《深度学习及其应用》实验报告

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 实验名称 | 深度生成模型实验 | | | 实验序号 | 4 | 实验日期 | 20231109 |
| 姓 名 | **黄瑞** | **院系** | 计算机学院 | 班 级 | 21104502 | 学 号 | 2110450219 |
| 专 业 | **人工智能** | | | 指导教师 | 林涛 | 成 绩 | 合格 |
| **一、实验目的和要求**  目的：学习GAN深度学习网络结构，并利用深度学习框架搭建GAN网络模型，完成图像风格迁移实验  要求：  1、熟悉GAN网络的基本结构。  2、利用PYTorch深度学习框架，完成GAN网络的搭建；  3、利用GAN实现图像风格迁移实验；  4、完成实验报告内容，提交报告。 | | | | | | | |
| **二、实验步骤**    1、参考实验平台上的视频：  用账号登录到实训平台，选择课程《深度学习及其应用》  在教学平台上学习实验视频，地址：10.2.253.234    2、在实验平台上完成代码，实验平台地址：10.2.253.243:10010。建议大家手动完成代码，最后将代码在实验平台上运行。      参考代码：  # 1、使用到的工具包  # from \_\_future\_\_ import print\_function  import report\_utils  import warnings  warnings.resetwarnings = lambda: None  warnings.filterwarnings('ignore')  report = report\_utils.Report()  # PyTorch神经网络包  import torch  import torch.nn as nn  import torch.nn.functional as F  import torch.optim as optim  # 加载和显示图像  from PIL import Image  import matplotlib.pyplot as plt  %matplotlib inline  import torchvision.transforms as transforms # 处理PIL图像并转换成Torch张量  import torchvision.models as models # 训练或加载预先训练的模型  import copy  # 2、CUDA  device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")  ######################################################################  # 3、加载图像  imsize = 512 if torch.cuda.is\_available() else 256 # use small size if no gpu  loader = transforms.Compose([  transforms.Resize(imsize), # scale imported image  transforms.ToTensor()]) # transform it into a torch tensor  def image\_loader(image\_name):  image = Image.open(image\_name)  # fake batch dimension required to fit network's input dimensions  image = loader(image).unsqueeze(0)  return image.to(device, torch.float)  style\_img = image\_loader(style\_image)  content\_img = image\_loader(content\_image)  assert style\_img.size() == content\_img.size(), "we need to import style and content images of the same size"  ######################################################################  # 4、显示图像  unloader = transforms.ToPILImage() # reconvert into PIL image  plt.ion()  def imshow(tensor, title=None):  image = tensor.cpu().clone() # we clone the tensor to not do changes on it  image = image.squeeze(0) # remove the fake batch dimension  image = unloader(image)  plt.imshow(image)  if title is not None:  plt.title(title)  plt.pause(0.001) # pause a bit so that plots are updated  plt.rcParams['font.sans-serif'] = ['SimHei']  plt.rcParams['axes.unicode\_minus'] = False  plt.figure()  imshow(style\_img, title='style image')  plt.figure()  imshow(content\_img, title='content image')  ######################################################################  # 5、Content loss  class ContentLoss(nn.Module):  def \_\_init\_\_(self, target, ):  super(ContentLoss, self).\_\_init\_\_()  self.target = target.detach()  def forward(self, input):  self.loss = F.mse\_loss(input, self.target)  return input  ######################################################################  # 6、风格损失  def gram\_matrix(input):  a, b, c, d = input.size() # a=batch size(=1)  # b=number of feature maps  # (c,d)=dimensions of a f. map (N=c\*d)  features = input.view(a \* b, c \* d) # resise F\_XL into \hat F\_XL  G = torch.mm(features, features.t()) # compute the gram product  # we 'normalize' the values of the gram matrix  # by dividing by the number of element in each feature maps.  return G.div(a \* b \* c \* d)  ######################################################################  class StyleLoss(nn.Module):  def \_\_init\_\_(self, target\_feature):  super(StyleLoss, self).\_\_init\_\_()  self.target = gram\_matrix(target\_feature).detach()  def forward(self, input):  G = gram\_matrix(input)  self.loss = F.mse\_loss(G, self.target)  return input  # print('成功')  ######################################################################  # 7、加载神经网络  cnn = models.vgg19(pretrained=True).features.to(device).eval()  cnn\_normalization\_mean = torch.tensor([0.485, 0.456, 0.406]).to(device)  cnn\_normalization\_std = torch.tensor([0.229, 0.224, 0.225]).to(device)  # create a module to normalize input image so we can easily put it in a  # nn.Sequential  class Normalization(nn.Module):  def \_\_init\_\_(self, mean, std):  super(Normalization, self).\_\_init\_\_()  # .view the mean and std to make them [C x 1 x 1] so that they can  # directly work with image Tensor of shape [B x C x H x W].  # B is batch size. C is number of channels. H is height and W is width.  self.mean = torch.tensor(mean).view(-1, 1, 1)  self.std = torch.tensor(std).view(-1, 1, 1)  def forward(self, img):  # normalize img  return (img - self.mean) / self.std  ######################################################################  # desired depth layers to compute style/content losses :  # 7、加载神经网络  content\_layers\_default = ['conv\_4']  style\_layers\_default = ['conv\_1', 'conv\_2', 'conv\_3', 'conv\_4', 'conv\_5']  def get\_style\_model\_and\_losses(cnn, normalization\_mean, normalization\_std,  style\_img, content\_img,  content\_layers=content\_layers\_default,  style\_layers=style\_layers\_default):  cnn = copy.deepcopy(cnn)  # normalization module  normalization = Normalization(normalization\_mean, normalization\_std).to(device)  # just in order to have an iterable access to or list of content/syle  # losses  content\_losses = []  style\_losses = []  # assuming that cnn is a nn.Sequential, so we make a new nn.Sequential  # to put in modules that are supposed to be activated sequentially  model = nn.Sequential(normalization)  i = 0 # increment every time we see a conv  for layer in cnn.children():  if isinstance(layer, nn.Conv2d):  i += 1  name = 'conv\_{}'.format(i)  elif isinstance(layer, nn.ReLU):  name = 'relu\_{}'.format(i)  # The in-place version doesn't play very nicely with the ContentLoss  # and StyleLoss we insert below. So we replace with out-of-place  # ones here.  layer = nn.ReLU(inplace=False)  elif isinstance(layer, nn.MaxPool2d):  name = 'pool\_{}'.format(i)  elif isinstance(layer, nn.BatchNorm2d):  name = 'bn\_{}'.format(i)  else:  raise RuntimeError('Unrecognized layer: {}'.format(layer.\_\_class\_\_.\_\_name\_\_))  model.add\_module(name, layer)  if name in content\_layers:  # add content loss:  target = model(content\_img).detach()  content\_loss = ContentLoss(target)  model.add\_module("content\_loss\_{}".format(i), content\_loss)  content\_losses.append(content\_loss)  if name in style\_layers:  # add style loss:  target\_feature = model(style\_img).detach()  style\_loss = StyleLoss(target\_feature)  model.add\_module("style\_loss\_{}".format(i), style\_loss)  style\_losses.append(style\_loss)  # now we trim off the layers after the last content and style losses  for i in range(len(model) - 1, -1, -1):  if isinstance(model[i], ContentLoss) or isinstance(model[i], StyleLoss):  break  model = model[:(i + 1)]  return model, style\_losses, content\_losses  ##8、输入图像  # input\_img = content\_img.clone()  # plt.figure()  # imshow(input\_img, title='Input Image')  # 9、梯度下降  def get\_input\_optimizer(input\_img):  # this line to show that input is a parameter that requires a gradient  optimizer = optim.LBFGS([input\_img.requires\_grad\_()])  return optimizer  # 在每个步骤中，纠正图像以将其值保持在0-1之间。  def run\_style\_transfer(cnn, normalization\_mean, normalization\_std,  content\_img, style\_img, input\_img, num\_steps=90,  style\_weight=1000000, content\_weight=1):  """Run the style transfer."""  # print('Building the style transfer model..')  model, style\_losses, content\_losses = get\_style\_model\_and\_losses(cnn,  normalization\_mean, normalization\_std, style\_img,  content\_img)  optimizer = get\_input\_optimizer(input\_img)  # print('Optimizing..')  run = [0]  while run[0] <= num\_steps:  def closure():  # correct the values of updated input image  input\_img.data.clamp\_(0, 1)  optimizer.zero\_grad()  model(input\_img)  style\_score = 0  content\_score = 0  for sl in style\_losses:  style\_score += sl.loss  for cl in content\_losses:  content\_score += cl.loss  style\_score \*= style\_weight  content\_score \*= content\_weight  loss = style\_score + content\_score  loss.backward()  run[0] += 1  if run[0] % 30 == 0:  # print("run {}:".format(run))  # print('Style Loss : {:4f} Content Loss: {:4f}'.format(  # style\_score.item(), content\_score.item()))  print()  return style\_score + content\_score  optimizer.step(closure)  # a last correction...  input\_img.data.clamp\_(0, 1)  return input\_img  # 运行算法  output = run\_style\_transfer(cnn, cnn\_normalization\_mean, cnn\_normalization\_std,  content\_img, style\_img, content\_img)  plt.figure()  imshow(output, title='Neural Style')  plt.ioff()  plt.show()  # report.writeToHtml(reportFileName) | | | | | | | |
| **三、实验结果与分析**  1、将代码和图像迁移后的显示结果截图放到报告中。  风格迁移结果如下图所示: | | | | | | | |