tomatoplantdiseasedetection-2

March 20, 2025

1 Tomato Plant Disease Detection Using Deep Learning - CNN

1.1 Importing The Libraries

[3]:
```
import cv2
import os
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from tensorflow.keras import models, layers
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.regularizers import l1, l2
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Dense, Flatten, Dropout,␣
 ↪GlobalAveragePooling2D, AveragePooling2D, MaxPooling2D, BatchNormalization
# Various types of layers for building neural networks
from tensorflow.keras.applications import DenseNet121, EfficientNetB4,␣
 ↪Xception, VGG16, VGG19
```

1.2 Data Preprocessing

1.2.1 Training Image preprocessing

[4]:
```
train_data = tf.keras.utils.image_dataset_from_directory(
    "/kaggle/input/tomatodiseasedleaves/tomato/train",
    labels="inferred",
    label_mode="categorical",
    class_names=None,
    color_mode="rgb",
    batch_size=32,
    image_size=(128, 128),
    shuffle=True,
    seed=None,
    validation_split=None,
```

```
    subset=None,
    interpolation="bilinear",
    follow_links=False,
    crop_to_aspect_ratio=False)

train_data = train_data.map(lambda x, y: (x / 255.0, y))
```

```
Found 11969 files belonging to 13 classes.
```

1.2.2 Validation Image Preprocessing

[5]:
```
val_data = tf.keras.preprocessing.image_dataset_from_directory(
    "/kaggle/input/tomatodiseasedleaves/tomato/val",
    labels="inferred",
    label_mode="categorical",
    class_names=None,
    color_mode="rgb",
    batch_size=32,
    image_size=(128, 128),
    shuffle=True,
    seed=None,
    validation_split=None,
    subset=None,
    interpolation="bilinear",
    follow_links=False,
    crop_to_aspect_ratio=False
)
val_data = val_data.map(lambda x, y: (x / 255.0, y))
```

```
Found 1479 files belonging to 13 classes.
```

[6]:
```
import tensorflow as tf

def preprocess_image(image, label):
    # Ensure image has 3 channels
    if image.shape[-1] == 1:  # if the image is grayscale
        image = tf.image.grayscale_to_rgb(image)
    # Resize the image to a fixed size
    resized_image = tf.image.resize(image, (256, 256))
    return resized_image, label
```

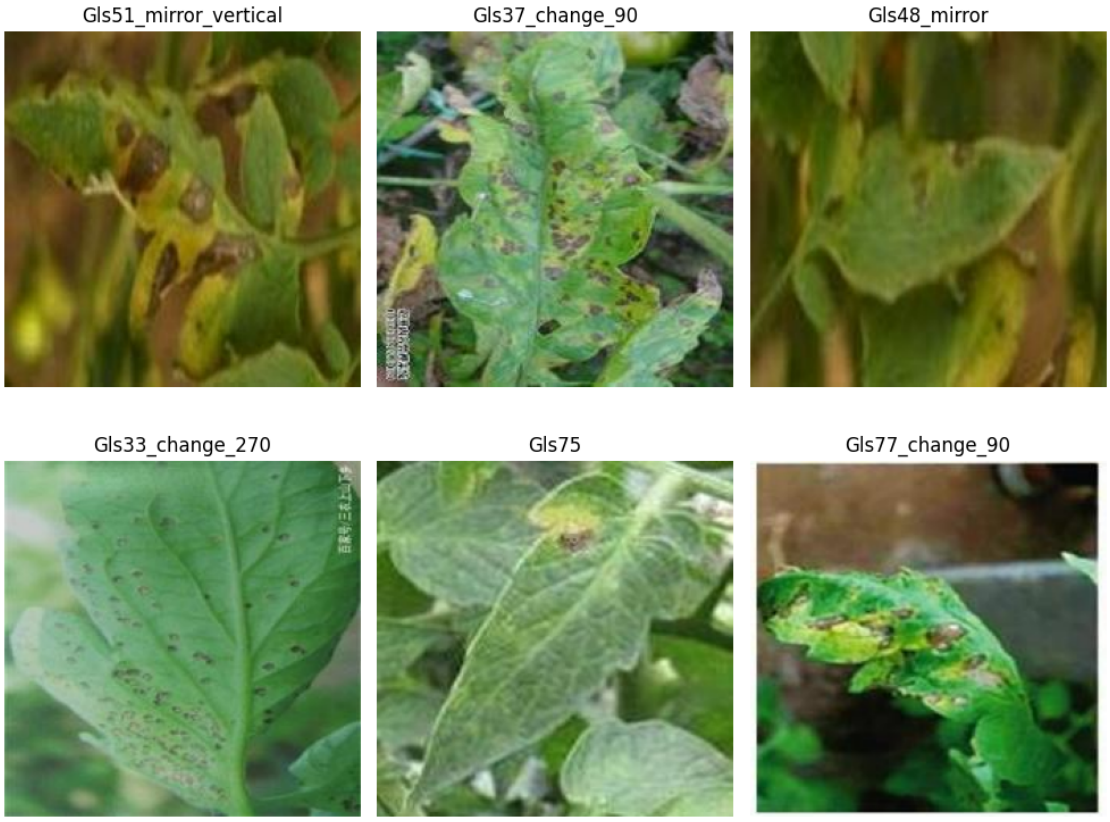
1.3 Visualizing The Data

1.3.1 Black Mold

[7]:
```
path = "/kaggle/input/tomatodiseasedleaves/tomato/train/Black mold"
image_files = [f for f in os.listdir(path) if os.path.isfile(os.path.join(path,␣
 ↪f))]

# Display the first 6 images with their labels
fig, axs = plt.subplots(2, 3, figsize=(10, 8))

for i in range(6):
    image_file = image_files[i]
    label = image_file.split('.')[0]

    img_path = os.path.join(path, image_file)
    img = mpimg.imread(img_path)
    ax = axs[i // 3, i % 3]
    ax.imshow(img)
    ax.axis('off')
    ax.set_title(label)

plt.tight_layout()
plt.show()
```

1.3.2 Gray Spot

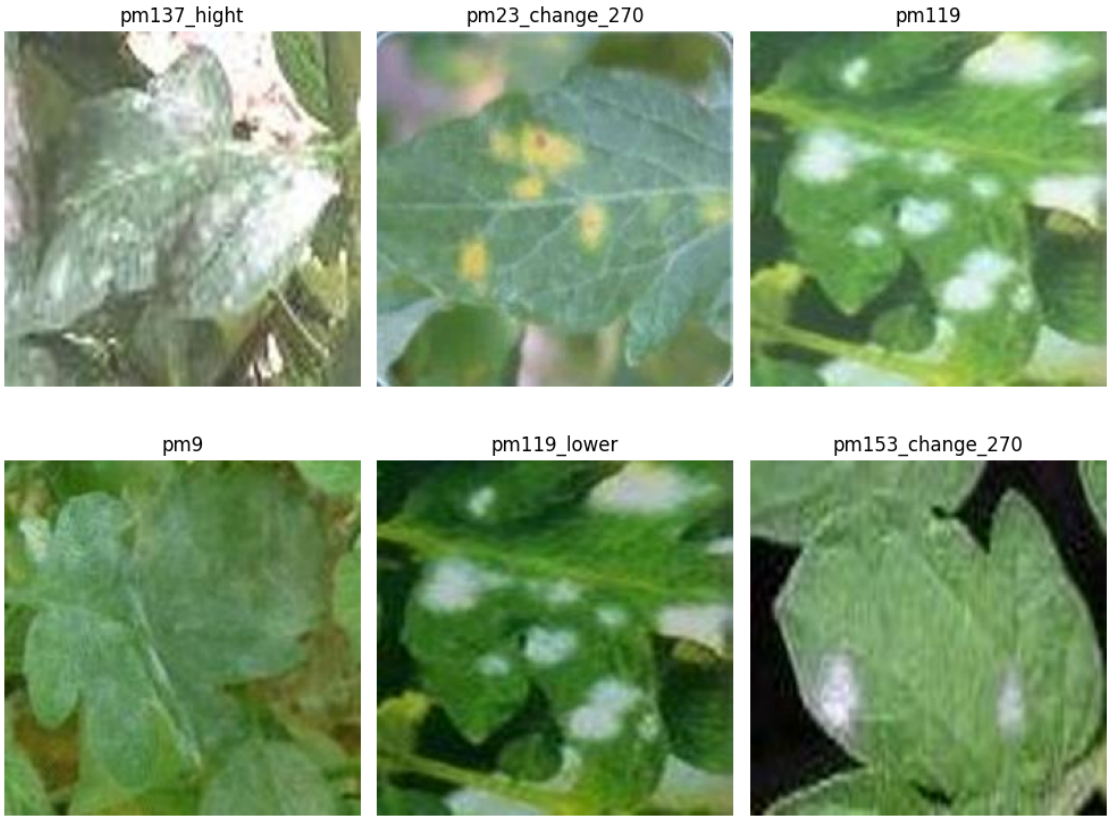
[8]:
```
path = "/kaggle/input/tomatodiseasedleaves/tomato/train/Gray spot"
image_files = [f for f in os.listdir(path) if os.path.isfile(os.path.join(path,␣
 ↪f))]

# Display the first 6 images with their labels
fig, axs = plt.subplots(2, 3, figsize=(10, 8))

for i in range(6):
    image_file = image_files[i]
    label = image_file.split('.')[0]

    img_path = os.path.join(path, image_file)
    img = mpimg.imread(img_path)
    ax = axs[i // 3, i % 3]
    ax.imshow(img)
    ax.axis('off')
    ax.set_title(label)

plt.tight_layout()
plt.show()
```



1.3.3 Powdery Mildew

[9]:
```
path = "/kaggle/input/tomatodiseasedleaves/tomato/train/powdery mildew"
image_files = [f for f in os.listdir(path) if os.path.isfile(os.path.join(path,␣
 ↪f))]

# Display the first 6 images with their labels
fig, axs = plt.subplots(2, 3, figsize=(10, 8))

for i in range(6):
    image_file = image_files[i]
    label = image_file.split('.')[0]

    img_path = os.path.join(path, image_file)
    img = mpimg.imread(img_path)
    ax = axs[i // 3, i % 3]
    ax.imshow(img)
    ax.axis('off')
    ax.set_title(label)

plt.tight_layout()
```

```
plt.show()
```



1.3.4 Yellow Leaf Curl Virus

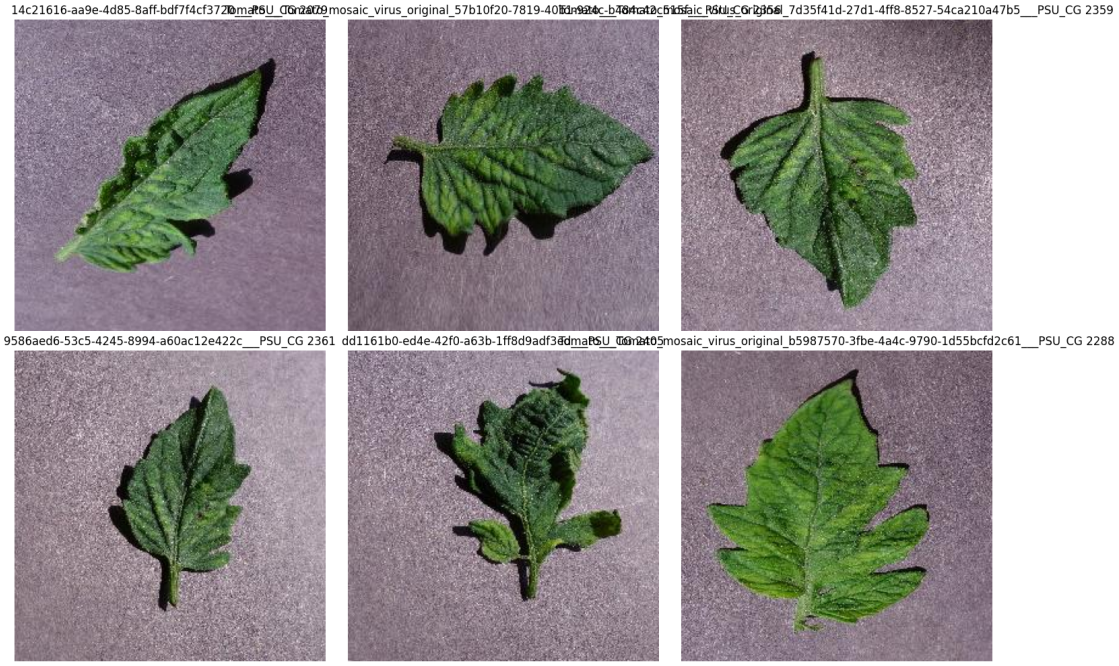
[10]:
```
path = "/kaggle/input/tomatodiseasedleaves/tomato/train/
 ↪Tomato___Tomato_Yellow_Leaf_Curl_Virus"
image_files = [f for f in os.listdir(path) if os.path.isfile(os.path.join(path,␣
 ↪f))]

# Display the first 6 images with their labels
fig, axs = plt.subplots(2, 3, figsize=(15, 10))

for i in range(6):
    image_file = image_files[i]
    label = image_file.split('.')[0]

    img_path = os.path.join(path, image_file)
    img = mpimg.imread(img_path)
    ax = axs[i // 3, i % 3]
    ax.imshow(img)
    ax.axis('off')
```

```
    ax.set_title(label)

plt.tight_layout()
plt.show()
```

1.3.5 Mosaic Virus

[11]:
```
path = "/kaggle/input/tomatodiseasedleaves/tomato/train/
 ↪Tomato___Tomato_mosaic_virus"

image_files = [f for f in os.listdir(path) if os.path.isfile(os.path.join(path,␣
 ↪f))]

# Display the first 6 images with their labels
fig, axs = plt.subplots(2, 3, figsize=(15, 10))

for i in range(6):
    image_file = image_files[i]
    label = image_file.split('.')[0]

    img_path = os.path.join(path, image_file)
    img = mpimg.imread(img_path)
    ax = axs[i // 3, i % 3]
```

```
    ax.imshow(img)
    ax.axis('off')
    ax.set_title(label)

plt.tight_layout()
plt.show()
```



1.3.6 Target Spots

[12]:
```
path = "/kaggle/input/tomatodiseasedleaves/tomato/train/Tomato___Target_Spot"

image_files = [f for f in os.listdir(path) if os.path.isfile(os.path.join(path,␣
 ↪f))]

# Display the first 6 images with their labels
fig, axs = plt.subplots(2, 3, figsize=(15, 10))

for i in range(6):
    image_file = image_files[i]
    label = image_file.split('.')[0]

    img_path = os.path.join(path, image_file)
    img = mpimg.imread(img_path)
    ax = axs[i // 3, i % 3]
    ax.imshow(img)
```

```
    ax.axis('off')
    ax.set_title(label)

plt.tight_layout()
plt.show()
```

1.3.7 Spider Mites Two-spotted spider mite

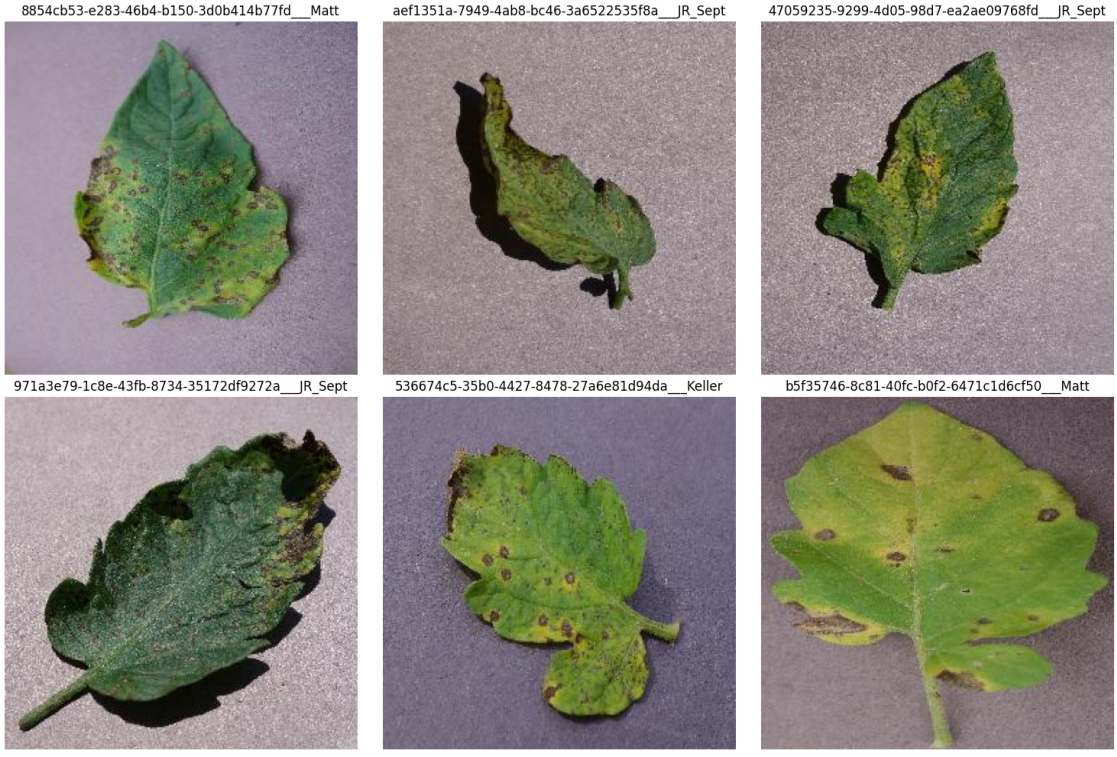
[13]:
```
path = "/kaggle/input/tomatodiseasedleaves/tomato/train/Tomato___Spider_mites␣
 ↪Two-spotted_spider_mite"

image_files = [f for f in os.listdir(path) if os.path.isfile(os.path.join(path,␣
 ↪f))]

# Display the first 6 images with their labels
fig, axs = plt.subplots(2, 3, figsize=(15, 10))

for i in range(6):
    image_file = image_files[i]
    label = image_file.split('.')[0]

    img_path = os.path.join(path, image_file)
```

```
    img = mpimg.imread(img_path)
    ax = axs[i // 3, i % 3]
    ax.imshow(img)
    ax.axis('off')
    ax.set_title(label)

plt.tight_layout()
plt.show()
```

1.3.8 Septoria Leaf Spot

[14]:
```
path = "/kaggle/input/tomatodiseasedleaves/tomato/train/
 ↪Tomato___Septoria_leaf_spot"

image_files = [f for f in os.listdir(path) if os.path.isfile(os.path.join(path,␣
 ↪f))]

# Display the first 6 images with their labels
fig, axs = plt.subplots(2, 3, figsize=(15, 10))

for i in range(6):
    image_file = image_files[i]
```

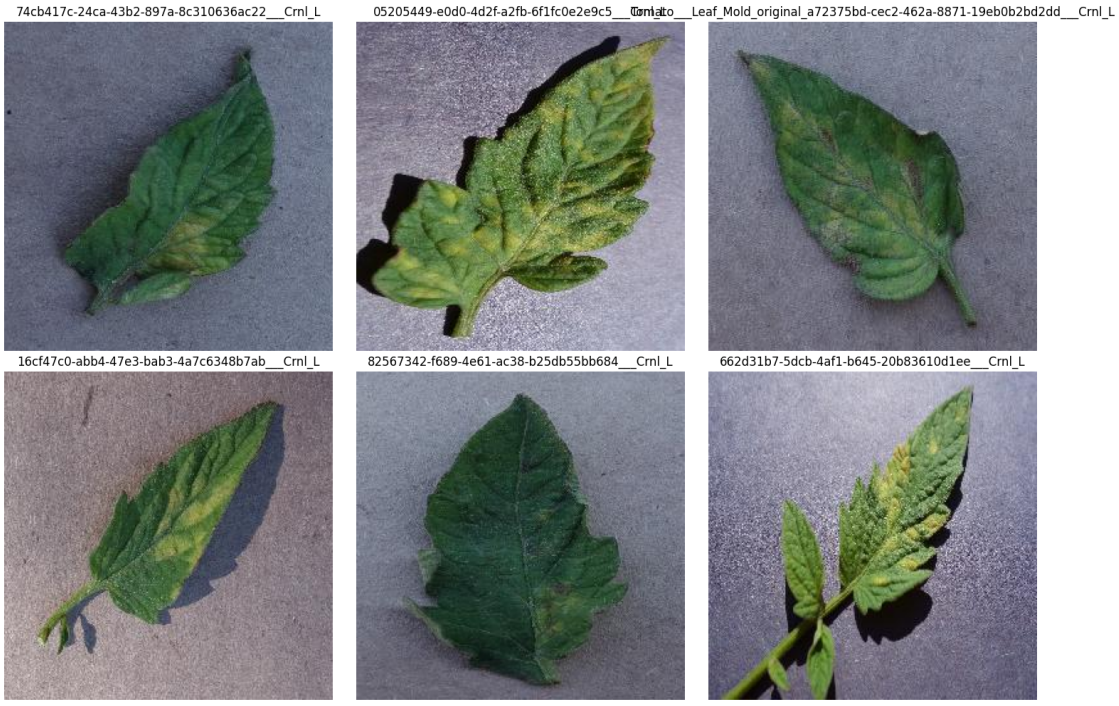
```
    label = image_file.split('.')[0]

    img_path = os.path.join(path, image_file)
    img = mpimg.imread(img_path)
    ax = axs[i // 3, i % 3]
    ax.imshow(img)
    ax.axis('off')
    ax.set_title(label)

plt.tight_layout()
plt.show()
```



1.3.9 Leaf Mold

[15]:
```
path = "/kaggle/input/tomatodiseasedleaves/tomato/train/Tomato___Leaf_Mold"

image_files = [f for f in os.listdir(path) if os.path.isfile(os.path.join(path,␣
 ↪f))]

# Display the first 6 images with their labels
fig, axs = plt.subplots(2, 3, figsize=(15, 10))

for i in range(6):
```

```
    image_file = image_files[i]
    label = image_file.split('.')[0]

    img_path = os.path.join(path, image_file)
    img = mpimg.imread(img_path)
    ax = axs[i // 3, i % 3]
    ax.imshow(img)
    ax.axis('off')
    ax.set_title(label)

plt.tight_layout()
plt.show()
```



1.3.10 Late Blight

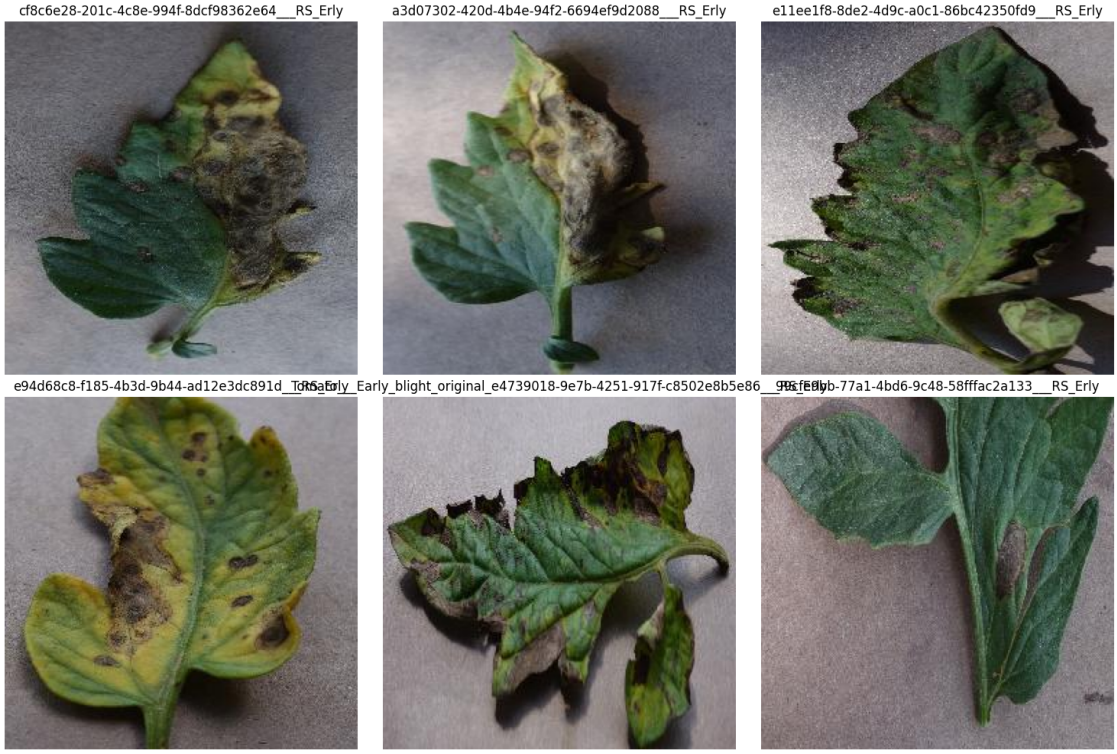
[16]:
```
path = "/kaggle/input/tomatodiseasedleaves/tomato/train/Tomato___Late_blight"

image_files = [f for f in os.listdir(path) if os.path.isfile(os.path.join(path,␣
 ↪f))]

# Display the first 6 images with their labels
fig, axs = plt.subplots(2, 3, figsize=(15, 10))

for i in range(6):
```

```
    image_file = image_files[i]
    label = image_file.split('.')[0]

    img_path = os.path.join(path, image_file)
    img = mpimg.imread(img_path)
    ax = axs[i // 3, i % 3]
    ax.imshow(img)
    ax.axis('off')
    ax.set_title(label)

plt.tight_layout()
plt.show()
```

1.3.11 Early Blight

[17]:
```
path = "/kaggle/input/tomatodiseasedleaves/tomato/train/Tomato___Early_blight"

image_files = [f for f in os.listdir(path) if os.path.isfile(os.path.join(path,␣
 ↪f))]

# Display the first 6 images with their labels
fig, axs = plt.subplots(2, 3, figsize=(15, 10))
```

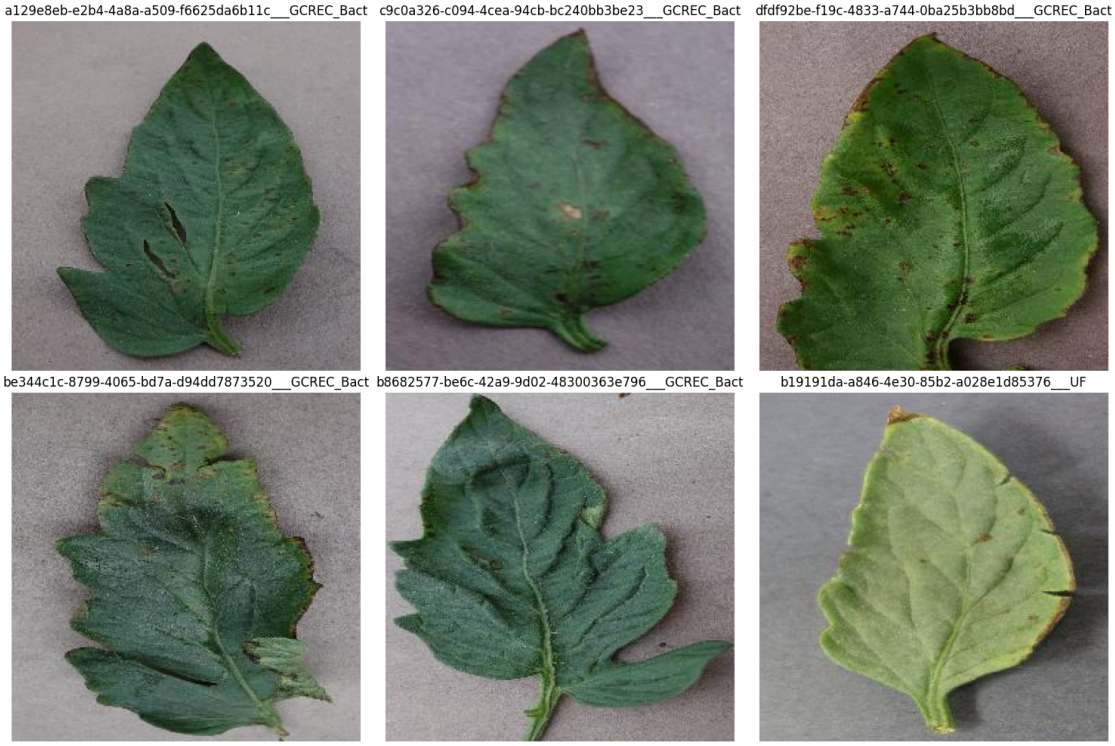
```
for i in range(6):
    image_file = image_files[i]
    label = image_file.split('.')[0]

    img_path = os.path.join(path, image_file)
    img = mpimg.imread(img_path)
    ax = axs[i // 3, i % 3]
    ax.imshow(img)
    ax.axis('off')
    ax.set_title(label)

plt.tight_layout()
plt.show()
```



1.3.12 Bacterial Spot

[18]:
```
path = "/kaggle/input/tomatodiseasedleaves/tomato/train/Tomato___Bacterial_spot"

image_files = [f for f in os.listdir(path) if os.path.isfile(os.path.join(path,␣
↪f))]

# Display the first 6 images with their labels
```

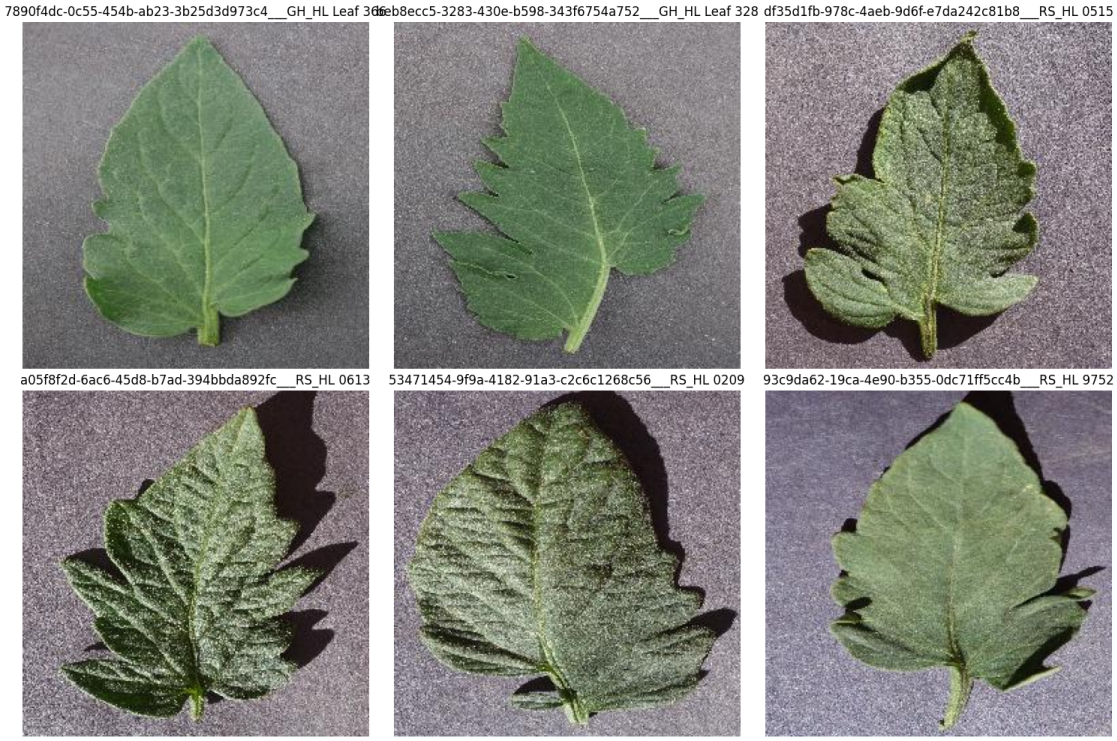
```
fig, axs = plt.subplots(2, 3, figsize=(15, 10))

for i in range(6):
    image_file = image_files[i]
    label = image_file.split('.')[0]

    img_path = os.path.join(path, image_file)
    img = mpimg.imread(img_path)
    ax = axs[i // 3, i % 3]
    ax.imshow(img)
    ax.axis('off')
    ax.set_title(label)

plt.tight_layout()
plt.show()
```



1.3.13 Tomato Healthy Leaves

[19]:
```
path = "/kaggle/input/tomatodiseasedleaves/tomato/train/Tomato___healthy"

image_files = [f for f in os.listdir(path) if os.path.isfile(os.path.join(path,␣
 ↪f))]
```

```
# Display the first 6 images with their labels
fig, axs = plt.subplots(2, 3, figsize=(15, 10))

for i in range(6):
    image_file = image_files[i]
    label = image_file.split('.')[0]

    img_path = os.path.join(path, image_file)
    img = mpimg.imread(img_path)
    ax = axs[i // 3, i % 3]
    ax.imshow(img)
    ax.axis('off')
    ax.set_title(label)

plt.tight_layout()
plt.show()
```



1.4 Model Building

1.4.1 Creating a Layer for Resizing and Normalization

Before we feed our images to network, we should be resizing it to the desired size. Moreover, to improve model performance, we should normalize the image pixel value (keeping them in range 0 and 1 by dividing by 256). This should happen while training as well as inference. Hence we can

add that as a layer in our Sequential Model.

This will be useful when we are done with the training and start using the model for predictions. At that time somone can supply an image that is not (256,256) and this layer will resize it

[20]:
```
IMAGE_SIZE = 256
from tensorflow.keras.layers import Resizing, Rescaling
resize_and_rescale = tf.keras.Sequential([
  Resizing(IMAGE_SIZE, IMAGE_SIZE),
  Rescaling(1./255)
])
```

1.5 Model Architecture

1.5.1 Data Augmentation

This boosts the accuracy of our model by augmenting the data.

[21]:
```
from tensorflow.keras.layers import RandomFlip, RandomRotation
data_augmentation = tf.keras.Sequential([
  RandomFlip("horizontal_and_vertical"),
  RandomRotation(0.2),
])
```

[22]:
```
train_ds = train_data.map(
    lambda x, y: (data_augmentation(x, training=True), y)
).prefetch(buffer_size=tf.data.AUTOTUNE)
```

[23]:
```
import tensorflow as tf

# Resize function
def resize_images(image, label):
    image = tf.image.resize(image, [256, 256])  # Resize to 256x256
    return image, label

# Apply resizing to your dataset
train_data = train_data.map(resize_images)
val_data = val_data.map(resize_images)
```

1.5.2 Adding L2-Regularization

[24]:
```
from tensorflow.keras import models, layers
from tensorflow.keras.layers import Input

BATCH_SIZE = 32
IMAGE_SIZE = 256
CHANNELS = 3
n_classes = 13
```

```
input_shape = (IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
model = models.Sequential([
    Input(shape=input_shape),
    resize_and_rescale,
    layers.Conv2D(32, kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(n_classes, activation='softmax'),
])
```

[25]:
```
conv_base = DenseNet121(
    weights='imagenet',
    include_top = False,
    input_shape=(256,256,3),
    pooling='avg'
)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/densenet/densenet121_weights_tf_dim_ordering_tf_kernels_notop.h5
29084464/29084464              0s
0us/step
```

[26]:
```
conv_base.trainable = False
```

[27]:
```
model = Sequential()
model.add(conv_base)
model.add(BatchNormalization())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.35))
model.add(BatchNormalization())
model.add(Dense(120, activation='relu'))
model.add(Dense(13, activation='softmax'))
```

1.6 Compiling the Model

We use adam Optimizer, SparseCategoricalCrossentropy for losses, accuracy as a metric

[28]:
```
model.compile(optimizer=Adam(), loss='categorical_crossentropy',␣
↪metrics=['accuracy'])
```

[30]:
```
import tensorflow as tf
import numpy as np
import random
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

#  1. Set a fixed seed for reproducibility
SEED = 42
tf.random.set_seed(SEED)
np.random.seed(SEED)
random.seed(SEED)

#  2. Define callbacks for stability
early_stopping = EarlyStopping(monitor="val_loss", patience=5,␣
↪restore_best_weights=True)
checkpoint_callback = ModelCheckpoint("best_model.keras",␣
↪monitor="val_accuracy", save_best_only=True, mode="max", verbose=1)

#  3. Train the model with controlled randomness
history = model.fit(
    train_data,
    epochs=100,
    validation_data=val_data,
    callbacks=[early_stopping, checkpoint_callback]
)

#  4. Load the best saved model for stable evaluation
model = tf.keras.models.load_model("best_model.keras")
```

```
Epoch 1/100
375/375            0s 119ms/step -
accuracy: 0.6436 - loss: 1.0839
Epoch 1: val_accuracy improved from -inf to 0.89520, saving model to
best_model.keras
375/375            94s 174ms/step -
accuracy: 0.6439 - loss: 1.0828 - val_accuracy: 0.8952 - val_loss: 0.3139
Epoch 2/100
374/375            0s 97ms/step -
accuracy: 0.8903 - loss: 0.3084
Epoch 2: val_accuracy improved from 0.89520 to 0.92765, saving model to
best_model.keras
375/375            42s 113ms/step -
```

```
accuracy: 0.8904 - loss: 0.3082 - val_accuracy: 0.9277 - val_loss: 0.1968
Epoch 3/100
374/375            0s 94ms/step -
accuracy: 0.9211 - loss: 0.2198
Epoch 3: val_accuracy improved from 0.92765 to 0.94253, saving model to
best_model.keras
375/375            41s 108ms/step -
accuracy: 0.9211 - loss: 0.2198 - val_accuracy: 0.9425 - val_loss: 0.1588
Epoch 4/100
374/375            0s 95ms/step -
accuracy: 0.9415 - loss: 0.1767
Epoch 4: val_accuracy improved from 0.94253 to 0.95199, saving model to
best_model.keras
375/375            41s 109ms/step -
accuracy: 0.9415 - loss: 0.1767 - val_accuracy: 0.9520 - val_loss: 0.1419
Epoch 5/100
374/375            0s 102ms/step -
accuracy: 0.9516 - loss: 0.1336
Epoch 5: val_accuracy improved from 0.95199 to 0.95605, saving model to
best_model.keras
375/375            43s 116ms/step -
accuracy: 0.9516 - loss: 0.1336 - val_accuracy: 0.9561 - val_loss: 0.1371
Epoch 6/100
374/375            0s 95ms/step -
accuracy: 0.9554 - loss: 0.1303
Epoch 6: val_accuracy did not improve from 0.95605
375/375            40s 107ms/step -
accuracy: 0.9554 - loss: 0.1302 - val_accuracy: 0.9459 - val_loss: 0.1486
Epoch 7/100
374/375            0s 95ms/step -
accuracy: 0.9617 - loss: 0.1071
Epoch 7: val_accuracy improved from 0.95605 to 0.96146, saving model to
best_model.keras
375/375            41s 109ms/step -
accuracy: 0.9617 - loss: 0.1071 - val_accuracy: 0.9615 - val_loss: 0.1087
Epoch 8/100
374/375            0s 95ms/step -
accuracy: 0.9629 - loss: 0.0977
Epoch 8: val_accuracy did not improve from 0.96146
375/375            40s 106ms/step -
accuracy: 0.9629 - loss: 0.0977 - val_accuracy: 0.9601 - val_loss: 0.1207
Epoch 9/100
374/375            0s 95ms/step -
accuracy: 0.9687 - loss: 0.0886
Epoch 9: val_accuracy did not improve from 0.96146
375/375            40s 106ms/step -
accuracy: 0.9687 - loss: 0.0886 - val_accuracy: 0.9588 - val_loss: 0.1121
Epoch 10/100
```

```
374/375             0s 95ms/step -
accuracy: 0.9700 - loss: 0.0860
Epoch 10: val_accuracy improved from 0.96146 to 0.96214, saving model to
best_model.keras
375/375             41s 108ms/step -
accuracy: 0.9700 - loss: 0.0859 - val_accuracy: 0.9621 - val_loss: 0.1182
Epoch 11/100
374/375             0s 95ms/step -
accuracy: 0.9707 - loss: 0.0875
Epoch 11: val_accuracy improved from 0.96214 to 0.96552, saving model to
best_model.keras
375/375             41s 109ms/step -
accuracy: 0.9707 - loss: 0.0874 - val_accuracy: 0.9655 - val_loss: 0.1123
Epoch 12/100
374/375             0s 96ms/step -
accuracy: 0.9751 - loss: 0.0690
Epoch 12: val_accuracy did not improve from 0.96552
375/375             54s 108ms/step -
accuracy: 0.9750 - loss: 0.0690 - val_accuracy: 0.9540 - val_loss: 0.1355
```

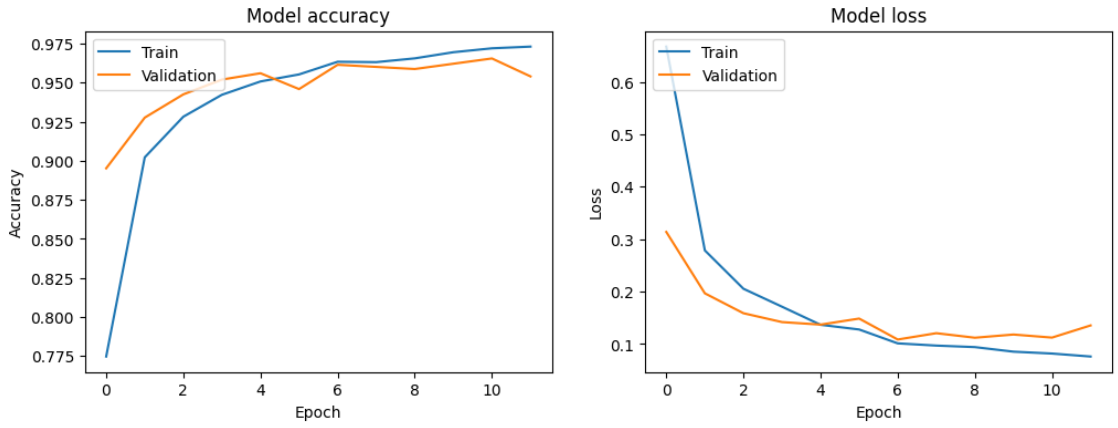
[31]:
```
evaluation = model.evaluate(val_data)

# Print the evaluation metrics
print("Validation Loss:", evaluation[0])
print("Validation Accuracy:", evaluation[1])
```

```
47/47             16s 162ms/step -
accuracy: 0.9618 - loss: 0.1327
Validation Loss: 0.1123444214463234
Validation Accuracy: 0.9655172228813171
```

1.7 Graphical Feature Representation

[32]:
```
# Plot training & validation accuracy values
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')

# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
```

```
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')

plt.show()
```



You can see above that we get 96.55% accuracy for our test dataset. This is considered to be a good accuracy

1.8 Results and Findings

[36]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns

# Manually define class names (update this with actual class names)
class_names = ['Black mold', 'Gray spot', 'Tomato__Bacterial_spot',␣
 ↪'Tomato__Early_blight', 'Tomato__Late_blight',
               'Tomato__Leaf_Mold', 'Tomato__Septoria_leaf_spot',␣
 ↪'Tomato__Spider_mites Two-spotted_spider_mite', 'Tomato__Target_Spot',␣
 ↪'Tomato__Tomato_Yellow_Leaf_Curl_Virus',
               'Tomato__healthy', 'powdery mildew',␣
 ↪'Tomato__Tomato_mosaic_virus']  # Modify based on your dataset

# Make predictions on the validation set
val_images = []
val_labels = []
val_predictions = []

for images, labels in val_data:
```

```
    preds = model.predict(images)
    val_images.extend(images)
    val_labels.extend(labels)
    val_predictions.extend(preds)

val_images = np.array(val_images)
val_labels = np.argmax(np.array(val_labels), axis=1)
val_predictions = np.argmax(np.array(val_predictions), axis=1)
```

```
1/1             0s 52ms/step
1/1             0s 53ms/step
1/1             0s 52ms/step
1/1             0s 54ms/step
1/1             0s 51ms/step
1/1             0s 54ms/step
1/1             0s 38ms/step
1/1             0s 37ms/step
1/1             0s 36ms/step
1/1             0s 51ms/step
1/1             0s 51ms/step
1/1             0s 37ms/step
1/1             0s 36ms/step
1/1             0s 51ms/step
1/1             0s 37ms/step
1/1             0s 36ms/step
1/1             0s 43ms/step
1/1             0s 54ms/step
1/1             0s 44ms/step
1/1             0s 38ms/step
1/1             0s 41ms/step
1/1             0s 52ms/step
1/1             0s 56ms/step
1/1             0s 38ms/step
1/1             0s 39ms/step
1/1             0s 53ms/step
1/1             0s 38ms/step
1/1             0s 52ms/step
1/1             0s 37ms/step
1/1             0s 36ms/step
1/1             0s 37ms/step
1/1             0s 37ms/step
1/1             0s 37ms/step
1/1             0s 53ms/step
1/1             0s 37ms/step
1/1             0s 38ms/step
1/1             0s 37ms/step
1/1             0s 53ms/step
```

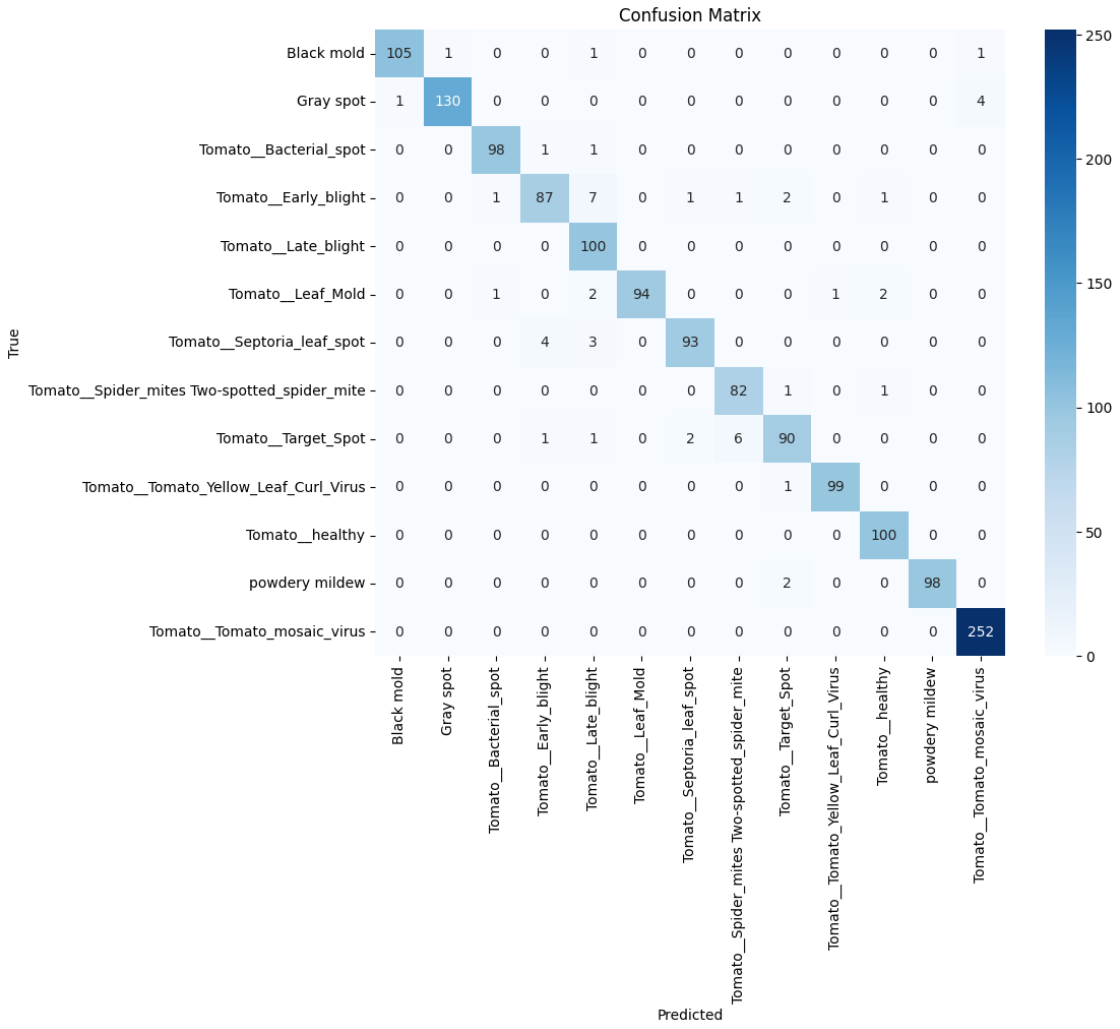
```
1/1             0s 49ms/step
1/1             0s 38ms/step
1/1             0s 36ms/step
1/1             0s 50ms/step
1/1             0s 36ms/step
1/1             0s 53ms/step
1/1             0s 36ms/step
1/1             0s 53ms/step
1/1             0s 29ms/step
```

[37]:
```
# Generate a classification report
report = classification_report(val_labels, val_predictions,␣
 ↪target_names=class_names)
print(report)
```

```
                                          precision    recall  f1-score
support

                                Black mold       0.99      0.97      0.98
108
                                 Gray spot       0.99      0.96      0.98
135
                    Tomato__Bacterial_spot       0.98      0.98      0.98
100
                      Tomato__Early_blight       0.94      0.87      0.90
100
                       Tomato__Late_blight       0.87      1.00      0.93
100
                         Tomato__Leaf_Mold       1.00      0.94      0.97
100
                Tomato__Septoria_leaf_spot       0.97      0.93      0.95
100
Tomato__Spider_mites Two-spotted_spider_mite       0.92      0.98      0.95
84
                       Tomato__Target_Spot       0.94      0.90      0.92
100
      Tomato__Tomato_Yellow_Leaf_Curl_Virus       0.99      0.99      0.99
100
                          Tomato__healthy       0.96      1.00      0.98
100
                           powdery mildew       1.00      0.98      0.99
100
                Tomato__Tomato_mosaic_virus       0.98      1.00      0.99
252

                                 accuracy                           0.97
1479
                                macro avg       0.96      0.96      0.96
```

```
1479
                                weighted avg       0.97      0.97      0.97
1479
```

[38]:
```
conf_matrix = confusion_matrix(val_labels, val_predictions)
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',␣
 ↪xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```



2 END