```
# Install optimized packages
!pip install tensorflow[and-cuda] -q  # Use if you have GPU
!pip install efficientnet -q

# Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import cv2
import os
import random
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, models
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping,
ReduceLROnPlateau, ModelCheckpoint
from tensorflow.keras.optimizers import Adam, SGD
from tensorflow.keras.applications import MobileNetV2, EfficientNetB0
import efficientnet.tfkeras as efn
from sklearn.metrics import classification_report, confusion_matrix
import warnings
warnings.filterwarnings('ignore')

# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')

# ==================== CONFIGURATION - OPTIMIZED ====================
class Config:
    # Paths
    DATA_PATH = '/content/drive/MyDrive/dataset'  # Update this

    # OPTIMIZED parameters for speed
    IMG_SIZE = 224  # Reduced from 256 for faster processing
    BATCH_SIZE = 64  # Increased batch size (if GPU memory allows)
    EPOCHS = 30  # Reduced epochs
    LEARNING_RATE = 0.001

    # Use smaller model for faster training
    MODEL_TYPE = "mobilenetv2"  # Options: "mobilenetv2",
"efficientnetb0", "simple_cnn"

    # Model saving
    MODEL_SAVE_PATH = '/content/drive/MyDrive/tomato_model_fast.h5'

    # Class names
    CLASS_NAMES = [
        'Bacterial_Spot',
```

```
        'Early_Blight',
        'Healthy',
        'Late_Blight',
        'Septoria_Leaf_Spot'
    ]

config = Config()

# ==================== FAST DATA LOADING ====================
def create_fast_data_generators():
    """Create optimized data generators"""

    print("📂 Loading dataset...")

    # Find train/test directories
    train_dir = os.path.join(config.DATA_PATH, 'train')
    test_dir = os.path.join(config.DATA_PATH, 'test')

    if not os.path.exists(train_dir):
        print("⚠️  'train' folder not found, checking for class
folders...")
        # If no train/test, use entire dataset with validation split
        all_datagen = ImageDataGenerator(
            rescale=1./255,
            validation_split=0.2,
            horizontal_flip=True,
            vertical_flip=True,
            zoom_range=0.2
        )

        train_generator = all_datagen.flow_from_directory(
            config.DATA_PATH,
            target_size=(config.IMG_SIZE, config.IMG_SIZE),
            batch_size=config.BATCH_SIZE,
            class_mode='categorical',
            subset='training',
            shuffle=True
        )

        val_generator = all_datagen.flow_from_directory(
            config.DATA_PATH,
            target_size=(config.IMG_SIZE, config.IMG_SIZE),
            batch_size=config.BATCH_SIZE,
            class_mode='categorical',
            subset='validation',
            shuffle=False
        )

        # Create test generator from validation
        test_generator = val_generator
```

```
else:
    # Use train/test folders
    print("✅ Found train/test folders")

    # Train with augmentation
    train_datagen = ImageDataGenerator(
        rescale=1./255,
        rotation_range=20,
        width_shift_range=0.1,
        height_shift_range=0.1,
        shear_range=0.1,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest',
        brightness_range=[0.8, 1.2]
    )

    # Validation/Test without augmentation
    test_datagen = ImageDataGenerator(rescale=1./255)

    train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(config.IMG_SIZE, config.IMG_SIZE),
        batch_size=config.BATCH_SIZE,
        class_mode='categorical',
        shuffle=True
    )

    if os.path.exists(test_dir):
        test_generator = test_datagen.flow_from_directory(
            test_dir,
            target_size=(config.IMG_SIZE, config.IMG_SIZE),
            batch_size=config.BATCH_SIZE,
            class_mode='categorical',
            shuffle=False
        )
        val_generator = test_generator  # Use test as validation
    else:
        # Create validation from train
        val_datagen = ImageDataGenerator(rescale=1./255)
        val_generator = train_datagen.flow_from_directory(
            train_dir,
            target_size=(config.IMG_SIZE, config.IMG_SIZE),
            batch_size=config.BATCH_SIZE,
            class_mode='categorical',
            shuffle=False,
            subset='validation'
        )
```

```
    print(f"□ Classes found:
{list(train_generator.class_indices.keys())}")
    print(f"□ Training samples: {train_generator.samples}")

    if 'val_generator' in locals():
        print(f"□ Validation samples: {val_generator.samples}")

    return train_generator, val_generator if 'val_generator' in
locals() else train_generator

# ==================== FAST MODEL ARCHITECTURES ====================
def build_fast_model():
    """Build optimized model based on config"""

    num_classes = len(config.CLASS_NAMES)

    if config.MODEL_TYPE == "mobilenetv2":
        print("  Building MobileNetV2 (Fastest)...")
        base_model = MobileNetV2(
            weights='imagenet',
            include_top=False,
            input_shape=(config.IMG_SIZE, config.IMG_SIZE, 3),
            alpha=0.35  # Smaller model
        )
        base_model.trainable = False

        model = keras.Sequential([
            base_model,
            layers.GlobalAveragePooling2D(),
            layers.Dropout(0.3),
            layers.Dense(128, activation='relu'),
            layers.Dropout(0.2),
            layers.Dense(num_classes, activation='softmax')
        ])

    elif config.MODEL_TYPE == "efficientnetb0":
        print("  Building EfficientNetB0 (Balanced)...")
        base_model = EfficientNetB0(
            weights='imagenet',
            include_top=False,
            input_shape=(config.IMG_SIZE, config.IMG_SIZE, 3)
        )
        base_model.trainable = False

        model = keras.Sequential([
            base_model,
            layers.GlobalAveragePooling2D(),
            layers.Dropout(0.4),
            layers.Dense(256, activation='relu'),
            layers.BatchNormalization(),
```

```
            layers.Dropout(0.3),
            layers.Dense(num_classes, activation='softmax')
        ])

    else:  # Simple CNN
        print("  Building Simple CNN (Fastest training)...")
        model = keras.Sequential([
            layers.Conv2D(32, (3, 3), activation='relu',
input_shape=(config.IMG_SIZE, config.IMG_SIZE, 3)),
            layers.MaxPooling2D(2, 2),
            layers.Conv2D(64, (3, 3), activation='relu'),
            layers.MaxPooling2D(2, 2),
            layers.Conv2D(128, (3, 3), activation='relu'),
            layers.MaxPooling2D(2, 2),
            layers.Flatten(),
            layers.Dropout(0.5),
            layers.Dense(128, activation='relu'),
            layers.Dense(num_classes, activation='softmax')
        ])

    return model

# ==================== MEDICINE RECOMMENDER (SAME)
====================
class FastMedicineRecommender:
    def __init__(self):
        self.recommendations = {
            'Bacterial_Spot': {
                'chemical': ['Copper hydroxide every 7-10 days',
'Streptomycin for severe cases'],
                'organic': ['Copper soap weekly', 'Neem oil spray'],
                'prevention': ['Use disease-free seeds', 'Avoid
overhead watering', 'Crop rotation']
            },
            'Early_Blight': {
                'chemical': ['Chlorothalonil every 7-10 days',
'Azoxystrobin systemic'],
                'organic': ['Copper fungicide', 'Baking soda spray'],
                'prevention': ['Remove lower leaves', 'Improve air
circulation', 'Mulch']
            },
            'Healthy': {
                'chemical': ['No treatment needed'],
                'organic': ['Continue organic practices'],
                'prevention': ['Regular monitoring', 'Proper
watering', 'Balanced fertilizer']
            },
            'Late_Blight': {
                'chemical': ['Chlorothalonil immediately', 'Metalaxyl
systemic'],
```

```
                'organic': ['Copper fungicide before rain', 'Potassium
bicarbonate'],
                'prevention': ['Destroy infected plants', 'Use
resistant varieties', 'Drip irrigation']
            },
            'Septoria_Leaf_Spot': {
                'chemical': ['Chlorothalonil weekly', 'Mancozeb
protective'],
                'organic': ['Copper soap', 'Sulfur spray'],
                'prevention': ['Remove infected leaves', 'Water at
base', 'Stake plants']
            }
        }

    def get_recommendation(self, disease, confidence):
        if disease not in self.recommendations:
            return "Disease not recognized"

        rec = self.recommendations[disease]
        return f"""
 DIAGNOSIS: {disease} ({confidence:.1%})

 CHEMICAL:
{chr(10).join(['• ' + t for t in rec['chemical']])}

 ORGANIC:
{chr(10).join(['• ' + t for t in rec['organic']])}

 PREVENTION:
{chr(10).join(['• ' + t for t in rec['prevention']])}
"""

# ==================== OPTIMIZED TRAINING ====================
def train_fast_model():
    """Fast training with optimizations"""

    # Enable mixed precision for GPU speedup
    try:
        tf.keras.mixed_precision.set_global_policy('mixed_float16')
        print(" Mixed precision enabled for faster training")
    except:
        print("⚠  Mixed precision not available, continuing normally")

    # Create data generators
    train_gen, val_gen = create_fast_data_generators()

    # Build model
    model = build_fast_model()

    # Optimizer with momentum for faster convergence
```

```
    optimizer = Adam(learning_rate=config.LEARNING_RATE)

    # For simple CNN, you can try SGD with momentum
    if config.MODEL_TYPE == "simple_cnn":
        optimizer = SGD(learning_rate=0.01, momentum=0.9)

    # Compile model
    model.compile(
        optimizer=optimizer,
        loss='categorical_crossentropy',
        metrics=['accuracy', 'precision', 'recall']
    )

    print(f"\n Model: {config.MODEL_TYPE.upper()}")
    print(f" Image size: {config.IMG_SIZE}x{config.IMG_SIZE}")
    print(f" Batch size: {config.BATCH_SIZE}")
    print(f"  Epochs: {config.EPOCHS}")
    print(f" Classes: {len(config.CLASS_NAMES)}")

    # Early stopping to prevent unnecessary epochs
    callbacks = [
        EarlyStopping(
            monitor='val_accuracy',
            patience=5,
            restore_best_weights=True,
            verbose=1
        ),
        ReduceLROnPlateau(
            monitor='val_loss',
            factor=0.5,
            patience=3,
            min_lr=1e-6,
            verbose=1
        ),
        ModelCheckpoint(
            filepath=config.MODEL_SAVE_PATH,
            monitor='val_accuracy',
            save_best_only=True,
            verbose=1
        )
    ]

    # Calculate steps per epoch
    train_steps = max(1, train_gen.samples // config.BATCH_SIZE)
    val_steps = max(1, val_gen.samples // config.BATCH_SIZE)

    print(f"\n Starting FAST training...")
    print(f"   Train steps per epoch: {train_steps}")
    print(f"   Val steps per epoch: {val_steps}")
```

```
    # Train
    history = model.fit(
        train_gen,
        steps_per_epoch=train_steps,
        epochs=config.EPOCHS,
        validation_data=val_gen,
        validation_steps=val_steps,
        callbacks=callbacks,
        verbose=1
    )

    return model, history, train_gen, val_gen

# ==================== QUICK VISUALIZATION ====================
def plot_quick_results(history, model, test_gen):
    """Quick visualization"""

    # Plot accuracy and loss
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))

    # Accuracy
    ax1.plot(history.history['accuracy'], label='Train')
    ax1.plot(history.history['val_accuracy'], label='Val')
    ax1.set_title('Accuracy')
    ax1.set_xlabel('Epoch')
    ax1.set_ylabel('Accuracy')
    ax1.legend()
    ax1.grid(True, alpha=0.3)

    # Loss
    ax2.plot(history.history['loss'], label='Train')
    ax2.plot(history.history['val_loss'], label='Val')
    ax2.set_title('Loss')
    ax2.set_xlabel('Epoch')
    ax2.set_ylabel('Loss')
    ax2.legend()
    ax2.grid(True, alpha=0.3)

    plt.tight_layout()
    plt.show()

    # Test accuracy
    test_loss, test_acc, test_prec, test_rec =
model.evaluate(test_gen, verbose=0)
    print(f"\n FINAL TEST RESULTS:")
    print(f"   Accuracy:  {test_acc:.2%}")
    print(f"   Precision: {test_prec:.2%}")
    print(f"   Recall:    {test_rec:.2%}")
    print(f"   Loss:      {test_loss:.4f}")
```

```python
# ==================== REAL-TIME PREDICTION ====================
def real_time_predict(image_path, model):
    """Ultra-fast prediction"""
    # Load image
    img = tf.keras.preprocessing.image.load_img(
        image_path,
        target_size=(config.IMG_SIZE, config.IMG_SIZE)
    )
    img_array = tf.keras.preprocessing.image.img_to_array(img)
    img_array = tf.expand_dims(img_array, 0) / 255.0

    # Predict
    start_time = tf.timestamp()
    predictions = model.predict(img_array, verbose=0)
    end_time = tf.timestamp()

    # Get results
    pred_idx = tf.argmax(predictions[0]).numpy()
    confidence = tf.reduce_max(predictions[0]).numpy()

    # Show image
    plt.figure(figsize=(6, 6))
    plt.imshow(img)
    plt.title(f"Prediction: {config.CLASS_NAMES[pred_idx]}\
nConfidence: {confidence:.1%}")
    plt.axis('off')
    plt.show()

    print(f"  Prediction time: {(end_time - start_time):.3f} seconds")

    return config.CLASS_NAMES[pred_idx], confidence

# ==================== MAIN - FAST EXECUTION ====================
print("="*60)
print(" ULTRA-FAST TOMATO DISEASE DETECTION")
print("="*60)

# Check for GPU
print("\n Checking hardware...")
print(f"TensorFlow version: {tf.__version__}")
print(f"GPU Available: {len(tf.config.list_physical_devices('GPU')) >
0}")
if tf.config.list_physical_devices('GPU'):
    print(f"GPU: {tf.config.list_physical_devices('GPU')[0]}")
    # Optimize GPU memory growth
    gpus = tf.config.experimental.list_physical_devices('GPU')
    if gpus:
        try:
            for gpu in gpus:
                tf.config.experimental.set_memory_growth(gpu, True)
```

```
        except RuntimeError as e:
            print(e)

# Model selection
print("\n MODEL OPTIONS:")
print("1. MobileNetV2 - Fastest (recommended)")
print("2. EfficientNetB0 - Balanced")
print("3. Simple CNN - Lightweight")
print(f"Selected: {config.MODEL_TYPE}")

# Train
model, history, train_gen, val_gen = train_fast_model()

# Results
plot_quick_results(history, model, val_gen)

# Initialize recommender
recommender = FastMedicineRecommender()

# Test with a random image
print("\n Testing with sample image...")

# Find any image in the dataset
found_image = None
for root, dirs, files in os.walk(config.DATA_PATH):
    for file in files:
        if file.lower().endswith(('.png', '.jpg', '.jpeg')):
            found_image = os.path.join(root, file)
            break
    if found_image:
        break

if found_image:
    print(f"Testing with: {os.path.basename(found_image)}")
    disease, confidence = real_time_predict(found_image, model)

    # Get recommendation
    print("\n TREATMENT RECOMMENDATION:")
    print(recommender.get_recommendation(disease, confidence))

# Save for future use
model.save(config.MODEL_SAVE_PATH)
print(f"\n Model saved to: {config.MODEL_SAVE_PATH}")

print("\n" + "="*60)
print("aTRAINING COMPLETE!")
print("="*60)

Mounted at /content/drive
============================================================
```

```
🚀 ULTRA-FAST TOMATO DISEASE DETECTION
============================================================

🔍 Checking hardware...
TensorFlow version: 2.19.0
GPU Available: True
GPU: PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')

📋 MODEL OPTIONS:
1. MobileNetV2 - Fastest (recommended)
2. EfficientNetB0 - Balanced
3. Simple CNN - Lightweight
Selected: mobilenetv2
⚡ Mixed precision enabled for faster training
📂 Loading dataset...
🔍 Found train/test folders
Found 5373 images belonging to 5 classes.
Found 1346 images belonging to 5 classes.
📊 Classes found: ['Bacterial_Spot', 'Early_Blight', 'Healthy',
'Late_Blight', 'Septoria_Leaf_Spot']
📊 Training samples: 5373
📊 Validation samples: 1346
  Building MobileNetV2 (Fastest)...
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/mobilenet_v2/
mobilenet_v2_weights_tf_dim_ordering_tf_kernels_0.35_224_no_top.h5
2019640/2019640 ━━━━━━━━━━━━━━━━━━━━ 1s 1us/step

📊 Model: MOBILENETV2
📊 Image size: 224x224
📊 Batch size: 64
  Epochs: 30
📊 Classes: 5

🚀 Starting FAST training...
   Train steps per epoch: 83
   Val steps per epoch: 21
Epoch 1/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 35s/step - accuracy: 0.5804 - loss:
1.0792 - precision: 0.6628 - recall: 0.4633
Epoch 1: val_accuracy improved from -inf to 0.84375, saving model
to /content/drive/MyDrive/tomato_model_fast.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.saving.save_model(model, 'my_model.keras')`.
```

```
 83/83 ━━━━━━━━━━━━━━━━━━━━ 3812s 46s/step - accuracy: 0.5818 - loss:
1.0757 - precision: 0.6642 - recall: 0.4652 - val_accuracy: 0.8438 -
val_loss: 0.4286 - val_precision: 0.8752 - val_recall: 0.8036 -
learning_rate: 0.0010
Epoch 2/30
 1/83 ━━━━━━━━━━━━━━━━━━━━ 2s 28ms/step - accuracy: 0.7500 - loss:
0.5804 - precision: 0.8246 - recall: 0.7344
Epoch 2: val_accuracy did not improve from 0.84375
83/83 ━━━━━━━━━━━━━━━━━━━━ 6s 76ms/step - accuracy: 0.7500 - loss:
0.5804 - precision: 0.8246 - recall: 0.7344 - val_accuracy: 0.8333 -
val_loss: 0.4451 - val_precision: 0.8689 - val_recall: 0.7991 -
learning_rate: 0.0010
Epoch 3/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.8115 - loss:
0.5110 - precision: 0.8448 - recall: 0.7681
Epoch 3: val_accuracy improved from 0.84375 to 0.87277, saving model
to /content/drive/MyDrive/tomato_model_fast.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.saving.save_model(model, 'my_model.keras')`.

 83/83 ━━━━━━━━━━━━━━━━━━━━ 106s 1s/step - accuracy: 0.8115 - loss:
0.5106 - precision: 0.8449 - recall: 0.7683 - val_accuracy: 0.8728 -
val_loss: 0.3517 - val_precision: 0.8968 - val_recall: 0.8467 -
learning_rate: 0.0010
Epoch 4/30
 1/83 ━━━━━━━━━━━━━━━━━━━━ 2s 26ms/step - accuracy: 0.8750 - loss:
0.3035 - precision: 0.9016 - recall: 0.8594
Epoch 4: val_accuracy did not improve from 0.87277
83/83 ━━━━━━━━━━━━━━━━━━━━ 6s 74ms/step - accuracy: 0.8750 - loss:
0.3035 - precision: 0.9016 - recall: 0.8594 - val_accuracy: 0.8720 -
val_loss: 0.3494 - val_precision: 0.8962 - val_recall: 0.8482 -
learning_rate: 0.0010
Epoch 5/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.8301 - loss:
0.4369 - precision: 0.8592 - recall: 0.8046
Epoch 5: val_accuracy improved from 0.87277 to 0.90179, saving model
to /content/drive/MyDrive/tomato_model_fast.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.saving.save_model(model, 'my_model.keras')`.

 83/83 ━━━━━━━━━━━━━━━━━━━━ 105s 1s/step - accuracy: 0.8301 - loss:
0.4368 - precision: 0.8592 - recall: 0.8046 - val_accuracy: 0.9018 -
```

```
val_loss: 0.2827 - val_precision: 0.9244 - val_recall: 0.8824 -
learning_rate: 0.0010
Epoch 6/30
 1/83 ━━━━━━━━━━━━━━━━━━━━ 2s 27ms/step - accuracy: 0.8750 - loss:
0.3679 - precision: 0.9032 - recall: 0.8750
Epoch 6: val_accuracy improved from 0.90179 to 0.90402, saving model
to /content/drive/MyDrive/tomato_model_fast.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.saving.save_model(model, 'my_model.keras')`.

 83/83 ━━━━━━━━━━━━━━━━━━━━ 7s 84ms/step - accuracy: 0.8750 - loss:
0.3679 - precision: 0.9032 - recall: 0.8750 - val_accuracy: 0.9040 -
val_loss: 0.2841 - val_precision: 0.9228 - val_recall: 0.8810 -
learning_rate: 0.0010
Epoch 7/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.8572 - loss:
0.3809 - precision: 0.8806 - recall: 0.8322
Epoch 7: val_accuracy did not improve from 0.90402
83/83 ━━━━━━━━━━━━━━━━━━━━ 106s 1s/step - accuracy: 0.8572 - loss:
0.3808 - precision: 0.8806 - recall: 0.8322 - val_accuracy: 0.8981 -
val_loss: 0.2836 - val_precision: 0.9136 - val_recall: 0.8810 -
learning_rate: 0.0010
Epoch 8/30
 1/83 ━━━━━━━━━━━━━━━━━━━━ 2s 30ms/step - accuracy: 0.8750 - loss:
0.4024 - precision: 0.8871 - recall: 0.8594
Epoch 8: ReduceLROnPlateau reducing learning rate to
0.0005000000237487257.

Epoch 8: val_accuracy did not improve from 0.90402
83/83 ━━━━━━━━━━━━━━━━━━━━ 7s 83ms/step - accuracy: 0.8750 - loss:
0.4024 - precision: 0.8871 - recall: 0.8594 - val_accuracy: 0.8996 -
val_loss: 0.2831 - val_precision: 0.9142 - val_recall: 0.8795 -
learning_rate: 0.0010
Epoch 9/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.8794 - loss:
0.3250 - precision: 0.9020 - recall: 0.8581
Epoch 9: val_accuracy improved from 0.90402 to 0.91295, saving model
to /content/drive/MyDrive/tomato_model_fast.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.saving.save_model(model, 'my_model.keras')`.
```

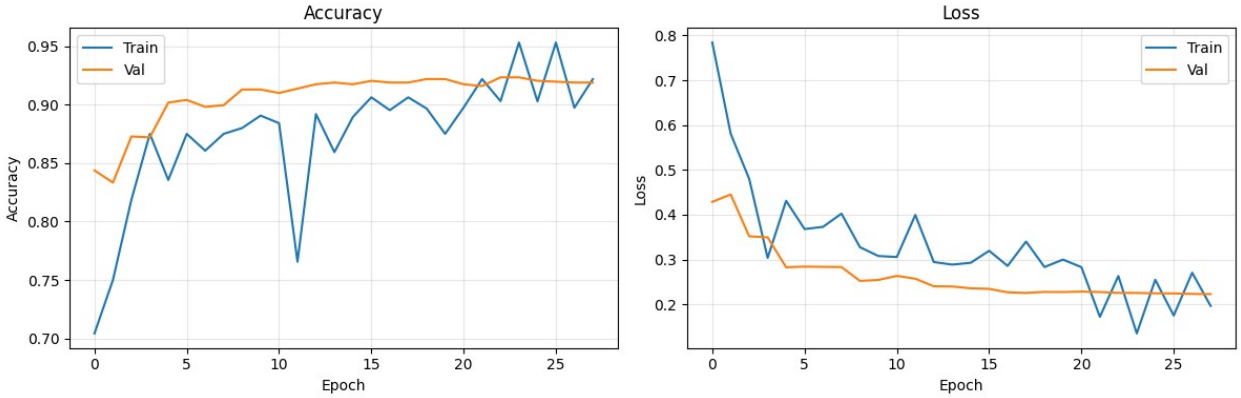
```
 83/83 ━━━━━━━━━━━━━━━━━━━━ 106s 1s/step - accuracy: 0.8795 - loss:
0.3251 - precision: 0.9020 - recall: 0.8580 - val_accuracy: 0.9129 -
val_loss: 0.2523 - val_precision: 0.9270 - val_recall: 0.8981 -
learning_rate: 5.0000e-04
Epoch 10/30
 1/83 ━━━━━━━━━━━━━━━━━━━━ 2s 33ms/step - accuracy: 0.8906 - loss:
0.3078 - precision: 0.9032 - recall: 0.8750
Epoch 10: val_accuracy did not improve from 0.91295
83/83 ━━━━━━━━━━━━━━━━━━━━ 7s 80ms/step - accuracy: 0.8906 - loss:
0.3078 - precision: 0.9032 - recall: 0.8750 - val_accuracy: 0.9129 -
val_loss: 0.2543 - val_precision: 0.9253 - val_recall: 0.8943 -
learning_rate: 5.0000e-04
Epoch 11/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.8846 - loss:
0.3072 - precision: 0.9068 - recall: 0.8674
Epoch 11: val_accuracy did not improve from 0.91295
83/83 ━━━━━━━━━━━━━━━━━━━━ 103s 1s/step - accuracy: 0.8846 - loss:
0.3072 - precision: 0.9068 - recall: 0.8674 - val_accuracy: 0.9100 -
val_loss: 0.2635 - val_precision: 0.9194 - val_recall: 0.8914 -
learning_rate: 5.0000e-04
Epoch 12/30
 1/83 ━━━━━━━━━━━━━━━━━━━━ 2s 31ms/step - accuracy: 0.7656 - loss:
0.3995 - precision: 0.8421 - recall: 0.7500
Epoch 12: ReduceLROnPlateau reducing learning rate to
0.0002500000118743628.

Epoch 12: val_accuracy improved from 0.91295 to 0.91369, saving model
to /content/drive/MyDrive/tomato_model_fast.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.saving.save_model(model, 'my_model.keras')`.

 83/83 ━━━━━━━━━━━━━━━━━━━━ 8s 92ms/step - accuracy: 0.7656 - loss:
0.3995 - precision: 0.8421 - recall: 0.7500 - val_accuracy: 0.9137 -
val_loss: 0.2571 - val_precision: 0.9231 - val_recall: 0.8936 -
learning_rate: 5.0000e-04
Epoch 13/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.8983 - loss:
0.2859 - precision: 0.9152 - recall: 0.8792
Epoch 13: val_accuracy improved from 0.91369 to 0.91741, saving model
to /content/drive/MyDrive/tomato_model_fast.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.saving.save_model(model, 'my_model.keras')`.
```

```
 83/83 ━━━━━━━━━━━━━━━━━━━━ 142s 2s/step - accuracy: 0.8982 - loss:
0.2860 - precision: 0.9151 - recall: 0.8791 - val_accuracy: 0.9174 -
val_loss: 0.2403 - val_precision: 0.9269 - val_recall: 0.9055 -
learning_rate: 2.5000e-04
Epoch 14/30
 1/83 ━━━━━━━━━━━━━━━━━━━━ 2s 27ms/step - accuracy: 0.8594 - loss:
0.2886 - precision: 0.9000 - recall: 0.8438
Epoch 14: val_accuracy improved from 0.91741 to 0.91890, saving model
to /content/drive/MyDrive/tomato_model_fast.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.saving.save_model(model, 'my_model.keras')`.

 83/83 ━━━━━━━━━━━━━━━━━━━━ 6s 76ms/step - accuracy: 0.8594 - loss:
0.2886 - precision: 0.9000 - recall: 0.8438 - val_accuracy: 0.9189 -
val_loss: 0.2398 - val_precision: 0.9269 - val_recall: 0.9062 -
learning_rate: 2.5000e-04
Epoch 15/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.8834 - loss:
0.3025 - precision: 0.9037 - recall: 0.8686
Epoch 15: val_accuracy did not improve from 0.91890
83/83 ━━━━━━━━━━━━━━━━━━━━ 102s 1s/step - accuracy: 0.8835 - loss:
0.3024 - precision: 0.9038 - recall: 0.8686 - val_accuracy: 0.9174 -
val_loss: 0.2357 - val_precision: 0.9274 - val_recall: 0.9033 -
learning_rate: 2.5000e-04
Epoch 16/30
 1/83 ━━━━━━━━━━━━━━━━━━━━ 2s 27ms/step - accuracy: 0.9062 - loss:
0.3195 - precision: 0.9062 - recall: 0.9062
Epoch 16: val_accuracy improved from 0.91890 to 0.92039, saving model
to /content/drive/MyDrive/tomato_model_fast.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.saving.save_model(model, 'my_model.keras')`.

 83/83 ━━━━━━━━━━━━━━━━━━━━ 7s 88ms/step - accuracy: 0.9062 - loss:
0.3195 - precision: 0.9062 - recall: 0.9062 - val_accuracy: 0.9204 -
val_loss: 0.2345 - val_precision: 0.9270 - val_recall: 0.9070 -
learning_rate: 2.5000e-04
Epoch 17/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.8889 - loss:
0.3066 - precision: 0.9055 - recall: 0.8695
Epoch 17: val_accuracy did not improve from 0.92039
83/83 ━━━━━━━━━━━━━━━━━━━━ 103s 1s/step - accuracy: 0.8890 - loss:
0.3063 - precision: 0.9055 - recall: 0.8696 - val_accuracy: 0.9189 -
```

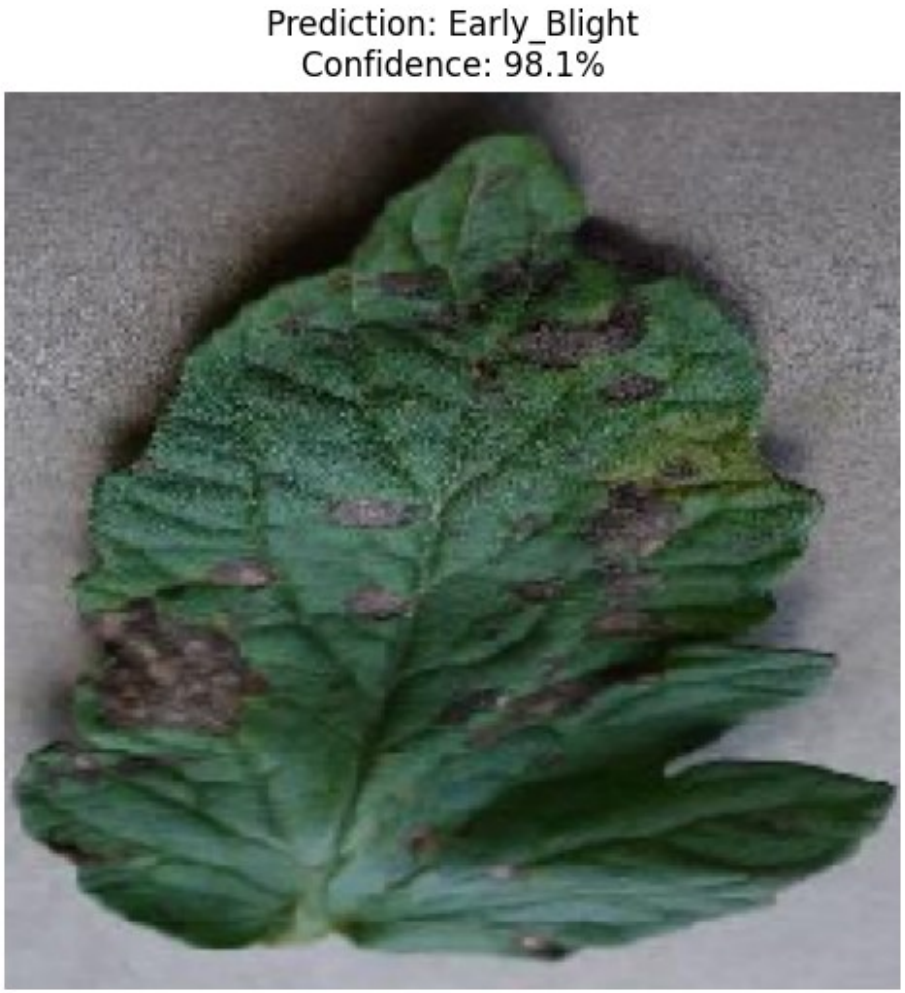
```
val_loss: 0.2268 - val_precision: 0.9329 - val_recall: 0.9100 -
learning_rate: 2.5000e-04
Epoch 18/30
 1/83 ━━━━━━━━━━━━━━━━━━━━ 2s 28ms/step - accuracy: 0.9062 - loss:
0.3401 - precision: 0.9048 - recall: 0.8906
Epoch 18: val_accuracy did not improve from 0.92039
83/83 ━━━━━━━━━━━━━━━━━━━━ 6s 70ms/step - accuracy: 0.9062 - loss:
0.3401 - precision: 0.9048 - recall: 0.8906 - val_accuracy: 0.9189 -
val_loss: 0.2254 - val_precision: 0.9322 - val_recall: 0.9100 -
learning_rate: 2.5000e-04
Epoch 19/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.8994 - loss:
0.2808 - precision: 0.9165 - recall: 0.8837
Epoch 19: val_accuracy improved from 0.92039 to 0.92188, saving model
to /content/drive/MyDrive/tomato_model_fast.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.saving.save_model(model, 'my_model.keras')`.

 83/83 ━━━━━━━━━━━━━━━━━━━━ 104s 1s/step - accuracy: 0.8993 - loss:
0.2808 - precision: 0.9165 - recall: 0.8837 - val_accuracy: 0.9219 -
val_loss: 0.2277 - val_precision: 0.9282 - val_recall: 0.9137 -
learning_rate: 2.5000e-04
Epoch 20/30
 1/83 ━━━━━━━━━━━━━━━━━━━━ 2s 27ms/step - accuracy: 0.8750 - loss:
0.2998 - precision: 0.9138 - recall: 0.8281
Epoch 20: val_accuracy did not improve from 0.92188
83/83 ━━━━━━━━━━━━━━━━━━━━ 6s 71ms/step - accuracy: 0.8750 - loss:
0.2998 - precision: 0.9138 - recall: 0.8281 - val_accuracy: 0.9219 -
val_loss: 0.2273 - val_precision: 0.9282 - val_recall: 0.9137 -
learning_rate: 2.5000e-04
Epoch 21/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.8965 - loss:
0.2869 - precision: 0.9116 - recall: 0.8786
Epoch 21: ReduceLROnPlateau reducing learning rate to
0.00012500000593718141.

Epoch 21: val_accuracy did not improve from 0.92188
83/83 ━━━━━━━━━━━━━━━━━━━━ 102s 1s/step - accuracy: 0.8965 - loss:
0.2869 - precision: 0.9116 - recall: 0.8786 - val_accuracy: 0.9174 -
val_loss: 0.2284 - val_precision: 0.9248 - val_recall: 0.9055 -
learning_rate: 2.5000e-04
Epoch 22/30
 1/83 ━━━━━━━━━━━━━━━━━━━━ 2s 27ms/step - accuracy: 0.9219 - loss:
0.1721 - precision: 0.9355 - recall: 0.9062
Epoch 22: val_accuracy did not improve from 0.92188
83/83 ━━━━━━━━━━━━━━━━━━━━ 6s 72ms/step - accuracy: 0.9219 - loss:
```

```
0.1721 - precision: 0.9355 - recall: 0.9062 - val_accuracy: 0.9159 -
val_loss: 0.2273 - val_precision: 0.9263 - val_recall: 0.9077 -
learning_rate: 1.2500e-04
Epoch 23/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.9065 - loss:
0.2575 - precision: 0.9218 - recall: 0.8914
Epoch 23: val_accuracy improved from 0.92188 to 0.92336, saving model
to /content/drive/MyDrive/tomato_model_fast.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.saving.save_model(model, 'my_model.keras')`.

 83/83 ━━━━━━━━━━━━━━━━━━━━ 102s 1s/step - accuracy: 0.9065 - loss:
0.2575 - precision: 0.9217 - recall: 0.8913 - val_accuracy: 0.9234 -
val_loss: 0.2255 - val_precision: 0.9316 - val_recall: 0.9122 -
learning_rate: 1.2500e-04
Epoch 24/30
 1/83 ━━━━━━━━━━━━━━━━━━━━ 2s 27ms/step - accuracy: 0.9531 - loss:
0.1350 - precision: 0.9839 - recall: 0.9531
Epoch 24: ReduceLROnPlateau reducing learning rate to
6.25000029685907e-05.

Epoch 24: val_accuracy did not improve from 0.92336
83/83 ━━━━━━━━━━━━━━━━━━━━ 6s 72ms/step - accuracy: 0.9531 - loss:
0.1350 - precision: 0.9839 - recall: 0.9531 - val_accuracy: 0.9234 -
val_loss: 0.2254 - val_precision: 0.9310 - val_recall: 0.9137 -
learning_rate: 1.2500e-04
Epoch 25/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.9059 - loss:
0.2521 - precision: 0.9230 - recall: 0.8934
Epoch 25: val_accuracy did not improve from 0.92336
83/83 ━━━━━━━━━━━━━━━━━━━━ 104s 1s/step - accuracy: 0.9058 - loss:
0.2522 - precision: 0.9229 - recall: 0.8933 - val_accuracy: 0.9204 -
val_loss: 0.2246 - val_precision: 0.9297 - val_recall: 0.9144 -
learning_rate: 6.2500e-05
Epoch 26/30
 1/83 ━━━━━━━━━━━━━━━━━━━━ 2s 30ms/step - accuracy: 0.9531 - loss:
0.1750 - precision: 0.9672 - recall: 0.9219
Epoch 26: val_accuracy did not improve from 0.92336
83/83 ━━━━━━━━━━━━━━━━━━━━ 6s 77ms/step - accuracy: 0.9531 - loss:
0.1750 - precision: 0.9672 - recall: 0.9219 - val_accuracy: 0.9196 -
val_loss: 0.2243 - val_precision: 0.9289 - val_recall: 0.9144 -
learning_rate: 6.2500e-05
Epoch 27/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.8982 - loss:
0.2760 - precision: 0.9124 - recall: 0.8837
Epoch 27: val_accuracy did not improve from 0.92336
```

```
83/83 ━━━━━━━━━━━━━━━━━━━━ 106s 1s/step - accuracy: 0.8982 - loss:
0.2759 - precision: 0.9124 - recall: 0.8837 - val_accuracy: 0.9189 -
val_loss: 0.2232 - val_precision: 0.9287 - val_recall: 0.9115 -
learning_rate: 6.2500e-05
Epoch 28/30
 1/83 ━━━━━━━━━━━━━━━━━━━━ 2s 27ms/step - accuracy: 0.9219 - loss:
0.1964 - precision: 0.9355 - recall: 0.9062
Epoch 28: val_accuracy did not improve from 0.92336
83/83 ━━━━━━━━━━━━━━━━━━━━ 6s 73ms/step - accuracy: 0.9219 - loss:
0.1964 - precision: 0.9355 - recall: 0.9062 - val_accuracy: 0.9189 -
val_loss: 0.2228 - val_precision: 0.9280 - val_recall: 0.9115 -
learning_rate: 6.2500e-05
Epoch 28: early stopping
Restoring model weights from the end of the best epoch: 23.
```




```
 FINAL TEST RESULTS:
   Accuracy:  92.35%
   Precision: 93.17%
   Recall:    91.23%
   Loss:      0.2253

 Testing with sample image...
Testing with: 2fa6cd46-eb76-481e-b15a-44c858f705b3___RS_Erly.B
9532.JPG
```




```
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.saving.save_model(model, 'my_model.keras')`.

  Prediction time: 5.668 seconds

 TREATMENT RECOMMENDATION:

 DIAGNOSIS: Early_Blight (98.1%)

 CHEMICAL:
• Chlorothalonil every 7-10 days
• Azoxystrobin systemic

 ORGANIC:
• Copper fungicide
• Baking soda spray
```

```
 PREVENTION:
• Remove lower leaves
• Improve air circulation
• Mulch


 Model saved to: /content/drive/MyDrive/tomato_model_fast.h5

============================================================
aTRAINING COMPLETE!
============================================================
```

```
from google.colab import drive
drive.mount('/content/drive')
```