Your First AI application

Going forward, AI algorithms will be incorporated into more and more everyday applications. For example, you might want to include an image classifier in a smart phone app. To do this, you'd use a deep learning model trained on hundreds of thousands of images as part of the overall application architecture. A large part of software development in the future will be using these types of models as common parts of applications.

In this project, you'll train an image classifier to recognize different species of flowers. You can imagine using something like this in a phone app that tells you the name of the flower your camera is looking at. In practice you'd train this classifier, then export it for use in your application. We'll be using this dataset (http://www.robots.ox.ac.uk/~vgg/data/flowers/102/index.html) from Oxford of 102 flower categories, you can see a few examples below.

The project is broken down into multiple steps:

- Load the image dataset and create a pipeline.
- Build and Train an image classifier on this dataset.
- Use your trained model to perform inference on flower images.

We'll lead you through each part which you'll implement in Python.

When you've completed this project, you'll have an application that can be trained on any set of labeled images. Here your network will be learning about flowers and end up as a command line application. But, what you do with your new skills depends on your imagination and effort in building a dataset. For example, imagine an app where you take a picture of a car, it tells you what the make and model is, then looks up information about it. Go build your own dataset and make something new.

Import Resources

Avoiding the error of Data Loading

In [ ]:
```
#The new version of dataset is only available in the tfds-nightly pac
kage.
%pip --no-cache-dir install tfds-nightly --user
!pip install tensorflow --upgrade --user
```

```
Collecting tfds-nightly
  Downloading https://files.pythonhosted.org/packages/38/7d/eb2dd9201
676baa3c2e19341197fb32868d32defdae9c3b13790758f27f0/tfds_nightly-4.2.
0.dev202103040106-py3-none-any.whl (3.8MB)
     |████████████████████████████████| 3.8MB 5.0MB/s eta 0:00:01
|█████████████████               | 2.0MB 5.0MB/s eta 0:00:01
Requirement already satisfied: tensorflow-metadata in /opt/conda/lib/
python3.7/site-packages (from tfds-nightly) (0.14.0)
Requirement already satisfied: attrs>=18.1.0 in /opt/conda/lib/python
3.7/site-packages (from tfds-nightly) (19.3.0)
Collecting typing-extensions; python_version < "3.8"
  Downloading https://files.pythonhosted.org/packages/60/7a/e881b5abb
54db0e6e671ab088d079c57ce54e8a01a3ca443f561ccadb37e/typing_extensions
-3.7.4.3-py3-none-any.whl
Requirement already satisfied: tqdm in /opt/conda/lib/python3.7/site-
packages (from tfds-nightly) (4.36.1)
Requirement already satisfied: dill in /opt/conda/lib/python3.7/site-
packages (from tfds-nightly) (0.3.1.1)
Collecting protobuf>=3.12.2
  Downloading https://files.pythonhosted.org/packages/5c/5f/4115a1fae
5245885dcc8337b2672697bdb28fdb5370706ad6f4961368e34/protobuf-3.15.4-c
p37-cp37m-manylinux1_x86_64.whl (1.0MB)
     |████████████████████████████████| 1.0MB 11.3MB/s eta 0:00:01
Requirement already satisfied: termcolor in /opt/conda/lib/python3.7/
site-packages (from tfds-nightly) (1.1.0)
Requirement already satisfied: promise in /opt/conda/lib/python3.7/si
te-packages (from tfds-nightly) (2.2.1)
Requirement already satisfied: numpy in /opt/conda/lib/python3.7/site
-packages (from tfds-nightly) (1.17.4)
Requirement already satisfied: six in /opt/conda/lib/python3.7/site-p
ackages (from tfds-nightly) (1.12.0)
Requirement already satisfied: requests>=2.19.0 in /opt/conda/lib/pyt
hon3.7/site-packages (from tfds-nightly) (2.22.0)
Requirement already satisfied: absl-py in /opt/conda/lib/python3.7/si
te-packages (from tfds-nightly) (0.8.1)
Collecting importlib-resources; python_version < "3.9"
  Downloading https://files.pythonhosted.org/packages/f5/6e/a5c7a7147
407a318cb421d10d84bb2049e81d0b7472eb0a91a30b9ea24a6/importlib_resourc
es-5.1.1-py3-none-any.whl
Requirement already satisfied: future in /opt/conda/lib/python3.7/sit
e-packages (from tfds-nightly) (0.18.2)
Requirement already satisfied: googleapis-common-protos in /opt/cond
a/lib/python3.7/site-packages (from tensorflow-metadata->tfds-nightl
y) (1.6.0)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.
1 in /opt/conda/lib/python3.7/site-packages (from requests>=2.19.0->t
fds-nightly) (1.24.2)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /opt/conda/li
b/python3.7/site-packages (from requests>=2.19.0->tfds-nightly) (3.0.
4)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/p
ython3.7/site-packages (from requests>=2.19.0->tfds-nightly) (2019.1
1.28)
Requirement already satisfied: idna<2.9,>=2.5 in /opt/conda/lib/pytho
n3.7/site-packages (from requests>=2.19.0->tfds-nightly) (2.8)
Requirement already satisfied: zipp>=0.4; python_version < "3.8" in /
opt/conda/lib/python3.7/site-packages (from importlib-resources; pyth
```

```
on_version < "3.9"->tfds-nightly) (0.6.0)
Requirement already satisfied: more-itertools in /opt/conda/lib/pytho
n3.7/site-packages (from zipp>=0.4; python_version < "3.8"->importlib
-resources; python_version < "3.9"->tfds-nightly) (8.0.2)
Installing collected packages: typing-extensions, protobuf, importlib
-resources, tfds-nightly
  WARNING: The script tfds is installed in '/root/.local/bin' which i
s not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppres
s this warning, use --no-warn-script-location.
Successfully installed importlib-resources-5.1.1 protobuf-3.15.4 tfds
-nightly-4.2.0.dev202103040106 typing-extensions-3.7.4.3
Note: you may need to restart the kernel to use updated packages.
Collecting tensorflow
  Downloading https://files.pythonhosted.org/packages/70/dc/e8c5e7983
866fa4ef3fd619faa35f660b95b01a2ab62b3884f038ccab542/tensorflow-2.4.1-
cp37-cp37m-manylinux2010_x86_64.whl (394.3MB)
     |████████████                    | 143.2MB 8.7MB/s eta 0:00:300
|▏                               | 2.3MB 3.9MB/s eta 0:01:42     |▍
| 3.5MB 3.9MB/s eta 0:01:41     |▌                               | 4.
6MB 3.9MB/s eta 0:01:41     |▌                               | 6.2MB
 3.9MB/s eta 0:01:41     |▊                               | 9.3MB 3.9
MB/s eta 0:01:40     |█                               | 11.1MB 14.1M
B/s eta 0:00:28     |█                               | 11.7MB 14.1MB/
s eta 0:00:28     |█                               | 12.9MB 14.1MB/s
 eta 0:00:28     |█▎                              | 15.4MB 14.1MB/s e
ta 0:00:27     |█▎                              | 16.0MB 14.1MB/s eta
0:00:27     |█▌                              | 18.5MB 14.1MB/s eta 0:
00:27     |█▊                              | 21.0MB 14.1MB/s eta 0:0
0:27     |█▊                              | 21.7MB 14.1MB/s eta 0:00:
27     |██                              | 22.9MB 13.0MB/s eta 0:00:29
|██                              | 24.7MB 13.0MB/s eta 0:00:29     |▉
█                              | 25.9MB 13.0MB/s eta 0:00:29     |██▍
| 29.7MB 13.0MB/s eta 0:00:29     |██▌                             |
 30.3MB 13.0MB/s eta 0:00:29     |██▊                             | 3
4.0MB 13.0MB/s eta 0:00:28     |███                             | 35.
2MB 13.0MB/s eta 0:00:28     |███                             | 35.8M
B 13.0MB/s eta 0:00:28     |███                             | 37.1MB
 13.0MB/s eta 0:00:28     |███                             | 38.3MB 1
3.0MB/s eta 0:00:28     |███▏                            | 39.5MB 13.
0MB/s eta 0:00:28     |███▍                            | 42.0MB 13.0M
B/s eta 0:00:28     |███▋                            | 44.4MB 11.2MB/
s eta 0:00:32     |███▋                            | 45.0MB 11.2MB/s
 eta 0:00:32     |███▊                            | 45.6MB 11.2MB/s e
ta 0:00:32     |████                            | 49.6MB 11.2MB/s eta
0:00:31     |████▏                           | 51.5MB 11.2MB/s eta 0:
00:31     |████▎                           | 52.1MB 11.2MB/s eta 0:0
0:31     |████▎                           | 52.7MB 11.2MB/s eta 0:00:
31     |████▋                           | 56.2MB 9.6MB/s eta 0:00:36
|████▋                           | 56.8MB 9.6MB/s eta 0:00:36     |██
██▊                           | 57.9MB 9.6MB/s eta 0:00:36     |█████
| 61.0MB 9.6MB/s eta 0:00:35     |█████                           | 6
1.6MB 9.6MB/s eta 0:00:35     |█████                           | 62.2
MB 9.6MB/s eta 0:00:35     |█████▏                          | 64.1MB
 9.6MB/s eta 0:00:35     |█████▎                          | 64.7MB 9.
6MB/s eta 0:00:35     |█████▍                          | 66.4MB 9.6M
B/s eta 0:00:35     |█████▌                          | 67.6MB 4.8MB/s
```

```
eta 0:01:09     |████                          | 69.3MB 4.8MB/s eta
0:01:09     |████                          | 69.9MB 4.8MB/s eta 0:0
1:09     |████                          | 70.5MB 4.8MB/s eta 0:01:0
8     |████                          | 71.7MB 4.8MB/s eta 0:01:08
|████                          | 72.4MB 4.8MB/s eta 0:01:08     |█
███                          | 73.6MB 4.8MB/s eta 0:01:08     |███
█                          | 74.2MB 4.8MB/s eta 0:01:08     |████
| 74.8MB 4.8MB/s eta 0:01:08     |████                          | 7
5.4MB 4.8MB/s eta 0:01:07     |████                          | 77.8
MB 6.9MB/s eta 0:00:46     |████                          | 79.6MB
 6.9MB/s eta 0:00:46     |████                          | 80.2MB 6.
9MB/s eta 0:00:46     |████                          | 80.8MB 6.9M
B/s eta 0:00:46     |████                          | 81.4MB 6.9MB/s
eta 0:00:46     |████                          | 82.0MB 6.9MB/s eta
0:00:45     |████                          | 82.6MB 6.9MB/s eta 0:0
0:45     |████                          | 83.2MB 6.9MB/s eta 0:00:4
5     |████                          | 83.8MB 6.9MB/s eta 0:00:45
|████                          | 84.4MB 6.9MB/s eta 0:00:45     |█
███                          | 85.0MB 6.9MB/s eta 0:00:45     |███
█                          | 87.4MB 6.9MB/s eta 0:00:45     |████
| 87.9MB 12.4MB/s eta 0:00:25     |████                          |
 88.5MB 12.4MB/s eta 0:00:25     |████                          | 8
9.2MB 12.4MB/s eta 0:00:25     |████                          | 89.
8MB 12.4MB/s eta 0:00:25     |████                          | 90.3M
B 12.4MB/s eta 0:00:25     |████                          | 91.5MB
 12.4MB/s eta 0:00:25     |████                          | 92.7MB 1
2.4MB/s eta 0:00:25     |████                          | 94.6MB 12.
4MB/s eta 0:00:25     |████                          | 95.2MB 12.4M
B/s eta 0:00:25     |████                          | 95.9MB 12.4MB/
s eta 0:00:25     |████                          | 96.5MB 12.4MB/s
 eta 0:00:25     |████                          | 97.1MB 12.4MB/s e
ta 0:00:24     |████                          | 97.8MB 12.4MB/s eta
0:00:24     |████                          | 98.4MB 12.4MB/s eta 0:
00:24     |████                          | 100.2MB 12.7MB/s eta 0:0
0:24     |████                          | 101.3MB 12.7MB/s eta 0:0
0:24     |████                          | 103.0MB 12.7MB/s eta 0:0
0:24     |████                          | 103.6MB 12.7MB/s eta 0:0
0:23     |████                          | 104.8MB 12.7MB/s eta 0:0
0:23     |████                          | 107.0MB 12.7MB/s eta 0:0
0:23     |████                          | 109.3MB 6.4MB/s eta 0:00:
45     |████                          | 110.2MB 6.4MB/s eta 0:00:45
|████                          | 111.2MB 6.4MB/s eta 0:00:45     |█
████                          | 111.8MB 6.4MB/s eta 0:00:44     |██
███                          | 112.3MB 6.4MB/s eta 0:00:44     |███
██                          | 112.7MB 6.4MB/s eta 0:00:44     |████
█                          | 113.7MB 6.4MB/s eta 0:00:44     |█████
|                          | 114.3MB 6.4MB/s eta 0:00:44     |█████
| 114.9MB 6.4MB/s eta 0:00:44     |█████                          |
 115.6MB 6.4MB/s eta 0:00:44     |█████                          | 1
16.2MB 6.4MB/s eta 0:00:44     |█████                          | 11
6.9MB 6.4MB/s eta 0:00:44     |█████                          | 118.
1MB 6.4MB/s eta 0:00:44     |█████                          | 118.7M
B 15.1MB/s eta 0:00:19     |█████                          | 119.3MB
15.1MB/s eta 0:00:19     |█████                          | 122.3MB 1
5.1MB/s eta 0:00:19     |█████                          | 122.9MB 1
5.1MB/s eta 0:00:19     |█████                          | 124.6MB 1
5.1MB/s eta 0:00:18     |█████                          | 125.2MB 1
```

```
5.1MB/s eta 0:00:18     |          | 125.7MB 1
5.1MB/s eta 0:00:18     |          | 126.3MB 1
5.1MB/s eta 0:00:18     |          | 126.9MB 1
5.1MB/s eta 0:00:18     |          | 128.7MB 1
5.1MB/s eta 0:00:18     |          | 130.0MB 1
2.9MB/s eta 0:00:21     |          | 130.5MB 1
2.9MB/s eta 0:00:21     |          | 131.1MB 1
2.9MB/s eta 0:00:21     |          | 131.7MB 1
2.9MB/s eta 0:00:21     |          | 132.8MB 1
2.9MB/s eta 0:00:21     |          | 133.9MB 1
2.9MB/s eta 0:00:21     |          | 134.5MB 1
2.9MB/s eta 0:00:21     |          | 135.1MB 1
2.9MB/s eta 0:00:21     |          | 136.2MB 1
2.9MB/s eta 0:00:20     |          | 136.8MB 1
2.9MB/s eta 0:00:20     |          | 137.4MB 1
2.9MB/s eta 0:00:20     |          | 139.1MB 1
2.9MB/s eta 0:00:20     |          | 140.3MB 8.
7MB/s eta 0:00:30     |          | 140.8MB 8.7M
B/s eta 0:00:30     |          | 141.4MB 8.7MB/
s eta 0:00:30     |          | 142.6MB 8.7MB/s
 eta 0:00:30     |          | 280.6MB 20.1MB/s
 eta 0:00:06     |          | 145.6MB 8.7MB/s et
a 0:00:29     |          | 146.3MB 8.7MB/s eta
 0:00:29     |          | 146.8MB 8.7MB/s eta
 0:00:29     |          | 147.4MB 8.7MB/s eta
 0:00:29     |          | 148.1MB 8.7MB/s eta
 0:00:29     |          | 150.4MB 8.7MB/s eta
 0:00:29     |          | 152.7MB 6.5MB/s eta
 0:00:37     |          | 153.3MB 6.5MB/s eta
 0:00:37     |          | 153.8MB 6.5MB/s eta
 0:00:37     |          | 154.9MB 6.5MB/s eta
 0:00:37     |          | 156.0MB 6.5MB/s eta
 0:00:37     |          | 157.7MB 6.5MB/s eta
 0:00:37     |          | 158.3MB 6.5MB/s eta
 0:00:37     |          | 158.9MB 6.5MB/s eta
 0:00:36     |          | 159.4MB 6.5MB/s eta
 0:00:36     |          | 160.5MB 6.5MB/s eta
 0:00:36     |          | 161.7MB 6.0MB/s eta
 0:00:40     |          | 162.3MB 6.0MB/s eta
 0:00:39     |          | 162.9MB 6.0MB/s eta
 0:00:39     |          | 163.5MB 6.0MB/s eta
 0:00:39     |          | 169.9MB 6.0MB/s eta
 0:00:38     |          | 170.3MB 6.0MB/s eta
 0:00:38     |          | 171.4MB 5.3MB/s eta
 0:00:43     |          | 172.9MB 5.3MB/s eta
 0:00:42     |          | 173.4MB 5.3MB/s eta
 0:00:42     |          | 174.3MB 5.3MB/s eta
 0:00:42     |          | 175.3MB 5.3MB/s eta
 0:00:42     |          | 175.9MB 5.3MB/s eta
 0:00:42     |          | 176.4MB 5.3MB/s eta
 0:00:42     |          | 178.0MB 5.3MB/s eta
 0:00:41     |          | 178.6MB 5.3MB/s eta
 0:00:41     |          | 179.2MB 5.3MB/s eta
 0:00:41     |          | 180.3MB 9.6MB/s eta
 0:00:23     |          | 182.6MB 9.6MB/s eta
 0:00:22     |          | 183.7MB 9.6MB/s eta
 0:00:22     |          | 186.0MB 9.6MB/s eta
```

```
0:00:22 |  | 187.2MB 9.6MB/s eta
0:00:22 |  | 191.4MB 11.7MB/s eta
0:00:18 |  | 192.9MB 11.7MB/s eta
0:00:18 |  | 193.5MB 11.7MB/s eta
0:00:18 |  | 195.1MB 11.7MB/s eta
0:00:18 |  | 198.9MB 11.7MB/s eta
0:00:17 |  | 199.4MB 8.5MB/s eta
0:00:23 |  | 201.9MB 8.5MB/s eta
0:00:23 |  | 208.2MB 8.5MB/s eta
0:00:22 |  | 209.4MB 8.5MB/s eta
0:00:22 |  | 211.0MB 4.9MB/s eta
0:00:38 |  | 212.3MB 4.9MB/s eta
0:00:38 |  | 212.8MB 4.9MB/s eta
0:00:37 |  | 213.9MB 4.9MB/s eta
0:00:37 |  | 214.5MB 4.9MB/s eta
0:00:37 |  | 216.2MB 4.9MB/s eta
0:00:37 |  | 217.3MB 4.9MB/s eta
0:00:36 |  | 220.7MB 12.4MB/s eta
0:00:14 |  | 221.1MB 12.4MB/s eta
0:00:14 |  | 221.6MB 12.4MB/s eta
0:00:14 |  | 222.5MB 12.4MB/s eta
0:00:14 |  | 223.4MB 12.4MB/s eta
0:00:14 |  | 224.6MB 12.4MB/s eta
0:00:14 |  | 226.0MB 12.4MB/s eta
0:00:14 |  | 226.6MB 12.4MB/s eta
0:00:14 |  | 227.1MB 12.4MB/s eta
0:00:14 |  | 228.2MB 12.4MB/s eta
0:00:14 |  | 228.5MB 12.4MB/s eta
0:00:14 |  | 228.9MB 12.4MB/s eta
0:00:14 |  | 229.8MB 12.9MB/s eta
0:00:13 |  | 230.4MB 12.9MB/s eta
0:00:13 |  | 232.4MB 12.9MB/s eta
0:00:13 |  | 233.5MB 12.9MB/s eta
0:00:13 |  | 234.1MB 12.9MB/s eta
0:00:13 |  | 234.6MB 12.9MB/s eta
0:00:13 |  | 235.2MB 12.9MB/s eta
0:00:13 |  | 235.7MB 12.9MB/s eta
0:00:13 |  | 236.1MB 12.9MB/s eta
0:00:13 |  | 237.2MB 12.9MB/s eta
0:00:13 |  | 238.2MB 12.9MB/s eta
0:00:13 |  | 241.4MB 14.6MB/s eta
0:00:11 |  | 243.0MB 14.6MB/s eta
0:00:11 |  | 244.1MB 14.6MB/s eta
0:00:11 |  | 245.3MB 14.6MB/s eta
0:00:11 |  | 245.8MB 14.6MB/s eta
0:00:11 |  | 248.9MB 14.6MB/s eta
0:00:10 |  | 249.8MB 22.0MB/s eta
0:00:07 |  | 250.2MB 22.0MB/s eta
0:00:07 |  | 250.7MB 22.0MB/s eta
0:00:07 |  | 251.2MB 22.0MB/s eta
0:00:07 |  | 252.2MB 22.0MB/s eta
0:00:07 |  | 252.7MB 22.0MB/s eta
0:00:07 |  | 253.2MB 22.0MB/s eta
0:00:07 |  | 254.2MB 22.0MB/s eta
0:00:07 |  | 254.7MB 22.0MB/s eta
0:00:07 |  | 255.8MB 22.0MB/s eta
0:00:07 |  | 256.4MB 22.0MB/s eta
```

```
0:00:07 || 257.0MB 22.0MB/s eta
0:00:07 || 257.5MB 22.0MB/s eta
0:00:07 || 258.5MB 22.0MB/s eta
0:00:07 || 259.0MB 12.6MB/s eta
0:00:11 || 259.6MB 12.6MB/s eta
0:00:11 || 261.5MB 12.6MB/s eta
0:00:11 || 262.0MB 12.6MB/s eta
0:00:11 || 262.5MB 12.6MB/s eta
0:00:11 || 263.0MB 12.6MB/s eta
0:00:11 || 264.8MB 12.6MB/s eta
0:00:11 || 265.8MB 12.6MB/s eta
0:00:11 || 267.3MB 12.6MB/s eta
0:00:11 || 267.7MB 12.6MB/s eta
0:00:11 || 269.2MB 14.5MB/s eta
0:00:09 || 270.7MB 14.5MB/s eta
0:00:09 || 271.2MB 14.5MB/s eta
0:00:09 || 271.7MB 14.5MB/s eta
0:00:09 || 272.3MB 14.5MB/s eta
0:00:09 || 273.4MB 14.5MB/s eta
0:00:09 || 273.9MB 14.5MB/s eta
0:00:09 || 274.4MB 14.5MB/s eta
0:00:09 || 275.8MB 14.5MB/s eta
0:00:09 || 276.3MB 14.5MB/s eta
0:00:09 || 276.9MB 14.5MB/s eta
0:00:09 || 278.0MB 20.1MB/s eta
0:00:06 || 278.5MB 20.1MB/s eta
0:00:06 || 279.5MB 20.1MB/s eta
0:00:06 || 280.1MB 20.1MB/s eta
0:00:06 || 356.6MB 4.7MB/s eta
0:00:095 || 282.1MB 20.1MB/s eta
0:00:06 || 283.6MB 20.1MB/s eta
0:00:06 || 284.0MB 20.1MB/s eta
0:00:06 || 284.3MB 20.1MB/s eta
0:00:06 || 284.7MB 20.1MB/s eta
0:00:06 || 285.2MB 20.1MB/s eta
0:00:06 || 285.6MB 20.1MB/s eta
0:00:06 || 286.1MB 6.9MB/s eta
0:00:16 || 287.1MB 6.9MB/s eta
0:00:16 || 287.6MB 6.9MB/s eta
0:00:16 || 288.1MB 6.9MB/s eta
0:00:16 || 288.4MB 6.9MB/s eta
0:00:16 || 289.9MB 6.9MB/s eta
0:00:16 || 290.3MB 6.9MB/s eta
0:00:16 || 291.3MB 6.9MB/s eta
0:00:15 || 291.8MB 6.9MB/s eta
0:00:15 || 292.8MB 6.9MB/s eta
0:00:15 || 293.9MB 6.9MB/s eta
0:00:15 || 294.5MB 6.9MB/s eta
0:00:15 || 295.0MB 30.9MB/s eta
0:00:04 || 297.8MB 30.9MB/s eta
0:00:04 || 301.6MB 30.9MB/s eta
0:00:03 || 302.2MB 30.9MB/s eta
0:00:03 || 302.7MB 30.9MB/s eta
0:00:03 || 303.2MB 30.9MB/s eta
0:00:03 || 304.8MB 30.9MB/s eta
0:00:03 || 305.8MB 3.4MB/s eta
0:00:26 || 306.3MB 3.4MB/s eta
```

```
0:00:26 |  | 306.8MB 3.4MB/s eta
0:00:26 |  | 307.8MB 3.4MB/s eta
0:00:26 |  | 308.6MB 3.4MB/s eta
0:00:26 |  | 309.1MB 3.4MB/s eta
0:00:25 |  | 310.4MB 3.4MB/s eta
0:00:25 |  | 311.5MB 3.4MB/s eta
0:00:25 |  | 312.8MB 3.4MB/s eta
0:00:24 |  | 313.3MB 7.7MB/s eta
0:00:11 |  | 314.3MB 7.7MB/s eta
0:00:11 |  | 314.8MB 7.7MB/s eta
0:00:11 |  | 315.8MB 7.7MB/s eta
0:00:11 |  | 317.1MB 7.7MB/s eta
0:00:10 |  | 320.4MB 7.7MB/s eta
0:00:10 |  | 320.9MB 7.7MB/s eta
0:00:10 |  | 321.5MB 7.7MB/s eta
0:00:10 |  | 322.0MB 7.7MB/s eta
0:00:10 |  | 322.5MB 13.9MB/s eta
0:00:06 |  | 323.5MB 13.9MB/s eta
0:00:06 |  | 324.6MB 13.9MB/s eta
0:00:06 |  | 325.7MB 13.9MB/s eta
0:00:05 |  | 326.2MB 13.9MB/s eta
0:00:05 |  | 326.8MB 13.9MB/s eta
0:00:05 |  | 327.4MB 13.9MB/s eta
0:00:05 |  | 328.5MB 13.9MB/s eta
0:00:05 |  | 331.2MB 13.9MB/s eta
0:00:05 |  | 331.7MB 13.9MB/s eta
0:00:05 |  | 332.7MB 3.6MB/s eta
0:00:18 |  | 335.9MB 3.6MB/s eta
0:00:17 |  | 336.4MB 3.6MB/s eta
0:00:17 |  | 337.0MB 3.6MB/s eta
0:00:17 |  | 337.6MB 3.6MB/s eta
0:00:16 |  | 337.9MB 3.6MB/s eta
0:00:16 |  | 341.5MB 4.0MB/s eta
0:00:14 |  | 342.1MB 4.0MB/s eta
0:00:14 |  | 343.1MB 4.0MB/s eta
0:00:13 |  | 344.1MB 4.0MB/s eta
0:00:13 |  | 345.1MB 4.0MB/s eta
0:00:13 |  | 346.3MB 4.0MB/s eta
0:00:13 |  | 347.4MB 4.0MB/s eta
0:00:12 |  | 348.5MB 4.0MB/s eta
0:00:12 |  | 349.1MB 4.0MB/s eta
0:00:12 |  | 350.2MB 4.0MB/s eta
0:00:12 |  | 350.8MB 4.0MB/s eta
0:00:11 |  | 353.5MB 4.7MB/s eta
0:00:09 |  | 354.0MB 4.7MB/s eta
0:00:09 |  | 355.0MB 4.7MB/s eta
0:00:09 |  | 356.1MB 4.7MB/s eta
0:00:09
```

In [ ]:

In [ ]:

In [ ]:

In [1]:
```
#%pip --no-cache-dir install tfds-nightly --user
#!pip install tensorflow --upgrade --use
```

In [2]:
```
#!pip install grpcio

#!pip install tensorflow --upgrade --user
```

In [2]:
```
#!python -m tensorflow_datasets.scripts.download_and_prepare --regist
er_checksums=True --datasets=oxford_flowers102
```

In [9]:
```
#!python -m tensorflow_datasets.scripts.download_and_prepare --regist
er_checksums=True --datasets=oxford_flowers102 --user
```

In [24]:
```
# The new version of dataset is only available in the tfds-nightly pa
ckage.
#%pip --no-cache-dir install tensorflow-datasets --user
# DON'T MISS TO RESTART THE KERNEL
```

In [1]:
```
# Import TensorFlow
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
%config InlineBackend.figure_format = 'retina'
import tensorflow as tf
import tensorflow_datasets as tfds
import tensorflow_hub as hub
```

In [2]:
```
# TODO: Make all other necessary imports.

import matplotlib.pyplot as plt
import numpy as np
import time as time
import json

import PIL
from PIL import Image
```

In [3]:
```
#!pip install -q -U tensorflow_datasets
#!pip install tfds-nightly;
```

Testing whether GPU is available!

In [4]:
```
print('Using:')
print('\t\u2022 TensorFlow version:', tf.__version__)
print('\t\u2022 tf.keras version:', tf.keras.__version__)
print('\t\u2022 Running on GPU' if tf.test.is_gpu_available() else '
\t\u2022 GPU device not found. Running on CPU')
```

```
Using:
	• TensorFlow version: 2.4.1
	• tf.keras version: 2.4.0
WARNING:tensorflow:From <ipython-input-4-9b322f91229a>:4: is_gpu_avai
lable (from tensorflow.python.framework.test_util) is deprecated and
will be removed in a future version.
Instructions for updating:
Use `tf.config.list_physical_devices('GPU')` instead.
	• GPU device not found. Running on CPU
```

Load the Dataset

Here you'll use `tensorflow_datasets` to load the [Oxford Flowers 102 dataset (https://www.tensorflow.org/datasets/catalog/oxford_flowers102)](https://www.tensorflow.org/datasets/catalog/oxford_flowers102). This dataset has 3 splits: `'train'` , `'test'` , and `'validation'` . You'll also need to make sure the training data is normalized and resized to 224x224 pixels as required by the pre-trained networks.

The validation and testing sets are used to measure the model's performance on data it hasn't seen yet, but you'll still need to normalize and resize the images to the appropriate size.

In [5]:
```
# Download data to default local directory "~/tensorflow_datasets"


#!python -m tensorflow_datasets.scripts.download_and_prepare --regist
er_checksums=True --datasets=oxford_flowers102

# TODO: Load the dataset with TensorFlow Datasets. Hint: use tfds.loa
d()


train_split = 50
test_val_split = 25

#splits = tfds.Split.ALL.subsplit([50,25, 25])

#dataset, dataset_info = tfds.load('oxford_flowers102', split=splits,
as_supervised=True, with_info=True)
dataset, dataset_info = tfds.load('oxford_flowers102', as_supervised=
True, with_info=True)

# TODO: Create a training set, a validation set and a test set.
```

```
Downloading and preparing dataset 328.90 MiB (download: 328.90 MiB, g
enerated: 331.34 MiB, total: 660.25 MiB) to /root/tensorflow_dataset
s/oxford_flowers102/2.1.1...
```

```
Dataset oxford_flowers102 downloaded and prepared to /root/tensorflow
_datasets/oxford_flowers102/2.1.1. Subsequent calls will reuse this d
ata.
```

In [6]:
```
dataset
```

Out[6]:
```
{Split('train'): <PrefetchDataset shapes: ((None, None, 3), ()), type
s: (tf.uint8, tf.int64)>,
 Split('test'): <PrefetchDataset shapes: ((None, None, 3), ()), type
s: (tf.uint8, tf.int64)>,
 Split('validation'): <PrefetchDataset shapes: ((None, None, 3), ()),
types: (tf.uint8, tf.int64)>}
```

Explore the Dataset

In [7]:

```
# TODO: Get the number of examples in each set from the dataset info.
print("The total number of examples in the train set is: {0}".format(
dataset_info.splits['train'].num_examples))
print("The total number of examples in the validation set is: {0}".fo
rmat(dataset_info.splits['validation'].num_examples))
print("The total number of examples in the test set is: {0}".format(d
ataset_info.splits['test'].num_examples))

# TODO: Get the number of classes in the dataset from the dataset inf
o.
print("The number of classes is: {}".format(dataset_info.features['la
bel'].num_classes))
```

```
The total number of examples in the train set is: 1020
The total number of examples in the validation set is: 1020
The total number of examples in the test set is: 6149
The number of classes is: 102
```

In [8]:

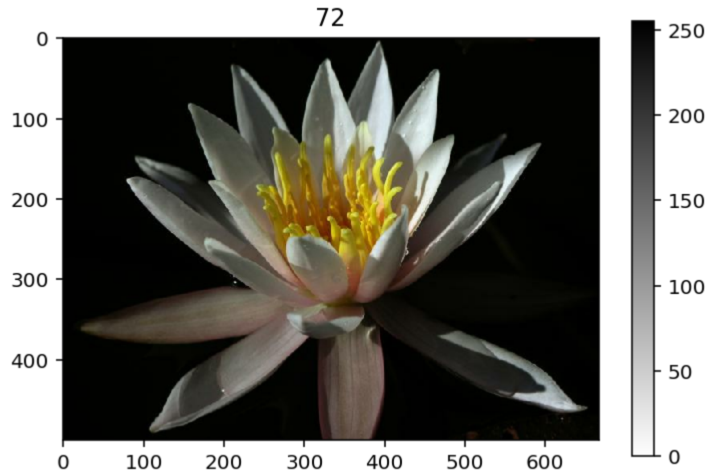
```
# TODO: Print the shape and corresponding label of 3 images in the tr
aining set.
#dataset['train']
for image, label in dataset['train'].take(3):
    print('The images in the training set have:\n\u2022 dtype:', imag
e.dtype, '\n\u2022 shape:', image.shape)
    print("label:",label.numpy())
```

```
The images in the training set have:
• dtype: <dtype: 'uint8'>
• shape: (500, 667, 3)
label: 72
The images in the training set have:
• dtype: <dtype: 'uint8'>
• shape: (500, 666, 3)
label: 84
The images in the training set have:
• dtype: <dtype: 'uint8'>
• shape: (670, 500, 3)
label: 70
```

In [9]:
```python
# TODO: Plot 1 image from the training set.

# Set the title of the plot to the corresponding image label.

for image, label in dataset['train'].take(1):
    image = image.numpy().squeeze()
    label = label.numpy()
plt.title(label)
plt.imshow(image, cmap= plt.cm.binary)
plt.colorbar()
plt.show()
```



Label Mapping

You'll also need to load in a mapping from label to category name. You can find this in the file `label_map.json`. It's a JSON object which you can read in with the [`json` module (https://docs.python.org/3.7/library/json.html)](https://docs.python.org/3.7/library/json.html). This will give you a dictionary mapping the integer coded labels to the actual names of the flowers.

In [10]:
```python
with open('label_map.json', 'r') as f:
    class_names = json.load(f)
```

In [12]:
```
class_names
```

Out[12]:
```
{'21': 'fire lily',
 '3': 'canterbury bells',
 '45': 'bolero deep blue',
 '1': 'pink primrose',
 '34': 'mexican aster',
 '27': 'prince of wales feathers',
 '7': 'moon orchid',
 '16': 'globe-flower',
 '25': 'grape hyacinth',
 '26': 'corn poppy',
 '79': 'toad lily',
 '39': 'siam tulip',
 '24': 'red ginger',
 '67': 'spring crocus',
 '35': 'alpine sea holly',
 '32': 'garden phlox',
 '10': 'globe thistle',
 '6': 'tiger lily',
 '93': 'ball moss',
 '33': 'love in the mist',
 '9': 'monkshood',
 '102': 'blackberry lily',
 '14': 'spear thistle',
 '19': 'balloon flower',
 '100': 'blanket flower',
 '13': 'king protea',
 '49': 'oxeye daisy',
 '15': 'yellow iris',
 '61': 'cautleya spicata',
 '31': 'carnation',
 '64': 'silverbush',
 '68': 'bearded iris',
 '63': 'black-eyed susan',
 '69': 'windflower',
 '62': 'japanese anemone',
 '20': 'giant white arum lily',
 '38': 'great masterwort',
 '4': 'sweet pea',
 '86': 'tree mallow',
 '101': 'trumpet creeper',
 '42': 'daffodil',
 '22': 'pincushion flower',
 '2': 'hard-leaved pocket orchid',
 '54': 'sunflower',
 '66': 'osteospermum',
 '70': 'tree poppy',
 '85': 'desert-rose',
 '99': 'bromelia',
 '87': 'magnolia',
 '5': 'english marigold',
 '92': 'bee balm',
 '28': 'stemless gentian',
 '97': 'mallow',
 '57': 'gaura',
 '40': 'lenten rose',
 '47': 'marigold',
 '59': 'orange dahlia',
```

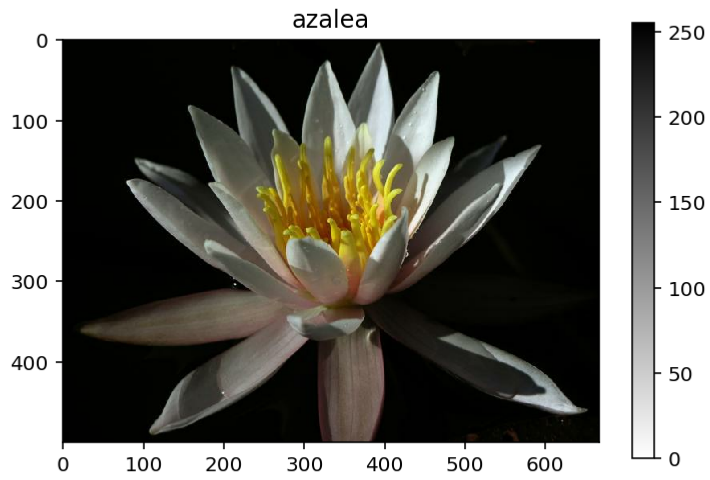
```
'48': 'buttercup',
'55': 'pelargonium',
'36': 'ruby-lipped cattleya',
'91': 'hippeastrum',
'29': 'artichoke',
'71': 'gazania',
'90': 'canna lily',
'18': 'peruvian lily',
'98': 'mexican petunia',
'8': 'bird of paradise',
'30': 'sweet william',
'17': 'purple coneflower',
'52': 'wild pansy',
'84': 'columbine',
'12': "colt's foot",
'11': 'snapdragon',
'96': 'camellia',
'23': 'fritillary',
'50': 'common dandelion',
'44': 'poinsettia',
'53': 'primula',
'72': 'azalea',
'65': 'californian poppy',
'80': 'anthurium',
'76': 'morning glory',
'37': 'cape flower',
'56': 'bishop of llandaff',
'60': 'pink-yellow dahlia',
'82': 'clematis',
'58': 'geranium',
'75': 'thorn apple',
'41': 'barbeton daisy',
'95': 'bougainvillea',
'43': 'sword lily',
'83': 'hibiscus',
'78': 'lotus lotus',
'88': 'cyclamen',
'94': 'foxglove',
'81': 'frangipani',
'74': 'rose',
'89': 'watercress',
'73': 'water lily',
'46': 'wallflower',
'77': 'passion flower',
'51': 'petunia'}
```

In [12]:
```
class_names['21'],len(class_names)
```

Out[12]:
```
('fire lily', 102)
```

In [13]:
```
# TODO: Plot 1 image from the training set. Set the title
# of the plot to the corresponding class name.
#class_names

for image, label in dataset['train'].take(1):
    image = image.numpy().squeeze()
    label = label.numpy()
plt.title(class_names[str(label)])
plt.imshow(image, cmap= plt.cm.binary)
plt.colorbar()
plt.show()
```



Create Pipeline

In [14]:
```
# TODO: Create a pipeline for each set.

train_num_examples = dataset_info.splits['train'].num_examples
val_num_examples = dataset_info.splits['validation'].num_examples
test_num_examples = dataset_info.splits['test'].num_examples
#batch_size = 32
batch_size = 64
image_size = 224

total_num_examples=train_num_examples+val_num_examples+test_num_examp
les

train_split=total_num_examples

#num_training_examples = (total_num_examples * train_split) // 100
num_training_examples = train_num_examples

def format_image(image, label):
    image = tf.cast(image, tf.float32)
    image = tf.image.resize(image, (image_size, image_size))
    image /= 255
    return image, label


training_batches = dataset['train'].shuffle(num_training_examples//4)
.map(format_image).batch(batch_size).prefetch(1)
validation_batches = dataset['validation'].map(format_image).batch(ba
tch_size).prefetch(1)
testing_batches = dataset['test'].map(format_image).batch(batch_size)
.prefetch(1)
```

In [ ]:

Build and Train the Classifier

Now that the data is ready, it's time to build and train the classifier. You should use the MobileNet pre-trained model from TensorFlow Hub to get the image features. Build and train a new feed-forward classifier using those features.

We're going to leave this part up to you. If you want to talk through it with someone, chat with your fellow students!

Refer to the rubric for guidance on successfully completing this section. Things you'll need to do:

- Load the MobileNet pre-trained network from TensorFlow Hub.
- Define a new, untrained feed-forward network as a classifier.
- Train the classifier.
- Plot the loss and accuracy values achieved during training for the training and validation set.
- Save your trained model as a Keras model.

We've left a cell open for you below, but use as many as you need. Our advice is to break the problem up into smaller parts you can run separately. Check that each part is doing what you expect, then move on to the next. You'll likely find that as you work through each part, you'll need to go back and modify your previous code. This is totally normal!

When training make sure you're updating only the weights of the feed-forward network. You should be able to get the validation accuracy above 70% if you build everything right.

Note for Workspace users: One important tip if you're using the workspace to run your code: To avoid having your workspace disconnect during the long-running tasks in this notebook, please read in the earlier page in this lesson called Intro to GPU Workspaces about Keeping Your Session Active. You'll want to include code from the workspace_utils.py module. Also, If your model is over 1 GB when saved as a checkpoint, there might be issues with saving backups in your workspace. If your saved checkpoint is larger than 1 GB (you can open a terminal and check with `ls -lh`), you should reduce the size of your hidden layers and train again.

In [16]:
```
URL = "https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vect
or/4"

feature_extractor = hub.KerasLayer(URL, input_shape=(image_size, imag
e_size,3))
```

In [17]:
```
# freezing the weights and biases
```

In [18]:
```
feature_extractor.trainable = False
```

In [19]:
```
# TODO: Build and train your network.
model_flowers = tf.keras.Sequential([
            feature_extractor,
#            tf.keras.layers.Flatten(input_shape=(image_size,image_siz
e,1)),
            tf.keras.layers.Dense(8192, activation = 'relu'),
            tf.keras.layers.Dropout(0.2),
            tf.keras.layers.Dense(1024, activation = 'relu'),
            tf.keras.layers.Dropout(0.2),
            tf.keras.layers.Dense(256, activation = 'relu'),
            tf.keras.layers.Dropout(0.2),
            tf.keras.layers.Dense(102, activation = 'softmax')
])
```

In [ ]:
```
```

In [20]:
```
model_flowers.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

#EPOCHS = 20
# 12 is enough
EPOCHS = 12

early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
patience=5)


history = model_flowers.fit(training_batches,
                    epochs=EPOCHS,
                    validation_data=validation_batches)
```

```
Epoch 1/12
16/16 [==============================] - 56s 3s/step - loss: 4.9735 -
accuracy: 0.0066 - val_loss: 4.3788 - val_accuracy: 0.0647
Epoch 2/12
16/16 [==============================] - 51s 3s/step - loss: 4.2077 -
accuracy: 0.0925 - val_loss: 3.4242 - val_accuracy: 0.2500
Epoch 3/12
16/16 [==============================] - 51s 3s/step - loss: 3.1971 -
accuracy: 0.2348 - val_loss: 2.4669 - val_accuracy: 0.4392
Epoch 4/12
16/16 [==============================] - 51s 3s/step - loss: 2.1406 -
accuracy: 0.4673 - val_loss: 1.7953 - val_accuracy: 0.5735
Epoch 5/12
16/16 [==============================] - 51s 3s/step - loss: 1.4453 -
accuracy: 0.6254 - val_loss: 1.4611 - val_accuracy: 0.6235
Epoch 6/12
16/16 [==============================] - 50s 3s/step - loss: 1.0310 -
accuracy: 0.7051 - val_loss: 1.2571 - val_accuracy: 0.6961
Epoch 7/12
16/16 [==============================] - 51s 3s/step - loss: 0.5703 -
accuracy: 0.8320 - val_loss: 1.1447 - val_accuracy: 0.7275
Epoch 8/12
16/16 [==============================] - 51s 3s/step - loss: 0.4542 -
accuracy: 0.8744 - val_loss: 1.2402 - val_accuracy: 0.6853
Epoch 9/12
16/16 [==============================] - 51s 3s/step - loss: 0.3286 -
accuracy: 0.9119 - val_loss: 1.0524 - val_accuracy: 0.7431
Epoch 10/12
16/16 [==============================] - 51s 3s/step - loss: 0.1947 -
accuracy: 0.9446 - val_loss: 1.0154 - val_accuracy: 0.7608
Epoch 11/12
16/16 [==============================] - 51s 3s/step - loss: 0.1642 -
accuracy: 0.9539 - val_loss: 1.0589 - val_accuracy: 0.7294
Epoch 12/12
16/16 [==============================] - 51s 3s/step - loss: 0.1384 -
accuracy: 0.9675 - val_loss: 0.9862 - val_accuracy: 0.7627
```

98% accuracy in the 15 epoch! This is awesome!

In [ ]:

In [21]:
```
model_flowers.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
keras_layer (KerasLayer)     (None, 1280)              2257984   
_________________________________________________________________
dense (Dense)                (None, 8192)              10493952  
_________________________________________________________________
dropout (Dropout)            (None, 8192)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              8389632   
_________________________________________________________________
dropout_1 (Dropout)          (None, 1024)              0         
_________________________________________________________________
dense_2 (Dense)              (None, 256)               262400    
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_3 (Dense)              (None, 102)               26214     
=================================================================
Total params: 21,430,182
Trainable params: 19,172,198
Non-trainable params: 2,257,984
_________________________________________________________________
```

In [ ]:

In [22]:
```
# TODO: Plot the loss and accuracy values achieved during training fo
r the training and validation set.
```

In [23]:
```
# TOOK from the classes
```

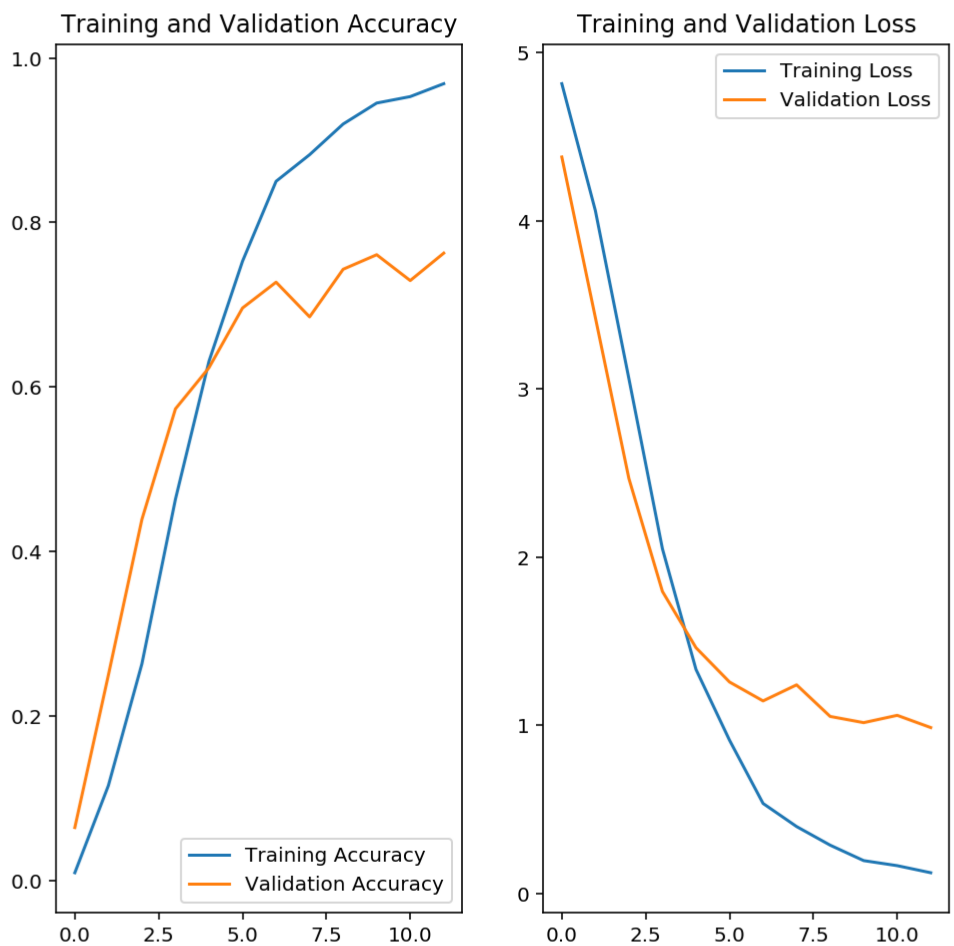
In [24]:
```
training_accuracy = history.history['accuracy']
validation_accuracy = history.history['val_accuracy']

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

epochs_range=range(len(training_accuracy))

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, training_accuracy, label='Training Accuracy')
plt.plot(epochs_range, validation_accuracy, label='Validation Accurac
y')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, training_loss, label='Training Loss')
plt.plot(epochs_range, validation_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



In [ ]:

Testing your Network

It's good practice to test your trained network on test data, images the network has never seen either in training or validation. This will give you a good estimate for the model's performance on completely new images. You should be able to reach around 70% accuracy on the test set if the model has been trained well.

In [28]:
```
# TODO: Print the loss and accuracy values achieved on the entire tes
t set.
loss_test, accuracy_test = model_flowers.evaluate(testing_batches)
```

```
97/97 [==============================] - 146s 1s/step - loss: 1.1224
- accuracy: 0.7333
```

In [ ]:

Hopefully, accuracy is greater than 70%!

In [ ]:

Save the Model

Now that your network is trained, save the model so you can load it later for making inference. In the cell below save your model as a Keras model (*i.e.* save it as an HDF5 file).

In [41]:
```
# TODO: Save your trained model as a Keras model.
t = time.time()

#saved_keras_model_filepath = './{}.h5'.format(int(t))
saved_keras_model_filepath = './{}.h5'.format('zeizer_model')

model_flowers.save(saved_keras_model_filepath)
```

Load the Keras Model

Load the Keras model you saved above.

In [14]:
```
# TODO: Load the Keras model

saved_keras_model_filepath = './{}.h5'.format('zeizer_model')

#reloaded_SavedModel =  tf.keras.models.load_model('zeizer_model.h5')

reloaded_SavedModel = tf.keras.models.load_model('./zeizer_model.h5',
custom_objects={'KerasLayer':hub.KerasLayer})


#reloaded_SavedModel = tf.saved_model.load('zeizer_model.h5')
```

In [15]:
```
reloaded_SavedModel.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
keras_layer (KerasLayer)     (None, 1280)              2257984   
_________________________________________________________________
dense (Dense)                (None, 8192)              10493952  
_________________________________________________________________
dropout (Dropout)            (None, 8192)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              8389632   
_________________________________________________________________
dropout_1 (Dropout)          (None, 1024)              0         
_________________________________________________________________
dense_2 (Dense)              (None, 256)               262400    
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_3 (Dense)              (None, 102)               26214     
=================================================================
Total params: 21,430,182
Trainable params: 19,172,198
Non-trainable params: 2,257,984
_________________________________________________________________
```

Inference for Classification

Now you'll write a function that uses your trained network for inference. Write a function called `predict` that takes an image, a model, and then returns the top K most likely class labels along with the probabilities. The function call should look like:

```
probs, classes = predict(image_path, model, top_k)
```

If `top_k=5` the output of the `predict` function should be something like this:

```
probs, classes = predict(image_path, model, 5)
print(probs)
print(classes)
> [ 0.01558163  0.01541934  0.01452626  0.01443549  0.01407339]
> ['70', '3', '45', '62', '55']
```

Your `predict` function should use `PIL` to load the image from the given `image_path` . You can use the Image.open (https://pillow.readthedocs.io/en/latest/reference/Image.html#PIL.Image.open) function to load the images. The `Image.open()` function returns an `Image` object. You can convert this `Image` object to a NumPy array by using the `np.asarray()` function.

The `predict` function will also need to handle pre-processing the input image such that it can be used by your model. We recommend you write a separate function called `process_image` that performs the pre-processing. You can then call the `process_image` function from the `predict` function.

Image Pre-processing

The `process_image` function should take in an image (in the form of a NumPy array) and return an image in the form of a NumPy array with shape `(224, 224, 3)` .

First, you should convert your image into a TensorFlow Tensor and then resize it to the appropriate size using `tf.image.resize` .

Second, the pixel values of the input images are typically encoded as integers in the range 0-255, but the model expects the pixel values to be floats in the range 0-1. Therefore, you'll also need to normalize the pixel values.

Finally, convert your image back to a NumPy array using the `.numpy()` method.

In [16]:

```
def process_image(image):
 #   im = Image.open(image)
#    im_tf=tf.image.resize(image,[224,224,3])
#    image=tf.image.convert_image_dtype(image, dtype=tf.float16, satu
rate=False)
    image=tf.convert_to_tensor(image,dtype=tf.float16)
    im_tf=tf.image.resize(image,[224,224])
    np_image=im_tf.numpy()
    np_image=np.array(np_image)/255
#    np_image=(np_image -np.array([0.485,0.456,0.406]))/np.array([0.2
29,0.224,0.225])
#    np_image=np_image.transpose((2, 0, 1))
    return np_image
```

In [ ]:

```

```

In [17]:
```
"""
def process_image(image):
    ''' Scales, crops, and normalizes a PIL image for a PyTorch mode
l,
    returns an Numpy array
    '''
# TODO: Process a PIL image for use in a PyTorch model
#
#
#
    im = Image.open(image)

    if im.size[0] > im.size[1]: #(if the width > height)
        im.thumbnail((1000000, 256)) #constrain the height to be 256
    else:
        im.thumbnail((256, 200000)) #otherwise constrain the width
    left_margin = (im.width-224)/2
    bottom_margin = (im.height-224)/2
    right_margin = left_margin + 224
    top_margin = bottom_margin + 224
    im = im.crop((left_margin, bottom_margin, right_margin,
    top_margin))
    np_image=np.array(im)/255
    np_image=(np_image -np.array([0.485,0.456,0.406]))/np.array([0.22
9,0.224,0.225])
    np_image=np_image.transpose((2, 0, 1))
    return np_image
"""
```

Out[17]:
```
"\ndef process_image(image):\n    ''' Scales, crops, and normalizes a
PIL image for a PyTorch model,\n    returns an Numpy array\n    '''\n
# TODO: Process a PIL image for use in a PyTorch model\n#\n#\n#\n
im = Image.open(image)\n    \n    if im.size[0] > im.size[1]: #(if th
e width > height)\n        im.thumbnail((1000000, 256)) #constrain th
e height to be 256\n    else:\n        im.thumbnail((256, 200000)) #o
therwise constrain the width\n    left_margin = (im.width-224)/2\n
bottom_margin = (im.height-224)/2\n    right_margin = left_margin + 2
24\n    top_margin = bottom_margin + 224\n    im = im.crop((left_marg
in, bottom_margin, right_margin,\n    top_margin))\n    np_image=np.a
rray(im)/255\n    np_image=(np_image -np.array([0.485,0.456,0.406]))/
np.array([0.229,0.224,0.225])\n    np_image=np_image.transpose((2, 0,
1))\n    return np_image\n"
```

In [ ]:

TODO: Create the process_image function

""" def imshow(image, ax=None, title=None): """Imshow for Tensor.""" if ax is None: fig, ax = plt.subplots() #

TF tensors assume the color channel is the first dimension

but matplotlib assumes is the third dimension

image = image.numpy().transpose((1, 2, 0))

```
image = image.transpose((1, 2, 0))
# Undo preprocessing
mean = np.array([0.485, 0.456, 0.406])
std = np.array([0.229, 0.224, 0.225])
image = std * image + mean
# Image needs to be clipped between 0 and 1 or it looks like noise when disp
#layed
image = np.clip(image, 0, 1)
ax.imshow(image)

return ax
```

"""

In [18]:
```
image='./test_images/cautleya_spicata.jpg'
```

In [19]:
```
#imshow(process_image(image))
```

In [ ]:
```

```

To check your `process_image` function we have provided 4 images in the `./test_images/` folder:

- cautleya_spicata.jpg
- hard-leaved_pocket_orchid.jpg
- orange_dahlia.jpg
- wild_pansy.jpg

The code below loads one of the above images using `PIL` and plots the original image alongside the image produced by your `process_image` function. If your `process_image` function works, the plotted image should be the correct size.

In [20]:
```
#imshow('./test_images/cautleya_spicata.jpg')
```

In [21]:
```
image_path = './test_images/hard-leaved_pocket_orchid.jpg'
im = Image.open(image_path)
test_image = np.asarray(im)
```

In [22]:
```
len(test_image[0][0])
```

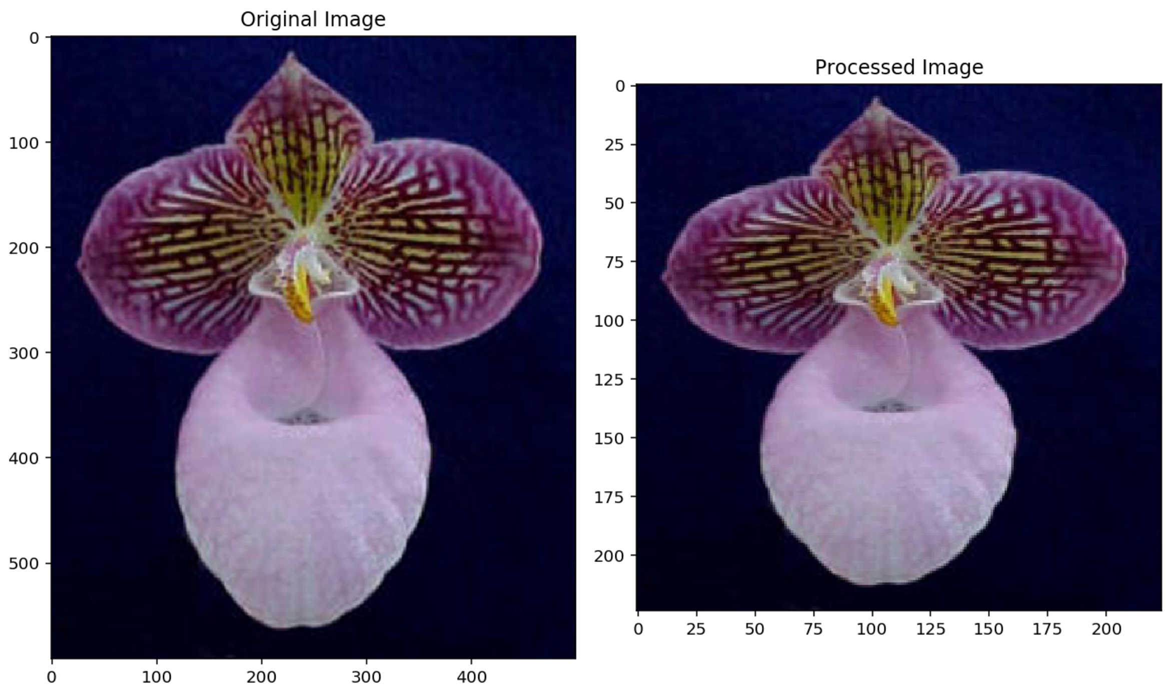
Out[22]: 3

In [23]:
```
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)
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()
```



Once you can get images in the correct format, it's time to write the `predict` function for making inference with your model.

Inference

Remember, the `predict` function should take an image, a model, and then returns the top K most likely class labels along with the probabilities. The function call should look like:

```
probs, classes = predict(image_path, model, top_k)
```

If `top_k=5` the output of the `predict` function should be something like this:

```
probs, classes = predict(image_path, model, 5)
print(probs)
print(classes)
> [ 0.01558163  0.01541934  0.01452626  0.01443549  0.01407339]
> ['70', '3', '45', '62', '55']
```

Your `predict` function should use `PIL` to load the image from the given `image_path`. You can use the [Image.open (https://pillow.readthedocs.io/en/latest/reference/Image.html#PIL.Image.open)](https://pillow.readthedocs.io/en/latest/reference/Image.html#PIL.Image.open) function to load the images. The `Image.open()` function returns an `Image` object. You can convert this `Image` object to a NumPy array by using the `np.asarray()` function.

Note: The image returned by the `process_image` function is a NumPy array with shape `(224, 224, 3)` but the model expects the input images to be of shape `(1, 224, 224, 3)`. This extra dimension represents the batch size. We suggest you use the `np.expand_dims()` function to add the extra dimension.

In [24]:
```
reloaded_SavedModel.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
keras_layer (KerasLayer)     (None, 1280)              2257984   
_________________________________________________________________
dense (Dense)                (None, 8192)              10493952  
_________________________________________________________________
dropout (Dropout)            (None, 8192)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              8389632   
_________________________________________________________________
dropout_1 (Dropout)          (None, 1024)              0         
_________________________________________________________________
dense_2 (Dense)              (None, 256)               262400    
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_3 (Dense)              (None, 102)               26214     
=================================================================
Total params: 21,430,182
Trainable params: 19,172,198
Non-trainable params: 2,257,984
_________________________________________________________________
```

In [142]:
```
#dict_cx=reloaded_SavedModel.class_to_index
#inverted_cx = dict([v,k] for k,v in dict_cx.items())
```

```
-----------------------------------------------------------------------
----
AttributeError                            Traceback (most recent call
last)
<ipython-input-142-3a78c0c63a8a> in <module>
----> 1 dict_cx=reloaded_SavedModel.class_to_index
      2 inverted_cx = dict([v,k] for k,v in dict_cx.items())

AttributeError: 'Sequential' object has no attribute 'class_to_index'
```

In [25]:
```
# TODO: Create the predict function

def predict(image_path, model, topk=5):
    im = Image.open(image_path)
    test_image = np.asarray(im)
#    test_image = tf.cast(test_image, tf.float32)

    inputs=process_image(test_image)
#    inputs=tf.convert_to_tensor(inputs, dtype=tf.float32)
    results=model.predict(np.expand_dims(inputs,axis=0))
    results=results[0].tolist()
    val,ind=tf.math.top_k(results,k=topk,sorted=True)
#    sorted_vals=np.argsort(results[0])[::-1][:len(results)]
#    pred_class=class_names[str(sorted_vals[:k])]
    top_k_val=val.numpy().tolist()
    top_k_ind=ind.numpy().tolist()
    flowers=[class_names[str(i+1)] for i in top_k_ind]
#    top_k_flowers=np.argsort(top[0])[::-1][:len(results)]
#    return top_k_val,top_k_ind#,sorted_vals#[13]
    return top_k_val,flowers
```

Sanity Check

It's always good to check the predictions made by your model to make sure they are correct. To check your predictions we have provided 4 images in the `./test_images/` folder:

- cautleya_spicata.jpg
- hard-leaved_pocket_orchid.jpg
- orange_dahlia.jpg
- wild_pansy.jpg

In the cell below use `matplotlib` to plot the input image alongside the probabilities for the top 5 classes predicted by your model. Plot the probabilities as a bar graph. The plot should look like this:

You can convert from the class integer labels to actual flower names using `class_names`.

In [26]:
```
# TODO: Plot the input image along with the top 5 classes
predict('./test_images/hard-leaved_pocket_orchid.jpg',reloaded_SavedM
odel)
#predict('./test_images/orange_dahlia.jpg',reloaded_SavedModel)
```

Out[26]:
```
([0.9999911785125732,
  2.7287348984827986e-06,
  2.4213045435317326e-06,
  1.1240359754083329e-06,
  7.010290232756233e-07],
 ['hard-leaved pocket orchid',
  'king protea',
  'passion flower',
  'carnation',
  'spring crocus'])
```

In [27]:
```
class_names['76']
```

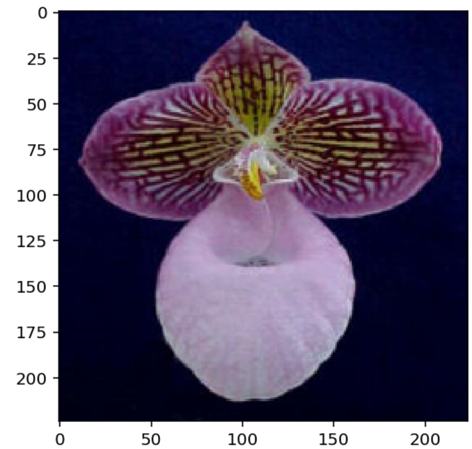
Out[27]:
```
'morning glory'
```

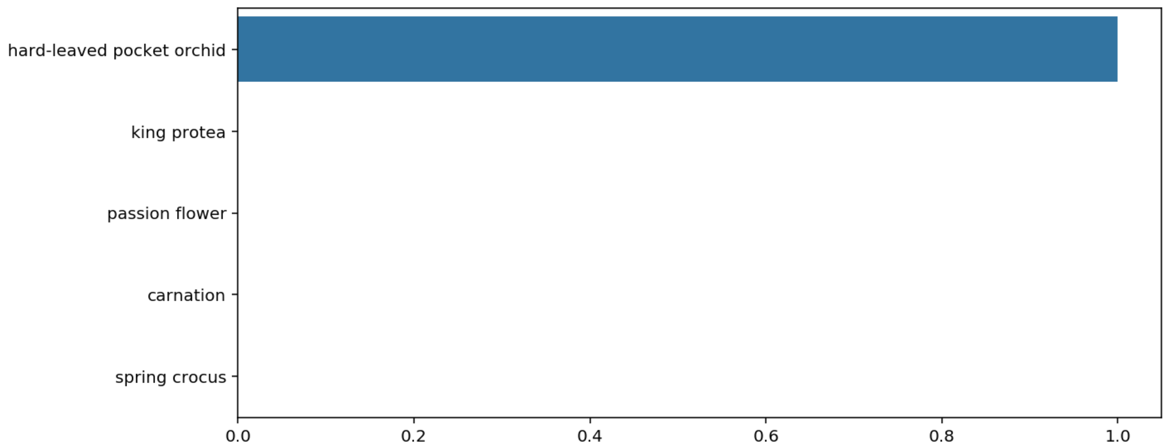
In [28]:
```
import seaborn as sns
```

In [31]:
```
def plot_testing(model,image_path):
# Setting up the plot
    plt.figure(figsize = (10,10))
    ax = plt.subplot(2,1,1)
    # Setting up the title
    # taking the third element from the splitting of the path
    flower_num = image_path.split('/')[2]
    # using the json from the beginning!
#    title_ = class_names[flower_num]
    title_ = 'Flower Classification'
    im = Image.open(image_path)
    test_image = np.asarray(im)

#processed_test_image = process_image(test_image)
    processed_test_image = process_image(test_image)
#    image = process_image(image_path)
#    imshow(processed_test_image, ax, title = title_);
    ax.imshow(processed_test_image)
    probs, flowers = predict(image_path, model)
    plt.subplot(2,1,2)
#    flowers=[class_names[str(inds[i])] for i in range(len(inds))]
    sns.barplot(x=probs, y=flowers, color=sns.color_palette()[0]);
    plt.show()
```

In [32]:
```
#plot_testing(reloaded_SavedModel,'./test_images/orange_dahlia.jpg')
plot_testing(reloaded_SavedModel,'./test_images/hard-leaved_pocket_or
chid.jpg')
```





In [ ]:

In [ ]: