Create Your Own Image Classifier

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Congratulations 👏🎉 You passed this Project, that too on your first attempt. I hope you had great experience with Udacity and will continue taking other Nanodegree too. I have provided a few tips for further improving upon this project.

If you wish to study the subject further, here are few of my recommendations:

1. A great blog on Object-Detection Object detection is more difficult task than image classification as you can have more than 1 object in the same image
2. A guide to Semantic Segmentation for predicting the object class for each pixel.
3. Tutorial for Image Captioning Image Captioning is an even more difficult task than object detection as it requires abstractive summarisation and natural language sequence generation.
4. Also, here is great PyTorch tutorial repo in general by Yunjey Choi

Files Submitted

✓

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

Part 1 - Development Notebook

✓

All the necessary packages and modules are imported in the first cell of the notebook

Nice work making all import like those of `json` and `PIL` in the first cell of the notebook in order to clearly state all requirements at the start of the project. Here's a thread explaining the advantages of doing so.

✓

torchvision transforms are used to augment the training data with random scaling, rotations, mirroring, and/or cropping

Good job. This is important for making sure the classifier is trained robustly and appropriately as the image are fed to pre-trained models which require 224 input image size.

Recommended Changes

However, it's recommended to first `resize` while maintaining the ratio and then do a `randomcrop` for training sets (center crop for validation/test sets). his helps crop the main area into a 224x224 square. If we did not perform the resize and crop the center, we could end up with a wrong crop. For example, say we have a 1024x1024 image, if we cropped a 224x224 square at the center, we would crop out a really zoomed part of the flower, which is why a normalised resize is important.

✓

The training, validation, and testing data is appropriately cropped and normalized

✓

The data for each set is loaded with torchvision's DataLoader

✓

The data for each set (train, validation, test) is loaded with torchvision's ImageFolder

Nice work using torchvision's ImageFolder and DataLoader for quickly getting data in the required format.

✓

A pretrained network such as VGG16 is loaded from torchvision.models and the parameters are frozen

Good choice of architecture.

✓

A new feedforward network is defined for use as a classifier using the features as input

✓

The parameters of the feedforward classifier are appropriately trained, while the parameters of the feature network are left static

✓

During training, the validation loss and accuracy are displayed

✓

The network's accuracy is measured on the test data

Good job! It's nice seeing you get an accuracy of ~80%.

✓

There is a function that successfully loads a checkpoint and rebuilds the model

✓

The trained model is saved as a checkpoint along with associated hyperparameters and the class_to_idx dictionary

✓

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

✓

The process_image function successfully converts a PIL image into an object that can be used as input to a trained model

✓

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

Part 2 - Command Line Application

✓

train.py successfully trains a new network on a dataset of images and saves the model to a checkpoint

✓

The training loss, validation loss, and validation accuracy are printed out as a network trains

✓

The training script allows users to choose from at least two different architectures available from torchvision.models

✓

The training script allows users to set hyperparameters for learning rate, number of hidden units, and training epochs

✓

The training script allows users to choose training the model on a GPU

✓

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

✓

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

✓

The predict.py script allows users to use the GPU to calculate the predictions

✓

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

DOWNLOAD PROJECT

RETURN TO PATH

Rate this project

START