```
import tensorflow as tf
```

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

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

```
from pathlib import Path
import numpy as np

data_dir = Path("/content/drive/MyDrive/tea sickness dataset")  # Convert string to Path

# Get class names (folder names inside dataset)
class_names = np.array(sorted([item.name for item in data_dir.iterdir() if item.is_dir()]))

print("Class Names:", class_names)
```

```
Class Names: ['Anthracnose' 'algal leaf' 'bird eye spot' 'brown blight' 'gray light'
 'healthy' 'red leaf spot' 'white spot']
```

```
import os
print(os.getcwd())  # This shows the current working directory
```

```
/content
```

```
target_dir = "/content/drive/MyDrive/tea sickness dataset"
```

```
import os
print(os.listdir("/content/drive/My Drive/tea sickness dataset"))
```

```
['red leaf spot', 'Anthracnose', 'healthy', 'bird eye spot', 'brown blight', 'algal leaf', 'gray light', 'white spot']
```

```
# View an image
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import random
import os

def view_random_image(target_dir, target_class):
    # Setup target directory
    target_folder = target_dir + "/" + target_class

    # Get a random image
    random_image = random.sample(os.listdir(target_folder), 1)

    # Read in the image and plot it using matplotlib
    img = mpimg.imread(target_folder + "/" + random_image[0])
    plt.imshow(img)
    plt.title(target_class)
    plt.axis("off")

    print(f"Image shape: {img.shape}")  # Show the shape of the image
    return img

# View a random image from the training dataset
img = view_random_image(target_dir="/content/drive/MyDrive/tea sickness dataset", target_class="healthy")
```

```
Image shape: (1024, 768, 3)
```

```
# View the img (actually just a big array/tensor)
img
```

```
array([[[168, 208, 244],
        [166, 206, 242],
        [163, 205, 243],
        ...,
        [158, 206, 244],
        [160, 205, 244],
        [163, 208, 247]],

       [[165, 205, 241],
        [163, 206, 241],
        [164, 206, 244],
        ...,
        [160, 208, 246],
        [161, 206, 245],
        [162, 207, 246]],

       [[166, 208, 246],
        [165, 207, 245],
        [163, 205, 243],
        ...,
        [164, 209, 248],
        [164, 208, 247],
        [163, 207, 246]],

       ...,

       [[ 99, 136, 165],
        [ 98, 135, 164],
        [ 99, 134, 164],
        ...,
        [125, 164, 195],
        [125, 164, 193],
        [126, 165, 194]],

       [[ 99, 136, 165],
        [ 98, 135, 164],
        [ 97, 134, 163],
        ...,
        [125, 166, 194],
        [124, 165, 193],
        [125, 166, 194]],

       [[ 94, 131, 160],
        [ 94, 131, 160],
        [ 95, 132, 161],
        ...,
        [125, 166, 194],
        [124, 165, 193],
        [123, 164, 192]]], dtype=uint8)
```

```
import os
import matplotlib.image as mpimg
from collections import Counter
```

```
def get_unique_image_sizes(root_directory):
    size_counter = Counter()

    # Loop through each category folder
    for category in os.listdir(root_directory):
        category_path = os.path.join(root_directory, category)

        if os.path.isdir(category_path):  # Ensure it's a folder
            for filename in os.listdir(category_path):
                file_path = os.path.join(category_path, filename)

                try:
                    img = mpimg.imread(file_path)  # Read the image
                    size_counter[img.shape] += 1  # Count unique image sizes
                except Exception as e:
                    print(f"Error reading {file_path}: {e}")

    return size_counter

# Set the root dataset directory
root_dataset_path = "/content/drive/My Drive/tea sickness dataset"

# Get unique image sizes
image_size_counts = get_unique_image_sizes(root_dataset_path)

# Print distinct image sizes and their counts
print("Distinct Image Sizes and Their Counts:")
for size, count in image_size_counts.items():
    print(f"Size {size}: {count} images")
```

```
Distinct Image Sizes and Their Counts:
Size (1024, 768, 3): 552 images
Size (768, 1024, 3): 1 images
Size (4160, 3120, 3): 175 images
Size (3120, 4160, 3): 125 images
Size (1024, 1024, 3): 32 images
```

```
!ls "/content/drive/My Drive/tea sickness dataset/healthy/"
```

```
UNADJUSTEDNONRAW_thumb_206.jpg  UNADJUSTEDNONRAW_thumb_21f.jpg  UNADJUSTEDNONRAW_thumb_238.jpg
UNADJUSTEDNONRAW_thumb_207.jpg  UNADJUSTEDNONRAW_thumb_220.jpg  UNADJUSTEDNONRAW_thumb_239.jpg
UNADJUSTEDNONRAW_thumb_208.jpg  UNADJUSTEDNONRAW_thumb_221.jpg  UNADJUSTEDNONRAW_thumb_23a.jpg
UNADJUSTEDNONRAW_thumb_209.jpg  UNADJUSTEDNONRAW_thumb_222.jpg  UNADJUSTEDNONRAW_thumb_23b.jpg
UNADJUSTEDNONRAW_thumb_20a.jpg  UNADJUSTEDNONRAW_thumb_223.jpg  UNADJUSTEDNONRAW_thumb_23c.jpg
UNADJUSTEDNONRAW_thumb_20b.jpg  UNADJUSTEDNONRAW_thumb_224.jpg  UNADJUSTEDNONRAW_thumb_23d.jpg
UNADJUSTEDNONRAW_thumb_20c.jpg  UNADJUSTEDNONRAW_thumb_225.jpg  UNADJUSTEDNONRAW_thumb_23e.jpg
UNADJUSTEDNONRAW_thumb_20d.jpg  UNADJUSTEDNONRAW_thumb_226.jpg  UNADJUSTEDNONRAW_thumb_23f.jpg
UNADJUSTEDNONRAW_thumb_20e.jpg  UNADJUSTEDNONRAW_thumb_227.jpg  UNADJUSTEDNONRAW_thumb_240.jpg
UNADJUSTEDNONRAW_thumb_20f.jpg  UNADJUSTEDNONRAW_thumb_228.jpg  UNADJUSTEDNONRAW_thumb_241.jpg
UNADJUSTEDNONRAW_thumb_210.jpg  UNADJUSTEDNONRAW_thumb_229.jpg  UNADJUSTEDNONRAW_thumb_242.jpg
UNADJUSTEDNONRAW_thumb_211.jpg  UNADJUSTEDNONRAW_thumb_22a.jpg  UNADJUSTEDNONRAW_thumb_243.jpg
UNADJUSTEDNONRAW_thumb_212.jpg  UNADJUSTEDNONRAW_thumb_22b.jpg  UNADJUSTEDNONRAW_thumb_244.jpg
UNADJUSTEDNONRAW_thumb_213.jpg  UNADJUSTEDNONRAW_thumb_22c.jpg  UNADJUSTEDNONRAW_thumb_245.jpg
UNADJUSTEDNONRAW_thumb_214.jpg  UNADJUSTEDNONRAW_thumb_22d.jpg  UNADJUSTEDNONRAW_thumb_246.jpg
UNADJUSTEDNONRAW_thumb_215.jpg  UNADJUSTEDNONRAW_thumb_22e.jpg  UNADJUSTEDNONRAW_thumb_247.jpg
UNADJUSTEDNONRAW_thumb_216.jpg  UNADJUSTEDNONRAW_thumb_22f.jpg  UNADJUSTEDNONRAW_thumb_248.jpg
UNADJUSTEDNONRAW_thumb_217.jpg  UNADJUSTEDNONRAW_thumb_230.jpg  UNADJUSTEDNONRAW_thumb_249.jpg
UNADJUSTEDNONRAW_thumb_218.jpg  UNADJUSTEDNONRAW_thumb_231.jpg  UNADJUSTEDNONRAW_thumb_24a.jpg
UNADJUSTEDNONRAW_thumb_219.jpg  UNADJUSTEDNONRAW_thumb_232.jpg  UNADJUSTEDNONRAW_thumb_24b.jpg
UNADJUSTEDNONRAW_thumb_21a.jpg  UNADJUSTEDNONRAW_thumb_233.jpg  UNADJUSTEDNONRAW_thumb_24c.jpg
UNADJUSTEDNONRAW_thumb_21b.jpg  UNADJUSTEDNONRAW_thumb_234.jpg  UNADJUSTEDNONRAW_thumb_24d.jpg
UNADJUSTEDNONRAW_thumb_21c.jpg  UNADJUSTEDNONRAW_thumb_235.jpg  UNADJUSTEDNONRAW_thumb_24e.jpg
UNADJUSTEDNONRAW_thumb_21d.jpg  UNADJUSTEDNONRAW_thumb_236.jpg  UNADJUSTEDNONRAW_thumb_24f.jpg
UNADJUSTEDNONRAW_thumb_21e.jpg  UNADJUSTEDNONRAW_thumb_237.jpg
```

```
import os
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from tensorflow import keras
from tensorflow.keras import layers, models, regularizers

# Define dataset path
dataset_path = "/content/drive/MyDrive/tea sickness dataset"

# Get all image paths and labels
```

```
image_paths = []
labels = []
class_names = sorted(os.listdir(dataset_path))

for class_idx, class_name in enumerate(class_names):
    class_dir = os.path.join(dataset_path, class_name)
    for img_name in os.listdir(class_dir):
        image_paths.append(os.path.join(class_dir, img_name))
        labels.append(class_idx)

# Convert to NumPy arrays
image_paths = np.array(image_paths)
labels = np.array(labels)

# Stratified split (80% train, 20% validation)
train_paths, val_paths, train_labels, val_labels = train_test_split(
    image_paths, labels, test_size=0.2, stratify=labels, random_state=123
)


import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
# Data Augmentation
data_augmentation = keras.Sequential([
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1),
    layers.RandomContrast(0.1)
])

# Function to load and preprocess images
def load_image(image_path, label, augment=False):
    image = tf.io.read_file(image_path)
    image = tf.image.decode_jpeg(image, channels=3)
    image = tf.image.resize(image, (224, 224))  # Resize to match model input size
    image = image / 255.0  # Normalize pixel values
    if augment:
        image = data_augmentation(image)
    return image, label

# Create TensorFlow datasets
train_ds = tf.data.Dataset.from_tensor_slices((train_paths, train_labels))
train_ds = train_ds.map(lambda x, y: load_image(x, y, augment=True)).batch(32).shuffle(1000).prefetch(tf.data.AUTOTUNE)

val_ds = tf.data.Dataset.from_tensor_slices((val_paths, val_labels))
val_ds = val_ds.map(load_image).batch(32).prefetch(tf.data.AUTOTUNE)


#somewhat good model
from tensorflow import keras
from tensorflow.keras import layers, models
from tensorflow.keras import regularizers

# Define CNN Model with increased Regularization & Dropout
model = models.Sequential([
    layers.Input(shape=(224, 224, 3)),
    layers.Conv2D(32, (3,3), activation='relu', padding='same', kernel_regularizer=regularizers.l2(0.002)),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(64, (3,3), activation='relu', padding='same', kernel_regularizer=regularizers.l2(0.002)),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(128, (3,3), activation='relu', padding='same', kernel_regularizer=regularizers.l2(0.002)),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(256, (3,3), activation='relu', padding='same', kernel_regularizer=regularizers.l2(0.002)),
    layers.MaxPooling2D(2,2),

    layers.Flatten(),
    layers.Dropout(0.3),  # New Dropout Layer to reduce overfitting
    layers.Dense(256, activation='relu', kernel_regularizer=regularizers.l2(0.005)),
    layers.Dropout(0.5),  # Increased Dropout Rate
    layers.Dense(len(class_names), activation='softmax')  # Output layer
])

# Compile Model
model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.001),  # Adjusted Learning Rate
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

# Model Summary
model.summary()
```

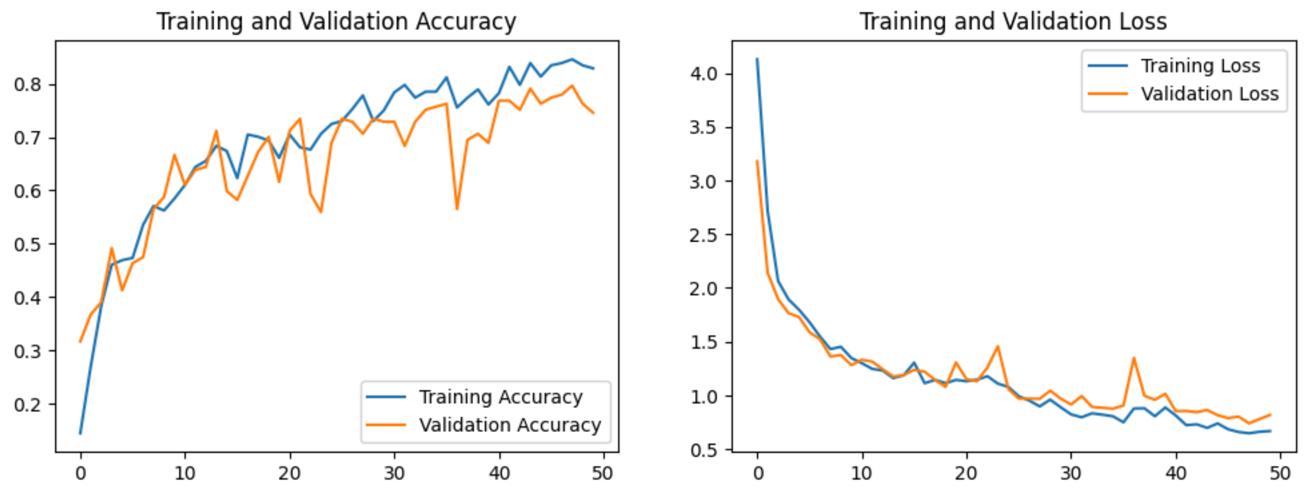
```
Model: "sequential_1"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃         Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ conv2d (Conv2D)                      │ (None, 224, 224, 32)        │             896 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d (MaxPooling2D)         │ (None, 112, 112, 32)        │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_1 (Conv2D)                    │ (None, 112, 112, 64)        │          18,496 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_1 (MaxPooling2D)       │ (None, 56, 56, 64)          │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_2 (Conv2D)                    │ (None, 56, 56, 128)         │          73,856 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_2 (MaxPooling2D)       │ (None, 28, 28, 128)         │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_3 (Conv2D)                    │ (None, 28, 28, 256)         │         295,168 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_3 (MaxPooling2D)       │ (None, 14, 14, 256)         │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ flatten (Flatten)                    │ (None, 50176)               │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dropout (Dropout)                    │ (None, 50176)               │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense (Dense)                        │ (None, 256)                 │      12,845,312 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dropout_1 (Dropout)                  │ (None, 256)                 │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_1 (Dense)                      │ (None, 8)                   │           2,056 │
└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘
 Total params: 13,235,784 (50.49 MB)
 Trainable params: 13,235,784 (50.49 MB)
```

```
# Callbacks for Early Stopping and Learning Rate Adjustment
callbacks = [
    keras.callbacks.EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True),
    keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, min_lr=1e-6)
]

# Train Model
EPOCHS = 50
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=EPOCHS,
    callbacks=callbacks
)
```

```
Epoch 1/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 639s 8s/step - accuracy: 0.1475 - loss: 5.0097 - val_accuracy: 0.3164 - val_loss: 3.1782 - learning_rate:
Epoch 2/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 172s 6s/step - accuracy: 0.2390 - loss: 2.9414 - val_accuracy: 0.3672 - val_loss: 2.1404 - learning_rate:
Epoch 3/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 173s 6s/step - accuracy: 0.3427 - loss: 2.1684 - val_accuracy: 0.3898 - val_loss: 1.8957 - learning_rate:
Epoch 4/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 166s 5s/step - accuracy: 0.4635 - loss: 1.9086 - val_accuracy: 0.4915 - val_loss: 1.7655 - learning_rate:
Epoch 5/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 208s 6s/step - accuracy: 0.4574 - loss: 1.7950 - val_accuracy: 0.4124 - val_loss: 1.7294 - learning_rate:
Epoch 6/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 210s 6s/step - accuracy: 0.4441 - loss: 1.7468 - val_accuracy: 0.4633 - val_loss: 1.5885 - learning_rate:
Epoch 7/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 185s 5s/step - accuracy: 0.5257 - loss: 1.5483 - val_accuracy: 0.4746 - val_loss: 1.5231 - learning_rate:
Epoch 8/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 171s 6s/step - accuracy: 0.5456 - loss: 1.4641 - val_accuracy: 0.5650 - val_loss: 1.3627 - learning_rate:
Epoch 9/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 201s 6s/step - accuracy: 0.5452 - loss: 1.4340 - val_accuracy: 0.5876 - val_loss: 1.3757 - learning_rate:
Epoch 10/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 179s 6s/step - accuracy: 0.5904 - loss: 1.3713 - val_accuracy: 0.6667 - val_loss: 1.2818 - learning_rate:
Epoch 11/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 194s 6s/step - accuracy: 0.5833 - loss: 1.2949 - val_accuracy: 0.6102 - val_loss: 1.3319 - learning_rate:
Epoch 12/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 194s 5s/step - accuracy: 0.6143 - loss: 1.2694 - val_accuracy: 0.6384 - val_loss: 1.3149 - learning_rate:
Epoch 13/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 218s 6s/step - accuracy: 0.6248 - loss: 1.2630 - val_accuracy: 0.6441 - val_loss: 1.2431 - learning_rate:
Epoch 14/50
```

```
23/23 ━━━━━━━━━━━━━━━━━━━━ 194s 6s/step - accuracy: 0.6847 - loss: 1.1672 - val_accuracy: 0.7119 - val_loss: 1.1762 - learning_rate:
Epoch 15/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 162s 5s/step - accuracy: 0.6938 - loss: 1.1265 - val_accuracy: 0.5989 - val_loss: 1.1891 - learning_rate:
Epoch 16/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 218s 6s/step - accuracy: 0.5976 - loss: 1.3349 - val_accuracy: 0.5819 - val_loss: 1.2385 - learning_rate:
Epoch 17/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 171s 6s/step - accuracy: 0.7115 - loss: 1.0945 - val_accuracy: 0.6271 - val_loss: 1.2207 - learning_rate:
Epoch 18/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 172s 6s/step - accuracy: 0.7061 - loss: 1.1374 - val_accuracy: 0.6723 - val_loss: 1.1447 - learning_rate:
Epoch 19/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 208s 6s/step - accuracy: 0.6514 - loss: 1.1759 - val_accuracy: 0.7006 - val_loss: 1.0795 - learning_rate:
Epoch 20/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 188s 5s/step - accuracy: 0.6464 - loss: 1.1436 - val_accuracy: 0.6158 - val_loss: 1.3072 - learning_rate:
Epoch 21/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 170s 6s/step - accuracy: 0.7075 - loss: 1.1172 - val_accuracy: 0.7119 - val_loss: 1.1511 - learning_rate:
Epoch 22/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 203s 6s/step - accuracy: 0.6847 - loss: 1.1315 - val_accuracy: 0.7345 - val_loss: 1.1323 - learning_rate:
Epoch 23/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 179s 6s/step - accuracy: 0.6536 - loss: 1.1674 - val_accuracy: 0.5932 - val_loss: 1.2600 - learning_rate:
Epoch 24/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 188s 5s/step - accuracy: 0.7147 - loss: 1.0802 - val_accuracy: 0.5593 - val_loss: 1.4579 - learning_rate:
Epoch 25/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 206s 6s/step - accuracy: 0.6943 - loss: 1.1453 - val_accuracy: 0.6893 - val_loss: 1.0604 - learning_rate:
Epoch 26/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 180s 6s/step - accuracy: 0.7405 - loss: 0.9804 - val_accuracy: 0.7345 - val_loss: 0.9703 - learning_rate:
Epoch 27/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 190s 6s/step - accuracy: 0.7662 - loss: 0.9366 - val_accuracy: 0.7288 - val_loss: 0.9709 - learning_rate:
Epoch 28/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 204s 6s/step - accuracy: 0.7859 - loss: 0.9099 - val_accuracy: 0.7062 - val_loss: 0.9687 - learning_rate:
Epoch 29/50
```

```python
import matplotlib.pyplot as plt

# Plot accuracy and loss
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(EPOCHS)

plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')

plt.show()
```



Start coding or generate with AI.