```
import io
import pandas as pd
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


#Load our data
#from google.colab import files
#uploaded = files.upload()




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

```
Downloading data from https://storage.googleapis.com/download.tensorflow.org/example
228813984/228813984 [==============================] - 6s 0us/step
```

```
count = len(list(data.glob("*/*.jpg")))
print(count)
# we have 3670 total images
```

```
3670
```

```
#Lets sort tulips:
tulips = list(data.glob("tulips/*"))
PIL.Image.open(str(tulips[0]))
PIL.Image.open(str(tulips[1]))


#Divide into train and test:
train = tf.keras.utils.image_dataset_from_directory(data, validation_split=0.2, subset="tr

validation = tf.keras.utils.image_dataset_from_directory(data, validation_split=0.2, subse

#Get our class names
class_name = train.class_names
print(class_name)
```

```
Found 3670 files belonging to 5 classes.
Using 2936 files for training.
Found 3670 files belonging to 5 classes.
Using 734 files for validation.
['daisy', 'dandelion', 'roses', 'sunflowers', 'tulips']
```

```
from tensorflow.python.ops.gen_data_flow_ops import barrier_eager_fallback
#Visual our data:
for image_batch, labels_batch in train:
  print(image_batch.shape)
  print(labels_batch.shape)
  break

  #This is the tensor shape
```

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

```
#Configure our dataset (performance)
Tune = tf.data.AUTOTUNE
train = train.cache().shuffle(1000).prefetch(buffer_size=Tune)
validation = validation.cache().prefetch(buffer_size=Tune)
```

```
#standardizing the data:
normalize_layer = layers.Rescaling(1./255)
#We are normalizing the data, because the RGB values are from 0 to 255. This will make the
normalize = train.map(lambda x, y: (normalize_layer(x), y))
image_batch, labels_batch = next(iter(normalize))
first_im = image_batch[0]

print(np.min(first_im), np.max(first_im))
#Our values are now from 0 to 1
```

```
0.0 1.0
```

```
#2. Create the sequential model (Note the model is not tuned for a high accuracy)
num_classes = len(class_name)

model = Sequential([layers.Rescaling(1./255, input_shape=(180, 180, 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)])
```

```
#Compile
model.compile(optimizer="adam", loss=tf.keras.losses.SparseCategoricalCrossentropy(from_lo

#model summary
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_6 (Rescaling)     (None, 180, 180, 3)       0

 conv2d_3 (Conv2D)           (None, 180, 180, 16)      448

 max_pooling2d_3 (MaxPooling  (None, 90, 90, 16)       0
 2D)

 conv2d_4 (Conv2D)           (None, 90, 90, 32)        4640

 max_pooling2d_4 (MaxPooling  (None, 45, 45, 32)       0
 2D)

 conv2d_5 (Conv2D)           (None, 45, 45, 64)        18496

 max_pooling2d_5 (MaxPooling  (None, 22, 22, 64)       0
 2D)

 flatten_1 (Flatten)         (None, 30976)             0

 dense_2 (Dense)             (None, 128)               3965056

 dense_3 (Dense)             (None, 5)                 645

=================================================================
Total params: 3,989,285
Trainable params: 3,989,285
Non-trainable params: 0
_________________________________________________________________
```

```
#Training our model
epochs=10
history=model.fit(train, validation_data=validation, epochs=epochs)
```

```
Epoch 1/10
92/92 [==============================] - 74s 800ms/step - loss: 1.3904 - accuracy: 0
Epoch 2/10
92/92 [==============================] - 82s 888ms/step - loss: 0.9991 - accuracy: 0
Epoch 3/10
92/92 [==============================] - 97s 1s/step - loss: 0.8370 - accuracy: 0.66
Epoch 4/10
92/92 [==============================] - 79s 858ms/step - loss: 0.6498 - accuracy: 0
Epoch 5/10
```

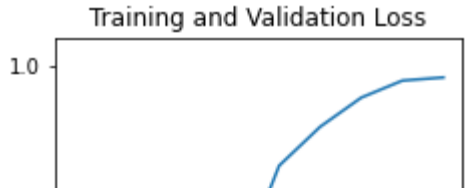
```
92/92 [==============================] - 73s 792ms/step - loss: 0.4368 - accuracy: 0
Epoch 6/10
92/92 [==============================] - 70s 761ms/step - loss: 0.2439 - accuracy: 0
Epoch 7/10
92/92 [==============================] - 71s 776ms/step - loss: 0.1583 - accuracy: 0
Epoch 8/10
92/92 [==============================] - 70s 763ms/step - loss: 0.0847 - accuracy: 0
Epoch 9/10
92/92 [==============================] - 71s 769ms/step - loss: 0.0468 - accuracy: 0
Epoch 10/10
92/92 [==============================] - 71s 767ms/step - loss: 0.0362 - accuracy: 0
```

```
#Visualized training results:
accuracy = history.history["accuracy"]
val_accuracy = history.history["val_accuracy"]

loss = history.history["loss"]
val_loss = history.history["val_loss"]

range_epochs = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(range_epochs, accuracy, label="Training Accuracy")
plt.plot(range_epochs, val_accuracy, label="Validation Accuracy")
plt.legend(loc="lower right")
plt.title("Training and Validation Loss")
plt.show()
```



Our training accuracy will increase over time, as shown in the plots. Overfitting will occur, and this will cause the model to have difficulty generalizing a new dataset.