In [4]:
```python
# ============================================
# 1. IMPORT LIBRARIES
# ============================================
import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import (
    accuracy_score,
    classification_report,
    confusion_matrix
)

# Classical ML models
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression

import tensorflow as tf
from tensorflow.keras import layers, models, callbacks
```

In [6]:
```python
# ============================================
# 2. LOAD CSV AND FILTER APPLE ROWS
# ============================================

csv_path = "/Users/fm/Desktop/AI/plant_disease_multimodal_dataset.csv"
df = pd.read_csv(csv_path)

print("Columns in CSV:")
print(df.columns)

# Filter only Apple rows using 'Label'
apple_mask = df["Label"].astype(str).str.startswith("Apple___")
df_apple = df[apple_mask].copy()

print("\nOriginal shape:", df.shape)
print("Apple-only shape:", df_apple.shape)

print("\nUnique raw labels in Apple subset (Label):")
print(df_apple["Label"].unique())

print("\nMapped Label value counts:")
print(df_apple["Mapped Label"].value_counts())
```

```
Columns in CSV:
Index(['N', 'P', 'K', 'temperature', 'humidity', 'ph', 'rainfall',
       'Mapped Label', 'Image Path', 'Label'],
      dtype='object')

Original shape: (220000, 10)
Apple-only shape: (40000, 10)

Unique raw labels in Apple subset (Label):
['Apple___Apple_scab' 'Apple___Black_rot' 'Apple___Cedar_apple_rust'
 'Apple___healthy']

Mapped Label value counts:
Mapped Label
Apple___Apple_scab          10000
Apple___Black_rot           10000
Apple___Cedar_apple_rust    10000
Apple___healthy             10000
Name: count, dtype: int64
```

In [8]:
```python
# ============================================
# 3. SELECT STRUCTURED FEATURES + LABEL
# ============================================

feature_cols = ["N", "P", "K", "temperature", "humidity", "ph", "rainfall"]
label_col = "Mapped Label"

X = df_apple[feature_cols].values   # structured features
y = df_apple[label_col].values      # class label

print("X shape:", X.shape)
print("y shape:", y.shape)
print("Unique labels:", np.unique(y))
```

```
X shape: (40000, 7)
y shape: (40000,)
Unique labels: ['Apple___Apple_scab' 'Apple___Black_rot' 'Apple___Cedar_apple_rust'
 'Apple___healthy']
```

In [10]:
```python
# ============================================
# 4. LABEL ENCODING + TRAIN/VAL/TEST SPLIT
# ============================================

label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

num_classes = len(label_encoder.classes_)
print("Classes:", label_encoder.classes_)

# 70% train, 15% val, 15% test
X_train_temp, X_test, y_train_temp, y_test = train_test_split(
    X, y_encoded,
    test_size=0.15,
    random_state=42,
    stratify=y_encoded
)

val_size = 0.15 / 0.85  # so final val is 15% of total

X_train, X_val, y_train, y_val = train_test_split(
    X_train_temp, y_train_temp,
    test_size=val_size,
    random_state=42,
    stratify=y_train_temp
)

print("\nTrain:", X_train.shape, len(y_train))
print("Val:  ", X_val.shape,   len(y_val))
print("Test: ", X_test.shape,  len(y_test))
```

```
Classes: ['Apple___Apple_scab' 'Apple___Black_rot' 'Apple___Cedar_apple_rust'
 'Apple___healthy']

Train: (28000, 7) 28000
Val:   (6000, 7) 6000
Test:  (6000, 7) 6000
```

In [12]:

```python
# ============================================
# 5. STANDARD SCALER (FIT ONLY ON TRAIN)
# ============================================

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled   = scaler.transform(X_val)
X_test_scaled  = scaler.transform(X_test)

print("Scaled shapes:")
print("Train:", X_train_scaled.shape)
print("Val:  ", X_val_scaled.shape)
print("Test: ", X_test_scaled.shape)
```

```
Scaled shapes:
Train: (28000, 7)
Val:   (6000, 7)
Test:  (6000, 7)
```

In [14]:

```python
# ============================================
# 6. MLP MODEL (STRUCTURED-ONLY)
# ============================================

def build_mlp_model(input_dim, num_classes):
    inputs = layers.Input(shape=(input_dim,))
    x = layers.Dense(64, activation="relu")(inputs)
    x = layers.Dropout(0.3)(x)
    x = layers.Dense(32, activation="relu")(x)
    x = layers.Dropout(0.2)(x)
    outputs = layers.Dense(num_classes, activation="softmax")(x)

    model = models.Model(inputs, outputs, name="MLP_Structured_Only")
    model.compile(
        optimizer=tf.keras.optimizers.Adam(1e-3),
        loss="sparse_categorical_crossentropy",
        metrics=["accuracy"]
    )
    return model

input_dim = X_train_scaled.shape[1]
mlp_model = build_mlp_model(input_dim, num_classes)
mlp_model.summary()
```

Model: "MLP_Structured_Only"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 7)	0
dense (Dense)	(None, 64)	512
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 32)	2,080
dropout_1 (Dropout)	(None, 32)	0
dense_2 (Dense)	(None, 4)	132

Total params: 2,724 (10.64 KB)
Trainable params: 2,724 (10.64 KB)
Non-trainable params: 0 (0.00 B)

In [21]:

```python
# ============================================
# 7. TRAIN MLP
# ============================================

early_stop = callbacks.EarlyStopping(
    monitor="val_loss",
    patience=10,
    restore_best_weights=True
)

history_mlp = mlp_model.fit(
    X_train_scaled, y_train,
    validation_data=(X_val_scaled, y_val),
    epochs=100,
    batch_size=32,
    callbacks=[early_stop],
    verbose=1
)
```

```
Epoch 1/100
875/875 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 1.0000 - loss: 1.2013e-06 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 2/100
875/875 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 1.0000 - loss: 2.2634e-06 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 3/100
875/875 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 1.0000 - loss: 2.4402e-07 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 4/100
875/875 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 1.0000 - loss: 2.6095e-07 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 5/100
875/875 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 1.0000 - loss: 3.4836e-07 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 6/100
875/875 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 1.0000 - loss: 4.8506e-07 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 7/100
875/875 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 1.0000 - loss: 2.2674e-07 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 8/100
875/875 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 1.0000 - loss: 9.8372e-08 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 9/100
875/875 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 1.0000 - loss: 5.2319e-06 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 10/100
875/875 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 1.0000 - loss: 3.3196e-07 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 11/100
875/875 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 1.0000 - loss: 3.7999e-06 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
```

In [23]:

```python
# ============================================
# 8. EVALUATE MLP (FULL METRICS)
# ============================================
```

```python
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report,
    confusion_matrix
)

# Predictions
y_proba_mlp = mlp_model.predict(X_test_scaled)
y_pred_mlp  = y_proba_mlp.argmax(axis=1)

# Basic metrics
acc_mlp  = accuracy_score(y_test, y_pred_mlp)
prec_mlp = precision_score(y_test, y_pred_mlp, average='macro', zero_division=0)
rec_mlp  = recall_score(y_test, y_pred_mlp, average='macro', zero_division=0)
f1_mlp   = f1_score(y_test, y_pred_mlp, average='macro', zero_division=0)

# Weighted metrics (for reporting)
prec_w = precision_score(y_test, y_pred_mlp, average='weighted', zero_division=0)
rec_w  = recall_score(y_test, y_pred_mlp, average='weighted', zero_division=0)
f1_w   = f1_score(y_test, y_pred_mlp, average='weighted', zero_division=0)

# Print results
print("============== MLP METRICS ==============")
print("Accuracy:        ", acc_mlp)
print("Precision:       ", prec_mlp)
print("Recall:          ", rec_mlp)
print("F1-score:        ", f1_mlp)

print("\nWeighted Precision:", prec_w)
print("Weighted Recall:   ", rec_w)
print("Weighted F1-score: ", f1_w)

print("\n============== CLASSIFICATION REPORT ==============")
print(classification_report(y_test, y_pred_mlp, target_names=label_encoder.classes_))

print("============== CONFUSION MATRIX ==============")
print(confusion_matrix(y_test, y_pred_mlp))
```

```
188/188 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
============== MLP METRICS ==============
Accuracy:         1.0
Precision:        1.0
Recall:           1.0
F1-score:         1.0

Weighted Precision: 1.0
Weighted Recall:    1.0
Weighted F1-score:  1.0

============== CLASSIFICATION REPORT ==============
                           precision    recall  f1-score   support

      Apple___Apple_scab       1.00      1.00      1.00      1500
       Apple___Black_rot       1.00      1.00      1.00      1500
Apple___Cedar_apple_rust       1.00      1.00      1.00      1500
         Apple___healthy       1.00      1.00      1.00      1500

                accuracy                           1.00      6000
               macro avg       1.00      1.00      1.00      6000
            weighted avg       1.00      1.00      1.00      6000

============== CONFUSION MATRIX ==============
[[1500    0    0    0]
 [   0 1500    0    0]
 [   0    0 1500    0]
 [   0    0    0 1500]]
```

In [ ]:

In [ ]:

In [ ]:

```python
# Decision Tree
```

In [25]:

```python
# ==========================================
# 9. DECISION TREE
# ==========================================

dt_clf = DecisionTreeClassifier(random_state=42)
dt_clf.fit(X_train_scaled, y_train)

y_pred_dt = dt_clf.predict(X_test_scaled)

acc_dt = accuracy_score(y_test, y_pred_dt)
print("Decision Tree - Accuracy:", acc_dt)

print("\nDecision Tree - Classification Report:")
print(classification_report(y_test, y_pred_dt, target_names=label_encoder.classes_))

print("Decision Tree - Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_dt))
```

```
Decision Tree - Accuracy: 1.0

Decision Tree - Classification Report:
                           precision    recall  f1-score   support

      Apple___Apple_scab       1.00      1.00      1.00      1500
       Apple___Black_rot       1.00      1.00      1.00      1500
Apple___Cedar_apple_rust       1.00      1.00      1.00      1500
         Apple___healthy       1.00      1.00      1.00      1500

                accuracy                           1.00      6000
               macro avg       1.00      1.00      1.00      6000
            weighted avg       1.00      1.00      1.00      6000

Decision Tree - Confusion Matrix:
[[1500    0    0    0]
 [   0 1500    0    0]
 [   0    0 1500    0]
 [   0    0    0 1500]]
```

In [ ]:

In [ ]:

```python
# RANDOM FOREST
```

In [27]:
```python
# ============================================
# 10. RANDOM FOREST
# ============================================

rf_clf = RandomForestClassifier(
    n_estimators=200,
    random_state=42,
    n_jobs=-1
)
rf_clf.fit(X_train_scaled, y_train)

y_pred_rf = rf_clf.predict(X_test_scaled)

acc_rf = accuracy_score(y_test, y_pred_rf)
print("Random Forest - Accuracy:", acc_rf)

print("\nRandom Forest - Classification Report:")
print(classification_report(y_test, y_pred_rf, target_names=label_encoder.classes_))

print("Random Forest - Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_rf))
```

```
Random Forest - Accuracy: 1.0

Random Forest - Classification Report:
                          precision    recall  f1-score   support

      Apple___Apple_scab       1.00      1.00      1.00      1500
       Apple___Black_rot       1.00      1.00      1.00      1500
Apple___Cedar_apple_rust       1.00      1.00      1.00      1500
         Apple___healthy       1.00      1.00      1.00      1500

               accuracy                           1.00      6000
              macro avg       1.00      1.00      1.00      6000
           weighted avg       1.00      1.00      1.00      6000

Random Forest - Confusion Matrix:
[[1500    0    0    0]
 [   0 1500    0    0]
 [   0    0 1500    0]
 [   0    0    0 1500]]
```

In [ ]:

In [29]:
```python
# ============================================
# 11. LOGISTIC REGRESSION (BASELINE)
# ============================================

log_reg = LogisticRegression(
    max_iter=2000,
    multi_class="multinomial",
    n_jobs=-1
)
log_reg.fit(X_train_scaled, y_train)

y_pred_lr = log_reg.predict(X_test_scaled)

acc_lr = accuracy_score(y_test, y_pred_lr)
print("Logistic Regression - Accuracy:", acc_lr)

print("\nLogistic Regression - Classification Report:")
print(classification_report(y_test, y_pred_lr, target_names=label_encoder.classes_))

print("Logistic Regression - Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_lr))
```

```
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_logistic.py:1247: FutureWarning: 'multi_class' was deprecated in version 1.5 and will be removed in 1.7. From then on, it will always use 'multinomial'. Leave it to its default value to avoid this warning.
  warnings.warn(
Logistic Regression - Accuracy: 1.0

Logistic Regression - Classification Report:
                          precision    recall  f1-score   support

      Apple___Apple_scab       1.00      1.00      1.00      1500
       Apple___Black_rot       1.00      1.00      1.00      1500
Apple___Cedar_apple_rust       1.00      1.00      1.00      1500
         Apple___healthy       1.00      1.00      1.00      1500

               accuracy                           1.00      6000
              macro avg       1.00      1.00      1.00      6000
           weighted avg       1.00      1.00      1.00      6000

Logistic Regression - Confusion Matrix:
[[1500    0    0    0]
 [   0 1500    0    0]
 [   0    0 1500    0]
 [   0    0    0 1500]]
```

In [ ]:

In [4]:
```python
import os
import shutil
from glob import glob
import random

import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input as mobilenet_preprocess

import matplotlib.pyplot as plt

# ====== 1. Paths ======
base_dir = "/Users/fm/Desktop/AI901"
color_dir = os.path.join(base_dir, "color")

# Directory for Apple-only image dataset with splits
apple_base = os.path.join(base_dir, "apple_image_only")
os.makedirs(apple_base, exist_ok=True)

# Apple classes
apple_classes = [
    "Apple___Apple_scab",
    "Apple___Black_rot",
    "Apple___Cedar_apple_rust",
    "Apple___healthy"
]

splits = ["train", "val", "test"]

for split in splits:
    for cls in apple_classes:
        os.makedirs(os.path.join(apple_base, split, cls), exist_ok=True)

print("✅ Folder structure for Apple subset created at:", apple_base)
```

```
✅ Folder structure for Apple subset created at: /Users/fm/Desktop/AI901/apple_image_only
```

In [6]:
```python
# ====== 2. Collect all Apple images from color folder ======
all_images = []

for cls in apple_classes:
    class_dir = os.path.join(color_dir, cls)
    pattern_jpg = os.path.join(class_dir, "*.JPG")
    pattern_jpeg = os.path.join(class_dir, "*.JPEG")
    pattern_png = os.path.join(class_dir, "*.PNG")

    images = glob(pattern_jpg) + glob(pattern_jpeg) + glob(pattern_png)
    print(f"{cls}: found {len(images)} images")
    for img_path in images:
        all_images.append((img_path, cls))

print("\nTotal Apple images:", len(all_images))

# Convert to lists for splitting
image_paths = [p for p, c in all_images]
labels = [c for p, c in all_images]

# ====== 3. Train/Val/Test split (70/15/15) ======
train_paths, temp_paths, train_labels, temp_labels = train_test_split(
    image_paths, labels, test_size=0.30, stratify=labels, random_state=42
)

val_paths, test_paths, val_labels, test_labels = train_test_split(
    temp_paths, temp_labels, test_size=0.50, stratify=temp_labels, random_state=42
)

print(f"\nTrain size: {len(train_paths)}")
print(f"Val size:   {len(val_paths)}")
print(f"Test size:  {len(test_paths)}")

# ====== 4. Helper to copy files ======
def copy_files(paths, labels, split):
    for img_path, label in zip(paths, labels):
        file_name = os.path.basename(img_path)
        dst_dir = os.path.join(apple_base, split, label)
        dst_path = os.path.join(dst_dir, file_name)
        if not os.path.exists(dst_path):
            shutil.copy2(img_path, dst_path)

# Copy train/val/test
copy_files(train_paths, train_labels, "train")
copy_files(val_paths, val_labels, "val")
copy_files(test_paths, test_labels, "test")

print("\n✅ Images copied into train/val/test for Apple only.")
```

```
Apple___Apple_scab: found 630 images
Apple___Black_rot: found 621 images
Apple___Cedar_apple_rust: found 275 images
Apple___healthy: found 1645 images

Total Apple images: 3171

Train size: 2219
Val size:   476
Test size:  476

✅ Images copied into train/val/test for Apple only.
```

In [8]:
```python
# ====== 5. Check train class counts BEFORE oversampling ======
train_dir = os.path.join(apple_base, "train")
```

```python
class_counts = {}
for cls in apple_classes:
    folder = os.path.join(train_dir, cls)
    files = [
        f for f in os.listdir(folder)
        if f.lower().endswith((".jpg", ".jpeg", ".png"))
    ]
    class_counts[cls] = len(files)

print("Train class counts BEFORE oversampling:")
for cls, cnt in class_counts.items():
    print(f"{cls}: {cnt}")

max_count = max(class_counts.values())
print("\nTarget (max) count for each class:", max_count)

# ====== 6. Oversampling (duplicate images of minority classes) ======
for cls, count in class_counts.items():
    folder = os.path.join(train_dir, cls)
    files = [
        f for f in os.listdir(folder)
        if f.lower().endswith((".jpg", ".jpeg", ".png"))
    ]

    if count == 0:
        continue

    num_to_add = max_count - count
    print(f"\nClass {cls}: current = {count}, will add = {num_to_add}")

    for i in range(num_to_add):
        src_name = random.choice(files)
        src_path = os.path.join(folder, src_name)

        name, ext = os.path.splitext(src_name)
        new_name = f"{name}_dup{i}{ext}"
        dst_path = os.path.join(folder, new_name)

        if not os.path.exists(dst_path):
            shutil.copy2(src_path, dst_path)

print("\n✅ Oversampling done for train set.")

# ====== 7. Check counts AFTER oversampling ======
class_counts_after = {}
for cls in apple_classes:
    folder = os.path.join(train_dir, cls)
    files = [
        f for f in os.listdir(folder)
        if f.lower().endswith((".jpg", ".jpeg", ".png"))
    ]
    class_counts_after[cls] = len(files)

print("\nTrain class counts AFTER oversampling:")
for cls, cnt in class_counts_after.items():
    print(f"{cls}: {cnt}")
```

```
Train class counts BEFORE oversampling:
Apple___Apple_scab: 441
Apple___Black_rot: 435
Apple___Cedar_apple_rust: 192
Apple___healthy: 1151

Target (max) count for each class: 1151

Class Apple___Apple_scab: current = 441, will add = 710

Class Apple___Black_rot: current = 435, will add = 716

Class Apple___Cedar_apple_rust: current = 192, will add = 959

Class Apple___healthy: current = 1151, will add = 0

✅ Oversampling done for train set.

Train class counts AFTER oversampling:
Apple___Apple_scab: 1151
Apple___Black_rot: 1151
Apple___Cedar_apple_rust: 1151
Apple___healthy: 1151
```

In [10]:

```python
# ====== 8. ImageDataGenerators and generators ======
img_size = (224, 224)
batch_size = 32

train_datagen = ImageDataGenerator(
    # No rescale here (MobileNetV2 preprocess will handle scaling)
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    zoom_range=0.1
)

val_test_datagen = ImageDataGenerator(
    # No rescale here
)

val_dir  = os.path.join(apple_base, "val")
test_dir = os.path.join(apple_base, "test")

train_gen = train_datagen.flow_from_directory(
    train_dir,
    target_size=img_size,
    batch_size=batch_size,
    class_mode="categorical"
)

val_gen = val_test_datagen.flow_from_directory(
    val_dir,
    target_size=img_size,
    batch_size=batch_size,
    class_mode="categorical"
)
```

```
test_gen = val_test_datagen.flow_from_directory(
    test_dir,
    target_size=img_size,
    batch_size=batch_size,
    class_mode="categorical",
    shuffle=False
)

print("\nClass indices:", train_gen.class_indices)
```

```
Found 4604 images belonging to 4 classes.
Found 476 images belonging to 4 classes.
Found 476 images belonging to 4 classes.

Class indices: {'Apple___Apple_scab': 0, 'Apple___Black_rot': 1, 'Apple___Cedar_apple_rust': 2, 'Apple___healthy': 3}
```

In [12]:

```
from tensorflow.keras import layers, models
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input as mobilenet_preprocess

num_classes = train_gen.num_classes
img_size = (224, 224)

def build_mobilenet_finetune(num_classes, image_size=(224, 224)):
    base_model = MobileNetV2(
        include_top=False,
        weights="imagenet",
        input_shape=(image_size[0], image_size[1], 3)
    )

    base_model.trainable = True
    for layer in base_model.layers[:-50]:
        layer.trainable = False

    inputs = layers.Input(shape=(image_size[0], image_size[1], 3))
    x = mobilenet_preprocess(inputs)
    x = base_model(x, training=True)
    x = layers.GlobalAveragePooling2D()(x)
    x = layers.Dropout(0.4)(x)
    outputs = layers.Dense(num_classes, activation="softmax")(x)

    model = models.Model(inputs, outputs)
    model.compile(
        optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
        loss="categorical_crossentropy",
        metrics=["accuracy"]
    )
    return model

mobilenet_model = build_mobilenet_finetune(num_classes, img_size)
mobilenet_model.summary()
```

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 224, 224, 3)	0
true_divide (TrueDivide)	(None, 224, 224, 3)	0
subtract (Subtract)	(None, 224, 224, 3)	0
mobilenetv2_1.00_224 (Functional)	(None, 7, 7, 1280)	2,257,984
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1280)	0
dropout (Dropout)	(None, 1280)	0
dense (Dense)	(None, 4)	5,124

Total params: 2,263,108 (8.63 MB)
Trainable params: 1,860,228 (7.10 MB)
Non-trainable params: 402,880 (1.54 MB)

In [14]:

```
epochs = 10

history = mobilenet_model.fit(
    train_gen,
    epochs=epochs,
    validation_data=val_gen
)
```

```
/opt/anaconda3/lib/python3.12/site-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init
__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will be ig
nored.
  self._warn_if_super_not_called()
Epoch 1/10
144/144 ━━━━━━━━━━━━━━━━━━━━ 175s 1s/step - accuracy: 0.9331 - loss: 0.1887 - val_accuracy: 0.9601 - val_loss: 0.0957
Epoch 2/10
144/144 ━━━━━━━━━━━━━━━━━━━━ 157s 1s/step - accuracy: 0.9917 - loss: 0.0259 - val_accuracy: 0.9664 - val_loss: 0.0895
Epoch 3/10
144/144 ━━━━━━━━━━━━━━━━━━━━ 155s 1s/step - accuracy: 0.9961 - loss: 0.0160 - val_accuracy: 0.9580 - val_loss: 0.0946
Epoch 4/10
144/144 ━━━━━━━━━━━━━━━━━━━━ 156s 1s/step - accuracy: 0.9965 - loss: 0.0102 - val_accuracy: 0.9979 - val_loss: 0.0080
Epoch 5/10
144/144 ━━━━━━━━━━━━━━━━━━━━ 150s 1s/step - accuracy: 0.9978 - loss: 0.0082 - val_accuracy: 0.9916 - val_loss: 0.0210
Epoch 6/10
144/144 ━━━━━━━━━━━━━━━━━━━━ 151s 1s/step - accuracy: 0.9987 - loss: 0.0064 - val_accuracy: 0.9874 - val_loss: 0.0329
Epoch 7/10
144/144 ━━━━━━━━━━━━━━━━━━━━ 148s 1s/step - accuracy: 0.9976 - loss: 0.0070 - val_accuracy: 0.9811 - val_loss: 0.0510
Epoch 8/10
144/144 ━━━━━━━━━━━━━━━━━━━━ 142s 985ms/step - accuracy: 0.9989 - loss: 0.0034 - val_accuracy: 1.0000 - val_loss: 0.0018
Epoch 9/10
144/144 ━━━━━━━━━━━━━━━━━━━━ 155s 1s/step - accuracy: 0.9985 - loss: 0.0034 - val_accuracy: 0.9853 - val_loss: 0.0497
Epoch 10/10
144/144 ━━━━━━━━━━━━━━━━━━━━ 151s 1s/step - accuracy: 0.9993 - loss: 0.0024 - val_accuracy: 0.9979 - val_loss: 0.0030
```

In [18]:

```
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

```python
y_prob = mobilenet_model.predict(test_gen)
y_pred = np.argmax(y_prob, axis=1)
y_true = test_gen.classes
class_names = list(test_gen.class_indices.keys())

print("\nClassification Report:\n")
print(classification_report(y_true, y_pred, target_names=class_names))


acc = accuracy_score(y_true, y_pred)
precision = precision_score(y_true, y_pred, average='macro')
recall = recall_score(y_true, y_pred, average='macro')
f1 = f1_score(y_true, y_pred, average='macro')

print("Accuracy:", acc)
print("Precision (macro):", precision)
print("Recall (macro):", recall)
print("F1-score (macro):", f1)


cm = confusion_matrix(y_true, y_pred)
print("\nConfusion Matrix:\n", cm)

unique, counts = np.unique(y_pred, return_counts=True)
print("\nPredicted class distribution:")
for u, c in zip(unique, counts):
    print(f"{u} -> {c}")

print("\nClass indices:", train_gen.class_indices)
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 10s 644ms/step

Classification Report:

                          precision    recall  f1-score   support

      Apple___Apple_scab       1.00      0.98      0.99        95
       Apple___Black_rot       1.00      0.99      0.99        93
Apple___Cedar_apple_rust       1.00      1.00      1.00        41
         Apple___healthy       0.99      1.00      0.99       247

                accuracy                           0.99       476
               macro avg       1.00      0.99      0.99       476
            weighted avg       0.99      0.99      0.99       476

Accuracy: 0.9936974789915967
Precision (macro): 0.997
Recall (macro): 0.9920486700622524
F1-score (macro): 0.9944800198546079

Confusion Matrix:
 [[ 93   0   0   2]
 [  0  92   0   1]
 [  0   0  41   0]
 [  0   0   0 247]]

Predicted class distribution:
0 -> 93
1 -> 92
2 -> 41
3 -> 250

Class indices: {'Apple___Apple_scab': 0, 'Apple___Black_rot': 1, 'Apple___Cedar_apple_rust': 2, 'Apple___healthy': 3}
```

In [ ]:

In [ ]:

In [20]:

```python
#InceptionV3

from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.applications.inception_v3 import preprocess_input as inception_preprocess
from tensorflow.keras import layers, models
import tensorflow as tf

img_size = (224, 224)
num_classes = train_gen.num_classes

def build_inception_model(num_classes, image_size=(224, 224)):
    base_model = InceptionV3(
        include_top=False,
        weights="imagenet",
        input_shape=(image_size[0], image_size[1], 3)
    )

    # Fine-tune
    base_model.trainable = True
    for layer in base_model.layers[:-50]:
        layer.trainable = False

    inputs = layers.Input(shape=(image_size[0], image_size[1], 3))
    x = inception_preprocess(inputs)
    x = base_model(x, training=True)
    x = layers.GlobalAveragePooling2D()(x)
    x = layers.Dropout(0.4)(x)
    outputs = layers.Dense(num_classes, activation="softmax")(x)

    model = models.Model(inputs, outputs)

    model.compile(
        optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
        loss="categorical_crossentropy",
        metrics=["accuracy"]
    )
    return model

inception_model = build_inception_model(num_classes, img_size)
inception_model.summary()
```

```
Model: "functional_1"
```

Layer (type)	Output Shape	Param #
input_layer_3 (InputLayer)	(None, 224, 224, 3)	0
true_divide_1 (TrueDivide)	(None, 224, 224, 3)	0
subtract_1 (Subtract)	(None, 224, 224, 3)	0
inception_v3 (Functional)	(None, 5, 5, 2048)	21,802,784
global_average_pooling2d_1 (GlobalAveragePooling2D)	(None, 2048)	0
dropout_1 (Dropout)	(None, 2048)	0
dense_1 (Dense)	(None, 4)	8,196

Total params: 21,810,980 (83.20 MB)
Trainable params: 7,181,508 (27.40 MB)
Non-trainable params: 14,629,472 (55.81 MB)

In [22]:
```
epochs = 10

history_inception = inception_model.fit(
    train_gen,
    epochs=epochs,
    validation_data=val_gen
)
```

```
Epoch 1/10
144/144 ━━━━━━━━━━━━━━━━━━━━ 221s 1s/step - accuracy: 0.9153 - loss: 0.2452 - val_accuracy: 0.9748 - val_loss: 0.0679
Epoch 2/10
144/144 ━━━━━━━━━━━━━━━━━━━━ 195s 1s/step - accuracy: 0.9833 - loss: 0.0522 - val_accuracy: 0.9916 - val_loss: 0.0241
Epoch 3/10
144/144 ━━━━━━━━━━━━━━━━━━━━ 205s 1s/step - accuracy: 0.9896 - loss: 0.0357 - val_accuracy: 0.9895 - val_loss: 0.0316
Epoch 4/10
144/144 ━━━━━━━━━━━━━━━━━━━━ 204s 1s/step - accuracy: 0.9896 - loss: 0.0298 - val_accuracy: 0.9937 - val_loss: 0.0178
Epoch 5/10
144/144 ━━━━━━━━━━━━━━━━━━━━ 195s 1s/step - accuracy: 0.9926 - loss: 0.0217 - val_accuracy: 0.9895 - val_loss: 0.0178
Epoch 6/10
144/144 ━━━━━━━━━━━━━━━━━━━━ 191s 1s/step - accuracy: 0.9954 - loss: 0.0151 - val_accuracy: 0.9853 - val_loss: 0.0475
Epoch 7/10
144/144 ━━━━━━━━━━━━━━━━━━━━ 192s 1s/step - accuracy: 0.9939 - loss: 0.0192 - val_accuracy: 0.9895 - val_loss: 0.0199
Epoch 8/10
144/144 ━━━━━━━━━━━━━━━━━━━━ 220s 2s/step - accuracy: 0.9985 - loss: 0.0064 - val_accuracy: 0.9979 - val_loss: 0.0083
Epoch 9/10
144/144 ━━━━━━━━━━━━━━━━━━━━ 231s 2s/step - accuracy: 0.9976 - loss: 0.0084 - val_accuracy: 0.9958 - val_loss: 0.0168
Epoch 10/10
144/144 ━━━━━━━━━━━━━━━━━━━━ 215s 1s/step - accuracy: 0.9965 - loss: 0.0086 - val_accuracy: 0.9979 - val_loss: 0.0090
```

In [24]:
```
from sklearn.metrics import (
    classification_report,
    confusion_matrix,
    accuracy_score,
    precision_score,
    recall_score,
    f1_score
)
import numpy as np

# 1️⃣ Predictions
y_prob_inc = inception_model.predict(test_gen)
y_pred_inc = np.argmax(y_prob_inc, axis=1)
y_true = test_gen.classes
class_names = list(test_gen.class_indices.keys())

# 2️⃣ Classification Report
print("\nClassification Report (InceptionV3):\n")
print(classification_report(y_true, y_pred_inc, target_names=class_names))

# 3️⃣ Confusion Matrix
cm_inc = confusion_matrix(y_true, y_pred_inc)
print("\nConfusion Matrix (InceptionV3):\n", cm_inc)

# 4️⃣ Accuracy + Precision + Recall + F1
acc_inc = accuracy_score(y_true, y_pred_inc)
precision_inc = precision_score(y_true, y_pred_inc, average='macro')
recall_inc = recall_score(y_true, y_pred_inc, average='macro')
f1_inc = f1_score(y_true, y_pred_inc, average='macro')

print("\n📌 Metrics (Macro Average):")
print("Accuracy:", acc_inc)
print("Precision:", precision_inc)
print("Recall:", recall_inc)
print("F1-score:", f1_inc)


unique, counts = np.unique(y_pred_inc, return_counts=True)
print("\nPredicted class distribution:")
for u, c in zip(unique, counts):
    print(f"{u} -> {c}")

print("\nClass indices:", test_gen.class_indices)
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 18s 1s/step

Classification Report (InceptionV3):

                          precision    recall  f1-score   support

      Apple___Apple_scab       0.99      1.00      0.99        95
       Apple___Black_rot       1.00      1.00      1.00        93
Apple___Cedar_apple_rust       1.00      1.00      1.00        41
         Apple___healthy       1.00      1.00      1.00       247

                accuracy                           1.00       476
               macro avg       1.00      1.00      1.00       476
            weighted avg       1.00      1.00      1.00       476


Confusion Matrix (InceptionV3):
 [[ 95   0   0   0]
 [  0  93   0   0]
 [  0   0  41   0]
 [  1   0   0 246]]
```

```
📌 Metrics (Macro Average):
Accuracy: 0.9978991596638656
Precision: 0.9973958333333334
Recall: 0.9989878542510121
F1-score: 0.998184000084959

Predicted class distribution:
0 -> 96
1 -> 93
2 -> 41
3 -> 246

Class indices: {'Apple___Apple_scab': 0, 'Apple___Black_rot': 1, 'Apple___Cedar_apple_rust': 2, 'Apple___healthy': 3}
```

In [26]:

```python
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.applications.resnet50 import preprocess_input as resnet_preprocess
from tensorflow.keras import layers, models
import tensorflow as tf

img_size = (224, 224)
num_classes = train_gen.num_classes

def build_resnet_model(num_classes, image_size=(224, 224)):
    # 1 Base ResNet50
    base_model = ResNet50(
        include_top=False,
        weights="imagenet",
        input_shape=(image_size[0], image_size[1], 3)
    )

    # 2 Fine-tune
    base_model.trainable = True
    for layer in base_model.layers[:-50]:
        layer.trainable = False

    inputs = layers.Input(shape=(image_size[0], image_size[1], 3))
    x = resnet_preprocess(inputs)
    x = base_model(x, training=True)
    x = layers.GlobalAveragePooling2D()(x)
    x = layers.Dropout(0.4)(x)
    outputs = layers.Dense(num_classes, activation="softmax")(x)

    model = models.Model(inputs, outputs)

    # 4 Compile
    model.compile(
        optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
        loss="categorical_crossentropy",
        metrics=["accuracy"]
    )

    return model

resnet_model = build_resnet_model(num_classes, img_size)
resnet_model.summary()
```

Model: "functional_2"

Layer (type)	Output Shape	Param #	Connected to
input_layer_5 (InputLayer)	(None, 224, 224, 3)	0	–
get_item (GetItem)	(None, 224, 224)	0	input_layer_5[0]…
get_item_1 (GetItem)	(None, 224, 224)	0	input_layer_5[0]…
get_item_2 (GetItem)	(None, 224, 224)	0	input_layer_5[0]…
stack (Stack)	(None, 224, 224, 3)	0	get_item[0][0], get_item_1[0][0], get_item_2[0][0]
add (Add)	(None, 224, 224, 3)	0	stack[0][0]
resnet50 (Functional)	(None, 7, 7, 2048)	23,587,712	add[0][0]
global_average_poo… (GlobalAveragePool…	(None, 2048)	0	resnet50[0][0]
dropout_2 (Dropout)	(None, 2048)	0	global_average_p…
dense_2 (Dense)	(None, 4)	8,196	dropout_2[0][0]

Total params: 23,595,908 (90.01 MB)
Trainable params: 16,958,724 (64.69 MB)
Non-trainable params: 6,637,184 (25.32 MB)

In [28]:

```python
epochs = 10

history_resnet = resnet_model.fit(
    train_gen,
    epochs=epochs,
    validation_data=val_gen
)
```

```
Epoch 1/10
144/144 ━━━━━━━━━━━━━━━━━━━━ 474s 3s/step - accuracy: 0.9572 - loss: 0.1165 - val_accuracy: 0.9853 - val_loss: 0.0381
Epoch 2/10
144/144 ━━━━━━━━━━━━━━━━━━━━ 453s 3s/step - accuracy: 0.9976 - loss: 0.0090 - val_accuracy: 1.0000 - val_loss: 0.0011
Epoch 3/10
144/144 ━━━━━━━━━━━━━━━━━━━━ 457s 3s/step - accuracy: 0.9957 - loss: 0.0119 - val_accuracy: 0.9916 - val_loss: 0.0161
Epoch 4/10
144/144 ━━━━━━━━━━━━━━━━━━━━ 421s 3s/step - accuracy: 0.9970 - loss: 0.0092 - val_accuracy: 0.9895 - val_loss: 0.0524
Epoch 5/10
144/144 ━━━━━━━━━━━━━━━━━━━━ 413s 3s/step - accuracy: 0.9976 - loss: 0.0086 - val_accuracy: 0.9916 - val_loss: 0.0230
Epoch 6/10
144/144 ━━━━━━━━━━━━━━━━━━━━ 453s 3s/step - accuracy: 0.9985 - loss: 0.0057 - val_accuracy: 0.9979 - val_loss: 0.0068
Epoch 7/10
144/144 ━━━━━━━━━━━━━━━━━━━━ 513s 4s/step - accuracy: 0.9991 - loss: 0.0039 - val_accuracy: 1.0000 - val_loss: 0.0010
Epoch 8/10
144/144 ━━━━━━━━━━━━━━━━━━━━ 418s 3s/step - accuracy: 0.9985 - loss: 0.0060 - val_accuracy: 1.0000 - val_loss: 0.0020
Epoch 9/10
144/144 ━━━━━━━━━━━━━━━━━━━━ 405s 3s/step - accuracy: 0.9987 - loss: 0.0028 - val_accuracy: 0.9958 - val_loss: 0.0170
Epoch 10/10
144/144 ━━━━━━━━━━━━━━━━━━━━ 417s 3s/step - accuracy: 0.9985 - loss: 0.0066 - val_accuracy: 1.0000 - val_loss: 0.0031
```

In [34]:

```python
import numpy as np
from sklearn.metrics import (
    classification_report,
    confusion_matrix,
    accuracy_score,
    precision_score,
    recall_score,
    f1_score
)

# 1️⃣ Predictions
y_prob_res = resnet_model.predict(test_gen)
y_pred_res = np.argmax(y_prob_res, axis=1)
y_true = test_gen.classes
class_names = list(test_gen.class_indices.keys())

# 2️⃣ Classification Report
print("\nClassification Report (ResNet50):\n")
print(classification_report(y_true, y_pred_res, target_names=class_names))

# 3️⃣ Confusion Matrix
cm_res = confusion_matrix(y_true, y_pred_res)
print("\nConfusion Matrix (ResNet50):\n", cm_res)

# 4️⃣ Accuracy + Precision + Recall + F1 (macro)
acc_res = accuracy_score(y_true, y_pred_res)
precision_res = precision_score(y_true, y_pred_res, average='macro')
recall_res = recall_score(y_true, y_pred_res, average='macro')
f1_res = f1_score(y_true, y_pred_res, average='macro')

print("\n📌 Metrics (ResNet50 - Macro Average):")
print("Accuracy:", acc_res)
print("Precision:", precision_res)
print("Recall:", recall_res)
print("F1-score:", f1_res)

# (اختياري) weighted avg لو حبيتي
precision_res_w = precision_score(y_true, y_pred_res, average='weighted')
recall_res_w = recall_score(y_true, y_pred_res, average='weighted')
f1_res_w = f1_score(y_true, y_pred_res, average='weighted')

print("\n📌 Metrics (ResNet50 - Weighted Average):")
print("Precision (weighted):", precision_res_w)
print("Recall (weighted):", recall_res_w)
print("F1-score (weighted):", f1_res_w)

unique, counts = np.unique(y_pred_res, return_counts=True)
print("\nPredicted class distribution (ResNet50):")
for u, c in zip(unique, counts):
    print(f"{u} -> {c}")

print("\nClass indices:", test_gen.class_indices)
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 20s 1s/step

Classification Report (ResNet50):

                          precision    recall  f1-score   support

      Apple___Apple_scab       1.00      0.98      0.99        95
       Apple___Black_rot       0.98      1.00      0.99        93
Apple___Cedar_apple_rust       1.00      1.00      1.00        41
         Apple___healthy       1.00      1.00      1.00       247

                accuracy                           1.00       476
               macro avg       0.99      0.99      0.99       476
            weighted avg       1.00      1.00      1.00       476


Confusion Matrix (ResNet50):
 [[ 93   2   0   0]
 [  0  93   0   0]
 [  0   0  41   0]
 [  0   0   0 247]]

📌 Metrics (ResNet50 - Macro Average):
Accuracy: 0.9957983193277311
Precision: 0.9947368421052631
Recall: 0.9947368421052631
F1-score: 0.9946808510638299

📌 Metrics (ResNet50 - Weighted Average):
Precision (weighted): 0.9958867757629368
Recall (weighted): 0.9957983193277311
F1-score (weighted): 0.9957983193277311

Predicted class distribution (ResNet50):
0 -> 93
1 -> 95
2 -> 41
3 -> 247

Class indices: {'Apple___Apple_scab': 0, 'Apple___Black_rot': 1, 'Apple___Cedar_apple_rust': 2, 'Apple___healthy': 3}
```

In [ ]: