SmallSEOTools

PLAGIARISM SCAN REPORT

Words	458	Date	December 13,2020
Characters	6237	Exclude URL	

0%	100%	0	9
Plagiarism	Unique	Plagiarized Sentences	Unique Sentences

Content Checked For Plagiarism

```
import matplotlib.pyplot as plt
import numpy as np
import torch
from torch import nn
from torch import optim
import torch.nn.functional as F
from torchvision import datasets, transforms, models
from PIL import Image
import time
import copy
import json
import argparse
#refer from https://docs.python.org/3/library/argparse.html
parser = argparse.ArgumentParser()
parser.add_argument('data_dir', action="store")
parser.add_argument('--save_dir', action="store", dest="save_dir", default='checkpoint')
parser.add_argument('--arch', action="store", dest="arch", default='vgg16')
parser.add_argument('--learning_rate', action="store", dest="learning_rate", default=0.001)
parser.add_argument('--hidden_units', action="store", dest="hidden_units", default=500)
parser.add_argument('--epochs', action="store", dest="epochs", default=3)
parser.add_argument('--gpu', action="store_const", dest="device", const="gpu", default='cpu')
arguments = parser.parse_args()
data_dir = arguments.data_dir
save_dir = arguments.save_dir
arch = arguments.arch
lr = arguments.learning_rate
hidden_units_count = arguments.hidden_units
epochs = arguments.epochs
device = arguments.device
data_dir = 'flowers'
train_dir = data_dir + '/train'
valid_dir = data_dir + '/valid'
test_dir = data_dir + '/test'
dirs=[train_dir,valid_dir,test_dir]
image_datasets=[]
dataloaders=[]
data_transforms=[
transforms.Compose([
transforms.RandomRotation(90),
```

```
transforms.RandomResizedCrop(255),
transforms.RandomHorizontalFlip(),
transforms.ToTensor(),
transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225)),]),
# Refrence for syntax and normalize value from here https://pytorch.org/docs/stable/torchvision/transforms.html
transforms.Compose([
transforms.Resize(255),
transforms.CenterCrop(224),
transforms.ToTensor(),
transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225)),]),
transforms.Compose([
transforms.Resize(255),
transforms.CenterCrop(224),
transforms.ToTensor(),
transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225)),])# Refrence for syntax and normalize value from
'Loading Image Data Exercise' of last chapter
]
for i in range(3):
datasets_var=datasets.ImageFolder(dirs[i], transform=data_transforms[i])
dataloaders.append(torch.utils.data.DataLoader(datasets_var, batch_size=64))
image_datasets.append(datasets_var)
with open('cat_to_name.json', 'r') as f:
cat_to_name = json.load(f)
model = models.vgg13(pretrained=True) if arch == 'vgg13' else models.vgg16(pretrained=True)
for param in model.parameters():
param.requires_grad = False
model.classifier = nn.Sequential( nn.Linear(25088, hidden_units_count), nn.ReLU(), nn.Linear(hidden_units_count,
102)) #I got reference of syntax from here:-> https://pytorch.org/docs/stable/generated/torch.nn.Sequential.html
criterion = nn.CrossEntropyLoss() # it is addition of LogSotfmax function and NLLloss funcion
optimizer = optim.SGD(model.parameters(), lr)
def my_nn(model, trainloader, testloader, epochs, criterion, optimizer, device):
device = 'cuda' if device == 'gpu' else 'cpu'
epochs = 30
steps = 0
train_losses, test_losses = [], []
for e in range(epochs):
running_loss = 0
for images, labels in trainloader:
optimizer.zero_grad()
log_ps = model(images)
loss = criterion(log_ps, labels)
loss.backward()
optimizer.step()
running_loss += loss.item()
else:
test_loss = 0
accuracy = 0
with torch.no_grad():
model.eval()
for images, labels in testloader:
log_ps = model(images)
test_loss += criterion(log_ps, labels)
ps = torch.exp(log_ps)
top_p, top_class = ps.topk(1, dim=1)
equals = top_class == labels.view(*top_class.shape)
accuracy += torch.mean(equals.type(torch.FloatTensor))
print("Epoch: {}/{}.. ".format(e+1, epochs),
"Training Loss: {:.3f}.. ".format(running_loss/len(trainloader)),
"Test Loss: {:.3f}.. ".format(test_loss/len(testloader)),
"Test Accuracy: {:.3f}".format(accuracy/len(testloader)))
model.train()
# This Function Is Refered from 'Inference and Validation Exersice' of pytorch lesson from neural network chapter
# Done: Do validation on the test set
def check_accuracy_of_my_nn(testloader, device):
test_loss = 0
```

```
accuracy = 0
device = 'cuda' if device == 'gpu' else 'cpu'
with torch.no_grad():
for images, labels in testloader:
log_ps = model(images)
test_loss += criterion(log_ps, labels)
ps = torch.exp(log_ps)
top_p, top_class = ps.topk(1, dim=1)
equals = top_class == labels.view(*top_class.shape)
accuracy += torch.mean(equals.type(torch.FloatTensor))
print('Accuracy: %d %%' % (100 * accuracy))
# This Function Is Refered from 'Inference and Validation Exersice' of pytorch lesson from neural network chapter
my_nn(model, dataloaders[0], dataloaders[1], epochs, criterion, optimizer, device)
check_accuracy_of_my_nn(dataloaders[0], 'gpu')
model.class_to_idx = image_datasets[0].class_to_idx
torch.save({'arch':'vgg16','state_dict':model.state_dict(),'class_to_idx':model.class_to_idx},'checkpoint.pth')
```

Sources	Similarity