
# TF Liteランタイムの初期化
interpreter = tf.lite.Interpreter(model_path='ssd.tflite')
interpreter.allocate_tensors()

# モデル情報の取得
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
target_height = input_details[0]['shape'][1]
target_width = input_details[0]['shape'][2]

# カメラへの接続
```

```
camera = cv2.VideoCapture(0)
time.sleep(2)
```

「TF Liteランタイムの初期化」のコメントが付いている箇所で、ダウンロードしたモデルを読み込んでいます。TF Liteランタイムは、このモデルを動かすために必要なプログラムです。

「モデル情報の取得」の部分では、読み込んだモデルがどのような入力・出力を持つのかを調べています。これらは後ほど入力する画像のサイズや人の判別結果を取り出すために使います。

「カメラへの接続」の部分は、カメラへの接続処理を行っています。このcamera変数を使うことによって映像を取り出すことができるようになります。

最後に2秒間待つ処理を加えているのは、カメラへの接続が完了し、映像の読み込みができるようになるのを待つためです。

関数の作成

画像内の人を認識し、その位置を描画する detect 関数を作ります。続けてプログラムを記入して下さい。

```
def detect(frame):
    height, width, _ = frame.shape
    resized = cv2.resize(frame, (target_width, target_height))
    input_data = np.expand_dims(resized, axis=0)
    # 画像データの入力
    interpreter.set_tensor(input_details[0]['index'], input_data)
    # 推論実行
    interpreter.invoke()
    # 結果の取得
    detected_boxes =
interpreter.get_tensor(output_details[0]['index'])
    detected_classes =
interpreter.get_tensor(output_details[1]['index'])
    detected_scores =
interpreter.get_tensor(output_details[2]['index'])
    num_boxes = interpreter.get_tensor(output_details[3]['index'])
    # 物体の描画処理
    for i in range(int(num_boxes)):
```

```

# 物体の位置を取得
top, left, bottom, right = detected_boxes[0][i]
# 物体のクラスを取得
class_id = int(detected_classes[0][i])
# 物体のスコア（尤度）を取得
score = detected_scores[0][i]
# スコアが50%を超えない物体は無視する
if score < 0.5:
    continue
if class_id != 0:
    continue
# 位置座標を元の画像と同じ尺に戻す
xmin = int(left * width)
ymin = int(top * height)
xmax = int(right * width)
ymax = int(bottom * height)
# クラスとスコアのタグを用意
tag = '{}: {:.2f}%'.format("person", score * 100)
# 画像に物体の位置を矩形として描画
cv2.rectangle(frame, (xmin, ymin), (xmax, ymax), (0, 255,
0), 3)
# 矩形の近くにタグを描画
cv2.putText(frame, tag, (xmin, ymin - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)

return frame

```

ここでの処理は大きく3つに分かれています。

1. 画像データに手を加え、モデルに与える
2. モデルを動かし、その判別結果を得る
3. 判別結果を画像に描画する

画像データに手を加え、モデルに与える

サンプルコード該当箇所：

```
height, width, _ = frame.shape
resized = cv2.resize(frame, (target_width, target_height))
input_data = np.expand_dims(resized, axis=0)
# 画像データの入力
interpreter.set_tensor(input_details[0]['index'], input_data)
```

画像を扱うAIモデルには大抵、前提としている入力があります。そのため、判別を使う画像をそのサイズに変換します。

ここで使うのが、初期化処理で調べておいたモデルの入力情報です。target_width変数とtarget_height変数が相当します(変数の中身を調べると、300×300の解像度であることがわかります)。

そして変換処理を行った後、interpreter.set_tensor(...)の部分で画像をモデルに入力しています。

モデルを動かし、その判別結果を得る

サンプルコード該当箇所:

```
# 推論実行
interpreter.invoke()
# 結果の取得
detected_boxes =
interpreter.get_tensor(output_details[0]['index'])
detected_classes =
interpreter.get_tensor(output_details[1]['index'])
detected_scores =
interpreter.get_tensor(output_details[2]['index'])
num_boxes = interpreter.get_tensor(output_details[3]['index'])
```

画像データを与えたら、モデルを動かして人の判別を行います。このように、モデルに予測を行わせることを「推論」と呼びます。

推論後は結果を4つの変数に分けて取り出しています。それぞれの意味は次のとおりです。

1. detected_boxes ... 検知した物体の座標
2. detected_classes ... 何の物体を検知したか
3. detected_scores ... 検知した物体の尤度
4. num_boxes 検知した物体の数

尤度(ゆうど)とは、検知した物体の確からしさを表す数値です。

判別結果を画像に描画する

サンプルコード該当箇所:

```
# 物体の描画処理

for i in range(int(num_boxes)):
    # 物体の位置を取得
    top, left, bottom, right = detected_boxes[0][i]
    # 物体のクラスを取得
    class_id = int(detected_classes[0][i])
    # 物体のスコア（尤度）を取得
    score = detected_scores[0][i]
    # スコアが50%を超えない物体は無視する
    if score < 0.5:
        continue
    # 人以外の物体は無視する
    if class_id != 0:
        continue
    # 位置座標を元の画像と同じ尺に戻す
    xmin = int(left * width)
    ymin = int(top * height)
    xmax = int(right * width)
    ymax = int(bottom * height)
    # クラスとスコアのタグを用意
    tag = '{}: {:.2f}%'.format("person", score * 100)
    # 画像に物体の位置を矩形として描画
    cv2.rectangle(frame, (xmin, ymin), (xmax, ymax), (0, 255,
0), 3)
    # 矩形の近くにタグを描画
    cv2.putText(frame, tag, (xmin, ymin - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)

    return frame
```

この部分では、取り出した結果を画像に描画することで、画像内のどこに人がいるのかを緑の枠で示します。

for文を使い、検知したすべての物体を画像に描画します。

それぞれのコードが行っている処理についてはコメントを確認してみてください。

カメラ映像の読み込み

カメラ映像からフレーム(映像の1コマ1コマ)を読み取り、人の判別を行います。

```
while True:
    # フレームの読み込み
    ret, frame = camera.read()
    if not ret:
        continue
    # フレームに写った人の判別
    frame = detect(frame)
    # 人の位置を描画したフレームを画面に表示
    cv2.imshow("frame", frame)
    # 0キーが押されたら、繰り返しを終了する
    if cv2.waitKey(1) == 0x30:
        break
camera.stop()
cv2.destroyAllWindows()
```

camera.read()の部分でカメラ映像からフレームを取り出し、detect関数を使って1枚ずつ人の判別を行っています。

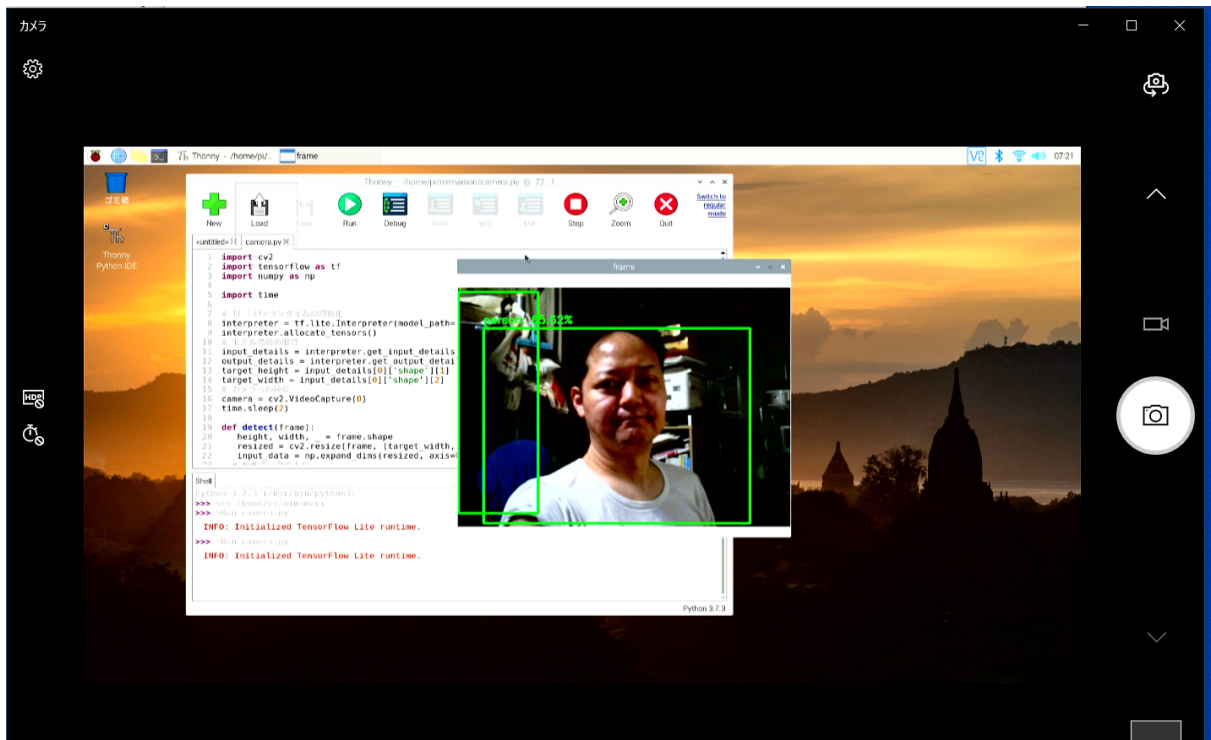
結果は cv2.imshow を使うことでデスクトップに表示します。

これらの処理を while文で繰り返すことにより、リアルタイムな映像の処理を実現しています。

うまく動いているか確認しよう

それでは、動かしてみましょう。ファイルをmimamoriディレクトリ内に保存します。

ファイルを保存したら、Thonny IDEのRunボタンを押しプログラムを実行します。



上手くいくと映像が表示され、人を認識したらその位置を緑色のボックスで囲みます。

もしエラーが出てしまう場合は、作成したコードとテキストのサンプルコードを見比べ、誤字やスペースの使い方に違いがないか確かめてみましょう。