**Meets Specifications**

Hello there,

Congratulations on finishing the project 🎉  
Keep doing the great work and all the best for future project.

**Files Submitted**

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

I can see that you have included all of the files required for the submission. Great Job!!!

**Part 1 - Development Notebook**

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

Good Job importing all the dependencies in the top cell of the notebook, There was one import json missing in the first cell. This is not really an issue but just a point to take note off

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

This is perfect! You used torchvision.transforms to implement data augmentation with rotation, mirroring and cropping.

Usually, the more data an ML algorithm has access to, the more effective it can be. I suggest this short paper if you want to learn more about the benefits and different methods of [data augmentation](http://cs231n.stanford.edu/reports/2017/pdfs/300.pdf) in Image Classification.

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

Well done. You were able to resize and normalize the data according to the requirements of the pretrained models.

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

The code looks good here as well. Each set is correctly loaded with torchvision's DataLoader and a reasonable batch size.

If you'd like to take a closer look at how the batch size affects training you can check out [this post](https://stats.stackexchange.com/a/141265).

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

Well done here! You used nn.Sequential perfectly to create a new classifier for your network.

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

Good job with freezing the parameters of the pretrained layer by:

# replace the pre-trained model's classifier, freezing model params

for param in model.parameters():

param.requires\_grad = False

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

**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**

Perfect! Training loss, validation loss and accuracy are all displayed. You definitely want to keep an eye on these metrics to catch any bugs or inconsistencies with your model quickly.

**The network's accuracy is measured on the test data**

Yes, good implementation and great job on that 88.3% accuracy on your test set!

**The trained model is saved as a checkpoint along with associated hyperparameters and the class\_to\_idx dictionary**

This is very well done. The model is successfully saved as a checkpoint and all of the necessary hyperparameters are included to rebuild the model later.

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

Good job! Your load\_full\_model\_from\_checkpoint function does exactly what it's supposed to do and successfully rebuilds the model from a checkpoint.

**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 looks great! You managed to make it almost identical to the example images, very well done.

**Part 2 - Command Line Application**

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

This works like a charm! I personally tried the application and trained a new model in no time.

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

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

Perfect here as well. All the required metrics are correctly printed out during training.

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

This is well implemented. Great use of torch.cuda.is\_available() when necessary.

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

It definitely does and they work as expected.

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

I personally tried the script and it works perfectly.

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

Very well done, I tried the script with different values of K and it works as expected.

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

This option is included as well. Good job!

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

Well done. I tried this as well and no problem at all.