レポート_2

泉　秀幸

課題：画像内に写っている「木」の本数を数えろ!!

```
# 必要なライブラリをインポートします
import torch
import cv2
import os
import csv

try:
    # YOLO モデルをパスのモデル 'best.pt' を使用してロードします
    model = torch.hub.load('ultralytics/yolov5', 'custom', path='./best.pt')
    # モデルを評価モードに設定します
    model.eval()

# 'best.pt'ファイルが存在しない場合のエラーハンドリング
except Exception as e:
    print(f"'best.pt'ファイルの読み込み中にエラーが発生しました: {e}")
    # 読み込み中にエラーが発生した場合は終了します
    exit()

# 処理対象となる画像フォルダのパス
folder_path = 'images_resize_0-25'

# フォルダが存在するかどうかを確認します
if not os.path.exists(folder_path):
    print(f"指定されたフォルダ '{folder_path}' が存在しません")
    # フォルダが存在しない場合は終了します
    exit()

# 結果を保存するための CSV ファイルを準備します
output_csv = 'results.csv'
header = ['画像ファイル名', 'クラス名', '検出された本数']

with open(output_csv, mode='w', newline='', encoding='utf-8') as csvfile:
```

```
writer = csv.writer(csvfile)
writer.writerow(header)

# フォルダ内の画像ファイルを処理
# 指定されたフォルダ内のすべてのファイルを取得します
files = os.listdir(folder_path)

# フォルダ内にある画像ファイルを読み込む
for file in files:
    image_path = os.path.join(folder_path, file)
    # OpenCV を使用して画像を読み込みます
    image = cv2.imread(image_path)

    # 画像が読み込めない場合のエラーハンドリング
    if image is None:
        print(f"画像 '{file}' の読み込み中にエラーが発生しました")
        continue # 次のファイルに進む

    try:
        # YOLO モデルによる物体検出
        # YOLO モデルを使用して画像内の物体を検出します
        results = model(image, size=640)

    # 物体検出中にエラーがでた場合のエラーハンドリング
    except Exception as e:
        print(f"画像 '{file}' で物体検出中にエラーが発生しました: {e}")
        continue # 次のファイルに進む

    # 検出結果の処理
    # 検出結果を Pandas DataFrame 形式で取得します
    detections = results.pandas().xyxy[0]  # pandas DataFrame で結果を取得
```

```
        # 各クラス名（検出された物体の種類）の出現回数をカウントします
        class_counts = detections['name'].value_counts()
        if class_counts.empty:
            writer.writerow([
                file,                   # 画像ファイル名
                'なし',                 # クラス名（‘なし’と記述）
                0,                      # 検出された本数
            ])
        else:
            for class_name, count in class_counts.items():
                writer.writerow([
                    file,               # 画像ファイル名
                    class_name,         # クラス名
                    count,              # 検出された本数
                ])

        # 検出結果の画像を表示します
        results.show()

        # 検出結果の画像をファイルとして保存します
        results.save()

print(f"検出結果が '{output_csv}' に保存されました。")
exit()
```

「用語説明」

『カスケード分類器』とは

カスケード分類器とは複数の識別器を組み合わせた分類器です。複数の識別器のすべてが正解画像だと判断した画像を正解画像、識別器のうち一つでも不正解であると判断した画像が不正解画像となるように識別器を調整した分類器をカスケード分類器といいます。

『YOLO』とは

YOLO(ヨロー)とは“You Only Look Once”という英文の頭文字と取って作られた言葉で、「一度見るだけで良い」という意味です。

YOLO は、画像中の物体を高速かつ高精度で検出するためのアルゴリズムであり、畳み込みニューラルネットワーク(CNN)を利用し、画像を一度だけ走査することで、物体のクラスと位置を同時に特定します。これにより、従来の手法に比べて処理速度が大幅に向上しています。YOLO の素晴らしい点は、リアルタイムでの利用が可能であることです。自動運転車、監視カメラ、ロボットなど、さまざまなシーンで物体を迅速に検出するのに役立っています。

「検出手法の説明」

※最初は「カスケード分類器」を使用して検出を試みましたが、最終的には、「YOLO（You Only Look Once)」を採用しました。

※YOLO のトレーニング(学習)には 1 番軽くて速い”yolov5s”を使用したため、前処理として、画像サイズを幅 640px にリサイズしました。

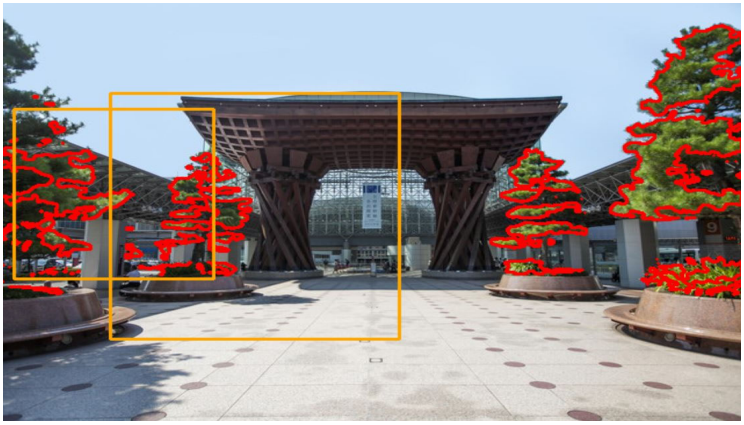
※検出画像は、runs¥detect¥exp{x}　に保存されています。

「YOLO」は、物体検出のためのディープラーニングアルゴリズムで、画像を固定サイズのグリッドに分割し、各セル内で物体の位置、サイズ、クラスを同時に予測します。バウンディングボックスに確信度スコアを付け、信頼性の低い予測をフィルタリングし、非最大抑制を用いて冗長なバウンディングボックスを削除します。高速で効率的なリアルタイム物体検出が可能で、自動運転、監視カメラ、ロボティクス、セキュリティなど多くの分野で利用されています。

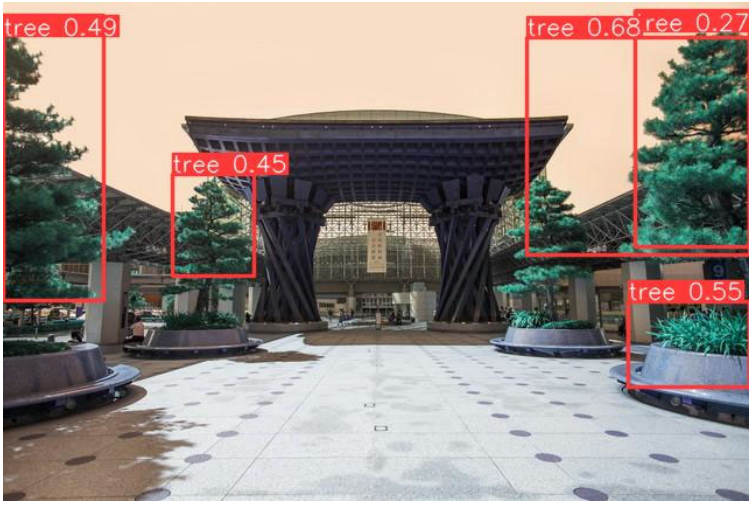
この方法を使用して課題を達成しました。YOLO の調査中に、別のアルゴリズムである「GroundingDINO（グランディングディーノ)」にもたどり着きましたが、DINO は高性能な GPU が必要で、私のパソコンのスペックでは試すことができませんでした。

「精度に関する考察」

※『カスケード分類器による検出』



※『YOLO による検出』



上記の写真は、物体検出において『カスケード分類器』と『YOLO（You Only Look Once)』の 2 つのアルゴリズムの違いを示しています。上は『カスケード分類器』による検出結果を示し、下は『YOLO』による検出結果を示しています。この画像を通じて、2 つのアルゴリズムの性能差や物体検出の質の違いを視覚的に理解することができます。
カスケード分類器による検出では、木の輪郭を強調するための加工が施され、分類器が判断しやすいように工夫しています。しかし、検出精度はご覧の通り

非常に低いです。一方、YOLO による検出では、確信度スコアが 50 前後であるとしても、一応すべての木を検出していることが確認できます。

検出精度は必ずしも高水準ではありません。最初の 50 枚のアノテーション画像を使用しましたが、その中でトレーニング(学習)に利用可能なのは約 30 枚程度でした。さらに多くの木の画像を収集し、アノテーションを行い、トレーニング(学習)を行えば、桜や雪で覆われた木などの物体を、検出する精度を、向上させることができると考えます。

「工夫点、悩んだこと」

1. まず、カスケード分類器を使用して木を検出しようと試みました。そのためにカスケード分類器を自作する方法をネットで調査し、トレーニングを繰り返しましたが、残念ながら木を正確に検出することができませんでした。木の輪郭を強調するためのコードを追加したり、閾値を変更してみたりしましたが、うまくいかないままでした。

2. 調査を繰り返し、物体検出の『YOLO』を知り取り入れることに決めました。まず、YOLO の使用方法について調査しコードを書きました。ただし YOLO はコードを書いただけでは意味をなさずトレーニング(学習)が必要です。トレーニング(学習)の方法は比較的簡単にネットで見つけることができましたが、コードの書き方は参考文献がなかったため、苦労しました。コードを書く過程で最も難しかったのは、トレーニング(学習)で得た「best.pt」というファイルをどのように配置するか迷ったことです。この問題について数日間も悩みました。解決策は Git を学習したときに勉強した Progate(プロゲート)にありました。best.pt の問題を何とか解決し、コードを完成させることができました。また、トレーニング(学習)には高性能な GPU が必要となり GoogleColab の勉強もしました。以下、参考にした Web サイトの URL を載せます。
   参考 URL　https://www.alpha.co.jp/blog/202108_02

3. 課題を達成した要因は、カスケード分類器による検出の試みを諦め、新しいアプローチを模索したことにあります。最初は「画像認識」に関する情報を探しましたが、ある時点で「物体検出」というキーワードを発見し、それを追求する中で「YOLO」にたどり着きました。YOLO について詳細に調査し、その手法を課題の解決に応用しました。熱心に学び、実装に取り組むことで、課題を解決する方法が明確になりました。

4. では何故、カスケード分類器では上手く行かなかったのか、ここからが私の考察です。

YOLO は、畳み込みニューラルネットワーク(CNN)を使用し、画像から特徴を自動的に抽出します。カスケード分類器は、ハンドクラフト(手作り)された特徴量を使用します。YOLO の方が画像から、より豊富で有用な特徴を抽出しやすく、その結果、検出性能が向上していると考えます。また、YOLO は多くのパラメータが調整可能であり、異なるタスクやデータに適応しやすくなります。一方、カスケード分類器は特定のタスクに合わせて手動で調整する必要があり、データに依存しやすいです。トレーニングに使用した画像の品質も悪かったことが上手く行かなかった要因と考えられます。

以上