Date Fruit Image Classification

Ahmed Alkhulayfi

CSC462 – Machine Learning

November 24, 2024

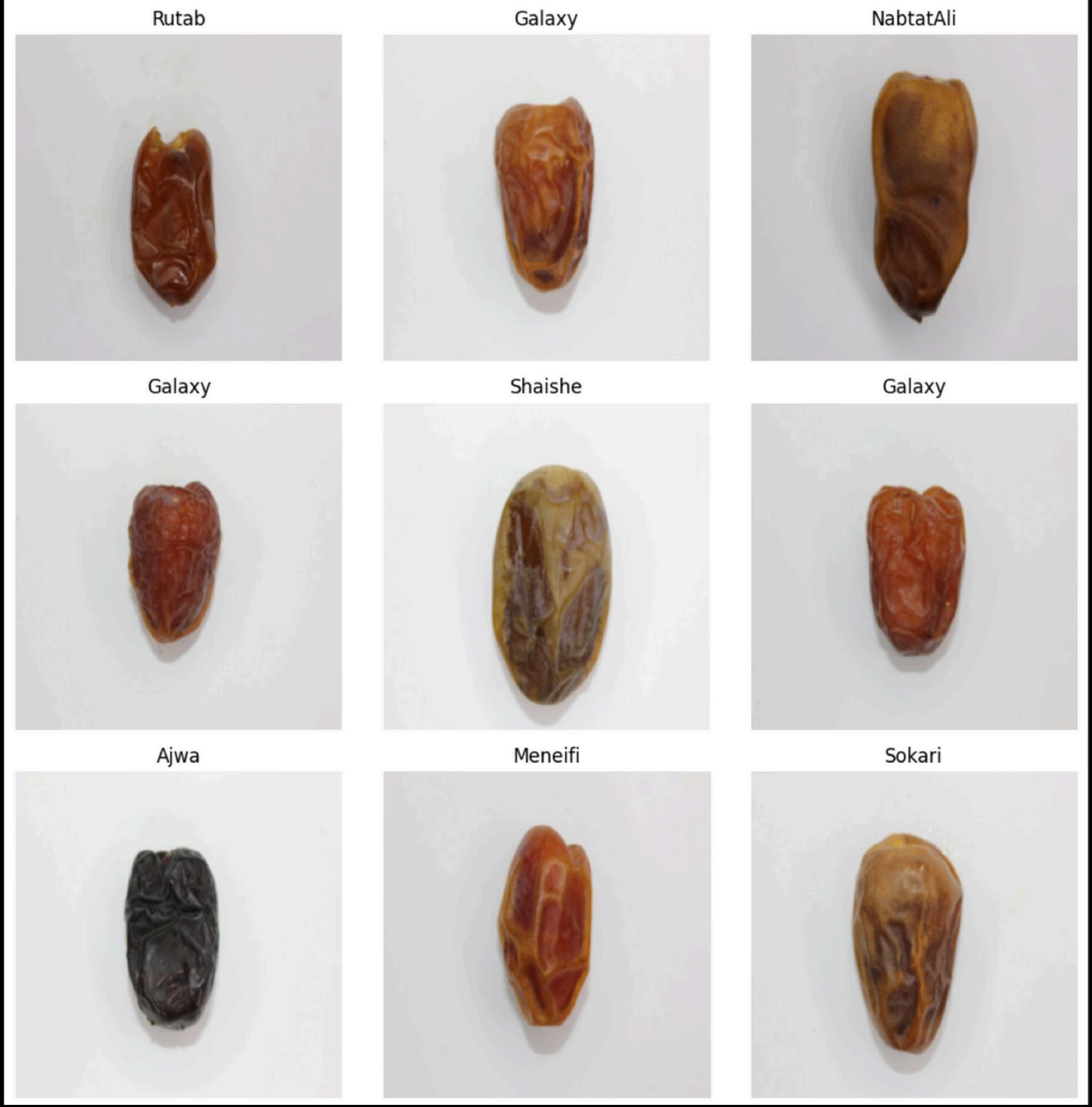
Overview

Introduction

This project centers on leveraging **machine learning** to automate the classification of **date fruit images**. By utilizing a custom **Convolutional Neural Network** (**CNN**), we aim to improve the efficiency and accuracy of identifying different date types, enhancing productivity and ensuring consistent quality standards in date classification.

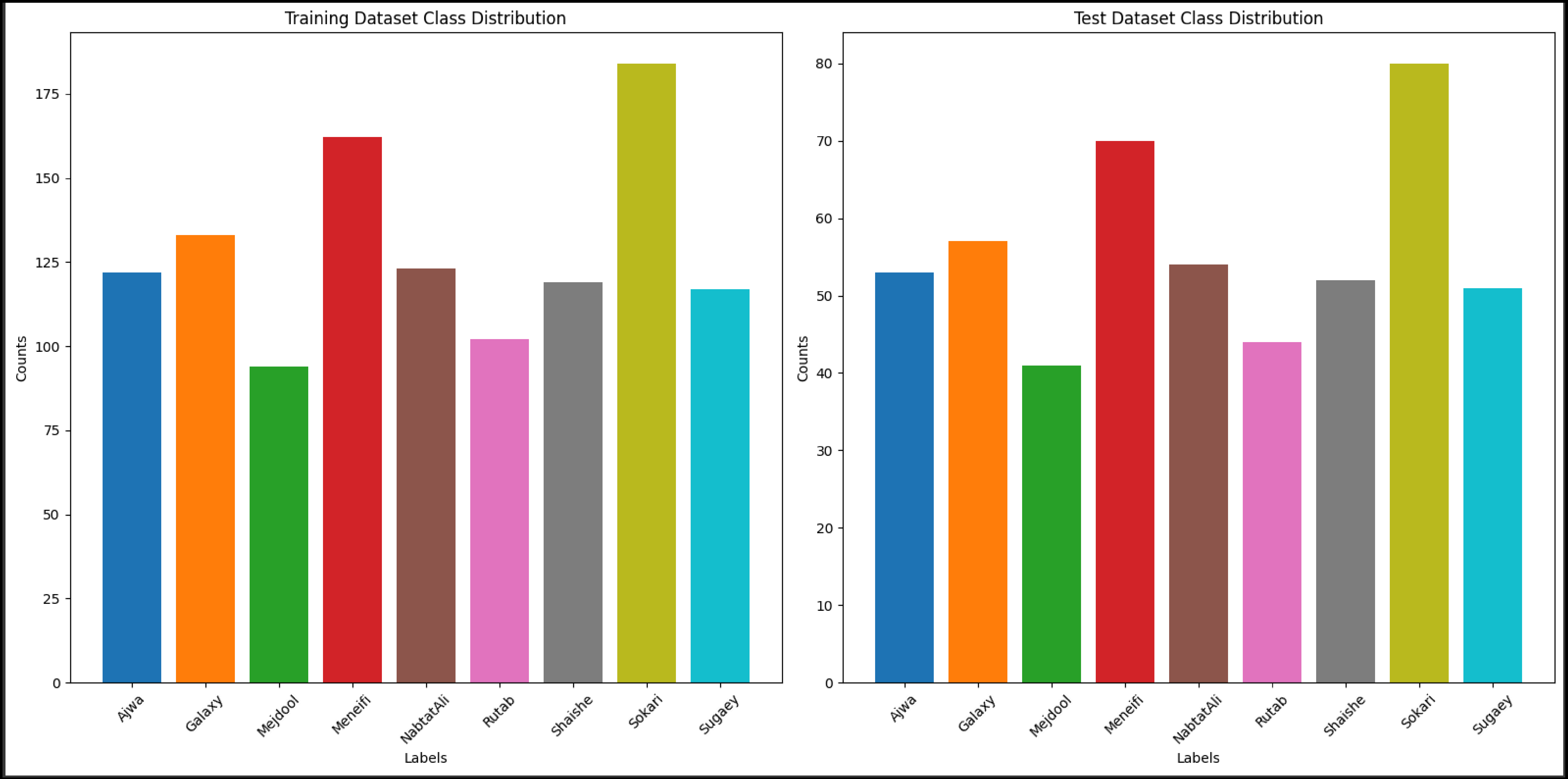
Dataset

The dataset consists of images of **9 date fruit types**, including Ajwa, Galaxy, Mejdool, Meneifi, NabtatAli, Rutab, Shaishe, Sokari and Sugaey.



Class Distribution

- Training Set: 1,156 images
- Test Set: 502 images



Model Architecture

A **custom** Convolutional Neural Network (CNN) designed specifically for classifying date fruit images.

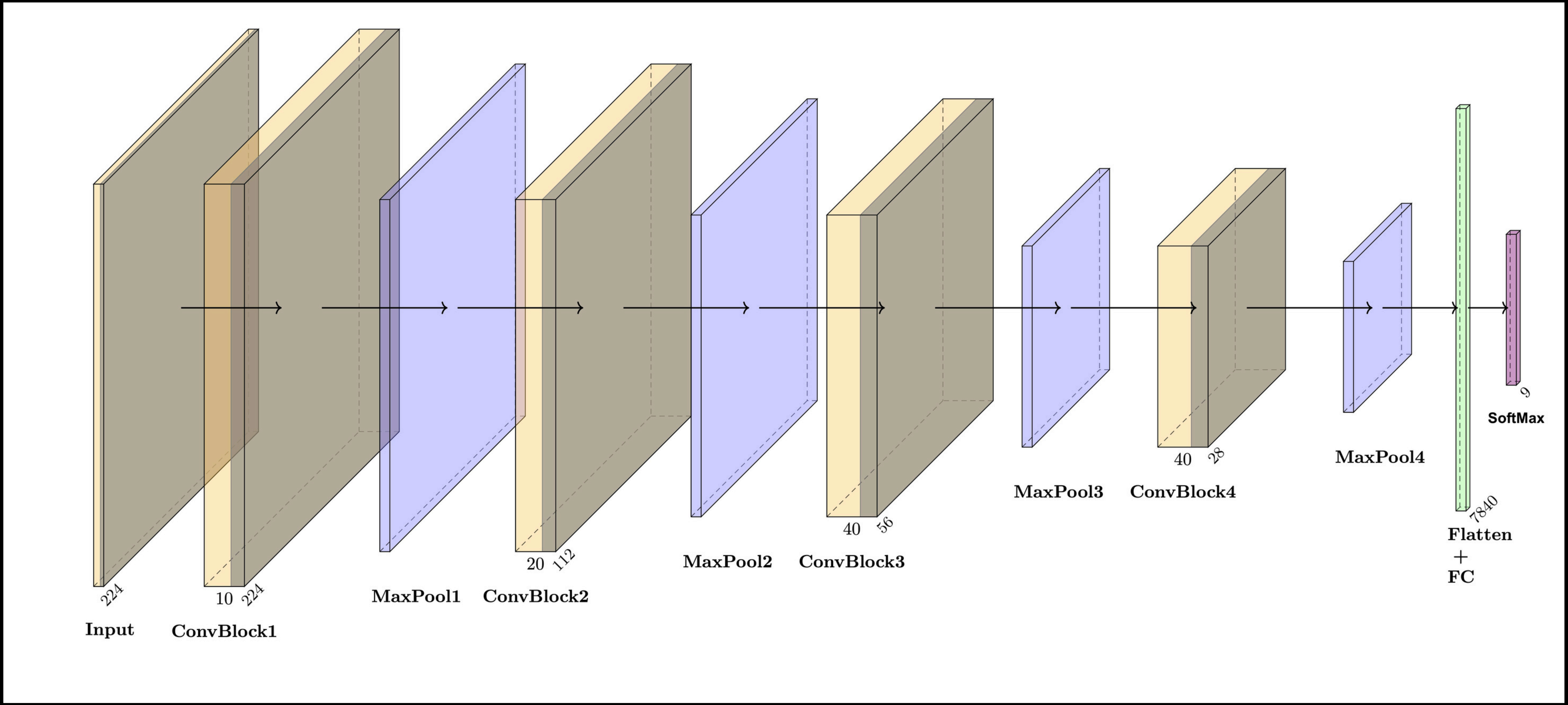


Diagram generated using the PlotNeuralNet library [1].

Model Architecture

Architecture Details

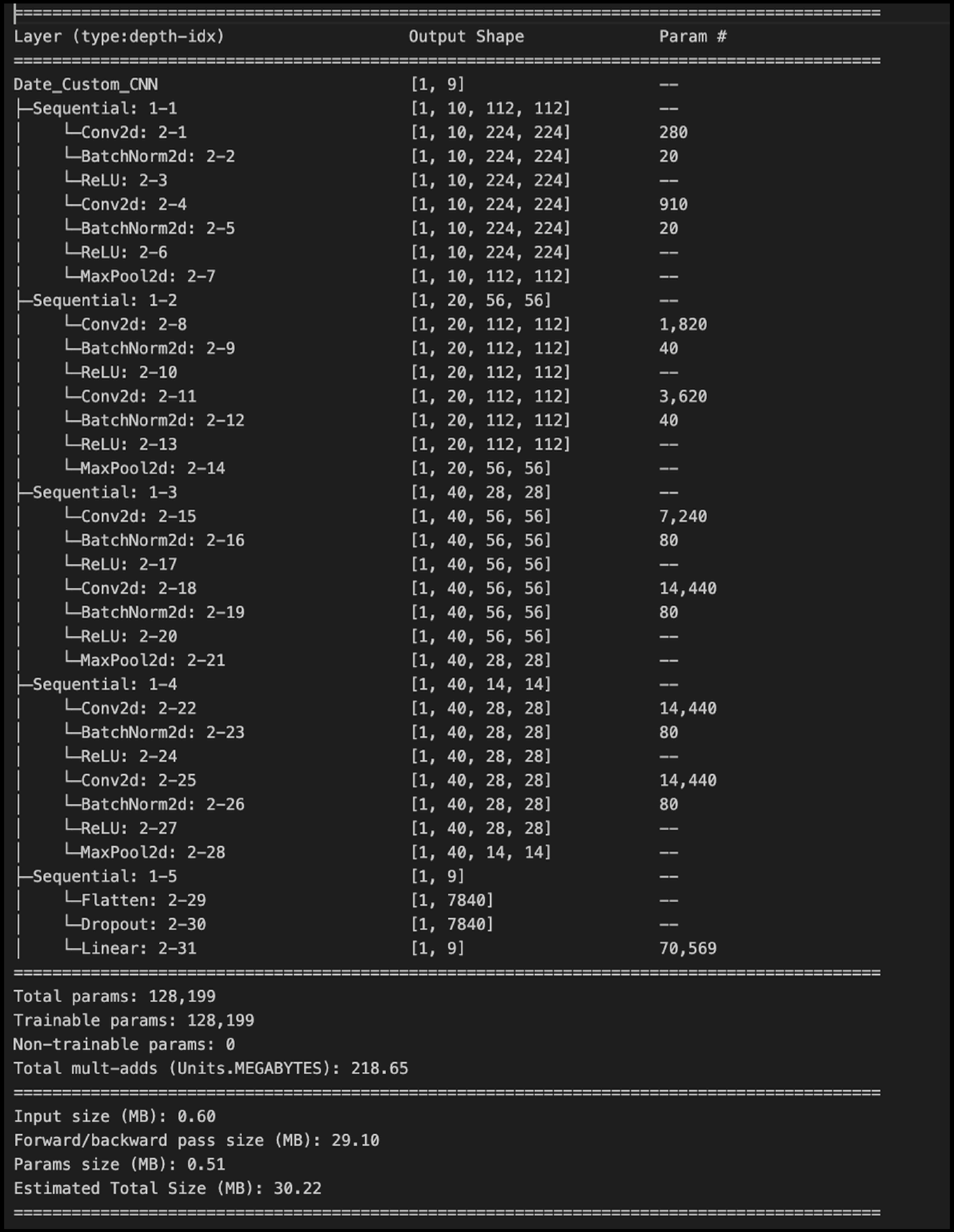
- Input Layer: Accepts images of size 224x224x3.
- Convolutional Blocks:
  - 4 convolutional blocks, each followed by:
    - Batch Normalization
    - ReLU Activation
    - Max Pooling
- Fully Connected Layer:
  Flattens the feature maps into a single vector.
  Applies Dropout (rate: 0.5) to reduce overfitting.
  Outputs class scores for 9 date fruit types.

Model Summary:

Total Parameters: 128,199

Activation Function: ReLU

Final Layer Activation: Softmax



```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
Date_Custom_CNN                          [1, 9]                    --
├─Sequential: 1-1                        [1, 10, 112, 112]         --
│    └─Conv2d: 2-1                       [1, 10, 224, 224]         280
│    └─BatchNorm2d: 2-2                  [1, 10, 224, 224]         20
│    └─ReLU: 2-3                         [1, 10, 224, 224]         --
│    └─Conv2d: 2-4                       [1, 10, 224, 224]         910
│    └─BatchNorm2d: 2-5                  [1, 10, 224, 224]         20
│    └─ReLU: 2-6                         [1, 10, 224, 224]         --
│    └─MaxPool2d: 2-7                    [1, 10, 112, 112]         --
├─Sequential: 1-2                        [1, 20, 56, 56]           --
│    └─Conv2d: 2-8                       [1, 20, 112, 112]         1,820
│    └─BatchNorm2d: 2-9                  [1, 20, 112, 112]         40
│    └─ReLU: 2-10                        [1, 20, 112, 112]         --
│    └─Conv2d: 2-11                      [1, 20, 112, 112]         3,620
│    └─BatchNorm2d: 2-12                 [1, 20, 112, 112]         40
│    └─ReLU: 2-13                        [1, 20, 112, 112]         --
│    └─MaxPool2d: 2-14                   [1, 20, 56, 56]           --
├─Sequential: 1-3                        [1, 40, 28, 28]           --
│    └─Conv2d: 2-15                      [1, 40, 56, 56]           7,240
│    └─BatchNorm2d: 2-16                 [1, 40, 56, 56]           80
│    └─ReLU: 2-17                        [1, 40, 56, 56]           --
│    └─Conv2d: 2-18                      [1, 40, 56, 56]           14,440
│    └─BatchNorm2d: 2-19                 [1, 40, 56, 56]           80
│    └─ReLU: 2-20                        [1, 40, 56, 56]           --
│    └─MaxPool2d: 2-21                   [1, 40, 28, 28]           --
├─Sequential: 1-4                        [1, 40, 14, 14]           --
│    └─Conv2d: 2-22                      [1, 40, 28, 28]           14,440
│    └─BatchNorm2d: 2-23                 [1, 40, 28, 28]           80
│    └─ReLU: 2-24                        [1, 40, 28, 28]           --
│    └─Conv2d: 2-25                      [1, 40, 28, 28]           14,440
│    └─BatchNorm2d: 2-26                 [1, 40, 28, 28]           80
│    └─ReLU: 2-27                        [1, 40, 28, 28]           --
│    └─MaxPool2d: 2-28                   [1, 40, 14, 14]           --
├─Sequential: 1-5                        [1, 9]                    --
│    └─Flatten: 2-29                     [1, 7840]                 --
│    └─Dropout: 2-30                     [1, 7840]                 --
│    └─Linear: 2-31                      [1, 9]                    70,569
==========================================================================================
Total params: 128,199
Trainable params: 128,199
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 218.65
==========================================================================================
Input size (MB): 0.60
Forward/backward pass size (MB): 29.10
Params size (MB): 0.51
Estimated Total Size (MB): 30.22
==========================================================================================
```

Data Augmentation

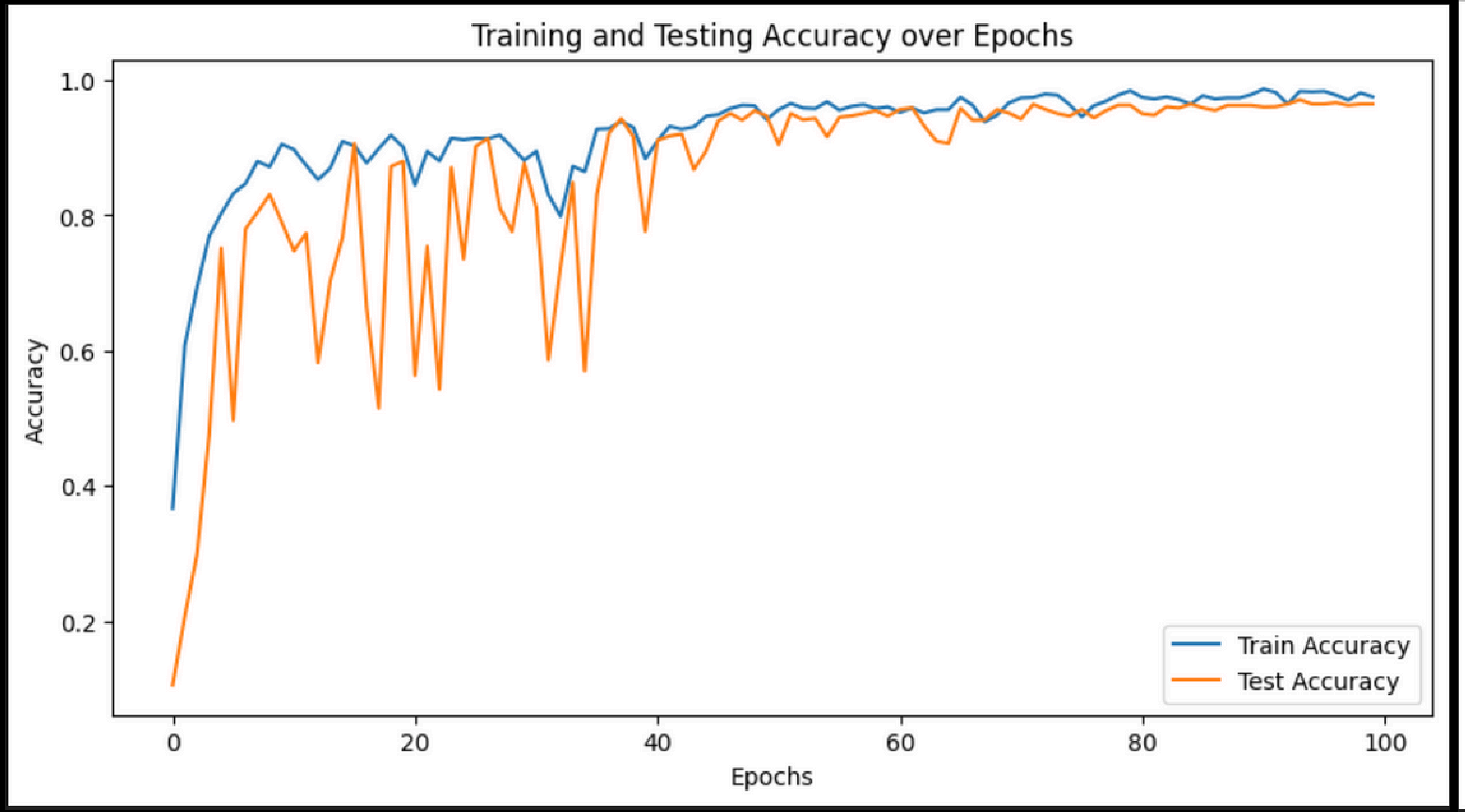
Techniques Applied:

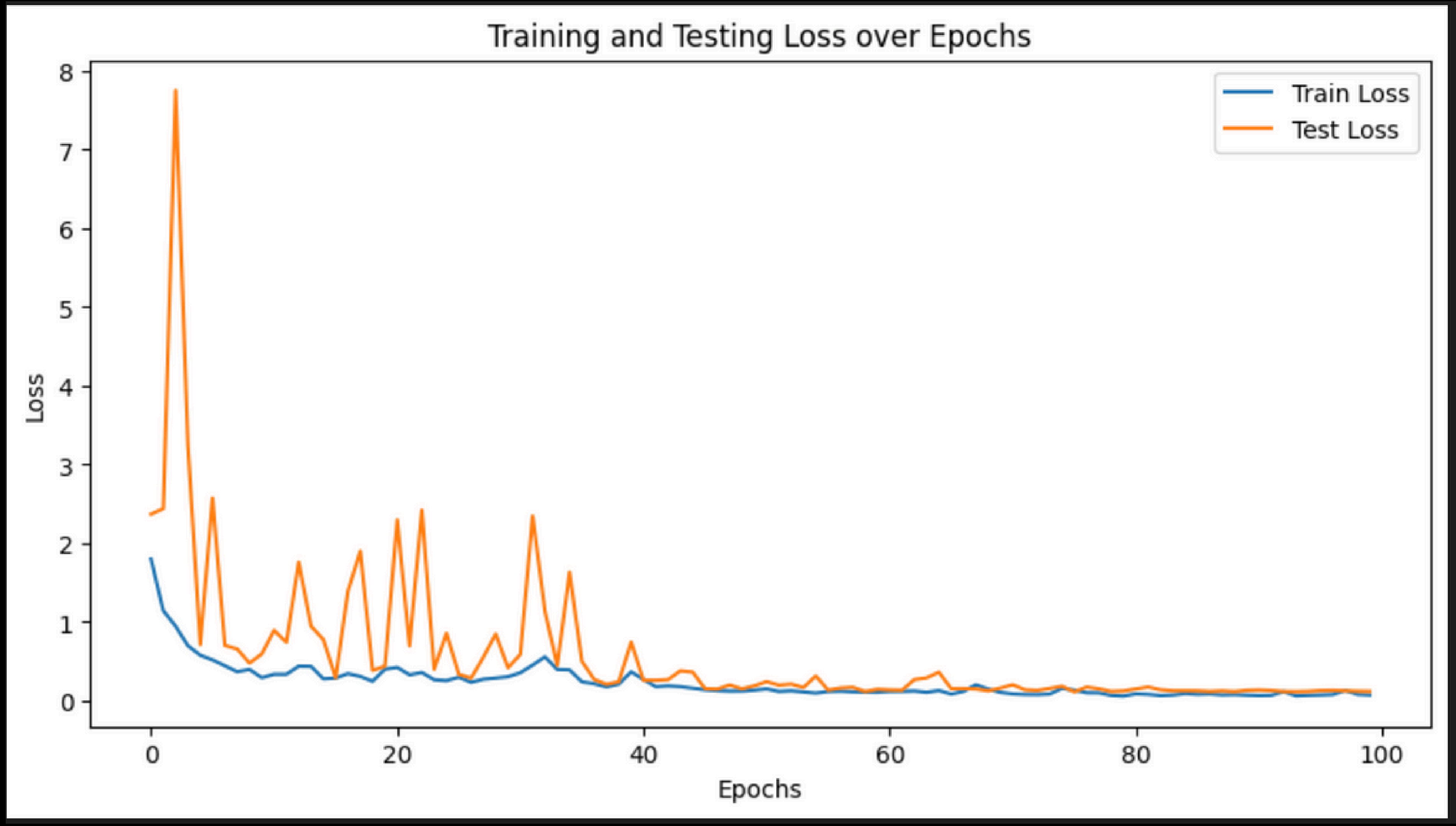
1. Resize:
   - All images resized to 224x224 pixels for uniform input size to the model.
2. Random Horizontal Flip:
   - Flips images horizontally with a 50% probability to simulate left-right variations.
3. Random Rotation:
   - Rotates images randomly by up to ±10 degrees to account for positional variations.
4. Normalization (via ToTensor()):
   - Converts pixel values to tensors with values in the range [0, 1].

Training and Hyperparameters

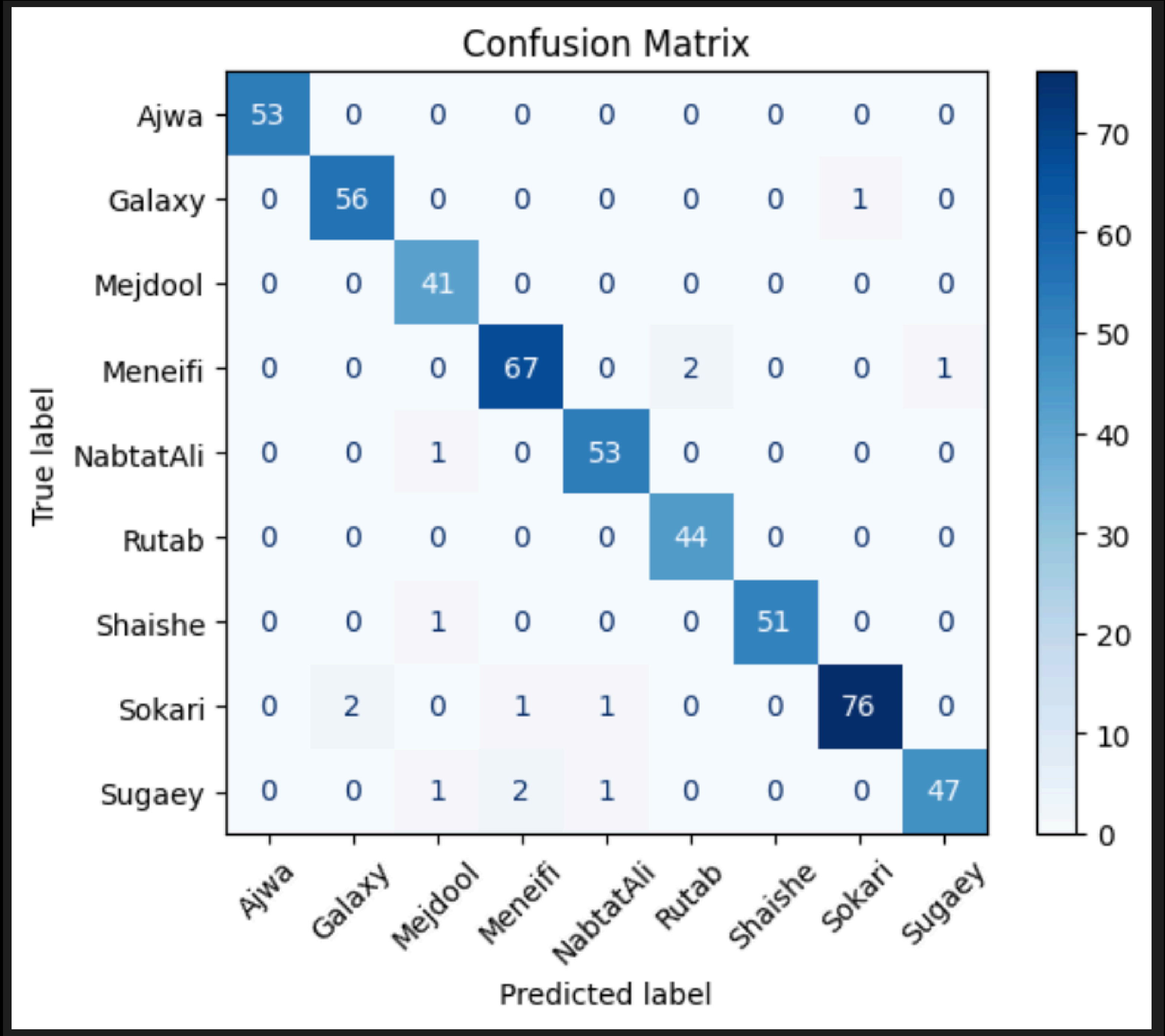
- Optimizer:
  - Stochastic Gradient Descent (SGD) with:
    - Learning Rate: 0.001
    - Momentum: 0.9
    - Weight Decay (L2): 0.00002

- Scheduler: Cosine Annealing LR.
- Batch Size: 64.
- Epochs: 100.

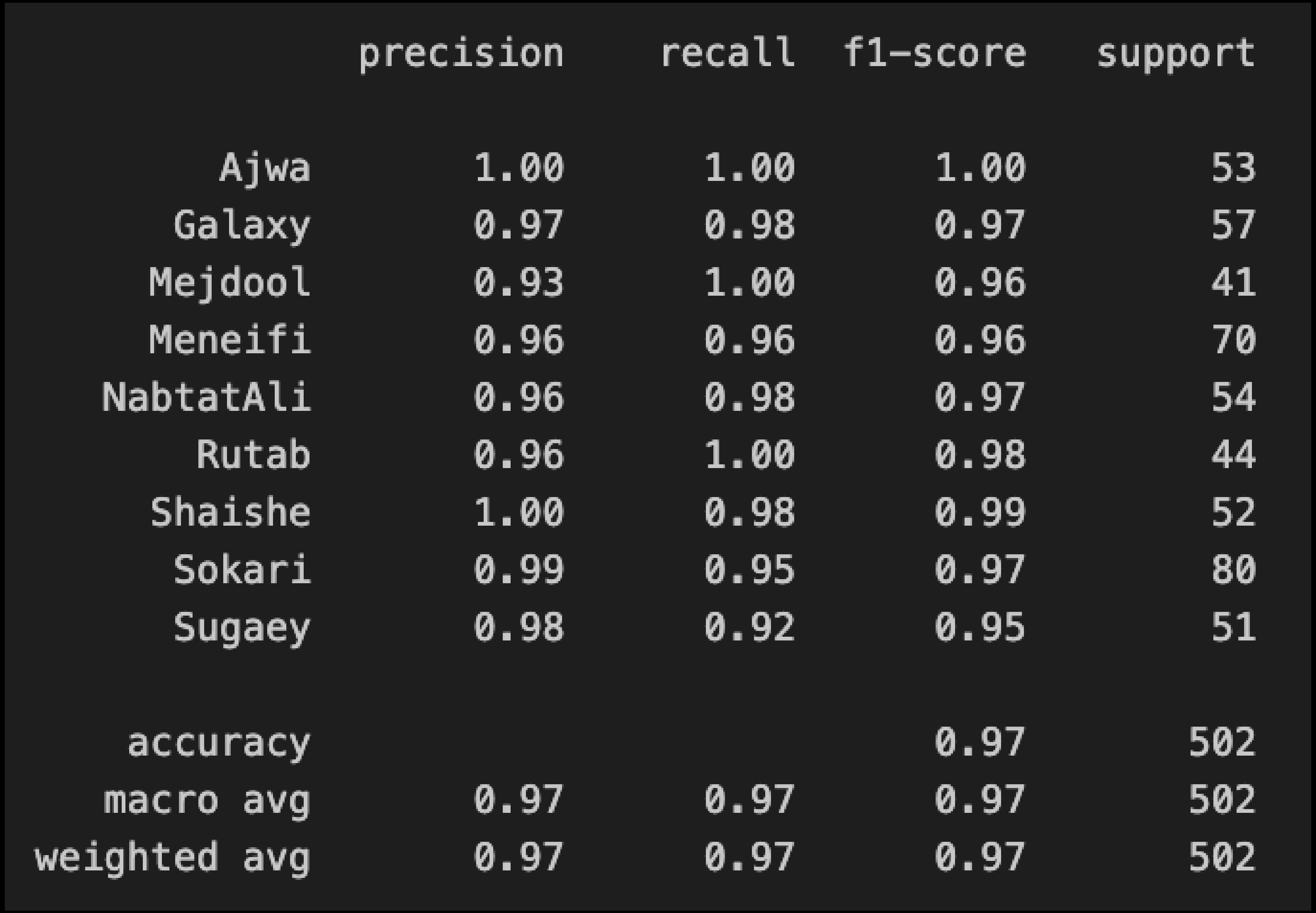




Results



True label \ Predicted label	Ajwa	Galaxy	Mejdool	Meneifi	NabtatAli	Rutab	Shaishe	Sokari	Sugaey
Ajwa	53	0	0	0	0	0	0	0	0
Galaxy	0	56	0	0	0	0	0	1	0
Mejdool	0	0	41	0	0	0	0	0	0
Meneifi	0	0	0	67	0	2	0	0	1
NabtatAli	0	0	1	0	53	0	0	0	0
Rutab	0	0	0	0	0	44	0	0	0
Shaishe	0	0	1	0	0	0	51	0	0
Sokari	0	2	0	1	1	0	0	76	0
Sugaey	0	0	1	2	1	0	0	0	47



```
              precision    recall  f1-score   support

        Ajwa       1.00      1.00      1.00        53
      Galaxy       0.97      0.98      0.97        57
     Mejdool       0.93      1.00      0.96        41
     Meneifi       0.96      0.96      0.96        70
   NabtatAli       0.96      0.98      0.97        54
       Rutab       0.96      1.00      0.98        44
     Shaishe       1.00      0.98      0.99        52
      Sokari       0.99      0.95      0.97        80
      Sugaey       0.98      0.92      0.95        51

    accuracy                           0.97       502
   macro avg       0.97      0.97      0.97       502
weighted avg       0.97      0.97      0.97       502
```

Comparison with EfficientNet-B0

Custom Model:

- Parameters: 128,199
- Model Size: ~30.22 MB
- Test Accuracy: 97%

EfficientNet-B0:

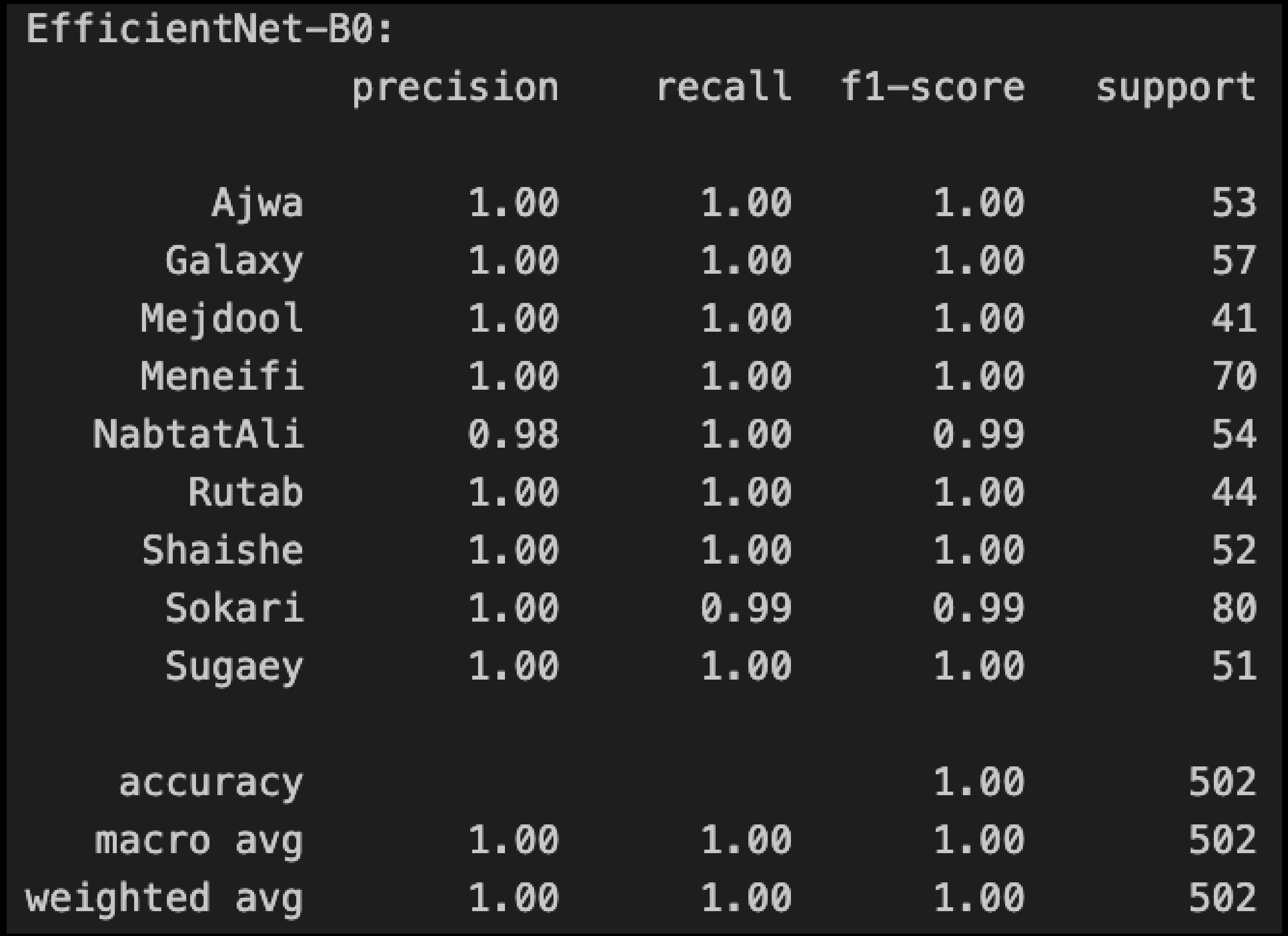
- Parameters: Original 5,288,548 (after modification: 4,019,077)
- Model Size: ~124.56 MB
- Test Accuracy: 100%

Model Size Comparison

- Calculation: 124.56 / 30.22 ≈ 4.12
- EfficientNet-B0 is approximately **4.12 times** bigger than the Custom Model.

Parameter Count Comparison

- Calculation: 4,019,077 / 128,199 ≈ 31.36
- EfficientNet-B0 has approximately **31.36 times** more parameters than the Custom Model.



```
EfficientNet-B0:
              precision    recall  f1-score   support

        Ajwa       1.00      1.00      1.00        53
      Galaxy       1.00      1.00      1.00        57
     Mejdool       1.00      1.00      1.00        41
     Meneifi       1.00      1.00      1.00        70
   NabtatAli       0.98      1.00      0.99        54
       Rutab       1.00      1.00      1.00        44
     Shaishe       1.00      1.00      1.00        52
      Sokari       1.00      0.99      0.99        80
      Sugaey       1.00      1.00      1.00        51

    accuracy                           1.00       502
   macro avg       1.00      1.00      1.00       502
weighted avg       1.00      1.00      1.00       502
```

Challenges and Learnings

Challenges

1. Dataset Imbalance:
    - Certain classes had fewer images, which could have impacted the model’s ability to generalize.
    - Solution: Applied data augmentation to enhance dataset diversity.
2. Overfitting:
    - Initial experiments showed high training accuracy but lower validation accuracy.
    - Solution: Used techniques like Dropout and regularization to mitigate overfitting.
3. Computational Limitations:
    - Training on large models like EfficientNet-B0 required significant computational resources.
    - Solution: Focused on optimizing the custom CNN for efficiency
4. Hyperparameter Tuning:
    - Finding the optimal learning rate, batch size, and momentum was time-intensive.
    - Solution: Iterative experimentation and fine-tuning

Challenges and Learnings

Learnings

1. Deep Learning Workflow:
    - Gained hands-on experience in designing and training a custom CNN.
2. Model Optimization:
    - Learned how to balance model size and accuracy, especially when comparing the custom model with pre-trained models.
3. Data Handling:
    - Understood the importance of data preprocessing and augmentation for improving model performance.
4. Evaluation Techniques:
    - Learned how to interpret metrics like precision, recall, and F1-score to evaluate model performance effectively.

Conclusion

- Project Summary:
  - Developed a custom CNN model for the classification of 9 types of date fruits.
  - Achieved a **97%** test accuracy, demonstrating the model’s effectiveness.
- Key Insights:
  - Data augmentation played a crucial role in enhancing model generalization.
  - The custom model provides a good trade-off between accuracy and efficiency compared to EfficientNet-B0.
- Challenges Overcome:
  - Addressed issues like overfitting, dataset imbalance, and hyperparameter tuning through systematic experimentation.
- Future Work:
  - Explore transfer learning with other pre-trained models to further improve accuracy.
  - Experiment with more complex architectures or larger datasets for better generalization.
  - Deploy the model in real-world applications for automated sorting of date fruits.