Project Planning Logic

Date	19 February 2026
Team ID	LTVIP2026TMIDS77295
Project Name	Smart Sorting: Transfer Learning for Identifying Rotten Fruits and Vegetables.

Data Collection

For Smart Sorting, data collection focuses on gathering a **balanced and diverse image dataset** of fresh and rotten fruits and vegetables so that the model can learn clear visual differences.

1. Data Sources

- **Open datasets and online resources**
    - Public image repositories (e.g., Kaggle fruit/vegetable datasets, Google Images, other academic datasets) are used as the primary source for both fresh and rotten samples.
    - Only images with clear visibility of the fruit or vegetable are selected to avoid confusion.
- **Manually collected images (optional enhancement)**
    - Additional photos can be captured using mobile cameras in natural environments such as homes, markets, or college canteens to increase variety (different lighting, backgrounds, and angles).
    - This helps the model generalize better and not overfit to studio-quality images only.

2. Class Definition and Labeling

- The dataset is organized into **two main classes**:
    - Fresh – fruits and vegetables that appear clean, firm, and without visible damage.
    - Rotten – items showing mold, discoloration, shriveling, or other spoilage signs.
- Each image is stored in a folder corresponding to its label:
    - dataset/all/fresh/
    - dataset/all/rotten/

- If images are downloaded in bulk, they are manually checked and moved to the correct folder to ensure label correctness. Mislabelled images are deleted or corrected to avoid confusing the model.

3. Variety and Diversity

To prevent bias, the dataset aims to include:

- **Multiple fruit and vegetable types** (e.g., apples, bananas, oranges, tomatoes, etc.) in both fresh and rotten conditions.
- **Different conditions**: lighting (bright, dim), backgrounds (plain, cluttered), and camera angles (top view, side view, close-up).
- **Different image resolutions**: high and low resolution, later standardized during preprocessing.

4. Train–Test Split

- After collection, images are split into **training** and **testing** subsets.
- A typical split is **80% for training** and **20% for testing**, maintaining the class balance in both sets.
- This can be done by a Python script (split_dataset.py) that:
    - Randomly shuffles the file list for each class.
    - Copies a percentage into dataset/train/fresh and dataset/train/rotten.
    - Copies the remaining into dataset/test/fresh and dataset/test/rotten.

Data Cleaning

Data cleaning ensures that all images used for training are **valid, consistent, and relevant**. This step directly affects the performance and stability of the model.

1. Removing Invalid or Corrupted Images

- Some downloaded files may be incomplete or not readable as images.
- A small script (using PIL or cv2) can be used to:
    - Attempt to open each image.
    - Delete it if an error occurs (corrupt or unsupported format).
- Files with zero size or suspicious extensions (e.g., .txt, .svg) in the dataset folders are also removed.

2. Filtering Irrelevant Images

- During manual review, images that:

    - Do not clearly show a fruit or vegetable.
    - Contain multiple objects where the target item is too small.
    - Are blurred beyond recognition.
      are removed from the dataset.
- This avoids training the model on noisy data that does not represent the problem.

3. Standardizing Image Size and Format

- Models like **MobileNetV2** expect a fixed input size (e.g., 224×224).
- During preprocessing (and optionally as a one-time cleaning step), each image is:
    - Resized to a fixed resolution (e.g., 224×224 pixels).
    - Converted to RGB if needed (some images may be grayscale or RGBA).
    - Normalized by scaling pixel values to $[0, 1]$ or $[-1,1]$ as required by the base model.

4. Handling Class Imbalance

- If the number of **Fresh** images is much larger than **Rotten** (or vice versa), the model may become biased.
- To reduce imbalance:
    - Additional images are collected for the minority class, or
    - **Data augmentation** is applied more aggressively to the smaller class.
- Augmentation can include rotations, flips, zooms, shifts, and brightness changes applied through ImageDataGenerator during training.

5. Data Augmentation (as part of cleaning + enrichment)

Although technically a training-time step, augmentation also helps "clean" the learning process by simulating realistic variations:

- **Random rotations** (e.g., ±20 degrees) to handle different angles.
- **Horizontal flips** to handle orientation changes.
- **Zoom and shift** to handle different object sizes and positions.
- **Brightness and contrast adjustments** to simulate different lighting conditions.

This makes the model more robust to real-world images and reduces overfitting to the original dataset.

6. Final Verification

Before training, the cleaned dataset is verified by:

- Quickly visually scanning a subset from each folder (fresh and rotten) to confirm labels.
- Checking that:
    - Both classes have a reasonable number of samples.
    - No non-image files remain in the directories.
    - The script used for loading data (e.g., ImageDataGenerator.flow_from_directory) runs without errors.