PROJECT REPORT

Smart Sorting: Transfer Learning for Identifying Rotten Fruits and Vegetables

Team members:

Pendupurala Pallavi

Dasari Hima Naga Sree

Dathri Sri Lakshmi Mamidi

Dasari Chaitanya Venkat

INTRODUCTION:

❖ Project Overview:

Smart Sorting is an AI-driven system designed to automatically detect and separate rotten fruits and vegetables using transfer learning techniques. By adapting powerful pre-trained deep learning models to a specialized dataset of fresh and rotten produce, the system can identify spoilage with high accuracy.

The project reduces reliance on manual sorting in the food industry, improving efficiency, accuracy, and hygiene. Using cameras and image recognition on a conveyor belt setup, Smart Sorting streamlines the quality control process and minimizes human error, making it ideal for deployment in agriculture, warehouses, and processing plants.

Key benefits:

- Automates sorting and quality checks
- Enhances consistency and speed
- Saves labor costs and reduces waste

❖ Purpose:

Automate the detection and sorting of rotten fruits and vegetables using transfer learning to improve quality control, reduce human error, and increase efficiency in the agricultural and food processing industries.

Specific Goals:

- ➢ Eliminate manual labor in sorting processes
- ➢ Ensure consistent quality of produce
- ➢ Minimize food waste by early and accurate detection of spoilage
- ➢ Leverage transfer learning to reduce training time and improve model performance on limited datasets
- ➢ Support scalable and real-time deployment in industrial environments

IDEATION PHASE:

- ## Problem Statement:

In the agricultural and food processing industries, the manual sorting of fruits and vegetables to identify and remove rotten produce is a time-consuming, labor-intensive, and error-prone process.

Human-based inspection can lead to inconsistent results, delayed processing, and increased operational costs, ultimately affecting the quality and safety of food products.

There is a critical need for an automated, accurate, and scalable solution that can detect spoilage in real time and maintain high standards of quality control.

In the agricultural and food processing industries, the manual sorting of fruits and vegetables to identify and remove rotten produce is a time-consuming, labor-intensive, and error-prone process. Human-

based inspection can lead to inconsistent results, delayed processing, and increased operational costs, ultimately affecting the quality and safety of food products.
There is a critical need for an automated, accurate, and scalable solution that can detect spoilage in real time and maintain high standards of quality control.

- Empathy Map Canvas:

  - Says (What the user says) :
    - Sorting rotten produce takes too much time.
    - We need a faster and more accurate way to check quality.
    - Manual inspection leads to mistakes.
    - Automation could save us money and reduce waste.

  - Thinks (What the user thinks)
    - I'm worried about maintaining consistent quality.
    - Is automation reliable enough to trust?
    - Can this system be easily integrated with our current process?
    - Will this reduce the burden on my team?

  - Sees (What the user sees)
    - Workers struggling with fatigue from repetitive inspection tasks.
    - Inconsistent sorting outcomes from shift to shift.
    - Rising demand for higher-quality produce from buyers and consumers.
    - Competing facilities adopting smarter technologies.

  - Does (What the user does)
    - Manually checks thousands of fruits/vegetables daily.
    - Supervises quality control processes and reports.
    - Deals with complaints about poor produce quality.
    - Trains new workers for sorting and inspection.

  - Pains
    - Human errors in sorting lead to product rejection or waste.
    - High labor costs and slow sorting processes.
    - Inability to scale up operations efficiently.
    - Pressure to maintain high standards without increasing costs.

  - Gains
    - Faster and more reliable sorting process.
    - Reduced operational costs and less food waste.
    - Better compliance with food safety and quality standards.
    - Scalable, consistent solution with minimal retraining required.

- Brainstorming:

  - Technical Implementation Ideas:

- Transfer Learning Models: Use pre-trained models like MobileNetV2, ResNet50, or EfficientNet to fine-tune on a custom dataset of fresh vs. rotten fruits and vegetables.
- Real-Time Sorting System: Integrate the model with a camera system and conveyor belt, using OpenCV for real-time image capture and classification.
- Edge Deployment: Optimize the model for edge devices (e.g., Raspberry Pi, NVIDIA Jetson Nano) for use in remote or low-infrastructure environments.
- Multi-Class Classification: Expand model to detect different types of fruits and multiple stages of spoilage (fresh, slightly spoiled, rotten).
- Image Augmentation: Use heavy augmentation (rotation, lighting, blur) to simulate real- world conditions and improve generalization.

➢ User-Oriented Features:

- Dashboard Interface: A user-friendly web dashboard to display real-time stats, visual results, and sorting accuracy.
- Spoilage Heatmaps: Highlight specific spoiled areas on each fruit using Grad-CAM or saliency maps.
- Manual Override Option: Allow human operators to review and override decisions for borderline cases.

➢ Data and Evaluation Ideas:

- Custom Dataset: Build or augment a dataset using publicly available datasets (e.g., Fruits- 360) + real images from markets/plants.
- Performance Metrics: Track precision, recall, F1-score, and inference time for real-world viability.
- Explainability: Integrate explainable AI (XAI) tools to justify model decisions for transparency.

➢ Sustainability & Impact Ideas:

- Food Waste Reduction Tracking: Estimate how much food waste the system has prevented over time.
- Integration with Supply Chain: Notify suppliers and logistics teams when poor-quality batches are detected early.
- Mobile App for Farmers: A lightweight version of the model for farmers to check produce before shipping.

REQUIREMENT ANALYSIS:

❖ Customer Journey map:

Stage	Actions	Thoughts	Emotions	Opportunities
1. Awareness	Learns about Smart Sorting via ad, demo, or word-of-mouth	“This might solve our quality control issues.”	Curious, hopeful	Provide informative marketing content and success stories

2. Consideration	Visits website, reads documentation, compares with manual/other systems	“Is this more reliable than our current process?”	Cautious, analytical	Offer demos, cost-benefit comparison tools, and technical details
3. Purchase Decision	Talks with team, allocates budget, places order or pilot request	“Can we integrate this into our existing setup easily?”	Confident, slightly uncertain	Offer flexible plans, clear onboarding, and system compatibility guides
4. Onboarding	Installs system, trains team, configures model with custom datasets	“Will our workers adapt to this?”	Anxious, interested	Provide hands-on training, video tutorials, and responsive tech support
5. Usage	Uses the system daily for sorting on conveyor belts	“This is saving time and reducing errors.”	Impressed, satisfied	Collect feedback, monitor accuracy, suggest improvements
6. Support	Contacts support if issues arise or updates are needed	“I hope we can resolve this quickly.”	Frustrated or appreciative	Ensure fast response, proactive maintenanc e alerts, and AI model updates
7. Advocacy	Recommends to other plants or industry peers	“This system transformed our sorting process.”	Proud, loyal	Ask for testimonials, case studies, offer referral discounts

❖ Solution Requirement :

➢ Functional Requirements :

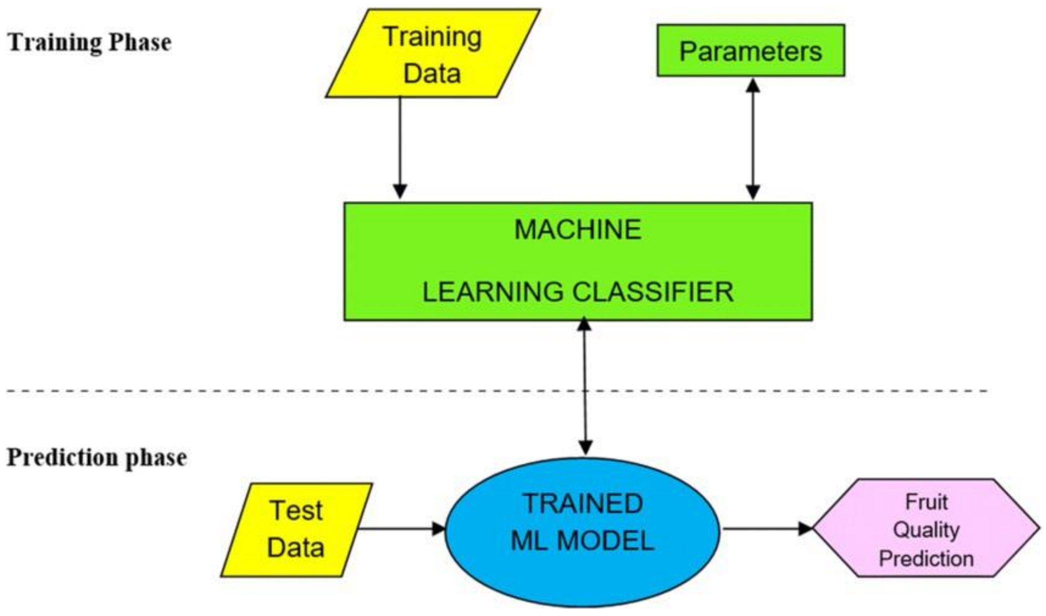
- Image Acquisition System
  - Capture high-resolution images of fruits and vegetables in real time using cameras.
- AI Classification Engine
  - Use a pre-trained deep learning model (e.g., ResNet, MobileNet) adapted via transfer learning to classify produce as fresh or rotten.
- Real-Time Detection and Sorting
  - Process images instantly and send signals to actuators/mechanisms for sorting on a conveyor belt.
- Dashboard/Interface
  - Provide a GUI for users to view classification results, system logs, performance metrics, and manual override options.
- Data Management

    - Store image data, classification results, and error logs for future training, auditing, and analysis.

- Non-Functional Requirements :
  - Performance
    - Classify images in under 1 second with >90% accuracy.
  - Scalability
    - Support high throughput (hundreds/thousands of items per hour) and the ability to add new fruit/vegetable classes.
  - Reliability
    - System should operate continuously with minimal downtime.
  - Usability
    - Interface should be intuitive for non-technical staff with minimal training.
  - Security
    - Data should be securely stored and access controlled.

- Technical Requirements :
  - Languages/Frameworks:
    - Python, TensorFlow/Keras or PyTorch, OpenCV for image processing
  - Hardware:
    - Camera(s), GPU/Edge device (e.g., Jetson Nano), conveyor belt with actuators
  - Software Libraries:
    - NumPy, Pandas, Matplotlib for data handling and visualization
    - Flask or Streamlit for the GUI/Dashboard
    - Scikit-learn for evaluation metrics
  - Dataset:
    - Custom or open-source dataset of fresh and rotten fruits and vegetables, labeled for training
  - Model Deployment:
    - On-device deployment (for offline use) or cloud-based (for centralized systems)

❖ Data Flow Diagram :



- Technology Stack :
  - Artificial Intelligence / Deep Learning
    - Frameworks:
      - TensorFlow / Keras *(for model building and transfer learning)*
      - PyTorch *(alternative deep learning framework)*
    - Pre-trained Models for Transfer Learning:
      - MobileNetV2, ResNet50, EfficientNet *(lightweight and accurate)*
    - Data Processing:
      - OpenCV *(for image capture and preprocessing)*
      - NumPy, Pandas *(for data manipulation)*
      - imgaug or Albumentations *(for image augmentation)*
  - Software & Development
    - Programming Language:
      - Python *(primary language for ML and integration)*
    - Development Tools:
      - Jupyter Notebook *(for experimentation and training)*
      - VS Code or PyCharm *(for coding and integration)*
  - Image Capture & Processing
    - Hardware:
      - High-resolution camera (USB/Webcam/Industrial)
      - Conveyor belt setup with lighting for consistent imaging
    - Libraries:
      - OpenCV *(for image capture and manipulation)*
      - Matplotlib, Seaborn *(for visualizing results)*
  - Model Deployment & Interface
    - Dashboard/UI (optional):
      - Flask or Streamlit *(lightweight web interface for monitoring and testing)*
    - Deployment:
      - Local Machine (for development)
      - Jetson Nano, Raspberry Pi (for edge deployment)
      - AWS EC2 / Google Colab *(for cloud training/deployment if needed)*
  - Data Storage & Management
    - Local:
      - File system for image dataset storage
    - Cloud:
      - Google Drive / AWS S3 for storing datasets or logs
  - Evaluation & Metrics
    - Scikit-learn *(for confusion matrix, accuracy, precision, recall, F1-score)*
    - TensorBoard *(for visualizing training performance)*

PROJECT DESIGN:

- Problem Solution Fit :
  - ✓ The Solution :

    Smart Sorting addresses these challenges by using Transfer Learning and Deep Learning to build an automated image classification system that can:
    - Quickly and accurately identify rotten produce
    - Automate the sorting process via cameras and actuators
    - Operate in real-time with minimal human intervention

- Improve efficiency, consistency, and quality control
- Reduce labor costs and food

waste ✚ Why the Fit Works :

Problem	Smart Sorting Solution
Manual inspection is slow and error-prone	AI-based classification ensures speed and accuracy
High labor cost and fatigue	Automation reduces manual effort and overhead
Inconsistent quality control	Deep learning ensures consistent performance across all shifts
Food waste due to late detection	Real-time detection prevents spoilage from spreading
Need for scalable, smart solutions	Transfer learning enables fast model deployment and adaptation

❖ Proposed Solution :

We propose a smart sorting system that uses deep learning and cameras to automatically detect rotten fruits and vegetables.

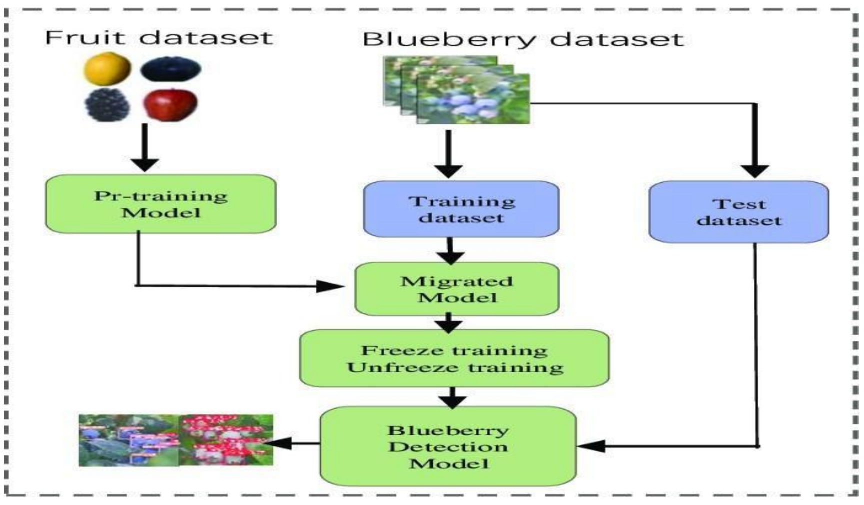
➢ How It Works:

- Camera takes pictures of fruits and vegetables on a conveyor belt.
- A trained AI model (using transfer learning) checks if the produce is fresh or rotten.
- If rotten, the system automatically removes it from the belt.
- An optional dashboard shows results and lets users monitor the system.

➢ Why This Works :

- Faster and more accurate than manual sorting
- Reduces mistakes and food waste
- Saves time and labor costs
- Easy to train for different types of produce

❖ Solution Architecture:



PROJECT PLANNING & SCHEDULING :

❖ Project Planning :

➢ **Phase 1: Research & Requirement Analysis**

- Study existing fruit detection systems
- Define system goals (accuracy, speed, automation)
- Identify required hardware & software
- Duration: 1 week

➢ **Phase 2: Dataset Collection & Preprocessing**

- Collect images of fresh and rotten fruits/vegetables
- Label the dataset
- Preprocess images (resize, normalize, augment)
- Duration: 1–2 weeks

➢ **Phase 3: Model Development**

- Choose pre-trained models (MobileNetV2, ResNet50)
- Apply transfer learning and fine-tune
- Evaluate with validation data (accuracy, precision, recall)
- Duration: 2 weeks

➢ **Phase 4: System Integration**

- Integrate the trained model with image input (OpenCV)
- Connect sorting mechanism (e.g., actuator)
- Build a simple dashboard (Flask/Streamlit)
- Duration: 2 weeks

➢ **Phase 5: Testing & Evaluation**

- Test system with real-time video/images
- Debug and fine-tune for better performance
- Record accuracy, errors, false positives/negatives
- Duration: 1 week

- **Phase 6: Deployment & Documentation**

  ? Deploy system locally or on edge device

  ? Create project report and user manual

  ? Prepare presentation, poster, or demo video

  ? Duration: 1 week

FUNCTIONAL AND PERFORMANCE TESTING :

- **Performance Testing :**

  - **Accuracy Testing :**

    Goal: Check how accurately the model classifies images.

    - Metric Used: Accuracy, Precision, Recall, F1-score
    - Test Method: Use a labeled test dataset (not used in training)
    - Example:
      - Accuracy ≥ 90%
      - Precision ≥ 88% (minimize false positives)
      - Recall ≥ 92% (minimize false negatives)

  - **Speed Testing :**

    Goal: Measure how fast the model classifies images.

    - Metric Used: Time per image (in milliseconds)
    - Test Method: Run the model on live video or image stream
    - Target:
      - ≤ 500 ms per image on GPU
      - ≤ 1 second per image on CPU or Edge Device

  - **Real-Time Testing :**

    Goal: Check system behavior in real-time conveyor belt conditions.

    - Test Items: Live camera feed + model + sorting actuator
    - Observation Points:
      - Detection success rate
      - Delay between detection and sorting
      - Missed or wrongly sorted items

  - **Confusion Matrix Analysis :**

    Goal: Visualize and evaluate true vs. predicted labels.

    - Helps you understand:

- How many rotten items were classified as fresh (false negatives)
- How many fresh items were incorrectly marked as rotten (false positives)

➢ **Robustness Testing :**

Goal: Test how the model performs under various conditions.

- Test Scenarios:
  - Poor lighting
  - Blurry or occluded images
  - Different fruit types or sizes

ADVANTAGES & DISADVANTAGES :

➢ **Advantages :**

- High Accuracy
  - Transfer learning enables precise classification with less data.
- Automation of Sorting
  - Reduces the need for manual labor and speeds up the sorting process.
- Real-Time Detection
  - Quickly identifies and removes rotten produce, improving efficiency.
- Cost-Effective
  - Cuts long-term labor costs and minimizes waste-related losses.
- Scalable Solution
  - Can be adapted to different types of fruits/vegetables and scaled to industrial levels.
- Reduces Food Waste
  - Early detection of rot prevents spoilage from spreading to other produce.
- Consistent Quality Control
  - Unlike humans, the system doesn’t get tired or make subjective errors.

➢ **Disadvantages :**

- Initial Setup Cost
  - Cameras, processors, and sorting hardware may be expensive initially.
- Limited Generalization

    - The model might not perform well on unseen fruit types or extreme conditions unless retrained.

- Lighting & Environment Sensitivity
    - Poor lighting or camera angles can reduce classification accuracy.
- Requires Technical Maintenance
    - Needs regular updates, retraining, and system calibration.
- Dependency on Labeled Data
    - Requires a good amount of correctly labeled fresh and rotten images for training.
- Edge Deployment Limitations
    - On low-power devices like Raspberry Pi, processing speed may be slower without optimization.

CONCLUSION:

The Smart Sorting project presents an innovative and efficient solution to a common problem in the agriculture and food industry—identifying and removing rotten fruits and vegetables. By applying transfer learning and deep learning techniques, the system can accurately classify produce in real time, significantly reducing manual effort, human error, and food waste.

This automated approach enhances quality control, improves processing speed, and offers a scalable solution that can be deployed in both small and large food facilities. With its blend of AI, computer vision, and automation, Smart Sorting contributes to smarter, more sustainable food handling and sets the foundation for future innovations in agricultural technology.

FUTURE SCOPE:

- ✓ **More Defect Types**: Detect bruises, mold, and different spoilage stages.
- ✓ **Broader Produce**: Add support for more fruits & vegetables.
- ✓ **Full Automation**: Integrate with robots and IoT sensors for hands-free sorting.
- ✓ **Edge & Mobile**: Run the AI on phones or small devices in the field.
- ✓ **Cloud Updates**: Push model improvements remotely and collect data for better accuracy.
- ✓ **Self-Learning**: Let the system learn from new images over time to stay up to date.

APPENDIX :

- ❖ **Source Code :**

```

!pip install tensorflow numpy matplotlib scikit-learn opencv-python-headless --quiet

import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import os
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from google.colab import files
```

```

IMG_SIZE = 224
BATCH_SIZE = 32
EPOCHS = 5

print("Please upload a ZIP of your dataset or upload 'dataset/' manually using
the Files tab")

if not os.path.isdir('dataset'):
    uploaded = files.upload()
    for fname in uploaded:
        if fname.lower().endswith('.zip'):
            zip_path = fname
            !unzip -oq "$zip_path"

datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2
)

train_data = datagen.flow_from_directory(
    'dataset',
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    class_mode='binary',
    subset='training',
    shuffle=True
)

val_data = datagen.flow_from_directory(
    'dataset',
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    class_mode='binary',
    subset='validation',
    shuffle=False
)

base = tf.keras.applications.MobileNetV2(
    input_shape=(IMG_SIZE, IMG_SIZE, 3),
    include_top=False,
    weights='imagenet'
)
base.trainable = False

model = tf.keras.Sequential([
    base,
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)

history = model.fit(
    train_data,
    validation_data=val_data,
    epochs=EPOCHS
)

loss, acc = model.evaluate(val_data)
print(f"Validation accuracy: {acc * 100:.2f}%")

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend()
plt.title('Accuracy')

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.title('Loss')

plt.tight_layout()
plt.show()

model.save('smart_sorting_model.h5')
print("Model saved as smart_sorting_model.h5")
```

❖ **GitHub & Project Demo Link :**

- **GITHUB LINK:** https://github.com/dathri1/smart-Sorting-Transfer-Learning-for-Identifying-Rotten-Fruits-and-Vegetables/tree/main/document

- **DATASET LINK :** https://www.kaggle.com/datasets/muhammad0subhan/fruit-and-vegetable-disease-healthy-vs-rotten