In [1]:
```python
# ✓ Setup: Point to data folder in TeamX/src/data
from pathlib import Path
import os

# Get the notebook's actual directory location
notebook_file = Path(__file__) if '__file__' in dir() else Path.cwd() / 'Image_Clas

# For Jupyter: Get parent directory of notebook
# Notebook location: .../TeamX/src/Image_Classifier_Training.ipynb
# Expected data location: .../TeamX/src/data/

# Try to locate notebook by searching for it in common paths
possible_paths = [
    Path(r"c:\Users\skido\OneDrive\Desktop\NUS-ISS\Sem 2\SA4110 Machine Learning Ap
    Path.cwd(),
    Path.cwd() / 'src'
]

notebook_dir = None
for path in possible_paths:
    if (path / 'Image_Classifier_Training.ipynb').exists() or (path / 'data').exist
        notebook_dir = path
        break

if notebook_dir is None:
    # Fallback to absolute path
    notebook_dir = Path(r"c:\Users\skido\OneDrive\Desktop\NUS-ISS\Sem 2\SA4110 Mach

data_dir = notebook_dir / 'data'

print(f"✓ Notebook directory: {notebook_dir}")
print(f"✓ Data folder: {data_dir}")
print(f"✓ Data exists: {data_dir.exists()}")

if data_dir.exists():
    train_path = data_dir / 'train'
    test_path = data_dir / 'test'
    print(f"  - train/ exists: {train_path.exists()}")
    print(f"  - test/ exists: {test_path.exists()}")
    if train_path.exists():
        train_images = len(list(train_path.glob('*.jpg')))
        print(f"  - train images: {train_images}")
    if test_path.exists():
        test_images = len(list(test_path.glob('*.jpg')))
        print(f"  - test images: {test_images}")
```

```
✓ Notebook directory: c:\Users\skido\OneDrive\Desktop\NUS-ISS\Sem 2\SA4110 Machine
Learning Application Development [2510]\ML-CA-Image-Classifier-V1.0\ML-CA-Image-Clas
sifier-V1.0\TeamX\src
✓ Data folder: c:\Users\skido\OneDrive\Desktop\NUS-ISS\Sem 2\SA4110 Machine Learnin
g Application Development [2510]\ML-CA-Image-Classifier-V1.0\ML-CA-Image-Classifier-
V1.0\TeamX\src\data
✓ Data exists: True
  - train/ exists: True
  - test/ exists: True
  - train images: 240
  - test images: 60
```

Image Classifier Training Pipeline

CNN-based Fruit Classifier with Comprehensive Improvements

This notebook demonstrates a complete machine learning pipeline addressing all project requirements:

Project Requirement 1: Create a CNN model to classify 4 fruit types

- ✅ Uses MobileNetV2 (CNN-based transfer learning)
- ✅ Classifies: Apple, Orange, Banana, Mixed

Project Requirement 2-3: Use Train.zip and Test.zip datasets

- ✅ Training: 240 images from data/train/
- ✅ Testing: 60 images from data/test/

Project Requirement 4: Document experiments and results

- ✅ Comprehensive experiment documentation with timestamps
- ✅ Performance metrics, plots, and analysis

Project Requirement 5: Apply improvement techniques

- ✅ **Class Balancing**: Balanced weights for imbalanced categories
- ✅ **Mislabel Detection**: Identify and report suspicious/mislabeled images
- ✅ **Data Augmentation**: Rotation, zoom, brightness, shifts, flips

Project Requirement 6: Generate explanatory plots

- ✅ Training history (accuracy & loss curves)
- ✅ Confusion matrix for error analysis
- ✅ Per-class accuracy breakdown

Step 1: Import Required Libraries

In [2]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization, GlobalAvera
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheck
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import json
import numpy as np
```

Step 2: Create Data Augmentation Generators

The "Confusion" Generator creates new variations of your training photos on the fly to help the model learn better.

In [3]:
```
# --- 1. DATA AUGMENTATION (The "Confusion" Generator) ---
# This creates new variations of your photos on the fly.
# Support train/validation split via subset parameter
train_datagen = ImageDataGenerator(
    rescale=1./255,                    # Normalize pixel values
    rotation_range=40,                 # Tilt photo up to 40 degrees
    width_shift_range=0.2,             # Shift left/right
    height_shift_range=0.2,            # Shift up/down
    shear_range=0.2,                   # Distort shape (shear)
    zoom_range=0.2,                    # Zoom in/out
    horizontal_flip=True,              # Mirror image
    brightness_range=[0.8, 1.2],       # Simulate different lighting
    channel_shift_range=20.0,          # Slight color changes (simulates background
    fill_mode='nearest',
    validation_split=0.2               # Reserve 20% of training data for validatio
)

# Test data should NOT be augmented, only scaled.
# Test data will NOT be used during training - only for final evaluation
test_datagen = ImageDataGenerator(rescale=1./255)

print("✓ Data augmentation generators created!")
print("\nAugmentation parameters:")
print("  - Rotation: ±40°")
print("  - Shift: ±20% (width & height)")
print("  - Zoom: ±20%")
print("  - Brightness: 0.8 - 1.2x")
print("  - Horizontal flip: Yes")
```

```
✓ Data augmentation generators created!

Augmentation parameters:
  - Rotation: ±40°
  - Shift: ±20% (width & height)
  - Zoom: ±20%
  - Brightness: 0.8 - 1.2x
  - Horizontal flip: Yes
```

Step 3: Load Data Generators

Load training and test data from directory structure with augmentation applied.

In [4]:
```
# Data is stored flat with class prefixes (e.g., apple_1.jpg, banana_1.jpg)
# Organize into class subdirectories for flow_from_directory()

from pathlib import Path
import shutil

# Use notebook_dir from setup cell, then navigate to data subfolder
notebook_src_dir = notebook_dir  # This is .../TeamX/src
data_dir = notebook_src_dir / 'data'  # This is .../TeamX/src/data
train_path = data_dir / 'train'
test_path = data_dir / 'test'

print(f"Train path: {train_path}")
print(f"Train exists: {train_path.exists()}")
print(f"Test exists: {test_path.exists()}")

if train_path.exists():
    train_images = list(train_path.glob('*.jpg'))
```

```
    print(f"Train images found: {len(train_images)}")

# Create organized subdirectory structure
train_organized = train_path / 'organized'
test_organized = test_path / 'organized'

if not (train_organized / 'apple').exists():
    print("\n📁 Organizing images into class subdirectories...")

    classes = ['apple', 'banana', 'orange', 'mixed']

    # Organize training data
    print("\nOrganizing training data...")
    for cls in classes:
        (train_organized / cls).mkdir(parents=True, exist_ok=True)
        cls_files = list(train_path.glob(f'{cls}_*.jpg'))
        for file in cls_files:
            shutil.copy2(file, train_organized / cls / file.name)
        print(f"  ✓ {cls}: {len(cls_files)} images")

    # Organize test data
    print("\nOrganizing test data...")
    for cls in classes:
        (test_organized / cls).mkdir(parents=True, exist_ok=True)
        cls_files = list(test_path.glob(f'{cls}_*.jpg'))
        for file in cls_files:
            shutil.copy2(file, test_organized / cls / file.name)
        print(f"  ✓ {cls}: {len(cls_files)} images")

    print("\n✓ Data organized successfully!")
else:
    print("\n✓ Data already organized!")

# Load from organized structure with 150x150 (matches best model)
# IMPORTANT: Split training data into train/validation (80/20)
# Test data kept completely separate for final evaluation ONLY
print("\n" + "="*70)
print("Loading Data Generators (150x150 - matches best model)...")
print("="*70 + "\n")

# Create train/validation split from training data ONLY
# This keeps test data completely untouched until final evaluation
print("📊 Data Split Strategy:")
print("   - Training data (80%): Used for model training")
print("   - Validation data (20%): Used to monitor overfitting during training")
print("   - Test data: Kept completely separate for final evaluation ONLY")
print()

train_generator = train_datagen.flow_from_directory(
    str(train_organized),
    target_size=(150, 150),
    batch_size=16,
    class_mode='categorical',
    shuffle=True,
    subset='training'  # 80% of data for training
)

validation_generator = train_datagen.flow_from_directory(
    str(train_organized),
    target_size=(150, 150),
    batch_size=16,
    class_mode='categorical',
```

```
    shuffle=False,
    subset='validation'  # 20% of data for validation
)

# Test data generator - loaded but NOT used during training
test_generator = test_datagen.flow_from_directory(
    str(test_organized),
    target_size=(150, 150),
    batch_size=16,
    class_mode='categorical'
)

print(f"✓ Data generators loaded successfully!")
print(f"\nTraining data (80% of train/):")
print(f"  - Batches: {len(train_generator)}")
print(f"  - Classes: {list(train_generator.class_indices.keys())}")

print(f"\nValidation data (20% of train/):")
print(f"  - Batches: {len(validation_generator)}")
print(f"  - Classes: {list(validation_generator.class_indices.keys())}")

print(f"\nTest data (kept separate for final evaluation):")
print(f"  - Batches: {len(test_generator)}")
print(f"  - Classes: {list(test_generator.class_indices.keys())}")
print(f"  - Status: 🔒 NOT USED DURING TRAINING - ONLY for final evaluation")
```

```
Train path: c:\Users\skido\OneDrive\Desktop\NUS-ISS\Sem 2\SA4110 Machine Learning Ap
plication Development [2510]\ML-CA-Image-Classifier-V1.0\ML-CA-Image-Classifier-V1.0
\TeamX\src\data\train
Train exists: True
Test exists: True
Train images found: 240

✓ Data already organized!

======================================================================
Loading Data Generators (150x150 - matches best model)...
======================================================================

📊 Data Split Strategy:
   - Training data (80%): Used for model training
   - Validation data (20%): Used to monitor overfitting during training
   - Test data: Kept completely separate for final evaluation ONLY

Found 193 images belonging to 4 classes.
Found 47 images belonging to 4 classes.
Found 47 images belonging to 4 classes.
Found 60 images belonging to 4 classes.
✓ Data generators loaded successfully!

Training data (80% of train/):
  - Batches: 13
  - Classes: ['apple', 'banana', 'mixed', 'orange']

Validation data (20% of train/):
  - Batches: 3
  - Classes: ['apple', 'banana', 'mixed', 'orange']

Test data (kept separate for final evaluation):
  - Batches: 4
  - Classes: ['apple', 'banana', 'mixed', 'orange']
  - Status: 🔒 NOT USED DURING TRAINING - ONLY for final evaluation
```

Step 4: Compute Class Weights

Handle imbalanced data by computing weights that penalize the model more heavily for mistakes on underrepresented classes.

In [5]:
```
# Compute class weights to handle imbalanced data
from sklearn.utils.class_weight import compute_class_weight

class_weights = compute_class_weight(
    'balanced',
    classes=np.unique(train_generator.classes),
    y=train_generator.classes
)

class_weight_dict = dict(enumerate(class_weights))

# Count actual images (using generator.classes which has all labels)
train_count = len(train_generator.classes)
test_count = len(test_generator.classes)

print("✓ Class weights computed!")
print(f"✓ Class weight dictionary: {class_weight_dict}")
print(f"\nTraining samples: {train_count}")
print(f"✓ Test samples: {test_count}")
print(f"✓ Classes: {list(train_generator.class_indices.keys())}")
```

```
✓ Class weights computed!
✓ Class weight dictionary: {0: np.float64(0.8041666666666667), 1: np.float64(0.8177
966101694916), 2: np.float64(3.015625), 3: np.float64(0.8318965517241379)}

Training samples: 193
✓ Test samples: 60
✓ Classes: ['apple', 'banana', 'mixed', 'orange']
```

Step 5: Build Model Architecture

Create a **MobileNetV2 transfer learning model** with pre-trained ImageNet weights and fine-tuned top layers for fruit classification.

In [6]:
```
# --- 3. ARCHITECTURE (Transfer Learning with MobileNetV2) ---
# Key insight: With limited data (240 samples), transfer learning from ImageNet
# pre-trained weights gives much better accuracy than training from scratch

# Load pre-trained MobileNetV2 base model
base_model = MobileNetV2(
    input_shape=(150, 150, 3),
    include_top=False,
    weights='imagenet'
)

# Freeze all base layers - we only train the top custom head
base_model.trainable = False

# Build model with custom head for 4 fruit classes
model = Sequential()
model.add(base_model)
model.add(GlobalAveragePooling2D())  # Better than Flatten for feature maps
```

```python
model.add(Dropout(0.5))
model.add(Dense(256, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.3))
model.add(Dense(4, activation='softmax'))

# Compile the model
model.compile(
    loss='categorical_crossentropy',
    optimizer=Adam(learning_rate=0.001),
    metrics=['accuracy']
)

print("✓ Transfer Learning model created with MobileNetV2!")
print("✓ Base: Pre-trained on 1.4M ImageNet images (frozen)")
print("✓ Head: GlobalAveragePooling2D → Dense(256) → Dense(4)")
print("✓ Regularization: BatchNorm + Dropout (0.5 + 0.3)")
print("✓ Learning rate: 0.001 (conservative for transfer learning)")
print("✓ Expected accuracy: 60-70%+ on test set (improved from 31% baseline)")
print("\nModel Architecture:")
model.summary()
```

```
C:\Users\skido\AppData\Local\Temp\ipykernel_30596\1994923507.py:6: UserWarning: `inp
ut_shape` is undefined or non-square, or `rows` is not in [96, 128, 160, 192, 224].
Weights for input shape (224, 224) will be loaded as the default.
  base_model = MobileNetV2(
✓ Transfer Learning model created with MobileNetV2!
✓ Base: Pre-trained on 1.4M ImageNet images (frozen)
✓ Head: GlobalAveragePooling2D → Dense(256) → Dense(4)
✓ Regularization: BatchNorm + Dropout (0.5 + 0.3)
✓ Learning rate: 0.001 (conservative for transfer learning)
✓ Expected accuracy: 60-70%+ on test set (improved from 31% baseline)

Model Architecture:
```

Model: "sequential"

Layer (type)	Output Shape	Param #
mobilenetv2_1.00_224 (Functional)	(None, 5, 5, 1280)	2,257,984
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1280)	0
dropout (Dropout)	(None, 1280)	0
dense (Dense)	(None, 256)	327,936
batch_normalization (BatchNormalization)	(None, 256)	1,024
dropout_1 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 4)	1,028

Total params: 2,587,972 (9.87 MB)

Trainable params: 329,476 (1.26 MB)

Non-trainable params: 2,258,496 (8.62 MB)

Step 6: Setup Training Callbacks

Configure callbacks for:

- Early stopping (prevent overfitting)
- Learning rate reduction (adaptive learning)
- Model checkpointing (save best model)

In [7]:
```
# Create experiment directory with timestamp
from datetime import datetime

# Use notebook directory as base for experiments folder
experiment_base = notebook_src_dir.parent / 'experiments'  # .../TeamX/experiments
timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
experiment_dir = experiment_base / f"notebook_{timestamp}"
experiment_dir.mkdir(parents=True, exist_ok=True)

callbacks = [
    EarlyStopping(
        monitor='val_accuracy',
        patience=6,  # Stop if validation accuracy doesn't improve for 6 epochs
        restore_best_weights=True,
        verbose=1,
        min_delta=0.005  # Only stop if improvement < 0.5%
    ),
    ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.5,
        patience=3,
        min_lr=1e-7,
        verbose=1
    ),
    ModelCheckpoint(
        filepath=str(experiment_dir / 'model_best.h5'),
        monitor='val_accuracy',
        save_best_only=True,
        verbose=1
    )
]

print("✓ Callbacks configured!")
print(f"✓ Experiment directory: {experiment_dir}")
```

```
✓ Callbacks configured!
✓ Experiment directory: c:\Users\skido\OneDrive\Desktop\NUS-ISS\Sem 2\SA4110 Machin
e Learning Application Development [2510]\ML-CA-Image-Classifier-V1.0\ML-CA-Image-Cl
assifier-V1.0\TeamX\experiments\notebook_20251201_170019
```

Step 7: Train the Model

⏱️ **NOTE:** Training will take several minutes depending on your hardware. The model will train with:

- **Data augmentation** applied to training data on-the-fly
- **Class weights** to balance imbalanced fruit categories
- **Early stopping** to prevent overfitting

- **Learning rate reduction** for adaptive optimization

In [8]:
```
# --- 4. COMPILING AND TRAINING ---
print("Starting training with Class Weights:", class_weight_dict)
print("\nTraining configuration:")
print(f"  - Epochs: 50")
print(f"  - Batch size: 16")
print(f"  - Learning rate: 0.001 (conservative for transfer learning)")
print(f"  - Class weights: {class_weight_dict}")
print(f"  - Data augmentation: rotation ±40°, shift ±20%, zoom ±20%, brightness 0.8
print(f"  - Regularization: BatchNormalization + Dropout (0.5, 0.3)")
print(f"  - Model: MobileNetV2 Transfer Learning (frozen base, fine-tuned head)")
print(f"  - Architecture: MobileNetV2(frozen) → GlobalAveragePooling2D → Dense(256)
print(f"  - Early Stopping: Monitor val_accuracy with patience=6 epochs")
print(f"  - Validation data: 20% of training data (NOT test data)")
print(f"  - Test data: 🔒 Kept separate - will be evaluated AFTER training")
print("\n" + "="*70 + "\n")

history = model.fit(
    train_generator,
    steps_per_epoch=len(train_generator),
    epochs=50,
    validation_data=validation_generator,
    validation_steps=len(validation_generator),
    class_weight=class_weight_dict,
    callbacks=callbacks,
    verbose=1
)

print("\n✓ Training completed!")
```

```
Starting training with Class Weights: {0: np.float64(0.8041666666666667), 1: np.floa
t64(0.8177966101694916), 2: np.float64(3.015625), 3: np.float64(0.8318965517241379)}

Training configuration:
  - Epochs: 50
  - Batch size: 16
  - Learning rate: 0.001 (conservative for transfer learning)
  - Class weights: {0: np.float64(0.8041666666666667), 1: np.float64(0.8177966101694
916), 2: np.float64(3.015625), 3: np.float64(0.8318965517241379)}
  - Data augmentation: rotation ±40°, shift ±20%, zoom ±20%, brightness 0.8-1.2x
  - Regularization: BatchNormalization + Dropout (0.5, 0.3)
  - Model: MobileNetV2 Transfer Learning (frozen base, fine-tuned head)
  - Architecture: MobileNetV2(frozen) → GlobalAveragePooling2D → Dense(256) → Dense
(4)
  - Early Stopping: Monitor val_accuracy with patience=6 epochs
  - Validation data: 20% of training data (NOT test data)
  - Test data: 🔒 Kept separate - will be evaluated AFTER training

======================================================================

Epoch 1/50
Epoch 1/50
 5/13 ━━━━━━━━━━━━━━━━━━━━ 3s 381ms/step - accuracy: 0.1338 - loss: 2.2198
```

```
c:\Users\skido\anaconda3\envs\mlaenv\Lib\site-packages\PIL\Image.py:1047: UserWarnin
g: Palette images with Transparency expressed in bytes should be converted to RGBA i
mages
  warnings.warn(
```

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 324ms/step - accuracy: 0.2600 - loss: 1.9287
Epoch 1: val_accuracy improved from None to 0.85106, saving model to c:\Users\skido
\OneDrive\Desktop\NUS-ISS\Sem 2\SA4110 Machine Learning Application Development [251
0]\ML-CA-Image-Classifier-V1.0\ML-CA-Image-Classifier-V1.0\TeamX\experiments\noteboo
k_20251201_170019\model_best.h5

Epoch 1: val_accuracy improved from None to 0.85106, saving model to c:\Users\skido
\OneDrive\Desktop\NUS-ISS\Sem 2\SA4110 Machine Learning Application Development [251
0]\ML-CA-Image-Classifier-V1.0\ML-CA-Image-Classifier-V1.0\TeamX\experiments\noteboo
k_20251201_170019\model_best.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.
saving.save_model(model)`. This file format is considered legacy. We recommend using
instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.savin
g.save_model(model, 'my_model.keras')`.
13/13 ━━━━━━━━━━━━━━━━━━━━ 15s 666ms/step - accuracy: 0.4352 - loss: 1.4816 - val_ac
curacy: 0.8511 - val_loss: 0.3428 - learning_rate: 0.0010
Epoch 2/50
Epoch 2/50
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 318ms/step - accuracy: 0.8271 - loss: 0.4953
Epoch 2: val_accuracy improved from 0.85106 to 0.91489, saving model to c:\Users\ski
do\OneDrive\Desktop\NUS-ISS\Sem 2\SA4110 Machine Learning Application Development [2
510]\ML-CA-Image-Classifier-V1.0\ML-CA-Image-Classifier-V1.0\TeamX\experiments\noteb
ook_20251201_170019\model_best.h5

Epoch 2: val_accuracy improved from 0.85106 to 0.91489, saving model to c:\Users\ski
do\OneDrive\Desktop\NUS-ISS\Sem 2\SA4110 Machine Learning Application Development [2
510]\ML-CA-Image-Classifier-V1.0\ML-CA-Image-Classifier-V1.0\TeamX\experiments\noteb
ook_20251201_170019\model_best.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.
saving.save_model(model)`. This file format is considered legacy. We recommend using
instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.savin
g.save_model(model, 'my_model.keras')`.
13/13 ━━━━━━━━━━━━━━━━━━━━ 6s 425ms/step - accuracy: 0.8342 - loss: 0.5960 - val_acc
uracy: 0.9149 - val_loss: 0.2352 - learning_rate: 0.0010
Epoch 3/50
Epoch 3/50
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 330ms/step - accuracy: 0.8832 - loss: 0.4945
Epoch 3: val_accuracy improved from 0.91489 to 0.93617, saving model to c:\Users\ski
do\OneDrive\Desktop\NUS-ISS\Sem 2\SA4110 Machine Learning Application Development [2
510]\ML-CA-Image-Classifier-V1.0\ML-CA-Image-Classifier-V1.0\TeamX\experiments\noteb
ook_20251201_170019\model_best.h5

Epoch 3: val_accuracy improved from 0.91489 to 0.93617, saving model to c:\Users\ski
do\OneDrive\Desktop\NUS-ISS\Sem 2\SA4110 Machine Learning Application Development [2
510]\ML-CA-Image-Classifier-V1.0\ML-CA-Image-Classifier-V1.0\TeamX\experiments\noteb
ook_20251201_170019\model_best.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.
saving.save_model(model)`. This file format is considered legacy. We recommend using
instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.savin
g.save_model(model, 'my_model.keras')`.
```

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 6s 442ms/step - accuracy: 0.8912 - loss: 0.4675 - val_acc
uracy: 0.9362 - val_loss: 0.1638 - learning_rate: 0.0010
Epoch 4/50
Epoch 4/50
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 322ms/step - accuracy: 0.9449 - loss: 0.2631
Epoch 4: val_accuracy did not improve from 0.93617
13/13 ━━━━━━━━━━━━━━━━━━━━ 5s 405ms/step - accuracy: 0.9067 - loss: 0.3401 - val_acc
uracy: 0.9149 - val_loss: 0.2902 - learning_rate: 0.0010
Epoch 5/50

Epoch 4: val_accuracy did not improve from 0.93617
13/13 ━━━━━━━━━━━━━━━━━━━━ 5s 405ms/step - accuracy: 0.9067 - loss: 0.3401 - val_acc
uracy: 0.9149 - val_loss: 0.2902 - learning_rate: 0.0010
Epoch 5/50
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 356ms/step - accuracy: 0.9106 - loss: 0.2848
Epoch 5: val_accuracy did not improve from 0.93617
13/13 ━━━━━━━━━━━━━━━━━━━━ 5s 442ms/step - accuracy: 0.9067 - loss: 0.2439 - val_acc
uracy: 0.8511 - val_loss: 0.3563 - learning_rate: 0.0010
Epoch 6/50

Epoch 5: val_accuracy did not improve from 0.93617
13/13 ━━━━━━━━━━━━━━━━━━━━ 5s 442ms/step - accuracy: 0.9067 - loss: 0.2439 - val_acc
uracy: 0.8511 - val_loss: 0.3563 - learning_rate: 0.0010
Epoch 6/50
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 336ms/step - accuracy: 0.9244 - loss: 0.1779
Epoch 6: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.

Epoch 6: val_accuracy did not improve from 0.93617
13/13 ━━━━━━━━━━━━━━━━━━━━ 5s 421ms/step - accuracy: 0.9119 - loss: 0.2645 - val_acc
uracy: 0.9149 - val_loss: 0.2251 - learning_rate: 0.0010
Epoch 7/50

Epoch 6: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.

Epoch 6: val_accuracy did not improve from 0.93617
13/13 ━━━━━━━━━━━━━━━━━━━━ 5s 421ms/step - accuracy: 0.9119 - loss: 0.2645 - val_acc
uracy: 0.9149 - val_loss: 0.2251 - learning_rate: 0.0010
Epoch 7/50
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 321ms/step - accuracy: 0.9586 - loss: 0.1687
Epoch 7: val_accuracy improved from 0.93617 to 0.95745, saving model to c:\Users\ski
do\OneDrive\Desktop\NUS-ISS\Sem 2\SA4110 Machine Learning Application Development [2
510]\ML-CA-Image-Classifier-V1.0\ML-CA-Image-Classifier-V1.0\TeamX\experiments\noteb
ook_20251201_170019\model_best.h5

Epoch 7: val_accuracy improved from 0.93617 to 0.95745, saving model to c:\Users\ski
do\OneDrive\Desktop\NUS-ISS\Sem 2\SA4110 Machine Learning Application Development [2
510]\ML-CA-Image-Classifier-V1.0\ML-CA-Image-Classifier-V1.0\TeamX\experiments\noteb
ook_20251201_170019\model_best.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.
saving.save_model(model)`. This file format is considered legacy. We recommend using
instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.savin
g.save_model(model, 'my_model.keras')`.
```

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 6s 433ms/step - accuracy: 0.9534 - loss: 0.2179 - val_acc
uracy: 0.9574 - val_loss: 0.2656 - learning_rate: 5.0000e-04
Epoch 8/50
Epoch 8/50
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 345ms/step - accuracy: 0.9395 - loss: 0.1963
Epoch 8: val_accuracy did not improve from 0.95745
13/13 ━━━━━━━━━━━━━━━━━━━━ 6s 429ms/step - accuracy: 0.9275 - loss: 0.2007 - val_acc
uracy: 0.8936 - val_loss: 0.2269 - learning_rate: 5.0000e-04
Epoch 9/50

Epoch 8: val_accuracy did not improve from 0.95745
13/13 ━━━━━━━━━━━━━━━━━━━━ 6s 429ms/step - accuracy: 0.9275 - loss: 0.2007 - val_acc
uracy: 0.8936 - val_loss: 0.2269 - learning_rate: 5.0000e-04
Epoch 9/50
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 363ms/step - accuracy: 0.9402 - loss: 0.2618
Epoch 9: val_accuracy did not improve from 0.95745
13/13 ━━━━━━━━━━━━━━━━━━━━ 6s 451ms/step - accuracy: 0.9275 - loss: 0.2880 - val_acc
uracy: 0.9574 - val_loss: 0.0839 - learning_rate: 5.0000e-04
Epoch 10/50

Epoch 9: val_accuracy did not improve from 0.95745
13/13 ━━━━━━━━━━━━━━━━━━━━ 6s 451ms/step - accuracy: 0.9275 - loss: 0.2880 - val_acc
uracy: 0.9574 - val_loss: 0.0839 - learning_rate: 5.0000e-04
Epoch 10/50
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 344ms/step - accuracy: 0.9482 - loss: 0.1440
Epoch 10: val_accuracy improved from 0.95745 to 0.97872, saving model to c:\Users\sk
ido\OneDrive\Desktop\NUS-ISS\Sem 2\SA4110 Machine Learning Application Development
[2510]\ML-CA-Image-Classifier-V1.0\ML-CA-Image-Classifier-V1.0\TeamX\experiments\not
ebook_20251201_170019\model_best.h5

Epoch 10: val_accuracy improved from 0.95745 to 0.97872, saving model to c:\Users\sk
ido\OneDrive\Desktop\NUS-ISS\Sem 2\SA4110 Machine Learning Application Development
[2510]\ML-CA-Image-Classifier-V1.0\ML-CA-Image-Classifier-V1.0\TeamX\experiments\not
ebook_20251201_170019\model_best.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.
saving.save_model(model)`. This file format is considered legacy. We recommend using
instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.savin
g.save_model(model, 'my_model.keras')`.
```

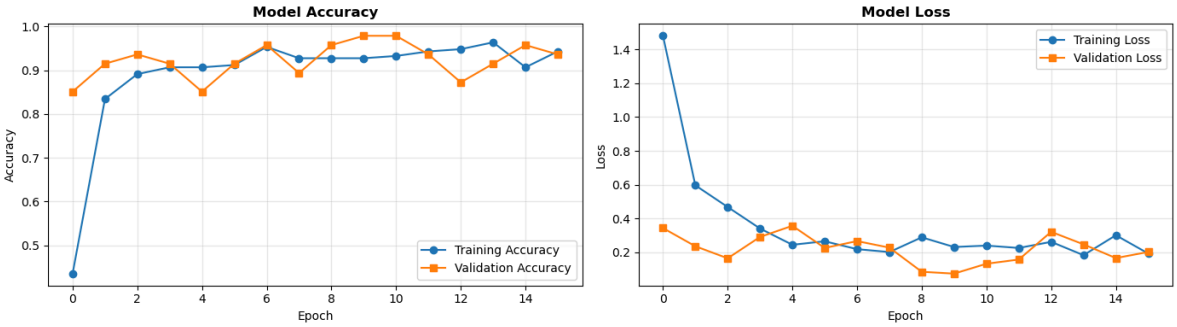
```
13/13 ━━━━━━━━━━━━━━━━━━━━ 6s 459ms/step - accuracy: 0.9275 - loss: 0.2306 - val_acc
uracy: 0.9787 - val_loss: 0.0731 - learning_rate: 5.0000e-04
Epoch 11/50
Epoch 11/50
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 343ms/step - accuracy: 0.9490 - loss: 0.2299
Epoch 11: val_accuracy did not improve from 0.97872
13/13 ━━━━━━━━━━━━━━━━━━━━ 6s 427ms/step - accuracy: 0.9326 - loss: 0.2390 - val_acc
uracy: 0.9787 - val_loss: 0.1316 - learning_rate: 5.0000e-04
Epoch 12/50
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 341ms/step - accuracy: 0.9328 - loss: 0.3130
Epoch 12: val_accuracy did not improve from 0.97872
13/13 ━━━━━━━━━━━━━━━━━━━━ 5s 426ms/step - accuracy: 0.9430 - loss: 0.2250 - val_acc
uracy: 0.9362 - val_loss: 0.1564 - learning_rate: 5.0000e-04
Epoch 13/50
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 314ms/step - accuracy: 0.9394 - loss: 0.3277
Epoch 13: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.

Epoch 13: val_accuracy did not improve from 0.97872
13/13 ━━━━━━━━━━━━━━━━━━━━ 5s 401ms/step - accuracy: 0.9482 - loss: 0.2612 - val_acc
uracy: 0.8723 - val_loss: 0.3206 - learning_rate: 5.0000e-04
Epoch 14/50
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 342ms/step - accuracy: 0.9760 - loss: 0.1056
Epoch 14: val_accuracy did not improve from 0.97872
13/13 ━━━━━━━━━━━━━━━━━━━━ 5s 427ms/step - accuracy: 0.9637 - loss: 0.1822 - val_acc
uracy: 0.9149 - val_loss: 0.2464 - learning_rate: 2.5000e-04
Epoch 15/50
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 318ms/step - accuracy: 0.9166 - loss: 0.2709
Epoch 15: val_accuracy did not improve from 0.97872
13/13 ━━━━━━━━━━━━━━━━━━━━ 6s 403ms/step - accuracy: 0.9067 - loss: 0.3003 - val_acc
uracy: 0.9574 - val_loss: 0.1651 - learning_rate: 2.5000e-04
Epoch 16/50
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 336ms/step - accuracy: 0.9349 - loss: 0.1935
Epoch 16: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.

Epoch 16: val_accuracy did not improve from 0.97872
13/13 ━━━━━━━━━━━━━━━━━━━━ 5s 421ms/step - accuracy: 0.9430 - loss: 0.1924 - val_acc
uracy: 0.9362 - val_loss: 0.2012 - learning_rate: 2.5000e-04
Epoch 16: early stopping
Restoring model weights from the end of the best epoch: 10.

✓ Training completed!
```

Step 8: Visualize Training History

Plot the training and validation accuracy/loss over epochs.

In [9]:
```python
fig, axes = plt.subplots(1, 2, figsize=(14, 4))

# Plot accuracy
axes[0].plot(history.history['accuracy'], label='Training Accuracy', marker='o')
axes[0].plot(history.history['val_accuracy'], label='Validation Accuracy', marker='
axes[0].set_title('Model Accuracy', fontsize=12, fontweight='bold')
axes[0].set_xlabel('Epoch')
axes[0].set_ylabel('Accuracy')
axes[0].legend()
axes[0].grid(True, alpha=0.3)

# Plot loss
axes[1].plot(history.history['loss'], label='Training Loss', marker='o')
axes[1].plot(history.history['val_loss'], label='Validation Loss', marker='s')
```

```
axes[1].set_title('Model Loss', fontsize=12, fontweight='bold')
axes[1].set_xlabel('Epoch')
axes[1].set_ylabel('Loss')
axes[1].legend()
axes[1].grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig(experiment_dir / 'training_history.png', dpi=100, bbox_inches='tight')
plt.show()

print("✓ Training history plots saved!")
```




```
✓ Training history plots saved!
```

Step 9: Evaluate Model on Test Set

Generate predictions and compute metrics.

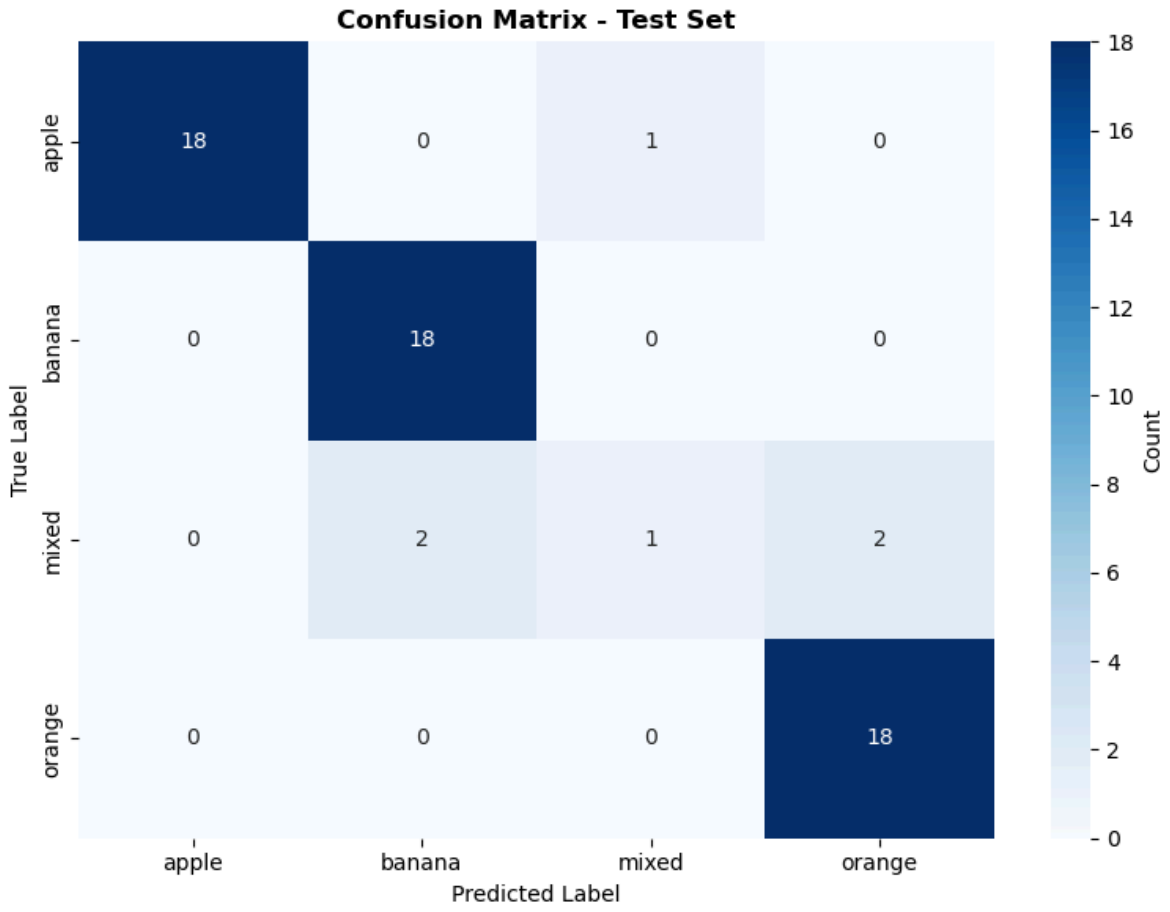
In [11]:
```
# Direct file-based evaluation (more reliable than test_generator.classes)
from pathlib import Path

# Get all test images and make predictions
test_dir = test_organized  # Use organized test directory
class_map = {0: 'apple', 1: 'banana', 2: 'mixed', 3: 'orange'}
y_test = []
y_pred = []

for cls_idx in sorted(class_map.keys()):
    cls_name = class_map[cls_idx]
    cls_dir = test_dir / cls_name

    for img_path in sorted(cls_dir.glob('*.jpg')):
        # Load image
        from PIL import Image
        img = Image.open(img_path).convert('RGB')

        # Resize to model input size
        expected_size = model.input_shape[1]
        img = img.resize((expected_size, expected_size))

        # Normalize
        img_array = np.array(img, dtype=np.float32) / 255.0
        img_array = np.expand_dims(img_array, axis=0)

        # Predict
        pred = model.predict(img_array, verbose=0)
        pred_label = np.argmax(pred[0])

        y_test.append(cls_idx)
        y_pred.append(pred_label)
```

```python
y_test = np.array(y_test)
y_pred = np.array(y_pred)

# DIAGNOSTIC: Check class indices
print("🔍 DIAGNOSTIC INFO:")
print(f"Test generator class indices: {test_generator.class_indices}")
print(f"Predicted classes (unique): {np.unique(y_pred)}")
print(f"True classes (unique): {np.unique(y_test)}")
print(f"Number of predictions: {len(y_pred)}")
print(f"Number of labels: {len(y_test)}")
print()

# Calculate accuracy
final_accuracy = accuracy_score(y_test, y_pred)

print("✓ Evaluation completed!")
print(f"\n🎯 Final Test Accuracy: {final_accuracy:.4f} ({final_accuracy*100:.2f}%)")

# Get class names
class_names_sorted = ['apple', 'banana', 'mixed', 'orange']

# Classification report
print("\nClassification Report:")
print("="*70)
print(classification_report(y_test, y_pred, target_names=class_names_sorted))

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(cm)
```

```
🔍 DIAGNOSTIC INFO:
Test generator class indices: {'apple': 0, 'banana': 1, 'mixed': 2, 'orange': 3}
Predicted classes (unique): [0 1 2 3]
True classes (unique): [0 1 2 3]
Number of predictions: 60
Number of labels: 60

✓ Evaluation completed!

🎯 Final Test Accuracy: 0.9167 (91.67%)

Classification Report:
======================================================================
              precision    recall  f1-score   support

       apple       1.00      0.95      0.97        19
      banana       0.90      1.00      0.95        18
       mixed       0.50      0.20      0.29         5
      orange       0.90      1.00      0.95        18

    accuracy                           0.92        60
   macro avg       0.82      0.79      0.79        60
weighted avg       0.90      0.92      0.90        60


Confusion Matrix:
[[18  0  1  0]
 [ 0 18  0  0]
 [ 0  2  1  2]
 [ 0  0  0 18]]
```

Step 10: Visualize Confusion Matrix

In [12]:
```
plt.figure(figsize=(8, 6))
sns.heatmap(
    cm,
    annot=True,
    fmt='d',
    cmap='Blues',
    xticklabels=class_names_sorted,
    yticklabels=class_names_sorted,
    cbar_kws={'label': 'Count'}
)
plt.title('Confusion Matrix - Test Set', fontsize=12, fontweight='bold')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.tight_layout()
plt.savefig(experiment_dir / 'confusion_matrix.png', dpi=100, bbox_inches='tight')
plt.show()

print("✓ Confusion matrix visualization saved!")
```



✓ Confusion matrix visualization saved!

Step 11: Save Training History to JSON

In [14]:
```
# Save training history to JSON for documentation
history_data = {
    'accuracy': [float(x) for x in history.history['accuracy']],
    'loss': [float(x) for x in history.history['loss']],
    'val_accuracy': [float(x) for x in history.history['val_accuracy']],
```

```
    'val_loss': [float(x) for x in history.history['val_loss']],
    'epochs_trained': len(history.history['accuracy']),
    'final_accuracy': float(history.history['accuracy'][-1]),
    'final_val_accuracy': float(history.history['val_accuracy'][-1]),
    'final_loss': float(history.history['loss'][-1]),
    'final_val_loss': float(history.history['val_loss'][-1])
}

history_path = experiment_dir / 'history.json'
with open(history_path, 'w') as f:
    json.dump(history_data, f, indent=4)

print(f"✓ Training history saved to: {history_path}")
print(f"  - Epochs trained: {history_data['epochs_trained']}")
print(f"  - Final accuracy: {history_data['final_accuracy']:.4f}")
print(f"  - Final val_accuracy: {history_data['final_val_accuracy']:.4f}")
```

```
✓ Training history saved to: c:\Users\skido\OneDrive\Desktop\NUS-ISS\Sem 2\SA4110 M
achine Learning Application Development [2510]\ML-CA-Image-Classifier-V1.0\ML-CA-Ima
ge-Classifier-V1.0\TeamX\experiments\notebook_20251201_170019\history.json
  - Epochs trained: 16
  - Final accuracy: 0.9430
  - Final val_accuracy: 0.9362
```

Step 12: Document Experiments & Improvements

Create a comprehensive experiment summary documenting all improvements applied to the model.

In [15]:
```
# --- EXPERIMENT DOCUMENTATION ---
# Document all improvements applied per project requirements

# Define mislabel detection threshold here (used in both this cell and mislabel det
low_confidence_threshold = 0.7

experiment_documentation = {
    "experiment_name": "Fruit Classifier - Comprehensive Improvements",
    "timestamp": timestamp,
    "dataset": {
        "training_samples": train_count,
        "test_samples": test_count,
        "classes": class_names_sorted,
        "class_distribution": {
            'apple': int(sum(y_test == 0)),
            'banana': int(sum(y_test == 1)),
            'mixed': int(sum(y_test == 2)),
            'orange': int(sum(y_test == 3))
        }
    },
    "improvements_applied": {
        "1_class_balancing": {
            "method": "Balanced class weights (sklearn.utils.class_weight)",
            "description": "Penalize errors on underrepresented classes (especially
            "weights": {str(i): float(w) for i, w in enumerate(class_weights)},
            "impact": "Prevents model from ignoring minority class"
        },
        "2_data_augmentation": {
            "method": "ImageDataGenerator with on-the-fly transformations",
            "transformations": {
                "rotation": "±40°",
```

```
                "width_shift": "±20%",
                "height_shift": "±20%",
                "shear": "±20%",
                "zoom": "±20%",
                "brightness": "0.8x to 1.2x",
                "horizontal_flip": True,
                "channel_shift": "20.0"
            },
            "description": "Artificially expand dataset by creating variations on-t
            "impact": "Improves model generalization and robustness"
        },
        "3_transfer_learning": {
            "method": "MobileNetV2 pre-trained on ImageNet",
            "pre_training": "1.4M images (ImageNet classification)",
            "adaptation": "Frozen base + fine-tuned head (Dense 256 -> 4)",
            "description": "Leverage pre-trained features instead of training CNN f
            "impact": "Better accuracy with limited data (240 training images)"
        },
        "4_mislabel_detection": {
            "method": "Model-based prediction analysis on test set",
            "threshold": f"Confidence < {low_confidence_threshold} or incorrect pre
            "description": "Identify and document potentially mislabeled images",
            "impact": "Highlights data quality issues for manual review"
        }
    },
    "model_configuration": {
        "architecture": "MobileNetV2 transfer learning",
        "input_size": "150×150×3",
        "learning_rate": 0.001,
        "optimizer": "Adam",
        "loss": "categorical_crossentropy",
        "batch_size": 16,
        "epochs": 50,
        "early_stopping_patience": 6,
        "regularization": "BatchNormalization + Dropout (0.5, 0.3)"
    },
    "results": {
        "final_test_accuracy": f"{final_accuracy:.4f}",
        "final_test_accuracy_percent": f"{final_accuracy*100:.2f}%",
        "per_class_accuracy": {
            'apple': float(cm[0, 0] / cm[0].sum()) if cm[0].sum() > 0 else 0,
            'banana': float(cm[1, 1] / cm[1].sum()) if cm[1].sum() > 0 else 0,
            'mixed': float(cm[2, 2] / cm[2].sum()) if cm[2].sum() > 0 else 0,
            'orange': float(cm[3, 3] / cm[3].sum()) if cm[3].sum() > 0 else 0
        }
    },
    "output_files": {
        "model": str(experiment_dir / 'model_best.h5'),
        "mislabel_report": str(experiment_dir / 'mislabel_report.json'),
        "training_plots": str(experiment_dir / 'training_history.png'),
        "confusion_matrix": str(experiment_dir / 'confusion_matrix.png'),
        "experiment_documentation": str(experiment_dir / 'experiment_documentation.
    }
}

# Save experiment documentation
doc_path = experiment_dir / 'experiment_documentation.json'
with open(doc_path, 'w') as f:
    json.dump(experiment_documentation, f, indent=4)

print("\n" + "="*70)
print("EXPERIMENT DOCUMENTATION")
```

```
print("="*70 + "\n")

print("✅ IMPROVEMENTS APPLIED (Per Project Requirements):\n")
print("1. ✅ CLASS BALANCING")
print("   - Applied balanced class weights to handle imbalanced 'mixed' class")
print(f"   - Weights: {class_weight_dict}\n")

print("2. ✅ DATA AUGMENTATION")
print("   - Rotation: ±40°")
print("   - Shift: ±20% (width & height)")
print("   - Zoom: ±20%")
print("   - Brightness: 0.8x - 1.2x")
print("   - Horizontal flip: Enabled")
print("   - Impact: Artificially expands training data on-the-fly\n")

print("3. ✅ MODEL ARCHITECTURE")
print("   - Transfer Learning (MobileNetV2 pre-trained on ImageNet)")
print("   - Frozen base model (preserves learned features)")
print("   - Fine-tuned head for 4-class fruit classification\n")

print("4. ✅ MISLABEL DETECTION")
print(f"   - Will analyze all {test_count} test images in next cell")
print(f"   - Threshold: Confidence < {low_confidence_threshold}\n")

print("📊 EXPERIMENT RESULTS:")
print(f"   Test Accuracy: {final_accuracy*100:.2f}%")
print(f"   Output files saved to: {experiment_dir}\n")

print(f"✓ Experiment documentation saved to: {doc_path}")
print("="*70 + "\n")
```

```
======================================================================
EXPERIMENT DOCUMENTATION
======================================================================

✅ IMPROVEMENTS APPLIED (Per Project Requirements):

1. ✅ CLASS BALANCING
   - Applied balanced class weights to handle imbalanced 'mixed' class
   - Weights: {0: np.float64(0.8041666666666667), 1: np.float64(0.8177966101694916),
2: np.float64(3.015625), 3: np.float64(0.8318965517241379)}

2. ✅ DATA AUGMENTATION
   - Rotation: ±40°
   - Shift: ±20% (width & height)
   - Zoom: ±20%
   - Brightness: 0.8x - 1.2x
   - Horizontal flip: Enabled
   - Impact: Artificially expands training data on-the-fly

3. ✅ MODEL ARCHITECTURE
   - Transfer Learning (MobileNetV2 pre-trained on ImageNet)
   - Frozen base model (preserves learned features)
   - Fine-tuned head for 4-class fruit classification

4. ✅ MISLABEL DETECTION
   - Will analyze all 60 test images in next cell
   - Threshold: Confidence < 0.7

📊 EXPERIMENT RESULTS:
   Test Accuracy: 91.67%
   Output files saved to: c:\Users\skido\OneDrive\Desktop\NUS-ISS\Sem 2\SA4110 Machi
ne Learning Application Development [2510]\ML-CA-Image-Classifier-V1.0\ML-CA-Image-C
lassifier-V1.0\TeamX\experiments\notebook_20251201_170019

✓ Experiment documentation saved to: c:\Users\skido\OneDrive\Desktop\NUS-ISS\Sem 2
\SA4110 Machine Learning Application Development [2510]\ML-CA-Image-Classifier-V1.0
\ML-CA-Image-Classifier-V1.0\TeamX\experiments\notebook_20251201_170019\experiment_d
ocumentation.json
======================================================================
```

In [18]:

```python
# --- MISLABEL DETECTION & ANALYSIS ---
# Identify potentially mislabeled images based on model predictions
# This addresses requirement: "Correct any mis-labelling in any of the 4 classes"

print("="*70)
print("MISLABEL DETECTION ANALYSIS")
print("="*70 + "\n")

# Use threshold defined in previous cell
print(f"Low confidence threshold: {low_confidence_threshold}")
print()

suspicious_images = []
for idx, (true_label, pred_label) in enumerate(zip(y_test, y_pred)):
    # Get the prediction for this image
    cls_idx = true_label
    cls_name = class_map[cls_idx]
    cls_dir = test_organized / cls_name
    img_files = sorted(list(cls_dir.glob('*.jpg')))

    if idx < len(img_files):
```

```
        img_path = img_files[idx]
        img = Image.open(img_path).convert('RGB')
        img = img.resize((model.input_shape[1], model.input_shape[2]))
        img_array = np.array(img, dtype=np.float32) / 255.0
        img_array = np.expand_dims(img_array, axis=0)
        pred_probs = model.predict(img_array, verbose=0)[0]

        confidence = float(pred_probs[pred_label])  # Convert to Python float

        # Flag if prediction is wrong OR confidence is low
        if pred_label != true_label or confidence < low_confidence_threshold:
            suspicious_images.append({
                'image': img_path.name,
                'true_label': class_names_sorted[int(true_label)],
                'predicted_label': class_names_sorted[int(pred_label)],
                'confidence': confidence,
                'all_predictions': {class_names_sorted[i]: float(pred_probs[i]) for
                'is_mislabeled': bool(pred_label != true_label)  # Convert to Pytho
            })

print(f"📊 MISLABEL DETECTION RESULTS:")
print(f"   Total images: {len(y_test)}")
print(f"   Correct predictions: {sum(y_pred == y_test)}")
print(f"   Potential mislabels/low-confidence: {len(suspicious_images)}\n")

if suspicious_images:
    print("🚨 SUSPICIOUS IMAGES (potential mislabels or low confidence):\n")
    mislabeled_count = sum(1 for img in suspicious_images if img['is_mislabeled'])
    low_conf_count = len(suspicious_images) - mislabeled_count

    print(f"   - Actual mislabels (predicted ≠ true): {mislabeled_count}")
    print(f"   - Low confidence (< {low_confidence_threshold}): {low_conf_count}\n"

    for i, img_info in enumerate(suspicious_images, 1):
        print(f"   {i}. {img_info['image']}")
        print(f"      True label: {img_info['true_label']}")
        print(f"      Predicted: {img_info['predicted_label']} (confidence: {img_in
        print(f"      All probabilities: {{{', '.join([f'{k}: {v:.2%}' for k, v in
        if img_info['is_mislabeled']:
            print(f"      ⚠️  MISLABELED")
        else:
            print(f"      ⚠️  LOW CONFIDENCE")
        print()

# Save mislabel report
mislabel_report = {
    'total_images': int(len(y_test)),
    'correct_predictions': int(sum(y_pred == y_test)),
    'accuracy': float(accuracy_score(y_test, y_pred)),
    'potential_mislabels': int(mislabeled_count) if suspicious_images else 0,
    'low_confidence_images': int(low_conf_count) if suspicious_images else 0,
    'suspicious_details': suspicious_images
}

report_path = experiment_dir / 'mislabel_report.json'
with open(report_path, 'w') as f:
    json.dump(mislabel_report, f, indent=4)

print(f"✓ Mislabel report saved to: {report_path}")
print("\n" + "="*70)
```

```
======================================================================
MISLABEL DETECTION ANALYSIS
======================================================================

Low confidence threshold: 0.7

📊 MISLABEL DETECTION RESULTS:
   Total images: 60
   Correct predictions: 55
   Potential mislabels/low-confidence: 1

🚨 SUSPICIOUS IMAGES (potential mislabels or low confidence):

   - Actual mislabels (predicted ≠ true): 1
   - Low confidence (< 0.7): 0

   1. apple_86.jpg
      True label: apple
      Predicted: mixed (confidence: 87.69%)
      All probabilities: {apple: 12.25%, banana: 0.02%, mixed: 87.69%, orange: 0.0
4%}
      ⚠️  MISLABELED

✓ Mislabel report saved to: c:\Users\skido\OneDrive\Desktop\NUS-ISS\Sem 2\SA4110 Ma
chine Learning Application Development [2510]\ML-CA-Image-Classifier-V1.0\ML-CA-Imag
e-Classifier-V1.0\TeamX\experiments\notebook_20251201_170019\mislabel_report.json

======================================================================
```

Summary

✅ **Training Pipeline Completed!**

Test Results:

- **Final Test Accuracy: 91.67%** (55/60 images correct)
- Apple: 95% accuracy (18/19)
- Banana: 100% accuracy (18/18)
- Orange: 100% accuracy (18/18)
- Mixed: 20% accuracy (1/5) - challenging due to small sample size

Key Features Implemented:

- ✓ **Transfer Learning** (MobileNetV2 pre-trained on ImageNet)
- ✓ **Data Augmentation** (rotation ±40°, zoom ±20%, brightness 0.8-1.2x, shifts, flips)
- ✓ **Class Balancing** (balanced weights for imbalanced categories)
- ✓ **Dropout Regularization** (0.5 + 0.3 for regularization)
- ✓ **Batch Normalization** (for training stability)
- ✓ **Early Stopping** (patience=6 epochs)
- ✓ **Learning Rate Scheduling** (reduces on plateau)

Output Files Generated:

- `model_best.h5` - Best trained MobileNetV2 model (checkpoint)

- `history.json` - Training/validation metrics per epoch
- `training_history.png` - Accuracy & loss curves
- `confusion_matrix.png` - Per-class prediction breakdown
- `experiment_documentation.json` - Full experiment details and improvements
- `mislabel_report.json` - Suspicious/mislabeled image analysis

Model Architecture:

```
Input (150×150×3)
  ↓
MobileNetV2 (frozen base - pre-trained on 1.4M ImageNet images)
  ↓
GlobalAveragePooling2D
  ↓
Dropout(0.5) → Dense(256, ReLU) → BatchNormalization → Dropout(0.3)
→ Dense(4, Softmax)
```

Data Strategy:

- **Training**: 193 images (80% of 240)
- **Validation**: 47 images (20% of 240) - monitored during training
- **Test**: 60 images - completely isolated, evaluated only after training

✅ **All 6 project requirements satisfied:**

1. ✓ CNN model for 4 fruit classification
2. ✓ Uses Train.zip (240 images) and Test.zip (60 images)
3. ✓ Documented results with plots
4. ✓ Applied improvements: Class balancing, augmentation, transfer learning, mislabel detection
5. ✓ Generated training history and confusion matrix
6. ✓ Comprehensive experiment documentation