Create Your Own Image Classifier

Review

Meets Specifications

Going over your project, it is really well done. I believe you must be working with PyTorch for the first time and it can be quite intimidating at times.
You should definitely go through Justin's introduction to PyTorch.
Further Reading:

- Often as with new paradigm learning on experience of others are really helpful in the long-run. This and this are some of the best gems I have used over the past year to learn cool tips and hacks.

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

All good here!
Moving all the imports to the top is just a good practice as it helps in looking at the dependencies and the import
requirements of the project in one go.

Pro Tip
Read this recommendations about the imports.

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

Good Job!

- 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 overfitting).
- You can learn more about the usefulness of augmenting data in Image Classification tasks here.

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

Good Job !

- You have resized and normalized the data in accordance with the input size accepted by the pre-trained models.

Pro Tip
In cases like here, cropping a very large image without resizing would probably give you a very zoomed-in part of that image instead of the desired object.

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

- The code looks good here as well. Well done!
- Your code is well commented and clean, hence easy to go through.

Pro Tip:
If you'd like to take a closer look at how the batch size affects training you can check out this post.

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

- Each set is correctly loaded and you have chosen an appropriate batch size. You can also try trimming the batch size to 32.
- Training an image classifier from scratch normally requires a very large amount of data and takes a very long time to reach the desired accuracy.
- This is because the whole network has to learn how to identify from low-level features like lines, corners or edges to the more high-level features like eyes, ears or a nose (for a cat/dog classifier).
- Luckily most images have similar characteristics, so you can reuse the low-level feature extractors from pretrained models.
- This significantly speeds up training and decreases the amount of data you need to train your classifier.

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

Nice work here creating a new classifier using nn.Sequential

Good work loading the pretrained networks and using transfer learning.

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

- I saw that you have perfectly freezed the parameters of the pre-trained layers so that there is not gradient computation on backpropagation call, backward(), using:
- # Freeze parameters so we don't backprop through them
- for param in model.parameters():
- param.requires_grad = False
- ...
- model = models.vgg16(pretrained=True)
- # Only train the classifier parameters, feature parameters are frozen

- for param in model.parameters():
- param.requires_grad = False
- Also, good use of dropout to avoid the risk of overfitting.

Pro Tip:
I would highly recommend reading this post, on a discussion on how to choose the number of hidden units?

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 classifier layers and not the layers from the pre-trained models. Good job.

Pro Tip:

- You can read about early stopping to prevent the model from overfitting and avoid any unnecessary training. Check this

The network's accuracy is measured on the test data

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.

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

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