

# 令和 4 年度卒業研究報告書

深層学習を用いたゴミ分別支援ツールの作成

情報技術科 遠藤 綾李

指導教員 ソソラ

# 目次

第 1 章	はじめに .....	1
第 2 章	研究概要 .....	2
2.1	概要 .....	2
2.2	YOLO .....	2
2.2.1	バウンディングボックス .....	3
2.2.2	信頼度 .....	4
2.2.3	予測とクラスごとの色分け .....	6
2.2.4	NMS(Non-maximum suppression) .....	7
第 3 章	システム設計 .....	8
3.1	システム概要 .....	8
3.2	クラス .....	8
3.3	開発環境 .....	9
3.4	各パッケージ紹介 .....	10
3.4.1	OpenCV .....	10
3.4.1	Keras .....	10
3.4.1	Python .....	11
3.4.1	TensorFlow .....	11
3.5	開発フロー .....	12
第 4 章	データセット作成 .....	13
4.1	画像収集 .....	13
4.2	ラベリング .....	13
4.2.1	ラベリング結果 .....	15
4.2.2	ラベリングデータ(txt ファイル) .....	17

4.2.3	クラス定義ファイル .....	17
4.3	モデル生成 .....	18
4.3.1	MyDrive のマウント .....	18
4.3.2	Google Colaboratory に YOLO をインストール .....	19
4.3.3	ラベリングしたデータセットの移動と定義ファイル記述 .....	19
4.3.4	モデル生成時のコード .....	21
4.3.5	TensorBoard 結果 .....	27
4.3.6	TensorBoard を使用しない場合 .....	29
4.4	モデル移植 .....	30
第 5 章	物体検出実装 .....	32
5.1	物体検出 .....	32
5.2	検出概要 .....	33
5.3	検出結果 .....	33
5.3.1	ペットボトル, カンを正しく認識するケース .....	34
5.3.2	誤って認識してしまうケース .....	34
5.3.1	その他のゴミを正しく認識するケース .....	35
第 6 章	問題点と精度の改善 .....	36
6.1	誤認識の問題 .....	36
6.2	解決方法 .....	36
6.3	検出率の変化 .....	37
第 7 章	多種多様な学習ツール .....	37
7.1	「Teachable Machine」 .....	38
7.2	「Azure Custom Vision」 .....	38
第 8 章	おわりに .....	40



# 第1章 はじめに

資源回収において、自動販売機の付近に設置されているリサイクルボックスにビニール傘や電池、弁当箱などの異物が混入し、分別作業に追われている状況を確認している。

一方、手をかざすと自動的に開くセンサー付きのゴミ箱が普及しつつある。このようなことから、カメラ付きのリサイクルボックスがあれば便利だと私は考えた。

そこで、授業で学習した深層学習アルゴリズムを用いて、カメラの画像からごみの回収ができるかできなかを判別するモデルを制作してみたいと考えた。

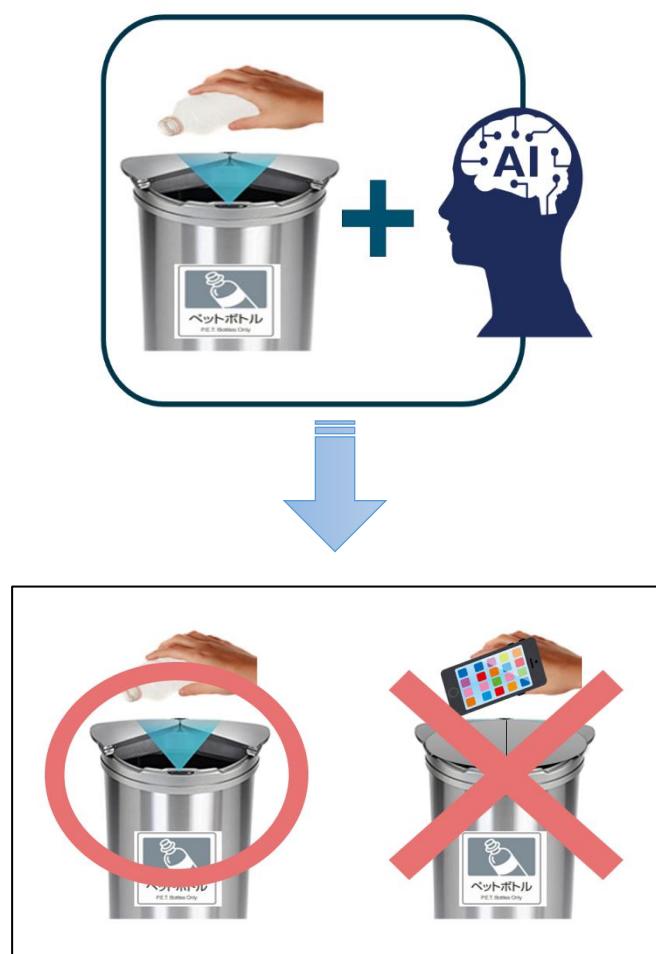


図1.イメージ

## 第2章 研究概要

### 2.1 概要

本研究では、「リアルタイムでカメラの画像からごみの検出をする」ことを目標とするため、物体検出アルゴリズムの一つである YOLO を用いて「ペットボトル」「カン」の二種類の判別モデルを作成し、実験検証を行う。なお、本研究は教師あり学習である。

### 2.2 YOLO

YOLO はリアルタイムオブジェクト検出アルゴリズムである。YOLO は You Look Only Once の頭文字を取ったスラングで、名前の通り検出窓をスライドさせる仕組みを用い、画像を一度 CNN に通すことでオブジェクト検出をすることができる。

以下に参考文献 (2) に基づき YOLO の検出アルゴリズムについてまとめて記述する。

#### 1) 画像を $S \times S$ の小さな正方形に分割する

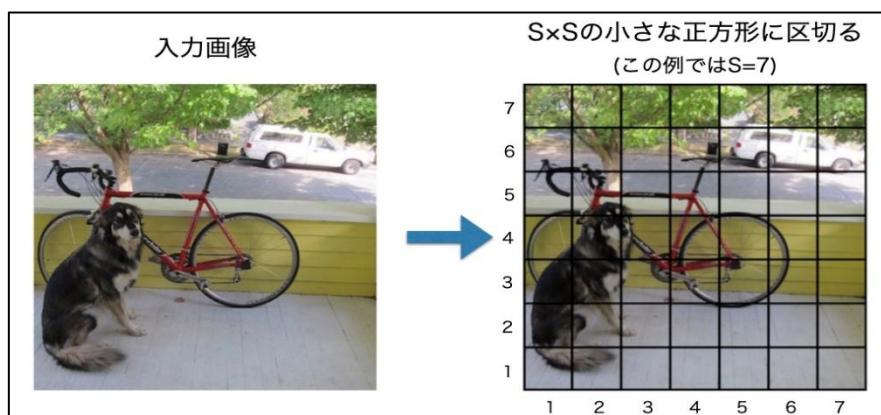


図 2. 画像の分割

- ・入力された正方形の画像をさらに小さな  $S \times S$  個の正方形に分割する。
- ・この小さな正方形を論文内では「グリッドセル(grid cell)」と呼ぶこととする

## 2) バウンディングボックスの推定

各グリッドセルは、B 個のバウンディングボックス（あらかじめ設定されるもので、論文では  $B=2$  に設定）を持ち、それらのボックスの信頼スコアを予測。

### 2.2.1 バウンディングボックス

各小さな正方形に B 個、配置される長方形や四角形を指す。

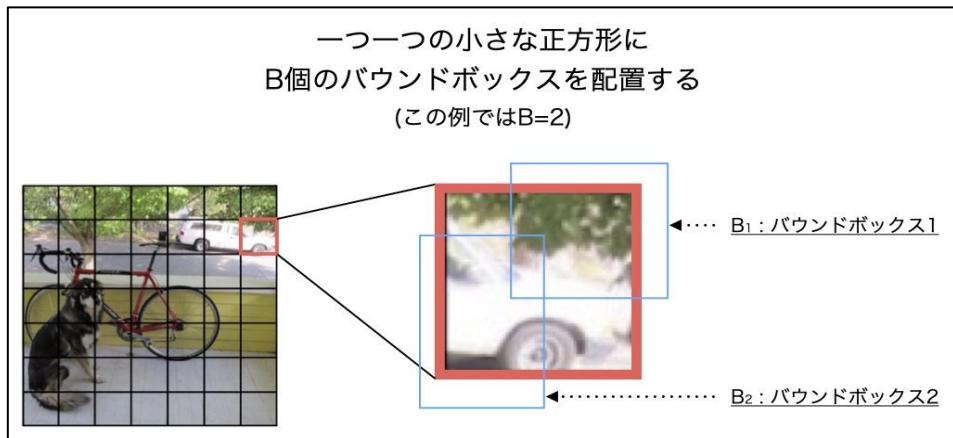


図 3. バウンディングボックスの配置

また、一つのバウンディングボックスは 5 個の特徴で構成されている。

表 1. 特徴の構成

x	バウンディングボックスの中心の x 座標
y	バウンディングボックスの中心の y 座標
w	バウンディングボックスの横幅
h	バウンディングボックスの縦幅
F	信頼度

## 2.2.2 信頼度

バウンディングボックスに物体が含まれているかどうかの確率を示す. 1に近づくほど物体がある可能性が高い. また, 信頼度の範囲は0から1の値である.

$$0 \leq \text{信頼度} \leq 1$$

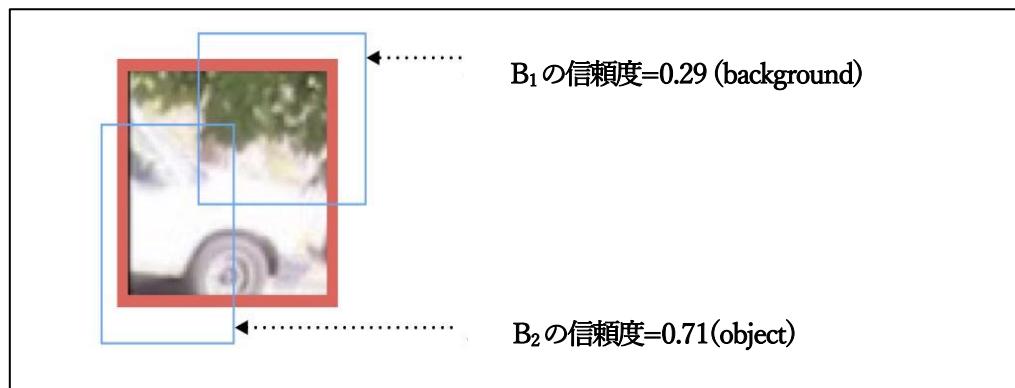


図 4.信頼度算出例

各バウンディングボックスの信頼度は以下の式で表される.

$$\text{信頼度}(Confidence) = Pr(\text{Object}) * IOU$$

すべての小さな正方形に対し, バウンディングボックスを描いた結果は以下の通りである.

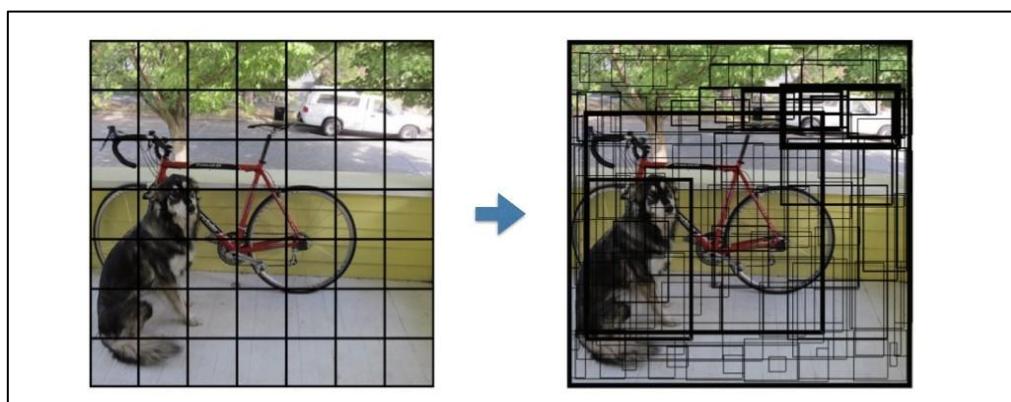


図 5.描写結果

### 3)物体の予測

各グリッドセルはC個のクラスに対する条件付きクラス確率  $P(\text{Class}_i | \text{Object})$  を予測（論文中では  $C=20$  とする）。ここで計算された「条件付きクラス確率」と 2.2.2 で算出される「個々のバウンディングボックスの信頼スコア」を掛け合わせることで、バウンディングボックス毎のクラスに対する信頼スコアを得ることができる。

$$Pr(\text{Class}_i / \text{Object}) * Pr(\text{Object}) * IOU = Pr(\text{Class}_i) * * IOU$$

この信頼度スコアは「各バウンディングボックスの信頼度」と「各クラスの予測確率」を表している。この信頼度スコアに基づき、どのバウンディングボックスが対象とするクラスの物体を正確に検出しているかを判断している。

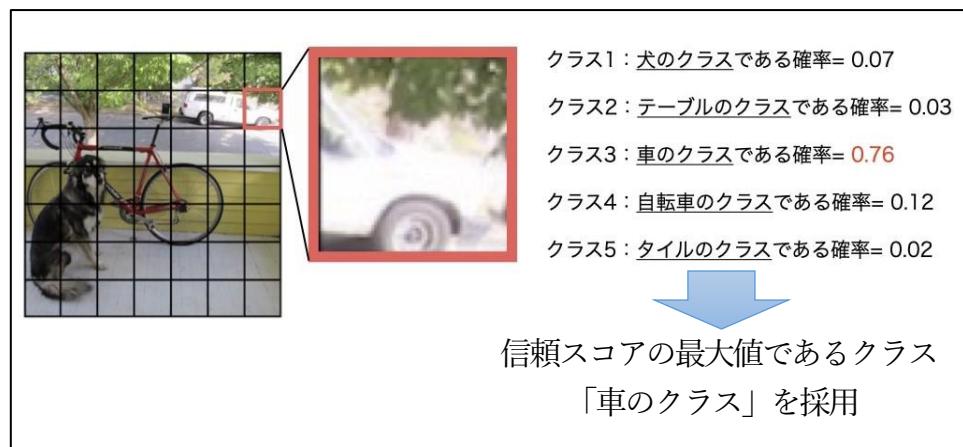


図 6. 物体の予測

### 2.2.3 予測とクラスごとの色分け

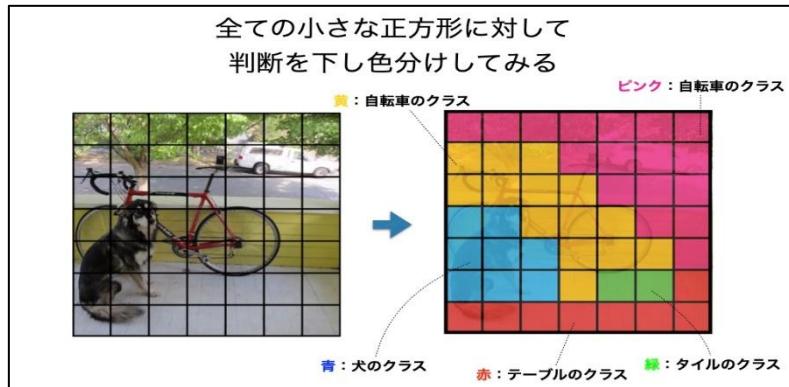


図7. クラスごとに色分け

全ての小さな正方形が何かしらの種類のクラスに所属する。

### 4) 2,3 の結果から物体検出

信頼度の高いバウンディングボックスを基準に NMS(Non-Maximum Suppression)という手法で選別する。

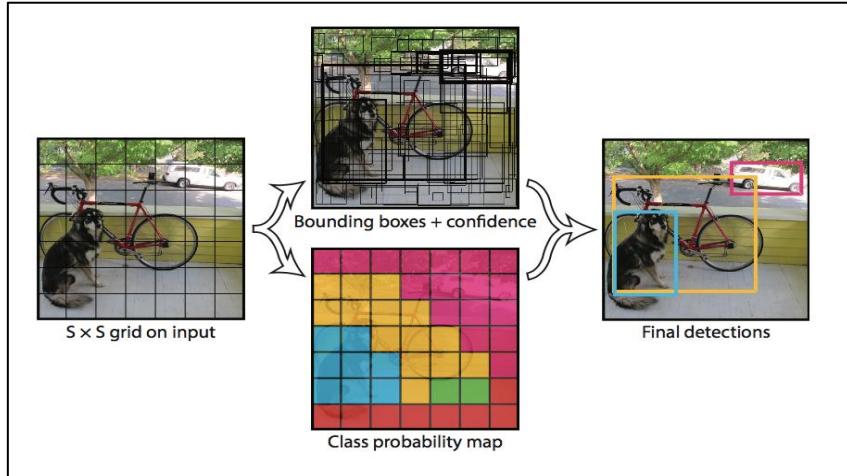


図8. 予測結果

#### 2.2.4 NMS(Non-maximum suppression)

NMS は,たくさんのバウンディングボックスから適切なものを選出するため,IoU(基準の枠とその他の枠の重なり具合)値が大きい領域を閾値で抑制する手法である.

NMS の流れを以下に示す.

1. クラスごとに、一番信頼度の高い枠を選出
2. その枠の IoU を調べ,一定以上の割合で重なっている枠を消去

# 第3章 システム設計

## 3.1 システム概要

カメラの画像から、ゴミの回収可能、不可能を判別するモデルを制作することに決定した。ペットボトルのリサイクルボックスであったら、ペットボトルが検出された場合のみ蓋が開き、ペットボトル以外のものが検出された場合蓋が開かないといったシステムとする。システムイメージを以下に記す。



図9.システムイメージ

## 3.2 クラス

データセット作成の際のクラスを以下の通り示す。

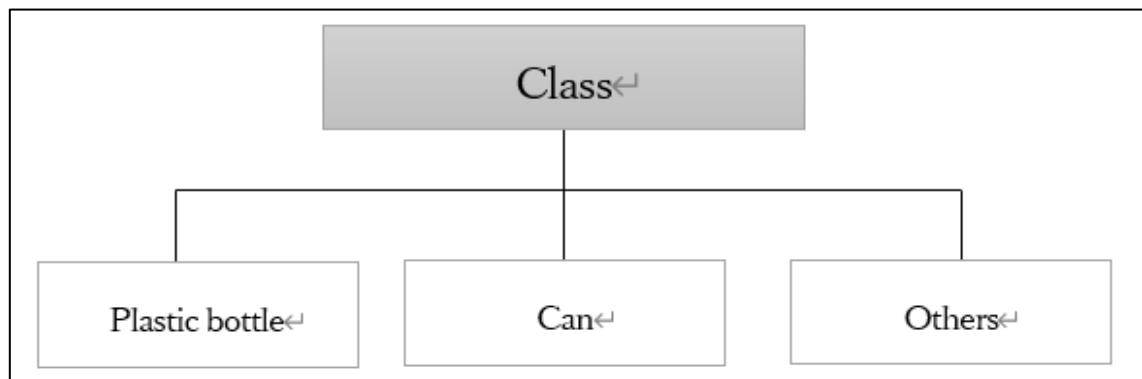


図10.クラス(Plastic bottle: ペットボトル, Can: カン, Others: 上記以外のゴミ(異物と認識))

また,データ詳細は以下の通りである.

表2.データの詳細

訓練データ	Plastic bottle : 100 枚 Can : 100 枚 Others : 100 枚
テストデータ	Plastic bottle : 50 枚 Can : 50 枚 Others : 30 枚
画像サイズ	640×480
学習回数	1000 回

### 3.3 開発環境

開発環境について表3.2に示す.

表3.開発環境

OS	Windows10
主ツール	labelImg Google Colaboratory
言語	Python = 3.8.3
パッケージ	OpenCV=4.5.4.60, Keras=2.10.0 Tensorflow=2.10.0,

## 3.4 各パッケージ紹介

各パッケージの詳細を以下に示す.

### 3.4.1 OpenCV

OpenCV (Open Source Computer Vision Library) は、画像・動画に関する処理機能をまとめたオープンソースのライブラリである。Intel が開発したライブラリで、OSS として提供されているため、誰でも無料で使用することができる。

OpenCV の導入により、画像や動画の中に存在する物体の位置情報やパターン、動きをプログラムが識別できるようになっている。

Windows や Linux、iOS や Android などさまざまな OS に対応している。Raspberry Pi などの端末上で利用することも可能。

### 3.4.1 Keras

ニューラルネットワークライブラリの一つ。以下のようなニューラルネットワークのモデルをコードで記述する際に必要なパッケージとなる。

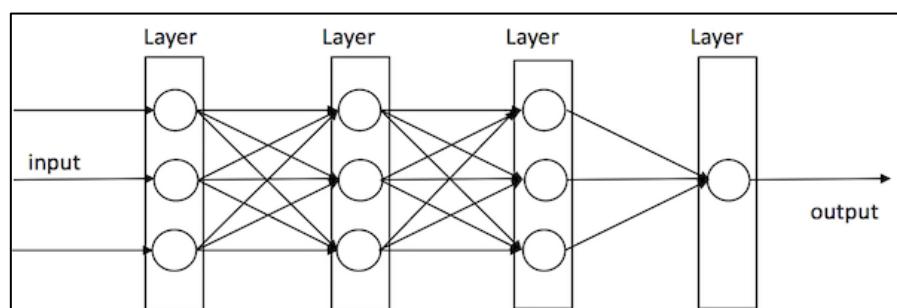


図 11.ニューラルネットワークモデル

### 3.4.1 Python

Python は人工知能分野での開発に広く用いられているプログラミング言語である。行列演算、画像認識をはじめとするさまざまなライブラリも豊富に存在している。Python で OpenCV を利用する場合は、専用のライブラリ「OpenCV-python」を利用するため、事前のインストールが必要である。

### 3.4.1 TensorFlow

TensorFlow とは、Google の機械学習/ディープラーニング/多層ニューラルネットワークライブラリである。データフローグラフを使用したライブラリで複雑なネットワークを分かりやすく記述できることが特徴である。高い汎用性により研究レベルから実プロダクトにまで活用できる。本研究では、TensorBoard を使用するために導入したパッケージである。

Tensorflow と Keras は役割が類似しているが、両者の比較を以下に示している。どのフレームワークを利用するかは「技術的な要件」、「他の必要条件」、「利便性」の三つの要素を考慮していくといふと考える。具体的には、以下のような場面で使用される。

Tensorflow

- ・大規模データセット
- ・高いパフォーマンスを要する
- ・画像リサイズなど学習以外の様々な機能が必要となる
- ・物体検出

Keras

- ・モデルの作成に時間がかかる（低レベルの工夫が少ない）
- ・小規模のデータセット
- ・複数のバックエンドのサポートがある

### 3.5 開発フロー

開発フローについて図 12 に示す。

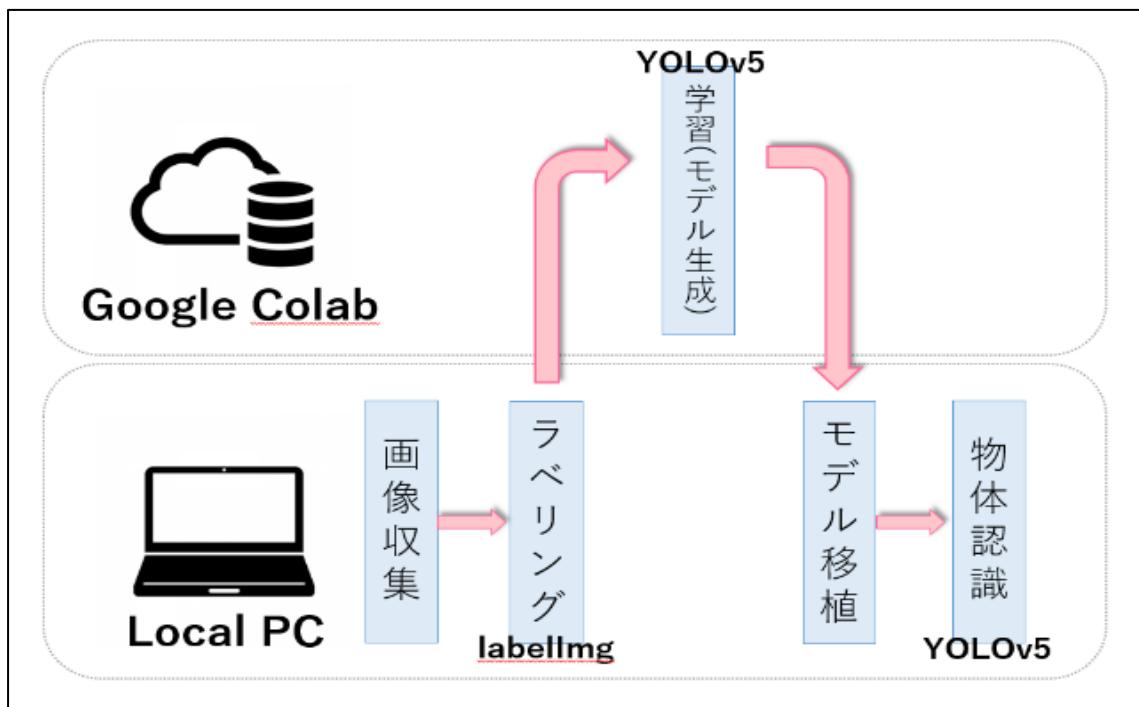


図 12. 開発フロー

1. データセット(画像)の用意
2. labelImg でタグ(正答)付け,YOLO 形式のファイルに変換
3. Google Colaboratory(GPU)で学習
4. 学習済みのモデルを local PC に移植
5. Local PC で YOLO を実行
6. リアルタイムで物体検出

# 第4章 データセット作成

## 4.1 画像収集

物体認識に使用する画像の収集を行う。LocalPCにDataというフォルダを作成し、撮影した画像を格納する。

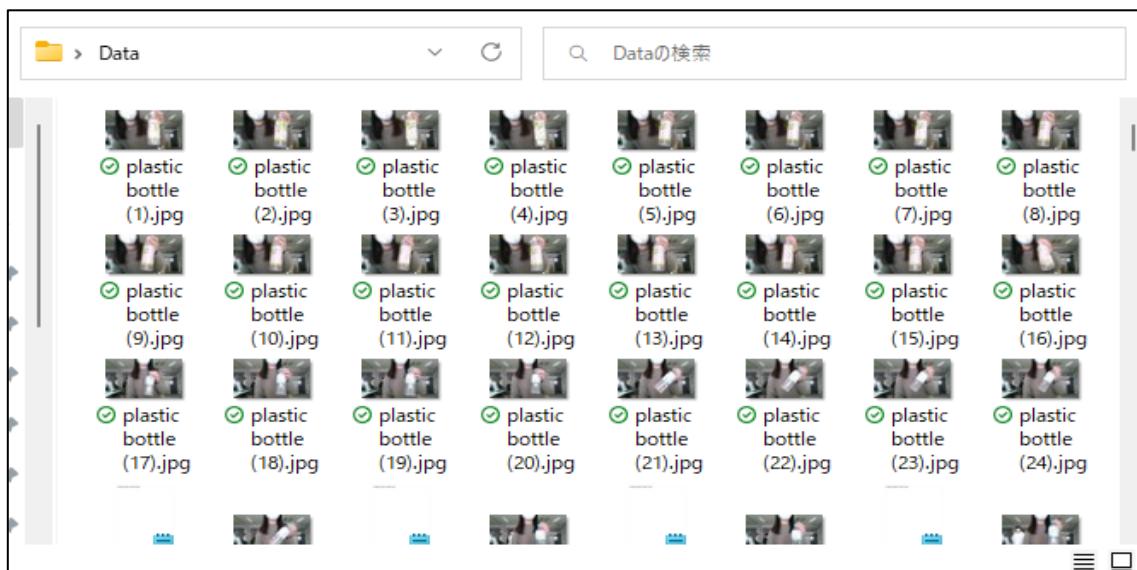


図13 画像収集の様子

## 4.2 ラベリング

ラベリングとは、データに正解となる分類情報を付与すること。いわばAIモデルの「正解」となる振舞いを用意する作業となる。本研究でのラベリングはlabelImgというツールを使用した。

labelImgは、公式サイトからツールをインストールした後、コマンドプロンプトにて labelImgと打ち込むことによって即座に起動をすることができる。

インストール方法や起動方法、使用方法は、別途付録の「labelImgのインストール方法、使い方」ファイルに記載している。

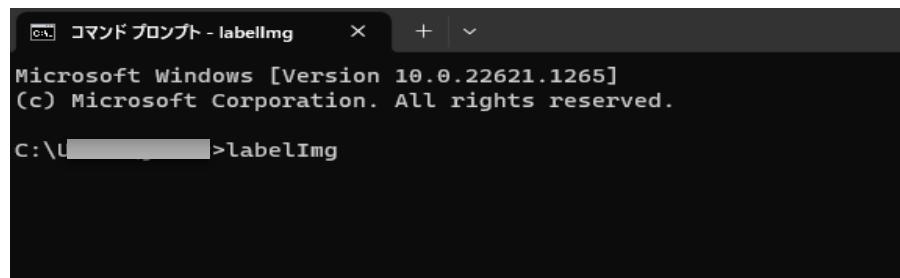


図 14.labelImg 起動方法

起動が完了すると,以下のような画面が表示される.

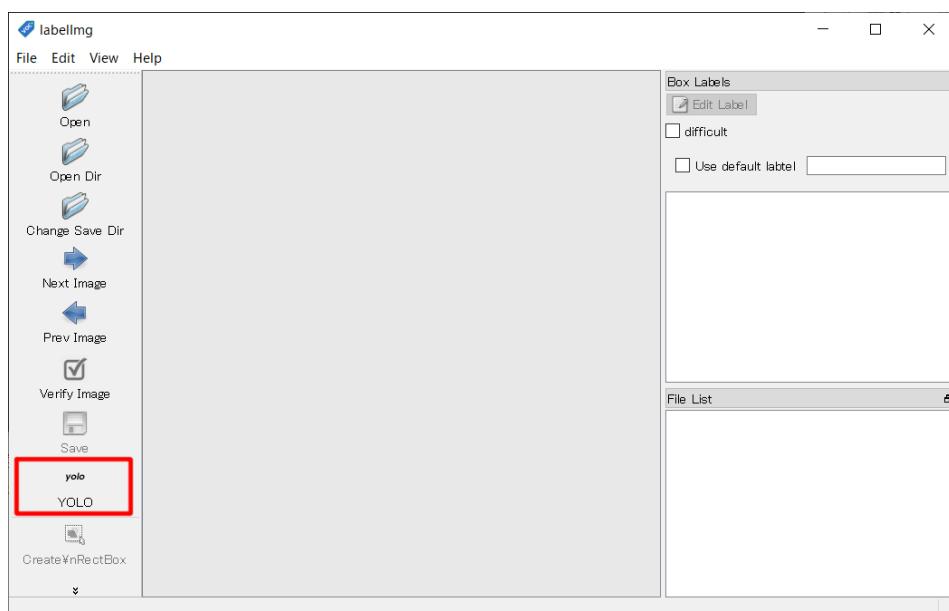


図 15.labelImg の UI

LabelImg はフォーマットをいくつか設定可能である.デフォルトでは「PascalVOC」に設定されているため,必ず「YOLO」に変更を行ってから作業に取り掛かること.

両者の違いは,PascalVOC のフォーマット形式が xml , YOLO の形式が txt という点である.

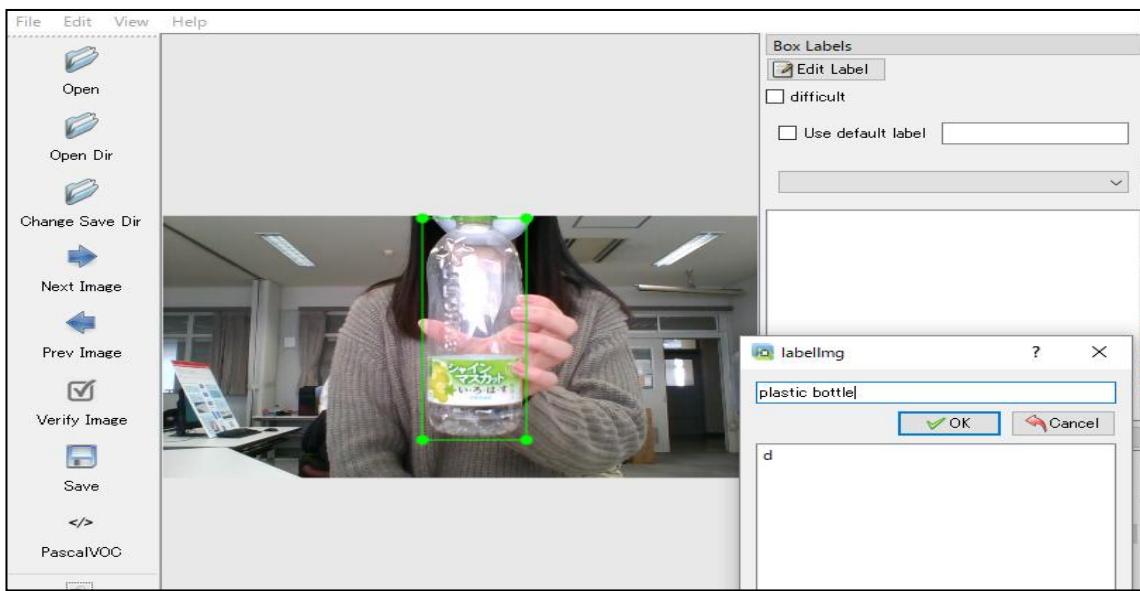


図 16. ラベリング作業の様子

「ペットボトルの画像」は「plastic bottle」、「カン」は「Can」、「その他のもの」は「Others」とラベリングを行った。

#### 4.2.1 ラベリング結果

ラベリングの作業が終わると、画像ファイルに対応した「ラベリングデータ(txt ファイル)」と「クラス定義ファイル」がディレクトリ内に作成される。

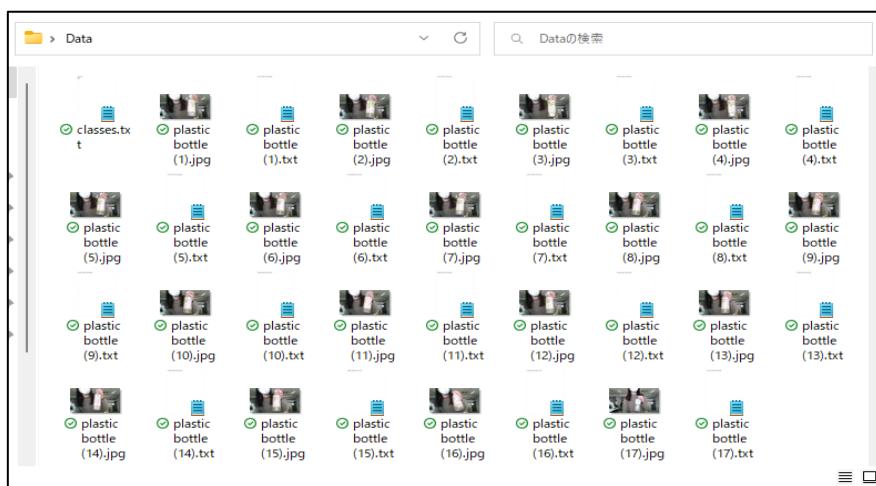


図 17. ラベリング後の様子

また,以下にディレクトリ構造を示す.

```
labelImg_Data/  
|   └── plastic bottle(1).jpg      .....画像ファイル  
|   └── plastic bottle(1).txt      .....ラベリングデータ(txt ファイル)  
|       :  
|   └── plastic bottle(100).jpg  
|   └── plastic bottle(100).txt  
|       :  
|   └── can(1).jpg  
|   └── can(1).txt  
|       :  
|   └── can(100).jpg  
|   └── can(100).txt  
|       :  
|   └── others(1).jpg  
|   └── others(1).txt  
|       :  
|   └── others(100).jpg  
|   └── others(100).txt  
└── classes.txt      .....クラス定義ファイル
```

#### 4.2.2 ラベリングデータ(txt ファイル)

txt ファイル内には、クラスとバウンディングボックスの位置座標が記入されている。

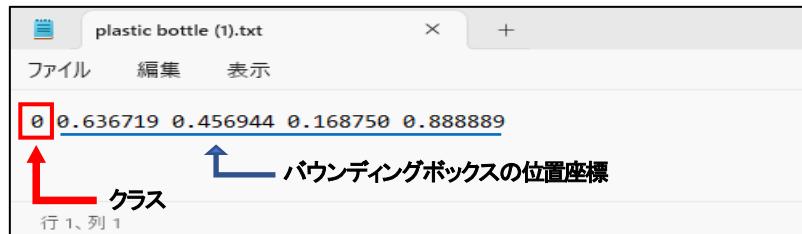


図 18.txt ファイル例

#### 4.2.3 クラス定義ファイル

自動で生成されるクラス定義ファイル「classes.txt」はモデル生成の際に使用しないが、正しくクラス分けが出来ているか確認をすることができる。今回のデータセットでは先頭行からクラス 0、クラス 1、クラス 2 となる。



図 19.classes.txt ファイル例

Plastic bottle……………クラス 0 が定義されていることを表す

Can……………クラス 1 が定義されていることを表す

Other……………クラス 2 が定義されていることを表す

## 4.3 モデル生成

ラベリング作業が終了したらモデル生成を行う。モデル生成は主に,学習ツールである Google Colaboratory を使用する。モデル生成は以下の段階で進める。

1. MyDrive のマウント
2. Google Colaboratory に YOLO をインストール
3. ラベリングしたデータセットの移動
4. 定義ファイルの記述
5. モデル生成時コード

### 4.3.1 MyDrive のマウント

本研究で使用する学習ツール「Google Colaboratory」は基本的には無料で使用が可能だが,新しいインスタンスを起動してから 12 時間が経過してしまうと Python の実行環境の接続が切れる,「ランタイムリセット」が発生してしまう。このランタイムリセットが発生してしまうと,格納していたファイルがすべて削除されてしまうため,作成したデータセットは Google ドライブに格納する。

マウント時,Google Colaboratory に以下のコードを記述する。



```
▶ from google.colab import drive  
drive.mount('/content/drive')
```

Mounted at /content/drive

---

図 20.Drive のマウント

実行結果に「Mounted at /content/drive」と表示されたら,マウントが成功している。

### 4.3.2 Google Colaboratory に YOLO をインストール

モデル生成は yolov5 の train.py プログラムを使用して行う。Google Colaboratory 上に yolov5 をインストールするため、コードを git からダウンロードする。ダウンロードの際のコードを以下に示す。

```
colabコード  
!git clone https://github.com/ultralytics/yolov5
```

図 21. コード

また、YOLOv5 を起動するために必要なライブラリを一括でインストールする。インストールの際のコードを以下に示す。

```
colabコード  
!pip install -r yolov5/requirements.txt
```

図 22. コード

### 4.3.3 ラベリングしたデータセットの移動と定義ファイル記述

Google ドライブの MyDrive 配下に「Data」フォルダを追加し、その中に「4.1.1 ラベリング結果」で作成したラベリングデータを全てコピーする。

*/content/drive/My Drive/Data*

以下にラベリングコピー後のディレクトリ構造を示す。

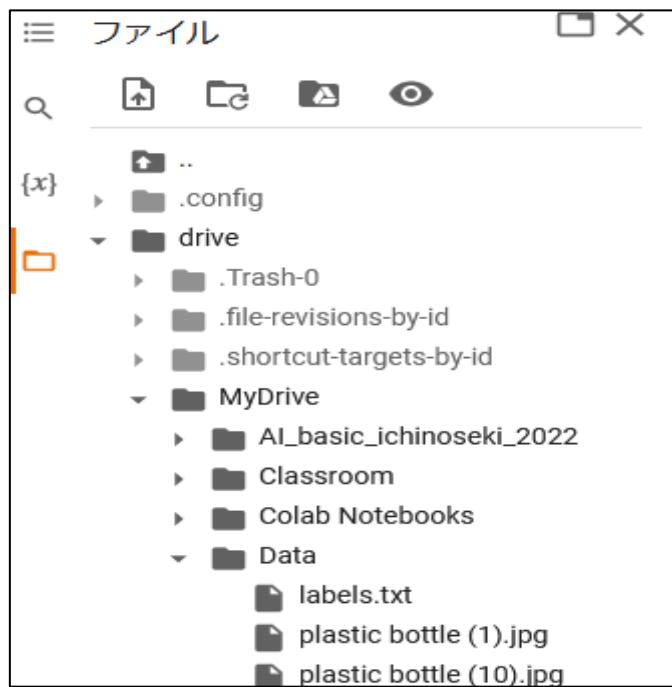


図 23.Google Colaboratory 内ディレクトリ構造

また、モデル作成のための「ラベリングデータ格納場所」、「クラス分類数」、「クラス名」について、設定ファイル「Data.yaml」で定義しなければならない。先程作った Google ドライブの「Data」フォルダ（/content/drive/My Drive/Data）に「Data.yaml」を作成する。

Data.yaml の記述内容は以下の通りである。

```
! Data.yaml ×
C: > [REDACTED] > ! Data.yaml
1 # Data dataset
2
3 # train and val data as 1) directory: path/images/, 2) file: path/images.txt, or 3)
4 #list: [path1/images/, path2/images/]
5 train: /content/drive/My Drive/Data
6 val: /content/drive/My Drive/Data
7
8 # number of classes
9 nc: 3
10
11 # class names
12 names: ['plastic bottle', 'can', 'others']
```

図 24.Data.yaml の記述例

#### 4.3.4 モデル生成時のコード

ここからモデル生成を行っていく。Google Colaboratory 上で cd コマンドを用い,yolov5 ファイルに移動し,学習開始のためのコードを記述,実行する。学習開始時のコードを以下に示す。

```
▶ %cd /content/drive/MyDrive/yolov5  
!python train.py --batch 20 --epochs 1000 --data '/content/drive/My Drive/Data/Data.yaml' --name Data
```

図 25.モデル生成時コード記入例

本研究では batch 数を 20,epoch 数(学習回数)を 1000 にして実施。

また,以下に「バッチ数」と「エポック数」について記す。

<バッチ数>

幾つかに分けたぞぞぞのサブセットに含まれるデータの数をバッチサイズと呼びます。

例えば、1,000 件のデータセットを 200 件ずつのサブセットに分ける場合、バッチサイズは 200 となります。

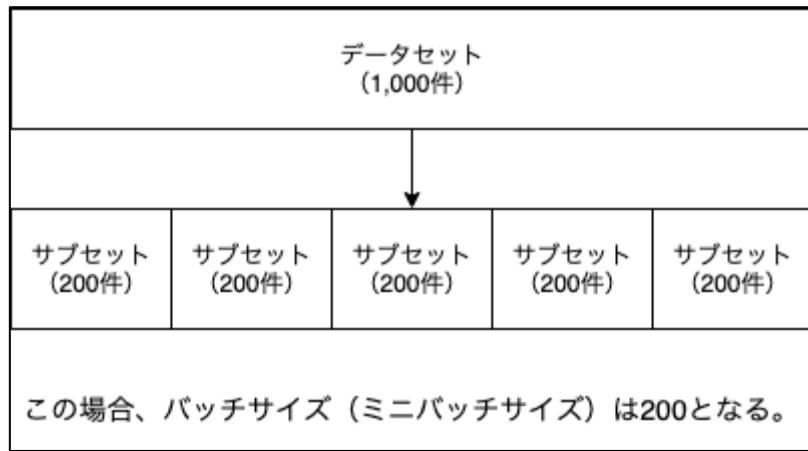


図 26.バッチ数のイメージ

バッチのサイズによって学習における変化について,主に

- ・1つのサンプルデータに対する反応度

- ・1epoch の計算速度

- ・メモリ使用量

が挙げられる。

### ●1つのサンプルデータに対する反応度

バッチの単位が小さければ小さいほど、1つ1つのデータに敏感に反応する。逆にミニバッチの単位が大きければ大きいほど平均化されるため、1つ1つのデータよりもミニバッチ全体の特徴を捉えることができる。

### ●1epoch の計算速度

1epoch の計算速度が変化する。重みの更新の回数が、ミニバッチのサイズが小さいほど多くなるためである。したがって、1つのバッチサイズを大きくすると学習時間が短縮される。

### ●メモリ使用量

ミニバッチ単位でデータを読みとり、そのデータを使用するため、バッチのサイズが大きければ大きいほど、データの読み取り数が多いためメモリの使用量が上がる。

### <エポック数>

エポック数とは、一つの訓練データの全てを使い切って一周した時を1とする、訓練データを何回用いたかを表す数である。エポック数が多くすぎると過学習が生じてしまう可能性があるので適当なエポック数で学習を止める必要がある。

上記の特徴を理解し、慎重にバッチ数とエポック数を選定することが認識率を上げる一つの方法であると考える。

ログの最終行にはどの場所にモデルが生成されたかのパスが表示され、今回は下記のディレクトリに学習モデルが生成される。

*/content/yolov5/runs/train/PET14*

```
Epoch    GPU_mem   box_loss   obj_loss   cls_loss   Instances      Size
49/49     4.66G    0.0177    0.007637  0.001093    2           640: 100% 12/12 [00:12<00:00,  1.03s/it]
          Class   Images   Instances      P       R   mAP50   mAP50-95: 100% 6/6 [00:04<00:00,  1.30it/s]
          all     221      196      0.922      1       0.98      0.876

50 epochs completed in 0.260 hours.
Optimizer stripped from runs/train/PET14/weights/last.pt, 14.3MB
Optimizer stripped from runs/train/PET14/weights/best.pt, 14.3MB

Validating runs/train/PET14/weights/best.pt...
Fusing layers...
Model summary: 157 layers, 7018216 parameters, 0 gradients, 15.8 GFLOPs
          Class   Images   Instances      P       R   mAP50   mAP50-95: 100% 6/6 [00:04<00:00,  1.25it/s]
          all     221      196      0.922      1       0.98      0.875
          plastic bottle  221      83       0.775      1       0.949      0.864
          can     221      51       0.994      1       0.995      0.901
          others   221      62       0.996      1       0.995      0.86

Results saved to runs/train/PET14
```

図27.モデル生成時のログ

モデル生成後に last.pt と best.pt が生成されるが、last.pt はトレーニングの最後のエポックからの重み、best.pt はトレーニング中に記録された最高の重みを示している。今回は best.pt のモデルを利用する。

50 epochs completed in 0.260 hours.

Optimizer stripped from runs/train/PET14/weights/last.pt, 14.3MB

Optimizer stripped from runs/train/PET14/weights/best.pt, 14.3MB

以降からYOLO プログラムを実行した際のログを示す。

```
/content/drive/MyDrive/yolov5

requirements: YOLOv5 requirement "gitpython" not found, attempting AutoUpdate...

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/

Collecting gitpython
  Downloading GitPython-3.1.31-py3-none-any.whl (184 kB)

184.3/184.3 KB 16.7 MB/s eta 0:00:00

Collecting gitdb<5,>=4.0.1
  Downloading gitdb-4.0.10-py3-none-any.whl (62 kB)

  :省略

  Downloading smmap-5.0.0-py3-none-any.whl (24 kB)

  Installing collected packages: smmap, gitdb, gitpython
    Successfully installed gitdb-4.0.10 gitpython-3.1.31 smmap-5.0.0
  requirements: 1 package updated per ['gitpython']

  requirements: ! Restart runtime or rerun command for updates to take effect

  train: weights=yolov5s.pt, cfg=, data=/content/drive/My Drive/PET/trash.yaml,
        hyp=data/hyps/hyp.scratch-low.yaml, epochs=800, batch_size=20, imgsz=640, rect=False,
        resume=False, nosave=False, noval=False, noautoanchor=False, noplots=False, evolve=None,
        bucket=, cache=None, image_weights=False, device=, multi_scale=False, single_cls=False,
        optimizer=SGD, sync_bn=False, workers=8, project=runs/train, name=PET, exist_ok=False,
  Requirement already satisfied: pygments in /usr/local/lib/python3.8/dist-packages (from ipython)
(2.6.1)
```

ToGray(p=0.01), CLAHE(p=0.01, clip\_limit=(1, 4.0), tile\_grid\_size=(8, 8))

train: Scanning /content/drive/My Drive/PET.cache... 196 images, 1 backgrounds, 0 corrupt: 100%

197/197 [00:00<?, ?it/s]

:省略

Plotting labels to runs/train/PET10/labels.jpg...

Unpacking objects: 100% (51/51), 16.80 KiB | 4.00 KiB/s, done.

Image sizes 640 train, 640 val

Using 2 dataloader workers

Logging results to runs/train/PET10

Starting training for 1000 epochs...

Epoch	GPU_mem	box_loss	obj_loss	cls_loss	Instances	Size
0/999	4.58G	0.117	0.02992	0.04436	39	640: 0% 0/10

[00:16<?, ?it/s]From https://github.com/ultralytics/yolov5

4db6757e..5c91daea master -> origin/master

6ac4c48c..8e0c488e benchmarks -> origin/benchmarks

00070f35..343134f7 exp13-soft -> origin/exp13-soft

0/999	4.65G	0.1112	0.02935	0.04092	42	640: 100% 10/10
-------	-------	--------	---------	---------	----	-----------------

[00:28<00:00, 2.82s/it]

Class	Images	Instances	P	R	mAP50	mAP50-95: 100% 5/5
-------	--------	-----------	---	---	-------	--------------------

[00:09<00:00, 1.86s/it]

all	197	196	0.00183	0.525	0.00346	0.000826
-----	-----	-----	---------	-------	---------	----------

Epoch	GPU_mem	box_loss	obj_loss	cls_loss	Instances	Size
-------	---------	----------	----------	----------	-----------	------

1/799	4.65G	0.08821	0.02946	0.03821	43	640: 100% 10/10
-------	-------	---------	---------	---------	----	-----------------

[00:08<00:00, 1.14it/s]

Class	Images	Instances	P	R	mAP50	mAP50-95: 100% 5/5
[00:04<00:00, 1.23it/s]						
all	197	196	0.0965	0.303	0.124	0.0297
Epoch	GPU_mem	box_loss	obj_loss	cls_loss	Instances	Size
2/799	4.65G	0.07216	0.02955	0.03387	45	640: 100% 10/10
[00:10<00:00, 1.07s/it]						
Class	Images	Instances	P	R	mAP50	mAP50-95: 100% 5/5
[00:03<00:00, 1.47it/s]						
all	197	196	0.36	0.307	0.311	0.089
Epoch	GPU_mem	box_loss	obj_loss	cls_loss	Instances	Size
3/799	4.65G	0.06389	0.0259	0.03176	34	640: 100% 10/10
[00:10<00:00, 1.08s/it]						
Class	Images	Instances	P	R	mAP50	mAP50-95: 100% 5/5
[00:03<00:00, 1.26it/s]						
all	197	196	0.259	0.716	0.357	0.105

### 4.3.5 TensorBoard 結果

TensorBoard はモデル生成の学習を可視化するツールである。TensorBoard 結果を取得したい場合は「4.2.4 モデル生成のコード」の実行直前で TensorBoard を起動する必要がある。  
起動方法を以下に記す。

```
%load_ext tensorboard  
%tensorboard --logdir runs
```

図 28. TensorBoard の起動方法

上記のコードを実行すると, TensorBoard が学習を記録する。記録画面は以下の通りである。

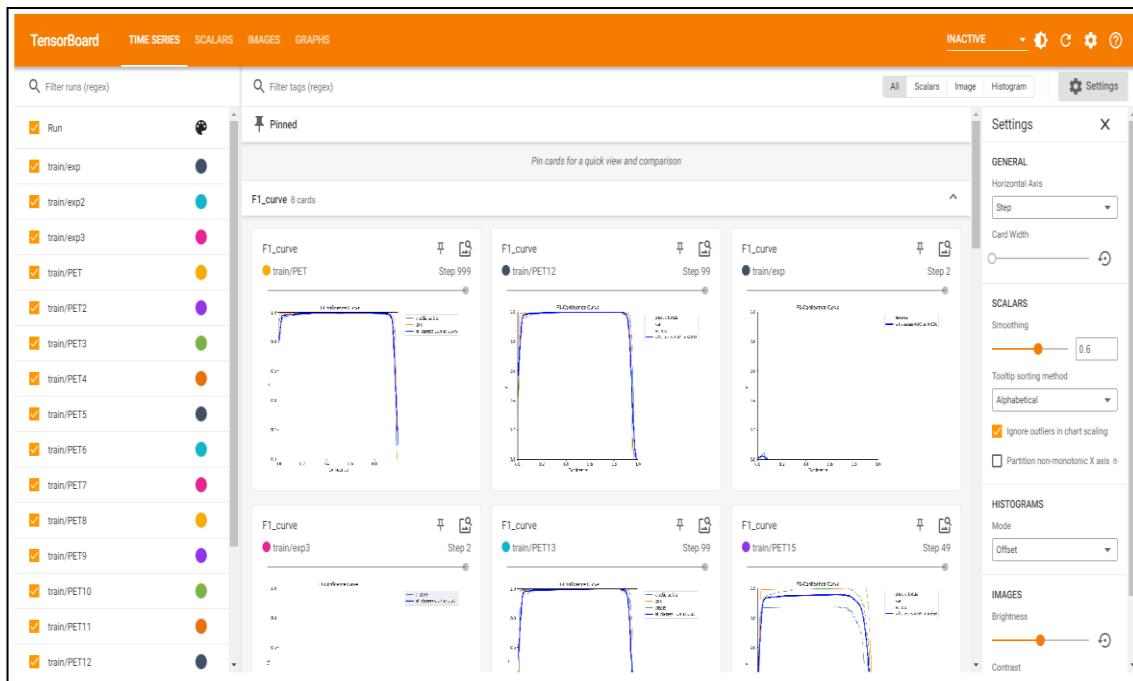


図 29. TensorBoard の記録画面

また,本研究の学習関数を以下に示す.

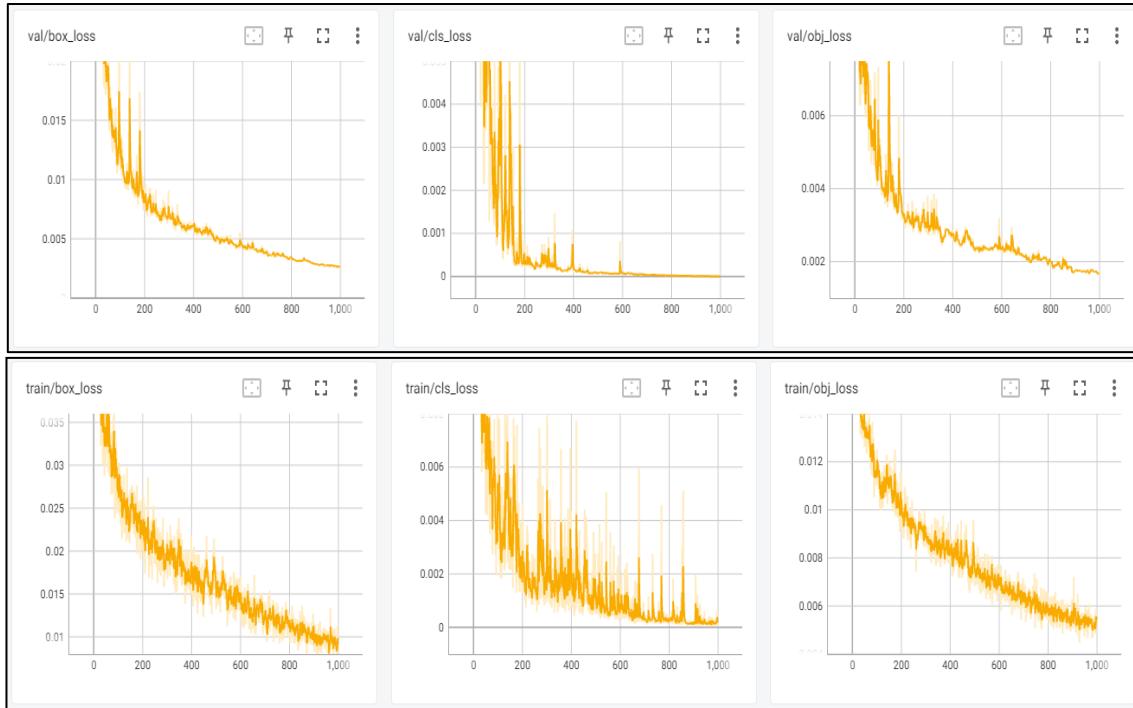


図 30. TensorBoard の学習関数

上記の図は「ペットボトル」モデルの学習における損失数のグラフである.この損失関数のグラフを見ると,初期段階で大きかった loss が徐々に小さくなっているため,正しく学習されていることが分かる.

#### 4.3.6 TensorBoard を使用しない場合

TensorBoard を使用しない場合,yolov5 で簡易的な「モデル生成時の学習結果」を取得することが出来る。「モデル生成時の学習結果」が格納されるディレクトリを以下に示す。

/content/yolov5/runs/train/xxxx

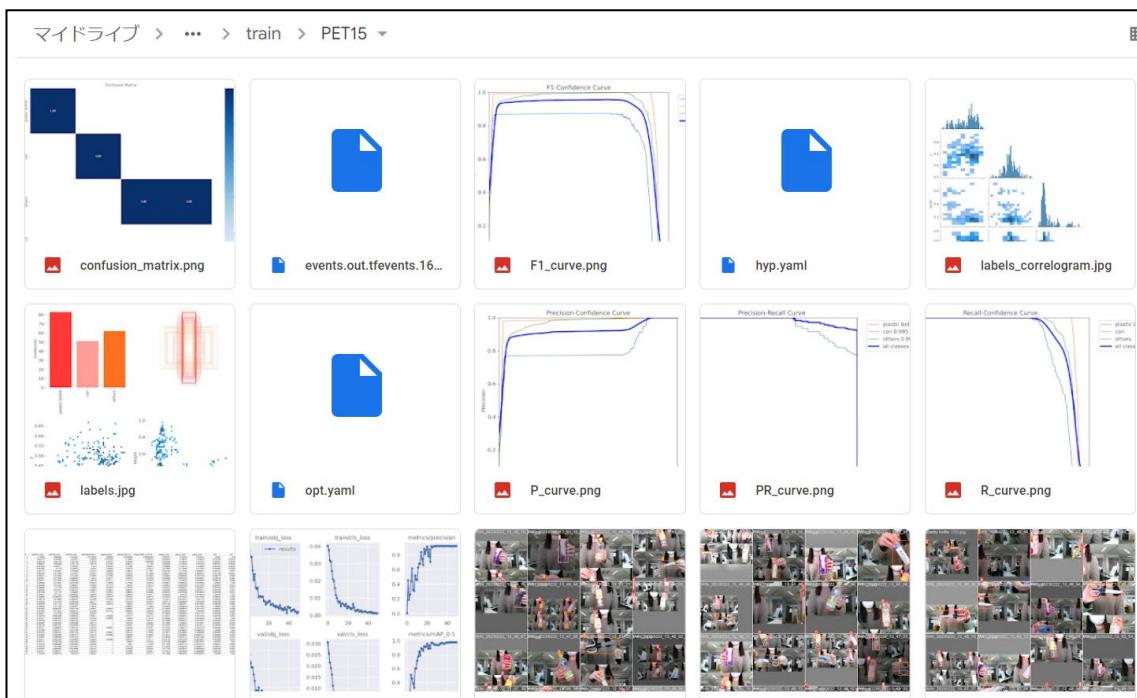


図31.yolov5 内のモデル生成時の学習結果

## 4.4 モデル移植

Local PC では python 仮想環境として anaconda を利用.

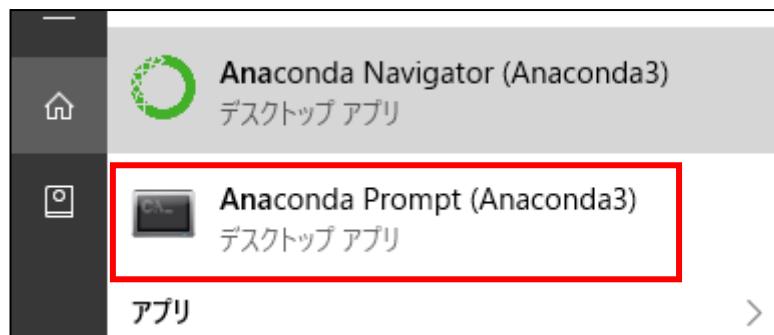


図 32.anaconda Prompt

コマンドプロンプトは「Anaconda Prompt」で行う.起動したら、以下のコマンドを実施して,Local PC に yolov5 の環境を構築する作業を行う.

```
c:¥>conda create -n yolov5 python=3.8 | ①  
c:¥>conda activate yolov5 | ②  
(yolov5) c:¥>git clone https://github.com/ultralytics/yolov5.git  
(yolov5) c:¥>cd yolov5  
(yolov5) c:¥yolov5>conda install pytorch torchvision -c pytorch | ③  
(yolov5) c:¥yolov5>pip install -U -r requirements.txt
```

- ① 「yolov5」 という名前で仮想環境を作成.(python のバージョン 3.8)
- ② 仮想環境 「yolov5」 を起動.
- ③ 必要パッケージのインストール.

また,anaconda のバージョンを以下に示す.

```
(base) C:\$>anaconda --version  
anaconda Command Line client (version 1.7.2)
```

図 33.anaconda バージョン

Google Colaboratory で作成したモデル「best.pt」をダウンロードし,Local PC の yolov5 のディレクトリ直下にコピーする.

C:\yolov5\best.pt

名前	更新日時	種類	サイズ
.github	2022/12/01 14:57	ファイル フォルダー	
__Pycache__	2023/01/26 15:16	ファイル フォルダー	
classify	2022/12/01 14:57	ファイル フォルダー	
data	2022/12/01 14:57	ファイル フォルダー	
models	2023/01/13 13:07	ファイル フォルダー	
runs	2023/01/26 15:16	ファイル フォルダー	
segment	2022/12/01 14:57	ファイル フォルダー	
utils	2023/01/26 15:16	ファイル フォルダー	
.dockerignore	2022/11/29 11:37	DOCKERIGNORE フ...	4 KB
.gitattributes	2022/11/29 11:37	テキスト ドキュメント	1 KB
.gitignore	2022/11/29 11:37	テキスト ドキュメント	4 KB
.pre-commit-config.yaml	2022/11/29 11:37	Yaml ソース ファイル	2 KB
benchmarks.py	2022/11/29 11:37	Python ソース ファイル	8 KB
<input checked="" type="checkbox"/> best.pt	2023/02/24 9:03	PT ファイル	55,395 KB
CONTRIBUTING.md	2022/11/29 11:37	Markdown ソース フ...	5 KB
detect.py	2023/02/24 9:30	Python ソース ファイル	14 KB
export.py	2022/11/29 11:37	Python ソース ファイル	31 KB
hubconfig.py	2022/11/29 11:37	Python ソース ファイル	8 KB
LICENSE	2022/11/29 11:37	ファイル	35 KB
README.md	2022/11/29 11:37	Markdown ソース フ...	38 KB
requirements.txt	2022/11/29 11:37	テキスト ドキュメント	2 KB
setup.cfg	2022/11/29 11:37	Configuration ソー...	2 KB
train.py	2022/11/29 11:37	Python ソース ファイル	33 KB
tutorial.ipynb	2022/11/29 11:37	Jupyter ソース ファイル	53 KB
val.py	2022/11/29 11:37	Python ソース ファイル	20 KB
yolov5s.pt	2023/01/26 15:16	PT ファイル	14,462 KB

図 34. Local PC の yolov5 ディレクトリ直下にコピー

# 第5章 物体検出実装

## 5.1 物体検出

Local PC で以下のコマンドを実行すると、プログラムの実行が可能である。

```
(yolov5) C:\yolov5> python detect.py --source 0 --weight best.pt
```

コマンドを実行すると、以下のようにプログラムが実行する。

```
C:\Users\jandc\yolov5>python detect.py --source 0 --weight best.pt
detect: weights=['best.pt'], source=0, data=data\coco128.yaml, imgsz=[640, 640], conf_thres=0.25, iou_thres=0.45, max_det=1000, device=, view_img=False, save_txt=False, save_conf=False, save_crop=False, nosave=False, classes=None, agnostic_nms=False, augment=False, visualize=False, update=False, project=runs\detect, name=exp, exist_ok=False, line_thickness=3, hide_labels=False, hide_conf=False, half=False, dnn=False, vid_stride=1
YOLOv5 v6.2-119-g03f2ca8 Python-3.8.3 torch-1.12.1+cpu CPU

Fusing layers...
Model summary: 157 layers, 7015519 parameters, 0 gradients, 15.8 GFLOPs
1/1: 0... Success (inf frames 640x480 at 30.00 FPS)

0: 480x640 (no detections), 357.9ms
0: 480x640 (no detections), 384.9ms
0: 480x640 (no detections), 277.2ms
0: 480x640 (no detections), 297.9ms
0: 480x640 (no detections), 328.1ms
0: 480x640 (no detections), 334.0ms
0: 480x640 (no detections), 468.2ms
0: 480x640 (no detections), 366.5ms
0: 480x640 (no detections), 448.8ms
0: 480x640 (no detections), 409.0ms
0: 480x640 (no detections), 355.8ms
0: 480x640 (no detections), 320.1ms
0: 480x640 (no detections), 335.0ms
0: 480x640 (no detections), 360.2ms
```

図35.コマンドプロンプトの様子

以下は、カメラが起動し、物体検出ができる状態の画面である。



図36.検出画面

## 5.2 検出概要

検証を行うにあたって、検証用のデータ(テストデータ)を用意した。用意したテストデータの詳細は以下の通りである。

表4.テストデータ詳細

テストデータ (640×48ss0)	Plastic bottle : 50 枚 Can : 50 枚 Others : 30 枚
-----------------------	--

130枚のデータを使用して、検証を行っていく。

## 5.3 検出結果

130枚のテストデータを検証すると、以下のような結果が見られた。検出結果の表を以下に示す。

表5.検出結果表

	○	×
ペットボトル (50枚)	48	2
カン(50枚)	44	6
その他(50枚)	25	5
計 (130枚)	117	13

130枚のうち、117枚が正しく認識され、13枚が誤認識したという結果となった。以下のページから、検出詳細を示す。

### 5.3.1 ペットボトル, カンを正しく認識するケース

「ペットボトル」「カン」が正しく認識するケースについて, 下記に示す.

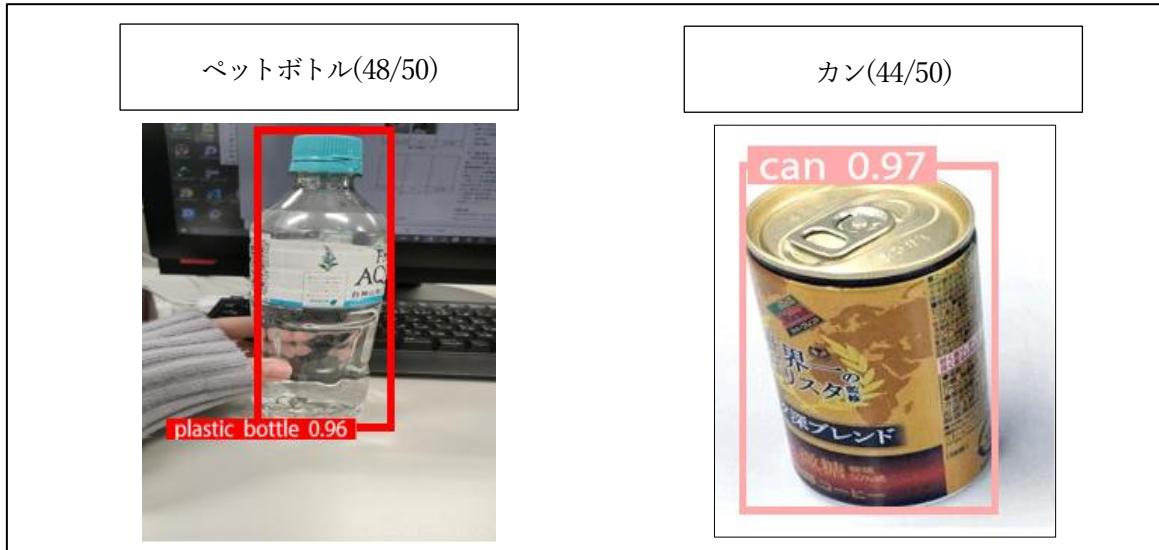


図37. 正しく認識するケース

### 5.3.2 誤って認識してしまうケース

誤認識してしまうケースについて, 下記に示す.

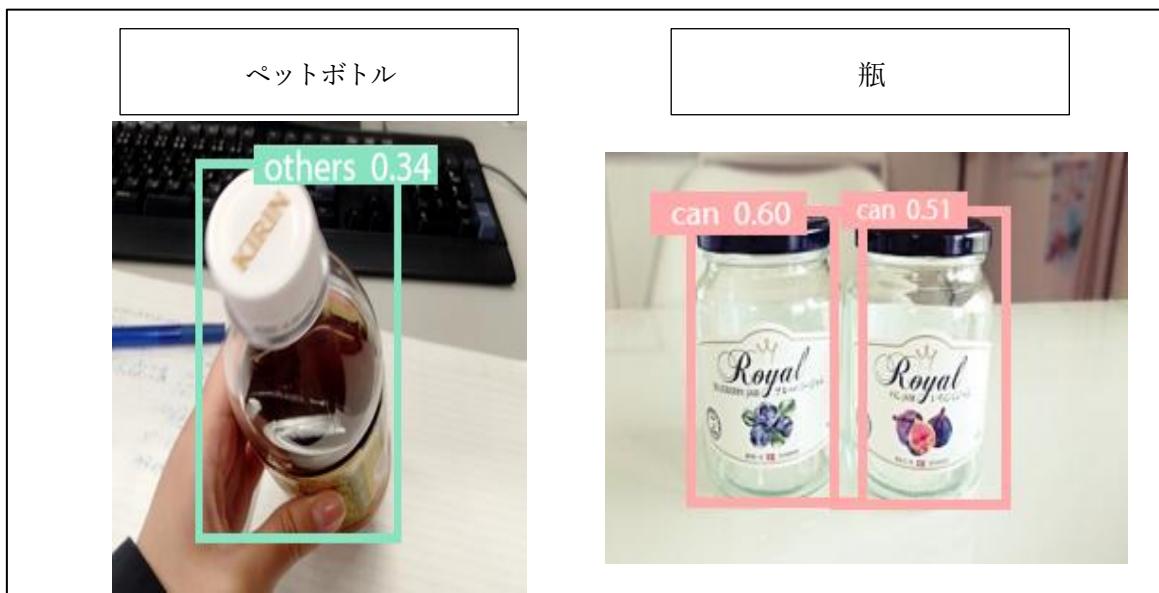


図38. 誤って認識してしまうケース

左手側の画像は、ペットボトルであるにもかかわらず「others」と誤認識している。これではリサイクルボックスの蓋は開かず、ゴミを捨てることが出来ない問題が発生してしまう。

右手側の画像はその他のものである瓶が「can」と誤認識されている。これではリサイクルボックスがカンと検出してしまい蓋が開いてしまうといった問題が発生する。

### 5.3.1 その他のゴミを正しく認識するケース

その他のゴミを正しく認識するケースについて、下記に示す。

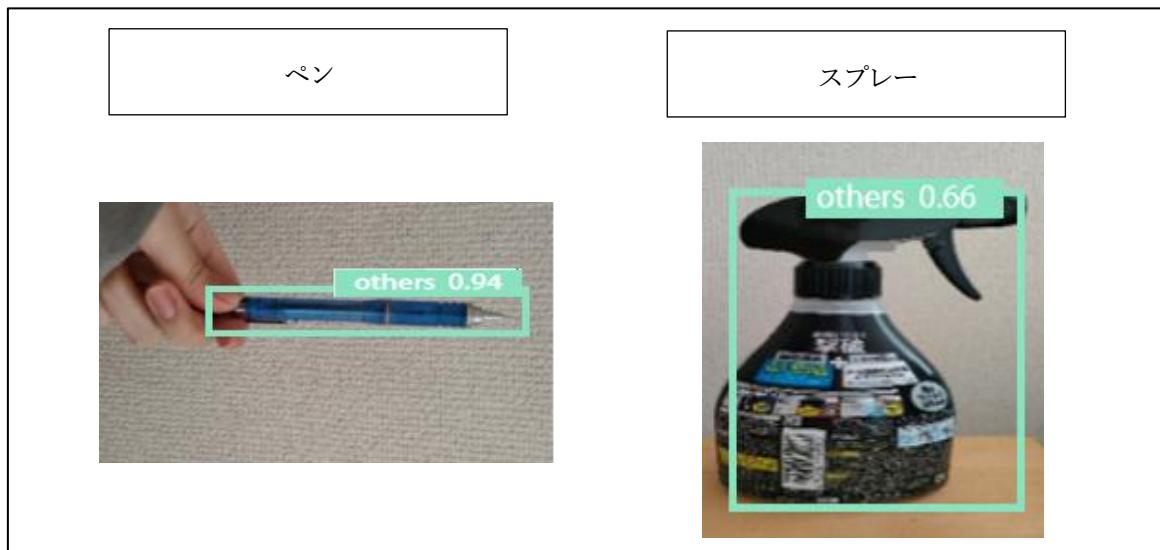


図39. その他のゴミを正しく認識するケース

# 第6章 問題点と精度の改善

## 6.1 誤認識の問題

用意したテストデータで検証を行った結果,5.3.2 のように誤認識が見られた.誤認識してしまう原因は様々あるが,その中でも特に可能性の高い原因をいくつか下記に挙げる.

- ・データ量の不足
- ・与えるデータの条件
- ・与えるデータの画像処理不足
- ・モデル学習の際のエポック数,バッチ数が不適切

## 6.2 解決方法

現時点での問題点として,「与えるデータの画像処理不足」に今回は着目した.画像認識に与える画像は,より質の良い画像を与えなければならない.しかし,本研究に施した画像処理はサイズの統一,変更のみであった.

したがって,画像の明るさ変更,シャープネス処理を行い,データの質を上げる工程を別途行った.

### 6.3 検出率の変化

6.2 の通りデータの画像処理を施したが、認識率に変化があったのは以下の画像のみであった。変化のあった画像を以下に示す。

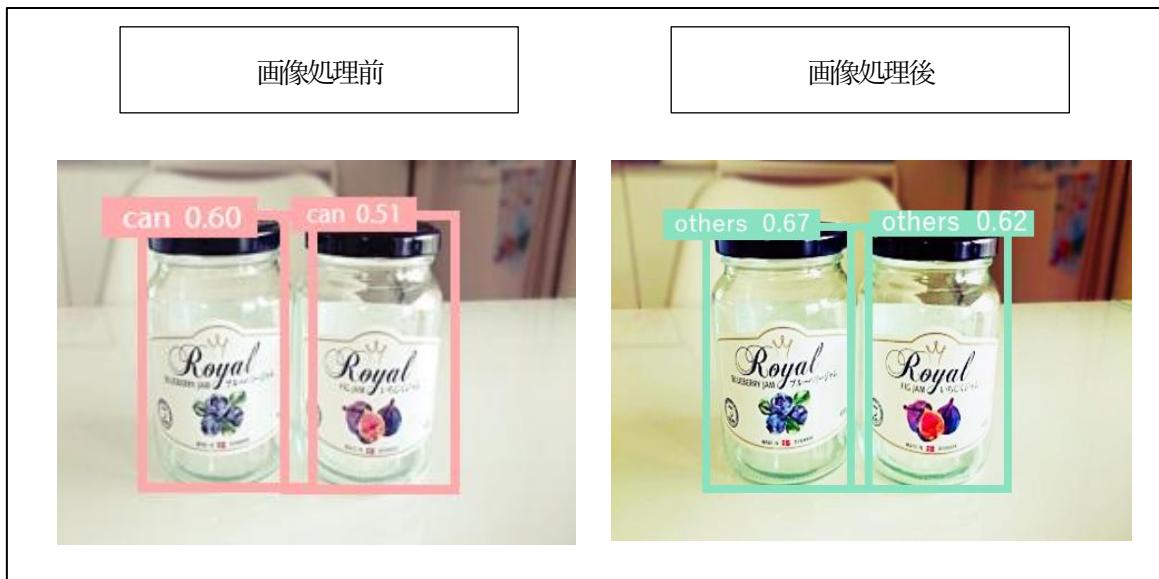


図 40. 認識の変化

従来の学習では、瓶を「カン」と認識していたが、画像処理を施した後では「その他」と正しく認識することができた。しかし、大幅な改善が見込まれなかったため、与えるデータだけでなく、モデル生成の時点のエポック数やバッチ数、学習率などが原因になっているのだと感じた。

## 第7章 多種多様な学習ツール

本研究では、Google Colaboratory を学習ツールとして使用したが、Google Colaboratory の他にも、様々な学習ツールがある。本文章では、「Teachable Machine」と「Azure Custom Vision」について紹介する。

## 7.1 「Teachable Machine」

「Teachable Machine」は Google が提供する、簡単に機械学習のモデルを作成できる学習ツールである。Teachable Machine では「画像プロジェクト」「音声プロジェクト」「ポーズプロジェクト」の3種類のモデルが作成可能で、モデル生成の際の所要時間も短いのが特徴である。



図 41. 「Teachable Machine」イメージ

## 7.2 「Azure Custom Vision」

「Azure Custom Vision」は画像認識のみに特化した学習ツールである。「Teachable Machine」のように音声等のプロジェクトは作成できない。

AI の学習の元となるわずか数十枚の画像教師データのみで画像認識モデルを生成できるため、画像データが思うように収集できない場面で使用すべきである。

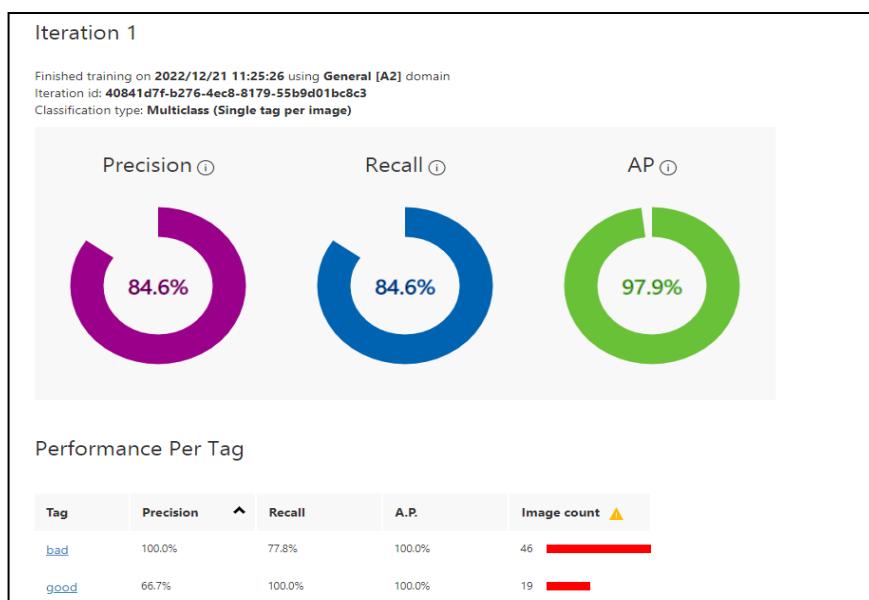


図 42. 「Azure Custom Vision」 使用感

また,図 42 のように,学習率が非常に見やすく表示されるため,初心者にも易しいツールである.

以上のこと参考にして ,自身の目的に合わせた学習ツールの使用が必要である.

## 第8章 おわりに

本研究を通じて OpenCV や深層学習等の知識を身につけ「ペットボトル」と「カン」、「他のゴミ」の三種類の判別モデルを作成し、実装することができた。

まず、深層学習において、環境設定の段階にとても時間をかけてしまった。AI の分野はブラックボックスな面が多く、エラーの原因が推測しにくい場面も多々あった。また、画像収集、ラベリング作業では、たくさんの画像を用意しなければならなかったため、多くの時間がかかった。苦労した点ではあったが、エラーの検索力の向上に大いに役立てることが出来、有意義な時間となった。

今後はこの研究で得た知識や経験を活かし、様々なシステムの開発ができるように精進していきたい。

### <課題点と解決方法について>

今回は「ペットボトル」と「カン」、「他のゴミ」の三種類の判別モデルを作成できたが、課題点が残っている。

1. 精度が一定以上上がらない

解決方法：画像の明るさ変更、シャープネス処理を行い、データの質を上げる工程を行う。

2. ラベリング作業や画像収集時間がかかる

解決方法：ラベリングが不要な「Teachable Machine」や「Azure Custom Vision」など、他の学習ツールを使用する。

3. モデル学習に時間がかかる

解決方法：より性能の良い GPU を使用して学習させる。

今後この研究が引き継がれがあれば、上記に述べたような点に注意して開発を行ってほしい。

## 第9章 参考文献

(1) 公明党:資源の3割 異物混入(2022)

<https://www.komei.or.jp/komeinews/p224430/>

(2) 【物体検出手法の歴史 :YOLO の紹介】

[https://qiita.com/cv\\_carnavi/items/68dcda71e90321574a2b](https://qiita.com/cv_carnavi/items/68dcda71e90321574a2b)

(3) エポック数,バッチ数について:「機械学習／ディープラーニングにおけるバッチサイズ、イテレーション数、エポック数の決め方」

<https://qiita.com/kenta1984/items/bad75a37d552510e4682>

(4) 自作データセット作成方法:「yolo v5 で自作データを学習させ、PC カメラですみっコぐらしたちをリアルタイムに物体検出してみる」

<https://qiita.com/enya314/items/1bd053d6a81a156ff814>

(5) OpenCV:【初心者向け】Python と OpenCV で画像処理を体験してみよう

<https://tech-blog.rakus.co.jp/entry/20201225/open-cv>