Create Your Own Image Classifier - TensorFlow

REVIEW | CODE REVIEW | HISTORY

Meets Specifications

Dear Student,

Congratulations!!!!🎉🎉🎉🎉
You've successfully passed all the specifications in this submission and I must admit that the structure of this project implementation is good. Going through the work, I could appreciate the time and effort put into this submission as it meets all the principal objectives. The results obtained from your models clearly proves how excellent your implementation was. :+1:

> used google colab for part 1

You did a good job.

Files Submitted

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

This submission includes all required files. Good work!!!

Part 1 - Development Notebook

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

Comment

Please, note that moving all the imports to the top is just good practice as it helps us understand the dependencies of the project from the beginning.

✓ The Oxford Flowers 102 dataset is loaded using TensorFlow Datasets.

Well done using `tfds.load()` to load the Oxford Flowers 102 dataset.

```
In [ ]: # TODO: Load the dataset with TensorFlow Datasets.
        dataset, dataset_info = tfds.load('oxford_flowers102', as_supervised = True, with_info = True)

        # Print the keys of the dataset dictionary
        print('\nThe keys of dataset are:', list(dataset.keys()))

        # TODO: Create a training set, a validation set and a test set.
        training_set, validation_set, test_set = dataset['train'], dataset['validation'], dataset['test']

        Downloading and preparing dataset oxford_flowers102/2.1.1 (download: 328.90 MiB, generated: 331.34 MiB, total: 660.25 MiB) to /
        root/tensorflow_datasets/oxford_flowers102/2.1.1...
```

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

Well done splitting the dataset into training, validation and test sets.

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

Nice job exploring each set, displaying the number of examples and classes in each.

Comment

One could format the output as follows for better clarity :

```
There are 1,020 images in the training set
There are 1,020 images in the validation set
There are 6,149 images in the test set
There are 102 classes in our dataset
```

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

Excellent job here examining the shapes of the first 3 images in the training set.

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

Awesome!
Nice job normalizing and resizing the datasets.

✓ 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.

Good job creating a pipeline for the training, validation and test sets.

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

Good job creating a pipeline for the training, validation and test sets.

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

Indeed, batches of images are returned for each set.

✓ The pipeline for each set should return batches of images.

Nice work with `.batch(batch_size).prefetch(1)` to create batches and prefetch the first one for each set.

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

Good job using `tf.keras.Sequential()` to create a new classifier.

✓ 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.

Comment

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. To read more about transfer learning, you can checkout this blog.
https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a

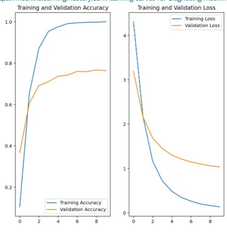
✓ The model is configured for training using the `compile` method with appropriate parameters. The model is trained using the `fit` method and incorporating the validation set.

Well done!!
To understand more about optimisers, check this link.
https://medium.com/datadriveninvestor/overview-of-different-optimizers-for-neural-networks-e0ed119440c3

✓ 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.

Good job plotting the learning curves!
The learning curves help us diagnose the behaviour of the model. To understand how these learning curves can give us insights, check this link.

https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/



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

Nice job getting a test accuracy.

> Accuracy on test set: 73.752%

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

Your method of saving the model is correct. Good Job!

✓ 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 `process_image()` function performs as expected. Well done!👍

✓ 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.

Great implementation here!
The top 5 most probably classes for the selected image and their probabilities are return.:+1:

An alternate way to approach this can be :

```
def predict(image_path, model, top_k):
    image = Image.open(image_path)
    image = np.asarray(image)
    image = process_image(image)
    predict = model.predict(np.expand_dims(image, axis=0))
    values, top_indices  = tf.math.top_k(predict, top_k)
    print("Propabilities: ",values.numpy()[0])
    class_= [class_names[str(value+1)] for value in top_indices.cpu().numpy()[0]]
    print('and the classes: ', class_)
    return values.numpy()[0], class_
```

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

Great job displaying your results .👍

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.

Good work done! The most probable class for a given image and its probability are return.:+1:

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

Well done!!

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

Nice, the script to load a JSON file that maps the class values to other category names.

DOWNLOAD PROJECT

RETURN TO PATH