Create Your Own Image Classifier - TensorFlow

Meets Specifications

Dear Learner,

Well done. Your Image classifier has correctly incorporated transfer learning components and leverages the weights configurations.

Model performance at around 77.5% is good , however there is overfit detected , which can be identified in graphs. Can experiments with model architecture and regularization methods to improve the result

- Data augmentation is very important to enrich the dataset for Deep learning models, DL frameworks such as Tensorflow, Pytorch offers various options for it

Building Complex Image Augmentation Pipelines with Tensorflow

- As a next steps , you can build model for object detection and learn how to incorporate pretrained frameworks such as YOLO

All the best

Files Submitted

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

All required files submitted as part of the attachment. Good work

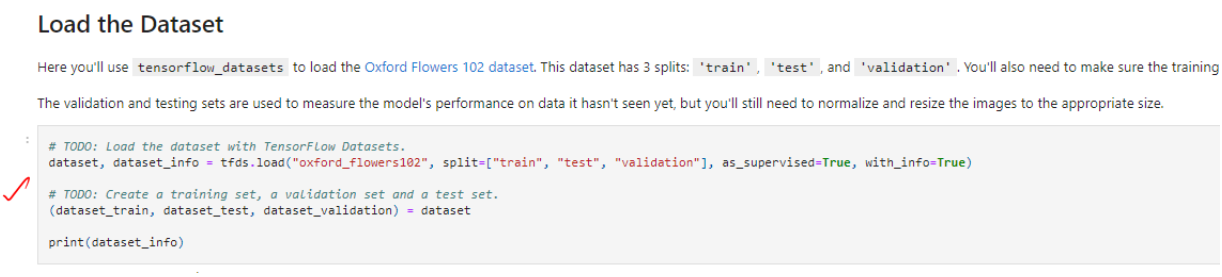
Part 1 - Development Notebook

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

Good work. All required packages and modules that are required for project are imported correctly

The Oxford Flowers 102 dataset is loaded using TensorFlow Datasets.

Good work. The required dataset imported correctly using tensorflow_datasets library.



Load the Dataset

Here you'll use `tensorflow_datasets` to load the Oxford Flowers 102 dataset. This dataset has 3 splits: `'train'`, `'test'`, and `'validation'`. You'll also need to make sure the training

The validation and testing sets are used to measure the model's performance on data it hasn't seen yet, but you'll still need to normalize and resize the images to the appropriate size.

```
# TODO: Load the dataset with TensorFlow Datasets.
dataset, dataset_info = tfds.load("oxford_flowers102", split=["train", "test", "validation"], as_supervised=True, with_info=True)

# TODO: Create a training set, a validation set and a test set.
(dataset_train, dataset_test, dataset_validation) = dataset

print(dataset_info)
```

- TFDS is a very useful library , however it is good to be aware of parameters in tfds.load API. As ignoring some parameters may trigger the download of hundreds of GB to disk.

A quick reference to the parameter will be helpful

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

The step correctly performed and the dataset is split into train , validation and test set .

- A reference on how to perform data split https://www.tensorflow.org/datasets/splits

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

Good work, You have correctly extracted these details from the dataset info

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

Good work. Steps correctly completed and shape of the images identified.

```
# TODO: Print the shape and corresponding label of 3 images in the training set.

for image, label in dataset_train.take(3):
    image = image.numpy()
    label = label.numpy()
    print(f'Image shape: {image.shape}, label: {label}, name: {dataset_info.features["label"].int2str(label)}'
```

```
Image shape: (500, 667, 3), label: 72, name: water lily
Image shape: (500, 666, 3), label: 84, name: desert-rose
Image shape: (670, 500, 3), label: 70, name: gazania
```

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

Correctly done. you have plotted the image with the label as title.

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 work. you have plotted the image with the class name as title correctly.

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

Good work. you have resized and normalized the incoming data.

Create Pipeline

```
# TODO: Create a pipeline for each set.

batch_size = 32
image_size = 224

def format_image(image, label):
    image = tf.cast(image, tf.float32)
    image = tf.image.resize(image, (image_size, image_size))
    image /= 255
    return image, label


training_batches = dataset_train.shuffle(num_examples_train).map(format_image).batch(batch_size).prefetch(1)
validation_batches = dataset_validation.map(format_image).batch(batch_size).prefetch(1)
testing_batches = dataset_test.map(format_image).batch(batch_size).prefetch(1)
```

Data augmentation is very important to enrich the dataset for Deep learning models, DL frameworks such as Tensorflow, Pytorch offers various options for it

- Building Complex Image Augmentation Pipelines with Tensorflow
- Data augmentation

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

Good work. Pipelines built for all the dataset and necessary transformation incorporated.

You can also make use of tf.data as part of latest release in TensorFlow for building pipelines from various sources https://www.tensorflow.org/guide/data

The pipeline for each set should return batches of images.

Good work. Pipeline returns the batches of images. Correctly configured.

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

Good work. You have used the TensorFlow hub to import the necessary pre-trained model.

```
# TODO: Build and train your network.
URL = "https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4"
feature_extractor = hub.KerasLayer(URL, input_shape=(image_size, image_size,3), output_shape=[1280])
feature_extractor.trainable = False

model = tf.keras.Sequential([
        feature_extractor,
        tf.keras.layers.Dropout(0.25),
        tf.keras.layers.Dense(num_classes, activation = 'softmax')
])
```

- TensorFlow Hub is a repository of various pre-trained models which are categorized based on usability , good exploration on it will give you valuable insights on various model available which can be used for your projects https://www.tensorflow.org/hub

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.

Perfectly done. you have made sure that the neurons aligns with the classes in the dataset.

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. you have complied the model with the required parameters. ADAM used as optimizer , a good choice for image based data. In-addition you can also explore on variants of it and check the performance.

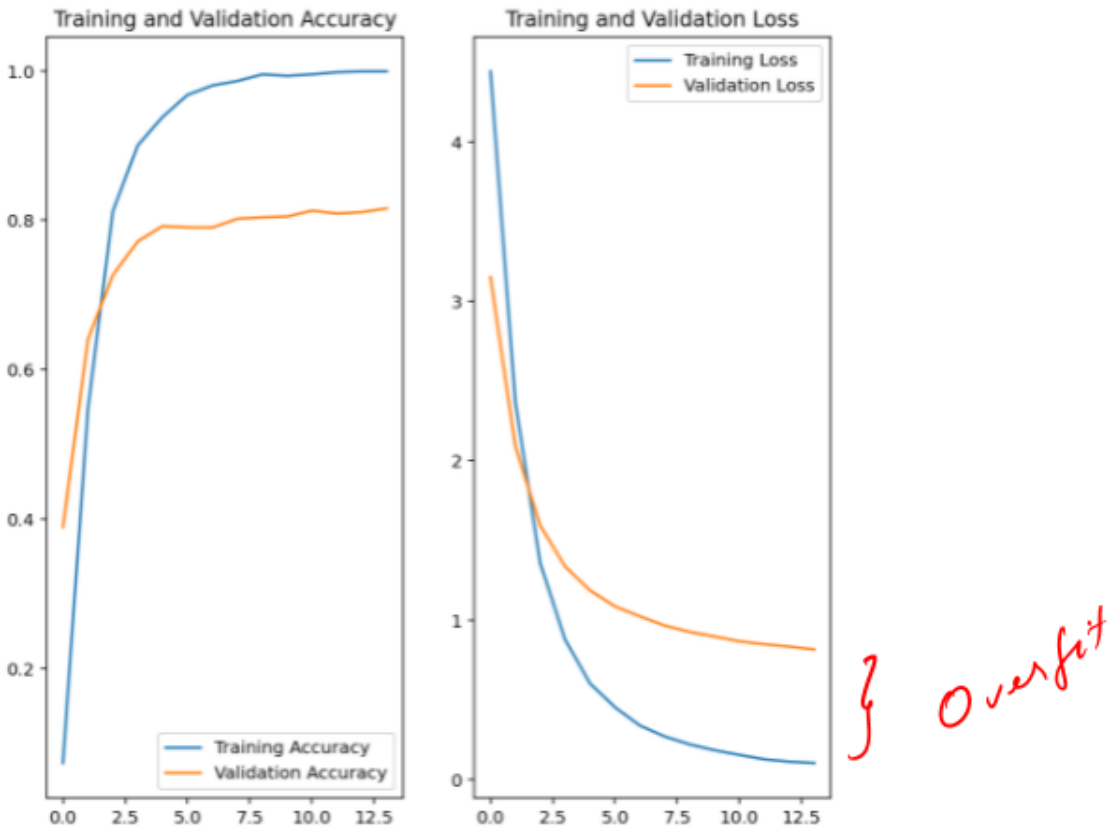
You can check for list of optimizers supported by TensorFlow
https://www.tensorflow.org/api_docs/python/tf/keras/optimizers
Optimizers such as Adamax , Adadelta can be experimented with

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 work. you have tracked the performance using plots. Overall

performance of the model is good.

- However , a case of overfit can be observed , recommend you to experiment on model architecture , use Dropouts , batch normalization to see the impact



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

Correctly done. Network accuracy measured using model.evalaute

Overall model performance is good at around 77.5%. However there is scope for improvement , mainly to reduce the overfit

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

Model saved correctly.

You can checkout various modal saving options https://www.tensorflow.org/guide/keras/save_and_serialize

The saved Keras model is successfully loaded.

Perfectly done. You have loaded the saved model correctly

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)`.

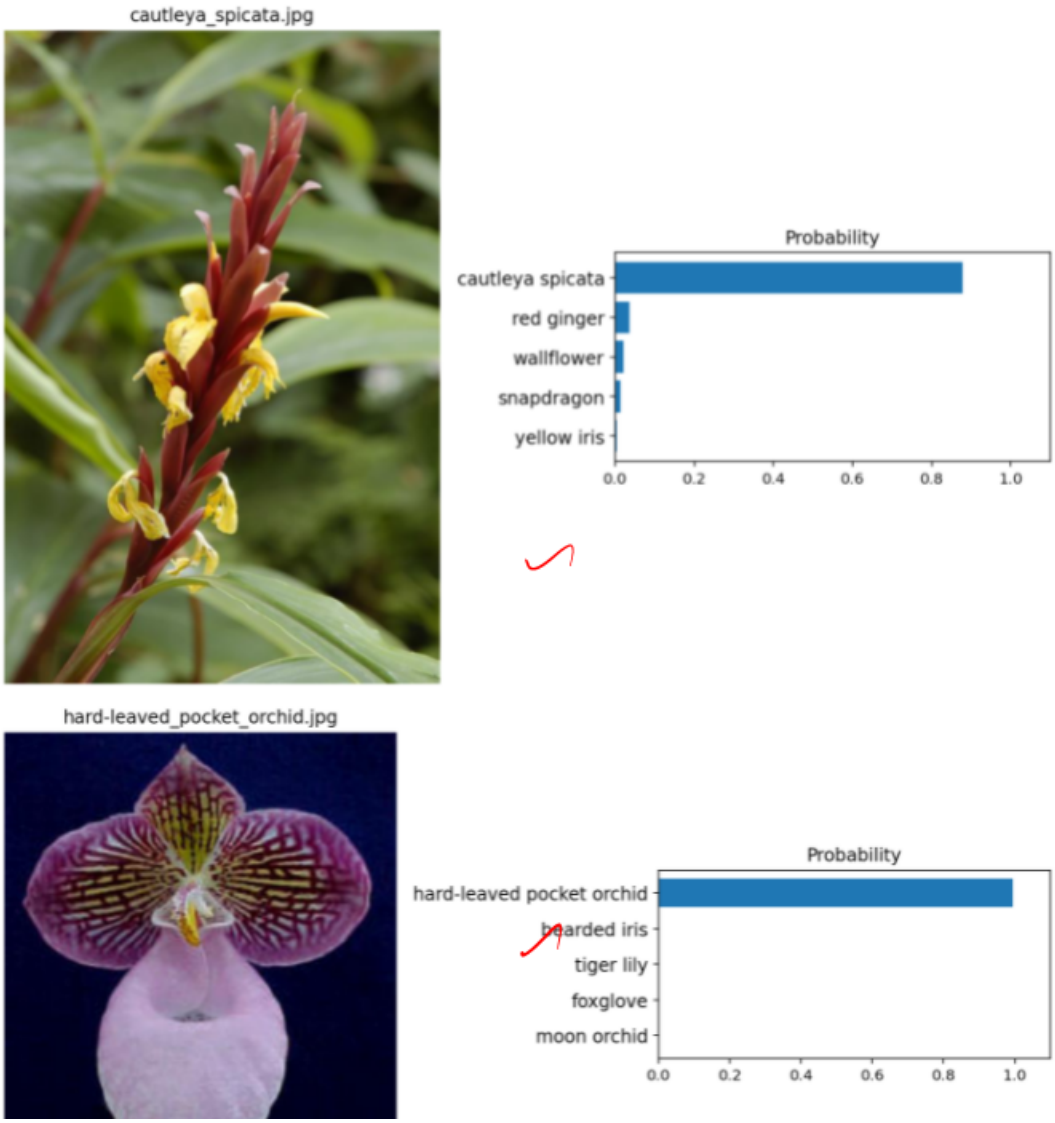
Good work. process_image functions created with correct expressions

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.

Correctly done. The function takes in the image data and gives out probable classes probabilities

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

Well done. Image are displayed with the correctly predicted labels



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. Correctly configured. The expressions for the predict steps aligns with the requirement

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.

Correctly done. JSON files loaded to map class values

DOWNLOAD PROJECT

2 CODE REVIEW COMMENTS ›

RETURN TO PATH

Rate this review

START