UDACITY

‹ Return to "Introduction to Machine Learning with TensorFlow" in the classroom

DISCUSS ON STUDENT HUB

Create Your Own Image Classifier - TensorFlow

REVIEW

CODE REVIEW

HISTORY

Requires Changes

4 SPECIFICATIONS REQUIRE CHANGES

Hi Student, that was an excellent effort and you did a great job! 🎉 🎊
You are close to complete the project as the required changes are very small.
Don't get upset or disappointed, you did a great job which deserves a big compliment, think that those changes are a great opportunity to learn more and perfect your skills.
If you have any doubts or queries over any comment, feel free to post them on Knowledge section and our whole community along with our mentors are waiting to help you resolve all your queries and doubts.
Stay Udacious😊

Files Submitted

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

You need to attach the .html version of your notebook file.

Part 1 - Development Notebook

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

Nice job of importing all modules.

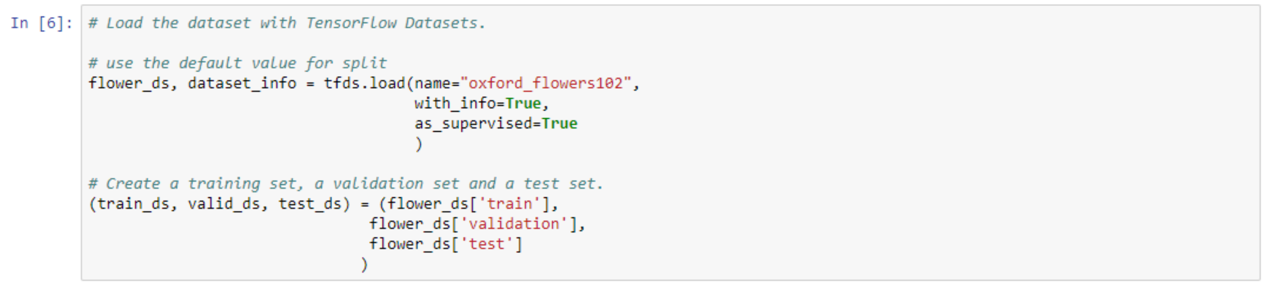
```
import tensorflow as tf
import tensorflow_datasets as tfds
import tensorflow_hub as hub
tfds.disable_progress_bar()
```

The Oxford Flowers 102 dataset is loaded using TensorFlow Datasets.

Nice work getting the appropriate dataset loaded!

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

Nice job using the built in set splits:



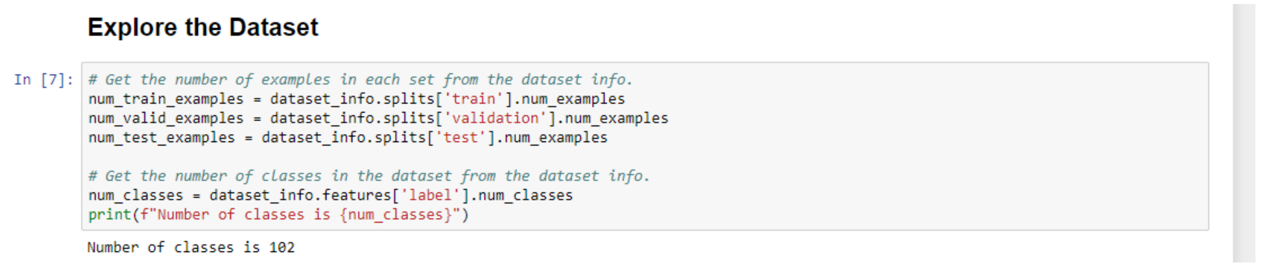
```
In [6]: # Load the dataset with TensorFlow Datasets.

        # use the default value for split
        flower_ds, dataset_info = tfds.load(name="oxford_flowers102",
                                            with_info=True,
                                            as_supervised=True
                                            )

        # Create a training set, a validation set and a test set.
        (train_ds, valid_ds, test_ds) = (flower_ds['train'],
                                         flower_ds['validation'],
                                         flower_ds['test']
                                        )
```

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

Once again, you've done well to use some of the built-in information for TensorFlow datasets to extract this data.



Explore the Dataset

```
In [7]: # Get the number of examples in each set from the dataset info.
        num_train_examples = dataset_info.splits['train'].num_examples
        num_valid_examples = dataset_info.splits['validation'].num_examples
        num_test_examples = dataset_info.splits['test'].num_examples

        # Get the number of classes in the dataset from the dataset info.
        num_classes = dataset_info.features['label'].num_classes
        print(f"Number of classes is {num_classes}")

        Number of classes is 102
```

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 [8]: # TODO: Print the shape and corresponding label of 3 images in the training set.
        for img, lbl in train_ds.take(3):
            print("Label: ", (lbl.numpy()), " -> ", img.shape)
```

```
Label:  72  ->  (500, 667, 3)
Label:  84  ->  (500, 666, 3)
Label:  70  ->  (670, 500, 3)
```

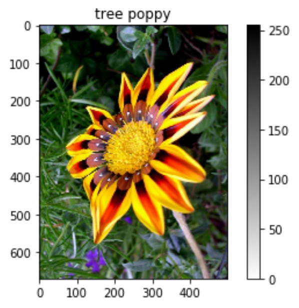
The first image from the training set is plotted with the title of the plot corresponding to the image label.

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.

Nice plot!

```
In [11]: # TODO: Plot 1 image from the training set. Set the title
         # of the plot to the corresponding class name.

         plt.imshow(img, cmap=plt.cm.binary)
         plt.colorbar()
         plt.title(f"{class_names[str(lbl.numpy())]}")
         plt.show()
```



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

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

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.

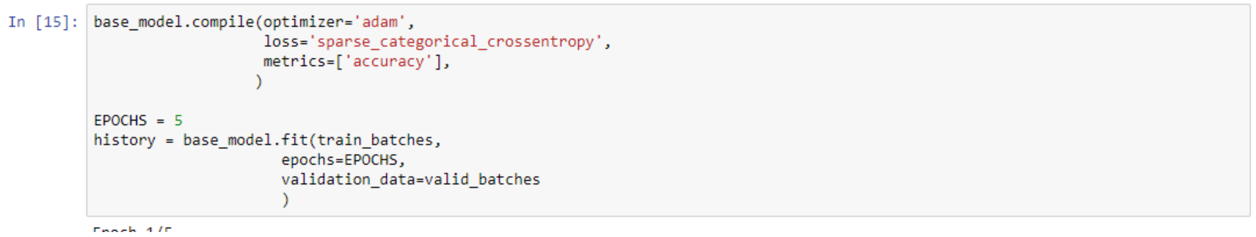
Great job loading the frozen MobileNet model.

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.

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.

Correctly compiled and fit.

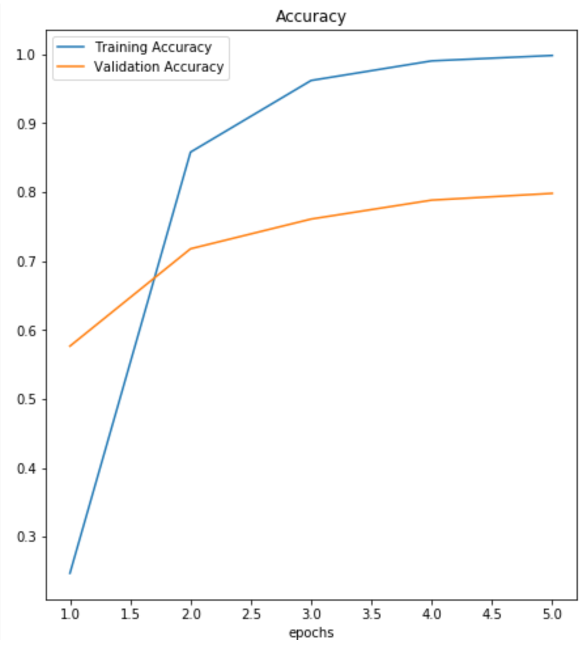


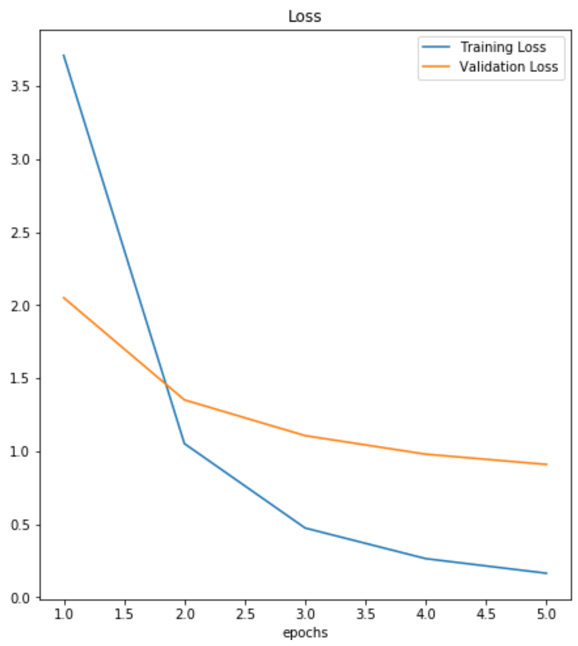
```
In [15]: base_model.compile(optimizer='adam',
                            loss='sparse_categorical_crossentropy',
                            metrics=['accuracy'],
                           )

         EPOCHS = 5
         history = base_model.fit(train_batches,
                                  epochs=EPOCHS,
                                  validation_data=valid_batches
                                  )
```

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.

Nice charts built with the history.





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`).

The saved Keras model is successfully loaded.

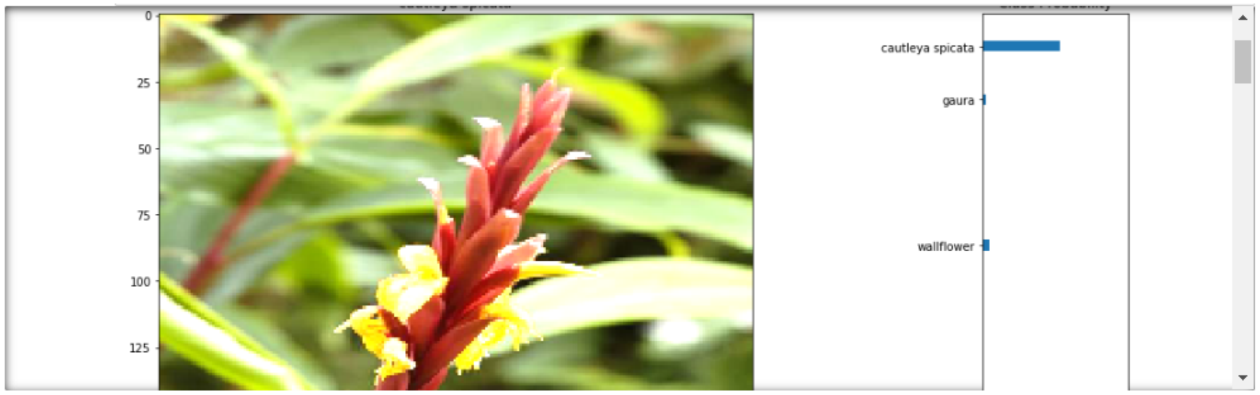
The `process_image` function successfully normalizes and resizes the input image. The image returned by the `process_image` function should be a NumPy array with shape `(224, 224, 3)`.

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.

Looks like your predict function works perfectly!

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

Great plot!



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.

You need to run the file after disabling the GPU. Run it on CPU environment. This should resolve your issue. If not, feel free to post the link of the error.

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

This will be checked after your issue will be resolved.

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

This will be checked after your issue will be resolved.

RESUBMIT

DOWNLOAD PROJECT

Best practices for your project resubmission

Ben shares 5 helpful tips to get you through revising and resubmitting your project.

Watch Video (3:01)

RETURN TO PATH

Rate this review