高阶关联鬼成像与鬼识别

fivech

2025 年 7 月 15 日

摘要

摘要：

目录

Chapter 1

题意解析

1.1 题目：高阶关联成像与鬼识别

这里是 section 部分

1.1.1 目的

这里是 subsection 部分

1.2 目标定位

Chapter 2

实验原理

Chapter 3

附录

3.1 代码展示

3.1.1 网络模型

```python
import torch
import torch.nn as nn
class Generator(nn.Module):
    def __init__(self, nz=1024, ngf=64, nc=3):
        super(Generator, self).__init__()
        self.ngf = ngf
        self.nz = nz
        self.nc = nc
        self.main = nn.Sequential(
            nn.ConvTranspose2d( nz, ngf * 16, 4, 1, 0, bias=False),
            nn.BatchNorm2d(ngf * 16),
            nn.ReLU(True),

            nn.ConvTranspose2d(ngf * 16, ngf * 8, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf * 8),
            nn.ReLU(True),

```

```python
            nn.ConvTranspose2d(ngf * 8, ngf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf * 4),
            nn.ReLU(True),

            nn.ConvTranspose2d( ngf * 4, ngf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf * 2),
            nn.ReLU(True),

            nn.ConvTranspose2d( ngf * 2, ngf, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf),
            nn.ReLU(True),

            nn.ConvTranspose2d( ngf, nc, 4, 2, 1, bias=False),
            nn.Tanh()
        )
    def forward(self, input):
        return self.main(input)
```

Listing 3.1: 生成器

```python
class Discriminator(nn.Module):
    def __init__(self, ndf=64, nc=3):
        super(Discriminator, self).__init__()
        self.ndf = ndf
        self.nc = nc
        self.main = nn.Sequential(
            nn.Conv2d(nc, ndf, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(ndf, ndf * 2, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(ndf * 2, ndf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 4),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(ndf * 4, ndf * 8, 4, 2, 1, bias=False),
```

```
                nn.BatchNorm2d(ndf * 8),
                nn.LeakyReLU(0.2, inplace=True),

                nn.Conv2d(ndf * 8, ndf * 16, 4, 2, 1, bias=False),
                nn.BatchNorm2d(ndf * 16),
                nn.LeakyReLU(0.2, inplace=True),

                nn.Conv2d(ndf * 16, 1, 4, 1, 0, bias=False),
                nn.Sigmoid()
            )

    def forward(self, input):
        return self.main(input)
```

Listing 3.2: 判别器

3.1.2 图片预处理

```
import os
from PIL import Image
import numpy as np
from tqdm import tqdm

folder1 = r"E:\out1\sjcj3"
folder2 = r'E:\out1\sjcj1'
folder3 = r'E:\out1\sjcj2'
output_folder = r'F:\XMU\schoolB_2\CUPEC\out\out_2'
os.makedirs(output_folder, exist_ok=True)

target_size = (128, 128)
h, w = target_size

# 分别获取并排序三个文件夹的文件名
files1 = sorted(os.listdir(folder1))
files2 = sorted(os.listdir(folder2))
files3 = sorted(os.listdir(folder3))

total = min(len(files1), len(files2), len(files3))

```

```python
for i in tqdm(range(total)):
    fname1 = files1[i]
    fname2 = files2[i]
    fname3 = files3[i]

    path1 = os.path.join(folder1, fname1)
    path2 = os.path.join(folder2, fname2)
    path3 = os.path.join(folder3, fname3)

    try:
        # 打开图像并统一尺寸
        img1 = Image.open(path1).convert('RGB').resize(
    target_size)
        img2 = Image.open(path2).convert('RGB')
        img3 = Image.open(path3).convert('RGB')

        img1_np = np.array(img1)
        img2_np = np.array(img2)
        img3_np = np.array(img3)

        # 求和后除以1000
        r_sum_2 = int(img2_np[:, :, 0].sum()) / 1000.
        r_sum_3 = int(img3_np[:, :, 0].sum()) / 1000.

        img1_np[:, :, 1] = r_sum_2
        img1_np[:, :, 2] = r_sum_3

        out_path = os.path.join(output_folder, fname1)
        Image.fromarray(img1_np).save(out_path)

    except Exception as e:
        print(f"处理 {fname1} 时出错: {e}")
```

Listing 3.3: 图片处理

3.1.3 训练

```python
import argparse
import os
```

```python
import random
import torch
import torch.nn as nn
import torch.nn.parallel
import torch.backends.cudnn as cudnn
import torch.optim as optim
import torch.utils.data
import torchvision.datasets as dset
import torchvision.transforms as transforms
import torchvision.utils as vutils
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from PIL import Image
from torchvision.datasets import DatasetFolder
from net import Generator
from net import Discriminator
import os, sys
import shutil

from tensorboardX import SummaryWriter
writer = SummaryWriter('logs') ## 创建一个SummaryWriter的示例，
    默认目录名字为runs

if os.path.exists("out1"):
    print("删除 out1 文件夹！")
    if sys.platform.startswith("win"):
        shutil.rmtree("./out1")
    else:
        os.system("rm -r ./out1")

print("创建 out1 文件夹！")
os.mkdir("./out1")

## 基本参数配置
# 数据集所在路径
dataroot = r"F:\XMU\schoolB_2\CUPEC\out"
# 数据加载的进程数
workers = 0
```

```python
# Batch size 大小
batch_size = 64
# 图片大小
image_size = 128
# 图片的通道数
nc = 1
# 尺寸
sizex = 32
sizey = 32
# 向量维度
nz = sizex * sizey
# 生成器特征图通道数量单位
ngf = 64
# 判别器特征图通道数量单位
ndf = 64

# 损失函数
criterion = nn.BCELoss()
# 真假标签
real_label = 1.0
fake_label = 0.0
# 是否使用GPU训练
ngpu = 1
device = torch.device("cuda:0" if (torch.cuda.is_available()
    and ngpu > 0) else "cpu")
# 创建生成器与判别器
netG = Generator(nz=nz, ngf=ngf, nc=nc).to(device)
netD = Discriminator(ndf=ndf, nc=nc).to(device)
# G和D的优化器，使用Adam
# Adam学习率与动量参数
lr = 0.0003
beta1 = 0.5
optimizerD = optim.Adam(netD.parameters(), lr=lr, betas=(beta1,
     0.999))
optimizerG = optim.Adam(netG.parameters(), lr=lr, betas=(beta1,
     0.999))

# 缓存生成结果
img_list = []
```

```python
# 损失变量
G_losses = []
D_losses = []

# batch变量
iters = 0

## 读取数据
dataset = dset.ImageFolder(root=dataroot,
                           transform=transforms.Compose([
                               transforms.Resize(image_size),
                               transforms.CenterCrop(image_size),
                               transforms.ToTensor()
                           ]))

print(f"Found {len(dataset)} .bmp images.")

dataloader = torch.utils.data.DataLoader(
    dataset, batch_size=batch_size,
    shuffle=True, num_workers=workers,
    drop_last=True  # 丢弃最后不足 batch_size 的 batch
)

# 多GPU训练
if (device.type == 'cuda') and (ngpu > 1):
    netG = nn.DataParallel(netG, list(range(ngpu)))
if (device.type == 'cuda') and (ngpu > 1):
    netD = nn.DataParallel(netD, list(range(ngpu)))

# 总epochs
num_epochs = 20
## 模型缓存接口
if not os.path.exists('all_model/models'):
    os.mkdir('all_model/models')
print("Starting Training Loop...")
fixed_noise = torch.randn(64, nz, 1, 1, device=device)

# 真实图片
```

```python
real_image_PIL = Image.open("binary_black_white.png")  # 灰度图片 shape: (128,128)
real_image = np.array(real_image_PIL)
real_image_tensor = torch.from_numpy(real_image).float()
real_image_tensor = real_image_tensor.unsqueeze(0).expand(batch_size, -1, -1, -1)  # shape: [64, 1, 128, 128]
real_image_tensor = real_image_tensor.to(device)

for epoch in range(num_epochs):
    lossG = 0.0
    lossD = 0.0
    for i, data in enumerate(dataloader, 0):
        ## 训练真实图片
        netD.zero_grad()
        # real_data = data[0].to(device)
        real_data = real_image_tensor
        b_size = batch_size  # real_data.size(0)
        label = torch.full((b_size,), real_label, device=device)
        output = netD(real_data).view(-1)
        # 计算真实图片损失，梯度反向传播
        errD_real = criterion(output, label)
        errD_real.backward()
        D_x = output.mean().item()

        ## 训练生成图片
        # 产生latent vectors (初始化)
        noise = torch.randn(b_size, nz, 1, 1, device=device)
        # 提取 R G B 通道的值
        r_channel = data[0][:, 0, :, :]  # R 通道 (batch_size, sizex, sizey)
        xl = 64 - sizex // 2
        xr = 64 + sizex // 2
        yl = 64 - sizey // 2
        yr = 64 + sizey // 2
        r_crop = r_channel[:, xl:xr, yl:yr]
        r_flat = r_crop.reshape(batch_size, sizex * sizey)
        r_norm = (r_flat - r_flat.min(dim=1, keepdim=True)[0]) / (
```

```python
                    r_flat.max(dim=1, keepdim=True)[0] - r_flat.min(dim=1, keepdim=True)[0] + 1e-8)
        g_values = data[0][:, 1, 0, 0]  # G 通道的 (0,0) 值 (batch_size,)
        b_values = data[0][:, 2, 0, 0]  # B 通道的 (0,0) 值 (batch_size,)
        # 将 R 通道的均值赋给 noise
        noise = r_norm.view(batch_size, nz, 1, 1)  # (batch_size,)
        # 替换 sizex * sizey 维度的后两个值为 G 和 B
        noise[:, -2, 0, 0] = g_values / 10.  # 倒数第二个位置赋 G 值
        noise[:, -1, 0, 0] = b_values / 10. # 最后一个位置赋 B 值
        noise = noise.to(device)

        # 使用G生成图片
        fake = netG(noise)
        label.fill_(fake_label)
        output = netD(fake.detach()).view(-1)
        # 计算生成图片损失，梯度反向传播
        errD_fake = criterion(output, label)
        errD_fake.backward()
        D_G_z1 = output.mean().item()

        # 累加误差，参数更新
        errD = errD_real + errD_fake
        optimizerD.step()

        netG.zero_grad()
        label.fill_(real_label)  # 给生成图赋标签
        # 对生成图再进行一次判别
        output = netD(fake).view(-1)
        # 计算生成图片损失，梯度反向传播
        errG = criterion(output, label)
        errG.backward()
        D_G_z2 = output.mean().item()
        optimizerG.step()

```

```python
        # 输出训练状态
        if i % 50 == 0:
            print('[%d/%d][%d/%d]\tLoss_D: %.4f\tLoss_G: %.4f\tD(x): %.4f\tD(G(z)): %.4f / %.4f'
                  % (epoch, num_epochs, i, len(dataloader),
                     errD.item(), errG.item(), D_x, D_G_z1, D_G_z2))

        # 存储损失
        lossG = lossG + errG.item()  # 累加batch损失
        lossD = lossD + errD.item()  # 累加batch损失

    writer.add_scalar('data/lossG', lossG, epoch)
    writer.add_scalar('data/lossD', lossD, epoch)
torch.save(netG, 'all_model/netG_16.pth')
torch.save(netD, 'all_model/netD.pth')
```

Listing 3.4: 训练