flower-recoginition

April 25, 2022

[1]:
```
# !pip install tqdm
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
from tqdm import tqdm
from datetime import datetime
%matplotlib inline
start_time = datetime.now()
```

[2]:
```
image_dir = '../input/flowers-recognition/flowers'
labels = ['daisy','dandelion','rose','sunflower','tulip']
nb = len(labels)
```

[3]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.layers import Dense, Flatten, GlobalAveragePooling2D,␣
↪Conv2D, MaxPooling2D,MaxPool2D,\
Input,Dropout,BatchNormalization
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ModelCheckpoint,␣
↪EarlyStopping,ReduceLROnPlateau
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import Adam
```

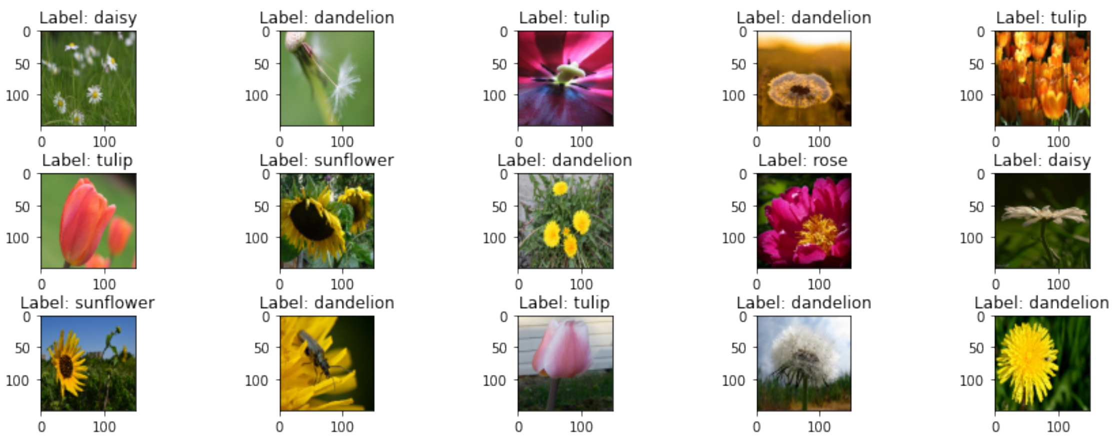
[4]:
```
#ref -> https://github.com/mrc03/Flower-Recognition-Kaggle-CNN-Keras/blob/
↪master/Flower_Recognition_VGG16(trans.%20learn).ipynb

shape = 150
input_shape = (shape,shape)
def get_XandY(train_dir,labels):
    dataset = []
    count = 0
    for label in labels:
        folder = os.path.join(train_dir,label)
        for image in tqdm(os.listdir(folder)):
```

```
            img=load_img(os.path.join(folder,image), target_size=input_shape)
            img=img_to_array(img)
            img=img/255.0
            dataset.append((img,count))
        print(">>> ",label)
        count+=1
    np.random.shuffle(dataset)
    X, y = zip(*dataset)

    return np.array(X),np.array(y)
```

[5]:
```
images,label = get_XandY(image_dir,labels)
```

```
100%|      | 764/764 [00:04<00:00, 182.44it/s]

>>>  daisy

100%|      | 1052/1052 [00:06<00:00, 170.41it/s]

>>>  dandelion

100%|      | 784/784 [00:04<00:00, 179.60it/s]

>>>  rose

100%|      | 733/733 [00:04<00:00, 153.60it/s]

>>>  sunflower

100%|      | 984/984 [00:06<00:00, 163.41it/s]

>>>  tulip
```

[6]:
```
plt.figure(figsize = (15 , 9))
n = 0
for i in range(15):
    n+=1
    plt.subplot(5 , 5, n)
    plt.subplots_adjust(hspace = 0.5 , wspace = 0.3)
    plt.imshow(images[i])
    plt.title(f'Label: {labels[label[i]]}')
```



[7]:
```
np.unique(label,return_counts=True)
```

[7]:
```
(array([0, 1, 2, 3, 4]), array([ 764, 1052,  784,  733,  984]))
```

[8]:
```
label = to_categorical(label,5)
```

[9]:
```
from sklearn.model_selection import train_test_split

xtrain,xtest,ytrain,ytest = train_test_split(images,label,stratify=label,random_state=42,test_size=0.25)

print(f"Train length:{len(xtrain)} \n Test length: {len(xtest)}")
```

```
Train length:3237
 Test length: 1080
```

[10]:
```
np.random.seed(42)
tf.random.set_seed(42)
np.random.seed()
```

[11]:
```
datagen = ImageDataGenerator(
        featurewise_center=False,  # set input mean to 0 over the dataset
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by std of the dataset
        samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        rotation_range=10,  # randomly rotate images in the range (degrees, 0 to 180)
        zoom_range = 0.1, # Randomly zoom image
        width_shift_range=0.2,  # randomly shift images horizontally (fraction of total width)
```

```
        height_shift_range=0.2,  # randomly shift images vertically (fraction␣
 ↪of total height)
        horizontal_flip=True,  # randomly flip images
        vertical_flip=False
)  # randomly flip images


datagen.fit(xtrain)
```

[12]:
```
base_model = tf.keras.applications.VGG16(include_top=False,␣
 ↪weights='imagenet',input_shape=(shape,shape,3), pooling='avg')
```

```
2022-04-25 03:57:36.886312: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-04-25 03:57:36.998234: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-04-25 03:57:36.999073: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-04-25 03:57:37.000360: I tensorflow/core/platform/cpu_feature_guard.cc:142]
This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical
operations:  AVX2 AVX512F FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
2022-04-25 03:57:37.000674: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-04-25 03:57:37.001380: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-04-25 03:57:37.001995: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-04-25 03:57:38.817591: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
```

```
2022-04-25 03:57:38.818457: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-04-25 03:57:38.819164: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-04-25 03:57:38.820383: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1510] Created device
/job:localhost/replica:0/task:0/device:GPU:0 with 15403 MB memory:  -> device:
0, name: Tesla P100-PCIE-16GB, pci bus id: 0000:00:04.0, compute capability: 6.0
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 0s 0us/step
58900480/58889256 [==============================] - 0s 0us/step
```

[13]:
```
base_model.summary()
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 150, 150, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 150, 150, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 150, 150, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 75, 75, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 75, 75, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 75, 75, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 37, 37, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 37, 37, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 37, 37, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 37, 37, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 18, 18, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 18, 18, 512)       1180160
_________________________________________________________________
```

```
block4_conv2 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 9, 9, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 4, 4, 512)         0
_________________________________________________________________
global_average_pooling2d (Gl (None, 512)               0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

[14]:

```
epochs=50
batch_size=128
red_lr=ReduceLROnPlateau(monitor='val_acc', factor=0.1, epsilon=0.0001,␣
 ↪patience=2, verbose=1)
filepath= "best_model.h5"
ck = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1,␣
 ↪save_best_only=True, mode='max', save_weights_only=False)

cb =[
    ck

]
```

[15]:

```
%%time
model=Sequential()
model.add(base_model)

model.add(Dense(256,activation='relu'))
model.add(BatchNormalization())
model.add(Dense(5,activation='softmax'))


for layer in base_model.layers:
    layer.trainable=True
```

```
model.
 ↪compile(optimizer=Adam(learning_rate=1e-4),loss='categorical_crossentropy',metrics=['accura
print("No fo Layers: ",len(model.layers))
History = model.fit(datagen.flow(xtrain,ytrain,␣
 ↪batch_size=batch_size),callbacks = cb,
                              epochs = 50, validation_data = (xtest,ytest),
                              verbose = 1, steps_per_epoch=xtrain.shape[0] //␣
 ↪batch_size)
```

```
No fo Layers:  4

2022-04-25 03:57:41.468580: I
tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR
Optimization Passes are enabled (registered 2)

Epoch 1/50

2022-04-25 03:57:43.895753: I tensorflow/stream_executor/cuda/cuda_dnn.cc:369]
Loaded cuDNN version 8005

25/25 [==============================] - 40s 1s/step - loss: 0.8586 - accuracy:
0.6777 - val_loss: 0.9232 - val_accuracy: 0.6935

Epoch 00001: val_accuracy improved from -inf to 0.69352, saving model to
best_model.h5
Epoch 2/50
25/25 [==============================] - 20s 791ms/step - loss: 0.4359 -
accuracy: 0.8437 - val_loss: 0.6788 - val_accuracy: 0.7907

Epoch 00002: val_accuracy improved from 0.69352 to 0.79074, saving model to
best_model.h5
Epoch 3/50
25/25 [==============================] - 20s 792ms/step - loss: 0.3536 -
accuracy: 0.8742 - val_loss: 0.7133 - val_accuracy: 0.7407

Epoch 00003: val_accuracy did not improve from 0.79074
Epoch 4/50
25/25 [==============================] - 20s 794ms/step - loss: 0.3043 -
accuracy: 0.8874 - val_loss: 0.4707 - val_accuracy: 0.8685

Epoch 00004: val_accuracy improved from 0.79074 to 0.86852, saving model to
best_model.h5
Epoch 5/50
25/25 [==============================] - 20s 801ms/step - loss: 0.2253 -
accuracy: 0.9161 - val_loss: 0.4596 - val_accuracy: 0.8500

Epoch 00005: val_accuracy did not improve from 0.86852
Epoch 6/50
25/25 [==============================] - 19s 772ms/step - loss: 0.1875 -
```

```
accuracy: 0.9337 - val_loss: 0.4617 - val_accuracy: 0.8472

Epoch 00006: val_accuracy did not improve from 0.86852
Epoch 7/50
25/25 [==============================] - 19s 760ms/step - loss: 0.1763 -
accuracy: 0.9353 - val_loss: 0.4536 - val_accuracy: 0.8343

Epoch 00007: val_accuracy did not improve from 0.86852
Epoch 8/50
25/25 [==============================] - 20s 787ms/step - loss: 0.1508 -
accuracy: 0.9485 - val_loss: 0.4908 - val_accuracy: 0.8231

Epoch 00008: val_accuracy did not improve from 0.86852
Epoch 9/50
25/25 [==============================] - 20s 783ms/step - loss: 0.1362 -
accuracy: 0.9543 - val_loss: 0.4171 - val_accuracy: 0.8546

Epoch 00009: val_accuracy did not improve from 0.86852
Epoch 10/50
25/25 [==============================] - 20s 793ms/step - loss: 0.1199 -
accuracy: 0.9579 - val_loss: 0.4515 - val_accuracy: 0.8583

Epoch 00010: val_accuracy did not improve from 0.86852
Epoch 11/50
25/25 [==============================] - 20s 790ms/step - loss: 0.0847 -
accuracy: 0.9698 - val_loss: 0.4608 - val_accuracy: 0.8389

Epoch 00011: val_accuracy did not improve from 0.86852
Epoch 12/50
25/25 [==============================] - 20s 778ms/step - loss: 0.1058 -
accuracy: 0.9617 - val_loss: 0.3500 - val_accuracy: 0.8889

Epoch 00012: val_accuracy improved from 0.86852 to 0.88889, saving model to
best_model.h5
Epoch 13/50
25/25 [==============================] - 20s 800ms/step - loss: 0.0696 -
accuracy: 0.9775 - val_loss: 0.4581 - val_accuracy: 0.8546

Epoch 00013: val_accuracy did not improve from 0.88889
Epoch 14/50
25/25 [==============================] - 19s 762ms/step - loss: 0.0476 -
accuracy: 0.9868 - val_loss: 0.3525 - val_accuracy: 0.8861

Epoch 00014: val_accuracy did not improve from 0.88889
Epoch 15/50
25/25 [==============================] - 20s 788ms/step - loss: 0.0545 -
accuracy: 0.9836 - val_loss: 0.4217 - val_accuracy: 0.8824
```

```
Epoch 00015: val_accuracy did not improve from 0.88889
Epoch 16/50
25/25 [==============================] - 21s 817ms/step - loss: 0.0475 -
accuracy: 0.9842 - val_loss: 0.4104 - val_accuracy: 0.8889

Epoch 00016: val_accuracy did not improve from 0.88889
Epoch 17/50
25/25 [==============================] - 19s 768ms/step - loss: 0.0522 -
accuracy: 0.9813 - val_loss: 0.4500 - val_accuracy: 0.8750

Epoch 00017: val_accuracy did not improve from 0.88889
Epoch 18/50
25/25 [==============================] - 20s 796ms/step - loss: 0.0958 -
accuracy: 0.9685 - val_loss: 0.5636 - val_accuracy: 0.8546

Epoch 00018: val_accuracy did not improve from 0.88889
Epoch 19/50
25/25 [==============================] - 19s 771ms/step - loss: 0.0829 -
accuracy: 0.9736 - val_loss: 0.8642 - val_accuracy: 0.8074

Epoch 00019: val_accuracy did not improve from 0.88889
Epoch 20/50
25/25 [==============================] - 20s 772ms/step - loss: 0.0467 -
accuracy: 0.9852 - val_loss: 0.3932 - val_accuracy: 0.8917

Epoch 00020: val_accuracy improved from 0.88889 to 0.89167, saving model to
best_model.h5
Epoch 21/50
25/25 [==============================] - 20s 798ms/step - loss: 0.0340 -
accuracy: 0.9913 - val_loss: 0.4859 - val_accuracy: 0.8750

Epoch 00021: val_accuracy did not improve from 0.89167
Epoch 22/50
25/25 [==============================] - 20s 789ms/step - loss: 0.0238 -
accuracy: 0.9949 - val_loss: 0.4619 - val_accuracy: 0.8926

Epoch 00022: val_accuracy improved from 0.89167 to 0.89259, saving model to
best_model.h5
Epoch 23/50
25/25 [==============================] - 20s 797ms/step - loss: 0.0367 -
accuracy: 0.9875 - val_loss: 0.4375 - val_accuracy: 0.8907

Epoch 00023: val_accuracy did not improve from 0.89259
Epoch 24/50
25/25 [==============================] - 20s 787ms/step - loss: 0.0447 -
accuracy: 0.9846 - val_loss: 0.6744 - val_accuracy: 0.8648

Epoch 00024: val_accuracy did not improve from 0.89259
```

```
Epoch 25/50
25/25 [==============================] - 20s 792ms/step - loss: 0.0489 -
accuracy: 0.9846 - val_loss: 0.4907 - val_accuracy: 0.8917

Epoch 00025: val_accuracy did not improve from 0.89259
Epoch 26/50
25/25 [==============================] - 20s 793ms/step - loss: 0.0371 -
accuracy: 0.9865 - val_loss: 0.4617 - val_accuracy: 0.8861

Epoch 00026: val_accuracy did not improve from 0.89259
Epoch 27/50
25/25 [==============================] - 19s 771ms/step - loss: 0.0320 -
accuracy: 0.9881 - val_loss: 0.5112 - val_accuracy: 0.8843

Epoch 00027: val_accuracy did not improve from 0.89259
Epoch 28/50
25/25 [==============================] - 20s 788ms/step - loss: 0.0187 -
accuracy: 0.9949 - val_loss: 0.4101 - val_accuracy: 0.9056

Epoch 00028: val_accuracy improved from 0.89259 to 0.90556, saving model to
best_model.h5
Epoch 29/50
25/25 [==============================] - 20s 814ms/step - loss: 0.0271 -
accuracy: 0.9916 - val_loss: 0.6405 - val_accuracy: 0.8509

Epoch 00029: val_accuracy did not improve from 0.90556
Epoch 30/50
25/25 [==============================] - 19s 770ms/step - loss: 0.0270 -
accuracy: 0.9913 - val_loss: 0.6000 - val_accuracy: 0.8620

Epoch 00030: val_accuracy did not improve from 0.90556
Epoch 31/50
25/25 [==============================] - 20s 801ms/step - loss: 0.0155 -
accuracy: 0.9955 - val_loss: 0.4184 - val_accuracy: 0.9074

Epoch 00031: val_accuracy improved from 0.90556 to 0.90741, saving model to
best_model.h5
Epoch 32/50
25/25 [==============================] - 20s 795ms/step - loss: 0.0279 -
accuracy: 0.9916 - val_loss: 0.9371 - val_accuracy: 0.8361

Epoch 00032: val_accuracy did not improve from 0.90741
Epoch 33/50
25/25 [==============================] - 19s 745ms/step - loss: 0.0169 -
accuracy: 0.9965 - val_loss: 0.5235 - val_accuracy: 0.9000

Epoch 00033: val_accuracy did not improve from 0.90741
Epoch 34/50
```

```
25/25 [==============================] - 21s 812ms/step - loss: 0.0138 -
accuracy: 0.9971 - val_loss: 0.6042 - val_accuracy: 0.8750

Epoch 00034: val_accuracy did not improve from 0.90741
Epoch 35/50
25/25 [==============================] - 20s 773ms/step - loss: 0.0140 -
accuracy: 0.9961 - val_loss: 0.6587 - val_accuracy: 0.8769

Epoch 00035: val_accuracy did not improve from 0.90741
Epoch 36/50
25/25 [==============================] - 20s 808ms/step - loss: 0.0373 -
accuracy: 0.9881 - val_loss: 0.7761 - val_accuracy: 0.8500

Epoch 00036: val_accuracy did not improve from 0.90741
Epoch 37/50
25/25 [==============================] - 19s 764ms/step - loss: 0.0466 -
accuracy: 0.9855 - val_loss: 0.6049 - val_accuracy: 0.8778

Epoch 00037: val_accuracy did not improve from 0.90741
Epoch 38/50
25/25 [==============================] - 20s 795ms/step - loss: 0.0529 -
accuracy: 0.9820 - val_loss: 0.6200 - val_accuracy: 0.8787

Epoch 00038: val_accuracy did not improve from 0.90741
Epoch 39/50
25/25 [==============================] - 20s 794ms/step - loss: 0.0344 -
accuracy: 0.9907 - val_loss: 0.5843 - val_accuracy: 0.8778

Epoch 00039: val_accuracy did not improve from 0.90741
Epoch 40/50
25/25 [==============================] - 19s 762ms/step - loss: 0.0228 -
accuracy: 0.9942 - val_loss: 0.6398 - val_accuracy: 0.8778

Epoch 00040: val_accuracy did not improve from 0.90741
Epoch 41/50
25/25 [==============================] - 21s 820ms/step - loss: 0.0205 -
accuracy: 0.9944 - val_loss: 0.6006 - val_accuracy: 0.8750

Epoch 00041: val_accuracy did not improve from 0.90741
Epoch 42/50
25/25 [==============================] - 19s 770ms/step - loss: 0.0271 -
accuracy: 0.9887 - val_loss: 0.4629 - val_accuracy: 0.8907

Epoch 00042: val_accuracy did not improve from 0.90741
Epoch 43/50
25/25 [==============================] - 19s 786ms/step - loss: 0.0308 -
accuracy: 0.9900 - val_loss: 0.6859 - val_accuracy: 0.8611
```

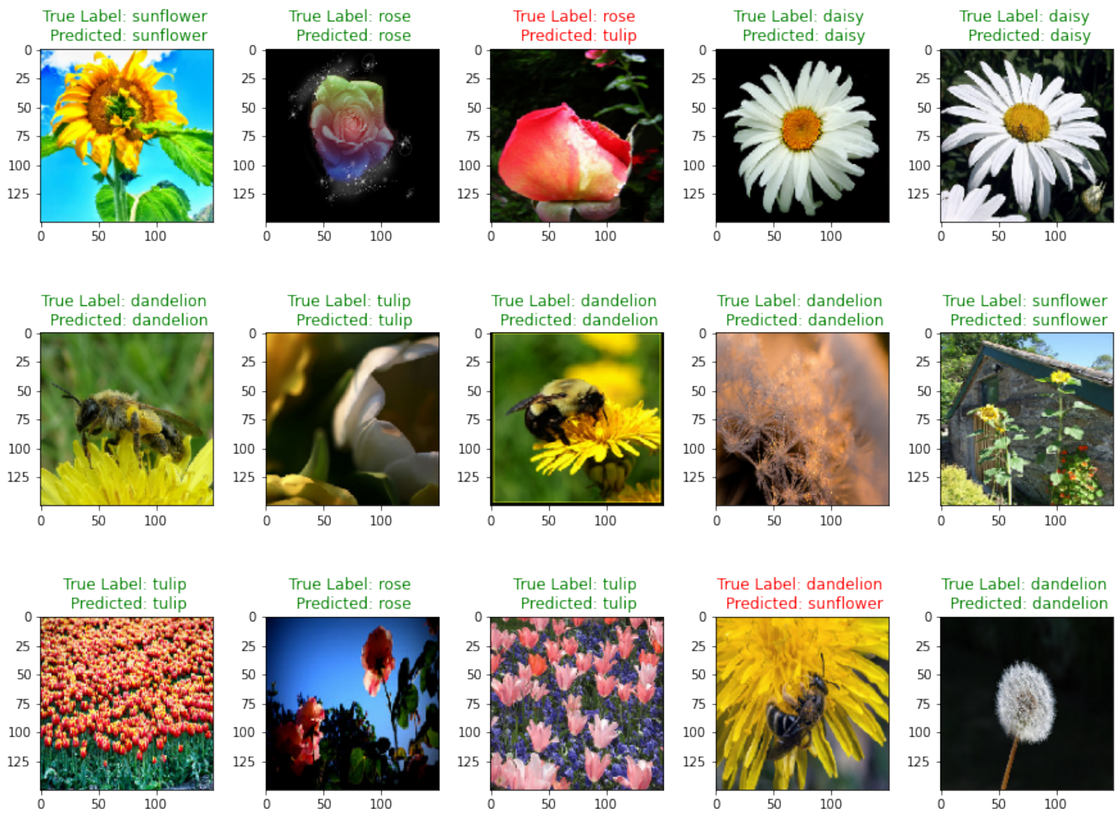
```
Epoch 00043: val_accuracy did not improve from 0.90741
Epoch 44/50
25/25 [==============================] - 20s 800ms/step - loss: 0.0397 -
accuracy: 0.9884 - val_loss: 0.4536 - val_accuracy: 0.9019

Epoch 00044: val_accuracy did not improve from 0.90741
Epoch 45/50
25/25 [==============================] - 19s 772ms/step - loss: 0.0236 -
accuracy: 0.9920 - val_loss: 0.6217 - val_accuracy: 0.8713

Epoch 00045: val_accuracy did not improve from 0.90741
Epoch 46/50
25/25 [==============================] - 20s 794ms/step - loss: 0.0164 -
accuracy: 0.9952 - val_loss: 0.4825 - val_accuracy: 0.8870

Epoch 00046: val_accuracy did not improve from 0.90741
Epoch 47/50
25/25 [==============================] - 20s 805ms/step - loss: 0.0146 -
accuracy: 0.9955 - val_loss: 0.4631 - val_accuracy: 0.9009

Epoch 00047: val_accuracy did not improve from 0.90741
Epoch 48/50
25/25 [==============================] - 19s 769ms/step - loss: 0.0127 -
accuracy: 0.9961 - val_loss: 0.5520 - val_accuracy: 0.8870

Epoch 00048: val_accuracy did not improve from 0.90741
Epoch 49/50
25/25 [==============================] - 20s 795ms/step - loss: 0.0154 -
accuracy: 0.9945 - val_loss: 0.7223 - val_accuracy: 0.8667

Epoch 00049: val_accuracy did not improve from 0.90741
Epoch 50/50
25/25 [==============================] - 20s 780ms/step - loss: 0.0136 -
accuracy: 0.9952 - val_loss: 0.6283 - val_accuracy: 0.8759

Epoch 00050: val_accuracy did not improve from 0.90741
CPU times: user 21min 19s, sys: 1min, total: 22min 20s
Wall time: 18min 15s
```

[16]:
```
# model.save("ACCU_90_shape_150.h5")
```

[17]:
```
transfer_learning_best_model =  tf.keras.models.load_model('best_model.h5')

from sklearn.metrics import classification_report
```

```python
print(classification_report(np.argmax(ytest,axis = 1),np.
 ↪argmax(transfer_learning_best_model.predict(xtest),axis = 1),target_names =␣
 ↪labels))


pred = transfer_learning_best_model.predict(xtest)
pred = np.argmax(pred,axis = 1)
ytest = np.argmax(ytest,axis = 1)
plt.figure(figsize = (15 , 19))
n = 0
for i in range(15):
    if pred[i]==ytest[i]:

        n+=1
        plt.subplot(5 , 5, n)
        plt.subplots_adjust(hspace = 0.5 , wspace = 0.3)
        plt.title(f'True Label: {labels[ytest[i]]} \n Predicted:␣
 ↪{labels[pred[i]]}',color = 'green')

        plt.imshow(xtest[i])
#         plt.xlabel(f"",color="green")
    else:
        n+=1
        plt.subplot(5 , 5, n)
        plt.subplots_adjust(hspace = 0.5 , wspace = 0.3)
        plt.title(f'True Label: {labels[ytest[i]]} \n Predicted:␣
 ↪{labels[pred[i]]}',color = 'red')

        plt.imshow(xtest[i])
```

```
              precision    recall  f1-score   support

       daisy       0.89      0.92      0.91       191
   dandelion       0.99      0.90      0.94       263
        rose       0.91      0.85      0.88       196
   sunflower       0.88      0.96      0.92       184
       tulip       0.86      0.91      0.88       246

    accuracy                           0.91      1080
   macro avg       0.91      0.91      0.91      1080
weighted avg       0.91      0.91      0.91      1080
```



[18]:
```
np.save("xtrain.npy",xtrain)
np.save("xtest.npy",xtest)
np.save("ytrain.npy",ytrain)
np.save("ytest.npy",ytest)
```

[19]:
```
np.load("xtrain.npy").shape
```

[19]: (3237, 150, 150, 3)

[20]:
```
print('Time elapsed (hh:mm:ss.ms) {}'.format(datetime.now() - start_time))
```

```
Time elapsed (hh:mm:ss.ms) 0:19:07.732549
```