"One-Shot" Plant Image Recognition (Image Retrieval) Based on CLIP

1. Function Overview

- **Goal:** This function aims to solve a highly challenging problem: achieving effective plant image recognition under the extreme data scarcity of having **only one reference image per plant class**. To this end, we are not building a traditional classification model, but rather an intelligent **Image Retrieval** system. When a user uploads a plant photo, the system can quickly and accurately find and rank the most visually similar plant species from a pre-built image database.
- **Implementation Approach:**
  - This solution cleverly bypasses traditional classification models that require large amounts of training data. Instead, it leverages OpenAI's powerful, pre-trained **CLIP (Contrastive Language–Image Pre-training)** model. The core capability of the CLIP model is its ability to map images and text into a unified, semantically rich "vector space".
  - **Core Idea**: **Similarity Search in a Vector Space**. We do not "train" a model to "classify". Instead, we use the CLIP model to "translate" all images into high-dimensional vectors (called "embeddings"), and then determine image similarity by calculating the distance between these vectors.
  - **Offline Indexing**: This is a one-time preparation process. We take each reference plant image from our database, generate multiple variations using **data augmentation** techniques, and then use the CLIP model to encode them all into vectors. We then calculate their **average** to form a more robust "class prototype vector". All these prototype vectors together form a "vector index" that can be quickly searched.
  - **Online Querying**: When a user uploads a new image, we encode it in the same way to create a "query vector". Then, by calculating the **cosine similarity** between this query vector and all the prototype vectors in the index, we can find the best matches.

2. Prerequisites & Environment

- **Libraries:**
  - Data Processing: `pandas`, `numpy`
  - Deep Learning & Image Processing: `torch`, `torchvision`, `transformers`, `Pillow (PIL)`
- **Hardware Requirements:**
  - A GPU is highly recommended for both the offline indexing and online querying phases, as it will significantly accelerate the vector encoding process.

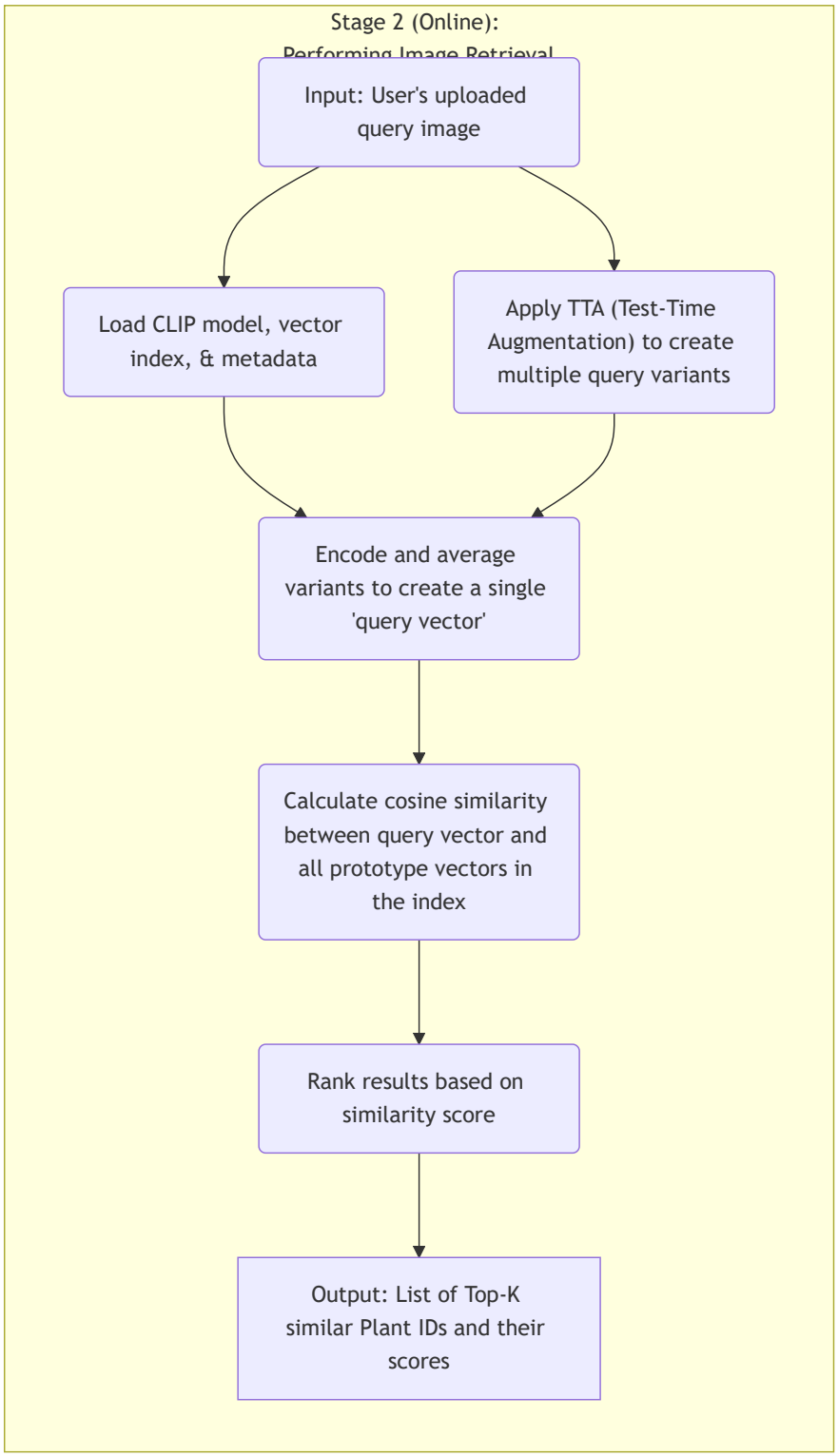
- **Data Dependencies:**
  - **Plant Thumbnails**: A local folder (`01_raw_data/05_thumbnail_image`) containing all the reference thumbnail images for the plants, downloaded from the data source, with one image per plant.

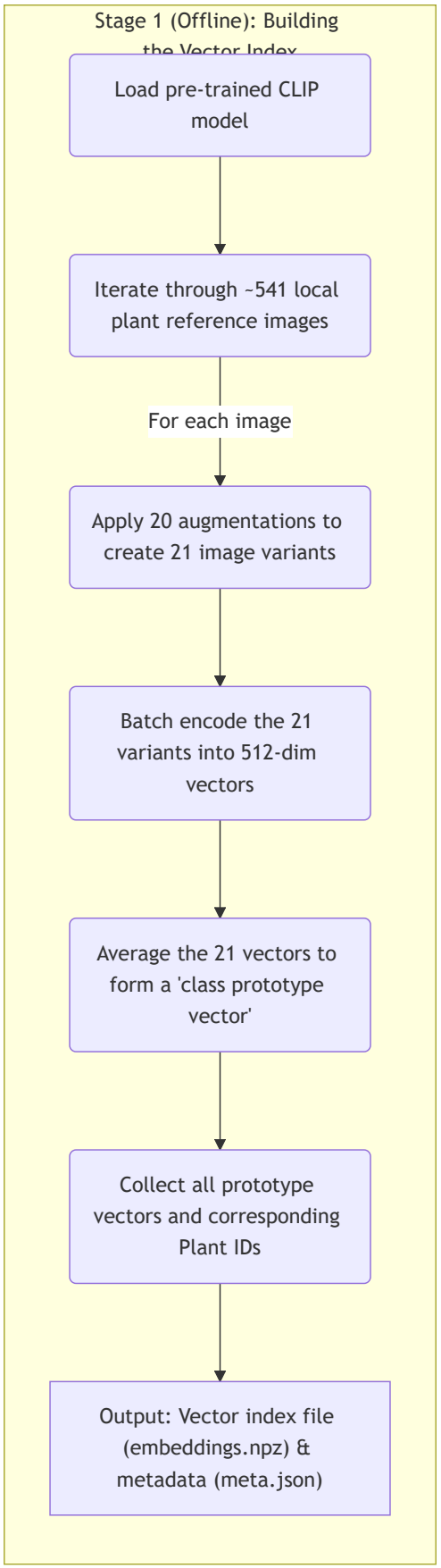
3. Data Source & Input

- **Source 1: Plant Thumbnail Database**
  - **Source**: The local folder `01_raw_data/05_thumbnail_image` . The images in this folder were downloaded using URLs recorded in `Table05_GeneralPlantImageTable.csv` , which was generated in a previous data preparation step. The data in `Table05_GeneralPlantImageTable.csv` is sourced from the Perenual API.
  - **Format**: Each image is in `.jpg` format and is named `plant_species_thumbnail_image_{plant_id}.jpg` , directly linking it to the plant's unique ID.
- **Source 2: User Query Image**
  - **Source**: A plant image to be identified, uploaded by the user through a front-end interface (e.g., in `.jpg` , `.png` , `.webp` format).

4. Workflow

The workflow for this function is divided into two independent parts: a one-time "offline indexing" process and a "online retrieval" process that executes for each query.





- **4.1. Stage 1 (Offline): Building the Vector Index**
  - **Step 1: Load CLIP Model**: Load the `openai/clip-vit-base-patch32` model and its corresponding image processor. This model is pre-trained on a massive dataset and requires

no additional training.

- **Step 2: Iterate Through Reference Images**: Loop through the approximately 541 well-named plant thumbnail images in the local folder.
- **Step 3: Generate Class Prototypes**: This is the core of solving the "one-shot" problem. For each reference image, perform the following operations:
  a. **Data Augmentation**: Apply a set of light image transformations (e.g., random flips, small rotations, color jitter) **20 times**. Together with the original image, this forms a batch of 21 image variants.
  b. **Batch Encoding**: Feed these 21 image variants into the CLIP image encoder to obtain 21 feature vectors, each 512-dimensional.
  c. **Calculate Average**: Compute the **mean** of these 21 vectors to get a single, more representative vector.
  d. **Normalization**: Apply L2 norm normalization to this mean vector to create the final **"prototype vector"** for that plant class.
- **Step 4: Save Index Files**: Stack all the plant prototype vectors into a large NumPy matrix and save the corresponding `plant_id` s in another array. Finally, save both arrays together in a compressed format (`.npz`) as `embeddings_fp16.npz` (using 16-bit floats to save space). Also, save a `meta.json` file containing metadata such as the model name and embedding dimension.

- **4.2. Stage 2 (Online): Performing Image Retrieval**
  - **Step 1: Load Index and Model**: When a query is initiated, the script loads the `embeddings_fp16.npz` index, the `meta.json` file, and the CLIP model from the offline stage.
  - **Step 2: Encode Query Image**: Receive the user's uploaded image. To improve query accuracy, apply **Test-Time Augmentation (TTA)** to it as well, for example, by creating 7 variants. All 8 images (1 original + 7 augmented) are then encoded, and their vectors are averaged to form a single, robust **"query vector"**.
  - **Step 3: Calculate Similarity**: This is the core of the retrieval process and is highly efficient. A single matrix-vector multiplication (`db_emb @ query_emb`) is performed to simultaneously calculate the cosine similarity between the "query vector" and all 541 "prototype vectors" in the index.
  - **Step 4: Rank and Return Results**: The calculated similarity scores are sorted in descending order, and the Top-K results, each containing the `plant_id` and its similarity score, are returned.

5. Core Algorithm & Strategy Interpretation

- **Why CLIP? The Power of Zero-Shot Learning**
  The CLIP model is not a traditional classifier. It was trained to learn the association between "image content" and "descriptive text". This has endowed its image encoder with powerful "zero-shot" generalization capabilities—meaning it can understand the rich semantics of images even without having seen any training samples for that specific class. We leverage this generic, high-quality image "translation" ability to generate meaningful vectors for our retrieval task.
- **How to Solve the "One-Shot" Problem: Prototypes via Augmentation**
  This is the most clever design aspect of this feature. Relying on the embedding of a single raw image is risky, as the result could be skewed by incidental factors like lighting, angle, or blemishes in that specific photo. By creating multiple slightly perturbed **augmented samples** and **averaging** their embeddings, we create a "class prototype vector". This prototype is more stable and better represents the "essence" of the plant class, rather than the accidental features of a single image. It is located at the center of the feature cluster for that class in the vector space, thus greatly enhancing the robustness and accuracy of the retrieval.
  > The way this system works is less like asking a model to answer a multiple-choice question (classification) and more like a police sketch artist showing a drawing to a room full of people (the database) and asking, "Who looks the most like this sketch?" (retrieval).
- **Efficiency of Cosine Similarity**
  Since all vectors are L2-normalized, calculating the cosine similarity between them is equivalent to computing their dot product (matrix multiplication). This is a highly optimized operation on modern CPUs and GPUs. Therefore, even if the database expands to tens or hundreds of thousands of classes, the retrieval process remains extremely efficient.

6. Output

- **Indexing Artifacts:**
  - `embeddings_fp16.npz` : The vector index, containing the prototype vectors for all plant classes and their corresponding `plant_id` s.
  - `meta.json` : The metadata file describing the index.
- **Retrieval Artifacts:**
  - For a given query image, the script outputs a Top-K list of results, ranked by similarity from highest to lowest. Each result is a combination of a `plant_id` and a `score` (the cosine similarity, ranging from -1 to 1, where a value closer to 1 indicates higher similarity).

7. Notes & Future Improvements

- **Notes**:
  - The quality of the retrieval is heavily dependent on the quality and representativeness of the single reference thumbnail image.
  - The choice of augmentation strategy is a balance between creating diversity and maintaining the core features, to avoid "prototype drift".
  - The system returns the "most similar" results, not an "identity confirmation". Even if there is no perfect match in the database, it will still return the closest options.
- **Future Improvements**:
  - **Use Multiple Reference Images**: If more reference images become available for each plant class in the future, they can all be included in the prototype calculation, which would make the prototype more statistically representative and further improve accuracy.
  - **Model Fine-Tuning**: To achieve ultimate precision in the plant domain, the CLIP model itself could be fine-tuned using a contrastive loss function on a curated dataset with multiple examples per class.
  - **Hybrid Search**: The image similarity score could be combined with other plant metadata (such as color, leaf shape, flowering season, etc.) to build a more powerful hybrid search engine.
  - **Dedicated Vector Databases**: For larger-scale applications (e.g., millions of images), the NumPy-based index could be replaced with a dedicated vector database (such as Faiss, Milvus, or Pinecone) to achieve even more efficient, memory-optimized approximate nearest neighbor search.