```
import tensorflow as tf
import kagglehub
import shutil
import os
from google.colab import drive
from tensorflow.keras.models import Sequential, load_model

#get latest version of dataset
path = kagglehub.dataset_download("seanscully29/flowers-classification")
print("Path to dataset files:",path)
```

```
Path to dataset files: /kaggle/input/flowers-classification
```

```
local_path="./flower_species_dataset"
os.makedirs(local_path, exist_ok=True)

try:
  for item in os.listdir(path):
    s = os.path.join(path,item)
    d = os.path.join(local_path,item)
    if os.path.isdir(s):
      shutil.copytree(s,d)
    else:
      shutil.copy2(s,d)
except FileExistsError: #If code accidentally runs again, print text signaling that the path already exists instead of throwing an error
  print(f"{local_path} already exists.")

print(f"Dataset saved at: {local_path}")
```

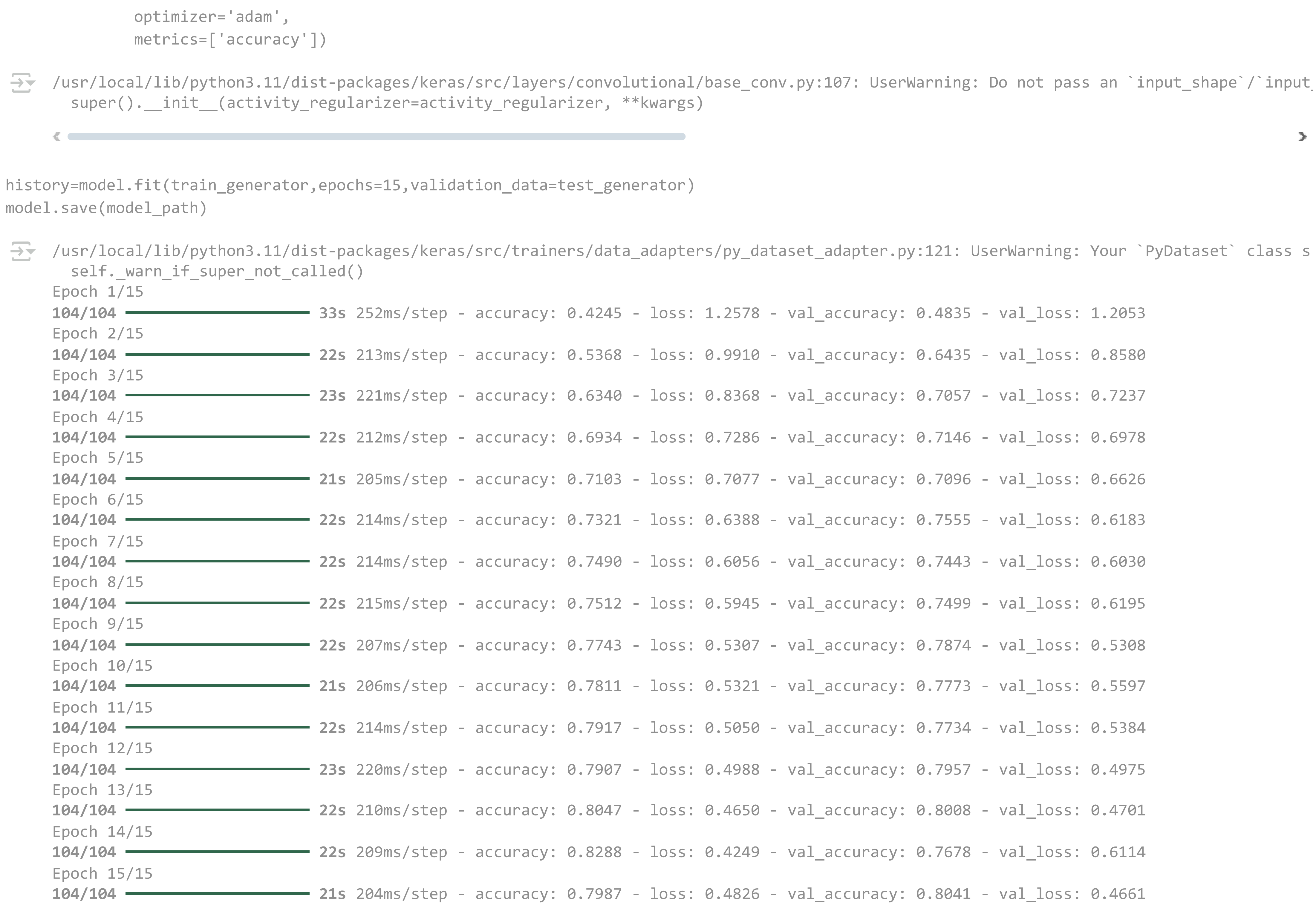
```
Dataset saved at: ./flower_species_dataset
```

```
source_path="/content/flower_species_dataset/flowerdataset"
training_path='./flower_species_dataset/training'
testing_path='./flower_species_dataset/testing'
classes=["black_eyed_susan","calendula","california_poppy","coreopsis","iris"]

model_path="flower_dataset_model.h5"
```

```
for cls in classes: #creates subdirectories for each flower class
  os.makedirs(os.path.join(training_path,cls),exist_ok=True)
  os.makedirs(os.path.join(testing_path,cls),exist_ok=True)

for cls in classes:
  files=os.listdir(os.path.join(source_path,cls)) #Lists image files in source directory

  #split 65% of the images to the training set, and 35% to the testing set
  split=int(0.65 * len(files))
  training_files=files[:split]
  testing_files=files[split:]

  for t in training_files: #Adds image files to recently created subdirectories on a 65/35 split
    shutil.move(os.path.join(source_path, cls, t),os.path.join(training_path,cls,t))
  for t in testing_files:
    shutil.move(os.path.join(source_path, cls, t),os.path.join(testing_path,cls,t))


# Define ImageDataGenerator for augmentation and rescaling
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_directory='/content/flower_species_dataset/training'
test_directory='/content/flower_species_dataset/testing'
train_datagen = ImageDataGenerator(
    rescale=1./255, #Normalizes pixel values
    rotation_range=20,  #Randomly rotates image by up to +- 20 degrees
    width_shift_range=0.2,
    height_shift_range=0.2, #Shifts height/width by up to 20%
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

test_datagen = ImageDataGenerator(rescale=1./255)

# Load training dataset
train_generator = train_datagen.flow_from_directory(
    train_directory,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical'
)

# Load testing dataset
test_generator = test_datagen.flow_from_directory(
    test_directory,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical'
)
```

```
Found 3314 images belonging to 5 classes.
Found 1787 images belonging to 5 classes.
```

```
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(150, 150, 3)),
    MaxPooling2D(2,2),

    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D(2,2),

    Conv2D(128, (3,3), activation='relu'),
    MaxPooling2D(2,2),

     Conv2D(256, (3,3), activation='relu'),
     MaxPooling2D(2,2),

    Flatten(),
    Dense(512, activation='relu'),
    Dense(5, activation='sigmoid')  # Categorical with 5 classes
])

# Compile the model
model.compile(loss='categorical_crossentropy',
```



```
          optimizer='adam',
          metrics=['accuracy'])
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

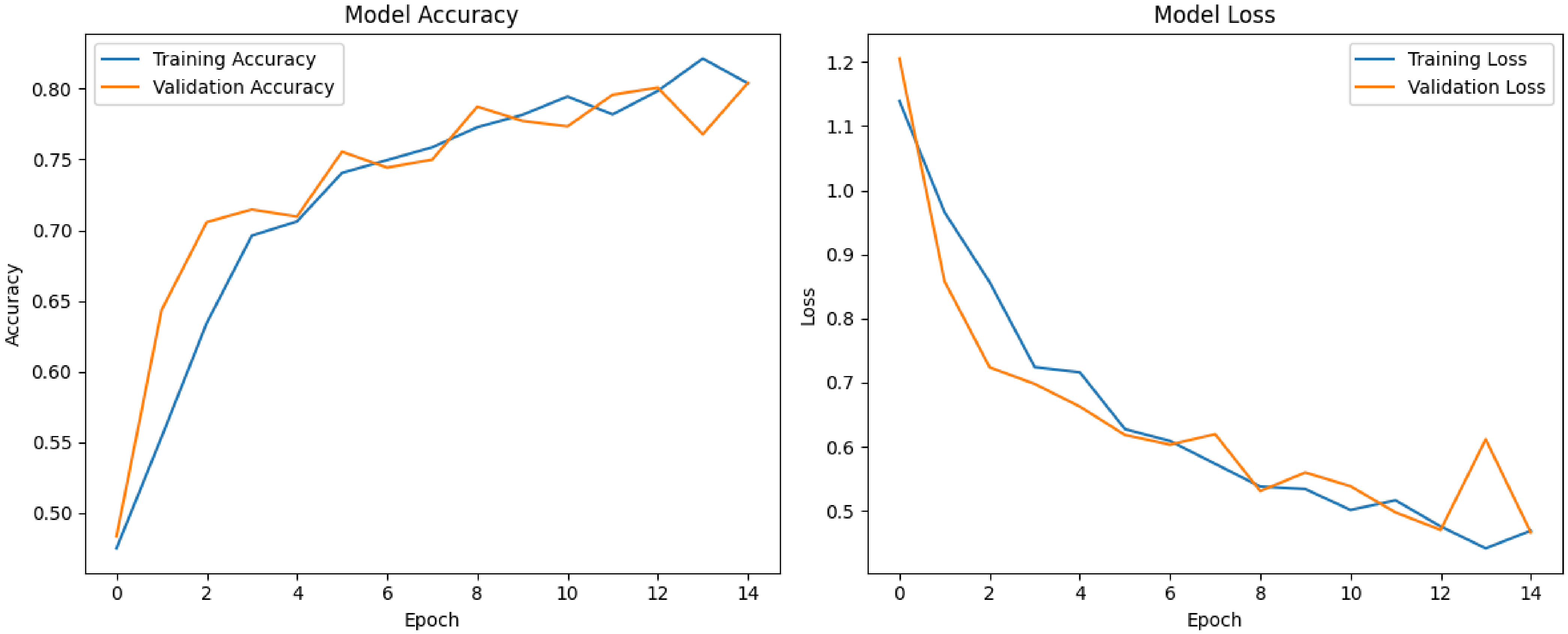
```
history=model.fit(train_generator,epochs=15,validation_data=test_generator)
model.save(model_path)
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class s
  self._warn_if_super_not_called()
Epoch 1/15
104/104 ━━━━━━━━━━━━━━━━━━━━ 33s 252ms/step - accuracy: 0.4245 - loss: 1.2578 - val_accuracy: 0.4835 - val_loss: 1.2053
Epoch 2/15
104/104 ━━━━━━━━━━━━━━━━━━━━ 22s 213ms/step - accuracy: 0.5368 - loss: 0.9910 - val_accuracy: 0.6435 - val_loss: 0.8580
Epoch 3/15
104/104 ━━━━━━━━━━━━━━━━━━━━ 23s 221ms/step - accuracy: 0.6340 - loss: 0.8368 - val_accuracy: 0.7057 - val_loss: 0.7237
Epoch 4/15
104/104 ━━━━━━━━━━━━━━━━━━━━ 22s 212ms/step - accuracy: 0.6934 - loss: 0.7286 - val_accuracy: 0.7146 - val_loss: 0.6978
Epoch 5/15
104/104 ━━━━━━━━━━━━━━━━━━━━ 21s 205ms/step - accuracy: 0.7103 - loss: 0.7077 - val_accuracy: 0.7096 - val_loss: 0.6626
Epoch 6/15
104/104 ━━━━━━━━━━━━━━━━━━━━ 22s 214ms/step - accuracy: 0.7321 - loss: 0.6388 - val_accuracy: 0.7555 - val_loss: 0.6183
Epoch 7/15
104/104 ━━━━━━━━━━━━━━━━━━━━ 22s 214ms/step - accuracy: 0.7490 - loss: 0.6056 - val_accuracy: 0.7443 - val_loss: 0.6030
Epoch 8/15
104/104 ━━━━━━━━━━━━━━━━━━━━ 22s 215ms/step - accuracy: 0.7512 - loss: 0.5945 - val_accuracy: 0.7499 - val_loss: 0.6195
Epoch 9/15
104/104 ━━━━━━━━━━━━━━━━━━━━ 22s 207ms/step - accuracy: 0.7743 - loss: 0.5307 - val_accuracy: 0.7874 - val_loss: 0.5308
Epoch 10/15
104/104 ━━━━━━━━━━━━━━━━━━━━ 21s 206ms/step - accuracy: 0.7811 - loss: 0.5321 - val_accuracy: 0.7773 - val_loss: 0.5597
Epoch 11/15
104/104 ━━━━━━━━━━━━━━━━━━━━ 22s 214ms/step - accuracy: 0.7917 - loss: 0.5050 - val_accuracy: 0.7734 - val_loss: 0.5384
Epoch 12/15
104/104 ━━━━━━━━━━━━━━━━━━━━ 23s 220ms/step - accuracy: 0.7907 - loss: 0.4988 - val_accuracy: 0.7957 - val_loss: 0.4975
Epoch 13/15
104/104 ━━━━━━━━━━━━━━━━━━━━ 22s 210ms/step - accuracy: 0.8047 - loss: 0.4650 - val_accuracy: 0.8008 - val_loss: 0.4701
Epoch 14/15
104/104 ━━━━━━━━━━━━━━━━━━━━ 22s 209ms/step - accuracy: 0.8288 - loss: 0.4249 - val_accuracy: 0.7678 - val_loss: 0.6114
Epoch 15/15
104/104 ━━━━━━━━━━━━━━━━━━━━ 21s 204ms/step - accuracy: 0.7987 - loss: 0.4826 - val_accuracy: 0.8041 - val_loss: 0.4661
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is consid
```

```
import matplotlib.pyplot as plt

# === Visualization of training ===
print("Visualizing training results...")

# Plot accuracy
plt.figure(figsize=(12,5))

plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend() #Adds necessary labeling to plot

# Plot loss
plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

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

```
Visualizing training results...
```



```
from tensorflow.keras.models import load_model
from PIL import Image, ImageOps
import numpy as np
import matplotlib.pyplot as plt

def predict_new_image(img_path):
    model = load_model(model_path)

    # Load image in color (RGB)
    img = Image.open(img_path).convert("RGB")
    img = img.resize((150, 150))
    img_array = np.array(img).astype("float32") / 255.0
    img_array = img_array.reshape(1, 150, 150, 3)

    prediction = model.predict(img_array)
    predicted_index = np.argmax(prediction)
    confidence = prediction[0][predicted_index]

    # Display
    plt.imshow(img)
    plt.axis("off")
    plt.title(f"Prediction: {classes[predicted_index]}, Confidence: ({confidence:.2%})")#Prints name of classes most likely to be the flower
    plt.show()


predict_new_image("/content/calendula.jpg")
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you tr
WARNING:tensorflow:6 out of the last 6 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed a
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 316ms/step
```

Prediction: calendula, Confidence: (94.02%)