Flower Type Recognizer

Imports

In [1]:
```
%matplotlib inline
%config InlineBackend.figure_format = 'retina'

import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import tensorflow_datasets as tfds
import tf_keras
import logging
import json
import tensorflow_hub as hub
import pandas as pd
```

```
WARNING:tensorflow:From C:\Users\HP\PycharmProjects\pythonProject\venv\Lib\site-pack
ages\tf_keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is
deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

In [2]:
```
tfds.disable_progress_bar()
```

In [3]:
```
logger = tf.get_logger()
logger.setLevel(logging.ERROR)
physical_devices = tf.config.list_physical_devices('GPU')
for device in physical_devices:
    tf.config.experimental.set_memory_growth(device, True)
```

Load the Data

- Use Tensorflow dataset `oxford_flowers102`
- split it into
    - train
    - test
    - validation

In [4]:
```
# Split the data to train, val, test as the splits the original data provides
splits = ['train', 'validation', 'test']

# Load the 'oxford_flowers102' dataset and split it into `splits`
dataset, dataset_info = tfds.load('oxford_flowers102', as_supervised=True, with_inf

validation_set, test_set, training_set = dataset
len(training_set), len(validation_set), len(test_set)
```

Out[4]: (6149, 1020, 1020)

Explor the Dataset

In [5]:
```
dataset_info.features['image']
```

Out[5]: Image(shape=(None, None, 3), dtype=uint8)

In [6]:
```
dataset_info.features['label']
```

Out[6]: ClassLabel(shape=(), dtype=int64, num_classes=102)

In [7]:
```
total_examples = dataset_info.splits['train'].num_examples + dataset_info.splits['t
num_training_examples = len(training_set)
num_validation_examples = len(validation_set)
num_test_examples = len(test_set)

print(f'There are {total_examples:,} images in Total')
print(f'There are {num_training_examples:,} images in the training set')
print(f'There are {num_validation_examples:,} images in the validation set')
print(f'There are {num_test_examples:,} images in the test set')
```

```
There are 8,189 images in Total
There are 6,149 images in the training set
There are 1,020 images in the validation set
There are 1,020 images in the test set
```

Data Info Summary

Image:

- **Shape**: (None, None, 3)
- **dtype**: uint8

Label:

- **Number of Classes**: 102
- **dtype**: int64

Examples

- There are **7,169** images in Total
- There are **816** images in the training set
- There are **204** images in the validation set

- There are **6,149** images in the test set

Exploring The Images

Print first three images shape

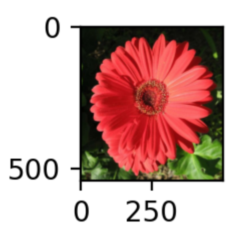
In [8]:
```
i = 1 # For counting which image we are in currently
for image, label in training_set.take(3):
    print(f'Image_{i} Shape: {image.shape}, Label: {label}')
    i += 1
```

```
Image_1 Shape: (542, 500, 3), Label: 40
Image_2 Shape: (748, 500, 3), Label: 76
Image_3 Shape: (500, 600, 3), Label: 42
```

Print first image

In [9]:
```
for image, label in training_set.take(1):
    image = image.numpy().squeeze() # to get rid of `1`
    label = label.numpy()

plt.figure(figsize=(2,1))
plt.imshow(image)
print(f"Label is {label}")
```

```
Label is 40
```



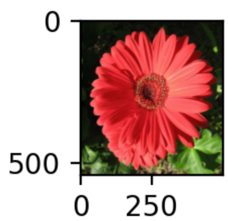
Attach the labels to theirs corresponding names

In [10]:
```
with open('label_map.json', 'r') as f:
    class_names = json.load(f)
for key, value in class_names.items():
    print(f"Label: `{key}` --  Name: `{value}`")
    if key == '5':
        break
```

```
Label: `0` --  Name: `pink primrose`
Label: `1` --  Name: `hard-leaved pocket orchid`
Label: `2` --  Name: `canterbury bells`
Label: `3` --  Name: `sweet pea`
Label: `4` --  Name: `english marigold`
Label: `5` --  Name: `tiger lily`
```

In [11]:
```python
for image, label in training_set.take(1):
    image = image.numpy().squeeze()
    label = str(label.numpy())
plt.figure(figsize=(2,1))
plt.imshow(image)
print(f"This is an image of {class_names[label]}")
```

```
This is an image of barbeton daisy
```



---

prepare The Dataset

Steps:

- Shuffle the data (shuffle each quarter for memory can handle)
- normalize the data
- make batches (size 64)
- prefetch (prepare the next batch while the first is execution)

In [12]:
```python
batch_size = 64
image_size = 224

def normalize(image, label):
    image = tf.cast(image, tf.float32)
    image = tf.image.resize(image, (image_size, image_size)) #resize to (224,224,3)
    image /= 255 # (normalize the range of 0-255 -> 0-1)
    return image, label

training_batches = training_set.shuffle(num_training_examples//4).map(normalize).ba

validation_batches = validation_set.shuffle(num_validation_examples//4).map(normali

testing_batches = test_set.shuffle(num_test_examples // 4).map(normalize).batch(bat
```

Use A Pre-trained Module

Steps:

- Load the module
- Freeze the weights
- Decide the input shape

In [13]:
```
mobilenet_URL = "https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4

# Load the MobileNet model from TensorFlow Hub
mobilenet_model = hub.KerasLayer(mobilenet_URL, input_shape=(224, 224, 3), trainabl
```

In [14]:
```
# Freeze the weights, small data set with similar data.
mobilenet_model.trainable = False
```

In [15]:
```
from tf_keras.utils import plot_model
```

Build the network

Steps:

- Connect the old network
- **Layer1**: Activation: `relu` - Height: `256`
- **Layer2**: Activation: `relu` - Height: `128`
- **Output Layer**: Activation: `Softmax` - Height: `102`

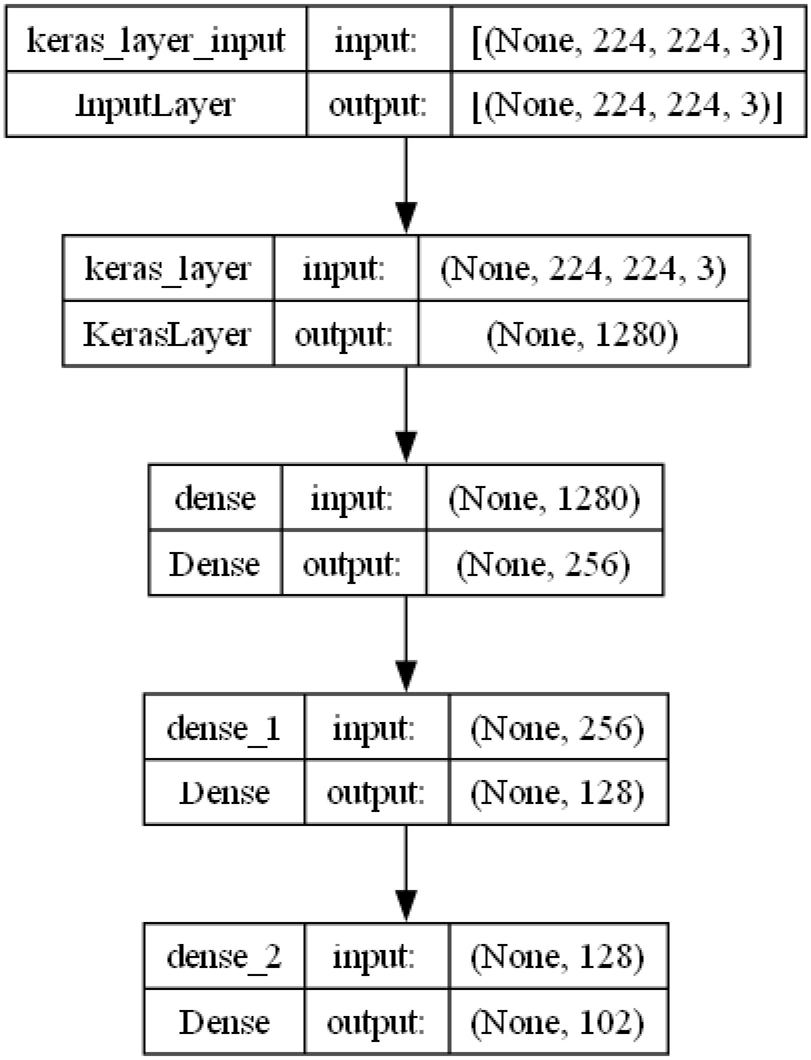
In [18]:
```
import tf_keras

number_of_classes = len(class_names)

model = tf_keras.Sequential([
        mobilenet_model,
        tf_keras.layers.Dense(256, activation='relu'),
        tf_keras.layers.Dense(128, activation='relu'),
        tf_keras.layers.Dense(number_of_classes,activation='softmax')
])

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 keras_layer (KerasLayer)    (None, 1280)              2257984

 dense_3 (Dense)             (None, 256)               327936

 dense_4 (Dense)             (None, 128)               32896

 dense_5 (Dense)             (None, 102)               13158

=================================================================
Total params: 2631974 (10.04 MB)
Trainable params: 373990 (1.43 MB)
Non-trainable params: 2257984 (8.61 MB)
_________________________________________________________________
```

Network plot

In [17]:
```
plot_model(model, to_file="mobilenet_model.png", show_shapes=True, show_layer_names
```

Out[17]:



Training The Module

Fetures:

- optimizer = `adam` loss = `SGD`
- Patience: `5` (validation accuracy loss)

In [19]:
```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

In [20]:
```
EPOCHS = 12
early_stopping = tf_keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)

history = model.fit(training_batches,
```

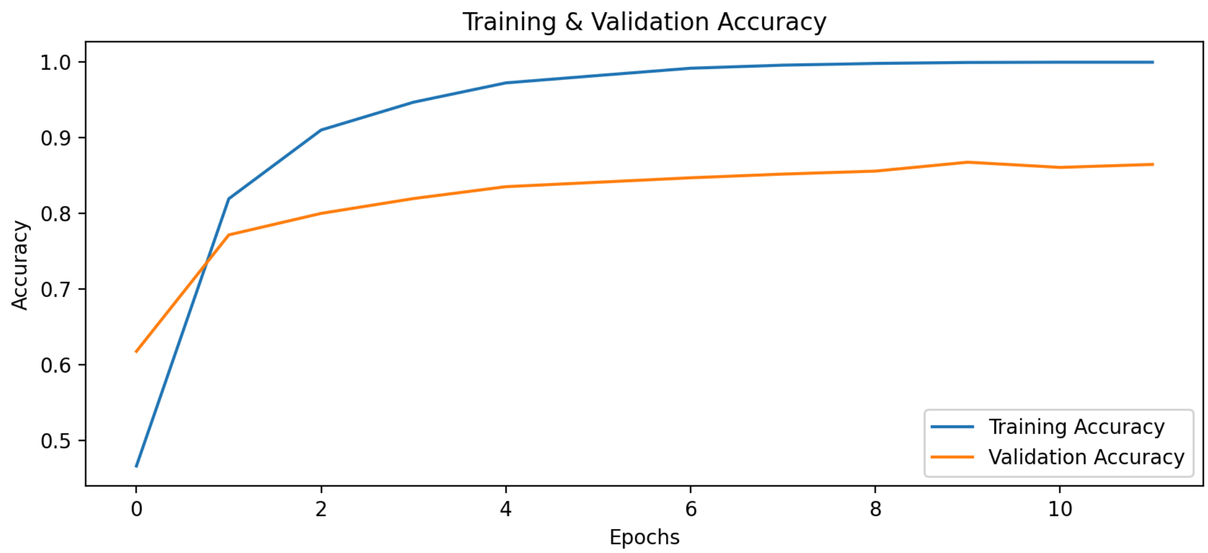
```
                        epochs=EPOCHS,
                        validation_data=validation_batches,
                        callbacks=[early_stopping])
```

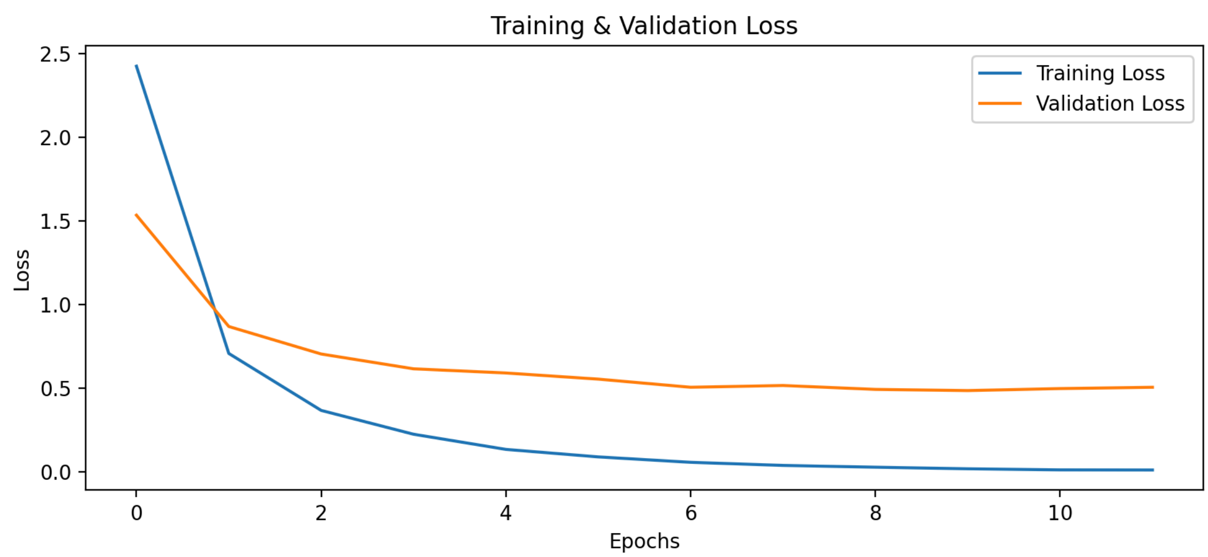
```
Epoch 1/12
97/97 [==============================] - 75s 680ms/step - loss: 2.4244 - accuracy:
0.4661 - val_loss: 1.5335 - val_accuracy: 0.6176
Epoch 2/12
97/97 [==============================] - 57s 576ms/step - loss: 0.7066 - accuracy:
0.8193 - val_loss: 0.8677 - val_accuracy: 0.7716
Epoch 3/12
97/97 [==============================] - 65s 659ms/step - loss: 0.3660 - accuracy:
0.9102 - val_loss: 0.7033 - val_accuracy: 0.8000
Epoch 4/12
97/97 [==============================] - 56s 573ms/step - loss: 0.2235 - accuracy:
0.9470 - val_loss: 0.6148 - val_accuracy: 0.8196
Epoch 5/12
97/97 [==============================] - 55s 564ms/step - loss: 0.1328 - accuracy:
0.9725 - val_loss: 0.5898 - val_accuracy: 0.8353
Epoch 6/12
97/97 [==============================] - 64s 651ms/step - loss: 0.0879 - accuracy:
0.9823 - val_loss: 0.5531 - val_accuracy: 0.8412
Epoch 7/12
97/97 [==============================] - 62s 632ms/step - loss: 0.0556 - accuracy:
0.9919 - val_loss: 0.5044 - val_accuracy: 0.8471
Epoch 8/12
97/97 [==============================] - 61s 617ms/step - loss: 0.0370 - accuracy:
0.9959 - val_loss: 0.5149 - val_accuracy: 0.8520
Epoch 9/12
97/97 [==============================] - 58s 592ms/step - loss: 0.0263 - accuracy:
0.9982 - val_loss: 0.4917 - val_accuracy: 0.8559
Epoch 10/12
97/97 [==============================] - 68s 690ms/step - loss: 0.0168 - accuracy:
0.9995 - val_loss: 0.4845 - val_accuracy: 0.8676
Epoch 11/12
97/97 [==============================] - 68s 695ms/step - loss: 0.0104 - accuracy:
0.9998 - val_loss: 0.4966 - val_accuracy: 0.8608
Epoch 12/12
97/97 [==============================] - 63s 645ms/step - loss: 0.0097 - accuracy:
0.9998 - val_loss: 0.5043 - val_accuracy: 0.8647
```

In [21]:

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure(figsize=(10, 4))
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training & Validation Accuracy')
plt.legend()
plt.show()
```

```
plt.figure(figsize=(10, 4))
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training & Validation Loss')
plt.legend()
plt.show()
```





Testing The module

In [22]:
```
loss, accuracy = model.evaluate(testing_batches)

print(f"Loss: {loss * 100}%\nAccuracy: {accuracy * 100}%")
```

```
16/16 [==============================] - 10s 572ms/step - loss: 0.4033 - accuracy: 0.8853
Loss: 40.3276264667511%
Accuracy: 88.52941393852234%
```

Save The Module

In [23]:
```
saved_keras_model_filepath = f'./Flower_Recognizer.h5'

model.save(saved_keras_model_filepath)
```

```
C:\Users\HP\PycharmProjects\pythonProject\venv\Lib\site-packages\tf_keras\src\engine\training.py:3098: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native TF-Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

In [24]:
```
!unzip -t {saved_keras_model_filepath}
```

```
'unzip' is not recognized as an internal or external command,
operable program or batch file.
```

In [25]:
```
reloaded_keras_model = tf_keras.models.load_model(
    saved_keras_model_filepath,
    custom_objects={'KerasLayer': hub.KerasLayer}
)
```

In [33]:
```
(model.get_weights()[0][0][0] == reloaded_keras_model.get_weights()[0][0][0])[:5]
```

Out[33]:
```
array([[ True],
       [ True],
       [ True],
       [ True],
       [ True]])
```

Some tests on the module

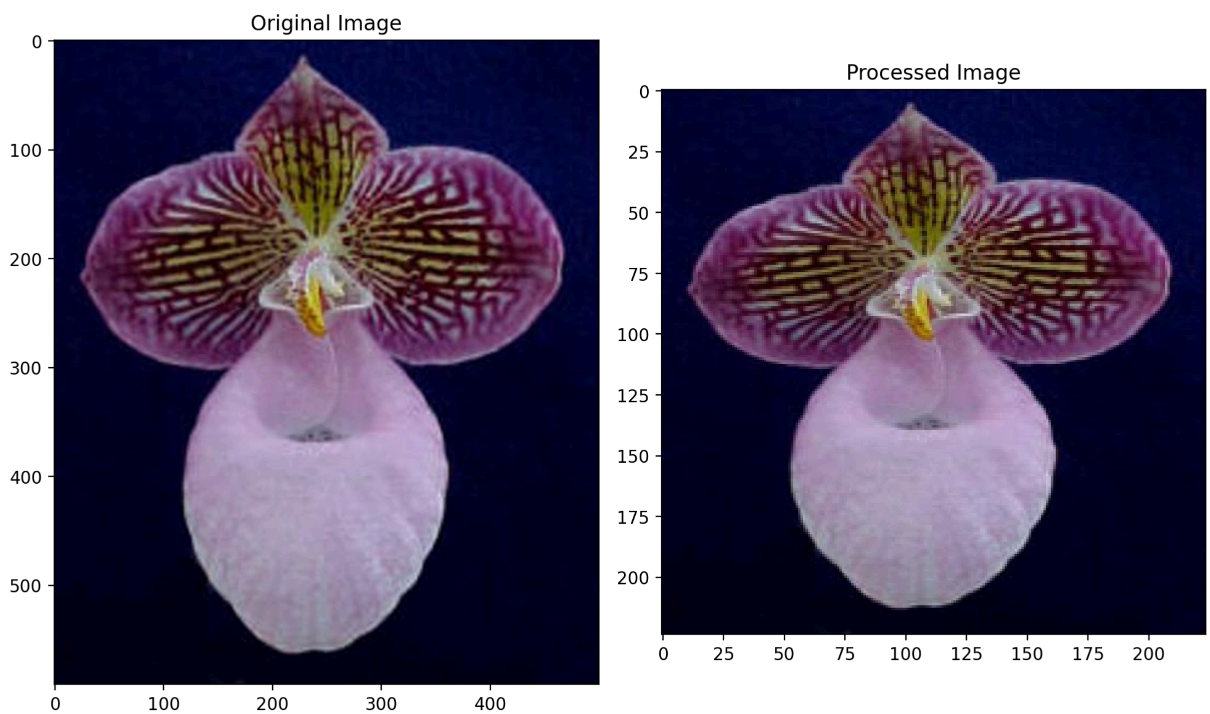
1- Prepare The Image:

- resize to what module take (224,224,3)
- normalize from (0-255) -> (0-1)

In [27]:
```
pre_image_size = 224

def process_image(image):
    image = tf.convert_to_tensor(image, dtype=tf.float32)
    image = tf.image.resize(image, (pre_image_size, pre_image_size))
    image /= 255
    image = image.numpy()
    return image
```

2- Print The Processed Image

In [28]:
```
from PIL import Image

image_path = './test_images/hard-leaved_pocket_orchid.jpg'
im = Image.open(image_path)
test_image = np.asarray(im)

processed_test_image = process_image(test_image)

fig, (ax1, ax2) = plt.subplots(figsize=(10,10), ncols=2)
ax1.imshow(test_image)
ax1.set_title('Original Image')
ax2.imshow(processed_test_image)
ax2.set_title('Processed Image')
plt.tight_layout()
plt.show()
```



3- Make Predict Function

In [29]:
```python
def predict(image_path, model, k):
    image = Image.open(image_path)
    image = np.asarray(image)
    image = process_image(image)
    image = np.expand_dims(image, axis=0)

    prediction_dict = model.predict(image)
    pandas_predictions = pd.DataFrame(prediction_dict)
    sorted_prediction = pandas_predictions.T.sort_values(by=0, ascending=False).hea
    probs = sorted_prediction.values[0]
    classes = sorted_prediction.keys().tolist()
    # print(sorted_prediction)
    return list(probs), list(classes)
```

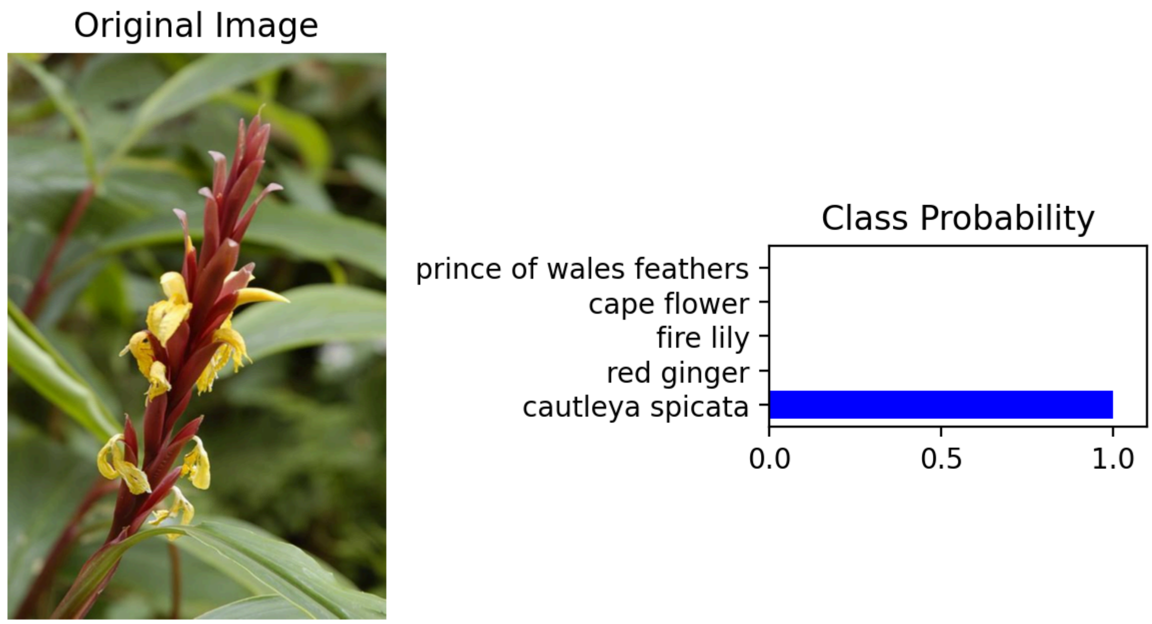
4- Function to plot the images

In [30]:
```python
def plot_image_probs(image_path, model= model, k=5):
    probs, classes = predict(image_path, model, k)
    names = [class_names[f"{i}"] for i in classes]

    fig, (ax1, ax2) = plt.subplots(figsize=(6, 9), ncols=2)

    ax1.imshow(Image.open(image_path))
    ax1.set_title('Original Image')
    ax1.axis('off')

    ax2.barh(names, probs, color='blue')
    ax2.set_aspect(0.1)
    ax2.set_yticks(np.arange(len(names)))
    ax2.set_yticklabels(names)
    ax2.set_title('Class Probability')
    ax2.set_xlim(0, 1.1)

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

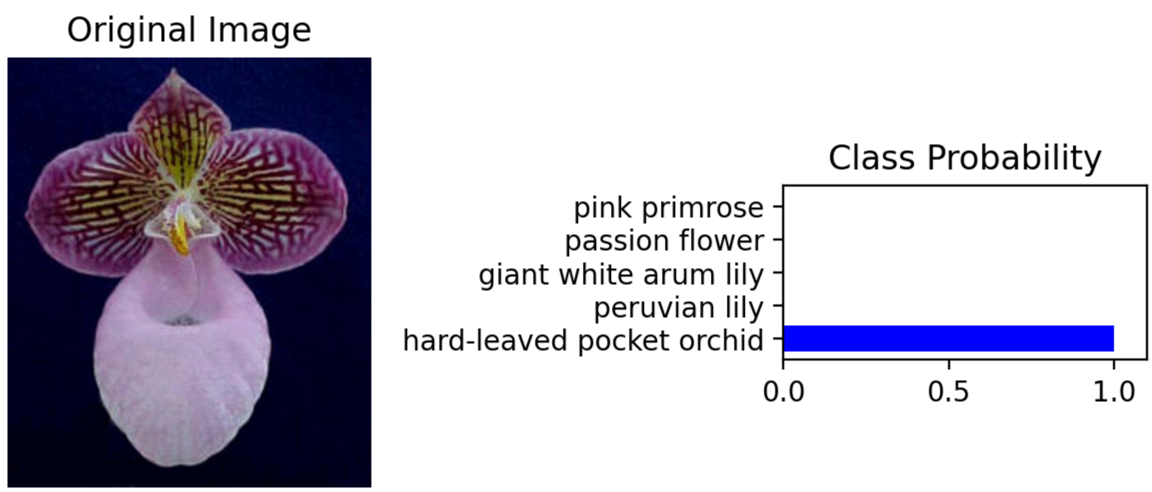
5- Test The photos on `test_images`

In [31]:
```python
import os

directory = "./test_images"
for i in os.listdir(directory):
    path = directory + "/" + i
    plot_image_probs(path, model, 5)
```

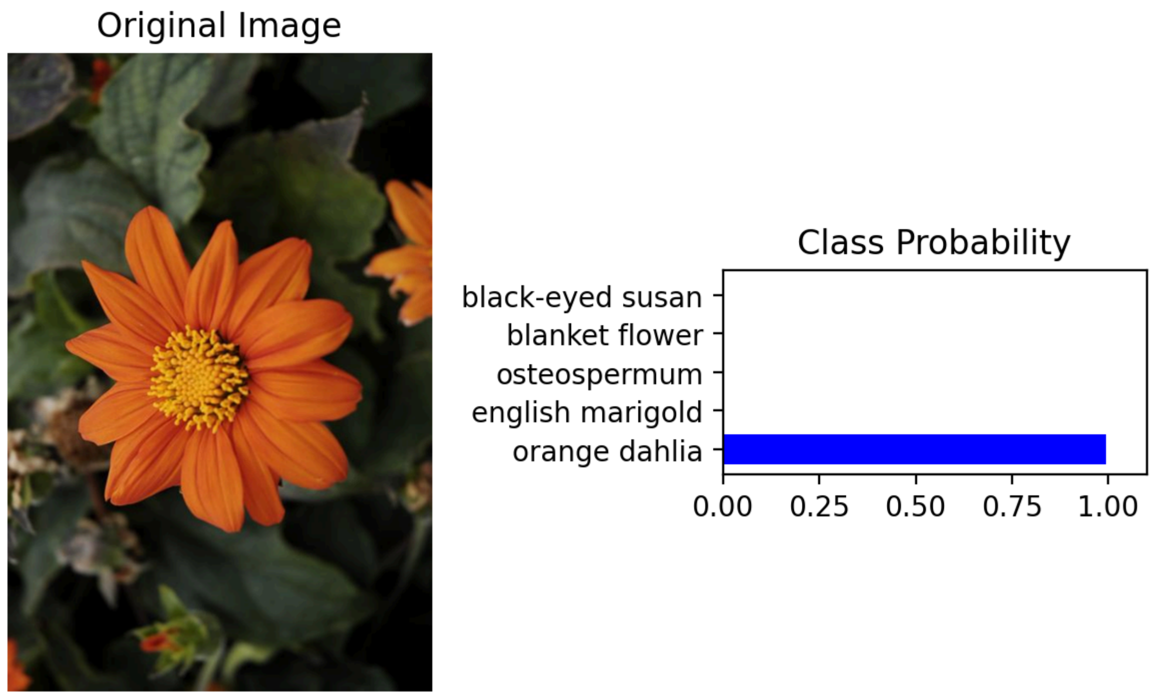
```
1/1 [==============================] - 2s 2s/step
```




```
1/1 [==============================] - 0s 53ms/step
```




```
1/1 [==============================] - 0s 50ms/step
```




```
1/1 [==============================] - 0s 50ms/step
```