Computer Vision Project ‘26

Food/Fruit Recognition and Calorie Estimation

No	Name	ID
1	تقى خالد مصطفى عبد الحميد	2022170106
2	جنى عصام عبدالفتاح عرفان	2022170112
3	جنى هانى محمد زكى الشيخ	2022170647
4	روان محمد طه عوض الله	2022170160
5	منار مصطفى فتحى محمد	2022170437
6	رانا ناصر محمد جودة	2022170145

Table of Contents

Introduction

Food/Fruit Recognition and Calorie Estimation is a crucial task that helps navigate the intricacies of diet-culture and health tracking. By training AI models to recognize certain types of fruits as well as dishes, one could create an application that specializes in guiding towards healthier lifestyle choices that align with their goals.

Our project aims to train several deep learning architectures to perform several classification and recognition tasks with the goal of having a cohesive pipeline capable of classifying different types of dishes as well as fruits and calculating the calories for each dish.

First: The Data Preparation Process

The first step to training any model, regardless of the task it is performing, is to first preprocess the data. Our dataset consists of **two directories**, **Fruit** and **Food**.

Each directory has several folders separated by the category of the Food/Fruit as well as subfolders within each category containing Training and Validation data with there being **2538 Food training images** and **1759 Fruit training images** as well as **261 Food validation images** and 150 Fruit validation images.

Each task in our image classification/recognition pipeline may require different preprocessing and data preparation to another task. That is why, for the purposes of this report, the data preparation process will be separated by task.

Part A

For the purposes of this task (classifying whether a certain image is related to the food or fruit category) the preprocessing pipeline was as follows:

- Augmenting the training data
- Preprocessing the validation

Augmenting the training data was necessary to create a more diverse, robust set of images that the model can learn to recognize. This technique was also crucial to preventing overfitting. Random Rotation, Random Flip Horizontal (left to right), Random Resize Crop, and Color Jitters were all performed to prevent the model from associating certain images with certain orientations, sizes, and colors/image saturations.

For the validation data, image resizing and normalization was performed.

For the overall data, class weight balancing was performed to ensure that certain classes of the data didn’t skew the results compared to other classes

Part B

In this part, Data Augmentation was performed. This was crucial in preventing overfitting of the data. Like the previous task, Random Rotation and Random Flip Horizontal were performed. Other augmentation techniques (Random Zoom where the image is either zoomed in or out, Random Translation where the image is slightly off-center, Random Contrast where the image contrast is randomly increased or decreased, and Random Brightness where the image brightness is randomly increased or decreased) were performed differently to the previous task. Of course, these techniques are performed only on the training part of the dataset.

The images are then converted into a friendly format to train tensor models on.

Part C

In this part, per-channel mean and standard deviation values derived from the ImageNet dataset are defined and used to normalize input images. These statistics align the input data distribution with that expected by pretrained convolutional neural networks, ensuring stable activation and effective transfer learning.

Like the previous parts, a unified preprocessing function is implemented to load images from disk. Each image is read from its file path, decoded into an RGB format, resized to a fixed spatial resolution, and converted to floating-point representation with pixel values scaled to the range $[0, 1]$.

An optional augmentation stage is included for training data. When enabled, this stage applies the random transformations that were performed in parts A and B, such as flipping, rotation, brightness adjustment, and contrast variation.

After augmentation (if applied), images are normalized using the predefined ImageNet mean and standard deviation values. The function outputs a tuple consisting of the preprocessed image tensor and its corresponding label.

Part D

Similar to previous parts, standard preprocessing is performed wherein the data is read, decoded, resized, and then cast. Minimal data augmentation is performed on the training data where the images are flipped both horizontally and vertically. Random Brightness and Random Contrast are both applied as well.

Part E

Similar to previous parts, standard preprocessing is performed on the data where the data is read, decoded, resized, and then cast. Data augmentation techniques were also performed on the training data where images are flipped left to right and randomly rotated.

Second: Used Models and Techniques

Part A

Models used:

1. **Custom CNN:** A manually implemented convolutional neural network inspired by the ResNet family. The architecture is built from **custom residual blocks** that include skip (shortcut) connections to enable deeper learning while maintaining training stability.
   The network consists of:
   - An initial convolutional "stem" for low-level feature extraction
   - Three residual stages with increasing channel depth (64 → 128 → 256)
   - Adaptive average pooling
   - A fully connected classification head for **binary classification (Food vs Fruit)**

   Techniques:

   - **Residual Learning (Skip Connections)**
     - Each residual block adds the input directly to the block's output.

    - This mitigates the vanishing gradient problem and allows the network to learn residual mappings rather than full transformations.
    - Enables deeper architectures to train effectively, even without pretrained weights.
- **Class-Weighted Loss Function**
    - Weighted Cross-Entropy Loss is used to handle class imbalance.
    - Ensures minority class samples contribute more strongly to gradient updates.
    - Improves recall and reduces bias toward the dominant class.
- **Optimization and Learning Rate Control**
    - Adam optimizer provides adaptive per-parameter learning rates.
    - ReduceLROnPlateau scheduler lowers the learning rate when validation loss stops improving, enabling finer convergence.

2. **ResNet-18 with Train FC only & Freeze Backbone, Fine Tuning With small LR:** Same as the regular ResNet-18 however, the pretrained backbone serves as a generic feature extractor, while the final fully connected layer is replaced to perform **binary classification (Food vs Fruit)**.

   Techniques:

    - **Transfer Learning**
        - Leveraging ImageNet-pretrained weights allows the model to reuse low-level and mid-level visual features (edges, textures, shapes), significantly reducing training time and data requirements.
    - **Class-Weighted Loss**

    - A weighted cross-entropy loss function compensates for class imbalance by penalizing misclassification of underrepresented classes more heavily.
- **Metric-Aware Learning**
    - Performance is evaluated using:
        - Accuracy
        - Precision
        - Recall
    - This ensures insight into class-specific behavior, not just overall correctness.

3. **Mobile-V2 with Freeze and Fine Tuning:** MobileNet-V2 is a lightweight convolutional neural network optimized for efficiency using **depthwise separable convolutions** and **inverted residual blocks with linear bottlenecks**.
   The pretrained backbone is retained, and the final classification layer is replaced to perform **binary classification (Food vs Fruit)**.

   Techniques:

   - **Transfer Learning**
       - The model is initialized with ImageNet pretrained weights, allowing reuse of strong generic visual features.
       - This reduces training time and improves performance on limited datasets.
   - **Class-Weighted Loss**
       - A weighted cross-entropy loss function compensates for class imbalance by penalizing misclassification of underrepresented classes more heavily.
   - **Metric-Aware Learning**
       - Performance is evaluated using:
           - Accuracy
           - Precision

        - Recall
    - This ensures insight into class-specific behavior, not just overall correctness.

4. **Multi_Fruits:** An attention-enhanced MobileNet-V2 is trained using progressive fine-tuning and ensemble test-time augmentation to achieve robust and discriminative Food vs Fruit classification.

Techniques:

- **Transfer Learning**
    - The MobileNet-V2 backbone is initialized with ImageNet pretrained weights.
    - Pretrained low-level and mid-level features accelerate convergence and improve generalization on limited data.
- **Class-Weighted Loss**
    - A weighted cross-entropy loss function compensates for class imbalance by penalizing misclassification of underrepresented classes more heavily.
- **Optimization and Learning Rate Control**
    - Adam optimizer provides adaptive per-parameter learning rates.
    - ReduceLROnPlateau scheduler lowers the learning rate when validation loss stops improving, enabling finer convergence.
    - Dropout layers at multiple depths reduce overfitting in the enhanced classifier.

Part B

Models used:

1. **Siamese Network:** For the food recognition task, we implemented a Siamese Network architecture using the triplet loss approach. This one-shot/few-shot learning method is particularly suitable for food recognition because it can quickly adapt to new, unseen food categories without requiring extensive retraining.

   Architecture Overview

   - We experimented with four different encoder architectures to find the optimal balance between accuracy and computational efficiency:

     1. Xception-based Encoder (Baseline)

     Base Model: Xception (ImageNet pretrained)

     Feature Extraction: Global Average Pooling

     Fine-tuning Strategy: Last 27 layers trainable

     Embedding Network:

     - Flatten layer
     - Dense layer (512 units, ReLU activation)
     - Batch Normalization
     - Dense layer (256 units, ReLU activation)
     - Unit Normalization (L2 normalization to unit hypersphere)

     Batch Size: Variable (original BATCH_SIZE)

     2. ConvNeXt Tiny Encoder

Base Model: ConvNeXt Tiny (modern ConvNet architecture)

Feature Extraction: Global Average Pooling

Fine-tuning Strategy: Last 20 layers trainable

Embedding Network:

- Flatten layer
- Dense layer (512 units, ReLU activation)
- Batch Normalization
- Dense layer (256 units, ReLU activation)
- Unit Normalization

Batch Size: BATCH_SIZE_convnext

*Special Note: Includes custom LayerScale layer for proper model serialization

3. EfficientNetV2-S Encoder

Base Model: EfficientNetV2-S (efficient scaling)

Feature Extraction: Global Average Pooling

Fine-tuning Strategy: Last 30 layers trainable

Embedding Network:

- **Base model**
  - Unit Normalization (direct normalization without additional dense layers)
  - **Batch Size:** 64
  - **Design Philosophy:** Minimal additional layers for faster inference

Techniques:

Key Technical Components

- Triplet Loss Function
    - The Siamese network uses triplet loss to learn discriminative embeddings:
        - Anchor: Reference image
        - Positive: Same food category as anchor
        - Negative: Different food category
        - Loss: Encourages anchor-positive distance < anchor-negative distance

- Distance Layer
    - Custom Keras layer computing:
        - AP Distance: Euclidean distance between anchor and positive embeddings
        - AN Distance: Euclidean distance between anchor and negative embeddings

- Unit Normalization
    - All embeddings are normalized to lie on a unit hypersphere, which:
        - Stabilizes training
        - Makes distances more meaningful
        - Improves few-shot learning performance

Part C

Models used:

1. **ResNet-50:** ResNet-50 convolutional neural network pretrained on ImageNet is used as the feature extractor. The original classification head is removed (include_top=False) and replaced with a custom classifier consisting of global average pooling, fully connected layers, and dropout. The final layer uses softmax activation to perform multi-class classification.

Techniques:

- **Transfer Learning**
    - ImageNet pretrained weights allow the model to reuse rich, general-purpose visual features, improving performance and reducing training time.
- **End-to-End Finetuning**
    - The ResNet-50 backbone is trained with training=True, allowing both pretrained layers and the new classifier head to adapt to the target dataset.
- **Global Average Pooling**
    - Reduces feature maps to a compact representation, lowering parameter count and minimizing overfitting.

2. **EfficientNet-B4:** Pretrained on ImageNet, is used as the backbone network for feature extraction.
    - The original classification head is removed (include_top=False) and replaced with a custom fully connected classifier. Global average pooling converts spatial feature maps into a compact feature vector before classification.
    - The final layer uses softmax activation to predict class probabilities.

Techniques:

- **Transfer Learning**
    - ImageNet pretrained weights enable efficient learning and improved generalization on the target dataset

- **End-to-End Finetuning**

  The EfficientNet backbone is trained with training=True, allowing both pretrained layers and the custom classifier to adapt to the task.

- **Compound Scaling Architecture**
  - EfficientNet-B4 balances network depth, width, and resolution to achieve high accuracy with fewer parameters compared to traditional CNNs.

3. **MobileNetV2:** pretrained on **ImageNet**, is used as the backbone for feature extraction. The top (classifier) is removed (include_top=False) and replaced with a **custom dense head**.

   Techniques:

   - **Transfer Learning**
     - Leveraging ImageNet-pretrained weights allows the model to speed up training and improve generalization.
   - **Regularization**
     - Dropout layer prevents overfitting on small datasets.
   - **Optimization**
     - Uses **Adam optimizer** with sparse categorical cross-entropy loss for multi-class classification.

4. **ConvNEXT Tiny:** consists of a pretrained ConvNeXtTiny backbone with a custom classification head. After global average pooling and layer normalization, it has a 512-unit dense layer with GELU activation, followed by dropout, and a final softmax output layer for multi-class classification.

   Techniques:

   - **Layer Normalization**

    - Applied before MLP-style layers in the blocks to stabilize training.
- **GELU Activation**

  A smooth activation function that improves gradient flow and learning dynamics compared to ReLU.
- **Downsampling with Strided Convolutions**
    - Replaces traditional pooling for more learnable spatial reduction.

5. **Custom_CNN:** is a sequential convolutional network designed for binary classification (Food vs Fruit). It consists of four convolutional blocks with increasing channel depth (32 → 64 → 128 → 256), each followed by batch normalization, ReLU activation, and max pooling, acting as feature extractors.

   Techniques:

- **Residual Learning (Skip Connections)**
    - **Flatten** transforms feature maps into a vector for dense layers.
    - **Dense** layers (1024 → 512) process high-level features.
    - Dropout layers (0.5, 0.3) prevent overfitting.
    - Final dense layer with **softmax** outputs probabilities for binary/multi-class classification
- **Optimization and Early Stopping**
    - Adam optimizer provides adaptive per-parameter learning rates.
    - ReduceLROnPlateau scheduler lowers the learning rate when validation loss stops improving, enabling finer convergence.

      EarlyStopping monitors validation accuracy and stops training if no improvement occurs, restoring the best weights.

- **Regularization Techniques**
    - Dropout in dense layers to reduce overfitting.
    - Implicit regularization from batch normalization.

Part D

Models used:

1. **U-Net:** classic **U-Net design**, which has a symmetric encoder-decoder structure:
    - **Encoder (Downsampling):** Three convolutional blocks with increasing filter sizes (32 → 64 → 128). Each block has **two Conv2D layers** with **Batch Normalization** and **ReLU** activations, followed by **MaxPooling** to reduce spatial dimensions.
    - **Bottleneck:** The deepest block (128 filters) captures high-level semantic features.
    - **Decoder (Upsampling):** Two upsampling stages that use **bilinear upsampling** and **skip connections** from the corresponding encoder blocks to recover spatial information. Each upsampling stage is followed by the same double convolution block.

Techniques:

- **Double Convolution Blocks**
    - Two sequential conv layers with BN and ReLU to extract features while stabilizing training.
- **Skip Connections**
    - Connect encoder and decoder layers to preserve spatial information and improve gradient flow.
- **Adam Optimizer**

Adaptive learning rates per parameter for efficient convergence.

2. **DeepLabV3:** The architecture uses an encoder-decoder design enhanced with **Atrous Spatial Pyramid Pooling (ASPP)**:
    - **Encoder:** A series of convolutional layers acting like a lightweight ResNet backbone to extract hierarchical features at multiple resolutions. Low-level features are preserved for the decoder.
    - **ASPP Module:** Uses multiple parallel convolutions with different dilation rates to capture multi-scale contextual information, plus global average pooling to incorporate image-level context.
    - **Decoder:** Upsamples the ASPP output, combines it with low-level features via concatenation, and refines the result with convolutional layers. Finally, the feature map is upsampled to the original image size.

    Techniques:

    - **Skip Connections (Low-Level Features)**
        - Preserve spatial detail and improve boundary accuracy by merging encoder and decoder features.
    - **Batch Normalization + ReLU**
        - Applied after each convolution to stabilize training and speed up convergence.
    - **Upsampling with Bilinear Interpolations**
        - Gradually restores spatial resolution in the decoder.
3. **SegNet:** follows a classic encoder-decoder architecture:
    - **Encoder:** Four convolutional blocks, each containing two convolution layers with batch normalization and ReLU activation, followed by max pooling for downsampling. The

number of channels increases progressively (64 → 128 → 256 → 512).

- **Decoder:** Four upsampling blocks that mirror the encoder. Each block upsamples using bilinear interpolation, followed by two convolution layers with batch normalization and ReLU.

Techniques:

- **Convolution + Batch Normalization + ReLU:**
  - Stabilizes training and allows deeper network learning.
- **Encoder-Decoder Architecture**
  - Reduces spatial resolution for feature extraction and restores resolution for segmentation.
- **No Skip Connections**
  - Makes SegNet lighter and simpler, but slightly less precise on fine object boundaries.

Part E

Models used:

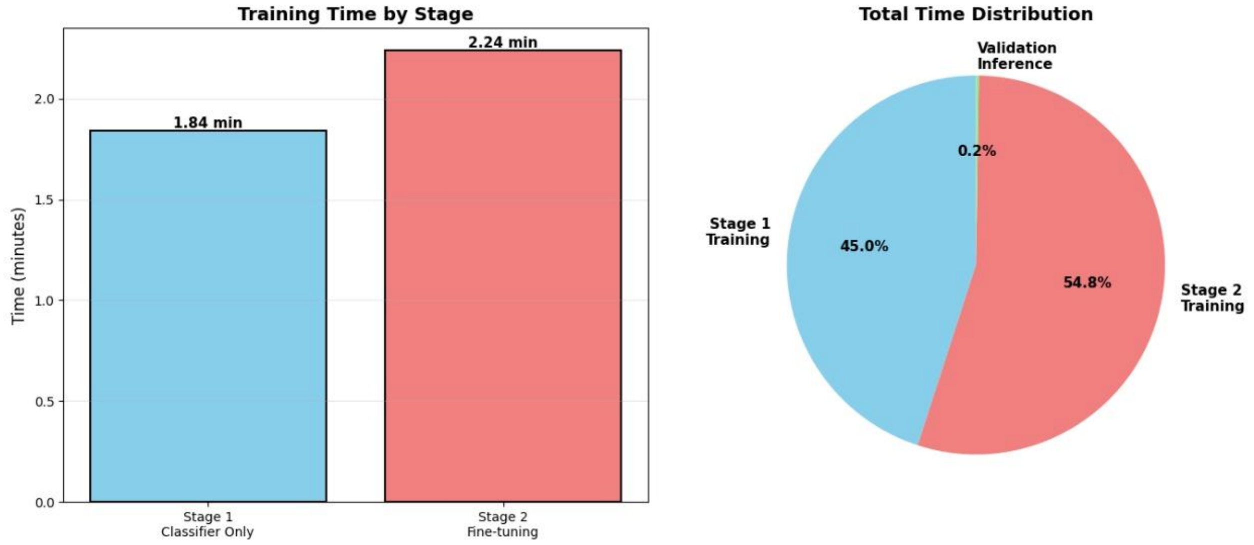
1. **ResNet50V2_UNet:** The architecture uses a **ResNet50V2 backbone** pretrained on ImageNet. Its intermediate feature maps are used as **skip connections** to preserve spatial details. The decoder progressively upsamples the features with **Conv2DTranspose layers**, concatenates them with corresponding encoder features, and applies convolution + batch normalization + ReLU sequences. Dropout layers are added for regularization, and the final output layer uses a **softmax activation** for multi-class segmentation.

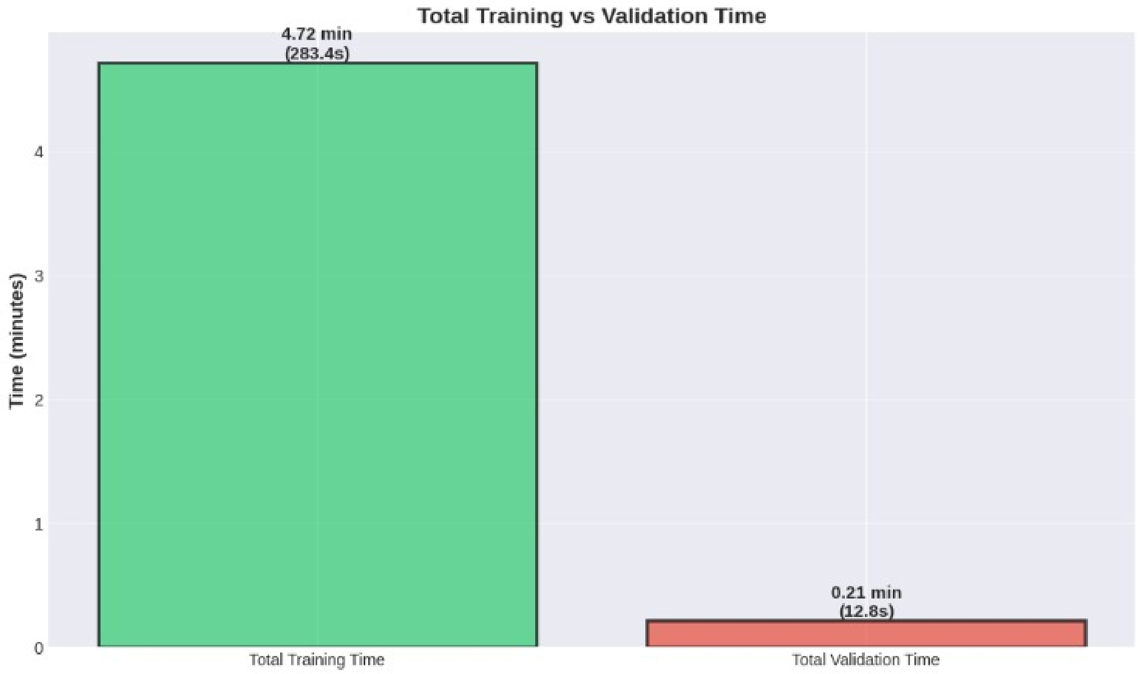
Techniques:

- **Convolution + Batch Normalization + ReLU:**
  - Stabilizes training and allows deeper network learning.
- **Skip Connections (U-Net Style)**
  - Connect encoder and decoder layers to preserve spatial information and improve gradient flow.
- **Attention-like Decoder Design**
  - Although not explicit attention layers, the concatenation and progressive feature refinement act like a spatial attention mechanism, focusing on important features from the encoder.

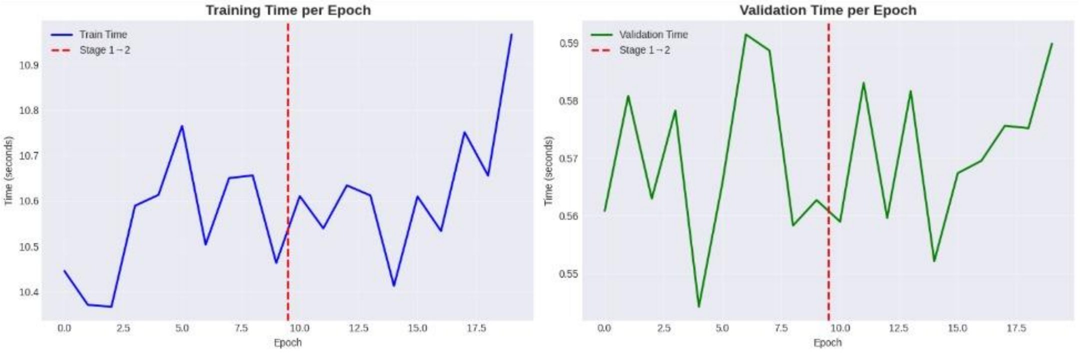
Third: Training and Testing Time per Model

Part A

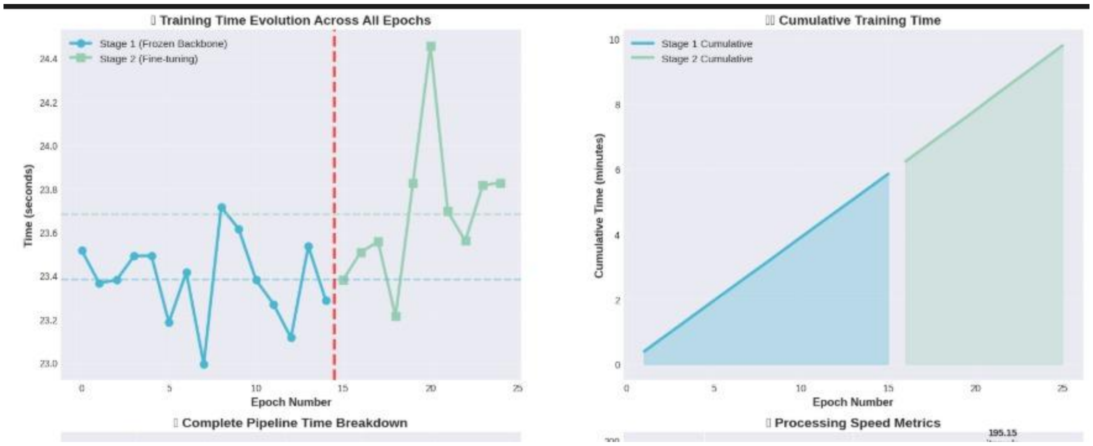


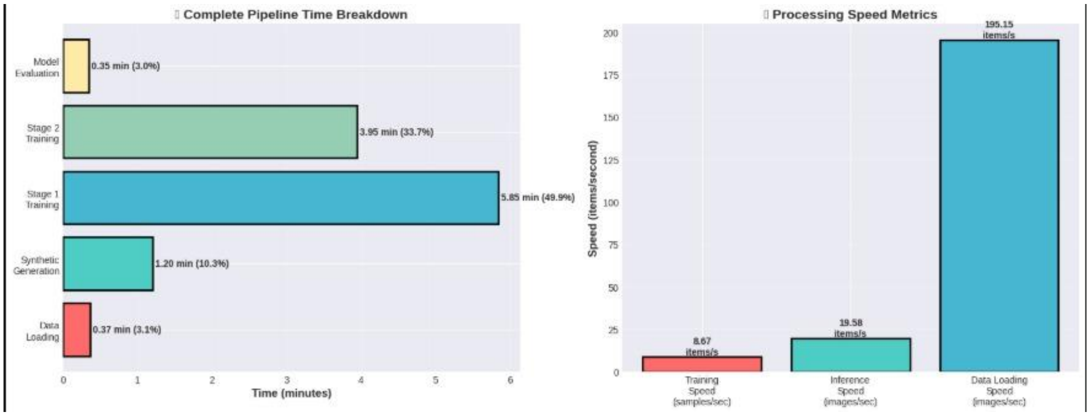


ResNet-18 with Finetuning :

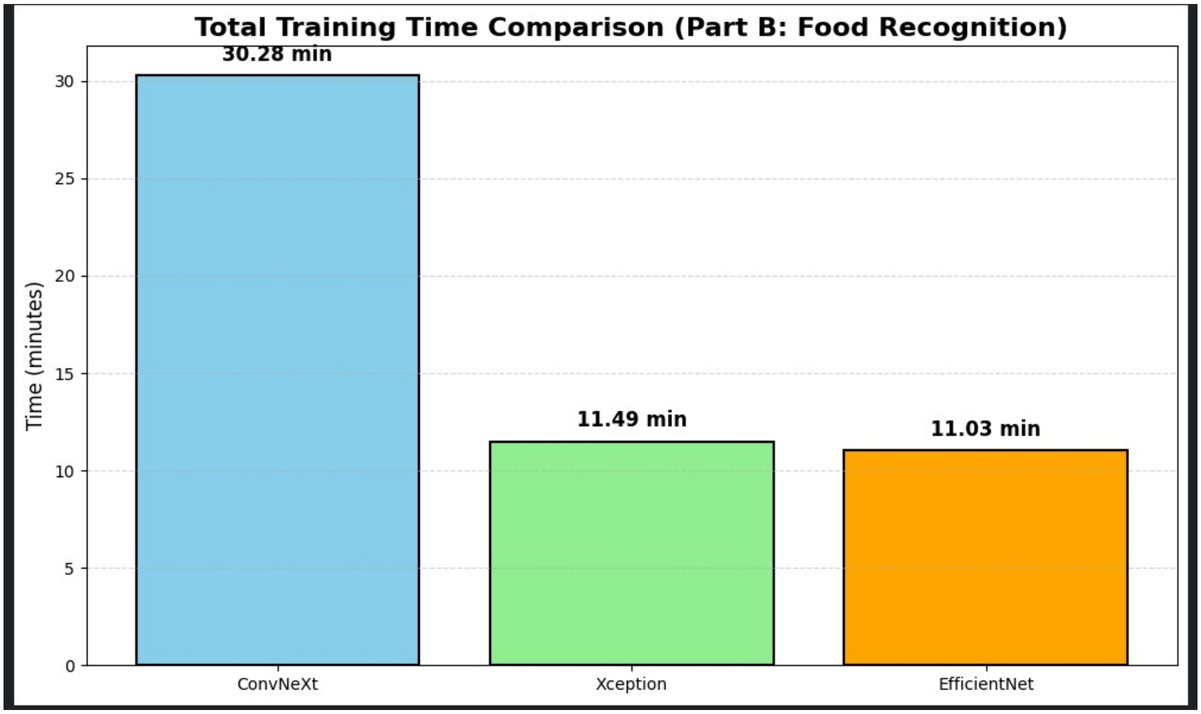


Multi Fruits:

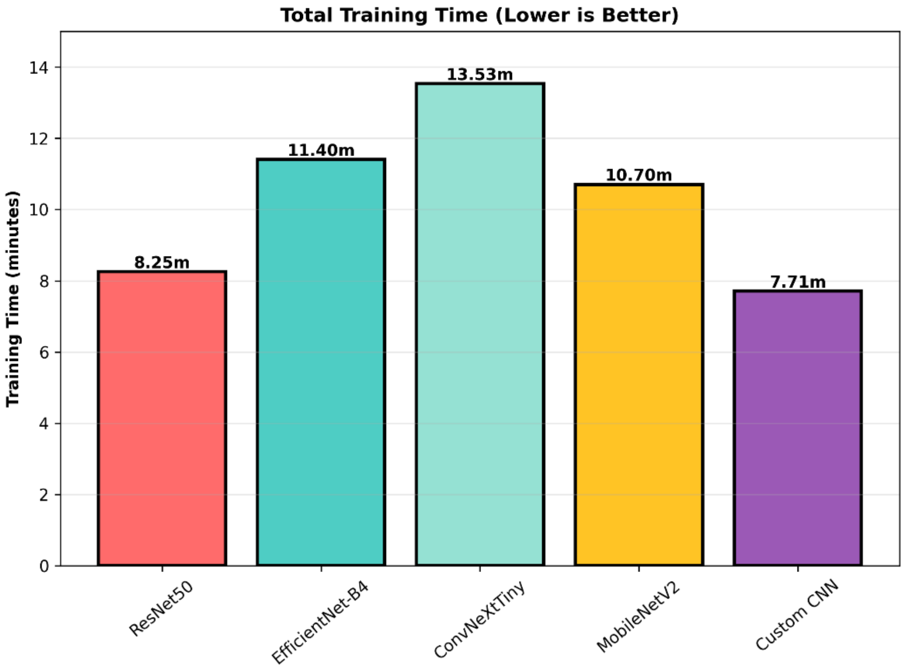

Training Time Evolution Across All Epochs
Stage 1 (Frozen Backbone)
Stage 2 (Fine-tuning)
Time (seconds)
24.4
24.2
24.0
23.8
23.6
23.4
23.2
23.0
0
5
10
15
20
25
Epoch Number
Cumulative Training Time
Stage 1 Cumulative
Stage 2 Cumulative
Cumulative Time (minutes)
10
8
6
4
2
0
0
5
10
15
20
25
Epoch Number
Complete Pipeline Time Breakdown
Processing Speed Metrics
195.15


Complete Pipeline Time Breakdown
Model Evaluation
0.35 min (3.0%)
Stage 2 Training
3.95 min (33.7%)
Stage 1 Training
5.85 min (49.9%)
Synthetic Generation
1.20 min (10.3%)
Data Loading
0.37 min (3.1%)
0
1
2
3
4
5
6
Time (minutes)
Processing Speed Metrics
Speed (items/second)
200
175
150
125
100
75
50
25
0
195.15 items/s
19.58 items/s
8.67 items/s
Training Speed (samples/sec)
Inference Speed (images/sec)
Data Loading Speed (images/sec)

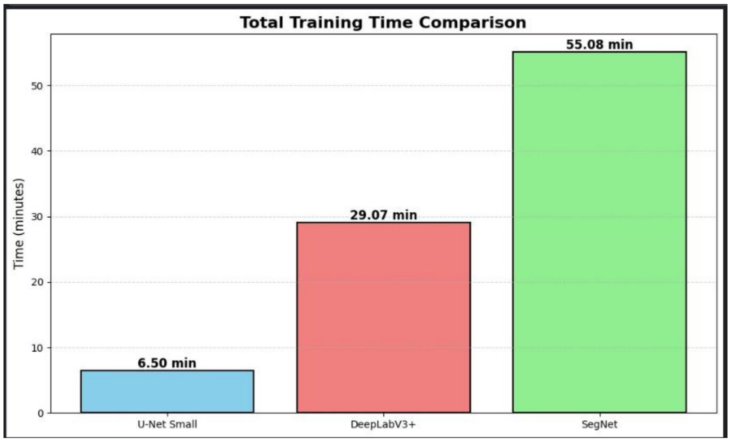
Part B


Total Training Time Comparison (Part B: Food Recognition)
Time (minutes)
30.28 min
11.49 min
11.03 min
0
5
10
15
20
25
30
ConvNeXt
Xception
EfficientNet

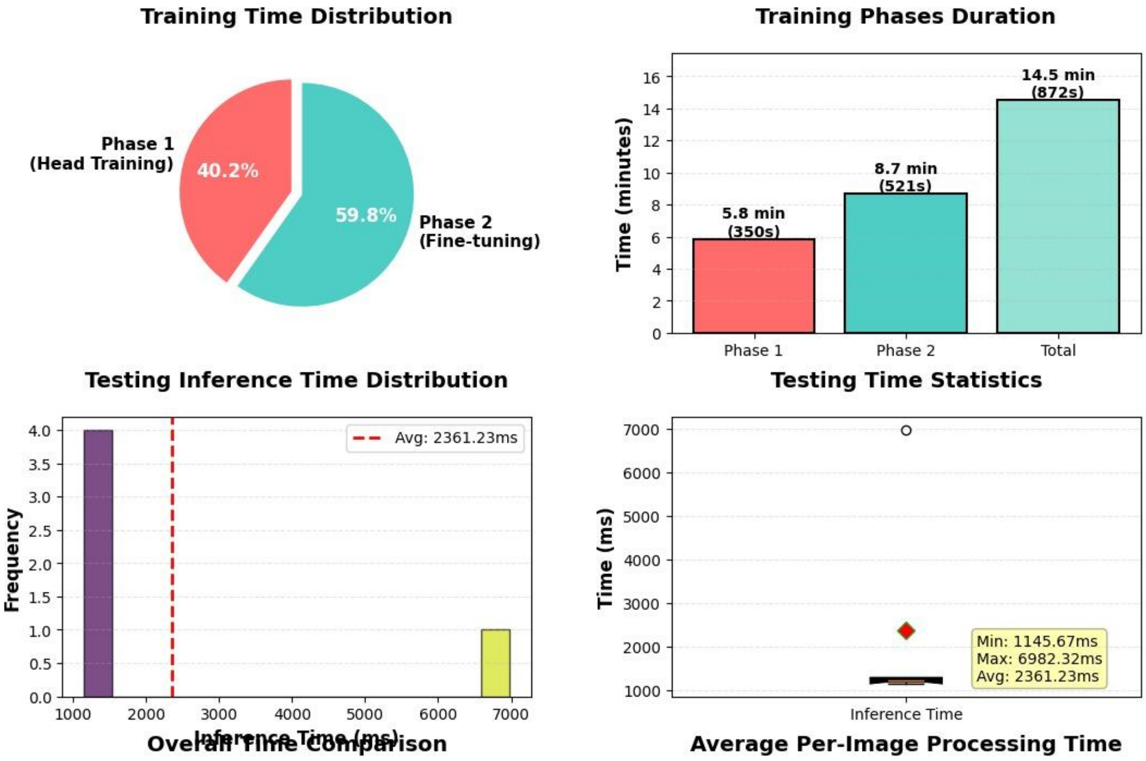
Part C


Total Training Time (Lower is Better)
Training Time (minutes)
8.25m
11.40m
13.53m
10.70m
7.71m
0
2
4
6
8
10
12
14
ResNet50
EfficientNet-B4
ConvNeXtTiny
MobileNetV2
Custom CNN

Part D

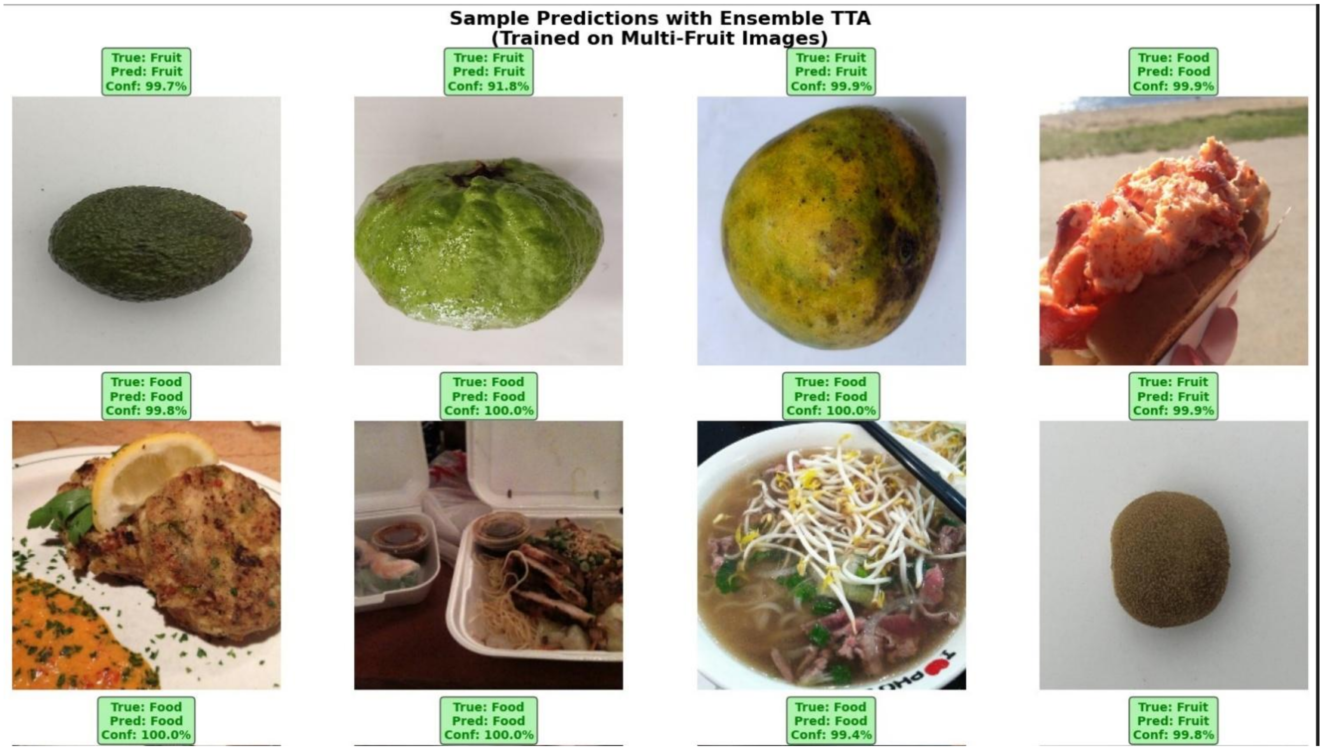

Total Training Time Comparison
Time (minutes)
0
10
20
30
40
50
6.50 min
29.07 min
55.08 min
U-Net Small
DeepLabV3+
SegNet

Part E


Training Time Distribution
Phase 1
(Head Training)
40.2%
59.8%
Phase 2
(Fine-tuning)
Training Phases Duration
Time (minutes)
0
2
4
6
8
10
12
14
16
5.8 min
(350s)
8.7 min
(521s)
14.5 min
(872s)
Phase 1
Phase 2
Total
Testing Inference Time Distribution
Frequency
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Avg: 2361.23ms
1000
2000
3000
4000
5000
6000
7000
Inference Time (ms)
Testing Time Statistics
Time (ms)
1000
2000
3000
4000
5000
6000
7000
Min: 1145.67ms
Max: 6982.32ms
Avg: 2361.23ms
Inference Time
Overall Time Comparison
Average Per-Image Processing Time

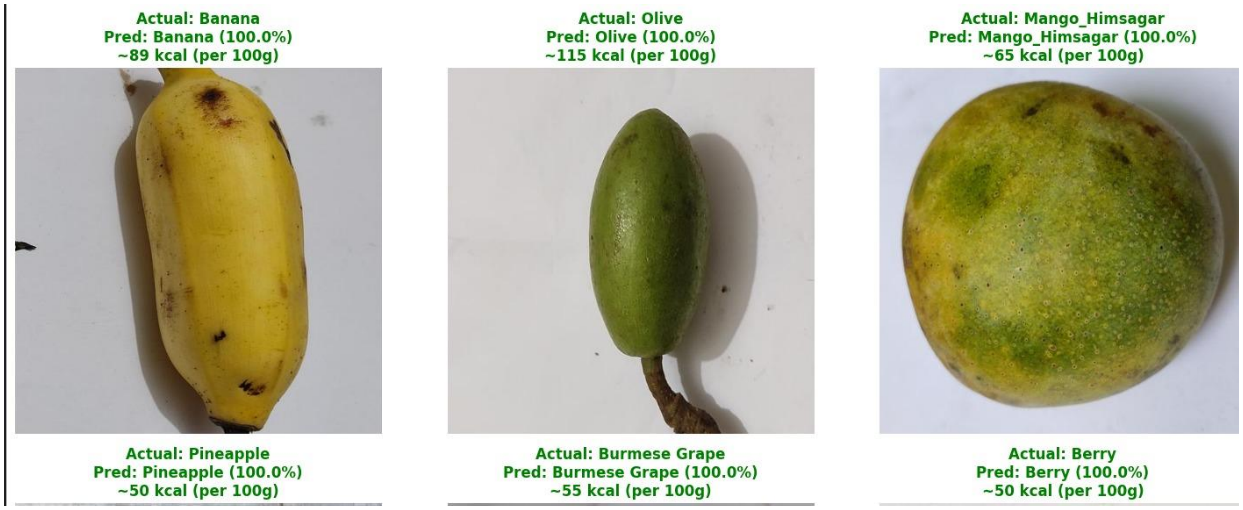
Fourth: Visualization of the Classification

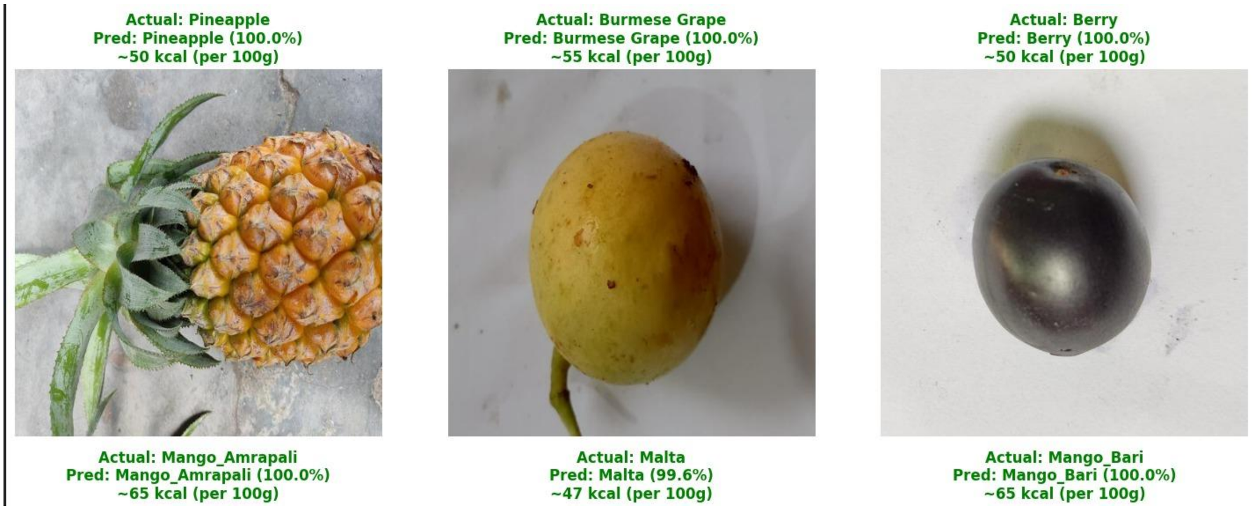
Part A

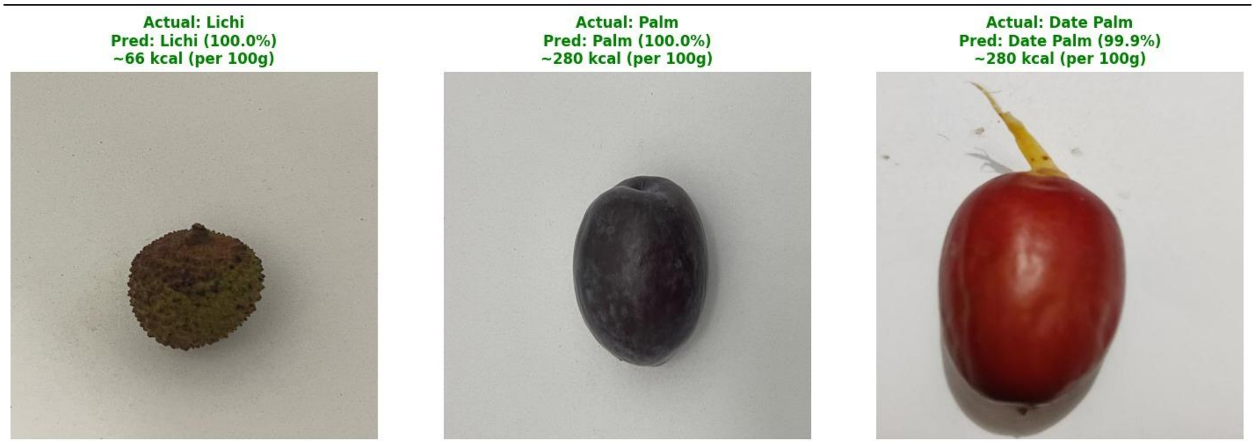


Sample Predictions with Ensemble TTA
(Trained on Multi-Fruit Images)

Part C

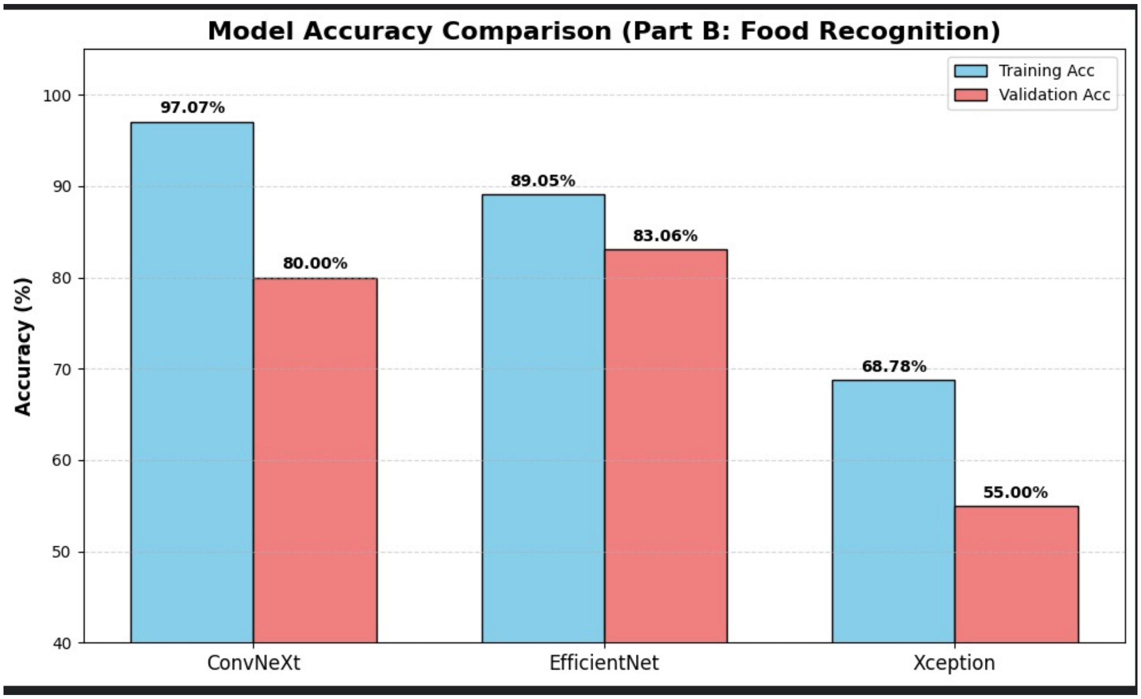

Actual: Banana
Pred: Banana (100.0%)
~89 kcal (per 100g)
Actual: Olive
Pred: Olive (100.0%)
~115 kcal (per 100g)
Actual: Mango_Himsagar
Pred: Mango_Himsagar (100.0%)
~65 kcal (per 100g)
Actual: Pineapple
Pred: Pineapple (100.0%)
~50 kcal (per 100g)
Actual: Burmese Grape
Pred: Burmese Grape (100.0%)
~55 kcal (per 100g)
Actual: Berry
Pred: Berry (100.0%)
~50 kcal (per 100g)


Actual: Pineapple
Pred: Pineapple (100.0%)
~50 kcal (per 100g)
Actual: Burmese Grape
Pred: Burmese Grape (100.0%)
~55 kcal (per 100g)
Actual: Berry
Pred: Berry (100.0%)
~50 kcal (per 100g)
Actual: Mango_Amrapali
Pred: Mango_Amrapali (100.0%)
~65 kcal (per 100g)
Actual: Malta
Pred: Malta (99.6%)
~47 kcal (per 100g)
Actual: Mango_Bari
Pred: Mango_Bari (100.0%)
~65 kcal (per 100g)


Actual: Lichi
Pred: Lichi (100.0%)
~66 kcal (per 100g)
Actual: Palm
Pred: Palm (100.0%)
~280 kcal (per 100g)
Actual: Date Palm
Pred: Date Palm (99.9%)
~280 kcal (per 100g)

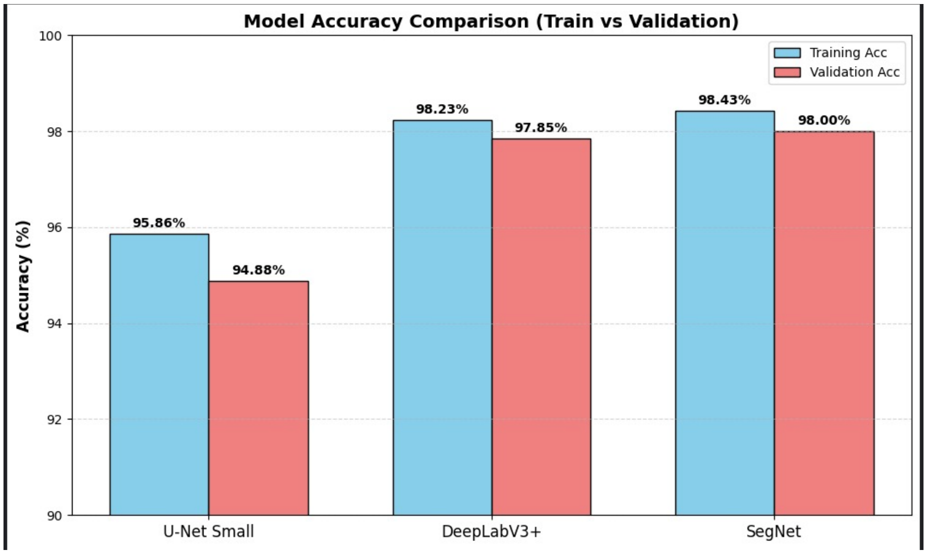
Fifth: Training and Validation Accuracy of the Siamese Model

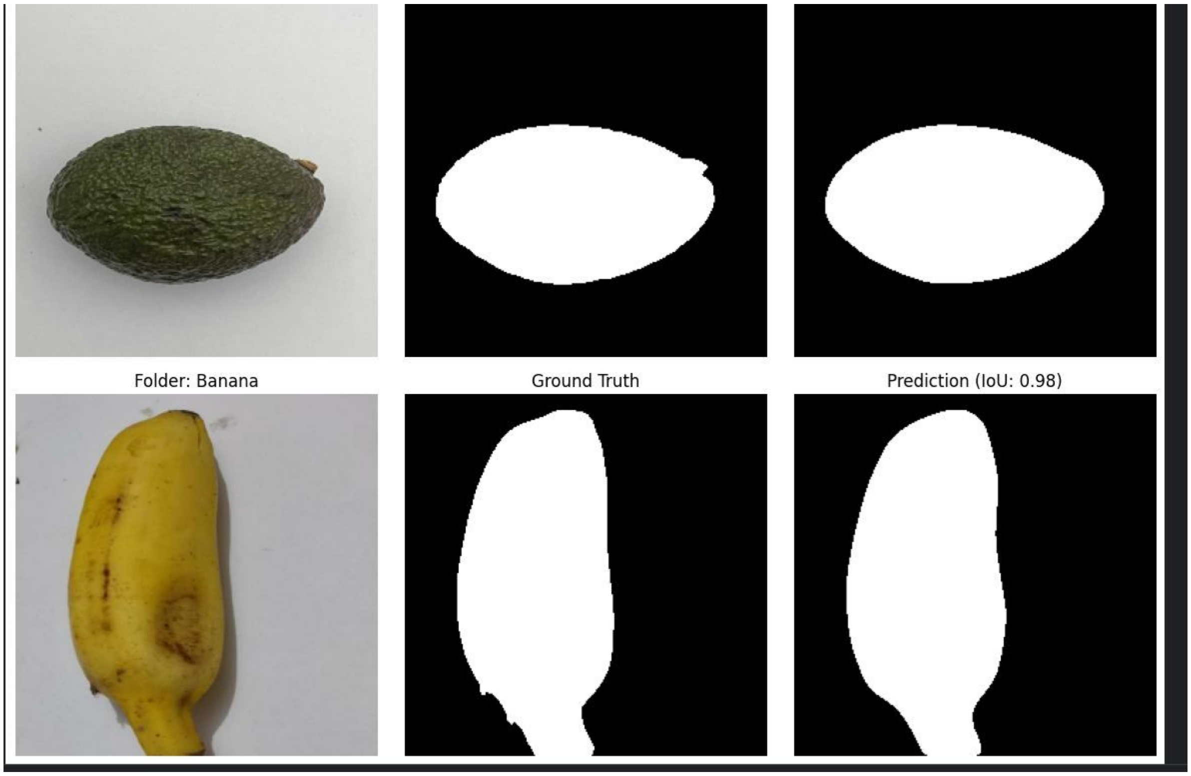
Part B

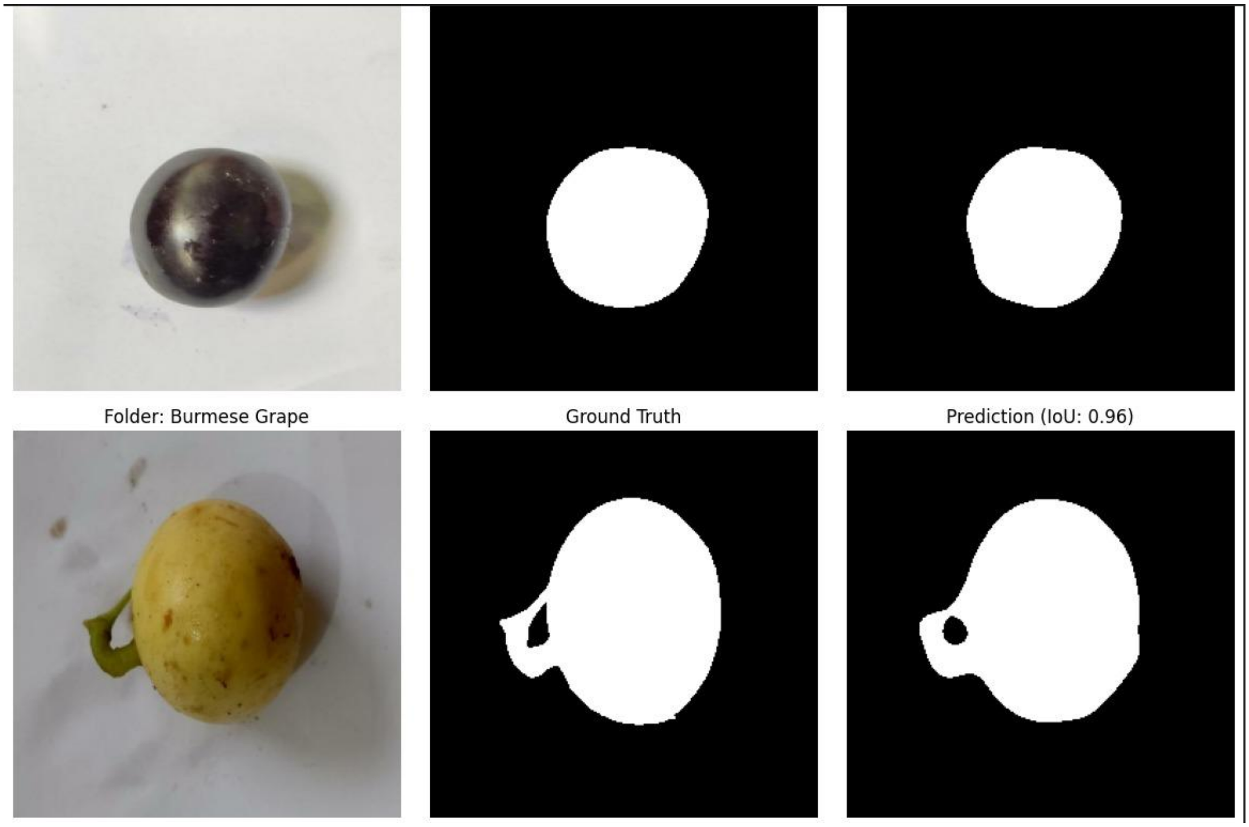


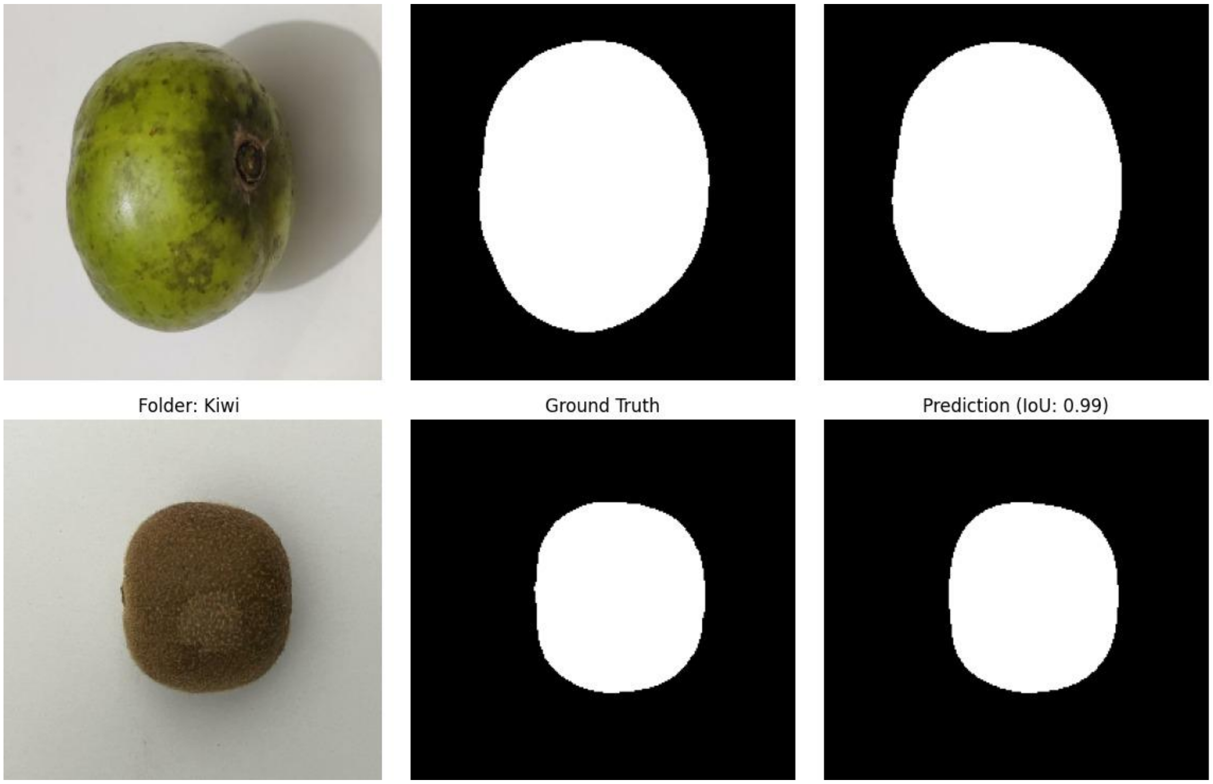
Sixth: Segmentation Training and Validation Accuracy + Visualization of the Segmentation Outputs

Part D

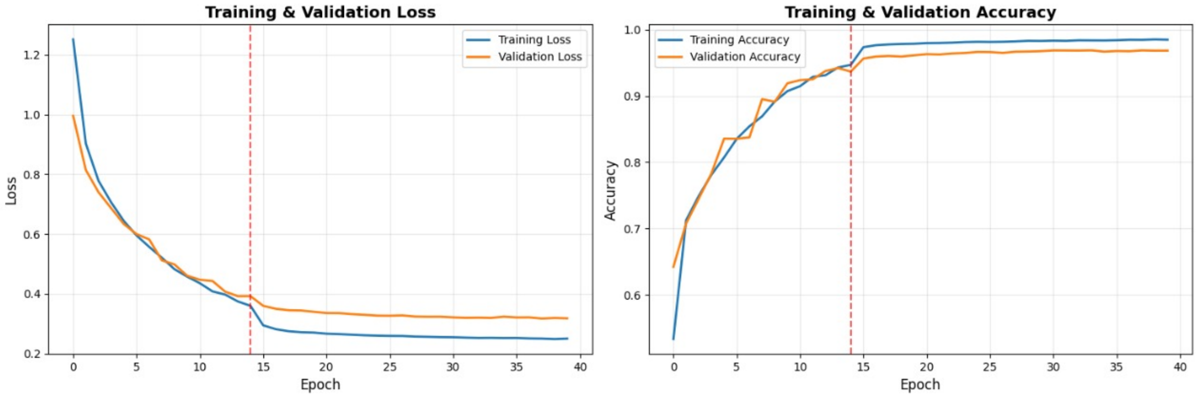

Model Accuracy Comparison (Train vs Validation)
Accuracy (%)
100
98
96
94
92
90
Training Acc
Validation Acc
95.86%
94.88%
98.23%
97.85%
98.43%
98.00%
U-Net Small
DeepLabV3+
SegNet

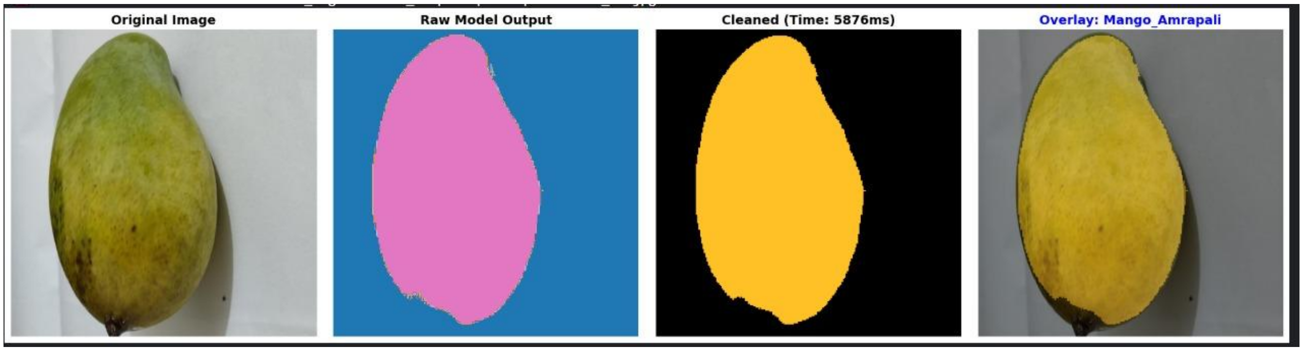

Folder: Banana
Ground Truth
Prediction (IoU: 0.98)

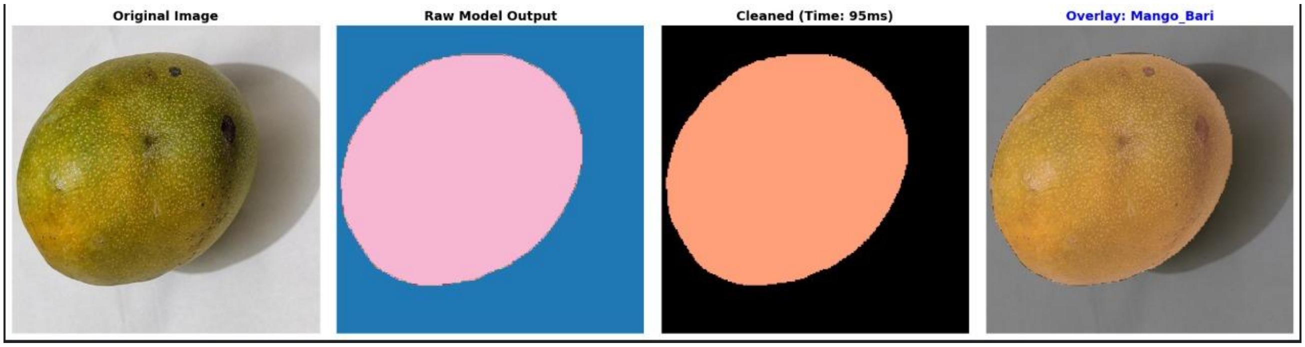

Folder: Burmese Grape
Ground Truth
Prediction (IoU: 0.96)

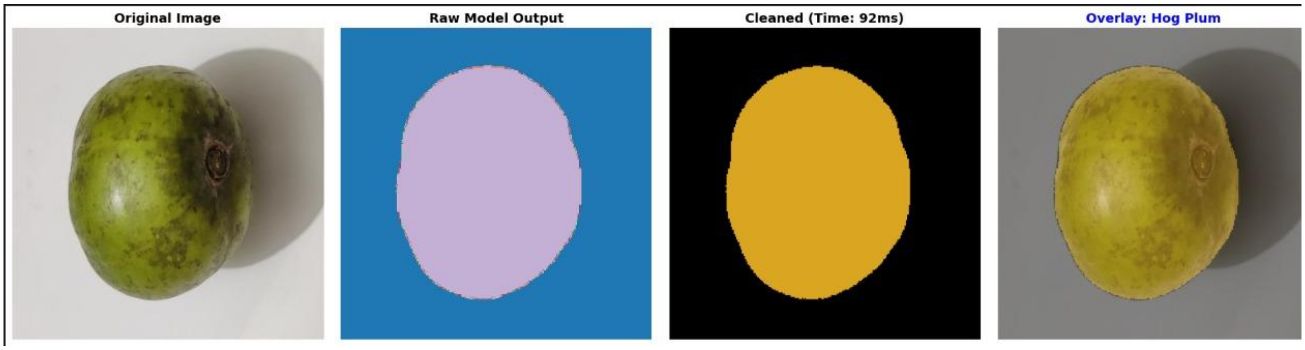

Folder: Kiwi
Ground Truth
Prediction (IoU: 0.99)

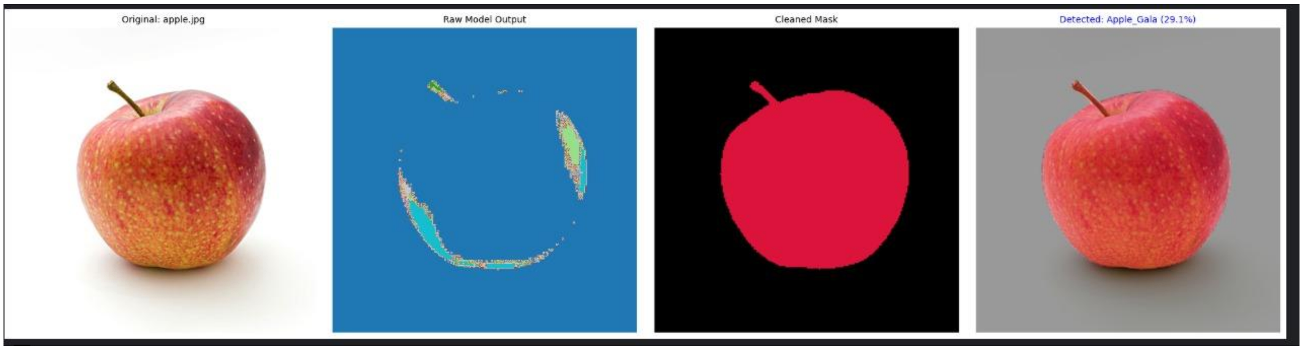
Part E


Training & Validation Loss
Training Loss
Validation Loss
Loss
1.2
1.0
0.8
0.6
0.4
0.2
0
5
10
15
20
25
30
35
40
Epoch
Training & Validation Accuracy
Training Accuracy
Validation Accuracy
Accuracy
1.0
0.9
0.8
0.7
0.6
0
5
10
15
20
25
30
35
40
Epoch


Original Image
Raw Model Output
Cleaned (Time: 5876ms)
Overlay: Mango_Amrapali


Original Image
Raw Model Output
Cleaned (Time: 95ms)
Overlay: Mango_Bari


Original Image
Raw Model Output
Cleaned (Time: 92ms)
Overlay: Hog Plum


Original: apple.jpg
Raw Model Output
Cleaned Mask
Detected: Apple_Gala (29.1%)