16. 組み込み（後半）

TensorFlow Lite を使った Raspberry Pi 上での物体認証

ここでは、モバイルデバイスや IoT デバイスのための Deep Learning フレームワークである（２）TensorFlow Lite を使った Raspberry Pi 上での物体認証を実現してみましょう。ここでは前回の組み込み（１）で行ったリアルタイム検出の TensorFlow のモデルを TensorFlow Lite のモデルに変換する演習と、TensorFlow Lite モデルメーカーを使って TensorFlow Lite のモデルを作成する演習の２つを実施しましょう。

前回の組み込み（１）で行ったリアルタイム検出の流れを以下に示します。

1. 対象物を撮影する
2. 背景画像を撮影する

Raspberry Pi

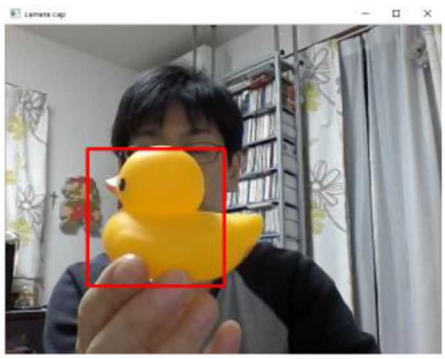
3. 対象物画像を水増しして 300 枚用意する
4. 背景画像を分割して 625 枚用意する
5. 対象物画像と背景画像を csv ファイルに記述する
6. 用意したデータで学習を行う

Google Colab

7. 学習されたデータを用いて、リアルタイムで判定させる

Raspberry Pi

完成図は以下になります（過去のスライドより転載）。



この演習（２）で利用するプログラムを以下の表にまとめます。

演習（２）で利用するプログラム	
15a_AL_ConvertTFL.ipynb	Colab 用プログラム。TFL への変換を追加
15b_detecRoi_TFL.py	Raspberry Pi 用プログラム。TFL を利用
15c_dRoi_TFL_fDemo.py	15b にデモ用モデルを適用したもの
15d_TFL_ModelMaker.ipynb	TensorFlow Lite モデルメーカーを使ったモデル作成
15e_detecRoi_mMaker.py	TensorFlow Lite モデルメーカーのモデルを適用したもの

それでは先に述べた流れに従って進めていきましょう。

1. 対象物を撮影する

まず初期設定を行いましょう。

①初期設定
インターネットに接続した後、ターミナルを起動します。

画面上部のアイコン（LXTerminal）
または
［左上プログラムメニュー］→［アクセサリ］→［LXTerminal］から起動

ターミナル上で、次の apt コマンドを使って Raspberry Pi OS にパッケージやライブラリをインストールします。※以下、インターネットに接続している必要があります。以下のコマンドで apt のリポジトリ情報（更新パッケージの場所の情報）を更新しましょう。なお、時間の関係上、すべてのソフトウェアをアップグレード（upgrade）する作業は省略します（ネット環境とデバイス数によっては数時間かかるケースが発生してしまいます）。

```
sudo apt update
```

②リモート接続の設定
※既に設定済みの場合はこの②は読み飛ばしてください。

リモートで Raspberry Pi を操作したいときは、RDP（Remote Desktop Protocol）や VNC（Virtual Network Computing）、X Windows System、SSH などを利用することができます。VNC や SSH を利用する場合は、［メニューボタン］→［Raspberry Pi の設定］で VNC や SSH を有効にして Raspberry Pi を再起動しましょう。なお、ここでは RDP での接続方法について見ていきます。

まず Raspberry Pi に RDP を以下のコマンドでインストールしましょう。

```
sudo apt install xrdp
```

RDP を起動するコマンドは以下になります。

```
sudo service xrdp restart
```

次に、リモートデスクトップクライアントから Raspberry Pi の IP アドレスを入力して、Raspberry Pi の ID とパスワードを入力すればリモートで作業ができます。Windows の場合は標準でリモートデスクトップクライアントがインストールされています。Windows のスタートメニュー［Windows アクセサリ］→［リモートデスクトップ接続］からリモートデスクトップ接続を起動してください。Mac OS では Microsoft Remote Desktop などのクライアントを利用しましょう。なお、Raspberry Pi の IP アドレスは以下のコマンドから確認できます。

```
Ifconfig
```

なお、Raspberry Pi のユーザ ID は pi になります。パスワードが未設定の場合、RDP でリ

モート接続できません。［左上プログラムメニュー］→［設定］→［Raspberry Pi の設定］から設定ウィンドウを開き、［システムの設定］タブにある［パスワードの変更］からパスワードを設定してください。

RDP を利用した場合、クリップボードを共有することで、互いの画面でコピーとペーストが利用できます。例えば、Windows 上のあるファイルをコピーして、Raspberry Pi のデスクトップなどにペーストすることができます（その逆もできます）。ドラッグ＆ドロップではない点に注意しましょう。また、フォルダはコピーすることができません。複数のファイルをコピー＆ペーストする場合は、ファイルを複数選択してコピー＆ペーストするか、ZIP 形式に圧縮するなどしてまとめてコピー＆ペーストするようにしましょう。

Raspberry Pi を再起動したときは RDP の起動コマンドを入力しなおす必要があります。ここでは、起動処理を行う systemd を使って xrdp を自動起動させましょう。まず、テキストエディタを使って以下のようなファイルを myxrdp.service という名前で作成します（今回は準備済みです）。

myxrdp.service

```
[Unit]
Description=myXRDP
After=syslog.target

[Service]
Type=simple
WorkingDirectory=/usr/share/xrdp
ExecStart=/usr/sbin/xrdp
TimeoutStopSec=5
StandardOutput=null

[Install]
WantedBy = multi-user.target
```

このファイルを /etc/systemd/system/以下に移動してロードします。移動は、以下の mv コマンドで移動させましょう。mv コマンドは、ターミナル上でファイルのある場所から実施する必要があります。例えば、myxrdp.service をデスクトップにコピーしている場合は、cd でホームディレクトリに移動し、cd Desktop を実行して Desktop ディレクトリに移動したり、cd /home/pi/Desktop で Desktop ディレクトリに絶対パスで直接移動して、以下のコマンドを実行する必要があります。

```
sudo mv myxrdp.service /etc/systemd/system/.
```

そして次に、以下のコマンドでファイルを読み込みます。

```
sudo systemctl daemon-reload
```

自動起動の設定が以下になります。これで Raspberry Pi を再起動したときに Raspberry Pi 上で自動的に RDP が起動し、リモート接続サービスを待ち受けている状態になります。あとは Windows 側でリモートデスクトップ接続を実施すれば遠隔で操作ができます。

```
sudo systemctl enable myxrdp.service
```

以下のコマンドでサービスの一覧が確認できます。myxrdp.service がリストにあれば登録されている状態です。なお、閲覧モードはスペースキーで次のページに移動し、終了する時はキーボードの q を押してください。

```
systemctl list-unit-files --type service
```

③Camera Module V2 の接続
カメラモジュールを Raspberry Pi に接続していない場合は接続しましょう。すでに接続している場合は、次の④に進んでください。

④Camera Module の有効化
カメラを接続済みの場合でも片づけたときの衝撃などでカメラが非認証になっていたりする場合があります。その場合は、Raspberry Pi の電源を OFF にしてカメラのリボンケーブルを一度外して付け直してから Raspberry Pi を起動してみましょう。

［左上プログラムメニュー］→［設定］→［Raspberry Pi の設定］をクリックして設定ウィンドウを開き、［インターフェイス］タブのカメラの有効にチェックが入っていることを確認します。無効になっていた場合には有効にチェックを入れて［OK］を押して Raspberry Pi を再起動します。

▼Camera Module V2 の場合
次に、起動後にターミナルで

```
vcgencmd get_camera
```

と入力して

```
supported=1 detected=1
```

とターミナルに表示されればカメラが有効になっています。もし、上記のようになっていない場合、カメラの接続をもう一度やり直したり、ソフトウェアを最新にしたりしてください。

⑤Camera Module V2 の写真撮影機能の確認

ターミナル上でホームディレクトリから以下のコマンドを入力すると、5 秒後に自動写真撮影が行われます。正しく動作していた場合、「/home/pi」のディレクトリ（ホームディレクトリ）に pict.jpg という名前で撮影した写真データが保存されています。

```
raspistill -o pict.jpg
```

ここでカメラの上下を確認しましょう。上下が逆の場合は、-hf -vf オプションをつけることで反転させて撮影することができます。

```
raspistill -o pict.jpg -hf -vf
```

これでようやく「1. 対象物を撮影する」ができるようになりました。以前の演習と同様に、リアルタイムで検出させる物体を各自で撮影してください。撮影した写真は任意の名前で保存しましょう（例えば、pict.jpg）。保存した画像は Windows 側に移し（例えばリモートデスクトップ接続でコピー＆ペースト）、前回同様に Windows PC 上で XnView を使用して対象物だけを切り抜いて、切り抜いた画像を target.jpg という名前で保存しましょう。

XnView でのおおよその作業の流れは以下のようになります。

1. XnView がまだインストールされていない場合は、オフィシャルサイト（https://www.xnview.com/）やダウンロードサイト（窓の杜など）からインストールしましょう。
2. ［編集］→［選択範囲縦横率設定］→［1:1 (1.00)］を選択しましょう。これで選択ツールを正方形に固定することができます。
3. 対象物を囲むように選択したのち、［編集］→［選択範囲トリミング］を選択し、画像を切り抜きましょう。
4. 切り抜いたのち、［画像］→［リサイズ］でサイズ変更画面を開き、縦横 128 ピクセルを指定してサイズ変更しましょう。
5. 最後に target.jpg という名前で切り抜いた画像を保存しましょう。

2. 背景画像を撮影する

前回同様、自分が映りこまないようにして背景画像を撮影します。撮影した写真の名前を other.jpg としましょう。

3. 対象物画像を水増しして 300 枚用意する
4. 背景画像を分割して 625 枚用意する
5. 対象物画像と背景画像を csv ファイルに記述する
6. 用意したデータで学習を行う

前回同様、Google Colab で実施しましょう。
Google Colab で 15a_AL_ConvertTFL.ipynb を開きましょう。このプログラムは前回のプログラムの最後に TensorFlow のモデルを Tensorflow Lite のモデルに変換する機能が追加されています。なお、プログラムの説明はプログラム内のコメントを参照してください。Google Colab のファイル操作を行う左ウィンドウ側で以下の 4 つのファイルをアップロードしましょう。

```
・target.jpg                  # 認証する対象が映っている 128×128 の画像
・other.jpg                   # 背景画像（大きさは任意）
・target_or_other_train.csv   # 学習（訓練用）ファイルパスとラベル
・target_or_other_test.csv    # 検証用ファイルパスとラベル
```

学習が終わったら、作成された重みファイル（converted_model.tflite)をダウンロードしましょう。

前述したプログラム群や作成した重みファイル（converted_model.tflite）は、Raspberry Pi のホームディレクトリに前回作成した Project ディレクトリ(/home/pi/Project)に RDP でコピー＆ペーストして移動しておきましょう（Project ディレクトリを未作成の場合は作成しましょう）。

```
・15b_detectRoi_TFL.py      ┐
・15c_dRoi_TFL_fDemo.py     ├  準備済みのファイル
・15e_detecRoi_mMaker.py    ┘

・converted_model.tflite    ┐  各自が作成したモデル
```

7. 学習されたデータを用いて、リアルタイムで判定させる

ここではまず Raspberry Pi 上で TensorFlow Lite（0.1.2）と OpenCV（4.1.0.25）が動くように準備していきましょう。以下、LXTerminal 上で操作を行いましょう。

まず各自のホームディレクトリ(/home/pi)に移動しましょう。

```
cd
```

次に Project ディレクトリ内に移動しましょう。

```
cd Project
```

①仮想環境のインストール
pip によるパッケージの導入状態をプロジェクトごとに独立させるために前回同様に仮想環境 venv を利用しましょう。以下のコマンドで仮想環境名を TFL として実行しましょう。

```
python3 -m venv TFL
```

仮想環境下のコマンドや情報を保存する仮想環境名のディレクトリ（ここでは TFL）が作成されます。

それでは以下のコマンドで仮想環境 TFL を有効にします。なお、以下のコマンドは Project ディレクトリ直下から相対パスで activate コマンドを読み込んでいます。

```
source TFL/bin/activate
```

仮想環境が有効になると、コマンドプロンプトの前のカッコ内に仮想環境名が表示されます。ここでは例えば以下の様に表示されるはずです。再起動したときに activate するのを忘れないように注意しましょう。

```
(TFL) pi@raspberry:~ $
```

以上で TensorFlow Lite と OpenCV をインストールする準備が整いました。なお、仮想環境を終了するときは以下のコマンドで終了します。試しに終了してみた人は、もう一度上記の activate コマンドを実行して仮想環境 TFL を有効にしてください。

```
deactivate
```

下の演習は venv の仮想環境 TFL 内で実施していきます。ここでは、学習済みのモデルを使った物体認証を Raspberry Pi 上で動作させてみましょう。

②OpenCV（4.1.0.25）のインストール

まず、OpenCV をインストールします。インストール方法は前回と同様に、OpenCV の動作に必要となるライブラリをインストールします。ライブラリの詳細は前回の資料を参照してください。操作になれていない人は 1 つずつインストールしましょう。

```
sudo apt -y install libhdf5-dev libhdf5-103 libqtgui4 libqtwebkit4
                libqt4-test python3-pyqt5 libatlas-base-dev libjasper-dev
```

バージョンによって　libhdf5-XXX　の XXX が変更する可能性があるのでエラーが出た場合はエラーメッセージに注意しましょう。

python 用の OpenCV（バージョン 4.1.0.25）を pip3 でインストールします。

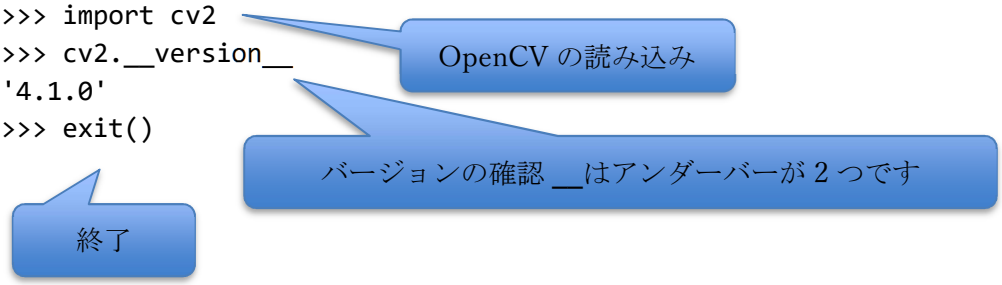
```
pip3 install opencv-python==4.1.0.25
```

ダウンロードが 100%になる前に終了してしまう場合は、以下の様にタイムアウトのオプションをつけて実行しましょう。

```
      pip3 install opencv-python==4.1.0.25 --default-timeout=1000
```

opencv-python をインストールしたら動作するかどうか python3 の対話モードから確認しましょう。

```
$ python3
Python 3.7.3 (default, Feb  21 2020, 18:55:00)
[GCC 8.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import cv2
>>> cv2.__version__
'4.1.0'
>>> exit()
```



もし、python の対話モードの import cv2 で以下の様なエラーが出た場合は、以下の設定を行いましょう。エラーが出なかった場合は次のページに進みましょう。

```
ImportError: /home/pi/cv2/cv2.cpython-37m-arm-linux-gnueabihf.so:
undefined symbol: __atomic_fetch_add_8
```

ターミナルでパスを通す以下の設定を実行しましょう。

```
LD_PRELOAD=/usr/lib/arm-linux-gnueabihf/libatomic.so.1
```

再起動後も上記の設定を有効にしたい場合はホームディレクトリにある隠し設定ファイルである.bashrc に設定を書き込みます。
テキストエディタで.bashrc を開き次の 1 行を追記します。

```
export LD_PRELOAD=/usr/lib/arm-linux-gnueabihf/libatomic.so.1
```

ファイルは、テキストエディタ mousepad で開きましょう。
修正した.bashrc は、以下のコマンドで読み込んで反映させます。

```
source .bashrc
```

③TensorFlow Lite インタープリターのインストール

次に、Raspberry Pi に TensorFlow Lite を実行する TensorFlow Lite インタープリターをインストールします。インストールするには、pip3 install にプラットフォームにあった Python ホイール URL を定します。例えば、Raspberry Pi OS（Python3.7）の場合は以下のようになります。

```
pip3 install https://dl.google.com/coral/python/tflite_runtime-
2.1.0.post1-cp37-cp37m-linux_armv7l.whl
```

Python ホイール URL の表

プラットホーム	Python のバージョン	URL
Linux (ARM 32)	3.6	https://dl.google.com/coral/python/tflite_runtime-2.1.0.post1-cp36-cp36m-linux_armv7l.whl
	3.7	https://dl.google.com/coral/python/tflite_runtime-2.1.0.post1-cp37-cp37m-linux_armv7l.whl
	3.8	https://dl.google.com/coral/python/tflite_runtime-2.1.0.post1-cp38-cp38-linux_armv7l.whl
Linux (ARM 64)	3.6	https://dl.google.com/coral/python/tflite_runtime-2.1.0.post1-cp36-cp36m-linux_aarch64.whl
	3.7	https://dl.google.com/coral/python/tflite_runtime-2.1.0.post1-cp37-cp37m-linux_aarch64.whl
	3.8	https://dl.google.com/coral/python/tflite_runtime-2.1.0.post1-cp38-cp38-linux_aarch64.whl
Linux (x86-64)	3.6	https://dl.google.com/coral/python/tflite_runtime-2.1.0.post1-cp36-cp36m-linux_x86_64.whl
	3.7	https://dl.google.com/coral/python/tflite_runtime-2.1.0.post1-cp37-cp37m-linux_x86_64.whl
	3.8	https://dl.google.com/coral/python/tflite_runtime-2.1.0.post1-cp38-cp38-linux_x86_64.whl
macOS 10.14	3.6	https://dl.google.com/coral/python/tflite_runtime-2.1.0.post1-cp36-cp36m-macosx_10_14_x86_64.whl
	3.7	https://dl.google.com/coral/python/tflite_runtime-2.1.0.post1-cp37-cp37m-macosx_10_14_x86_64.whl
Windows 10	3.6	https://dl.google.com/coral/python/tflite_runtime-2.1.0.post1-cp36-cp36m-win_amd64.whl
	3.7	https://dl.google.com/coral/python/tflite_runtime-2.1.0.post1-cp37-cp37m-win_amd64.whl

TensorFlow Lite インタープリターのインストールは以上で終わりです。

それでは TensorFlow Lite のオフィシャルサイト（https://www.tensorflow.org/lite）が公開しているイメージ分類（image classification）のデモを動かして TensorFlow Lite インタープリターが動作するか確認してみましょう。

まず Project ディレクトリから以下のコマンドを実行して、サンプルファイルをダウンローします。

```
git clone https://github.com/tensorflow/examples --depth 1
```

次に、以下のコマンドでダウンロードを実行する shell プログラムのあるディレクトリ（examples/lite/examples/image_classification/raspberry_pi）まで移動します。

```
cd examples/lite/examples/image_classification/raspberry_pi
```

移動したら次のコマンドでデモに必要なファイルをホームディレクトリに作成した Project ディレクトリ内にダウンロードします。なお、ディレクトリに名前が Project でない場合は、自分のディレクトリ名に修正して実行してください。

```
bash download.sh /home/pi/Project/.
```

ダウンロードが終わったら次のコマンドでデモを起動します。なお、このデモでは Raspberry Pi に直接接続されたディスプレイの最前面にデモ画面を重ねて表示するようにプログラムされています。画面の移動や縮小などのウィンドウ操作はできないので注意してください. また、終了するときもコマンドプロンプト上で [Ctrl] ＋ [c] キーを押してプロセスを終了する仕様になっています。リモート接続ではなく直接接続した画面で操作していた場合、コマンドプロンプトにデモ画面が重なって終了操作が難しくなる点に注意してください。

```
python3 classify_picamera.py
  --model /home/pi/Project/mobilenet_v1_1.0_224_quant.tflite
  --labels /home/pi/Project/labels_mobilenet_quant_v1_224.txt
```

1 行

動作が確認できたら TensorFlow Lite インタープリターのインストールは成功です。

カレントディレクトリが ~/examples/lite/examples/image_classification/raspberry_pi になっているので、以下のコマンドでホームディレクトリに作成した Project ディレクトリに戻っておきましょう。

```
cd ~/Project
```

④15b_detectRoi_TFL.py によるリアルタイム検出
前回同様のリアルタイム検出を Raspberry Pi 上で実現しましょう。学習済みの重みファイルである converted_model.tflite と 15b_detecRoi_TFL.py を Project ディレクトリに入れて、15b_detecRoi_TFL.py を以下のコマンドで実行しましょう。

```
python3 15b_detecRoi_TFL.py
```

検出されると緑色の枠が赤色の枠に色が変わります。

プログラムを終了するときは ESC キーを押しましょう。

以上で前回の演習の中身を TensorFlow Lite に入れ替えて Raspberry Pi 上で実現する事ができました。

⑤MobileNet によるリアルタイム検出
デモ用にダウンロードしてきた mobilenet_v1_1.0_224_quant.tflite と labels_mobilenet_quant_v1_224.txt を使ってリアルタイム検出を実施してみましょう。ここではラベル labels_mobilenet_quant_v1_224.txt を表示するように改良したプログラム 15c_dRoi_TFL_fDemo.py を使いましょう。15c_dRoi_TFL_fDemo.py を以下のコマンドで実行します。

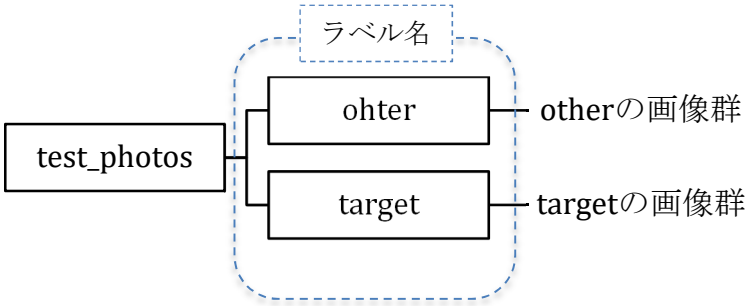
```
python3 15c_dRoi_TFL_fDemo.py
```

デモと同じように動作しているか確認しましょう。プログラムを終了するときは ESC キーを押しましょう。

⑥TensorFlow Lite モデルメーカーを使ったモデル作成
Google colab で 15d_TFL_ModelMaker.ipynb を開き、TensorFlow Lite 用のモデルを作成するモデルメーカーを用いてモデルを作成してみましょう。ここでは最初に利用したプログラム 14-1a_AL_ConvertTFL.ipynb と同じ流れでモデルメーカーからモデルを作ってみましょう。target.jpg と other.jpg を 2 つ用意して（先ほど作った画像でよい）、15d_TFL_ModelMaker.ipynb を使って画像を水増ししてモデルを作成しましょう。作成したらモデル（mode.tflite）とラベル（labels.txt）をダウンロードして、15e_detecRoi_mMaker.py からそれらを読み出してリアルタイム検出をしてみましょう。15e_detecRoi_mMaker.py を以下のコマンドで実行します。

```
python3 15e_detecRoi_mMaker.py
```

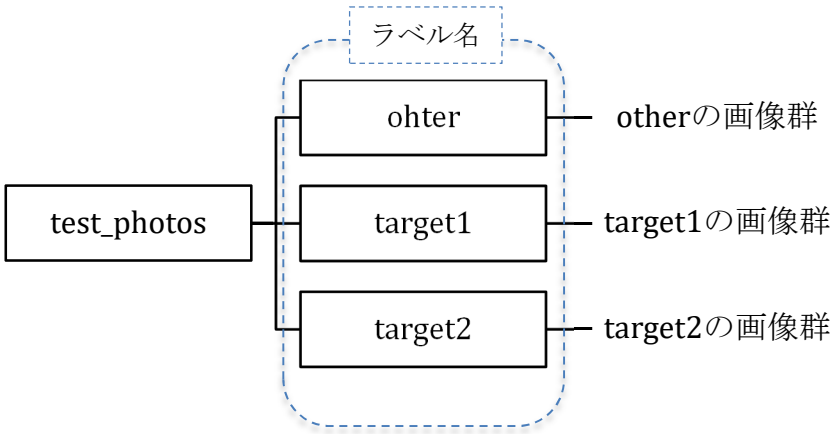
ここでは Google colab 内で学習用の画像を以下の図のように配置しています。なお、以下の図の□はフォルダになります。各画像をまとめているフォルダ名がラベルの名前になります。



最初のリアルタイム検出と同じように動作したか確認しましょう。プログラムを終了するときは ESC キーを押しましょう。

演習：target を複数にしたモデルの作成

先ほど利用したプログラム 15d_TFL_ModelMaker.ipynb をもとに、target を複数にして検出するモデルを各自で作成し、15e_detecRoi_mMaker.py で動作するか確認してみましょう。なお、フォルダが異なっていれば画像の名前は同じものがあっても特に問題ありません。例えば、target1/target.1.jpg と target2/target.1.jpg は画像名が同じですが、フォルダが異なるので特に問題ありません。



Tensorflow Lite モデルメーカーの主なハイパーパラメータ

```
model = image_classifier.create(train_data,
                                model_spec=model_spec.mobilenet_v2_spec,
                                validation_data=validation_data,
                                epochs=10,
                                batch_size=30,
                                train_whole_model=True,
                                learning_rate=0.01,
                                shuffle=True)
```

train_data：学習データの指定

model_spec：もとになるモデルの指定。デフォルトは EfficientNet-Lite0。現在、EfficientNet-Lite models（model_spec.efficientnet_lite0_spec、model_spec.efficientnet_lite1_spec、model_spec.efficientnet_lite2_spec、model_spec.efficientnet_lite3_spec、model_spec.efficientnet_lite4_spec）、MobileNetV2（model_spec.mobilenet_v2_spec）、ResNet50（model_spec.resnet_50_spec）をサポートしている。

validation_data：検証用データの指定。無指定の場合、検証をスキップする。デフォルトではスキップ。例えば、以下の様にデータ分割を続けて実行して、検証用データを準備することができる。

```
train_data, rest_data = data.split(0.8) #train_data:rest_data=8:2 に分割
validation_data, test_data = rest_data.split(0.5) #rest_data を更に半分に分割
```

batch_size：バッチサイズの指定

epochs：エポック数の指定

train_whole_model：True/False。True の場合はモデル全体で学習する。False の場合は最上位層の分類層のみ。デフォルトでは False。

learning_rate：学習率の指定

shuffle：True/False。データをシャッフルするかどうか。デフォルトでは False。