```
import matplotlib.pyplot as plt
import numpy as np
import PIL
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
```

```
import pathlib

# dataset_url is now a local file path
dataset_path = "/content/flower_photos.tgz"

# Use tf.keras.utils.get_file with the 'file://' scheme
# or, simpler, tf.io.gfile.copy
# or simply use the dataset_path directly
data_dir = tf.keras.utils.get_file(
    'flower_photos.tgz',
    origin='file://' + dataset_path,  # Prepend 'file://' to the path
    extract=True,
    cache_dir="/content"
)
data_dir = pathlib.Path(data_dir).with_suffix('')
```

```
Downloading data from file:///content/flower_photos.tgz
228813984/228813984 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
```

```
image_count = len(list(data_dir.glob('*/*.jpg')))
print(image_count)
```

```
0
```

```
import pathlib
import os

dataset_url = "/content/flower_photos.tgz"
# Change fname to just the filename, not the full path
data_dir = tf.keras.utils.get_file('flower_photos.tgz', origin=dataset_url, extract=True, cache_dir='/content')
data_dir = pathlib.Path(data_dir).with_suffix('')

# Check if the 'roses' subdirectory exists
roses_dir = data_dir / 'roses'
if roses_dir.exists() and os.listdir(roses_dir): #check if it exists and is not empty
    roses = list(data_dir.glob('roses/*'))
    PIL.Image.open(str(roses[0]))
else:
    print("No rose images found in the directory.")
```

```
No rose images found in the directory.
```

```
import pathlib
import os

dataset_url = "/content/flower_photos.tgz"
# Assuming 'dataset_url' is a path to a local .tgz file
# You can use tf.keras.utils.get_file with 'file://' for local files
# or simply extract the file manually. Here's the manual extraction:
import tarfile

# Open the tar.gz file and extract it to the current directory
with tarfile.open(dataset_url, 'r:gz') as tar:
    tar.extractall('/content')  # Change to desired extraction path

# Now you can set data_dir to the extracted folder
data_dir = pathlib.Path('/content/flower_photos') # Assuming it extracts to 'flower_photos'

# Check if the 'roses' subdirectory exists and has at least 2 images
roses_dir = data_dir / 'roses'
if roses_dir.exists() and len(os.listdir(roses_dir)) >= 2: # check if it exists and has at least 2 images
    roses = list(data_dir.glob('roses/*'))
    PIL.Image.open(str(roses[0])) # Open the first rose image
    PIL.Image.open(str(roses[1])) # Open the second rose image (now safe)
else:
    print("Not enough rose images found in the directory to open the second image.")
```

```
import pathlib
import os
```

```
dataset_url = "/content/flower_photos.tgz"
# Correct usage of get_file: fname is just the filename, cache_dir specifies the directory
data_dir = tf.keras.utils.get_file('flower_photos.tgz', origin=dataset_url, extract=True, cache_dir='/content')
data_dir = pathlib.Path(data_dir).with_suffix('')

tulips = list(data_dir.glob('tulips/*'))

# Check if the tulips list is empty before trying to access elements
if tulips:
    PIL.Image.open(str(tulips[0]))
else:
    print("No tulip images found in the directory.")
```

```
No tulip images found in the directory.
```

```
import pathlib
import os

dataset_url = "/content/flower_photos.tgz"
# Correct usage: fname is just the filename, cache_dir specifies the directory
data_dir = tf.keras.utils.get_file('flower_photos.tgz', origin=dataset_url, extract=True, cache_dir='/content')
data_dir = pathlib.Path(data_dir).with_suffix('')

tulips = list(data_dir.glob('tulips/*'))

# Check if the tulips list has at least 2 elements before trying to access the second element
if len(tulips) >= 2:  # Check if it has at least 2 images
    PIL.Image.open(str(tulips[1])) # Open the second tulip image (now safe)
else:
    print("Not enough tulip images found in the directory to open the second image.")
```

```
Not enough tulip images found in the directory to open the second image.
```

```
batch_size = 32
img_height = 180
img_width = 180
```

```
train_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
Found 3670 files belonging to 1 classes.
Using 2936 files for training.
```

```
val_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

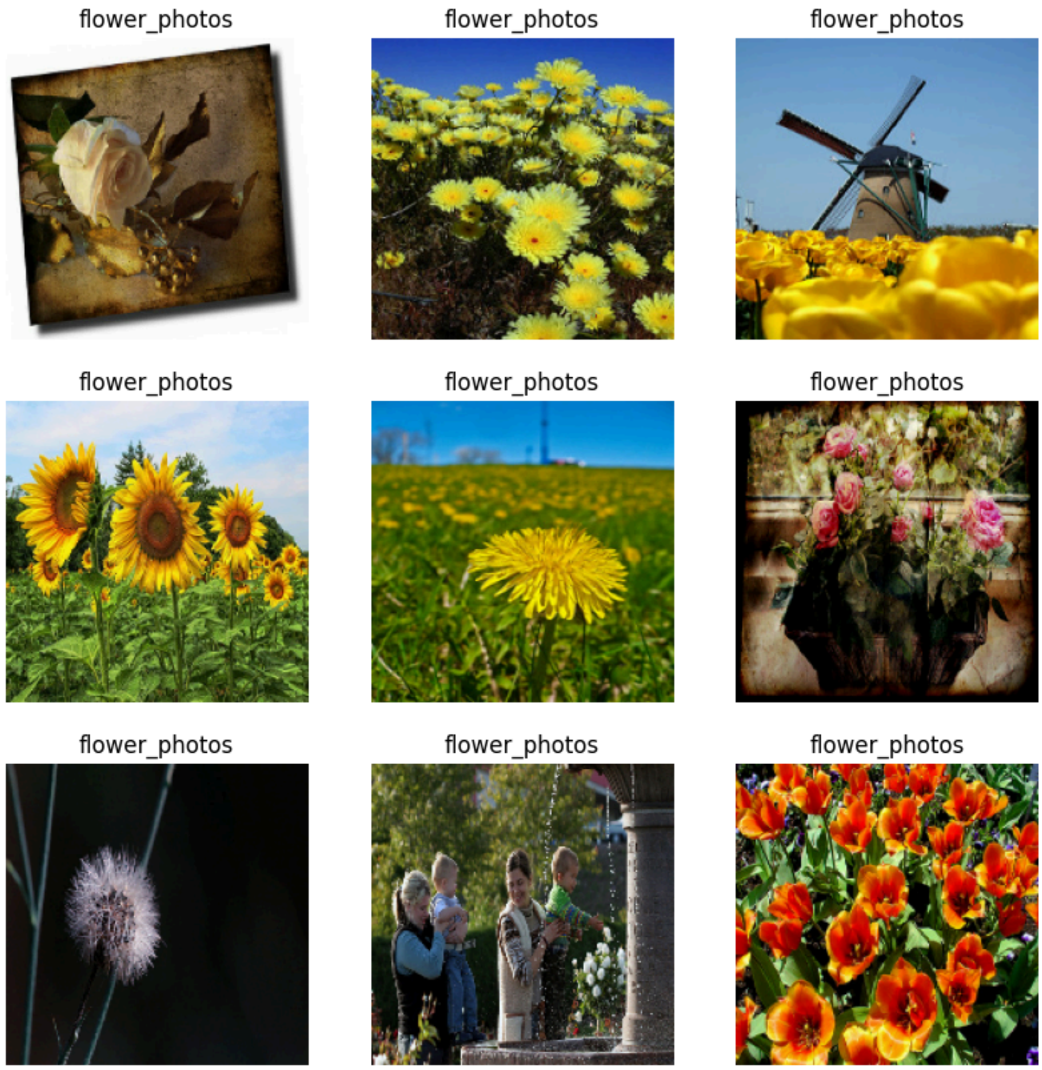
```
Found 3670 files belonging to 1 classes.
Using 734 files for validation.
```

```
class_names = train_ds.class_names
print(class_names)
```

```
['flower_photos']
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```




```
for image_batch, labels_batch in train_ds:
  print(image_batch.shape)
  print(labels_batch.shape)
  break
```

```
(32, 180, 180, 3)
(32,)
```

```
AUTOTUNE = tf.data.AUTOTUNE

train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

```
normalization_layer = layers.Rescaling(1./255)
```

```
normalization_layer = layers.Rescaling(1./255)
```

```
num_classes = len(class_names)

model = Sequential([
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/preprocessing/tf_data_layer.py:19: UserWarning: Do not pass an `input_shape
  super().__init__(**kwargs)
```

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
rescaling_2 (Rescaling)	(None, 180, 180, 3)	0
conv2d (Conv2D)	(None, 180, 180, 16)	448
max_pooling2d (MaxPooling2D)	(None, 90, 90, 16)	0
conv2d_1 (Conv2D)	(None, 90, 90, 32)	4,640
max_pooling2d_1 (MaxPooling2D)	(None, 45, 45, 32)	0
conv2d_2 (Conv2D)	(None, 45, 45, 64)	18,496
max_pooling2d_2 (MaxPooling2D)	(None, 22, 22, 64)	0
flatten (Flatten)	(None, 30976)	0
dense (Dense)	(None, 128)	3,965,056
dense_1 (Dense)	(None, 1)	129

Total params: 3,988,769 (15.22 MB)
Trainable params: 3,988,769 (15.22 MB)
Non-trainable params: 0 (0.00 B)

```
epochs=10
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

```
Epoch 1/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 100s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 2/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 89s 973ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 3/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 95s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 4/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 89s 967ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 5/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 91s 986ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 6/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 92s 997ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 7/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 141s 987ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 8/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 89s 969ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 9/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 94s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 10/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 94s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
```

```
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=(8, 8))
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()
```


```
data_augmentation = keras.Sequential(
  [
    layers.RandomFlip("horizontal",
                      input_shape=(img_height,
                                   img_width,
                                   3)),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1),
  ]
)


plt.figure(figsize=(10, 10))
for images, _ in train_ds.take(1):
  for i in range(9):
    augmented_images = data_augmentation(images)
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(augmented_images[0].numpy().astype("uint8"))
    plt.axis("off")
```

```
model = Sequential([
  data_augmentation,
  layers.Rescaling(1./255),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes, name="outputs")
])
```

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "sequential_2"
```

Layer (type)	Output Shape	Param #
sequential_1 (Sequential)	(None, 180, 180, 3)	0
rescaling_3 (Rescaling)	(None, 180, 180, 3)	0
conv2d_3 (Conv2D)	(None, 180, 180, 16)	448
max_pooling2d_3 (MaxPooling2D)	(None, 90, 90, 16)	0
conv2d_4 (Conv2D)	(None, 90, 90, 32)	4,640
max_pooling2d_4 (MaxPooling2D)	(None, 45, 45, 32)	0
conv2d_5 (Conv2D)	(None, 45, 45, 64)	18,496
max_pooling2d_5 (MaxPooling2D)	(None, 22, 22, 64)	0
dropout (Dropout)	(None, 22, 22, 64)	0
flatten_1 (Flatten)	(None, 30976)	0
dense_2 (Dense)	(None, 128)	3,965,056
outputs (Dense)	(None, 1)	129

```
Total params: 3,988,769 (15.22 MB)
Trainable params: 3,988,769 (15.22 MB)
Non-trainable params: 0 (0.00 B)
```

```
epochs = 15
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

```
Epoch 1/15
92/92 ━━━━━━━━━━━━━━━━━━━━ 124s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 2/15
92/92 ━━━━━━━━━━━━━━━━━━━━ 109s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 3/15
92/92 ━━━━━━━━━━━━━━━━━━━━ 110s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 4/15
92/92 ━━━━━━━━━━━━━━━━━━━━ 108s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 5/15
92/92 ━━━━━━━━━━━━━━━━━━━━ 107s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 6/15
92/92 ━━━━━━━━━━━━━━━━━━━━ 109s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 7/15
92/92 ━━━━━━━━━━━━━━━━━━━━ 107s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 8/15
92/92 ━━━━━━━━━━━━━━━━━━━━ 112s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 9/15
92/92 ━━━━━━━━━━━━━━━━━━━━ 139s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 10/15
92/92 ━━━━━━━━━━━━━━━━━━━━ 107s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 11/15
92/92 ━━━━━━━━━━━━━━━━━━━━ 141s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 12/15
92/92 ━━━━━━━━━━━━━━━━━━━━ 112s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 13/15
92/92 ━━━━━━━━━━━━━━━━━━━━ 109s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 14/15
92/92 ━━━━━━━━━━━━━━━━━━━━ 109s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 15/15
92/92 ━━━━━━━━━━━━━━━━━━━━ 109s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
```

```
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=(8, 8))
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()
```


```
import pathlib

sunflower_url = "/content/flower_photos.tgz"  # Path to the .tgz file
data_dir = tf.keras.utils.get_file(
    'flower_photos.tgz',  # Use the original filename for extraction
    origin='file://' + sunflower_url,
    extract=True,
    cache_dir="/content"
)
data_dir = pathlib.Path(data_dir).with_suffix('')

# List the files in the sunflowers directory to find the actual filename
sunflower_files = list(data_dir.glob('sunflowers/*.jpg'))

# Check if any sunflower images were found
if sunflower_files:
    # Use the first sunflower image found
    sunflower_path = sunflower_files[0]

    # Now you can load and predict:
    img = tf.keras.utils.load_img(
        sunflower_path, target_size=(img_height, img_width)
    )
    img_array = tf.keras.utils.img_to_array(img)
    img_array = tf.expand_dims(img_array, 0)  # Create a batch

    predictions = model.predict(img_array)
    score = tf.nn.softmax(predictions[0])

    print(
        "This image most likely belongs to {} with a {:.2f} percent confidence."
        .format(class_names[np.argmax(score)], 100 * np.max(score))
    )
else:
    print("No sunflower images found in the directory.")
```

```
No sunflower images found in the directory.
```

```
# Convert the model.
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save the model.
with open('model.tflite', 'wb') as f:
  f.write(tflite_model)
```

```
Saved artifact at '/tmp/tmpfhdcxflu'. The following endpoints are available:

* Endpoint 'serve'
  args_0 (POSITIONAL_ONLY): TensorSpec(shape=(None, 180, 180, 3), dtype=tf.float32, name='keras_tensor_15')
Output Type:
  TensorSpec(shape=(None, 1), dtype=tf.float32, name=None)
Captures:
  134995323004752: TensorSpec(shape=(), dtype=tf.resource, name=None)
  134995323002448: TensorSpec(shape=(), dtype=tf.resource, name=None)
  134995322997648: TensorSpec(shape=(), dtype=tf.resource, name=None)
  134995322995344: TensorSpec(shape=(), dtype=tf.resource, name=None)
  134995323002256: TensorSpec(shape=(), dtype=tf.resource, name=None)
  134995323000528: TensorSpec(shape=(), dtype=tf.resource, name=None)
  134995323003216: TensorSpec(shape=(), dtype=tf.resource, name=None)
  134995323000720: TensorSpec(shape=(), dtype=tf.resource, name=None)
  134995322999184: TensorSpec(shape=(), dtype=tf.resource, name=None)
  134995322998992: TensorSpec(shape=(), dtype=tf.resource, name=None)
```

```
TF_MODEL_FILE_PATH = 'model.tflite' # The default path to the saved TensorFlow Lite model

interpreter = tf.lite.Interpreter(model_path=TF_MODEL_FILE_PATH)
```

```
interpreter.get_signature_list()
```

```
{'serving_default': {'inputs': ['keras_tensor_15'], 'outputs': ['output_0']}}
```

```
classify_lite = interpreter.get_signature_runner('serving_default')
classify_lite
```

```
<tensorflow.lite.python.interpreter.SignatureRunner at 0x7ac6fdd1e550>
```

```
import pathlib

sunflower_url = "/content/flower_photos.tgz"  # Path to the .tgz file
data_dir = tf.keras.utils.get_file(
    'flower_photos.tgz',  # Use the original filename for extraction
    origin='file://' + sunflower_url,
    extract=True,
    cache_dir="/content"
)
data_dir = pathlib.Path(data_dir).with_suffix('')

# List the files in the sunflowers directory to find the actual filename
sunflower_files = list(data_dir.glob('sunflowers/*.jpg'))

# Check if any sunflower images were found
if sunflower_files:
    # Use the first sunflower image found
    sunflower_path = sunflower_files[0]

    # Now you can load and predict:
    img = tf.keras.utils.load_img(
        sunflower_path, target_size=(img_height, img_width)
    )
    img_array = tf.keras.utils.img_to_array(img) # This line was missing, causing the error
    img_array = tf.expand_dims(img_array, 0)
else:
    # If no sunflower images are found, load a default image or handle the case appropriately
    print("No sunflower images found in the directory. Using a default image.")
    # Example: Load a placeholder image. Replace with actual path if needed
    # Get a list of all image files in all subdirectories
    all_image_files = list(data_dir.glob('*/*.jpg'))

    # Check if there are any images at all
    if all_image_files:
        default_image_path = all_image_files[0]  # Use the first image found
    else:
        # If no images are found in the dataset, print the path being checked and
        # download the dataset or check if the path is correct
        print(f"Checking for images in: {data_dir}")

        # Download the dataset if it's not found locally
        # Or fix the path to 'sunflower_url' if it is incorrect
```

```
        !wget -nc -P /content https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tg

        # Update data_dir after download
        data_dir = pathlib.Path('/content/flower_photos')
        all_image_files = list(data_dir.glob('*/*.jpg'))
        if not all_image_files:  # Check again if the download worked
            raise FileNotFoundError("Dataset download failed or invalid path. Please check the URL and path.")

        default_image_path = all_image_files[0]

    img = tf.keras.utils.load_img(default_image_path, target_size=(img_height, img_width))
    img_array = tf.keras.utils.img_to_array(img)
    img_array = tf.expand_dims(img_array, 0)


# Get the correct input tensor name from the signature
signature = interpreter.get_signature_list()['serving_default']
# Access the first element of the 'inputs' list directly
input_tensor_name = signature['inputs'][0]

# Get the correct output tensor name from the signature
output_tensor_name = signature['outputs'][0] # Get output name from signature

# Use the correct input name in the call
predictions_lite = classify_lite(**{input_tensor_name: img_array})[output_tensor_name] # Use correct output name
```

```
No sunflower images found in the directory. Using a default image.
Checking for images in: /content/datasets/flower_photos_extracted
File '/content/flower_photos.tgz' already there; not retrieving.
```

```
print(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(score_lite)], 100 * np.max(score_lite))
)
```

```
This image most likely belongs to flower_photos with a 100.00 percent confidence.
```

```
import pathlib

sunflower_url = "/content/flower_photos.tgz"  # Path to the .tgz file
data_dir = tf.keras.utils.get_file(
    'flower_photos.tgz',  # Use the original filename for extraction
    origin='file://' + sunflower_url,
    extract=True,
    cache_dir="/content"
)
```