UDACITY

Logo

‹ Back to AI Programming with Python Nanodegree

Image Classifier Application

REVIEW | CODE REVIEW 4 | HISTORY

▾ train.py 2

```
import argparse

```

SUGGESTION

The code is really well done. You can extract some of the helper functions in both train/predict scripts anc module.
This could further improve readability.

```
import torch
from torch import nn
from torch import optim
from torch.autograd import Variable
from torchvision import datasets, transforms, models
from collections import OrderedDict


def parse_args():
    """ Parses commandline arguments and assigns default values if needed"""
    parser = argparse.ArgumentParser(description="Ezhil's Image Classifier")
    parser.add_argument('data_dir', action="store")
    parser.add_argument('--save_dir', action='store',
                        default='checkpoint.pth', dest='checkpoint_dir')
    supported_arch = ['densenet121', 'vgg13', 'vgg16']
    parser.add_argument('--arch', action='store',
                        default='densenet121', choices=supported_arch, dest='a
    parser.add_argument('--learning_rate',
                        action='store', default='0.001',
                        dest='learning_rate', type=float)
    parser.add_argument('--hidden_units',
                        nargs='+', type=int, default=[], dest='hidden_units')
    parser.add_argument('--epochs',
                        action='store', dest='epochs', default=3, type=int)
    parser.add_argument('--gpu',
                        action='store_true', dest='gpu')

    args = parser.parse_args()
    if len(args.hidden_units) == 0:
        args.hidden_units = [400]

    print()
    print("Starting to train using these inputs")
    print("{0: <30}".format('Data Directory:'), args.data_dir)
    print("{0: <30}".format('Checkpoint Dir:'), args.checkpoint_dir)
    print("{0: <30}".format('arch:'), args.arch)
    print("{0: <30}".format('learning_rate:'), args.learning_rate)
    print("{0: <30}".format('hidden_units:'), args.hidden_units)
    print("{0: <30}".format('epochs:'), args.epochs)
    print("{0: <30}".format('gpu:'), args.gpu)
    print()
    print()
    return args


def load_data(data_dir):
    """Loads Data from the training and validation sets and do necessary trans
    train_dir = data_dir + '/train'
    valid_dir = data_dir + '/valid'

    train_data_transforms = transforms.Compose([transforms.RandomRotation(30),
                                                transforms.RandomResizedCrop(2
                                                transforms.RandomHorizontalFli
                                                transforms.ToTensor(),
                                                transforms.Normalize([0.485, 0
                                                                     [0.229, 0

    valid_data_transforms = transforms.Compose([transforms.Resize(225),
                                                transforms.CenterCrop(224),
                                                transforms.ToTensor(),
                                                transforms.Normalize([0.485, 0.
                                                                     [0.229, 0.

    train_image_data = datasets.ImageFolder(train_dir, transform=train_data_tra
    valid_image_data = datasets.ImageFolder(valid_dir, transform=valid_data_tra

    return train_image_data, valid_image_data


def get_loader(train_image_data, valid_image_data):
    """Creates training and validation DataLoaders"""
    train_loader = torch.utils.data.DataLoader(train_image_data, batch_size=64,
    valid_loader = torch.utils.data.DataLoader(valid_image_data, batch_size=32

    return train_loader, valid_loader


def create_classifier(hidden_units):
  layers = []
  total = len(hidden_units)
  for idx, features in enumerate(hidden_units):
    if idx+1 == total:
      layers.append(('output', nn.LogSoftmax(dim=1)))
    else:
      name = 'fc'+str(idx+1)
      layers.append((name, nn.Linear(features, hidden_units[idx+1])))
      if idx+2 < total:
        relu_name = 'relu'+str(idx+1)
        dropout_name = 'dropout'+str(idx+1)
        layers.append((relu_name, nn.ReLU()))
        layers.append((dropout_name, nn.Dropout(p=0.5)))
  print()
  print('Using the classifier as below')
  print(layers)
  print()
  return nn.Sequential(OrderedDict(layers))


def create_model(arch, hidden_units):
    """Creates model from the architecture and hidden units provided in the in
    model = getattr(models, arch)(pretrained=True)
    for param in model.parameters():
        param.requires_grad = False
    hidden_units.insert(0, model.classifier.in_features)
    hidden_units.append(102)
    classifier = create_classifier(hidden_units)
    model.classifier = classifier
    return model


def train(model, learning_rate, epochs, gpu):
    """Trains the model and prints the running
        and validation losses, validation accuracy"""
    criterion = nn.NLLLoss()
    optimizer = optim.Adam(model.classifier.parameters(), lr=learning_rate)
    if gpu:
```

SUGGESTION

Along with the user provided GPU check, you should also have a condition to check if GPU is available so a

```
        model.cuda()

    print_every = 40
    running_loss = 0
    step = 0

    for e in range(epochs):
        model.train()
        for ii, (inputs, labels) in enumerate(train_loader):
            step += 1
            optimizer.zero_grad()
            inputs, labels = Variable(inputs), Variable(labels)
            if gpu:
                inputs, labels = inputs.cuda(), labels.cuda()

            outputs = model.forward(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.data[0]

            if step % print_every == 0:
                model.eval()
                accuracy = 0
                valid_loss = 0
                for idx, (v_inputs, v_labels) in enumerate(valid_loader):
                    v_inputs, v_labels = \
                        Variable(v_inputs, volatile=True), Variable(v_labels, v
                    if gpu:
                        v_inputs, v_labels = v_inputs.cuda(), v_labels.cuda()
```

Rate this review

```
                    v_outputs = model.forward(v_inputs)
                    valid_loss += criterion(v_outputs, v_labels).data[0]
                    ps = torch.exp(v_outputs).data
                    equality = (v_labels.data == ps.max(1)[1])
                    accuracy += equality.type_as(torch.FloatTensor()).mean()
                print("Epoch: {}/{}.. ".format(e + 1, epochs),
                      "Training Loss: {:.3f}.. ".format(running_loss / print_ev
                      "Validation Loss: {:.3f}.. ".format(valid_loss / len(vali
                      "Validation Accuracy: {:.3f}".format(accuracy / len(valid
                running_loss = 0
                model.train()
    print('Model Training Done')
    return model


def save_model(model, train_image_data, args):
    """Saves the mode in the checkpoint directory file mentioned"""
    model.class_to_idx = train_image_data.class_to_idx
    checkpoint = {'hidden_units': args.hidden_units,
                  'class_to_idx': model.class_to_idx,
                  'state_dict': model.state_dict(),
                  'arch': args.arch,
                  'epochs': args.epochs,
                  'gpu': args.gpu,
                  'learning_rate': args.learning_rate,
                  'data_dir': args.data_dir}

    torch.save(checkpoint, args.checkpoint_dir)
    print('Model saved here: ', args.checkpoint_dir)


if __name__ == "__main__":
    args = parse_args()
    train_image_data, valid_image_data = load_data(args.data_dir)
    train_loader, valid_loader = get_loader(train_image_data, valid_image_data
    model = create_model(args.arch, args.hidden_units)
    train(model, args.learning_rate, args.epochs, args.gpu)
    save_model(model, train_image_data, args)
```

▸ predict.py 2

▸ README.md

RETURN TO PATH

Student FAQ