In [8]:
```
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

import pathlib

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

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

```
3670
```

In [9]:
```
roses = list(data_dir.glob('roses/*'))
PIL.Image.open(str(roses[0]))
```

Out[9]:

In [10]:
```
# Training split
train_ds = tf.keras.utils.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=(180, 180),
    batch_size=32)
```

```
Found 3670 files belonging to 5 classes.
Using 2936 files for training.
```

In [11]:
```
# Testing or Validation split
val_ds = tf.keras.utils.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=(180,180),
    batch_size=32)
```

```
Found 3670 files belonging to 5 classes.
Using 734 files for validation.
```

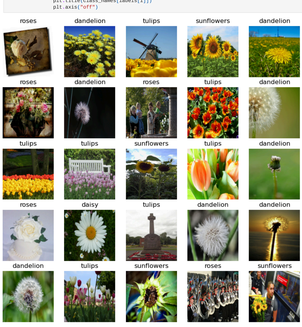
In [12]:
```
class_names = train_ds.class_names
print(class_names)
```

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

In [13]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))

for images, labels in train_ds.take(1):
    for i in range(25):
        ax = plt.subplot(5, 5, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(class_names[labels[i]])
        plt.axis("off")
```



In [15]:
```
# Define the number of classes (you need to define class_names somewhere in your code)
class_names = ['daisy', 'dandelion', 'roses', 'sunflowers', 'tulips']
num_classes = len(class_names)

# Define the model
model = Sequential([
    layers.Input(shape=(180, 180, 3)),
    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.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(num_classes)
])

# Print the model summary
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
rescaling_1 (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
flatten_1 (Flatten)	(None, 30976)	0
dense_2 (Dense)	(None, 128)	3,965,056
dense_3 (Dense)	(None, 5)	645

Total params: 3,989,285 (15.22 MB)
Trainable params: 3,989,285 (15.22 MB)
Non-trainable params: 0 (0.00 B)

In [23]:
```
# Compile the model
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

# Train the model
epochs = 10
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=epochs
)

# Evaluate the model on the validation set
val_loss, val_acc = model.evaluate(val_ds)

# Predict on the validation set
y_pred = np.argmax(model.predict(val_ds), axis=-1)

# Get true labels
y_true = np.concatenate([y for x, y in val_ds], axis=0)

# Generate classification report
from sklearn.metrics import classification_report

report = classification_report(y_true, y_pred, target_names=class_names)
print(report)
```

```
Epoch 1/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 16s 144ms/step - accuracy: 0.9743 - loss: 0.0676 - val_accuracy: 0.6458 - val_loss: 1.9442
Epoch 2/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 13s 146ms/step - accuracy: 0.9795 - loss: 0.0626 - val_accuracy: 0.6676 - val_loss: 1.8916
Epoch 3/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 17s 189ms/step - accuracy: 0.9957 - loss: 0.0173 - val_accuracy: 0.6676 - val_loss: 2.1380
Epoch 4/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 22s 234ms/step - accuracy: 1.0000 - loss: 0.0035 - val_accuracy: 0.6649 - val_loss: 2.2106
Epoch 5/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 22s 237ms/step - accuracy: 0.9999 - loss: 0.0024 - val_accuracy: 0.6580 - val_loss: 2.6211
Epoch 6/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 19s 207ms/step - accuracy: 0.9673 - loss: 0.1177 - val_accuracy: 0.6553 - val_loss: 1.8801
Epoch 7/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 19s 209ms/step - accuracy: 0.9907 - loss: 0.0273 - val_accuracy: 0.6335 - val_loss: 2.1043
Epoch 8/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 18s 199ms/step - accuracy: 0.9921 - loss: 0.0293 - val_accuracy: 0.6199 - val_loss: 2.5368
Epoch 9/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 21s 230ms/step - accuracy: 0.9743 - loss: 0.0843 - val_accuracy: 0.6499 - val_loss: 2.0772
Epoch 10/10
92/92 ━━━━━━━━━━━━━━━━━━━━ 22s 240ms/step - accuracy: 0.9983 - loss: 0.0051 - val_accuracy: 0.6526 - val_loss: 2.1659
23/23 ━━━━━━━━━━━━━━━━━━━━ 2s 96ms/step - accuracy: 0.6349 - loss: 2.1946
23/23 ━━━━━━━━━━━━━━━━━━━━ 3s 99ms/step
              precision    recall  f1-score   support

       daisy       0.19      0.19      0.19       129
   dandelion       0.25      0.28      0.27       176
       roses       0.19      0.18      0.19       120
   sunflowers       0.16      0.14      0.15       152
      tulips       0.22      0.22      0.22       157

    accuracy                           0.21       734
   macro avg       0.20      0.20      0.20       734
weighted avg       0.21      0.21      0.21       734
```

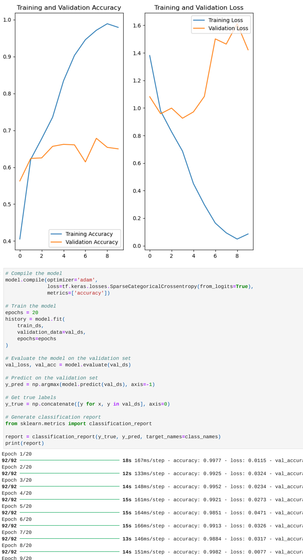
In [22]:
```
#Accuracy
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

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

#epochs
epochs_range = range(epochs)

#Plotting graphs
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()
```



In [24]:
```
# Compile the model
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

# Train the model
epochs = 20
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=epochs
)

# Evaluate the model on the validation set
val_loss, val_acc = model.evaluate(val_ds)

# Predict on the validation set
y_pred = np.argmax(model.predict(val_ds), axis=-1)

# Get true labels
y_true = np.concatenate([y for x, y in val_ds], axis=0)

# Generate classification report
from sklearn.metrics import classification_report

report = classification_report(y_true, y_pred, target_names=class_names)
print(report)
```

```
Epoch 1/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 16s 167ms/step - accuracy: 0.9977 - loss: 0.0115 - val_accuracy: 0.6689 - val_loss: 2.5806
Epoch 2/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 12s 133ms/step - accuracy: 0.9925 - loss: 0.0324 - val_accuracy: 0.6580 - val_loss: 2.4683
Epoch 3/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 14s 148ms/step - accuracy: 0.9952 - loss: 0.0234 - val_accuracy: 0.6662 - val_loss: 2.7037
Epoch 4/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 15s 161ms/step - accuracy: 0.9921 - loss: 0.0273 - val_accuracy: 0.6267 - val_loss: 2.8589
Epoch 5/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 15s 164ms/step - accuracy: 0.9851 - loss: 0.0471 - val_accuracy: 0.6390 - val_loss: 2.6442
Epoch 6/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 15s 166ms/step - accuracy: 0.9913 - loss: 0.0326 - val_accuracy: 0.6022 - val_loss: 2.6565
Epoch 7/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 13s 146ms/step - accuracy: 0.9884 - loss: 0.0317 - val_accuracy: 0.6512 - val_loss: 2.5424
Epoch 8/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 14s 151ms/step - accuracy: 0.9982 - loss: 0.0077 - val_accuracy: 0.6512 - val_loss: 2.6663
Epoch 9/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 14s 149ms/step - accuracy: 0.9991 - loss: 0.0034 - val_accuracy: 0.6567 - val_loss: 2.6091
Epoch 10/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 14s 153ms/step - accuracy: 1.0000 - loss: 3.5889e-04 - val_accuracy: 0.6567 - val_loss: 2.7070
Epoch 11/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 16s 171ms/step - accuracy: 1.0000 - loss: 2.1926e-04 - val_accuracy: 0.6580 - val_loss: 2.7617
Epoch 12/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 16s 177ms/step - accuracy: 1.0000 - loss: 1.6937e-04 - val_accuracy: 0.6580 - val_loss: 2.8085
Epoch 13/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 19s 205ms/step - accuracy: 1.0000 - loss: 1.2953e-04 - val_accuracy: 0.6580 - val_loss: 2.8486
Epoch 14/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 16s 173ms/step - accuracy: 1.0000 - loss: 1.0320e-04 - val_accuracy: 0.6580 - val_loss: 2.8842
Epoch 15/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 17s 181ms/step - accuracy: 1.0000 - loss: 8.7264e-05 - val_accuracy: 0.6594 - val_loss: 2.9161
Epoch 16/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 16s 172ms/step - accuracy: 1.0000 - loss: 6.8263e-05 - val_accuracy: 0.6594 - val_loss: 2.9448
Epoch 17/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 16s 169ms/step - accuracy: 1.0000 - loss: 6.4049e-05 - val_accuracy: 0.6580 - val_loss: 2.9719
Epoch 18/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 15s 162ms/step - accuracy: 1.0000 - loss: 5.2259e-05 - val_accuracy: 0.6580 - val_loss: 2.9978
Epoch 19/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 15s 160ms/step - accuracy: 1.0000 - loss: 4.7733e-05 - val_accuracy: 0.6580 - val_loss: 3.0218
Epoch 20/20
92/92 ━━━━━━━━━━━━━━━━━━━━ 16s 169ms/step - accuracy: 1.0000 - loss: 4.3818e-05 - val_accuracy: 0.6594 - val_loss: 3.0449
23/23 ━━━━━━━━━━━━━━━━━━━━ 1s 56ms/step - accuracy: 0.6707 - loss: 2.9890
23/23 ━━━━━━━━━━━━━━━━━━━━ 2s 68ms/step
              precision    recall  f1-score   support

       daisy       0.18      0.19      0.18       129
   dandelion       0.24      0.26      0.25       176
       roses       0.18      0.15      0.16       120
   sunflowers       0.20      0.20      0.20       152
      tulips       0.21      0.22      0.21       157

    accuracy                           0.21       734
   macro avg       0.20      0.20      0.20       734
```

```
weighted avg       0.20      0.21      0.20       734
```

In [ ]: