Image Classifier Application

train.py 2

predict.py 2

```
1 import argparse
2
```

AWESOME

The code is really well-written!

```
3 import torch
4 from torch import nn
5 from torch.autograd import Variable
6 from torchvision import models
7 from collections import OrderedDict
8 import numpy as np
9 from PIL import Image
10 import json
11
12
13 def parse_args():
14     """ Parses commandline arguments and assigns default values if needed"""
15     parser = argparse.ArgumentParser(description="Ezhil's Image Classifier")
16     parser.add_argument('inp', action="store")
17     parser.add_argument('checkpoint', action="store")
18     parser.add_argument('--top_k', action='store',
19                         default='3', dest='top_k', type=int)
20     parser.add_argument('--category_names', action='store',
21                         default='cat_to_name.json', dest='category_names')
22     parser.add_argument('--gpu',
23                         action='store_true', dest='gpu')
24
25     args = parser.parse_args()
26     print()
27     print("Starting to predict using these inputs")
28     print("{0: <30}".format('Image:'), args.inp)
29     print("{0: <30}".format('Checkpoint:'), args.checkpoint)
30     print("{0: <30}".format('top_k:'), args.top_k)
31     print("{0: <30}".format('category_names:'), args.category_names)
32     print("{0: <30}".format('gpu:'), args.gpu)
33     print()
34     print()
35     return args
36
37
38 def load_categories(category_names):
39     with open(category_names, 'r') as f:
40         cat_to_name = json.load(f)
41
42     return cat_to_name
43
44
45 def create_classifier(hidden_units):
46     layers = []
47     total = len(hidden_units)
48     if total < 2:
49         raise ("Invalid number of input layers. Has to be more than 1")
50     for idx, features in enumerate(hidden_units):
51         if (idx + 1 == total):
52             layers.append(('output', nn.LogSoftmax(dim=1)))
53         else:
54             name = 'fc' + str(idx + 1)
55             layers.append((name, nn.Linear(features, hidden_units[idx + 1])))
56             if (idx + 2 < total):
57                 relu_name = 'relu' + str(idx + 1)
58                 dropout_name = 'dropout' + str(idx + 1)
59                 layers.append((relu_name, nn.ReLU()))
```

Rate this review

```
                layers.append((dropout_name, nn.Dropout(p=0.5)))

    return nn.Sequential(OrderedDict(layers))


def load_checkpoint(checkpoint_file, gpu):
    if gpu:
        checkpoint = torch.load(checkpoint_file)
    else:
        checkpoint = torch.load(checkpoint_file,
                                map_location=lambda storage, loc: storage)
    model = getattr(models, checkpoint['arch'])(pretrained=True)
    hidden_units = checkpoint['hidden_units']
    classifier = create_classifier(hidden_units)
    model.classifier = classifier
    model.load_state_dict(checkpoint['state_dict'])
    model.class_to_idx = checkpoint['class_to_idx']

    return model


def process_image(img):
    """ Scales, crops, and normalizes a PIL image for a PyTorch model,
        returns an Numpy array
    """
    width, height = img.size
    if width > height:
        img.thumbnail((height, 256), Image.ANTIALIAS)
```

AWESOME

It's done wonderfully!

```
    else:
        img.thumbnail((256, width), Image.ANTIALIAS)
    half_the_width = img.size[0] / 2
    half_the_height = img.size[1] / 2
    img = img.crop((
        half_the_width - 112,
        half_the_height - 112,
        half_the_width + 112,
        half_the_height + 112
    ))

    np_image = np.array(img)
    img = np_image / 255
    img = (img - np.array([0.485, 0.456, 0.406])) / np.array([0.229, 0.224, 0.2

    img = img.transpose((2, 0, 1))
    img = np.expand_dims(img, axis=0)
    img = torch.from_numpy(img).float()
    img = Variable(img, volatile=True)
    return img


def predict(image_path, model, topk, gpu):
    """Predict the class (or classes) of an image using a trained deep learning
    model.eval()
    image = Image.open(image_path)
    image = process_image(image)
    output = model.forward(image)
    output = torch.exp(output).data
    probs, classes = output.topk(topk)
    ind = model.class_to_idx
    res = dict((v, k) for k, v in ind.items())
    classes = [res[x] for x in classes.cpu().numpy()[0]]
    return probs, classes


def display_result(inp, probs, classes, categories):
    print("=" * 7)
    print("RESULTS")
    print("=" * 7)
    print()
    print("{0: <30}".format('Input:'), inp)
    print("{0: <30}".format('Prediction:'), \
          categories.get(classes[0], classes[0]))
    print("{0: <30}".format('Probability:'), "{:.3f}%".format(probs[0][0] * 10
    print()
    print("***** topk classes and probabilities *****")
    print("{0: <5}".format("Rank"),
          "{0: <25}".format("Prediction"), "{0: <25}".format("Probability"))
    for i in range(len(classes)):
        print("{0: <5}".format(str(i + 1)),
              "{0: <25}".format( \
                  categories.get(classes[i], classes[i])), \
              "{0: <25}".format("{:.3f}%".format(probs[0][i] * 100)))
    print()


if __name__ == "__main__":
    args = parse_args()
    model = load_checkpoint(args.checkpoint, args.gpu)
    categories = load_categories(args.category_names)
    probs, classes = predict(args.inp, model, args.top_k, args.gpu)
```

```
150     display_result(args.inp, probs, classes, categories)
151
152
```

▸ README.md

RETURN TO PATH

Student FAQ