2024.7.1 作成

---本資料目次

注；（1 の仮想環境 env-ga と generate_image.py と 3 について，7/1 現在実行確認中です．
お待たせして申し訳ございません．7/3 朝までに確認予定です）

１．元画像の準備～前処理～データ拡張まで（1 部 2 章）

前処理とデータ拡張用フォルダを作成

次の表 1 の No.1 の出力フォルダを作成します．

表 1　前処理とデータ拡張のために使う入出力フォルダ

区分	No	処理内容	プログラム，ツール	入力フォルダ	出力フォルダ
前処理	1	フォルダ作成	md コマンド	-	sample_img tmp_mask1 prepro_output tmp_1 tmp_2 tmp_mask2 labels bbox sd_input sd_mask sd_output
	2	sample_img に，1 つの jpg ファイルを格納	エクスプローラー，copy コマンドなど	-	sample_img
	3	トリミング用マスク画像を作成	Rembg コマンド	sample_img	tmp_mask1
	4	画像をトリミング	preprocessing.py	sample_img tmp_mask1	prepro_output

データ拡張	5	画像をデータ拡張 （弾性変形）	dataexpansion_1.py	prepro_output	tmp_1
	6	画像をデータ拡張 （回転，縮小拡大，トリミング）	dataexpansion_2.py	tmp_1	tmp_2
	7	画像の背景を除去	Rembg コマンド	tmp_2	sd_input
	8	画像生成用マスク画像を作成	Rembg コマンド	tmp_2	tmp_mask2
	9	画像生成用マスク画像変換 バウンディングボックス付き画像を作成 ラベルファイル作成	annotation.py	tmp_mask2	sd_mask bbox labels
	10	Stable Diffusion で画像を生成	generate_image.py	sd_input sd_mask	sd_output

---

拡張したい画像 1 枚（jpg）を sample_img フォルダに格納

画像ファイル名は任意のものをつけて OK です.

データ拡張パイプライン（データ拡張の流れ）を定義

今回は 1 部 2 章表 4 のデータ拡張の流れにします.

データ拡張のためのプロンプトを生成（generate_prompt.py）

---

generate_prompt.py は，読者がプロンプトを思いつかなかったときに，参考となる画像に似たような画像を生成したいとき，その画像からプロンプトを生成するために使用します.
生成されたプロンプトを使って, 次に説明する設定ファイル settings.ini のプロンプトを記述することで，プロンプトを考えることが難しいときのサポートする役割を果たしています.

設定ファイル settings.ini の用意

Python プログラムの設定情報は, settings.ini に格納しています（図 1）. settings.ini の中に, カギ括弧で囲われた文字列（例：[PREPROCESSING]）で記述されているのがセクションです．各 Python プログラムに対応したセクションに設定値などが記載されています．これらを読み取って Python プログラムの中で利用します.

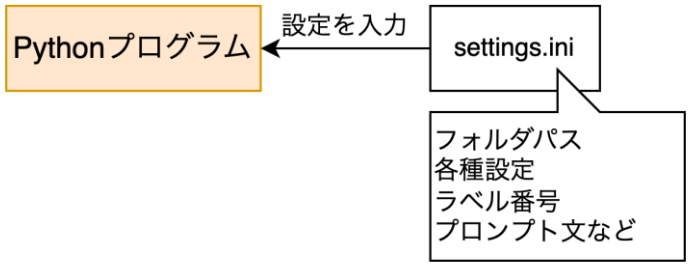

図 1　settings.ini の利用イメージ

　主な記載内容は，入出力フォルダのパス，画像サイズや枚数などの各種設定，ラベル番号（クラス数），プロンプト文やネガティブプロンプト文などです．必要に応じて適宜書き換えて利用します．Python プログラムは，settings.ini に記載されているセクション名と変数名で対象データを文字列を比較して識別します.

前処理からデータ拡張までの処理

仮想環境 env-da の設定，仮想環境 env-ga の設定（Stable Diffusion 関連のインストール）を行い，次の処理を行っていきます（バッチファイルで一括処理）.
（環境設定についてはサポートページの資料参照）

preprocessing.py
↓
(env-da) C:¥Users¥if>rembg p -m　u2net -om sample_img tmp_mask1
↓
dataexpansion_1.py
↓
dataexpansion_2.py

↓
(env-da) C:¥Users¥if>rembg p -m　u2net tmp_2 sd_input
↓
(env-da) C:¥Users¥if>rembg p -m　u2net -om tmp_2 tmp_mask2
↓
annotation.py
↓
generate_image.py

バッチファイル（data_exp.bat）で一括処理する

ここまででご紹介した Python プログラムと Rembg コマンドは，一連の処理をバッチファイルで一括処理もできるでしょう．ここでは，バッチファイル data_exp.bat を作成しました．注意点としては，事前に settings.ini を先的な内容にしておく必要があります．

ここで，バッチファイルを実行する前に，user フォルダ内に sample_img フォルダを作り，元画像を 1 つ格納します．この 1 枚の画像を入力として，バッチファイルで実行される処理によって，1 万枚にデータ拡張されます．バッチファイルには，以下の表 2 の処理内容が含まれます．別のラベルの処理を実行するには入力画像を入れ替えます．user フォルダに別の画像を 1 枚格納して，同様にバッチファイルを実行します．

表 2　data_exp.bat の処理内容

No	処理内容	プログラム，ツール
1	フォルダ作成	md コマンド
2	トリミング用マスク画像を作成	Rembg コマンド
3	画像をデータ拡張（弾性変形）	dataexpansion_1.py
4	画像をデータ拡張（回転，縮小拡大，トリミング）	dataexpansion_2.py
5	画像の背景を除去	Rembg コマンド
6	画像生成用マスク画像を作成	Rembg コマンド
7	画像生成用マスク画像変換 バウンディングボックス付き画像，ラベルファイル作成	annotation.py
8	Stable Diffusion で画像を生成	generate_image.py
9	画像とラベルをデータセットにまとめる	setupdatasets.py

バッチファイルのソースコードは，以下のようになります．コマンドや Python プログラムの実行の他に，実行の開始と終了時間を記録するために「@echo %date% %time%」を加

えていますので，このコマンドが実行された日付と時刻をコマンドライン上に表示します．この一連の処理を実行すると，コンピュータの性能によっては処理に要する時間が異なると思います．

日時を記録することで，1 つのラベル分のデータ拡張にどの程度の時間がかかるか把握できます．バッチファイルは，一度実行すると途中で人の手が入らずに処理を終えられます．処理の所要時間がわかると，夜間や休日のパソコンを使わない時間帯のどこで実行すればよいかの目安になります．パソコンを使用しない時間やパソコン自体の計算資源を有効活用して，大量の画像データやラベルファイルを準備できます．バッチファイルの処理が終わったら，ラベル番号のフォルダを作り，sample_img フォルダと data_exp.bat の処理で作成したフォルダを移動させておくと管理が楽でしょう．

２．データセット作成（1 部 3 章）

設定ファイル settings.ini の用意（１．データ拡張で使用したもの）
↓
データセット処理用のフォルダを作成（実行は setup_datasets.bat で一括処理）
↓
データ拡張したデータを作成したフォルダにコピー(実行は setup_datasets.bat で一括処理)
↓
setupdatasets.py（ファイル名を連番に変える）(実行は setup_datasets.bat で一括処理)
↓
データセットのフォルダを作成
・先に作成した datasets/images フォルダ内に，
　train，val，test のサブフォルダを作成
・先に作成した datasets/label フォルダ内に，
　train，val のサブフォルダを作成
↓
datasets フォルダ下に data.yaml を作成し置く（完成したデータセットのフォルダ構成は 1 部 3 章図 10 参照）
↓
evaluation.py（類似度計算．結果は similarity.txt として出力される）

３．YOLO モデル・トレーニング＆物体検出（1 部 4 章）

3-1 モデルトレーニング

仮想環境 env-ai を作成（サポートページの資料参照）

↓
仮想環境の user フォルダに以下ファイルを置く
　setupdatasets.py
　settings.ini
　setup_datasets.dat
　データセットの画像，ラベル，data.yaml を格納した datasets フォルダ
　evaluation.py
↓
動作確認・GPU デバイスの動作確認のため以下のコマンドをうつ
C:¥Users¥if>nvcc -V
↓
動作確認・YOLOv8 をコマンドライン実行
（Ultralytics サイトで用意されている画像ファイルが自動ダウンロードされて，物体検出モデル（yolov8n.pt）を使って予測モードで実行される）
(env-ai) C:¥Users¥if¥env-ai¥user>yolo predict model=yolov8n.pt source='https://ultralytics.com/images/zidane.jpg' imgsz=320
↓
Ultralytics の公式 Github から yolov8m.pt をダウンロード
↓
yolov8m.pt のチェックポイントに 30 エポック分の追加学習を以下のコマンドで実行
(env-ai) C:¥Users¥if¥env-ai¥user>yolo task=detect mode=train data=datasets¥data.yaml model=yolov8m.pt epochs=30 imgsz=640
↓
画面に「30 epochs completed in 4.182 hours.」が表示され，トレーニング終了
（筆者環境では約 4 時間 12 分かかった）

3-2 物体検出

　前節のモデル・トレーニングまでは外付け GPU 搭載パソコンで行ってきました．
　今回，物体検出は内蔵 GPU 搭載パソコン（CPU：Intel N100）で物体検出実験を行います．この際に仮想環境 env-yl を作成します．外付け GPU がないパソコンでも十分高速に使えることが確認できるでしょう．

仮想環境 env-yl を作成　（サポートページの資料参照）
↓
仮想環境下に user フォルダを作成し，そこに表 3 のファイルやフォルダを配置

表３　作成するフォルダとファイル

種類	名前	説明
フォルダ	runs	仮想環境 env-ai の user フォルダからコピーした学習済みモデルが保存されているフォルダです.
〃	test2	テスト用の画像ファイルを格納したフォルダです.
ファイル	predict.bat	test2 フォルダ内の画像ファイルで物体検出を実行します.

↓

バッチファイル predict.bat をダブルクリックして物体検出処理を実行