UDACITY

Logout

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

DISCUSS ON STUDENT HUB ›

Create Your Own Image Classifier

REVIEW | CODE REVIEW 3 | HISTORY

Meets Specifications

Dear Student,

Great work passing this project. I am sure you have learnt a lot from this project. You have demonstrated great python skills and understanding of implementing neural netwoks in pytorch. Your code is absolutely fantastic. Keep up this great work !!

You have achieved good accuracy on test set , you can read further by going to the following links:

- Comparison of Deep Learning Models for Image Classification
- Summary of the current state of Image Classification

Best Wishes !!
PS: Kindly leave a feedback and rating if you find this review helpful :)

Files Submitted

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

Well done submitting all of the required files!

Part 1 - Development Notebook

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

Well done organizing all of the import statements on the first cell of the notebook.

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

Great work augmenting the training data, this will not only increase the data but also the trained model will be highly robust.

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

Good work, normalizing the data using the mean and standard deviation of the flowers dataset

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

Good job, loading the dataset using ImageFolder . Also, you have written the code very concisely.

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

Good job, loading the dataset using DataLoader correctly.

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

You have done a good job using pretrained VGG16 and you have frozen the parameters correctly.

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

Your feedforward network is suitable for this task and dataset. You can experiment with deeper network, but do introduce dropout layer because model will quickly start overfitting.

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

Only the feedforward classifier is being appropriately trained while the feature network parameters are left static.

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

Good job achieving an accuracy of 82% on the test dataset.

✓ **During training, the validation loss and accuracy are displayed**

Well done clearly logging the validation loss and accuracy at each step!

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

Great job saving the major hyperparameters in the checkpoint! This practice will make it easy to retrain your model in the future!

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

Nicely done writing the load_checkpoint method to successfully load the checkpoint and rebuild the model

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

Yep, the code has been modified to correctly resize, crop and normalize the image values

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

Awesome job finding the top K classes along with the associated probabilities. Well done inverting the class_to_idx mappings to get the correct flower classes.

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

Your plot of predicted probability is correctly implemented, 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**

Your script is really good and shows you have worked all the possible cases and tested your code on various parameters. Great job !!

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

Well done clearly logging the validation loss and accuracy at each step!

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

You have correctly chosen two different architectures. Here is how you can find all input features for all the architectures:

```
from torchvision import models
# then print the model architecture:
model  = model.densenet201()
print(model)
# in the classifier find that there are in_features for each layer. the in_features of the first cl
assifier layer is the value of in_features you should use
# to access this for densenet201
print(model.classifier.in_features)

# Now try this for resnet
model = models.resnet101()
print(model)
# As you can see the classifier of renset is called `fc` to print it's input features
print(model.fc.in_features)

# Now try this for alexnet
model = models.alexnet()
print(model)
# As you can see the first classifier layer is dropout, but we are required first linear layer of c
lassifier and it's in_features
print(model.classifier[1].in_features)
```

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

Good job you have provided command line option for all the hyperparameters asked in the specification.

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

Excellent job providing the support for gpu and handling the edge case of using it only if it's available on the device, not only because the user said so.

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

Predict script is very well implemented, also you have taken care of most of the edge cases.

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

Awesome job finding the top K classes along with the associated probabilities. Well done inverting the class_to_idx mappings to get the correct flower classes.

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

You have correctly loaded the json file from the provided path using command line option. Good work !!
You have correctly loaded the json file from the provided path using command line option. Good work !!

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

Excellent job providing the support for gpu and handling the edge case of using it only if it's available on the device, not only because the user said so.

DOWNLOAD PROJECT

3 CODE REVIEW COMMENTS ›

RETURN TO PATH