KERAS UTILITY METHODS FOR STREAMLINING TRAINING OF CNN

Done by : Rawan Al-Qahtani

Dr : Mejdal Al-Qahtani

Introduction

This code creates a convolutional neural net (CNN) classifier to classify multiple (5) classes of flowers. A proper data preprocessing, by sorting the folders of dataset then uniform the shape of all images and the color channels.5 layers was created for this model defining each convolution and max pooling then flatten the results to feed into a dense layer. A 128 neuron in the fully-connected layer. 5 output neurons for 5 classes with the softmax activation. The module is saved also the wights. Finely, a prediction with new image was made.

Data set previwe

Daisy

Data set previwe

Dandelion

Data set previwe

Rose

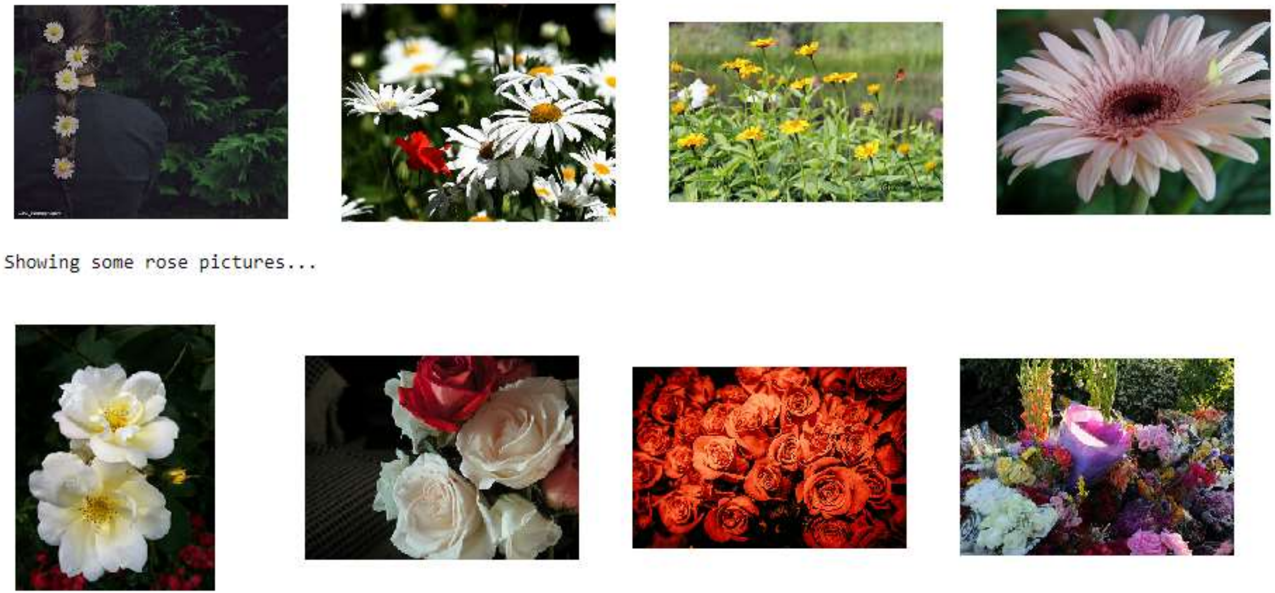
Data set previwe

Sunflower

Data set previwe

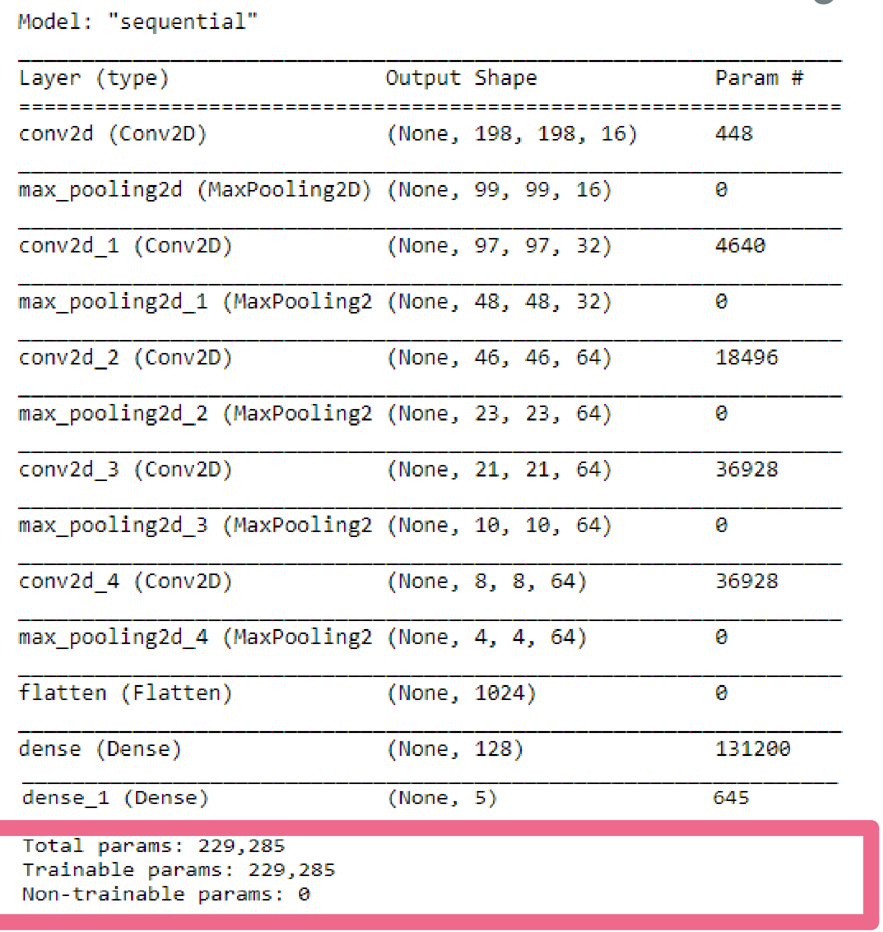
Tulip

Data set previwe in code



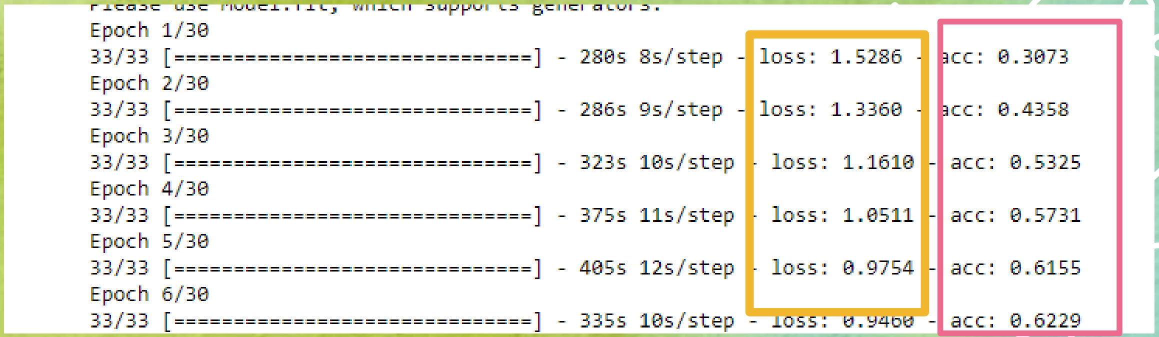
```
Showing some rose pictures...
```

Model Summury

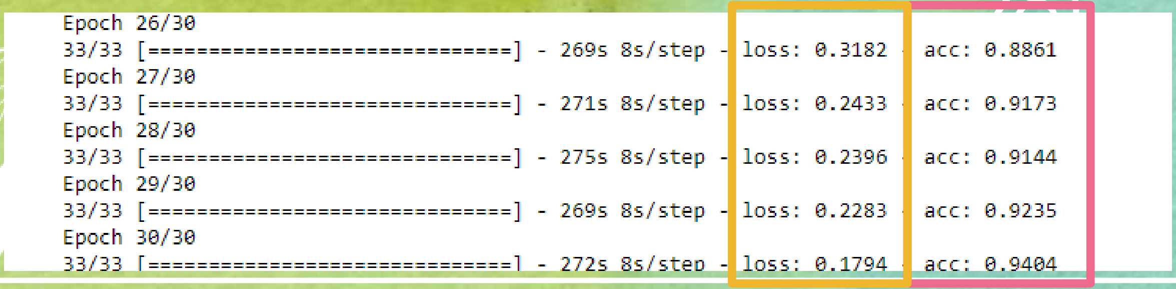


```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 198, 198, 16)      448       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 99, 99, 16)        0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 97, 97, 32)        4640      
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 48, 48, 32)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 46, 46, 64)        18496     
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 23, 23, 64)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 21, 21, 64)        36928     
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 10, 10, 64)        0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 8, 64)          36928     
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 4, 4, 64)          0         
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0         
_________________________________________________________________
dense (Dense)                (None, 128)               131200    
_________________________________________________________________
dense_1 (Dense)              (None, 5)                 645       
Total params: 229,285
Trainable params: 229,285
Non-trainable params: 0
```

Model train

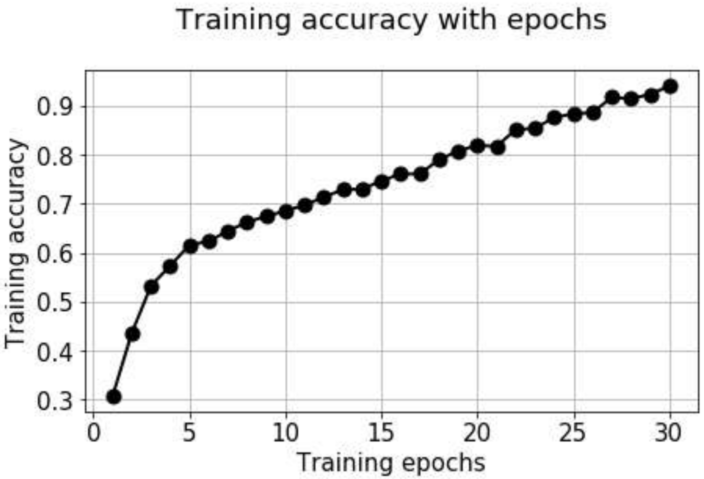


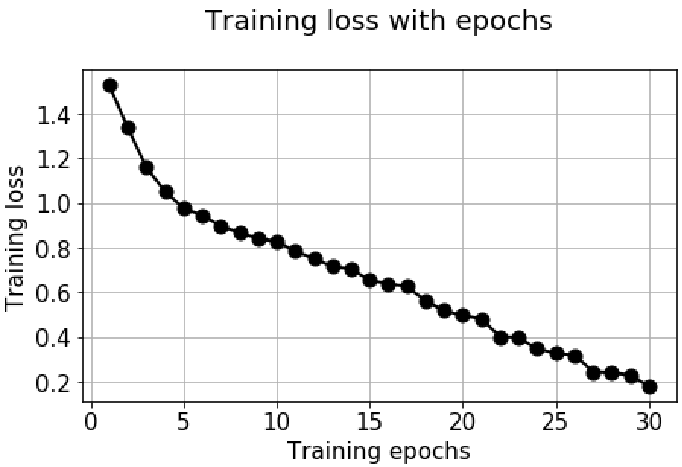
```
Please use Model.fit, which supports generators.
Epoch 1/30
33/33 [==============================] - 280s 8s/step - loss: 1.5286 - acc: 0.3073
Epoch 2/30
33/33 [==============================] - 286s 9s/step - loss: 1.3360 - acc: 0.4358
Epoch 3/30
33/33 [==============================] - 323s 10s/step - loss: 1.1610 - acc: 0.5325
Epoch 4/30
33/33 [==============================] - 375s 11s/step - loss: 1.0511 - acc: 0.5731
Epoch 5/30
33/33 [==============================] - 405s 12s/step - loss: 0.9754 - acc: 0.6155
Epoch 6/30
33/33 [==============================] - 335s 10s/step - loss: 0.9460 - acc: 0.6229
```



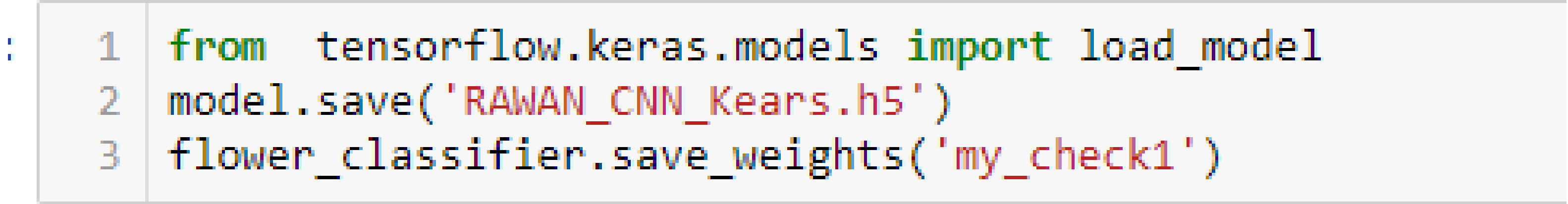
```
Epoch 26/30
33/33 [==============================] - 269s 8s/step - loss: 0.3182 - acc: 0.8861
Epoch 27/30
33/33 [==============================] - 271s 8s/step - loss: 0.2433 - acc: 0.9173
Epoch 28/30
33/33 [==============================] - 275s 8s/step - loss: 0.2396 - acc: 0.9144
Epoch 29/30
33/33 [==============================] - 269s 8s/step - loss: 0.2283 - acc: 0.9235
Epoch 30/30
33/33 [==============================] - 272s 8s/step - loss: 0.1794 - acc: 0.9404
```

Training accuracy & loss with epochs





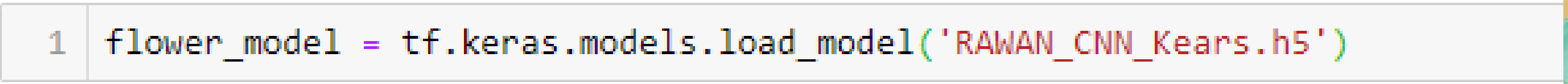
Saving the model & weights



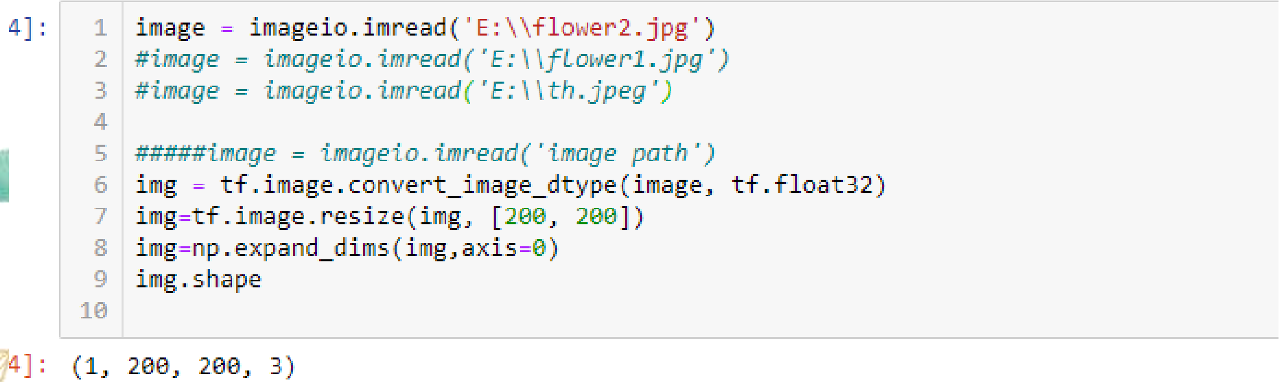
```
from  tensorflow.keras.models import load_model
model.save('RAWAN_CNN_Kears.h5')
flower_classifier.save_weights('my_check1')
```

Testing the model

Prediction



```
flower_model = tf.keras.models.load_model('RAWAN_CNN_Kears.h5')
```



```
4]: image = imageio.imread('E:\\flower2.jpg')
    #image = imageio.imread('E:\\flower1.jpg')
    #image = imageio.imread('E:\\th.jpeg')

    #####image = imageio.imread('image path')
    img = tf.image.convert_image_dtype(image, tf.float32)
    img=tf.image.resize(img, [200, 200])
    img=np.expand_dims(img,axis=0)
    img.shape

4]: (1, 200, 200, 3)
```

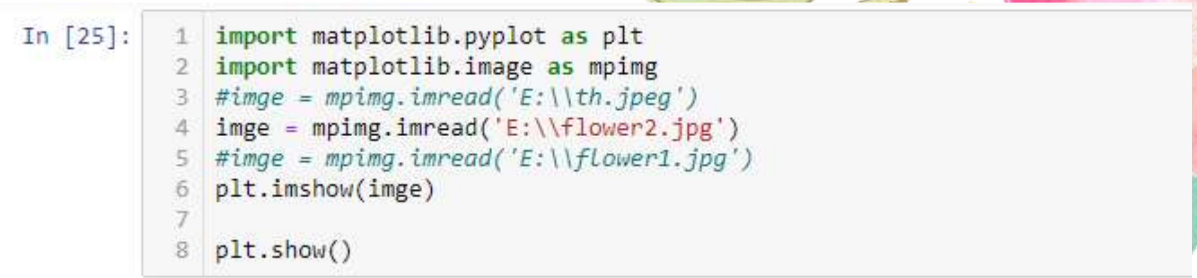
Daisy

Dandelion

Rose

Sunflower

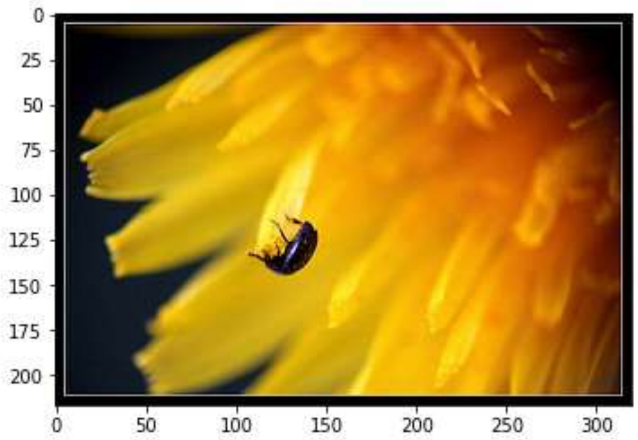
Tulip



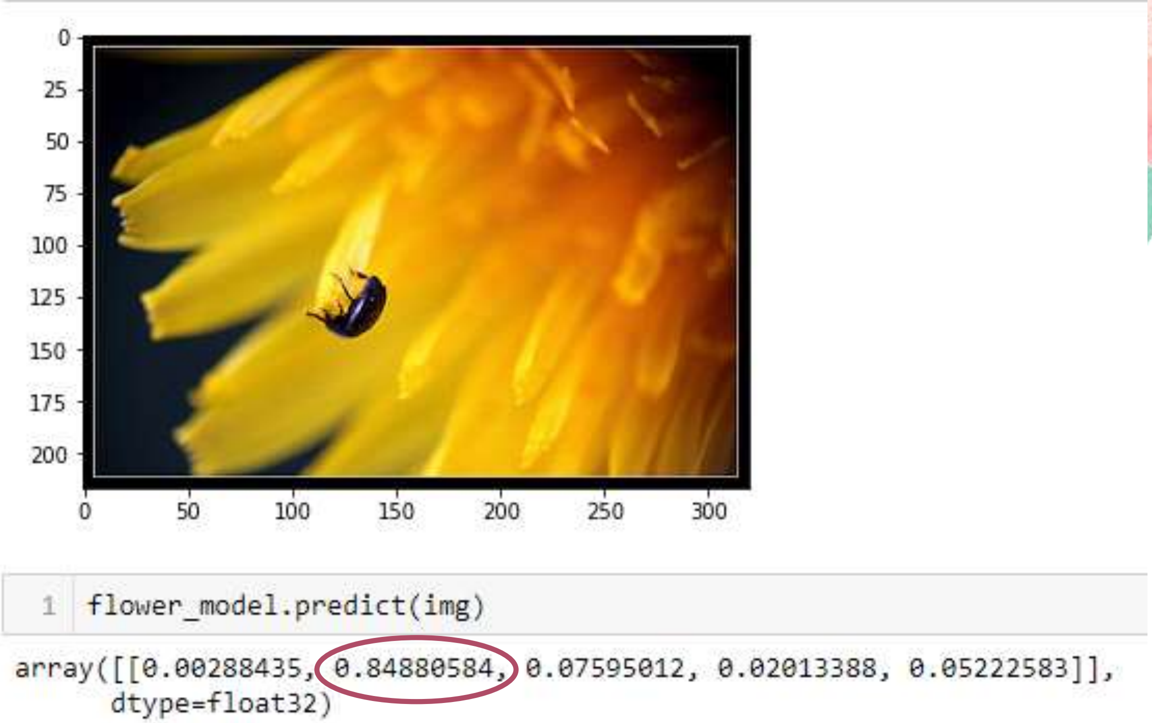
In [25]:

```python
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
#imge = mpimg.imread('E:\\th.jpeg')
imge = mpimg.imread('E:\\flower2.jpg')
#imge = mpimg.imread('E:\\flower1.jpg')
plt.imshow(imge)

plt.show()
```



Daisy
Dandelion
Rose
Sunflower
Tulip



```
flower_model.predict(img)
```

```
array([[0.00288435, 0.84880584, 0.07595012, 0.02013388, 0.05222583]],
      dtype=float32)
```

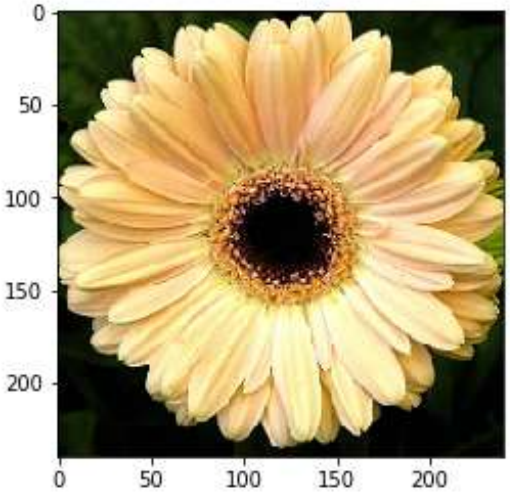
Daisy

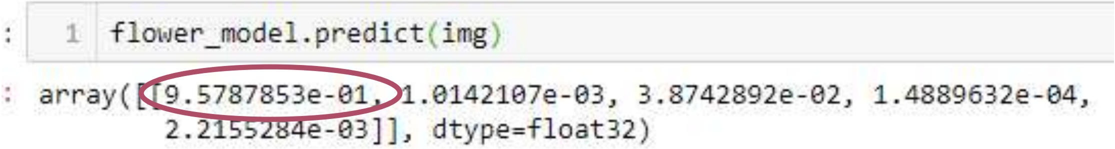
Dandelion

Rose

Sunflower

Tulip

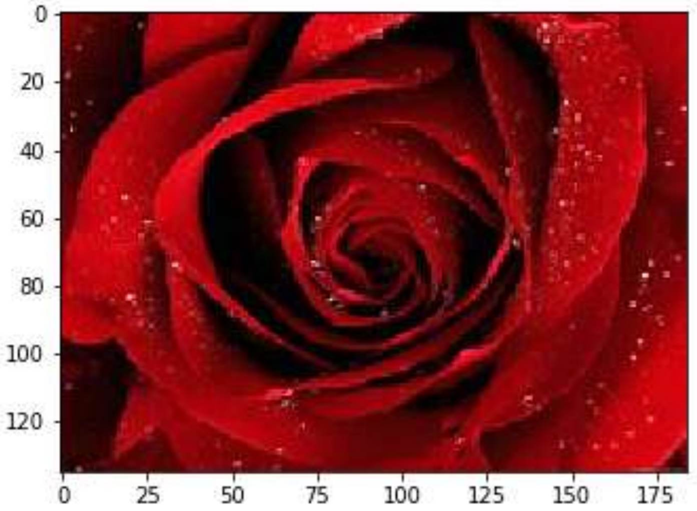


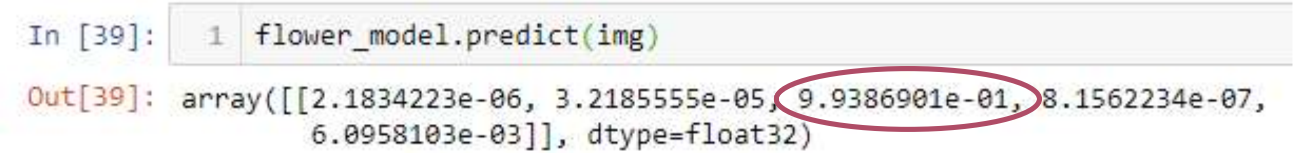


```
: flower_model.predict(img)

: array([[9.5787853e-01, 1.0142107e-03, 3.8742892e-02, 1.4889632e-04,
          2.2155284e-03]], dtype=float32)
```

Daisy
Dandelion
Rose
Sunflower
Tulip





```
In [39]:  flower_model.predict(img)

Out[39]: array([[2.1834223e-06, 3.2185555e-05, 9.9386901e-01, 8.1562234e-07,
                 6.0958103e-03]], dtype=float32)
```

Future work

- Develop new models for objects other than flowers.
- Develop models with better libraries.

Thanks