**Smart Sorting Transfer Learning for Identifying Rotten Fruits and Vegetables**

**Team ID**: LTVIP2025TMID35946

**Team Members**:

## Kavya Sree Anke (Team Lead)

## S Charan Kumar

## C Mohammad Bilal

## B Pavan Sai

## Phase 1: Brainstorming & Ideation

### 1.1 Objective

To develop an intelligent image-based system that detects spoiled fruits and vegetables using deep learning with transfer learning. This helps reduce food waste and improves quality control in food processing, retail, and home automation.

### 1.2 Problem Statement

A structured empathy-driven table format will be embedded manually post export.

### 1.3 Proposed Solution

We propose using a pretrained CNN (EfficientNetB0) with MixUp augmentation, cosine learning rate decay, and fine-tuning, to detect rotten fruits and vegetables. The model will be deployed through a Flask-based web app with an interactive UI for uploading images and viewing real-time results.

#### 1.3.1 Dataset

Source: Kaggle: Fruits Fresh and Rotten for Classification

~104,000 images  
206 classes (fresh/rotten fruit & vegetable variants)  
Image Size: 100x100 (resized to 224x224 during training)

#### 1.3.2 Workflow

Data Preprocessing  
 - Resize, normalize, and augment  
 - Generate class\_indices.json for model mapping

Model Training  
 - EfficientNetB0 + custom head  
 - MixUp, data augmentation, label smoothing

Evaluation  
 - Accuracy, loss, confusion matrix

Model Saving  
 - .keras model and class\_indices.json

Deployment  
 - Flask API for real-time prediction  
 - HTML/CSS/JS frontend

### 1.4 Target Users

User | Need | Benefit  
--- | --- | ---  
Supermarkets | Auto-check spoilage | Reduce customer complaints, save stock  
Farmers | Quick post-harvest grading | Save time, improve profit margin  
Smart Homes | Kitchen freshness detection | Less waste, better food safety  
Food Tech Startups | ML-based QC tools | SaaS/IoT-based freshness monitors

### Expected Outcome

🍎 A highly accurate deep learning model for spoilage detection  
🧠 ~93–99% validation accuracy using EfficientNetB0 + advanced training  
⚙️ Lightweight Flask API and UI for real-time predictions  
📂 fruit\_classifier\_effnet\_light.keras and class\_indices.json as deployable assets  
🧪 Predicts based on single image upload with class name and confidence

## Phase 2: Requirement Analysis

### 2.1 Prerequisites

|  |  |  |
| --- | --- | --- |
| Component | Version | Purpose |
| Python | 3.9+ | Core scripting |
| TensorFlow | 2.11+ | Model training |
| Flask | 2.0+ | Web API |
| NumPy, Pillow | Latest | Image processing |
| Anaconda | Any | Env & dependency management |

### 2.2 Functional Requirements

• Upload a fruit/vegetable image (JPG/PNG)  
• Resize & normalize the image  
• Load pretrained model + class index map  
• Return JSON result: class\_name + confidence (%)

## Phase 3: Architecture & UI Design

[User Uploads Image] --> [Flask API]  
 |  
 v  
[Preprocessing] --> [EfficientNetB0 Model]  
 |  
 v  
[Class Name + Confidence Score] --> [Display on UI]

### 3.2 Flask API

|  |  |  |
| --- | --- | --- |
| Endpoint | Method | Description |
| / | GET | Home page |
| /predict | POST | Accepts image, returns prediction |
| /health | GET | Checks if model is loaded |

### 3.3 Frontend UI

• HTML/CSS with JavaScript  
• Image upload interface  
• Spinner during prediction  
• Prediction result with class + confidence  
• Responsive for desktop/mobile

## Phase 4: Model Development

Feature | Description  
--- | ---  
Base | EfficientNetB0 (ImageNet pretrained)  
Augmentations | Flip, Zoom, Rotation, Translation  
MixUp | tfp.distributions.Beta for blend  
Optimizer | Adam with CosineDecay LR  
Regularization | Dropout, Label smoothing  
Fine-tuning | Unfreeze last 50 layers  
Exported Model | fruit\_classifier\_effnet\_light.keras

## Phase 5: Deployment & Testing

Backend Server:  
Route | Action  
--- | ---  
/predict | Receives image, returns result  
/health | Confirms if model + mapping loaded  
• CORS enabled  
• Hosted locally or via Render (optional)

### Frontend Preview

Page | Function  
--- | ---  
Home | Upload image  
Results | View predicted class + score  
About | Project overview  
Contact | (Optional) form to reach dev team

### Performance

|  |  |
| --- | --- |
| Metric | Value |
| Validation Accuracy | ~99.25% |
| Inference Time | ~1–1.5 sec |
| Compatible Devices | Mobile, Desktop |
| Model Size | ~20–30 MB |
| RAM Usage | ~500–700 MB |

## Phase 6: Challenges & Fixes

|  |  |
| --- | --- |
| Issue | Fix |
| Mismatched image sizes | Standardized to 224x224 |
| Class mismatch | Synced class\_indices.json |
| Low confidence | Retrained with MixUp and fine-tuning |
| Model loading error | Used .keras format |
| 500 errors | Added exception handling + image validation |

## Final Output

✅ Model + Flask app working with high precision  
✅ Confidence score displayed only if above 55%  
✅ Frontend allows single-click prediction  
✅ Deployment-ready folder with models/, templates/, and static/