レポート_1

泉　秀幸

課題：横浜市のルールに則った、
ごみ分別判定 AI を実装せよ !!

```
"""
CLIP（Contrastive Language-Image Pre-training：対照的言語イメージ事前トレーニング）モデルを採用しています。
※"ViT-L/14@336px"モデルを使用しているため、写真は推奨サイズの 336px×336px にリサイズしました。
※検証を簡易化するため、写真には 001 からの番号を付けています。
"""
import os  # ファイルやディレクトリの操作用モジュール
import cv2  # 画像処理用モジュール(OpenCV)
import torch  # PyTorch 深層学習フレームワーク
import clip  # CLIP モデル用ライブラリ
from PIL import Image, UnidentifiedImageError  # 画像操作用ライブラリ(Python Imaging Library)
import csv  # CSV ファイル操作用モジュール

# CLIP モデルのロード
device = "cuda" if torch.cuda.is_available() else "cpu"  # CUDA が利用可能なら GPU、そうでない場合は CPU を使用
model, preprocess = clip.load("ViT-L/14@336px", device=device)  # CLIP モデルと前処理関数を読み込み

# 入力画像の準備
folder_path = "resize_01-45"  # 画像が保存されているフォルダのパス

if not os.path.exists(folder_path):  # フォルダが存在しない場合の確認
    print(f"指定されたフォルダ '{folder_path}' が存在しません")  # エラーメッセージを表示
```

```
    exit()  # プログラムを終了

# フォルダ内のアイテムにアクセスし情報を取得
files = os.listdir(folder_path)  # フォルダ内のファイル一覧を取得

# 結果を保存するためのリストを初期化
all_results = []  # 結果を保存するリスト

# フォルダ内にある画像ファイルを読み込む
for index, file in enumerate(files):  # フォルダ内の各ファイルに対してループを実行
    image_path = os.path.join(folder_path, file)  # ファイルのフルパスを取得
    try:
        image = cv2.imread(image_path)  # 画像を読み込む

        # 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:  # 画像が正常に読み込めなかった場合
            print(f"画像 '{file}' の読み込み中にエラーが発生しました")  # エラーメッセージを表示
            continue  # 次のファイルに進む

        # 画像の読み込みと前処理
        image = preprocess(Image.open(image_path)).unsqueeze(0).to(device)  # 画像を前処理し、バッチ次元を追加してデバイス
```

```
に移動
    except (UnidentifiedImageError, OSError) as e:
        print(f"画像 '{file}' の処理中にエラーが発生しました: {e}")  # エラーメッセージを表示
        continue  # 次のファイルに進む
    except Exception as e:
        print(f"予期しないエラーが発生しました: {e}")  # 予期しないエラーメッセージを表示
        continue  # 次のファイルに進む

    # テキストのラベルを準備
    # 予測したいラベルをリストで指定
    labels = ["can", "bottle", "pet bottle", "cup", "tube", "shoes", "pen", "measure", "mug cup",
              "headphones", "paper bag", "socks", "paper", "scissors", "milk carton", "plush toy",
              "book", "fish", "dry cell battery", "keyholder", "button battery", "nail nippers",
              "spanner", "spring", "pan", "mouse", "notebook computer", "shaver", "calculator",
              "cardboard", "ring", "CD", "plastic bag", "hanger", "wood", "clothespin", "remote",
              "extension cord", "cushioning", "hand towel", "paper", "magazine", "drier", "Kleenex",
              "hexagon wrench", "diamond", "pick", "home electrical appliance", "mask", "toothbrush",
              "spray can", "cap", "plastic fork", "orange", "rubber band", "lip balm", "mirror", "shirt",
              "packing tape", "seal", "cutter knife", "ear pick", "umbrella", "hair comb", "hand fan",
              "shaver", "USB thumb drive", "camera", "nipper", "bis", "driver", "pincers", "lighter",
              "dumbbell", "vacuum cleaner"]  # ラベルのリスト
```

```
# CLIP に渡すテキストプロンプト
text_inputs = torch.cat([clip.tokenize(f"A photo of a {label}") for label in labels]).to(
    device)  # 各ラベルに対応するテキストプロンプトをトークン化して連結

# モデルによる画像とテキストのエンコーディング
try:
    with torch.no_grad():  # 勾配計算を無効化
        image_features = model.encode_image(image)  # 画像をエンコードして特徴ベクトルを取得
        text_features = model.encode_text(text_inputs)  # テキストをエンコードして特徴ベクトルを取得

        # 特徴ベクトルを正規化
        image_features /= image_features.norm(dim=-1, keepdim=True)  # 画像の特徴ベクトルを正規化
        text_features /= text_features.norm(dim=-1, keepdim=True)  # テキストの特徴ベクトルを正規化

        # 画像とテキストの類似度を計算
        similarities: torch.Tensor = (image_features @ text_features.T).squeeze(0)  # 画像とテキストのコサイン類似度を計算
except Exception as e:
    print(f"モデルのエンコーディング中にエラーが発生しました: {e}")  # エラーメッセージを表示
    continue  # 次のファイルに進む
```

```
# 最も類似度が高いラベルを取得
best_match_index = similarities.argmax().item()  # 類似度が最大となるインデックスを取得
predicted_label = labels[best_match_index]  # 予測ラベルを取得

# ごみの分別品目
if predicted_label in ["cup", "shoes", "pen", "headphones", "bag", "remote", "fish", "paper",
                       "plush toy", "keyholder", "mouse", "shaver", "calculator", "hanger",
                       "wood", "clothespin", "extension cord", "hand towel", "drier", "pick",
                       "mask", "rubber band", "lip balm", "packing tape", "seal", "ear pick",
                       "hair comb", "hand fan", "USB thumb drive", "camera", "lighter"]:
    category = "(燃やすごみ)"  # 燃やすごみのカテゴリー分け

elif predicted_label in ["diamond", "orange", "mirror", "mug cup"]:
    category = "(燃えないごみ)"  # 燃えないごみのカテゴリー分け

elif predicted_label in ["spray can"]:
    category = "(スプレー缶)"  # スプレー缶のカテゴリー分け

elif predicted_label in ["dry cell battery", "button battery"]:
    category = "(乾電池)"  # 乾電池のカテゴリー分け
```

```
elif predicted_label in ["toothbrush", "tube", "CD", "plastic bag", "cushioning", "plastic fork"]:
    category = "(プラスチック資源)"  # プラスチック資源のカテゴリー分け

elif predicted_label in ["can", "bottle", "pet bottle"]:
    category = "(缶・びん・ペットボトル)"  # 缶・びん・ペットボトルのカテゴリー分け

elif predicted_label in ["spanner", "bis", "measure", "scissors", "spring", "nail nippers",
                         "ring", "hexagon wrench", "cutter knife", "umbrella", "shaver",
                         "nipper", "driver", "pincers", "dumbbell"]:
    category = "(小さな金属類)"  # 小さな金属類のカテゴリー分け

elif predicted_label in ["paper bag", "milk carton", "book", "cardboard", "paper", "magazine",
                         "Kleenex"]:
    category = "(古紙)"  # 古紙のカテゴリー分け

elif predicted_label in ["socks", "cap", "shirt"]:
    category = "(古布)"  # 古布のカテゴリー分け

elif predicted_label in ["pan", "vacuum cleaner"]:
    category = "(粗大ごみ)"  # 粗大ごみのカテゴリー分け
```

```
        elif predicted_label in ["home electrical appliance"]:
            category = "(市では取り扱えないもの)"  # 市で取り扱えないもののカテゴリー分け

        elif predicted_label in ["notebook computer"]:
            category = "(小型家電製品)"  # 小型家電製品のカテゴリー分け

        else:
            category = "(その他)"  # その他のカテゴリー分け

        # 結果をリストに追加
        all_results.append((index + 1, predicted_label, category))  # 結果をリストに追加

        print(f"画像ファイル名: {index + 1}, 予測ラベル名: {predicted_label}, 分別品目: {category}")  # 結果を表示

# 結果を CSV ファイルに書き込み
try:
    with open("results.csv", mode="w", newline='',encoding="utf-8") as file:  # CSV ファイルを新規作成または上書きモードで開く
        writer = csv.writer(file)  # CSV ライターオブジェクトを作成
        # ヘッダーの書き込み
        writer.writerow(["画像ファイル名", "予測ラベル名", "分別品目"])  # ヘッダーを書き込む
        # 各行の書き込み
```

```
            writer.writerows(all_results)  # 結果リストの各行を書き込む
        print("結果が 'results.csv' に保存されました。")  # 成功メッセージを表示
except IOError as e:
        print(f"CSV ファイルの書き込み中にエラーが発生しました: {e}")  # エラーメッセージを表示
```

【用語解説】

CLIP（Contrastive Language-Image Pre-training）：
“対照的言語イメージ事前トレーニング” とは

CLIP は、OpenAI が開発した機械学習モデルで、テキストと画像の関連性を理解するために設計されたものです。CLIP は、自然言語と画像を結びつける「コントラスト学習*(関連するものを近づけ、無関係なものを離す)*」を用いて訓練されており、幅広いタスクに対応可能なゼロショット性能を持っています。

“CLIP の主な特徴”

・ゼロショット学習(※_1)

① CLIP は、特定のタスク専用に再訓練する必要がありません。

② 新しいタスクにおいても、適切なプロンプト(例：分類タスクで「This is a photo of a …」)を用意するだけで、既存の知識を活用できます。

(※_1)**ゼロショット学習（Zero-Shot Learning, ZSL）**とは、**一度も見たことのないクラスやカテゴリについて推論や予測を行う機械学習の手法**です。
通常、モデルはトレーニングデータに含まれるクラス（カテゴリ）のみを扱えますが、ゼロショット学習では、トレーニングデータには登場しないクラスに対しても予測が可能になります。

“CLIP のモデルの構成”

・CLIP は以下の 2 つのエンコーダを持っています(※_2)。

① 画像エンコーダ(Image Encoder)

・通常、Vision Transformer(ViT) や Residual Network(ResNet)が使用されます。

・画像を埋め込みベクトルに変換します。

② テキストエンコーダ(Text Encoder)

・Transformer を使用

・テキストを埋め込みベクトルに変換します。

これら 2 つのベクトル間の角度のコサイン値を利用して類似度を計算します。

(※_2)**エンコーダとはモデル全体の中の一部。**

“CLIP のメリット”

① ラベルのないデータで学習可能

・画像と説明文のペアがあれば、明示的なラベル(例：「犬」「猫」など)が不要です。

② スケーラビリティ

・インターネット上の大規模なデータセットを用いて学習可能

③ 新しいタスクへの適応力

・従来のモデルのように、特定タスク用に微調整する必要がなく、一般的な知識を利用して幅広いタスクに取り組めます。

“CLIP の応用例”

① 画像分類

・ラベルが少ない環境でも、高精度な分類が可能

② 画像検索

・テキストで検索したい画像を特定するシステム(例：「海辺の夕日」と検索して関連画像を表示)

③ 不適切な画像やテキストを検出

“CLIP の限界”

① トレーニングデータの偏り

・インターネットデータに依存しているため、バイアスが含まれている可能性があります。

② 解像度や細部の理解

・高解像度画像や微細な情報の認識力は限定的です。

③ テキストと画像の直接的な関連性に依存

・テキストと画像のペアが明示的に関連付けられていない場合、正確性が低下することがあります。

“まとめ”

CLIP は、テキストと画像を結びつける汎用性の高いモデルで、ゼロショット学習やマルチモーダルタスク(※_3)において革新的な成果をもたらしました。その一方で、バイアスや高精度な理解には限界があります。

(※_3)**マルチモーダルタスク**とは、異なる種類（モーダル）のデータを組み合わせて処理するタスクのことを指します。たとえば、画像、テキスト、音声、動画、センサーデータなど、複数のデータ形式を統合して分析や処理を行うものです。

【学習用データの収集方法】

CLIP は「ゼロショット学習」により再訓練が不要なため、新しいタスクやクラスに対しても、その都度学習用データを収集し訓練する必要がありません。そのため改めて学習用データの収集は行っていません。

【判定手法の説明】

CLIP は、OpenAI によって開発された AI モデルで、テキスト（言葉）と画像の関連性を高い精度で判定できる仕組みを持っています。このモデルは、テキストと画像をそれぞれ数値化し、それを共通の空間上に配置することで、両者の関係性を分析します。

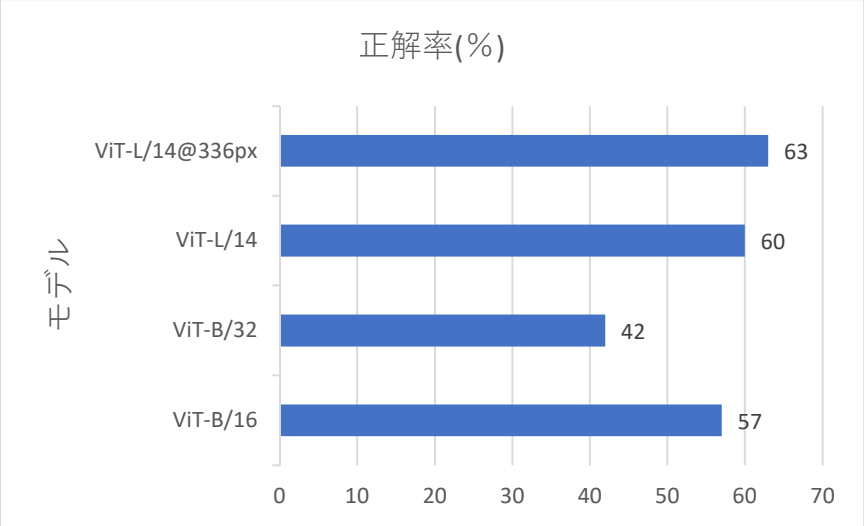
具体的には、似ているテキストと画像は空間上で近くに配置され、関係の薄いものは遠くに配置されるように学習が行われます。この学習には「コントラスト学習(※_4)」という手法が使われ、大量のデータを用いた事前学習によって、新しいデータに対しても高い汎化能力を発揮します。また、テキストと画像の関連性は「コサイン類似度」と呼ばれる指標を使って数値的に計算されます。

簡単に言うと、CLIP は「このテキストに最もふさわしい画像はどれか」を判断するのが得意な AI モデルです。

(※_4)**コントラスト学習**とは、関連するものを近づけ、無関係なものを離す学習法です。

【検証用データに対する判定結果、精度分析、精度に関する考察】
検証用データ(test)を使用してモデルの判定結果を評価したところ、画像とテキストの関連性判定において、モデルに ViT-L/14@336px を選定したとき精度は最高で 63%を記録しました(図 1)。特に缶・びん・ペットボトルの画像に対する判定は高精度でした。一方で、抽象的なテキストや曖昧な画像に対しては誤判定が見られました。

(図 1)「モデルの違いによる、正解率の違い」



誤判定の例として、本来は縦向きが正しい写真ですが、あえて横向きで撮影されたため、誤判定が生じました。
また、被写体が小さすぎて写真の大部分が床となっている場合、「wood」と誤判定されることがありました。

モデルの全体的な精度が 63%であることから、画像とテキストの関連性判定において改善の余地があることがわかります。特に、具体的な対象物(缶・びん・ペットボトル)については高精度である一方、抽象的・曖昧な内容に対する対応力が不足している点が課題として浮かび上がりました。この課題を克服するために画像の事前処理(例えば向きの正規化)が必要と考えられます。

また、ViT-L/14@336px の選定は、高解像度画像での認識精度を期待できますが、画像処理の複雑さが増すため、モデルの設計や学習で想定された標準的な環境から外れた場合、限界が顕著に表れた可能性があると考えられます。

つまり、ViT-L/14@336px のような高解像度向けモデルを活用する場合、その性能を最大限に引き出すためには追加の工夫が必要です。具体的には、画像の特徴をより効果的に抽出するための事前処理や、他のモデルや軽量化された構造を試すことで、処理効率と精度のバランスを考慮することが求められます。

この様に改善策を組み合わせることで、モデルの判定精度を向上させ、特に抽象的・曖昧なケースへの対応力を高めることが可能になると考えられます。

【工夫点、悩んだこと】
正常に動作するだけでなく、読みやすさと保守性を重視したコードを書くことを心掛けました。例外処理や型ヒントを用い、必要に応じて詳細なコメントを追加することで、コードの意図や処理内容を明確にしました。

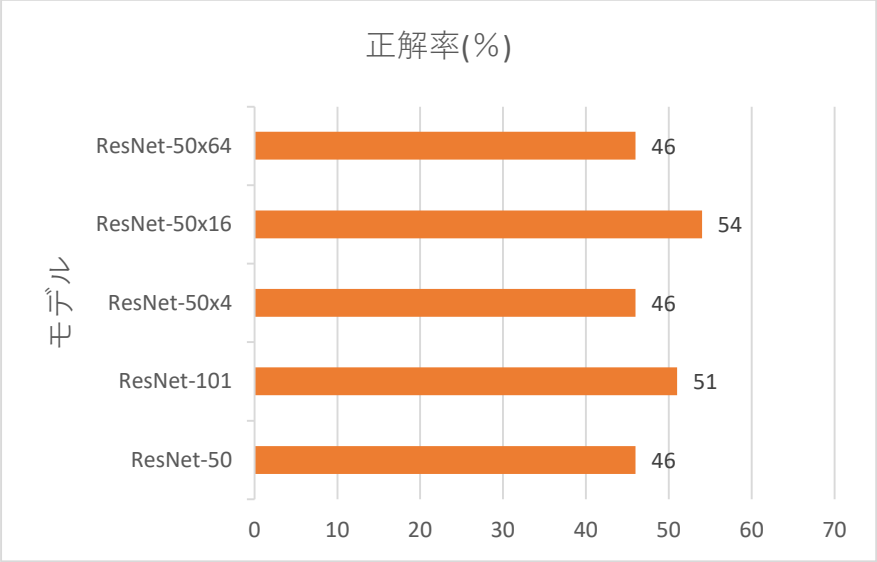
また、精度向上のためにモデルの選定を見直し、データ収集を行いました。さらに、複数のプロンプトを試行することで精度の向上を図っています。

当初は、CNN をアーキテクチャに採用した YOLO を使用して検出を行うことを検討しましたが、学習項目の多さが課題となり、採用を断念しました。
その後の調査で、ゼロショット学習が可能な Grounding DINO と CLIP に注目しました。特に、トレーニングデータに含まれていない新しいカテゴリに対しても分類や検索が可能であるという点から、CLIP を採用しました。

【仮説と実験・検証】

モデルには ViT（Vision Transformer）と ResNet（Residual Network）の 2 種類があります。ViT は ResNet よりも優れた性能を発揮すると考えられています。その理由は、大量のデータを用いてトレーニングすることで、ViT が高度な特徴を効果的に学習し、ResNet を上回る性能を示すためです。

（図 2）「ResNet を選定したときの、正解率の違い」



この仮説を検証するために、ResNet を選定してモデルの違いによる正解率の差を調査しました（図 2）。その結果、ResNet の精度は最高で 54%に達しましたが、ViT-L/14@336px の最高精度である 63%を上回ることはありませんでした（図 1）。また、各モデルの平均精度を比較すると、ViT は 55.5%、ResNet は 48.6%となり、ViT の方が優れた性能を示していることが確認されました。

これらの結果から、大量のデータでトレーニングされると、ViT は高度な特徴を学習し、ResNet を上回る性能を発揮することが明らかになりました。一方で、ResNet は構造的な制約により、大量データでの性能向上が ViT ほど目立たないことが示唆されます。

【AI を活用したごみ分別率が向上する仕組みづくりの提案】

①市民の分別意識の向上と家庭内での誤廃棄防止のため、市民参加型のアプリを開発します。例えば、市民が正しい分別をするとポイントを獲得し、地域限定の通貨として使えるようにします。

②子供たちの分別意識向上のため、ランキング形式を取り入れた分別ゲームを開発します。

③収集ルートを AI で分析して最適化します。

単語帳

bis	ビス
book	本
bottle	ボトル
botton battery	ボタン電池
calculator	電卓
camera	カメラ
can	缶
cap	帽子
cardboard	段ボール
CD	CD
clothespin	洗濯ばさみ
cup	カップ
cushioning	緩衝材
cutter knife	カッターナイフ
diamond	ダイアモンド
drier	ドライヤー
driver	ドライバー
dry cell battery	乾電池
dumbbell	ダンベル
ear pick	耳かき
extension cord	延長コード
fish	魚
hair comb	くし
hand fan	うちわ
hand towel	タオル
hanger	ハンガー
headphones	ヘッドホン
hexagon wrench	六角レンチ
home electrical appliance	家電
keyholder	キーホルダー
Kleenex	テッシュ
lighter	ライター
lip balm	口紅
magazine	雑誌
mask	マスク
measure	メジャー
milk carton	牛乳パック

mirror	鏡
mouse	マウス
mug cup	マグカップ
nail nippers	爪切り
nipper	ニッパー
notebook computer	ノートパソコン
orange	オレンジ
packing tape	ガムテープ
pan	鍋
paper	紙
paper bag	紙袋
pen	ペン
pet bottle	ペットボトル
pick	つまようじ
pincers	ペンチ
plastic bag	ビニール袋
plastic fork	プラスチックフォーク
plush toy	ぬいぐるみ
remote	リモコン類
ring	リング
rubber band	輪ゴム
scissors	はさみ
seal	印鑑
shaver	髭剃り
shirt	シャツ
shoes	靴
socks	ソックス
spanner	スパナ
spray can	スプレー缶
spring	スプリング
toothbrush	歯ブラシ
tube	チューブ
umbrella	傘
USB thumb driver	USBメモリ
vacuum cleaner	掃除機
wood	木

以上