Create Your Own Image Classifier

REVIEW

CODE REVIEW 5

HISTORY

train.py 2

```
# ../ImageClassifier/flowers

__author__ = "Chris"

import argparse

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

from utilities import pipeline
from functions import build_classifier, train_model, save_model

# Create parser object and tell it what arguments to expect
parser = argparse.ArgumentParser(description='NN Trainer')
# ../ImageClassifier/flowers
# Specify argument for the training data directory
parser.add_argument('train_data_dir',
                    action='store',
                    help='Training data path')
# Specify argument for pretrained neural network
parser.add_argument('--arch',
                    action='store',
                    dest='pretrained_model',
```

```
                    default='vgg11',
                    help = 'Pretrained model to implement; defaults to VGG-11;
                    can work with VGG and Densenet architectures')
# Specify argument to store model checkpoint
parser.add_argument('--save_dir',
                    action='store',
                    dest='save_dir',
                    default='checkpoint.pth',
                    help='Location to save the model checkpoint')
# Specify argument for the learning rate
parser.add_argument('--learn_rate',
                    action='store',
                    dest='lr',
                    type=float,
                    default=0.03,
                    help='Learning rate for the training model; default 0.03; `
                    float type')
# Specify argument for the dropout probability
parser.add_argument('--dropout',
                    action='store',
                    dest='drop_out',
                    type=float,
                    default=0.02,
                    help='Dropout for training model; default 0.02; \
                    float type')
# Specify argument for the number of hiden units
parser.add_argument('--hidden_units',
                    action='store',
                    dest='hidden_units',
                    type=int,
                    default=500,
                    help='Number of hidden classifier units; default 500; \
                    int type')
# Specify argument for the number of classes to categorize
parser.add_argument('--classes',
                    action='store',
                    dest='classes',
                    type=int,
                    default=102,
                    help='Number of classes to categorize; default 102; \
                    int type')
# Specify argument for the number of epochs
parser.add_argument('--epochs',
                    action='store',
                    dest='epochs',
                    type=int,
                    default=1,
                    help='Number of training epochs; default 1; \
                    int type')
# Specify argument for GPU mode
parser.add_argument('--gpu',
                    action='store_true',
                    default=False,
                    help='Turn GPU mode on; default False; \
                    bool type')
```

AWESOME

- Nice job parsing in the arguments

```
# Assign arguments
results = parser.parse_args()
data_dir = results.train_data_dir
save_dir = results.save_dir
learning_rate = results.lr
dropout = results.drop_out
hidden_units = results.hidden_units
classes = results.classes
epochs = results.epochs
gpu = results.gpu
## Completion of argument assignment ##

## Define data and model specifics

# Data pipeline
train_loader, valid_loader, test_loader, train_data, valid_data, test_data = p
# Load model
# Returns the value of the named attribute of an object
pre_trained_model = results.pretrained_model
model = getattr(models, pre_trained_model)(pretrained=True)

# Build and attach a new classifier
input_units = model.classifier[0].in_features
```

AWESOME

- Nice job taking care of different input sizes

```
build_classifier(model, input_units, hidden_units, classes, dropout)
criterion = nn.NLLLoss()
optimizer = optim.Adam(model.classifier.parameters(), learning_rate)

# Train the model
model, optimizer = train_model(model,epochs,train_loader,valid_loader,criterio
# Test the model
test_model(model,test_loader,gpu)
# Save the model
save_model(loaded_model,train_data,optimizer,save_dir,epochs)

```

▸ functions.py 2

▸ predict.py 1

▸ utilities.py