UDACITY

Logout

‹ Return to Classroom

Create Your Own Image Classifier - TensorFlow

REVIEW | CODE REVIEW 1 | HISTORY

Meets Specifications

Hello There,

Amazing work in this submission! I was truly happy to see the results of all your learnings come to play in this project.
Your technical skills are clearly displayed here through the logic used in writing your code.
Congratulations on completing the project, your efforts have paid off 🎉

I appreciate the fact that all aspects of your part-one of the project have been implemented correctly. Your Sanity check produces the correct flower names in the y-axis labels of the plot, and your predict.py script runs without errors. Kudos!! 👏🏻

You could Improve your model by incorporating all or any of these steps below

- Data Augmentation
- Adding Dropout
- Build a neural network model with batch normalization

To further improve or learn more about image classification, I would recommend you to read the following articles:

- Deep Convolutional Neural Network for Image Classification on CUDA Platform
- Understanding Learning Rates and How It Improves Performance in Deep Learning
- Deep Learning for Image Recognition: why it's challenging, where we've been, and what's next

.
All the best with your next project, and have a good one

Files Submitted

The submission includes all required files. (Model checkpoints not required.)

You submitted all the required files needed to run your project.

Part 1 - Development Notebook

All the necessary packages and modules are imported at the beginning of the notebook.

You have followed the recommended good practice by importing all the packages at the beginning of the notebook.

The Oxford Flowers 102 dataset is loaded using TensorFlow Datasets.

You correctly loaded the correct dataset 👍🏻

You could also load the data if it were available in your local directory to TensorFlow as explained in the links below

- https://towardsdatascience.com/introduction-to-keras-part-one-data-loading-43b9c015e27c
- https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing

The dataset is divided into a training set, a validation set, and a test set.

The number of examples in each set and the number classes in the dataset are extracted from the dataset info.

You've done well to use some of the built-in information for Tensorflow Datasets to extract this data.

Rate this review

START

- To see the available datasets in TensorFlow, you should refer to TensorFlow Datasets

In [10]:
```
# TODO: Get the number of examples in each set from the dataset info.
total_examples = dataset_info.splits['train'].num_examples + dataset_info.splits['test'].num_examples +dataset_info.splits['validation'].num_examples

num_training_examples = dataset_info.splits['train'].num_examples
lentraining = len(training_set)
num_validation_examples = dataset_info.splits['validation'].num_examples
lenvalidation = len(val_set)
num_test_examples = dataset_info.splits['test'].num_examples
lentest = len(test_set)

print('There are {:} images according dataset info in training set and length of array is {:}'.format(num_training_examples,lentraining))
print('There are {:} images according dataset info in validation set and length of array is {:}'.format(num_validation_examples,lenvalidation))
print('There are {:} images according dataset info in test set and length of array is {:}'.format(num_test_examples,lentest))
```

```
There are 1020 images according dataset info in training set and length of array is 1020
There are 1020 images according dataset info in validation set and length of array is 1020
There are 6149 images according dataset info in test set and length of array is 6149
```

The shape of the first 3 images in the training set is printed using a `for` loop and the `take()` method.

Great job here! It's interesting to note here we have images of various sizes.

In [12]:
```
# TODO: Print the shape and corresponding label of 3 images in the training set.
for image, label in training_set.take(5):
    print('The images in the training set have:\n\u2022 dtype:', image.dtype,
          '\n\u2022 shape:', image.shape ,
          '\n\u2022 label:', label.numpy())

#          '\n\u2022 name : ' , myclasses[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
The images in the training set have:
• dtype: <dtype: 'uint8'>
• shape: (500, 505, 3)
• label: 51
The images in the training set have:
• dtype: <dtype: 'uint8'>
• shape: (500, 672, 3)
• label: 48
```

The first image from the training set is plotted with the title of the plot corresponding to the image label.

In [20]:
```
# TODO: Plot 1 image from the training set. Set the title
# of the plot to the corresponding image label.

for image, label in training_set.take(1):
    images = image.numpy()
#     images = image.numpy().squeeze()
    labels = label.numpy()

plt.figure(figsize=(10,10))
plt.imshow(images)
plt.title(''.join(['label nr: ',str(labels)]), color='blue')
plt.axis('on')
```

Out[20]: (-0.5, 666.5, 499.5, -0.5)

The first image from the training set is plotted with the title of the plot corresponding to the class name using label mapping from the JSON file.

Great job mapping back to the flower name from the JSON file.

Rate this review

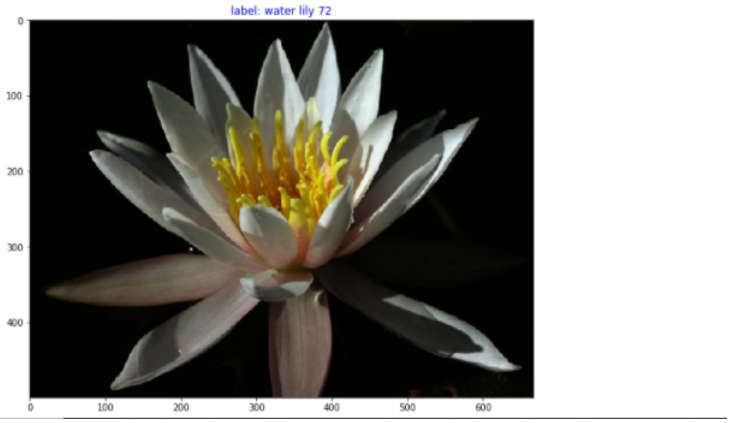
START

In [24]:
```
# TODO: Plot 1 image from the training set. Set the title
# of the plot to the corresponding class name.

for image, label in training_set.take(1):
    images = image.numpy()
#     images = image.numpy().squeeze()
    labels = label.numpy()

plt.figure(figsize=(10,10))
plt.imshow(images)
plt.title(''.join(['label: ',class_names[str(labels+1)],' ',str(labels)]), color='blue')
plt.axis('on')
```

Out[24]: (-0.5, 666.5, 499.5, -0.5)



The training, validation, and testing data is appropriately resized and normalized.

You correctly used the defined normalize function to appropriately resize the images.

A pipeline for each set is constructed with the necessary transformations.

Good job creating a pipeline for all the sets. You explored the different functions while creating the pipelines.

This is the documentation for these functions

The pipeline for each set should return batches of images.

The pre-trained network, MobileNet, is loaded using TensorFlow Hub and its parameters are frozen.

Great job loading the pre-trained MobileNet using TensorFlow hub, and freezing the parameters.

You could also read more about other models like VGG16 and ResNet50

A new neural network is created using transfer learning. The number of neurons in the output layer should correspond to the number of classes of the dataset.

You've done well to create a new neural network with transfer learning by using the previous model as the first part of a new Sequential model.

You could read more about transfer learning from this medium post

The model is configured for training using the `compile` method with appropriate parameters. The model is trained using the `fit` method and incorporating the validation set.

You correctly configured the model using the compile method with appropriate parameters and trained using the fit method and also incorporated the validation set.

The loss and accuracy values achieved during training for the training and validation set are plotted using the `history` dictionary return by the `fit` method.

> You correctly plotted the charts using history and fit methods

Rate this review

START

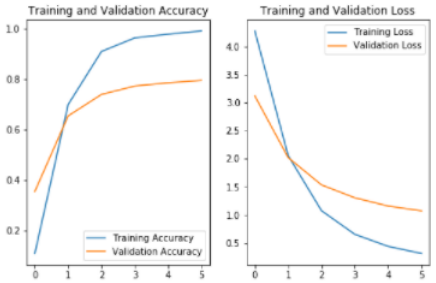
In [44]:
```
training_accuracy = history.history['accuracy']
validation_accuracy = history.history['val_accuracy']

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

epochs_range=range(EPOCHS)

plt.figure(figsize=(8, 5))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, training_accuracy, label='Training Accuracy')
plt.plot(epochs_range, validation_accuracy, label='Validation Accuracy')
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()
```



> These charts are a useful tool to give you first-hand information on whether your model is overfitting or underfitting as the case may be.
>
> - Overfitting is easy to diagnose with the accuracy visualizations you have available. If "Accuracy" (measured against the training set) is very good and "Validation Accuracy" (measured against a validation set) is not as good, then your model is overfitting.
> - Underfitting is a bit harder to diagnose. If Accuracy and Validation Accuracy are similar but are both poor, then you may be underfitting.

The network's accuracy is measured on the test data.

The trained model is saved as a Keras model (i.e. saved as an HDF5 file with extension `.h5`).

You correctly saved the model.

You could also save the model while training it by using ModelCheckpoint callback

The saved Keras model is successfully loaded.

The `process_image` function successfully normalizes and resizes the input image. The image returned by the `process_image` function should be a NumPy array with shape `(224, 224, 3)`.

> Great job creating a generalized function here!

The `predict` function successfully takes the path to an image and a saved model, and then returns the top K most probably classes for that image.

> The predict function is implemented correctly and returns the correct top classes and their probabilities.

A `matplotlib` figure is created displaying an image and its associated top 5 most probable classes with actual flower names.

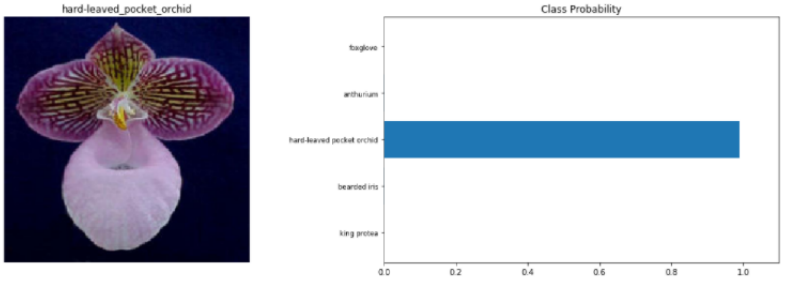
> The model predicts the images correctly 👍
> Great plots! The top 5 probable classes chart is a great visual tool, especially when the

Rate this review

START

model isn't completely confident in a class.

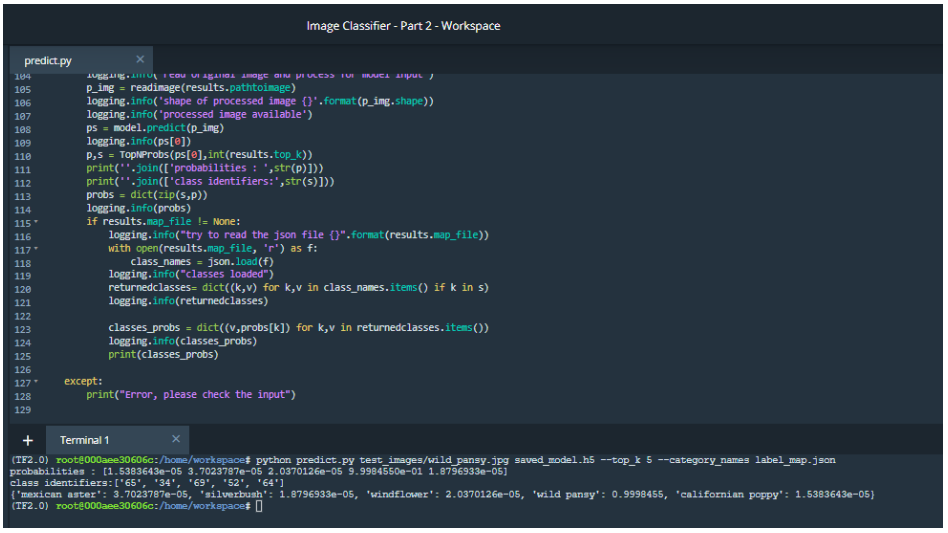
```
In [61]: fig, (ax1,ax2) = plt.subplots(figsize = (15,5), ncols = 2)
         ax1.imshow(processed_test_image)
         ax1.axis('off')
         ax1.set_title('hard-leaved_pocket_orchid')
         ax2.barh(np.arange(df.shape[0]),df['prob'])
         ax2.set_yticks(np.arange(df.shape[0]))
         ax2.set_yticklabels(df['class_name'],size='small')
         ax2.set_title('Class Probability')
         ax2.set_xlim(0, 1.1)
         plt.tight_layout()
```



Part 2 - Command Line Application

The `predict.py` script successfully reads in an image and a saved Keras model and then prints the most likely image class and it's associated probability.

> Great job, running your script gives the correct flower name as the prediction



The `predict.py` script allows users to print out the top K classes along with associated probabilities.

The `predict.py` script allows users to load a JSON file that maps the class values to other category names.

DOWNLOAD PROJECT

1 CODE REVIEW COMMENTS ›

Rate this review

START

RETURN TO PATH

Rate this review

START