```
import os
import numpy as np

import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.preprocessing.image import load_img, ImageDataGenerator
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Dropout, Flatten
```

```
import os

count = 0
base_path = '/content/drive/MyDrive/Flower Recognition/Images'
dirs = os.listdir(base_path)

for dir in dirs:
    files = os.listdir(os.path.join(base_path, dir))
    print(dir + ' Folder has ' + str(len(files)) + ' Images')
    count += len(files)

print(" Total Images:", count)
```

```
dandelion Folder has 1062 Images
rose Folder has 794 Images
tulip Folder has 994 Images
daisy Folder has 764 Images
sunflower Folder has 743 Images
 Total Images: 4357
```

```
base_dir = '/content/drive/MyDrive/Flower Recognition/Images'
img_size = 180
batch = 32
```

```
train_ds = tf.keras.utils.image_dataset_from_directory( base_dir,
                                                       seed = 123,
                                                       validation_split=0.2,
                                                       subset = 'training',
                                                       batch_size=batch,
                                                       image_size=(img_size,img_size))

val_ds = tf.keras.utils.image_dataset_from_directory( base_dir,
                                                       seed = 123,
                                                       validation_split=0.2,
                                                       subset = 'validation',
                                                       batch_size=batch,
                                                       image_size=(img_size,img_size))
```

```
Found 4357 files belonging to 5 classes.
Using 3486 files for training.
Found 4357 files belonging to 5 classes.
Using 871 files for validation.
```

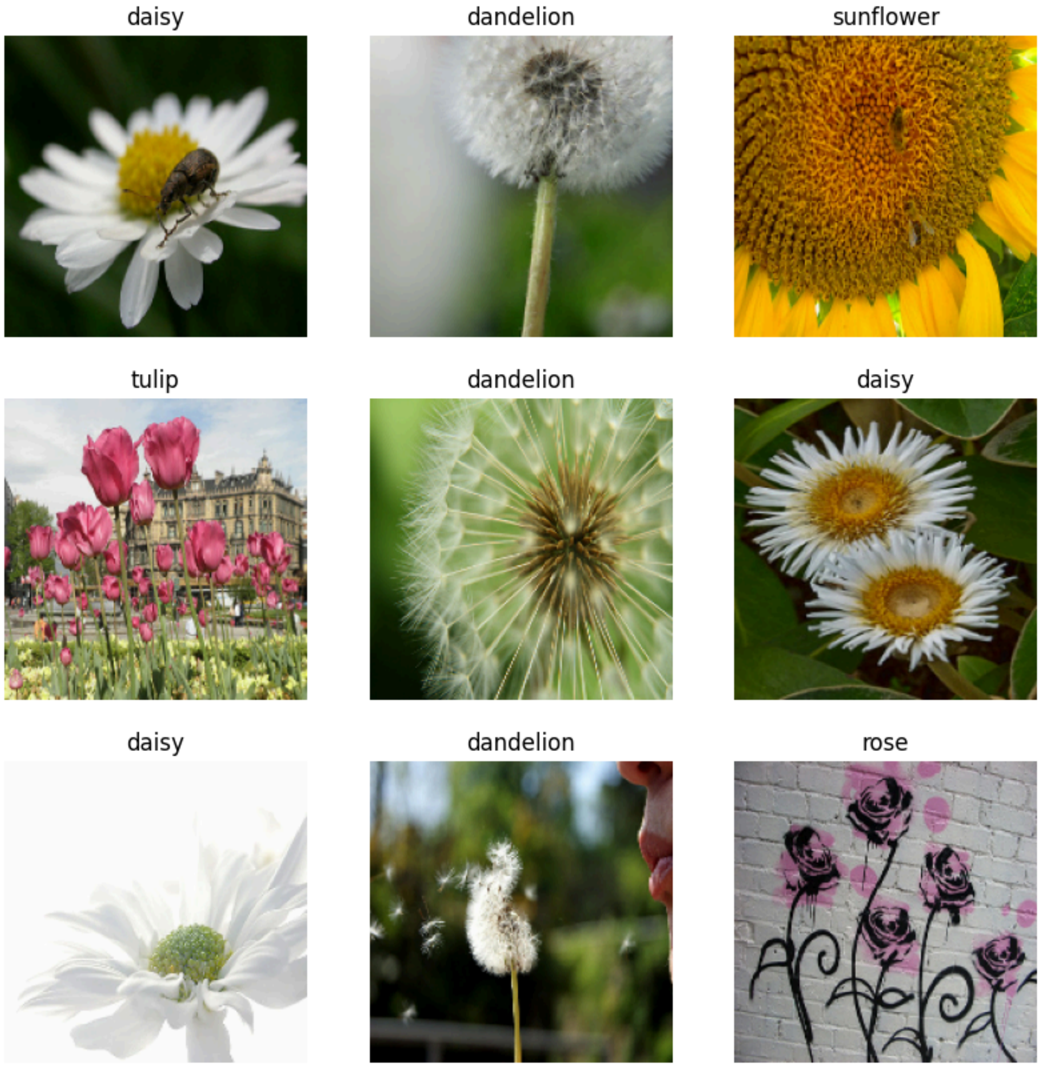
```
flower_names = train_ds.class_names
flower_names
```

```
['daisy', 'dandelion', 'rose', 'sunflower', 'tulip']
```

```
import matplotlib.pyplot as plt
```

```
i = 0
plt.figure(figsize=(10,10))

for images, labels in train_ds.take(1):
    for i in range(9):
        plt.subplot(3,3, i+1)
        plt.imshow(images[i].numpy().astype('uint8'))
        plt.title(flower_names[labels[i]])
        plt.axis('off')
```



```
AUTOTUNE = tf.data.AUTOTUNE


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


val_ds = val_ds.cache().prefetch(buffer_size = AUTOTUNE)


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

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/preprocessing/tf_data_layer.py:19: UserWarning: Do not pass an `input_shape`/`i
  super().__init__(**kwargs)
```

```
i = 0
plt.figure(figsize=(10,10))

for images, labels in train_ds.take(1):
    for i in range(9):
        images = data_augmentation(images)
        plt.subplot(3,3, i+1)
        plt.imshow(images[0].numpy().astype('uint8'))
        plt.axis('off')
```

```
model = Sequential([
    data_augmentation,
    layers.Rescaling(1./255),
    Conv2D(16, 3, padding='same', activation='relu'),
    MaxPooling2D(),
    Conv2D(32, 3, padding='same', activation='relu'),
    MaxPooling2D(),
    Conv2D(64, 3, padding='same', activation='relu'),
    MaxPooling2D(),
    Dropout(0.2),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(5)
])


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


model.summary()
```

```
Model: "sequential_1"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ sequential (Sequential)         │ (None, 180, 180, 3)    │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ rescaling (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 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout (Dropout)               │ (None, 22, 22, 64)     │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ flatten (Flatten)               │ (None, 30976)          │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense (Dense)                   │ (None, 128)            │     3,965,056 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                 │ (None, 5)              │           645 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 3,989,285 (15.22 MB)
```

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

```
Epoch 1/15
109/109 ━━━━━━━━━━━━━━━━━━━━ 286s 3s/step - accuracy: 0.3558 - loss: 1.4801 - val_accuracy: 0.5235 - val_loss: 1.1358
Epoch 2/15
109/109 ━━━━━━━━━━━━━━━━━━━━ 146s 1s/step - accuracy: 0.5650 - loss: 1.0594 - val_accuracy: 0.5993 - val_loss: 0.9821
Epoch 3/15
109/109 ━━━━━━━━━━━━━━━━━━━━ 145s 1s/step - accuracy: 0.6129 - loss: 0.9723 - val_accuracy: 0.6315 - val_loss: 0.8943
Epoch 4/15
109/109 ━━━━━━━━━━━━━━━━━━━━ 146s 1s/step - accuracy: 0.6572 - loss: 0.9093 - val_accuracy: 0.6728 - val_loss: 0.8393
Epoch 5/15
109/109 ━━━━━━━━━━━━━━━━━━━━ 202s 1s/step - accuracy: 0.6786 - loss: 0.8310 - val_accuracy: 0.6521 - val_loss: 0.9190
Epoch 6/15
109/109 ━━━━━━━━━━━━━━━━━━━━ 145s 1s/step - accuracy: 0.6912 - loss: 0.8168 - val_accuracy: 0.6877 - val_loss: 0.8248
Epoch 7/15
109/109 ━━━━━━━━━━━━━━━━━━━━ 202s 1s/step - accuracy: 0.7044 - loss: 0.7732 - val_accuracy: 0.6889 - val_loss: 0.8167
Epoch 8/15
109/109 ━━━━━━━━━━━━━━━━━━━━ 204s 1s/step - accuracy: 0.7210 - loss: 0.7537 - val_accuracy: 0.6946 - val_loss: 0.8203
Epoch 9/15
109/109 ━━━━━━━━━━━━━━━━━━━━ 146s 1s/step - accuracy: 0.7016 - loss: 0.7505 - val_accuracy: 0.7153 - val_loss: 0.7369
Epoch 10/15
109/109 ━━━━━━━━━━━━━━━━━━━━ 203s 1s/step - accuracy: 0.7420 - loss: 0.6790 - val_accuracy: 0.7072 - val_loss: 0.7636
Epoch 11/15
109/109 ━━━━━━━━━━━━━━━━━━━━ 200s 1s/step - accuracy: 0.7609 - loss: 0.6431 - val_accuracy: 0.6912 - val_loss: 0.8587
Epoch 12/15
109/109 ━━━━━━━━━━━━━━━━━━━━ 145s 1s/step - accuracy: 0.7606 - loss: 0.6490 - val_accuracy: 0.7118 - val_loss: 0.8121
Epoch 13/15
109/109 ━━━━━━━━━━━━━━━━━━━━ 203s 1s/step - accuracy: 0.7548 - loss: 0.6270 - val_accuracy: 0.7107 - val_loss: 0.8380
Epoch 14/15
109/109 ━━━━━━━━━━━━━━━━━━━━ 147s 1s/step - accuracy: 0.7566 - loss: 0.6449 - val_accuracy: 0.7279 - val_loss: 0.7201
Epoch 15/15
109/109 ━━━━━━━━━━━━━━━━━━━━ 147s 1s/step - accuracy: 0.7871 - loss: 0.5469 - val_accuracy: 0.6854 - val_loss: 0.8450
```

```
def classify_images(image_path):
    input_image = tf.keras.utils.load_img(image_path, target_size=(180,180))
    input_image_array = tf.keras.utils.img_to_array(input_image)
    input_image_exp_dim = tf.expand_dims(input_image_array,0)

    predictions = model.predict(input_image_exp_dim)
    result = tf.nn.softmax(predictions[0])
    outcome = 'The Image belongs to ' + flower_names[np.argmax(result)] + ' with a score of '+ str(np.max(result)*100)
    return outcome
```

```
classify_images('/content/drive/MyDrive/Flower Recognition/Sample/rose.jpg')
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
'The Image belongs to rose with a score of 85.67389'
```

```
model.save("/content/drive/MyDrive/Flower Recognition/flower_model.keras")
```

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

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