UDACITY

‹ Back to AI Programming with Python Nanodegree

Create Your Own Image Classifier

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Congratulations on clearing all the rubrics. Looking at your code, I feel that your understanding of the concepts of both transfer learning, neural networks and PyTorch are on point. I am sure you will have an exciting journey ahead with these knowledge that you have gained.

Files Submitted

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

All the required files are included 👌

Part 1 - Development Notebook

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

You forgot to move 'import json' to the first cell but it's ok

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

You have used the torchvision.transforms well in order to enhance the volume of the data by augmenting it with cropping, mirroring and rotation. Programmatic data augmentation increases the size of your training data set. Even better, your model will often be more robust (and less prone to overtting).

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

You have resized and normalized the data in accordance with the input size accepted by the pretrained models

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

The code looks good here as well. Well done!

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

Each set is correctly loaded and you have chosen an appropriate batch size 👌

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

Good work loading the pretrained networks and using transfer learning. I saw that you have perfectly freezed the parameters of the pretrained layers so that there is not gradient computation on back propagation call

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

Nice work here creating a new classier using nn.Sequential

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

This is perfect! You are only training the classier layers and not the layers from the pretrained models. Good job

During training, the validation loss and accuracy are displayed

Good work printing the logs.Both validation loss and accuracy are being captured in the logs.

The network's accuracy is measured on the test data

Great job on getting 81% accuracy on the testing set 👍

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

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

Great implementation here as well.

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

This is awesome. You have done well with the inference and its display 👌

Part 2 - Command Line Application

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

Great job with the command line utility. This work smoothly 👌

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

Training loss, validation loss etc. are being printed correctly during the training 👌

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

Great implementation and good work using cuda at the places required 👌

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

Tried it with varying values of K and it works like a charm.

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

The script has the capability to load from a JSON 👌

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

DOWNLOAD PROJECT

RETURN TO PATH