Image Classifier Application

REVIEW | CODE REVIEW | HISTORY

Requires Changes

8 SPECIFICATIONS REQUIRE CHANGES

Dear Student,

This was a very good first attempt. I think that your concepts are very clear and most of the majors tasks of the project are done and well implemented in this submission. Only a little fine tuning remains. I have suggested a few important changes, which I hope you implement. I believe that in the next review you will clear all the rubrics.

Good luck.

P.S. +1 for the clean and modular code in the command line application.

Files Submitted

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

Please upload your checkpoint file along with the project, so that it's easier and quicker for the next reviewer to complete their review.

Part 1 - Development Notebook

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

Good job. Importing all the packages on the top part of the file helps understand the dependencies of the program easily instead of going through the program and searching for `import` statements.

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

Good job here in using torchvision transforms to quickly and easily augment the data. You just need to make a small change. The `Resize(225)` is not required for the training set, since you are using `RandomResizedCrop` anyway.

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

Great work on using the 256 resize on the validation and test sets. This helps crop the main area into a 224x244 square. If we did not perform the resize and crop the centre, we would end up with the wrong crop. For example, say we have a 1000x1024 image, if we cropped a 224x224 square at the centre, we would crop out a really zoomed part of the flower, which is why the 256 resize is important.

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

Awesome. `Dataloader` helps load the data in batches, shuffle it and also allow usage of multiple workers to load large amount of data quickly. Great work on implementing it.

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

Great work on freezing the pre-trained parameters. This is the essence of Transfer Learning. With this step, you don't have to train the whole network again, which would otherwise had taken HUGE amount of time and a very large Dataset to accomplish.

Check this article to know how long these networks like resnet, vgg etc. take to train. It's fascinating.

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

Good work on using `nn.LogSoftmax`. This gives you a probability distribution of the predictions. With this you can say how much percentage confident the model is about a certain prediction. Example, the flower is 60% rose, 30% lili & 10% lotus.

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

Good job here. The line `optimizer = optim.Adam(model.classifier.parameters(), lr = 0.001)` only optimises the parameters of the `model.classifier` network, which is your custom feed forward classifier.

The network's accuracy is measured on the test data

Good accuracy of 78% on the training set.

During training, the validation loss and accuracy are displayed

Great job here. Logging the validation accuracy and loss helps you determine if the model is overfitting or under-fitting and help make required changes.

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

Good work here. Just a small change is required. Please store the `learning rate` and number of `epochs` as well in the checkpoint, which are very important hyperparameters as well. You should also store the name of the architecture(densenet, vgg, resnet, etc) that you have used, so that you can check that and load the respective pre-trained model when rebuilding.

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

Good job here.

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. A very small change required here is to execute the predict function in the next cell, and print out the predicted class and probability lists as shown in the Notebook. I'm sure, this should not take you more than a few seconds.

```
probs, classes = predict(image_path, model)
print(probs)
print(classes)
> [ 0.01558163  0.01541934  0.01452626  0.01443549  0.01407339]
> ['70', '3', '45', '62', '55']
```

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

While the centre crop and normalisation of the input image are on point. The initial resize implementation is missing. You need to resize the images where the shortest side is 256 pixels, keeping the aspect ratio. For example, if you have an image of size 512x1024, you need to resize the image to 256x512 and not 256x256. As I mentioned in one of my earlier comments "This helps crop the main area into a 224x244 square. If we did not perform the resize and crop the centre, we would end up with the wrong crop. For example, say we have a 1000x1024 image, if we cropped a 224x224 square at the centre, we would crop out a really zoomed part of the flower, which is why the 256 resize is important."

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

Good work implementing the training logic and saving the model with all the required hyperparameters.

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

Good options of `densenet` and `vgg` s given to the user

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

Good work on taking all the three inputs from the user here. Here one of the very important change required is the number of input and output Neurons needs to be dynamic, unlike that you have mentioned. These informations are not available with the user, and should be taken dynamically from the Pre-trained network that you are using.

You can get the `in_features` from the `in_features` of the first layer of the existing classifier of the pre-trained model and the out_features is a constant 102 for the project.

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

Great work supporting the use of GPU here and pushing the tensors to the GPU when required. Please allow the use of densenet on CPU as well.

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

Great work implementing this application. Please do the required change in the `process_image` function as mentioned in one of the above comments and you are all set.

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

+1 for the `display_result` function. Great work.

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

Great work in supporting models trained on CPU & GPU as well using `map_location` to load the model into the right place while loading the model.

RESUBMIT PROJECT

DOWNLOAD PROJECT

Best practices for your project resubmission

Ben shares 5 helpful tips to get you through revising and resubmitting your project.

Watch Video (3:01)

RETURN TO PATH

Student FAQ