UDACITY

< Back to AI Programming with Python Nanodegree

Image Classifier Application

REVIEW

CODE REVIEW 8

HISTORY

train.py 4

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# *AIPND/ImageClassifierApp/train.py
#
# trains a new network on a dataset of images
# specs :
# The training loss, validation loss, and validation accuracy are printed out a
# allows users to choose from at least two different architectures available fr
# allows users to set hyperparameters for learning rate, number of hidden units
# allows users to choose training the model on a GPU
#
# Example calls:
# Ex 1, use data_dir 'flowers': python train.py flowers
# Ex 2, use save_dir 'chksav' to save checkpoint: python train.py --save_dir ch
# Ex 3, use densenet161 and hidden_units '1000, 500': python train.py --arch de
# Ex 4, set epochs to 10: python train.py -e 10
# Ex 5, set learning rate to 0.002: python train.py -lr 0.002
# Ex 6, train in GPU mode (subject to device capability): python train.py --gpu

import argparse
import torch
import numpy as np
from torch import nn
from torch import optim
import torch.nn.functional as F
from torchvision import datasets, transforms, models
from PIL import Image
```

```
28
29 from datetime import datetime
30 import os
31 import glob
32 import sys
33
34 from workspace_utils import active_session
35
36 device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
37 model_names = ['densenet121', 'densenet161', 'vgg16']
38 datadir = 'flowers'
39 savedir = 'chksav'
40
41 # main
42 def main():
43     # get input arguments and print
44     args = get_input_args()
45     print('\n*** command line arguments ***')
46     print('architecture:', args.arch, '\ndata dir:', args.data_dir, '\nchkpt di
47           '\nlearning rate:', args.learning_rate, '\nhidden layer:', args.hidde
48            '\nepochs:', args.epochs, '\nGPU mode:', args.gpu, '\n')
```

AWESOME

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

```
49
50     if len(glob.glob(args.data_dir)) == 0:
51         print('*** data dir: ', args.data_dir, ', not found ... exiting\n')
52         sys.exit(1)
53
54     if args.learning_rate <= 0:
55         print('*** learning rate cannot be negative or 0 ... exiting\n')
56         sys.exit(1)
57
58     # if hidden units supplied, check values are numeric
59     if args.hidden_units:
60         try:
61             list(map(int, args.hidden_units.split(',')))
62         except ValueError:
63             print("hidden units contain non numeric value(s) :[", args.hidden_u
64             sys.exit(1)
65
66     if args.epochs < 1:
67         print('*** epochs cannot be less than 1 ... exiting\n')
68         sys.exit(1)
69
70     # transform and load training, validatation and testing sets
71     dataloaders = transform_load(args)
72
73     # load pre-trained model and replace with custom classifier
74     model = models.__dict__[args.arch](pretrained=True)
75     model = build_classifier(model, args, dataloaders)
76     print('\n*** model architecture:',args.arch,'\n*** Classifier:\n', model.cl
77
78     # set training criterion and optimizer
79     criterion = nn.NLLLoss()
       optimizer = optim.Adam(model.classifier.parameters(), args.learning_rate)
```

```
80
82      # start model training and testing
83      if device.type == 'cuda':
84          if args.gpu:
85              print('*** GPU is available, using GPU ...\n')
86          else:
87              print('*** training classifier in GPU mode ...\n')
88      else:
89          if args.gpu:
90              print('*** GPU is unavailable, using CPU ...\n')
91          else:
92              print('*** training classifier in CPU mode ...\n')
93
94      with active_session():
95          model = train(model, dataloaders, optimizer, criterion, args.epochs, 4(
96          model = test(model, dataloaders, criterion)
97
98      # save to checkpoint
99      model = model.cpu() # back to CPU mode post training
100     model.class_to_idx = dataloaders['train'].dataset.class_to_idx
101
102     # if checkpoint dir not exists, create it
103     if not os.path.isdir(args.save_dir):
104         os.makedirs(args.save_dir)
105
106     checkpoint = {
107         'classifier': model.classifier,
108         'state_dict': model.state_dict(),
109         'class_to_idx' : model.class_to_idx,
110         'optimizer': optimizer.state_dict(),
111         'arch': args.arch,
112         'lrate': args.learning_rate,
113         'epochs': args.epochs}
114
115     chkpt = datetime.now().strftime('%Y%m%d_%H%M%S') + '_' + args.arch + '.pth
116     checkpt = os.path.join(args.save_dir, chkpt)
117
118     torch.save(checkpoint, checkpt)
119     print('\n*** checkpoint: ', chkpt, ', saved to: ', os.path.dirname(checkpt)
120
121
122 def get_input_args():
123     # create parser
124     parser = argparse.ArgumentParser()
125
126     parser.add_argument('data_dir', type=str, nargs='?', default=datadir,
127                         help='path to datasets')
128
129     parser.add_argument('--save_dir', type=str, default=savedir,
130                         help='path to checkpoint directory')
131
132     parser.add_argument('--arch', dest='arch', default='densenet121',
133                         choices=model_names, help='model architecture: ' +
134                         ' | '.join(model_names) + ' (default: densenet121)')
135
136     parser.add_argument('-lr','--learning_rate', dest='learning_rate', default=
137                         help='learning rate (default: 0.001)')
138
139     parser.add_argument('-hu','--hidden_units', dest='hidden_units', default=N(
140                         help='hidden units, one or multiple values (comma sepa
```

```
                        """ enclosed in single quotes. Ex1. one value: '500'
                            Ex2. multiple values: '1000, 500' """)

    parser.add_argument('-e','--epochs', dest='epochs', default=3, type=int,
                        help='total no. of epochs to run (default: 3)')

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

AWESOME

Well done declaring the arguments for configuring the command line application!

```

    return parser.parse_args()

def transform_load(args):
    train_dir = args.data_dir + '/train'
    valid_dir = args.data_dir + '/valid'
    test_dir = args.data_dir + '/test'

    normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                             std=[0.229, 0.224, 0.225])
    # define transforms
    data_transforms = {
        'train': transforms.Compose([
            transforms.RandomRotation(30),
            transforms.RandomResizedCrop(224),
            transforms.RandomHorizontalFlip(),
            transforms.ToTensor(),
            normalize]),
        'valid': transforms.Compose([
            transforms.Resize(256),
            transforms.CenterCrop(224),
            transforms.ToTensor(),
            normalize]),
        'test': transforms.Compose([
            transforms.Resize(256),
            transforms.CenterCrop(224),
            transforms.ToTensor(),
            normalize])
        }

    # define datasets

    image_datasets = {
        'train': datasets.ImageFolder(train_dir, transform=data_transforms['tra
        'valid': datasets.ImageFolder(valid_dir, transform=data_transforms['val
        'test': datasets.ImageFolder(test_dir, transform=data_transforms['test
        }

    #  define dataloaders
    dataloaders = {
        'train': torch.utils.data.DataLoader(image_datasets['train'], batch_si
        'valid': torch.utils.data.DataLoader(image_datasets['valid'], batch_si
        'test': torch.utils.data.DataLoader(image_datasets['test'], batch_size=
        }
```

```
    return dataloaders

def build_classifier(model, args, dataloaders):
    # Freeze parameters so we don't backprop through them
    for param in model.parameters():
        param.requires_grad = False

    in_size = {
        'densenet121': 1024,
        'densenet161': 2208,
        'vgg16': 25088,
        }

    hid_size = {
        'densenet121': [500],
        'densenet161': [1000, 500],
        'vgg16': [4096, 4096,1000],
```

AWESOME

Great job allowing the user to specify the model architecture between DenseNet121, DenseNet161 and VG

```
        }

    output_size = len(dataloaders['train'].dataset.classes)
    relu = nn.ReLU()
    dropout = nn.Dropout()
    output = nn.LogSoftmax(dim=1)

    if args.hidden_units:
        h_list = args.hidden_units.split(',')
        h_list = list(map(int, h_list)) # convert list from string to int
    else:
        h_list = hid_size[args.arch]

    h_layers = [nn.Linear(in_size[args.arch], h_list[0])]
    h_layers.append(relu)
    if args.arch[:3] == 'vgg':
        h_layers.append(dropout)

    if len(h_list) > 1:
        h_sz = zip(h_list[:-1], h_list[1:])
        for h1,h2 in h_sz:
            h_layers.append(nn.Linear(h1, h2))
            h_layers.append(relu)
            if args.arch[:3] == 'vgg':
                h_layers.append(dropout)

    last = nn.Linear(h_list[-1], output_size)
    h_layers.append(last)
    h_layers.append(output)

    print(h_layers)
    model.classifier = nn.Sequential(*h_layers)

    return model

# validate model
```

```
247 def validate(model, dataloaders, criterion):
248     valid_loss = 0
249     accuracy = 0
250 
251     for images, labels in iter(dataloaders['valid']):
252 
253         images, labels = images.to(device), labels.to(device)
254 
255         output = model.forward(images)
256         valid_loss += criterion(output, labels).item()
257         ps = torch.exp(output)
258         equality = (labels.data == ps.max(dim=1)[1])
259         accuracy += equality.type(torch.FloatTensor).mean()
260 
261     return valid_loss, accuracy
262 
263 
264 # train model
```

SUGGESTION

Please use `docstrings` to document your major classes and methods. Here's the reference

```
265 def train(model, dataloaders, optimizer, criterion, epochs=2, print_freq=20, l
266 
267     model.to(device)
268     start_time = datetime.now()
269 
270     print('epochs:', epochs, ', print_freq:', print_freq, ', lr:', lr, '\n')
271 
272     steps = 0
273 
274     for e in range(epochs):
275         model.train()
276         running_loss = 0
277         for images, labels in iter(dataloaders['train']):
278             steps +=1
279 
280             images, labels = images.to(device), labels.to(device)
281 
282             optimizer.zero_grad()
283 
284             output = model.forward(images)
285             loss = criterion(output, labels)
286             loss.backward()
287             optimizer.step()
288 
289             running_loss += loss.item()
290 
291             if steps % print_freq == 0:
292                 model.eval()
293 
294                 with torch.no_grad():
295                     valid_loss, accuracy = validate(model, dataloaders, criter
296 
297                 print('Epoch: {}/{}..'.format(e+1, epochs),
298                       'Training Loss: {:.3f}..'.format(running_loss/print_freq
                      'Validation Loss: {:.3f}..'.format(valid_loss/len(datalo
```

```
                      'Validation Accuracy: {:.3f}%'.format(accuracy/len(datal
                    )
                running_loss = 0

                model.train()

    elapsed = datetime.now() - start_time

    print('\n*** classifier training done ! \nElapsed time[hh:mm:ss.ms]: {}'.fo
    return model

# test model
def test(model, dataloaders, criterion):
    print('\n*** validating testset ...\n')
    model.cpu()
    model.eval()

    test_loss = 0
    total = 0
    match = 0

    start_time = datetime.now()

    with torch.no_grad():
        for images, labels in iter(dataloaders['test']):

            model, images, labels = model.to(device), images.to(device), labels

            output = model.forward(images)
            test_loss += criterion(output, labels).item()
            total += images.shape[0]
            equality = labels.data == torch.max(output, 1)[1]
            match += equality.sum().item()

    model.test_accuracy = match/total * 100
    print('Test Loss: {:.3f}'.format(test_loss/len(dataloaders['test'])),
          'Test Accuracy: {:.2f}%'.format(model.test_accuracy))

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

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

▸ predict.py 3

▸ README.md 1

▸ workspace_utils.py

Rate this review

Student FAQ