‹ Back to AI Programming with Python Nanodegree

Image Classifier Application

REVIEW

CODE REVIEW 8

HISTORY

▸ train.py 4

▾ predict.py 3

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# *AIPND/ImageClassifierApp/predict.py

#
# reads in an image and a checkpoint then prints the most likely image class
# and it's associated probability.
# specs :
# The script reads in an image and a checkpoint to print the most likely image c
# Allow user to load a JSON file that maps the class values to other category na
# Allow user to request a print out of top k classes and probabilities
# Allow user to request the predictions be done in GPU model
#
# About command line arguments of predict.py :
#   checkpoint : specify a saved checkpoint in dir 'chksav'. If not supplied, us
#   --img_pth : specify an image in dir 'flowers'. If not supplied, use 'flowers
#   --category_names : specify a category name JSON mapper file. If not supplied
#   --top_k : specify no. of top k classes to print. Default is 1
#   --gpu : run predict in GPU mode (subject to device capability), default is
# Example Calls
# Ex 1, use checkpoint 'chkpt.pth' in 'chksav': python predict.py chksav/chkpt.
# Ex 2, use top_k 4 and GPU : python predict.py --top_k 4 --gpu
# Ex 3, use img_pth 'flowers/test/91/image_08061.jpg' and cat name mapper 'cat_
#   python predict.py --img_pth flowers/test/91/image_08061.jpg --category_names
```

```python
#
import argparse
import torch
from torchvision import models
import numpy as np
from PIL import Image

import json
from datetime import datetime
import os
import glob
import sys

from workspace_utils import active_session

device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
data_dir = 'flowers'
test_dir = data_dir + '/test'
chkptdir = 'chksav'
# use the last checkpoint saved as default
if len(glob.glob(chkptdir+'/*.pth')) > 0 :
    checkpt = max(glob.glob(chkptdir+'/*.pth'), key=os.path.getctime)
else:
    checkpt = None
    print('\n*** no saved checkpoint to load ... exiting\n')
    sys.exit(1)

def main():
    # collecting start time
    start_time = datetime.now()

    # get input arguments
    args = get_input_args()
    print('\n*** command line arguments ***')
    print('checkpoint:', args.checkpoint, '\nimage path:', args.img_pth,
          '\ncategory names mapper file:', args.category_names, '\nno. of top k
          '\nGPU mode:', args.gpu, '\n')

    if len(glob.glob(args.checkpoint)) == 0:
        print('*** checkpoint: ', args.checkpoint, ', not found ... exiting\n')
        sys.exit(1)

    if len(glob.glob(args.img_pth)) == 0:
        print('*** img_pth: ', args.img_pth, ', not found ... exiting\n')
        sys.exit(1)

    if len(glob.glob(args.category_names)) == 0 :
        print('*** category names mapper file: ', args.category_names, ', not f
        sys.exit(1)

    if args.top_k < 1:
        print('*** no. of top k classes to print must >= 1 ... exiting\n')
        sys.exit(1)

    if device.type == 'cuda':
        if args.gpu:
            print('*** GPU is available, using GPU ...\n')
        else:
            print('*** using GPU ...\n')
    else:
        if args.gpu:
            print('*** GPU is unavailable, using CPU ...\n')
        else:
            print('*** using CPU ...\n')

    # retrieve model from checkpoint saved
    model = load_checkpoint(args)

    # call predic function with required and optional input parameters
    with active_session():
        predict(model, args)

    elapsed = datetime.now() - start_time
    print('\n*** prediction done ! \nElapsed time[hh:mm:ss.ms]: {}'.format(elap

def get_input_args():
    # create parser
    parser = argparse.ArgumentParser()

    parser.add_argument('checkpoint', type=str, nargs='?', default=checkpt,
                        help='path to saved checkpoint')

    parser.add_argument('-img','--img_pth', type=str, default=test_dir + '/91/i
                        help='path to an image file')

    parser.add_argument('-cat','--category_names', dest='category_names', defau
                        type=str, help='path to JSON file for mapping class valu

    parser.add_argument('-k','--top_k', dest='top_k', default=1, type=int,
                        help='no. of top k classes to print')

    parser.add_argument('--gpu', dest='gpu', default=False, action='store_true'
                        help='predict in gpu mode')

    return parser.parse_args()

def load_checkpoint(args):
    if device.type == 'cuda':
        print('*** loading chkpt', args.checkpoint,' in cuda ...\n')
        checkpoint = torch.load(args.checkpoint)
    else:
        print('*** loading chkpt', args.checkpoint,' in cpu ...\n')
        checkpoint = torch.load(args.checkpoint, map_location=lambda storage, l

    model = models.__dict__[checkpoint['arch']](pretrained=True)
    model.classifier = checkpoint['classifier']
    model.class_to_idx = checkpoint['class_to_idx']
    model.load_state_dict(checkpoint['state_dict'])

    return model

def process_image(image):
    ''' Scales, crops, and normalizes a PIL image for a PyTorch model,
        returns an Numpy array
    '''
```

AWESOME

> **Great job using docstrings to document major classes and methods!**
>
> To learn more about the benefits of documentation, check out this article.

```
    pil_img=image

    sz = image.size
    h = min(image.size)
    w = max(image.size)
    #print('size:',sz, ', h:',h, ', w:',w)

    # calculate ratio_aspect using original height & width
    # chosen h is 256, ratio aspect for adjusted w is original w/h
    ratio_aspect = w/h

    # get indices of short and long sides
    x = image.size.index(min(image.size))
    y = image.size.index(max(image.size))

    # calc new size with short side 256 pixels keeping ratio aspect
    new_sz = [0, 0]
    new_sz[x] = 256
    new_sz[y] = int(new_sz[x] * ratio_aspect)

    #print('new_sz:',new_sz, '\npre resized img:', pil_img)

    # resize base on short side of 256 pixels
    pil_img=image.resize(new_sz)
    #print('post resized image:', pil_img)

    # crop out the center 224x224 portion
    wid, hgt = new_sz
    #print('wid:', wid, ', hgt:', hgt)

    # calc left, top, right, bottom margin pos
    l_margin = (wid - 224)/2
    t_margin = (hgt - 224)/2
    r_margin = (wid + 224 )/2
    b_margin = (hgt + 224)/2

    #print('left:',l_margin, ', top:',t_margin, ', right:',r_margin, ', bottom:

    # crop the image
    pil_img=pil_img.crop((l_margin, t_margin, r_margin, b_margin))
    #print('cropped img:', pil_img)

    # convert to np array for normalization purpose
    np_img = np.array(pil_img)

    print('np_img.shape',np_img.shape)

    np_img = np_img/255
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    np_img = (np_img - mean)/std

    # transpose to get color channel to 1st pos
    np_img = np_img.transpose((2, 0, 1))

    return np_img


def predict(model, args):
    ''' Predict the class (or classes) of an image using a trained deep learnin
    '''

    model.cpu()
    model.eval()

    pil_img = Image.open(args.img_pth)
    image = process_image(pil_img)
    image = torch.FloatTensor(image)

    model, image = model.to(device), image.to(device)
```

AWESOME

Nicely done configuring the application to run on GPU!

```

    print('\nori image.shape:', image.shape)
    image.unsqueeze_(0) # add a new dimension in pos 0
    print('new image.shape:', image.shape, '\n')


    output = model.forward(image)

    # get the top k classes of prob
    ps = torch.exp(output).data[0]
    topk_prob, topk_idx = ps.topk(args.top_k)

    # bring back to cpu and convert to numpy

    topk_probs = topk_prob.cpu().numpy()
    topk_idxs = topk_idx.cpu().numpy()

    # map topk_idx to classes in model.class_to_idx
    idx_class={i: k for k, i in model.class_to_idx.items()}
    topk_classes = [idx_class[i] for i in topk_idxs]

    print('*** Top ', args.top_k, ' classes ***')
    # map class to class name
    if args.category_names:
        with open(args.category_names, 'r') as f:
```

AWESOME

Nicely done using the user specified JSON file for category mapping!

```
            cat_to_name = json.load(f)

        topk_names = [cat_to_name[i] for i in topk_classes]
        print('class names:   ', topk_names)
```

```

    print('classes:       ', topk_classes)
    print('probabilities: ', topk_probs)



# Call to main function to run the program
if __name__ == "__main__":
    main()

```

▸ README.md 1

▸ workspace_utils.py

RETURN TO PATH

Rate this review

Student FAQ