Project Design Phase
Proposed Solution Template

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

Proposed Solution: MobileNetV2-based Transfer Learning

The proposed solution for Smart Sorting is to use a MobileNetV2-based Convolutional Neural Network (CNN) with transfer learning to classify fruit and vegetable images into two classes: Fresh and Rotten.

---

1. Overall Idea of the Solution

Instead of manually designing rules (e.g., "if brown pixels > X then rotten"), the system lets a deep learning model learn patterns directly from images.
A pre-trained MobileNetV2 model, originally trained on the large ImageNet dataset, is reused as a powerful feature extractor, and a small custom classifier is added on top and trained on your labeled fruit dataset.

This approach:

- Reduces training time and data requirements.
- Achieves high accuracy even when the dataset is not extremely large.
- Produces predictions fast enough to be used in real-time web applications.

---

2. Architecture of the Proposed Model

1. **Input Layer**
   - Accepts fruit/vegetable images resized to a fixed size (e.g., 224×224×3).
   - Images are normalized using MobileNetV2's recommended preprocessing (scaling pixel values).
2. **Pre-trained Base: MobileNetV2**
   - Loaded from Keras with include_top=False and weights='imagenet'.
   - All convolutional layers from MobileNetV2 act as a generic feature extractor, detecting edges, textures, colors, shapes, and high-level visual patterns.
   - Initially, the base layers are frozen so that their weights do not change during the first training phase.
3. **Global Average Pooling Layer**
   - Converts the final feature maps from MobileNetV2 into a single feature vector by averaging spatial dimensions.
   - This reduces the number of parameters and helps prevent overfitting.
4. **Dense (Fully Connected) Layer(s)**

- One or more dense layers with ReLU activation are added to learn task-specific combinations of features (e.g., specific color–texture patterns that indicate rot).
- An optional Dropout layer randomly deactivates neurons during training to improve generalization.

5. **Output Layer**
    - Final dense layer with:
        - 1 neuron and sigmoid activation (output between 0 and 1), or
        - 2 neurons and softmax activation for the Fresh and Rotten classes.
    - The output is interpreted as the probability of the image belonging to each class.

---

3. Training Strategy (How the Solution Learns)

1. **Compilation**
    - Loss: Binary cross-entropy for Fresh vs Rotten.
    - Optimizer: Adam with a small learning rate (e.g., 1e-4).
    - Metrics: Accuracy (primary), with precision/recall and F1 used in evaluation.
2. **Data Generators and Augmentation**
    - ImageDataGenerator (or similar) loads images from train and validation directories.
    - Real-time data augmentation is applied to training images:
        - Random rotations, horizontal flips, shifts, zoom, and brightness changes.
    - This makes the model robust to different camera angles, lighting, and backgrounds.
3. **Phase 1 – Train Only the New Head**
    - All MobileNetV2 base layers are frozen; only the added dense layers are trainable.
    - The model is trained for several epochs until training and validation curves stabilize.
    - This allows the classifier head to adapt MobileNet features to the Fresh/Rotten task without disturbing pre-trained weights.
4. **Phase 2 – Fine-Tuning (Optional but Recommended)**
    - A small number of upper layers of MobileNetV2 are unfrozen.
    - The model is re-compiled with a lower learning rate (e.g., 1e-5) to carefully adjust deeper filters.
    - Further training is done, improving validation accuracy and reducing domain mismatch between ImageNet images and fruit images.
5. **Model Saving**
    - The best performing model (based on validation loss/accuracy) is saved to model/healthy_vs_rotten.h5.
    - This file is loaded by the Flask backend for prediction without retraining.

4. How the Proposed Solution Works in Deployment

1. When the Flask app starts, it loads the trained MobileNetV2-based model into memory.
2. When a user uploads an image on the Predict page:
    - The image is saved temporarily and passed to the prediction function.
    - The prediction function:
        - Loads and resizes the image (e.g., 224×224).
        - Applies the same preprocessing used during training.
        - Calls model.predict() to get a probability score.
    - If probability ≥ threshold (e.g., 0.5), the class is Fresh, otherwise Rotten.
    - The result and confidence are returned and shown on the Result page.

This way, the same MobileNetV2 model is used both during training (offline) and at runtime (online), ensuring consistency between performance testing and real-world usage.

---

5. Advantages of the Proposed Solution

- **Accuracy and Robustness**
    - Deep CNN features from MobileNetV2 capture subtle patterns of decay (spots, mold, color changes) better than manual rules or shallow models.
- **Efficiency and Practicality**
    - Lightweight design of MobileNetV2 provides fast inference on CPUs, making the solution usable in small shops and warehouses with basic hardware.
- **Scalability and Extensibility**
    - The same architecture can be extended to:
        - More classes (e.g., multiple fruit types or different defect categories).
        - Larger datasets or different domains by retraining the head or fine-tuning more layers.
- **Easy Integration**
    - Tight integration with TensorFlow/Keras and Flask keeps the codebase simple, maintainable, and suitable for deployment as a web application or API.

Proposed Solution Template:

Project team shall fill the following information in the proposed solution template.

S.No.	**Parameter**	**Description**
1.	Problem Statement (Problem to be solved)	Farmers, warehouse managers, vendors, and supermarket quality inspectors struggle to quickly and accurately identify rotten fruits and vegetables in bulk. Manual visual inspection is slow, tiring, and inconsistent, especially when dealing with thousands of items. Small signs of spoilage are often missed, leading to food wastage, reduced shelf life of stock, financial loss, and customer complaints. There is a clear need for a simple, low-cost AI system that can automatically detect spoilage from images and support better quality control.
2.	Idea / Solution description	The solution is an AI-powered **Smart Sorting web application** that uses **Deep Learning with Transfer Learning (MobileNetV2-based CNN)** to classify fruits and vegetables as **Fresh or Rotten**. Through a web interface (Home, About, Predict, Contact), the user uploads an image on the **Predict** page. The Flask backend saves the image, preprocesses it to 224 × 224, and sends it to the trained model healthy_vs_rotten.h5. The model outputs a prediction label and confidence score, which are displayed instantly along with the image preview on the result page. The system can log predictions for future analysis (waste tracking, accuracy monitoring).
3.	Novelty / Uniqueness	• Uses **Transfer Learning (MobileNetV2)** to achieve high accuracy (~94% on validation data) even with a moderate-sized dataset (around 3200 images). • Focuses specifically on **fresh vs rotten classification**, making it practical for small and medium markets, not only big industries. • Implemented as a **lightweight Flask web app**, so it can run on a normal laptop, local server, or be containerized and moved to cloud/edge devices. • Clean, modern UI (dark/light theme, preview image, confidence score) that non-technical users can operate easily. • Designed with a **modular pipeline** (split_dataset.py, train.py, predict.py, app.py) that can be extended to multi-class fruit types or added to IoT sorting hardware later.
4.	Social Impact / Customer Satisfaction	• Helps **reduce food wastage** by detecting spoilage earlier and more consistently than manual checks.

		• Improves **quality control** in retail stores, mandis, and warehouses, leading to safer and fresher food for consumers. • Supports **better profits** for farmers and vendors by lowering losses from unsold or rejected rotten produce. • Builds **customer trust** because markets can guarantee higher freshness and fewer complaints. • Encourages **digital and AI adoption in agriculture**, aligning with smart farming and AgriTech initiatives.
5.	Business Model (Revenue Model)	• **Free local deployment / open prototype** for farmers, students, and small shops to encourage adoption. • **Subscription model for supermarkets and warehouses**: monthly fee for unlimited predictions, dashboard, and history tracking. • **Enterprise package** for food processing units and large chains: integration with conveyor belts, APIs, and reporting features. • **Government / NGO partnerships** for deploying the system in cooperative markets and farmer producer organizations (FPOs). • Optional **cloud-hosted SaaS version** where customers pay per number of images or per device connected.
6.	Scalability of the Solution	• Can be deployed on scalable cloud infrastructure (AWS / GCP / Azure / IBM Cloud) with the model and Flask app running behind a WSGI server and load balancer. • Stateless prediction API means multiple instances can handle parallel image uploads and predictions. • Dataset and model can be expanded to **multiple fruit/vegetable. categories** and even **disease or ripeness-level detection**. • Future integration with **IoT camera systems and automatic sorting machines** in warehouses and packing units. • Architecture can evolve into **microservices**: separate services for upload, prediction, training, and analytics, enabling independent scaling and maintenance.