```
# entrenar_modelo.py

# Importación de librerías necesarias para el desarrollo del modelo
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam
import os

# ==============================
# Definición de rutas del dataset
# ==============================

# Carpeta principal donde se encuentra el dataset estructurado
base_dir = "dataset"

# Rutas específicas para entrenamiento, validación y prueba
train_dir = os.path.join(base_dir, "train")
val_dir = os.path.join(base_dir, "val")
test_dir = os.path.join(base_dir, "test")

# ==============================
# Parámetros del modelo
# ==============================

# Dimensiones a las que serán redimensionadas las imágenes
img_height, img_width = 224, 224
```

```
# Número de imágenes procesadas por iteración (batch)
batch_size = 32

# Número de épocas de entrenamiento (iteraciones completas sobre el dataset)
epochs = 10

# Tasa de aprendizaje del optimizador
learning_rate = 1e-4

# Número de clases del problema (botón, corte, abierta)
num_classes = 3

# ==============================
# Generadores de datos
# ==============================

# Generador para el conjunto de entrenamiento con aumento de datos (data augmentation)
# Esto permite mejorar la generalización del modelo creando variaciones artificiales de las imágenes
train_datagen = ImageDataGenerator(
    rescale=1./255,           # Normalización de los píxeles (0-255 → 0-1)
    rotation_range=20,        # Rotación aleatoria de imágenes
    width_shift_range=0.2,    # Desplazamiento horizontal
    height_shift_range=0.2,   # Desplazamiento vertical
    shear_range=0.2,          # Transformación de corte
    zoom_range=0.2,           # Zoom aleatorio
    horizontal_flip=True,     # Volteo horizontal
    fill_mode='nearest'       # Método de relleno de píxeles
```

```
)

# Generador para validación (solo normalización, sin aumento)
val_datagen = ImageDataGenerator(rescale=1./255)

# Carga de imágenes desde las carpetas para entrenamiento
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(img_height, img_width),  # Redimensionamiento automático
    batch_size=batch_size,
    class_mode='categorical'            # Clasificación multiclase
)

# Carga de imágenes para validación
val_generator = val_datagen.flow_from_directory(
    val_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical'
)

# ==============================
# Definición del modelo
# ==============================

# Se utiliza MobileNetV2 preentrenado en ImageNet como extractor de características
# include_top=False elimina la capa final de clasificación original
base_model = MobileNetV2(
    weights='imagenet',
```

```
    include_top=False,
    input_shape=(img_height, img_width, 3)
)

# Se congelan los pesos del modelo base para evitar su entrenamiento
# Esto permite usarlo como extractor de características
base_model.trainable = False

# ==============================
# Construcción de capas personalizadas
# ==============================

# Salida del modelo base
x = base_model.output

# Reducción de dimensiones mediante pooling global
x = GlobalAveragePooling2D()(x)

# Capa densa intermedia para aprendizaje de patrones
x = Dense(128, activation='relu')(x)

# Capa de salida con activación softmax para clasificación multiclase
predictions = Dense(num_classes, activation='softmax')(x)

# Definición final del modelo completo
model = Model(inputs=base_model.input, outputs=predictions)

# ==============================
# Compilación del modelo
```

```
# ==============================

# Se define el optimizador, función de pérdida y métricas de evaluación
model.compile(
    optimizer=Adam(learning_rate=learning_rate),
    loss='categorical_crossentropy',  # Adecuada para clasificación multiclase
    metrics=['accuracy']           # Métrica de evaluación
)

# ==============================
# Entrenamiento del modelo
# ==============================

# Se entrena el modelo usando los datos de entrenamiento y validación
history = model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=epochs
)

# ==============================
# Guardado del modelo
# ==============================

# Se guarda el modelo entrenado en formato HDF5
model.save("modelo_rosas.h5")

# Confirmación en consola
print("✅ Modelo entrenado y guardado como modelo_rosas.h5")
```

```
# ==============================
# Evaluación del modelo
# ==============================

# Generador para el conjunto de prueba (solo normalización)
test_datagen = ImageDataGenerator(rescale=1./255)

# Carga del conjunto de prueba
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False  # Importante para evaluación consistente
)

# Evaluación del modelo en el conjunto de prueba
loss, acc = model.evaluate(test_generator)

# Impresión de la precisión final
print(f"📊 Precisión en test: {acc*100:.2f}%")
```