SmallSEOTools

PLAGIARISM SCAN REPORT

Words	956	Date	December 13,2020
Characters	10563	Exclude URL	

0% Plagiarism	100% Unique	0 Plagiarized Sentences	34 Unique Sentences

Content Checked For Plagiarism

```
# Imports here
import matplotlib.pyplot as plt
%matplotlib inline
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
## Ruberic:Package Imports :-> All the necessary packages and modules are imported in the first cell of the notebook
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
## Ruberic:Training data augmentation :-> torchvision transforms are used to augment the training data with random scaling, rotations, mirroring, and/or cropping
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
]
## Ruberic:Data normalization:-> The training, validation, and testing data is appropriately cropped and normalized
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)
## Ruberic:Data batching :-> The data for each set is loaded with torchvision's DataLoader
## Ruberic:Data loading :-> The data for each set (train, validation, test) is loaded with torchvision's ImageFolder
with open('cat_to_name.json', 'r') as f:
cat_to_name = json.load(f)
model = models.vgg16(pretrained=True)
## Ruberic:Pretrained Network :-> A pretrained network such as VGG16 is loaded from torchvision.models and the parameters are frozen
hidden_units_count=600
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
## Ruberic:Feedforward Classifier :-> A new feedforward network is defined for use as a classifier using the features as input
criterion = nn.CrossEntropyLoss() # it is addition of LogSotfmax function and NLLloss funcion
optimizer = optim.SGD(model.parameters(), lr=0.01)
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()
## Ruberic:Validation Loss and Accuracy :-> During training, the validation loss and accuracy are displayed
my_nn(model, dataloaders[0], dataloaders[1], 3, criterion, optimizer, 'gpu')
# This Function Is Refered from 'Inference and Validation Exersice' of pytorch lesson from neural network chapter
## Ruberic:Training the network :-> The parameters of the feedforward classifier are appropriately trained, while the parameters of the feature network are left static
# 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))
check_accuracy_of_my_nn(dataloaders[0], 'gpu')# This Function Is Refered from 'Inference and Validation Exersice' of
pytorch lesson from neural network chapter
## Ruberic:Testing Accuracy :-> The network's accuracy is measured on the test data
model.class_to_idx = image_datasets[0].class_to_idx
# Done: Save the checkpoint
torch.save({'arch':'vgg16','state_dict':model.state_dict(),'class_to_idx':model.class_to_idx},'checkpoint.pth')
## Ruberic:Saving the model :-> The trained model is saved as a checkpoint along with associated hyperparameters
and the class_to_idx dictionary
checkpoint = torch.load('checkpoint.pth')
checkpoint.keys()
model = models.vgg16(pretrained=True)
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
## Ruberic:Loading checkpoints :-> There is a function that successfully loads a checkpoint and rebuilds the model
model.classifier = classifier
model.class_to_idx = checkpoint['class_to_idx']
model.load_state_dict(checkpoint['state_dict'])
def process_image(path_to_image):
image = Image.open(path_to_image)
#Refrence is from https://pillow.readthedocs.io/en/latest/reference/Image.html
image=image.resize((image.width*256//image.height,256)) if image.width > image.height else
image.resize((256,image.height*256//image.width))
#in above expression,I have resize the images where the shortest side is 256 pixels, keeping the aspect ratio.
return ((((np.array(image.crop(((image.width-224)/2,(image.height-224)/2,(image.width-224)/2+224, (image.height-
224)/2+224))))/255) # image converted to np array from PIL
- np.array([0.485, 0.456, 0.406]))#substracting mean from resulting np array
/np.array([0.229, 0.224, 0.225])#Dividing by Standard Deviation
).transpose((2, 0, 1)) #transposing it to make third channel (color channel) to first and remaining both maintaining the
sequence
process_image(train_dir+'/18/image_04266.jpg')
## Ruberic:Image Processing :-> The process_image function successfully converts a PIL image into an object that can
be used as input to a trained model
def imshow(image, ax=None, title=None):
"""Imshow for Tensor."""
if ax is None:
fig, ax = plt.subplots()
image = image.numpy().transpose((1, 2, 0))

# Undo preprocessing
mean = np.array([0.485, 0.456, 0.406])
std = np.array([0.229, 0.224, 0.225])
image = std * image + mean

image = np.clip(image, 0, 1)

ax.imshow(image)

return ax
def predict(image_path):
return (torch.exp(model.forward(
(torch.from_numpy( process_image(image_path)).type(torch.FloatTensor)).unsqueeze_(0) #Converting np array to float
tensor using from_numpy
))).topk(5) #considering top 5 result refer from here https://pytorch.org/docs/master/torch.html#torch.topk
predict(train_dir+'/18/image_04266.jpg')
## Ruberic:Class Prediction :-> The predict function successfully takes the path to an image and a checkpoint, then
returns the top K most probably classes for that image
def display_img(image_path):
```

```
category = [] #Empty list
name_list=[] #Empty list
flower=[] #Empty list
imshow(process_image(image_path), plt.subplot(2,1,1)) #Displaying Image
probs, classes = predict(image_path) #predicting image with top 5 results
for i in classes[0]:
flower.append(i) #adding new element to flower
for x in flower:
for category_name, category_value in checkpoint['class_to_idx'].items(): #We are getting category key and value from class_to_idx dictonary
category.append(category_name) if category_value == x else continue #Storing Category Names if present in flower list
for i in category:
name_list.append(cat_to_name[i]) #if category value is present in cat_to_name then storing the label name to store it into name_list so that it would be use to print on bar chart
## Ruberic:Sanity Checking with matplotlib :-> A matplotlib figure is created displaying an image and its associated top 5 most probable
```

Sources	Similarity