# Project 2: Flower Identification Model

Introduction

This tutorial assumes you have intermediate programming skills, preferably in Python, and basic knowledge on how to use a command prompt. We will guide you through the process of setting up a GPU (graphics processing unit) cloud, the ssh log-in, and how to build and train a deep learning model. We will show you how to prepare your image data for testing and training the model, train a pre-trained model, and verify the accuracy of your newly trained model. By the end of this tutorial, you will have an understanding of the process of how neural networks work and how neural networks can be used to solve many of the world's problems.

OVERALL LEARNING OBJECTIVES:

1. Organize data into usable categories
2. Build and train a deep learning model
3. Verify the accuracy of your model by using it to predict on test data

GENERAL TIMELINE:

|  |  |
| --- | --- |
| Part 1: Introduction to Deep Learning | Challenge #1: Getting Started  Challenge #2: Deep Learning Intro  Challenge #3: Flowers vs Leaves |
| Part 2: Flower Identification Model | Challenge #1: Flowers Classification  Challenge #2: Model-Mobile App Integration |

REQUIRED PROJECT PARTS:

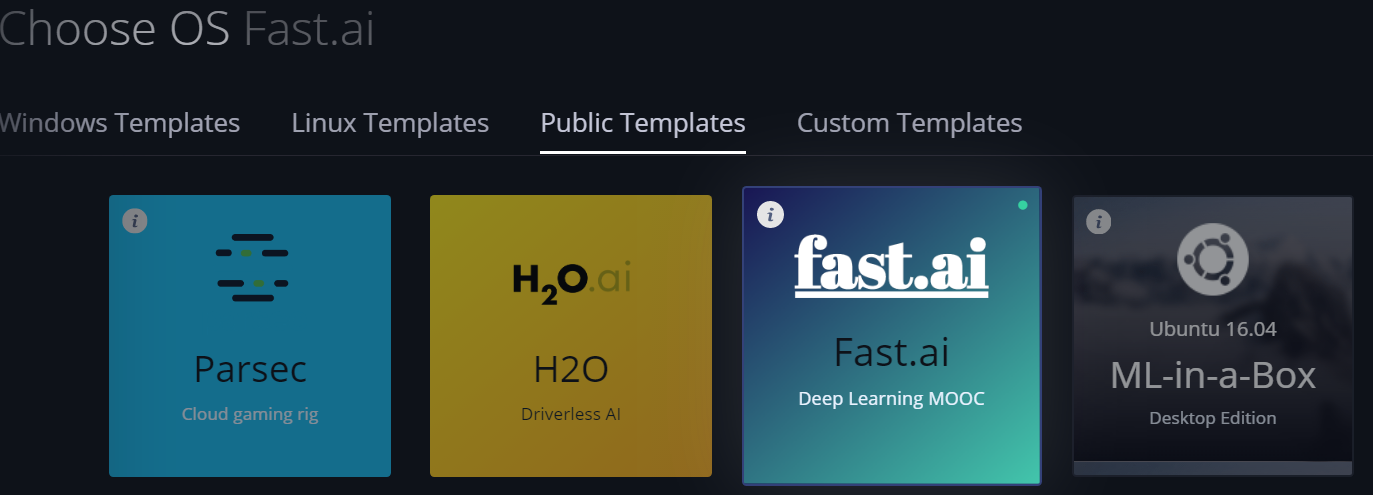
None

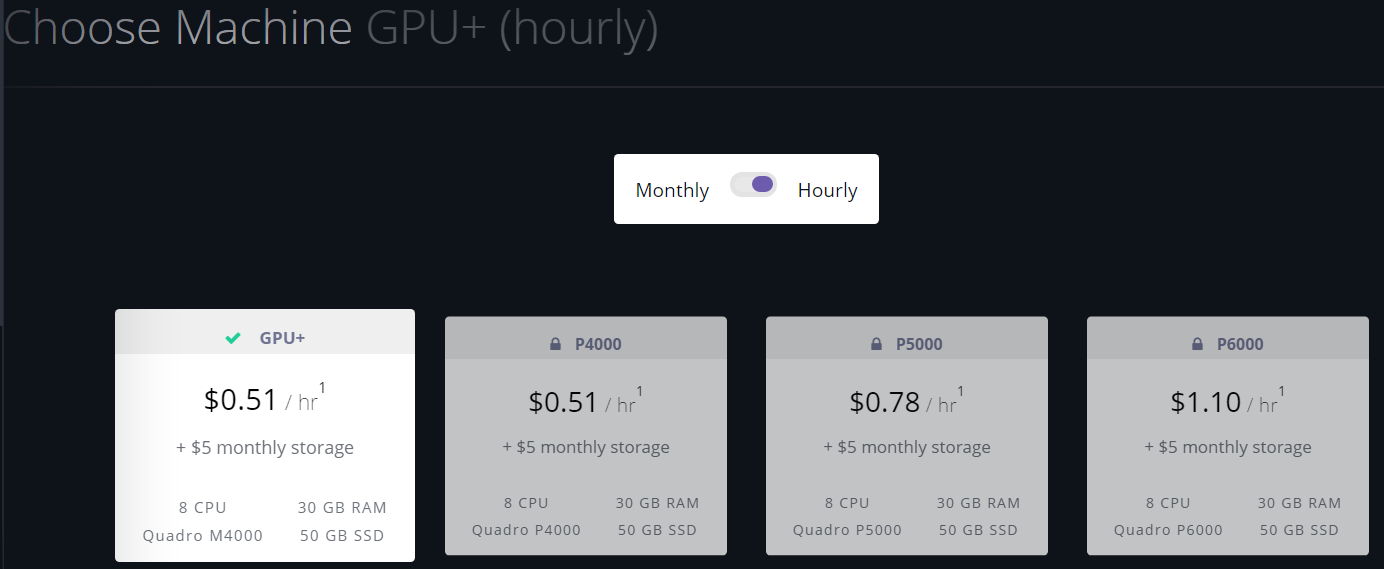
REQUIRED PROJECT TOOLS/EQUIPMENT:

* Laptop Computer with a command prompt terminal
* Monitor
* Mouse
* Keyboard
* GPU like Paperspace

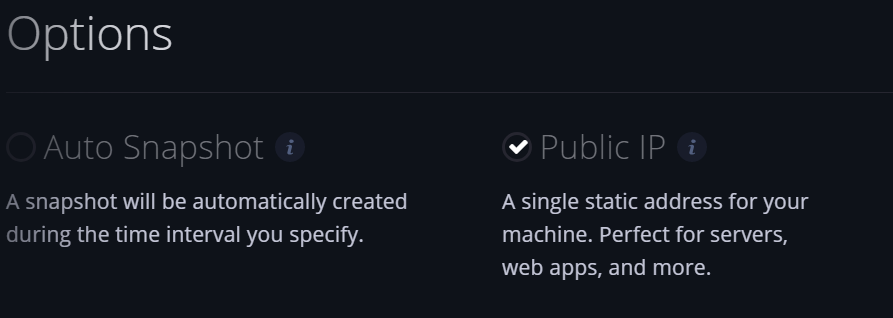
## Challenge 1: Getting Started

### Paperspace Set Up

1. Set up an account on <https://www.paperspace.com>
2. Go to <https://www.paperspace.com/console/machines>
3. Click 
4. Region: US West
5. OS: Public Templates > fast.ai
   1. 
6. Machine > Hourly > GPU+



1. Leave the rest at default; scroll down to Options and select Public IP (deselect Auto Snapshot)



1. Add a card and click Create Your Paperspace
   1. Costs will be reimbursed
2. Check your email confirming the creation of your machine with a temporary password

### SSH and Jupyter Notebook

For Windows users: We will be using Git Bash as a Unix environment. Download here: <https://gitforwindows.org/>

Follow the steps starting from Part II to Part V

<https://github.com/reshamas/fastai_deeplearn_part1/blob/master/tools/paperspace.md>

These **optional** steps below are recommended if you would like to ssh into your machine easily:

Extra steps for Mac:

1. Check that homebrew is installed if using Mac. If not, install homebrew using this command  
   /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
2. Install ssh-copy-id if not previously done before (Mac only) with command  
   brew install ssh-copy-id

For both Mac and Unix

1. Change directories to ~/.ssh (make a .ssh directory if one does not have one with mkdir ssh)
2. If there is not a id\_rsa.pub file in the .ssh directory, create one using ssh-keygen and hitting enter when prompted.
3. Run the command   
   ssh-copy-id -i ~/.ssh/id\_rsa.pub paperspace@184.###.###.###  
   By first replacing the ### signs for the public IP address for your paperspace machine.
4. Still in the .ssh directory, use a text editor like nano or vim to create and edit the config file.
   1. If no config file exists, make one by entering vim config
5. Put this in the config file, again replacing ### with the machine’s public IP address, and save the file.   
   Host paperspace   
   HostName 184.###.###.###  
   IdentityFile ~/.ssh/id\_rsa  
   # StrictHostKeyChecking no   
   User paperspace  
   LocalForward 8888 localhost:8888
6. Start Paperspace machine
7. On command prompt or terminal, ssh onto the machine’s server with the command   
   ssh -Y paperspace  
   (Make sure that the paperspace machine is up and running before you run this command in the terminal. If the command seems to not be loading, cancel the command with <Ctrl C>, check that the machine’s status says “On/Ready”, and rerun the command)
8. If everything has been done properly so far, the command line prompt should say   
   (fastai) paperspace@\_\_\_\_\_\_:~$
9. Type command   
   jupyter notebook   
   To open jupyter notebook   
   (This command will provide a url. Open a web browser and type localhost:8888 into the browser which will take you to the jupyter notebook log in page. Log in by pasting the token provided into the window.   
   <http://localhost:8888/?token=2f2a510fefd035d65ef86637597ec5159e97b211cc1088ed>  
   Is an example of a url the command may provide you in which everything after “token=” is the token to log in.

## Challenge 2: Deep Learning Intro

### Tutorials

A video on Jupyter Notebook and deep learning fundamentals - please follow along on Jupyter Notebook

<http://course18.fast.ai/lessons/lesson1.html>

Start: 12:15

Questions:

What are the three lines of code necessary to train a model?

What is an epoch?

Where can you find the accuracy of the model?

What is a neural network?

What is deep learning?

What does a kernel do?

What is gradient descent and what important number does it give us?

What function gives us the learning rate?

In Jupyter notebook, what do you press to show a list of methods for a variable?

What do you press to show what arguments you need for that method?

How do you find the source code?

## Challenge 3: Flowers vs Leaves

### Flowers vs. Leaves

We will be using the Dogs vs. Cats notebook (lesson1.ipynb) to create a Flowers vs. Leaves model

1. Download 10 different images of flowers and leaves from Google Images
2. Upload to Jupyter Notebook under the data folder
3. Separate into train and validation folders with 5 images each
4. Create a copy of lesson1.ipynb and rename it
5. Change the PATH to where your train and validation folders are
6. Run through the notebook cell by cell and change paths when needed

## 

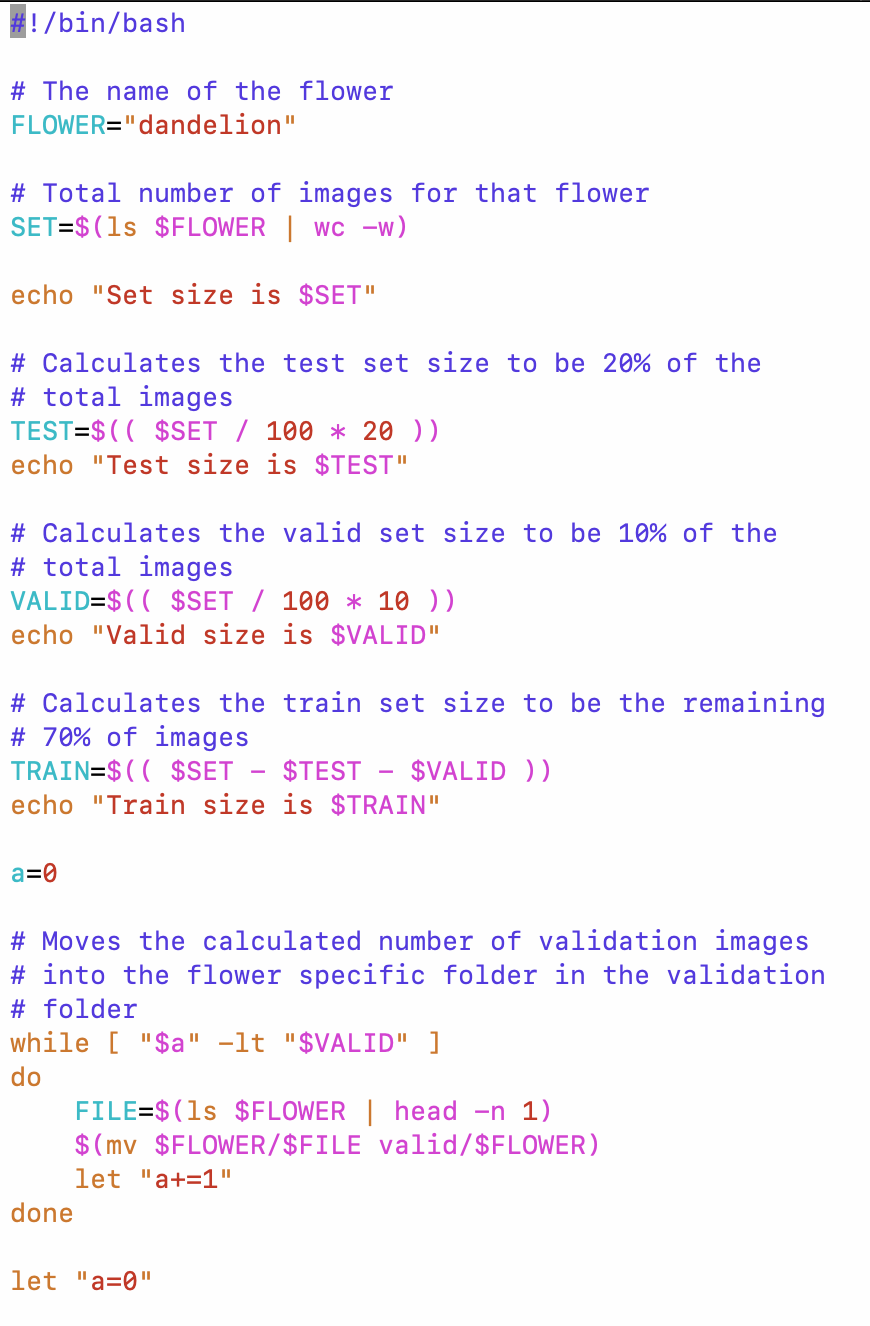
# Project 3: Flower Identification Model cont.

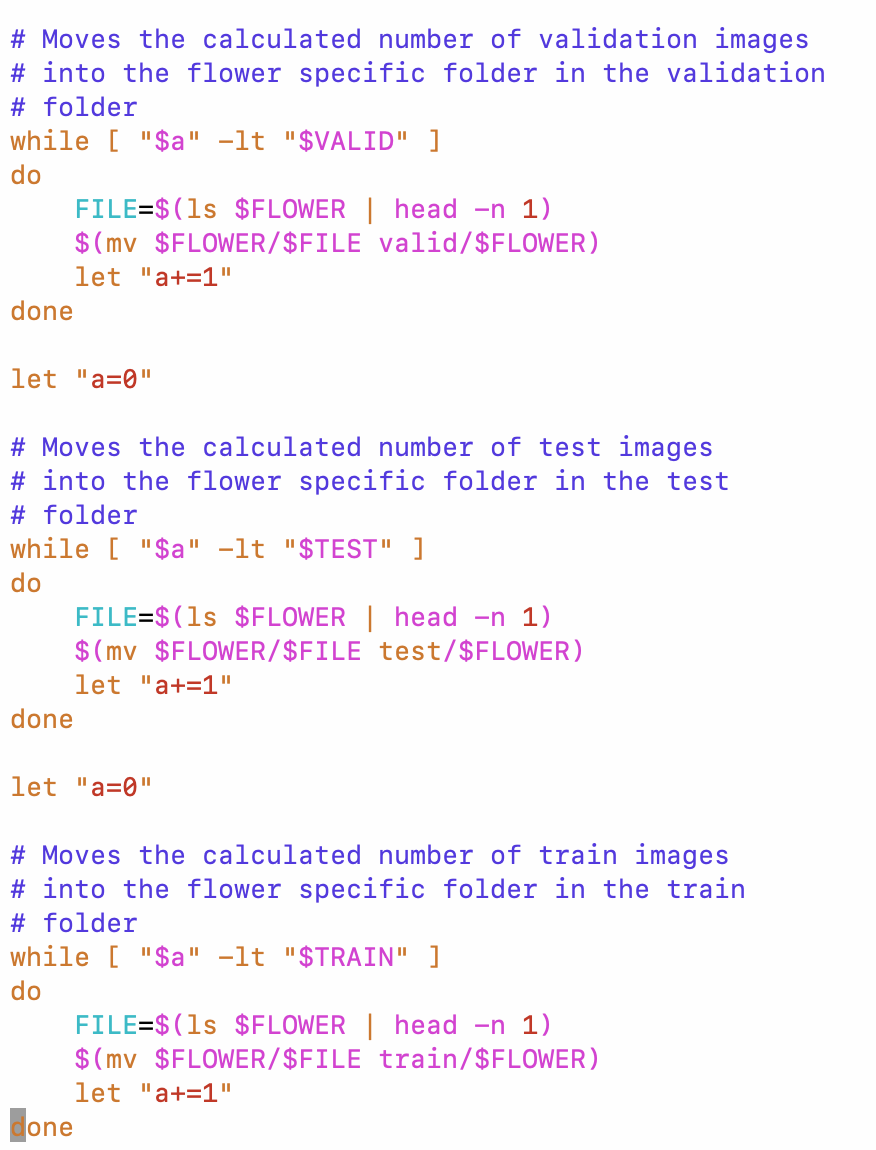
## Challenge 1: Flowers Classification

We will be using a Keras tutorial to create a Flowers Species model. This tutorial specifically describes how to create a Keras model that is able to identify different species of flowers.

**Preprocessing Data**

1. Download the dataset on <https://www.kaggle.com/alxmamaev/flowers-recognition>
2. Upload to Jupyter under the data folder
3. Create a directory to store all data
4. Make a copy of the unzipped data
5. Use the command line to unzip the folder resulting in five subdirectories in the data file. One for each flower species.
6. Create three more subdirectories called “test”, “valid”, and “train”
7. Create subdirectories within each subdirectory (test, valid, and train) for all the flower species
8. Use this script to copy all pictures into the test, validation, and train directories





This script calculates the number of images that should be moved to each directory based on the percentage of the total number of images in each category, and then, moves the calculated number of images to the specified flower directory.

1. Run the script five times, changing the name of the flower at the beginning of the script to match the name of the flower subdirectories resulting in five full subdirectories in each of the test, valid, and train subdirectories. Or, add on to the script to have it do that for you!
2. If at any point in the process, a mistake is made, one can remove all the subdirectories created and restart the data sorting process with the data copied in step 4.

**Training the Model**

[**https://hackernoon.com/tf-serving-keras-mobilenetv2-632b8d92983c**](https://hackernoon.com/tf-serving-keras-mobilenetv2-632b8d92983c)

1. Import the following packages as follows:

import numpy as np

import keras

from keras.applications import \*

from keras.layers import Dense, Input, Dropout

from keras.models import Model

from keras import backend as K

from keras.layers.core import Dense, Flatten

from keras.optimizers import Adam

from keras.metrics import categorical\_crossentropy

from keras.preprocessing.image import ImageDataGenerator

from keras.layers.normalization import BatchNormalization

from keras.layers.convolutional import \*

from matplotlib import pyplot as plt

from sklearn.metrics import confusion\_matrix

import itertools

import matplotlib.pyplot as plt

%matplotlib inline   
from keras.applications.mobilenet import MobileNet

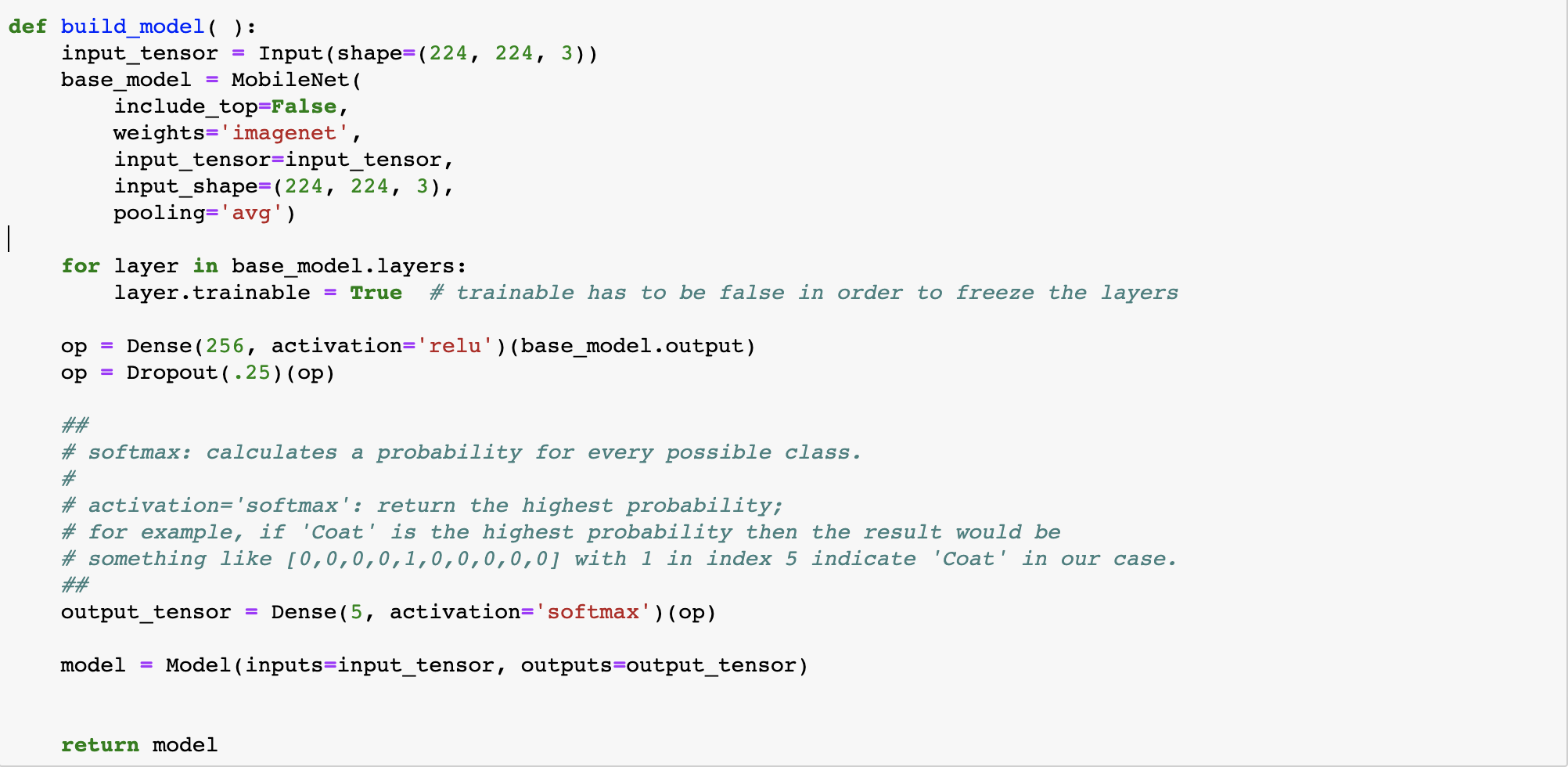
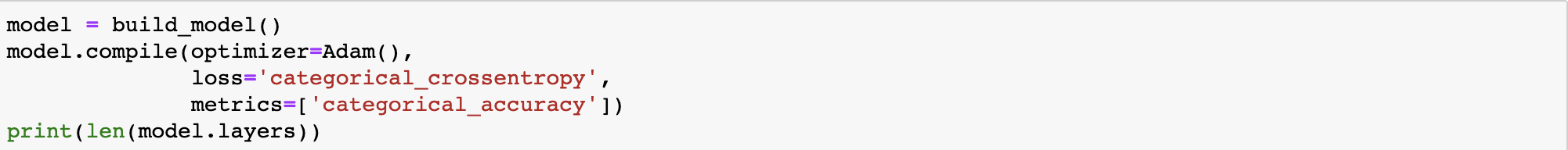
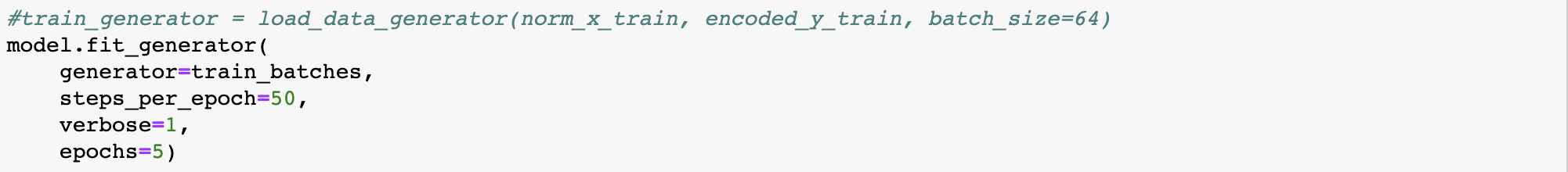
from keras.layers import Dense, Input, Dropout

from keras.models import Model  
from sklearn.utils import shuffle

from keras.models import load\_model

1. Define a variable to store the relative path to the train, test, and valid directory beginning from the directory your project is stored in.
2. Initialize an ImageDataGenerator for the train, test, and validation images. Specify the flow from the directory by passing the path to the train directory, a target size of (224, 224), classes with names that match the subdirectories of the species, and a batch\_size that easily divides the total number of train images.

Do the same for the test and validation batches. The only part of the code that may have to be changed between the train, test, and validation batches is the batch\_size.  
  
The way that the model is trained is in batches. The batch size value specifies how large each batch is. For our purposes, what’s important is not the value of the batch size, but rather that we train on all the images that we have to maximize the use of our data. So choosing a batch\_size that is divisible, or almost divisible by the batch size is important.

1. Use the function next() to get the images and labels for the next set of training images.
2. Define a function that initializes the Mobile Net model. In the following function, the Mobile Net model is instantiated and all the values in the model are unfrozen. This means that when training, those values can be adjusted to better fit the data that we are inputting into the model.   
   
3. Initialize and compile the Mobile Net model with the specified fields. The optimizer Adam specifies certain preset parameters like the learning rate and decay.   
   
4. Train the model using these fields. The generator specifies the data that will be used to train the model. The steps per epoch, when multiplied with the batch size, should equal the number of data you have. Verbose indicates the amount of information is displayed by the training process. Epochs refer to the number of full run-throughs of the data the training process goes through during.   
   
5. Save the trained model for later loading under a descriptive file name and delete the local model object.



**Predictions**  
  
Packages  
import numpy as np

from keras.models import load\_model

import keras

from keras import backend as K

from keras.models import Sequential

from keras.layers import Activation

from keras.layers.core import Dense, Flatten

from keras.optimizers import Adam

from keras.metrics import categorical\_crossentropy

from keras.preprocessing.image import ImageDataGenerator

from keras.layers.normalization import BatchNormalization

from keras.layers.convolutional import \*

from matplotlib import pyplot as plt

from sklearn.metrics import confusion\_matrix

import itertools

import matplotlib.pyplot as plt

%matplotlib inline

from keras.utils.generic\_utils import CustomObjectScope

1. Load a previously saved model.

In python, the with statement is a kind of shortcut that allows you to reference certain objects by name instead of referencing it through function calls and data fields. For example:   
  
with name As Object.name:   
 print(name)

Will print the name of the Object. It creates an alias for the name of the object to be used within the with block.

The statement below is a special kind of with statement because it uses the CustomObjectScope() function which changes the global custom objects. So it allows the keras.applications.mobilenet.relu6 to be referenced globally by relu6. This allows you to load the model with the proper relu6 and DepthwiseConv2D objects from mobile\_net.

# returns a compiled model

# identical to the previous one

With CustomObjectScope({‘relu6’:

keras.applications.mobilenet.relu6, ‘DepthwiseConv2D’:

keras.applications.mobilenet.DepthwiseConv2D}):

# Load previously saved model   
 model = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  


1. Define a function that plots the image data and labels within the jupyter notebook. This allows us to see the data and true labels. It is a good idea to make sure that the data looks as you expect it to be. You may need to tweak the function a little bit so the data is readable.   
     
   #plots images with labels within jupyter notebook

def plots(ims, figsize=(12,6), rows=1, interp=False, titles=None):

if type(ims[0]) is np.ndarray:

ims = np.array(ims).astype(np.uint8)

if (ims.shape[-1] != 3):

ims = ims.transpose((0,2,3,1))

f = plt.figure(figsize=figsize)

cols = len(ims)//row if len(ims) % 2 == 0

else len(ims)//rows + 1

for i in range(len(ims)):

sp = f.add\_subplot(rows, cols, i+1)

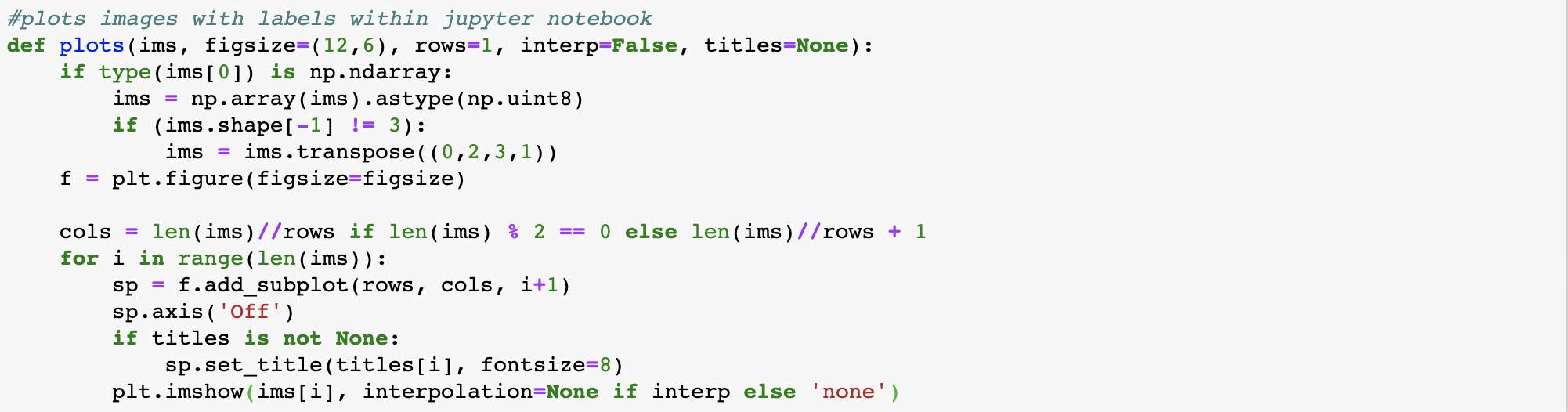
sp.axis(‘Off’)

if titles is not None:

sp.set\_title(titles[i], fontsize=8)

plt.imshow(ims[i], interpolation=None if interp

else ‘none’)



1. Define the test batches using images from the test data folder. The image data generator will generate batches of tensor image data. The test path specifies the relative path of the test data. The target size specifies the height and width all the test images will be resized. And the classes specify the classes the images come from. Then collect all the test images and test labels.   
     
   test\_path = ‘flower-data/test’

# initialize the image generator

image\_generator = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# create test batches from the flowers classes

test\_batches = image\_generator.flow\_from\_directory(test\_path,

target\_size=(224,224), classes=[‘rose’,

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_], batch\_size=35)

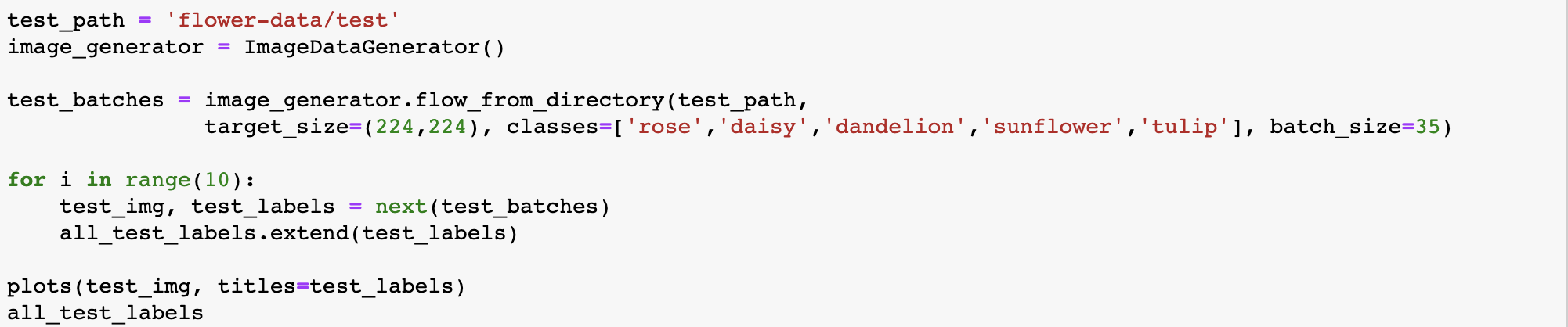
for i in range(10):

# get the next batch of test images

test\_img, test\_labels = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# add the test labels to the end of the all\_test\_labels list

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

plots(test\_img, titles=test\_labels)   
  


1. Turn the two-dimensional test labels into one-dimensional data ranging from 1-5 (one for each flower)  
     
   for row in range(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_):

col = \_\_\_\_\_

while all\_test\_labels[row][col] == 0:

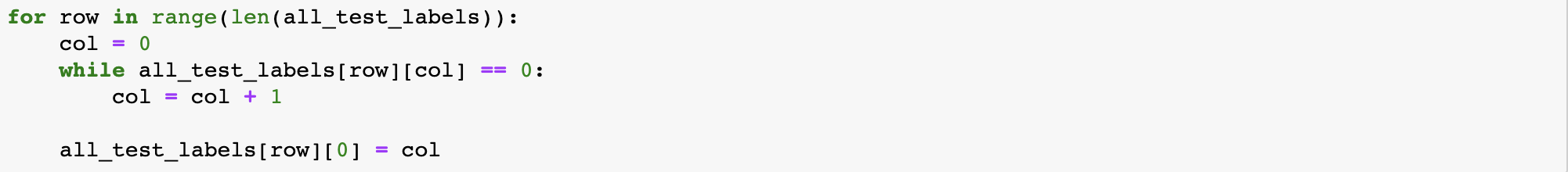
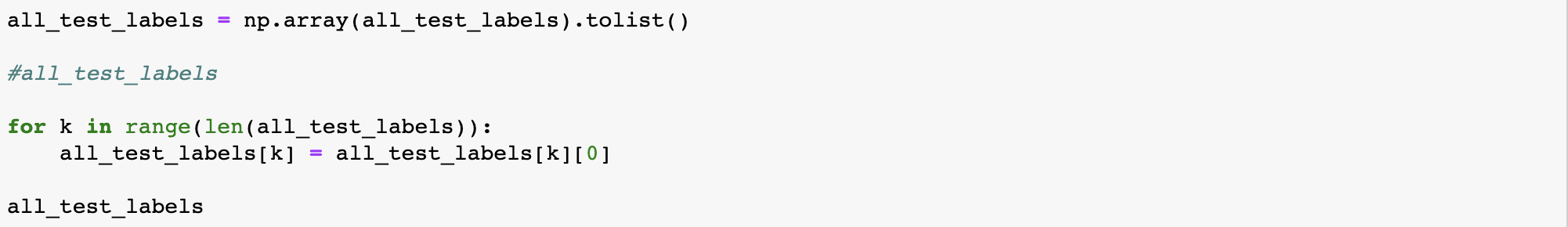
col = col + 1

all\_test\_labels[row][0] = col

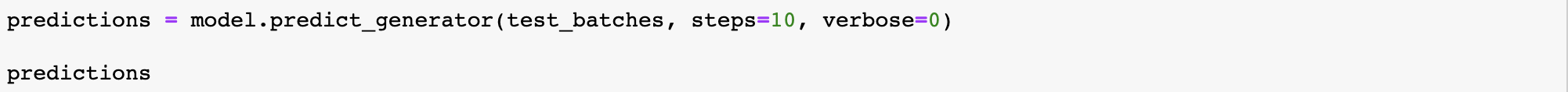
# Convert all\_test\_labels numpy array to a list   
all\_test\_labels = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

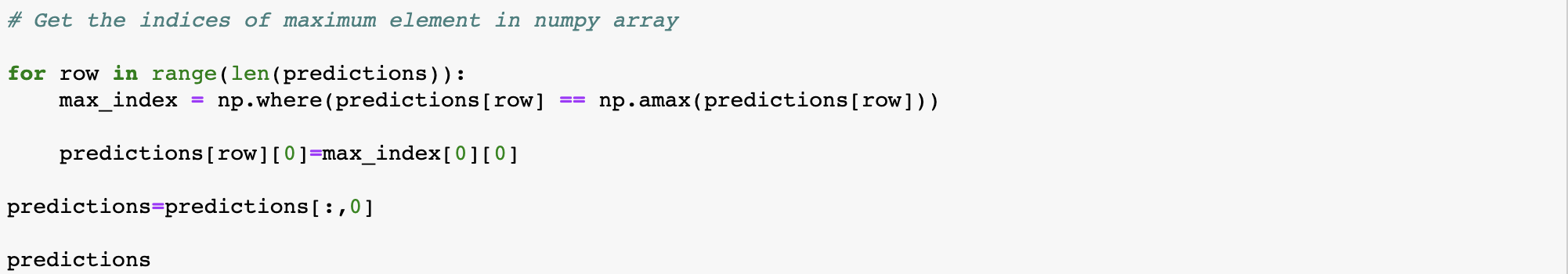
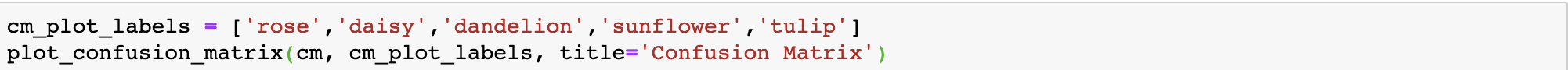
# loop through all the test labels   
for k in \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_:

all\_test\_label[k] = all\_test\_labels[k][0]

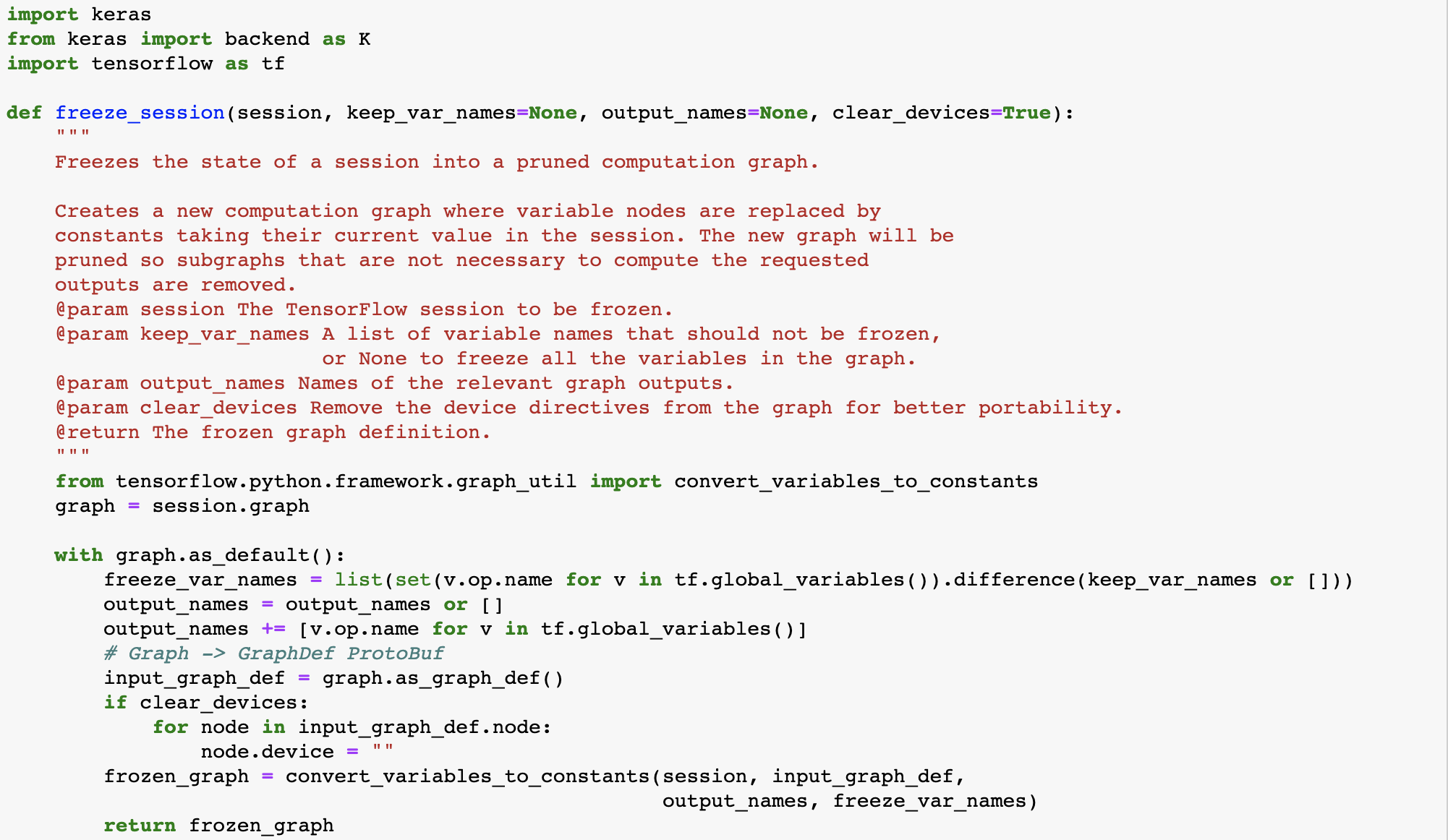
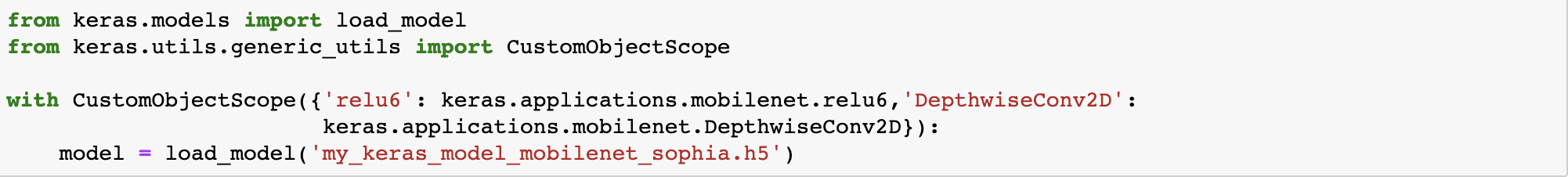
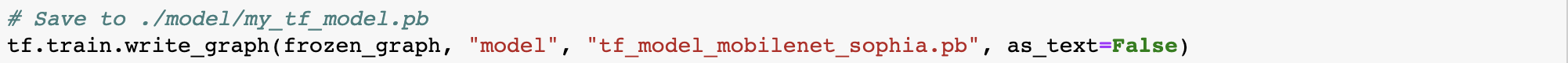
  


1. Run a prediction. Test batches are the collection of test images. Steps should equal the number of test images divided by the batch size.  
     
   predictions = model.predict\_generator(test\_batches, steps=10,

verbose=0)  
   


1. Get the index of the maximum element in the numpy array by traversing through the predictions made. This will give us the model’s prediction of the image. The high the value for a specific class is on an image, the more confident the model is that the prediction is true.   
   
2. Create a confusion matrix with this data. A confusion matrix is a plot of the test data true classifications by the test data predictions.   
   
3. Define a function that creates a visual confusion matrix that displays the accuracy of the trained model.  
   
4. Plot the confusion matrix that reveals the accuracy of the trained model  
   

### Converting To TensorFlow

1. Import listed libraries and define a function to freeze the weights of the graph   
   
2. Import listed libraries and load in the saved model  
   
3. Freeze the graph with previously defined session  
   
4. Save the newly converted TensorFlow model   
   

Challenge 2: Combining Flowers and Leaves (might take out)

### Individual Prediction

## Challenge 3: Model-Mobile App Integration