UDACITY

‹ Return to "AI Programming with Python Nanodegree" in the classroom

Create Your Own Image Classifier

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Nice job on your submission, and congratulations in passing the project. I've added a couple of links that I hope you find useful.

https://cv-tricks.com/cnn/understand-resnet-alexnet-vgg-inception/
https://chrisalbon.com/
https://machinelearningmastery.com/transfer-learning-for-deep-learning/

Good luck!

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

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

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

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

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

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

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

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

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 model is correctly trained and saved to the checkpoint file.

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

The losses and accuracy are printed during training. This is important because it allows you to monitor the model's progress, and lets you know if it starts overfitting.

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

Vgg13 and Densenet121 are properly supported. My suggestion here would be to provide the user with the names of the models supported, so that they do not enter unsupported models.

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

The required hyperparameters can be set by the user. I would suggest making the training on GPU by default, for practical purposes, as faster is always better.

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

Both GPU and CPU are supported. Make sure that even though the GPU flag is true, that the GPU is in fact available before switching to cuda. You can use `torch.cuda_is_available()` for that.

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 load a JSON file that maps the class values to other category names

Nice job using the json file provided by the user for querying class names.

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

See comment in the training part, as it applies here as well.

DOWNLOAD PROJECT

RETURN TO PATH