In [1]:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:

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

In [3]:

```
import pathlib
data_url = "https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz"
data_dir = tf.keras.utils.get_file('flower_photos', origin=data_url, untar=True)
data_dir = pathlib.Path(data_dir)
```

```
Downloading data from https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz
228813984/228813984 [==============================] - 5s 0us/step
```

In [4]:

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

```
3670
```

In [5]:

```
roses = list(data_dir.glob('roses/*'))
PIL.Image.open(str(roses[8]))
```

Out[5]:

In [6]:

```
sunflowers = list(data_dir.glob('sunflowers/*'))
```

```
PIL.Image.open(str(sunflowers[9]))
```

Out[6]:

In [7]:

```
dandelions = list(data_dir.glob('dandelion/*'))
PIL.Image.open(str(dandelions[45]))
```

Out[7]:

In [8]:

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

In [9]:

```
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 5 classes.
Using 2936 files for training.
```

In [10]:

```
test_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 5 classes.
Using 734 files for validation.
```

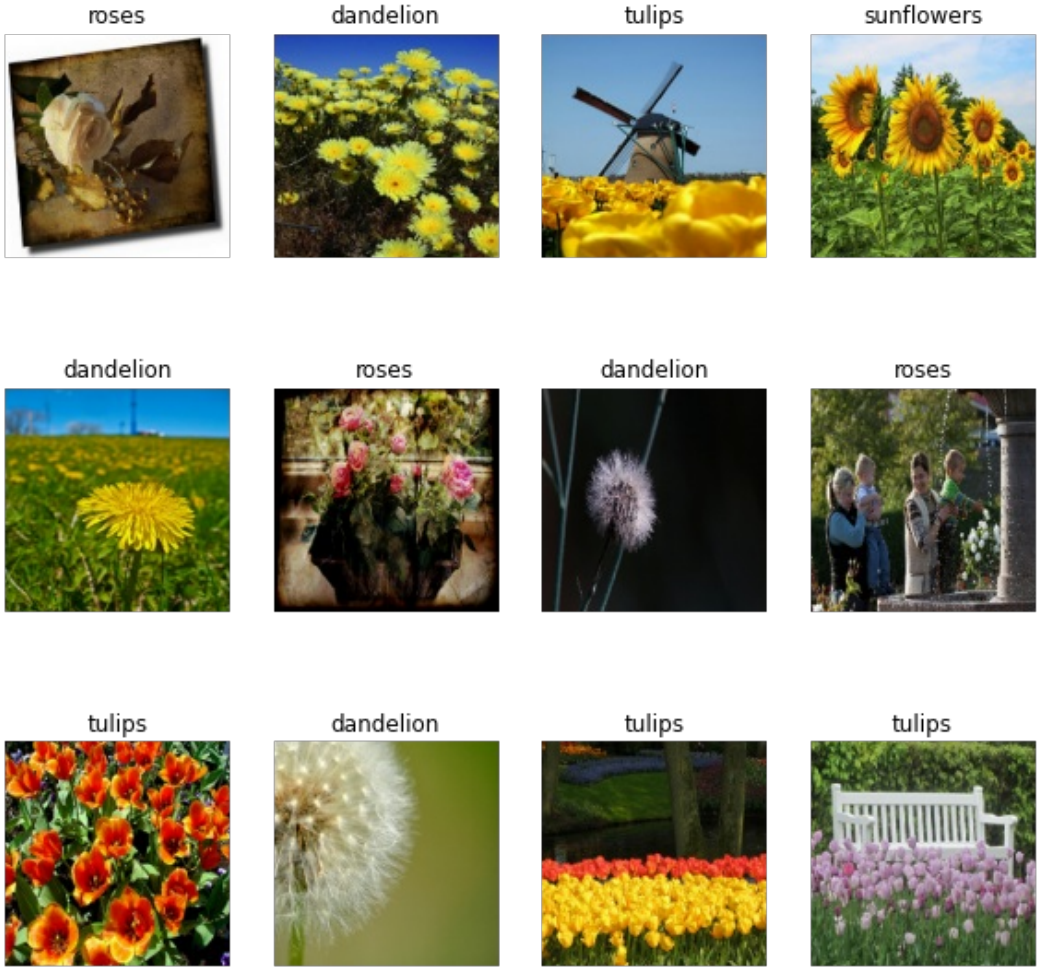
In [11]:

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

```
['daisy', 'dandelion', 'roses', 'sunflowers', 'tulips']
```

In [12]:

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
  for i in range(12):
    ax = plt.subplot(3, 4, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_name[labels[i]])
    plt.axis("off")
```



In [13]:

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

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

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

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

```
(32, 180, 180, 3)
(32,)
(32, 180, 180, 3)
(32,)
(32, 180, 180, 3)
(32,)
(32, 180, 180, 3)
(32,)
(32, 180, 180, 3)
(32,)
(32, 180, 180, 3)
(32,)
(32, 180, 180, 3)
(32,)
(32, 180, 180, 3)
(32,)
(32, 180, 180, 3)
(32,)
(32, 180, 180, 3)
(32,)
(32, 180, 180, 3)
(32,)
(32, 180, 180, 3)
(32,)
(32, 180, 180, 3)
(32,)
(24, 180, 180, 3)
(24,)
```

In [14]:

```
AUTOTUNE = tf.data.AUTOTUNE

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

In [15]:

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

In [16]:

```
normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
image_batch, labels_batch = next(iter(normalized_ds))
first_image = image_batch[0]
# Notice the pixel values are now in `[0,1]`.
print(np.min(first_image), np.max(first_image))
```

```
0.0 1.0
```

In [17]:

```
num_classes = len(class_name)
model = Sequential([
    layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
    layers.Conv2D(16, 3 ,padding='same' , activation='relu'),
    layers.MaxPool2D(),
    layers.Conv2D(32, 3 ,padding='same' , activation='relu'),
    layers.MaxPool2D(),
    layers.Conv2D(64, 3 ,padding='same' , activation='relu'),
    layers.MaxPool2D(),
    layers.Conv2D(128, 3 ,padding='same' , activation='relu'),
    layers.MaxPool2D(),
    layers.Flatten(),
    layers.Dense(256,activation='relu'),
    layers.Dense(num_classes),

])
```

In [18]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_1 (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)        4640

 max_pooling2d_1 (MaxPooling  (None, 45, 45, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 45, 45, 64)        18496

 max_pooling2d_2 (MaxPooling  (None, 22, 22, 64)       0
 2D)

 conv2d_3 (Conv2D)           (None, 22, 22, 128)       73856

 max_pooling2d_3 (MaxPooling  (None, 11, 11, 128)      0
 2D)

 flatten (Flatten)           (None, 15488)             0

 dense (Dense)               (None, 256)               3965184

 dense_1 (Dense)             (None, 5)                 1285

=================================================================
Total params: 4,063,909
Trainable params: 4,063,909
Non-trainable params: 0
_________________________________________________________________
```

In [19]:

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

In [20]:

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

```
Epoch 1/10
92/92 [==============================] - 80s 863ms/step - loss: 1.3827 - accuracy: 0.3886
- val_loss: 1.0927 - val_accuracy: 0.5477
Epoch 2/10
92/92 [==============================] - 80s 866ms/step - loss: 1.0380 - accuracy: 0.5797
- val_loss: 1.0145 - val_accuracy: 0.5831
Epoch 3/10
92/92 [==============================] - 80s 865ms/step - loss: 0.9118 - accuracy: 0.6410
- val_loss: 0.8895 - val_accuracy: 0.6376
Epoch 4/10
92/92 [==============================] - 79s 856ms/step - loss: 0.7748 - accuracy: 0.6962
- val_loss: 0.8691 - val_accuracy: 0.6621
Epoch 5/10
92/92 [==============================] - 80s 870ms/step - loss: 0.6772 - accuracy: 0.7333
- val_loss: 0.9149 - val_accuracy: 0.6703
Epoch 6/10
92/92 [==============================] - 80s 870ms/step - loss: 0.5261 - accuracy: 0.8007
- val_loss: 0.9638 - val_accuracy: 0.6608
Epoch 7/10
92/92 [==============================] - 78s 853ms/step - loss: 0.4132 - accuracy: 0.8518
- val_loss: 1.1648 - val_accuracy: 0.6267
Epoch 8/10
92/92 [==============================] - 80s 869ms/step - loss: 0.2985 - accuracy: 0.8883
```

```
- val_loss: 1.1880 - val_accuracy: 0.6703
Epoch 9/10
92/92 [==============================] - 79s 856ms/step - loss: 0.1954 - accuracy: 0.9329
- val_loss: 1.3127 - val_accuracy: 0.6839
Epoch 10/10
92/92 [==============================] - 80s 870ms/step - loss: 0.1262 - accuracy: 0.9567
- val_loss: 1.4783 - val_accuracy: 0.6608
```

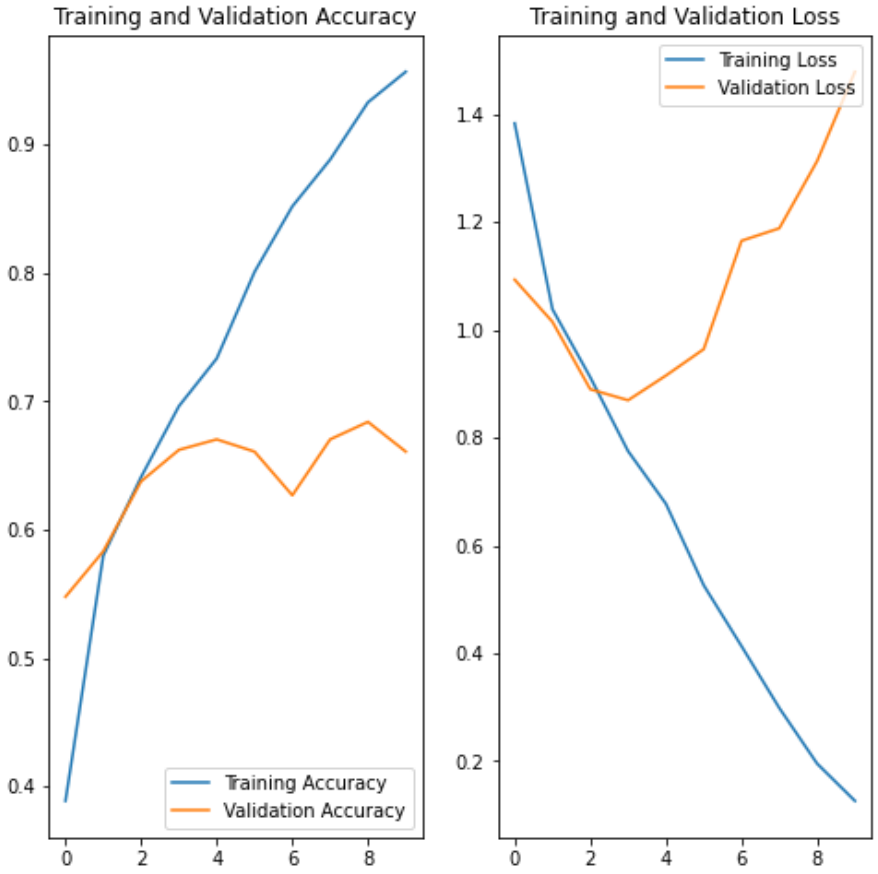
In [21]:

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs=10
epoch_range = range(epochs)
```

In [22]:

```
plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epoch_range, acc, label='Training Accuracy')
plt.plot(epoch_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

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



In [23]:

```
model.save('Flower.h5')
```

In [24]:

```
flower_url = 'https://storage.googleapis.com/download.tensorflow.org/example_images/592px
-Red_sunflower.jpg'
```

```
flower_path = tf.keras.utils.get_file('Red_flower', origin=flower_url )
```

```
Downloading data from https://storage.googleapis.com/download.tensorflow.org/example_imag
es/592px-Red_sunflower.jpg
117948/117948 [==============================] - 0s 0us/step
```

In [25]:

```
PIL.Image.open(flower_path)
```

Out[25]:

In [ ]:

```

```