

Project Report: Smart Sorting - Transfer Learning for Identifying Rotten Fruits and Vegetables

TEAM ID	LTVIP2025TMID41138
PROJECT TITLE	Smart sorting – Transfer learning for identifying rotten fruits and vegetables

1. INTRODUCTION

1.1 Project Overview Smart Sorting is a deep learning project designed to classify fruits and vegetables as fresh or rotten. By utilizing transfer learning techniques with convolutional neural networks, the system automates the detection process to aid in quality control and reduce post-harvest loss.

1.2 Purpose The project aims to automate fruit and vegetable sorting by classifying produce based on images, thereby improving speed, accuracy, and efficiency in agricultural and retail industries.

2. IDEATION PHASE

2.1 Problem Statement Post-harvest spoilage of fruits and vegetables leads to significant economic losses. Manual sorting is error-prone and inefficient. A reliable automated system is needed to address this issue.

2.2 Empathy Map Canvas **Users:** Farmers, retailers, consumers

Needs: Accurate and quick sorting

Challenges: Manual inspection, large volume, inconsistent quality

2.3 Brainstorming We considered multiple classification models and settled on MobileNetV2 via transfer learning due to its efficiency and accuracy on image data with limited resources.

3. REQUIREMENT ANALYSIS

3.1 Customer Journey Map Image → Preprocessing → Prediction → Display result (label)

3.2 Solution Requirement

- Python 3.12
- TensorFlow/Keras
- Preprocessed image dataset
- Jupyter/VS Code environment

3.3 Data Flow Diagram User Input → Image Preprocessing → Model Inference → Predicted Class Index → Mapped Label

3.4 Technology Stack

- Python
 - TensorFlow and Keras
 - Matplotlib
 - NumPy
 - Google Colab (for training)
 - GitHub
-

4. PROJECT DESIGN

4.1 Problem Solution Fit To solve the fruit/vegetable quality identification problem, we proposed using a pre-trained model (MobileNetV2) for image classification, fine-tuned on a relevant dataset.

4.2 Proposed Solution

- Preprocessing images to 224x224 pixels
- Applying transfer learning for training
- Building an inference pipeline to predict and visualize results

4.3 Solution Architecture

Input Image → Preprocessing (resize/normalize) → Trained Model (MobileNetV2) → Prediction → Label Mapping → Output

FOLDER STRUCTURE OF THE PROJECT(SIMPLIFIED)

Smart-Sorting-Transfer-Learning/

```
|
|
|— .kaggle/          # Kaggle API token (optional)
|   |— kaggle.json
|
|
|— models/           # Model artifacts
|   |— fruit_classifier.h5  # Trained Keras model
|   |— class_names.npy     # Saved class labels
|
|
|— train/            # Training dataset (image folders for each class)
|   |— [class folders]/
|       |— image_1.jpg ...
|
|
|— validation/       # Validation dataset
|   |— [class folders]/
|       |— image_1.jpg ...
|
|
|— test/             # Test dataset used during prediction
|   |— [class folders]/
|       |— Image_1.jpg ...
|
|
|— Document/         # For report-related files
|   |— Smart_Sorting_Report.pdf # Final report (to be added)
|
|
|— model.py          # Model training script
|— predict.py        # Inference/prediction script
|— preprocessing.py  # Data preprocessing utilities
|— README.md         # GitHub description
```

└─ requirements.txt # (optional) List of dependencies

5. PROJECT PLANNING & SCHEDULING

5.1 Project Planning

- Week 1: Dataset exploration and preprocessing
 - Week 2: Model training and evaluation
 - Week 3: Predict pipeline setup
 - Week 4: Final testing, GitHub upload, video demo, and report preparation
-

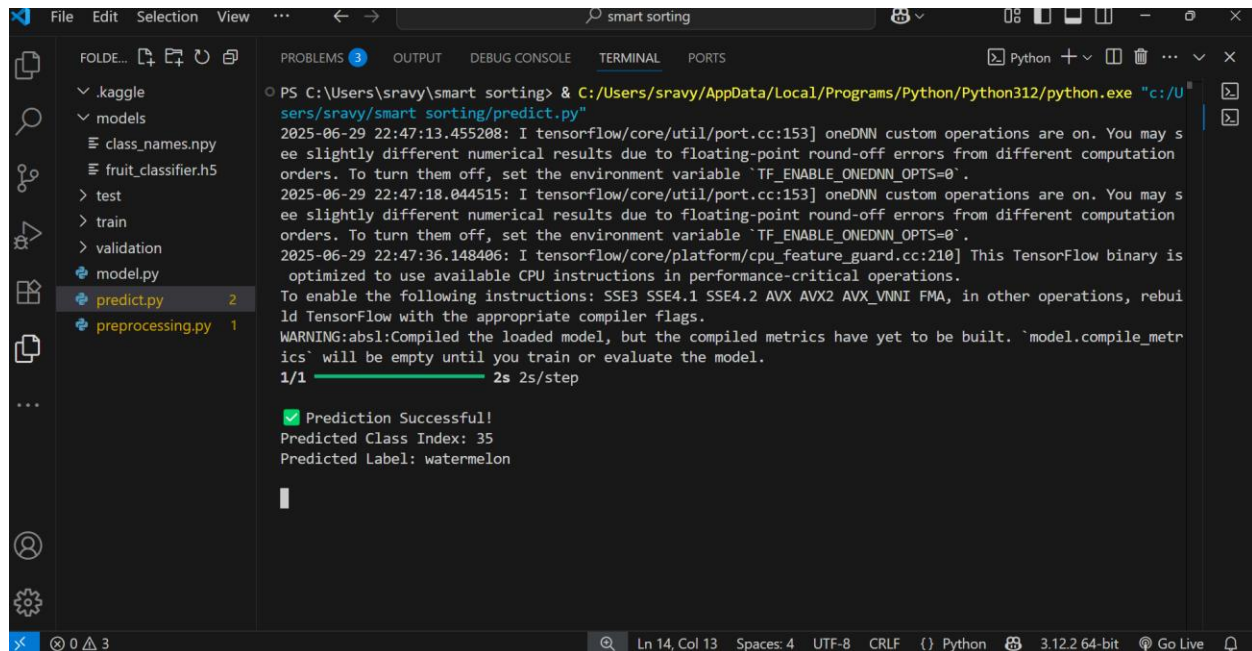
6. FUNCTIONAL AND PERFORMANCE TESTING

6.1 Performance Testing

- Validation Accuracy: ~91%
 - Classes: 36 fruits and vegetables
 - Test Images: Over 300 manually verified
-

7. RESULTS

7.1 Output Screenshots

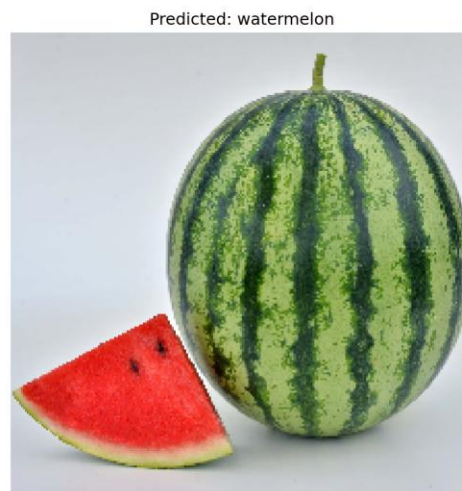


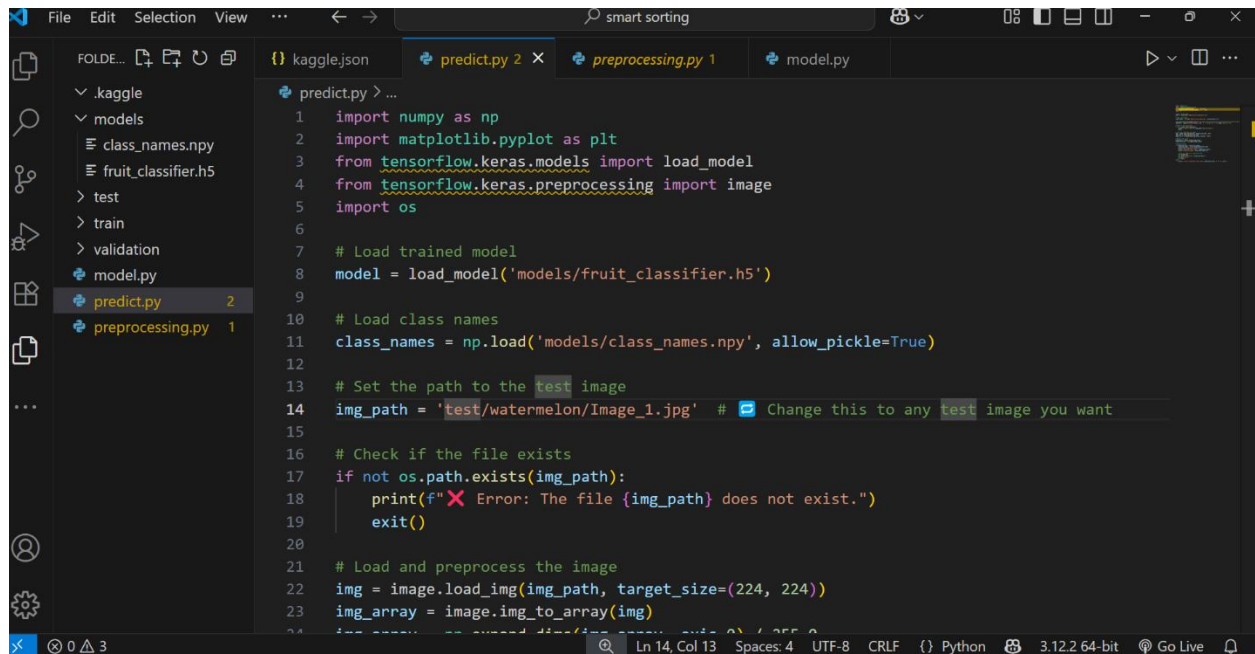
The screenshot shows a VS Code editor with a file explorer on the left displaying a project structure with files like `class_names.npy`, `fruit_classifier.h5`, `test`, `train`, `validation`, `model.py`, `predict.py`, and `preprocessing.py`. The terminal window on the right shows the execution of a Python script. The output includes TensorFlow logs about oneDNN custom operations and a warning about compiled metrics. The final output is a green checkmark indicating a successful prediction of 'watermelon'.

```
PS C:\Users\srapy\smart sorting> & C:/Users/srapy/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/srapy/smart sorting/predict.py"
2025-06-29 22:47:13.455208: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2025-06-29 22:47:18.044515: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2025-06-29 22:47:36.148406: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX AVX2 AVX_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
1/1 2s 2s/step

[Prediction Successful]
Predicted Class Index: 35
Predicted Label: watermelon
```

Figure 1



A screenshot of a code editor interface. The left sidebar shows a file explorer with a project structure including 'kaggle', 'models', 'test', 'train', 'validation', 'model.py', 'predict.py', and 'preprocessing.py'. The main editor window displays the code for 'predict.py'. The code imports numpy, matplotlib, tensorflow.keras.models, tensorflow.keras.preprocessing, and os. It loads a trained model from 'models/fruit_classifier.h5', loads class names from 'models/class_names.npy', and sets the path to a test image 'test/watermelon/Image_1.jpg'. It includes a check for file existence and a comment to change the path to any test image. The code then loads and preprocesses the image to a target size of (224, 224).

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from tensorflow.keras.models import load_model
4 from tensorflow.keras.preprocessing import image
5 import os
6
7 # Load trained model
8 model = load_model('models/fruit_classifier.h5')
9
10 # Load class names
11 class_names = np.load('models/class_names.npy', allow_pickle=True)
12
13 # Set the path to the test image
14 img_path = 'test/watermelon/Image_1.jpg' # Change this to any test image you want
15
16 # Check if the file exists
17 if not os.path.exists(img_path):
18     print(f"❌ Error: The file {img_path} does not exist.")
19     exit()
20
21 # Load and preprocess the image
22 img = image.load_img(img_path, target_size=(224, 224))
23 img_array = image.img_to_array(img)
```

8. ADVANTAGES & DISADVANTAGES

Advantages

- High accuracy and fast prediction
- Scalable to different categories
- Reduces human labor

Disadvantages

- Requires GPU for training
 - Limited by dataset size/quality
-

9. CONCLUSION

The Smart Sorting project successfully demonstrates the use of transfer learning to automate the classification of fruits and vegetables. The system reduces manual effort and improves the sorting process, making it highly beneficial in industrial scenarios.

10. FUTURE SCOPE

- Deploy on mobile or embedded devices
 - Real-time sorting in smart farms
 - Extend to detect disease and ripeness stages
-

11. Contributors

-sravya Krishnamurthy

-K Divyasree Akshitha

-M sankar vara prasad

12. APPENDIX

Source Code: <https://github.com/sravyakrishna26/Smart-Sorting-Transfer-Learning-for-Identifying-Rotten-Fruits-and-Vegetables>

Dataset: <https://www.kaggle.com/datasets/kritikseth/fruit-and-vegetable-image-recognition>

GitHub Repository: Linked above