上海大学

SHANGHAI UNIVERSITY

课程设计（论文）

UNDERGRADUATE COURSE (THESIS)

题目:基于 GAN 的动漫头像生成

学院	计算机工程与科学学院
专业	计算机科学与技术
学号	20121034
学生姓名	胡才郁
指导教师	武星
起讫日期	2022.03.07 – 2022.06.04

目录

基于 GAN 的动漫头像生成

摘要

本文对于 Kaggle 动漫头像数据集，使用 GAN 生成对抗网络生成动漫头像。数据通过图像数据预处理后进行模型训练，在此过程中进行了可视化分析，并使用训练好的模型完成生成任务，并对于模型性能进行了评估。

关键词：生成，生成对抗网络，FID，生成器，判别器

GAN-based Animation Avatar Generation

ABSTRACT

This article uses the Kaggle Anime Faces dataset to generate anime avatars using GAN generative adversarial network. The data is preprocessed through the image data for model training. In the process, visual analysis is performed, and the trained model is used to complete the generation task, and the performance of the model is evaluated.

Keywords: Generative model, Generative Adversarial Network, FID, generator，discriminator

第 1 章 绪论

本章主要描述了本文主要进行的实验与工作，提出本文的实验内容与实验目标，并对于文章结构进行概括与分层。

§1.1 本文实验内容及目标

本文使用 GAN 对抗生成网络，处理动漫头像数据集，训练模型。此模型可以将输入的简单向量转化为出风格相似的动漫头像，以完成生成动漫头像的目标。

§1.1.1 实验内容

本文基于 GAN 对抗生成网络，具体实验内容有以下几个方面。

（1）动漫头像数据集的采集；

（2）动漫头像数据集的数据预处理；

（3）GAN 对抗生成网络的构建与训练过程可视化；

（4）模型评估；

（5）动漫头像的生成结果。

§1.1.2 实验目标

针对本文的研究内容，制定了以下几项目标：

（1）实现动漫图片的生成任务；

（2）可视化 GAN 对抗生成网络的训练过程；

（3）处理分析模型训练过程中遇到的困难；

（4）使用 FID 指标对于模型性能进行评估。

§1.2 本文组织结构

整篇论文分为六章。

第一章提出了本文的研究内容以及研究目标。

第二章主要介绍了动漫头像数据集的构建过程。

第三章首先介绍了图片的预处理过程，并且完成了预处理后的数据可视化。

第四章主要介绍了 GAN 对抗生成网络的模型架构，算法描述与训练过程。

第五章对于 GAN 生成的结果图片进行展示，并且分析了模型训练中遇到的困难并且对于训练所得的模型进行评估分析。

第六章对全文进行了总结，归纳了本文的主要工作。

第 2 章 动漫头像数据集的采集

本章具体描述了本文中数据集的来源以及数据集的类型，介绍了此数据集的构建方式，并且对于数据集的特点展开了分析。

§2.1 数据集的采集

本节介绍了动漫头像数据集的采集过程。

§2.1.1 数据集的来源

本文中采用的数据集来自于 Kaggle 中 Anime Faces 项目，本数据集共由 71314 张动漫头像组成，并且图像尺寸为 96*96。该数据集是由提供数据集者爬取动漫中的图片，并使用动漫面孔检测算法，对于脸部进行打框标定，截取所得。并提前进行过数据清洗，整理所得的数据集较为规整。

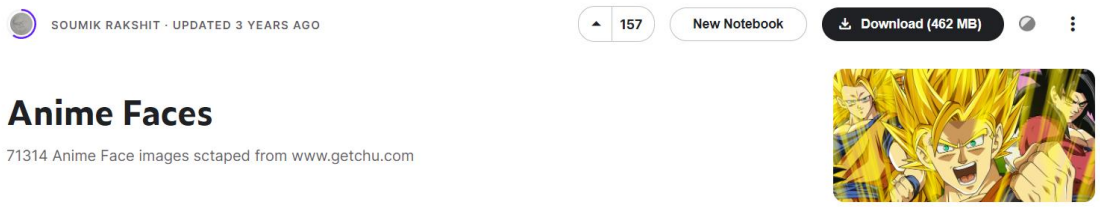

图 2-1 数据集在 Kaggle 中的来源

图 2-2 数据集提供者的数据采集过程

数据集提供者对于图片进行目标检测，给定固定的标定框大小，截取类似于上图中的图片，制作整理而成。

将获取到的图像数据集存放在相应的文件夹下，便于使用时读取。采集到的动漫头像类型与风格如下图所示：

图 2-3 动漫头像数据集

非结构化数据指的是既没有按照预定义的数据模型进行结构化，也没有按照预定义的方式组织的数据。这种类型的数据可以是人生成的，也可以是机器生成的，并且具有内部结构。

本文使用到的动漫头像并无标签标注，其数据分布相似，基本采集于风格相近的动漫。动漫头像为非结构化数据，此数据集的详细信息如下表所示：

表 2.1 动漫头像数据集信息

动漫头像数据集信息	
图像数量	71314
图像尺寸	96 * 96 * 3
数据类型	非结构化数据

§2.2 本章小结

在这一章中，介绍了本文中动漫头像数据集的采集。

训练数据对于模型是非常重要的，在改变模型架构来尝试提高模型预测准确率的之前，首先需要注意提高输入数据的质量，通过了解此动漫头像数据集中数据的构成与分布，有利于对于模型输入有一个更加直观的理解。

第 3 章 动漫头像数据预处理

在机器学习任务之中，如果将错误的、无意义的数据输入到机器学习模型之中，模型自然也一定会输出错误、无意义的结果。“Rubbish in, Rubbish out.”数据预处理为训练出较好模型的前提，本章对于图像数据预处理方法与流程展开了介绍。

§3.1 数据预处理概述

§3.1.1 数据清洗

数据集通常包含大量数据，这些数据可能以不易于使用的格式存储。因此，首先需要确保数据格式正确并符合规则集。数据清洗是指发现并纠正数据文件中可识别的错误的最后一道程序，包括检查数据一致性，处理无效值和缺失值等。数据清洗就是将“脏”的数据“洗掉”，指发现并纠正数据文件中可识别的错误的最后一道程序。

由于数据仓库中的数据是面向某一主题的数据的集合，这些数据从多个业务系统中抽取而来而且包含历史数据，这样就避免不了有的数据是错误数据、有的数据相互之间有冲突，这些错误的或有冲突的数据显然是我们不想要的。

数据清洗大多针对于结构化数据，而对于本文中处理的非结构化数据，并不与目标检测、图像分类等任务相同，此任务不存在标签，因此并不存在缺失值等指标。此处的数据清洗操作主要为检查图像的风格是否统一、图像的尺寸是否统一。

此数据集为 Kaggle 上公开数据集，数据清洗工作已经由数据集提供者提前完成，因此本章着重强调数据预处理过程。

§3.1.2 数据预处理

本文中，针对图像数据的预处理方式分为以下集中方式：

（1）图像缩放；

对于此图像数据集而言，由于数据量较大，共 71314 张图片，且尺寸为 96 * 96，神经网络训练难度较大，因此，将图像尺寸缩放为 64 * 64，帮助模型的训练，减少训练所需要的时间。

（2）图像矩阵形式存储；

神经网络可以接收的数据格式/类型是固定的，因此在训练过程之前，需要将动漫头像数据样本预处理成为可以被神经网络读取的格式类型。

将图像以矩阵的形式存储，作为向量输入神经网络之中，得以进行梯度下降反向传播等操作。

（3）图像数据标准化

数字图像处理中常用到图像数据标准化的操作，图像标准化是将数据通过去均值实现中心化的处理，根据凸优化理论与数据概率分布相关知识，数据中心化符合数据分布规律，减少模型学到数据分布的可能性，提升模型的泛化能力，更容易取得训练之后的泛化效果。

$$output = \frac{input - mean\left(input\right)}{std\left(input\right)}$$

公式中，$input$ 表示输入的图像像素值；$mean\left(input\right)$表示输入图像的像素均值。$std\left(input\right)$表示输入图像像素的标准差。经过标准化，图像像素被调整到[-1,1]区间内。

标准化处理后的图像，大小与通道数目与原图像保持一致。图像中的像素值经过标准化之后的分布没有改变，变的只是其值大小。以下为标准化前后训练资料的对比(上方为标准化前，下方为标准化后）。在进行图像标准化处理后，可以明显观察到进行标准化处理后，图像仍保留着动漫头像的特征。

图 3-1 图像标准化前后对比

（4）图像数据归一化

在神经网络中加入 Batch Norm 层，对于图像数据进行归一化操作。在使用梯度下降法训练神经网络的时候，归一化可以加快梯度下降的求解速度，进而加快网络的收敛。Batch Norm 层将隐藏层的输入分布从饱和区拉到了非饱和区，减小了梯度弥散，提升了训练速度，收敛过程大大加快，还能增加分类效果。BatchNorm 本身上也是一种正则的方式（主要缓解了梯度消失），可以代替其他正则方式如 dropout 等，有利于缓解模型的过拟合。

（5）图像增强

图像增广在对训练图像进行一系列的随机变化之后，生成相似但不同的训练样本，从而扩大了训练集的规模。此外，应用图像增广可以随机改变训练样本可以减少模型对某些属性的依赖，从而提高模型的泛化能力。例如，可以使用不同的方式裁剪图像，使得感兴趣的对象主体出现在不同的位置，减少模型对于对象出现位置的依赖。还可以调整亮度、颜色等因素来降低模型对颜色的敏感度。

经过对于动漫头像原始数据集的观察之后，发现原有图像数据集头像位置较为合适，无需进行局部放大或者缩小，因此并未选择使用随机裁剪的处理方法；又由于图像上下翻转对于生成动漫头像的意义不大，因此本文只采用了随机头像左右反转与图像亮度调整的数据增广方法。下图中可明显观察到第 1 张的亮度增加，后 3 张的亮度降低，并且第 4 张的头像进行了左右翻转。

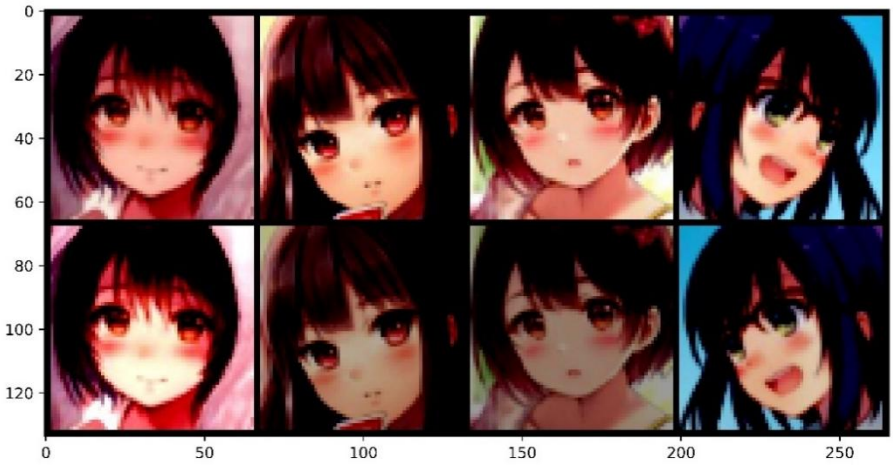

图 3-2 图像标准化前后对比

§3.2 本章小结

本章主要介绍了常用的数据清洗与预处理方法，并且针对本文中使用到的数据集的特点，应用了这些常用方法。

对于图像这种非结构化数据而言，进行数据预处理时，着重使用了本文使用的图像缩放、图像矩阵存储、图像数据标准化、归一化与图像增广等数据预处理方法。

第 4 章 动漫头像数据生成

本章是全文的重点章节，全面阐述了本文的工作内容。首先简单介绍了使用的开发平台与工具，随后将模型训练的过程完整的进行了描述，并将训练结果进行了展示。

§4.1 GAN 对抗生成网络

§4.1.1 生成问题概述

深度生成模型基本都是以某种方式寻找并表达（多变量）数据的概率分布。深度生成模型还被用于可视物体识别，语音识别，降维，信息获取，自然语言处理，机器人等各个应用领域。

生成模型的训练是一个非监督过程，输入只需要无标签的数据。除了可以生成数据，还可以用于半监督的学习。比如，先利用大量无标签数据训练好模型，然后利用模型去提取数据特征，之后用数据特征结合标签去训练最终的网络模型。

§4.1.2 GAN 概述

生成对抗网络 (GAN) 是由 Goodfellow 于 2014 年提出的一种对抗网络，是非监督学习的一种方法。这个网络框架包含两个部分，一个生成模型 (generative model) 和一个判别模型 (discriminative model)。其中，生成模型可以理解为一个伪造者，试图通过构造假的数据骗过判别模型的甄别；判别模型可以理解为一个警察，尽可能甄别数据是来自于真实样本还是伪造者构造的假数据。两个模型都通过不断的学习提高自己的能力，即生成模型希望生成更真的假数据骗过判别模型，而判别模型希望能学习如何更准确的识别生成模型的假数据。

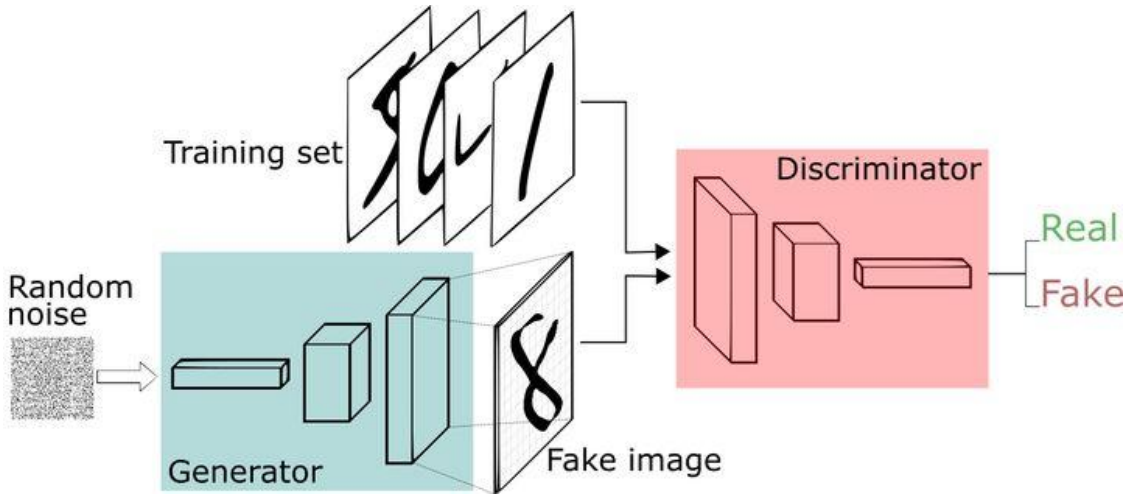

图 4-1 GAN 结构图

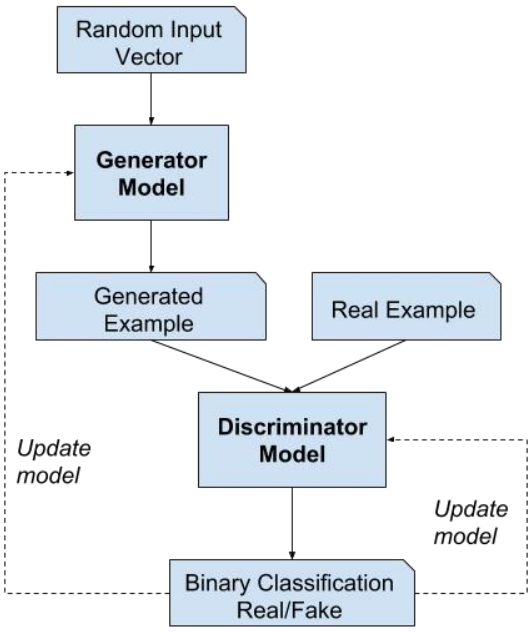

图 4-2 GAN 算法流程图

GAN 的算法流程图如上。在每一轮训练之中，分别进行以下操作：

（1）固定生成器 G，训练辨别器 D。

随机选取一些服从简单分布(例如高斯分布)的向量，把这些向量输入到生成器 G 之中，由生成器 G 产生图片，在训练开始阶段，生成器 G 的参数并未很好的适应生成图片的任务，因此图片质量较差。同时取训练集之中的真实图片。让辨别器 D 对于真实图片与生成器 G 产生的图片进行打分。训练辨别器 D 分辨真正的图片与生成器 G 生成的图片之间的区别。这是一个分类问题，由训练集中取样得到的真实图片标签为 1，由生成器 G 产生的图片标签为 0。此处辨别器 D 为一个二元分类的分类器，对于辨别器 D 进行训练。

在此过程中，辨别器 D 学习到了：给从训练集中的采样的真实图片高分，给由生成器 G 生成的图片低分。

（2）固定辨别器，训练生成器。

生成器 G 的目标是使它产生的这张图片，骗过辨别器 D，在辨别器 D 中的打分越高越好。

在此过程中，生成器 G 学习到了：生成的图片欺骗可以辨别器 D，获得辨别器 D 的高分。

以上两步骤在 GAN 对抗生成网络中不断循环，理想情况下，可以获得性能较好的生成器 G 与辨别器 D。

而再次选取一些服从简单分布(例如高斯分布)的向量，把这些向量输入训练结果较好的生成器 G 之中，便可以获得生成的图片。生成器 G 即为完成生成任务

的模型。

§4.2 模型训练

§4.2.1 开发环境简介

本文中完成模型训练的计算资源与开发工具与为谷歌 Colab 深度学习平台与深度学习框架 Pytorch。

Colaboratory 简称“Colab”，它由 Google Research 团队开发，任何人都可以通过浏览器编写和执行任意 Python 代码，尤其适合机器学习、数据分析、教育目的。Colab 是一种托管式 Jupyter 笔记本服务，用户无需设置，就可直接使用，还能免费使用 GPU/TPU 计算资源。本文中模型训练时用到的 GPU 资源均来自于 Colab。

Pytorch 是 torch 的 python 版本，是由 Facebook 开源的神经网络框架，专门针对 GPU 加速的深度神经网络（DNN）编程，其底层由 C++实现。Torch 是一个经典的对多维矩阵数据进行操作的张量库，在机器学习和其他数学密集型应用有广泛应用例如自然语言处理、计算机视觉等等。作为经典机器学习库 Torch 的端口，PyTorch 为 Python 语言使用者提供了舒适的代码选择。本文神经网络的架构与数据预处理的工作均依赖于 Pytorch。

§4.2.2 训练流程简介

训练流程将分为以下步骤进行构建：

（1）超参数设置。

配置训练中的全局超参数。

超参数是指在机器学习中，超参数是在开始学习过程之前设置值的参数，而不是通过训练得到的参数数据。由于使用的是 Colab 的 GPU 计算资源，内存资源有限，因此不可以选择过大的 batch_size，防止在训练过程中超出内存，从而导致训练失败。

表 4.1 全局超参数设置

动漫头像数据集信息		
参数名称	参数意义	参数值
batch_size	批量大小	64
lr	学习率	1e-4
n_epoch	训练轮数	20
z_dim	输入的向量维度	100

（2）构建生成器 G

此处生成器 G 的神经网络架构如下。

表 4.2 生成器 G 神经网络架构

生成器 G 神经网络架构	
参数名称	参数意义
输入尺寸	(batch, in_dim)
输出尺寸	(batch, 3, 64, 64)
激活函数	Relu
网络	卷积层

对于生成器 G 而言，它的任务为将输入的服从简单分布的向量通过神经网络转化为图片。因此选择对于处理图像效果更好的卷积层进行处理，在本文中，其具体架构如下图所示：

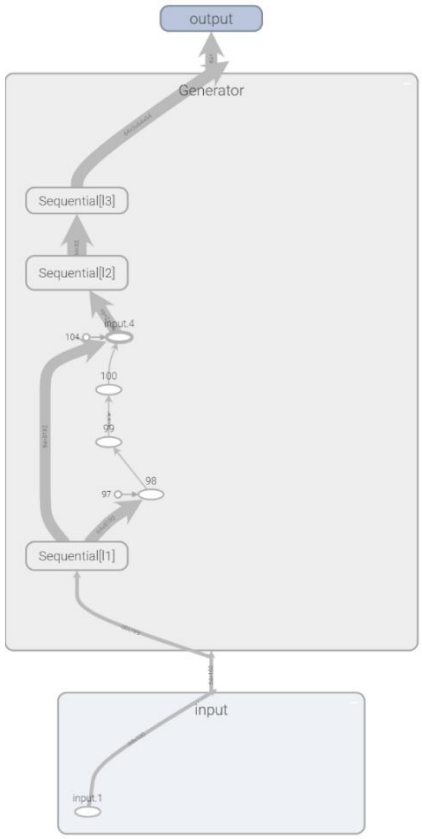

图 4-3 生成器 G 网络架构

（3）构建辨别器 D

此处辨别器 D 的神经网络架构如下。

表 4.3 辨别器 D 神经网络架构

辨别器 D 神经网络架构	
参数名称	参数内容
输入尺寸	(batch,3,64,64)
输出尺寸	(batch)
激活函数	Relu
网络	卷积层

对于辨别器 D 而言，它的任务为将分辨从训练集中采样得到的真实图像与由生成器产生的图像，并且对于这类图像进行二分类任务。网络采用卷积层可以更好的提取图像信息，因此选择对于处理图像效果更好的卷积层进行处理。其输入为图片向量，输出为分数。在本文中，其具体架构如下图所示：

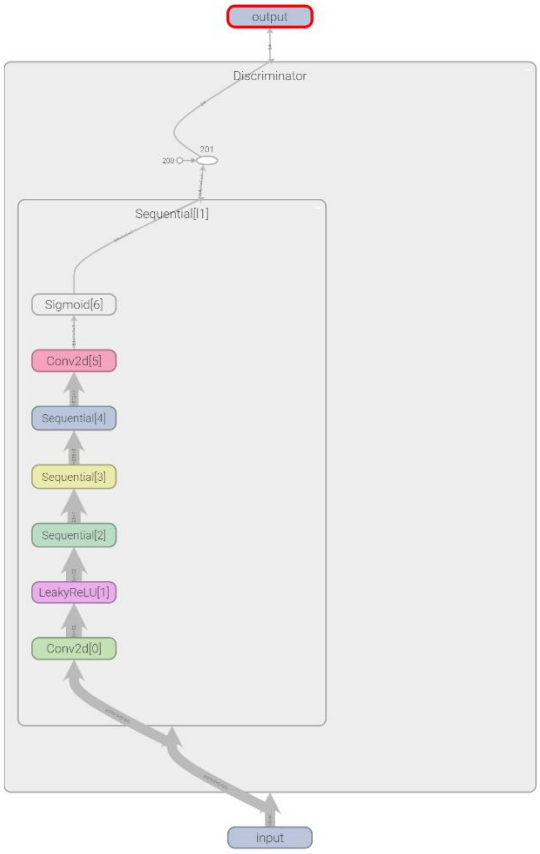

图 4-4 辨别器 D 结构图

（4）训练模型

表 4.4 参数优化配置

参数优化配置		
参数名称	生成器 G	辨别器 D
优化器	Adam	
损失函数	BCELoss	

对于整体的训练流程而言，可以将生成器 G 与辨别器 D 之间组合看作一个大的神经网络，此网络前半部分为生成器 G，负责将向量转化为图片，后半部分为辨别器 D，负责对于两类图片打分。因此，在训练过程中，分别对于生成器 G 与辨别器 D 进行参数更新，二者损失函数不同。

对于生成器 G 而言：

$$loss\_G = loss(f\_logit, r\_label)$$

其中损失函数为 BCELoss，f_logit为由辨别器 D 对于生成器所生成图片所打的标签，r_label为真实图片的标签，即为 1。生成器 G 的损失函数为由辨别器 D 对于生成器所生成图片所打的标签与真实图片标签的 BCELoss 值。

对于辨别器 D 而言：

$$loss\_D = (r\_loss + f\_loss)/2$$

其中：

$$r\_loss = loss(r\_logit, r\_label)$$

$$f\_loss = loss(f\_logit, f\_label)$$

r_loss与f_loss分别为真实的图片与虚假图片，通过辨别器 D 后得到辨别器的分类与实际分类标签的 BCELoss 值。

在获取到生成器 G 与辨别器 D 的损失函数表达方式之后，就可以在每一轮训练中进行正向传播与反向传播操作，更新生成器 G 与辨别器 D 的值。

§4.3 本章小结

本章主要介绍了本文的核心内容，动漫头像模型的训练配置与过程，介绍了生成问题的解决思路，并且介绍了 GAN 模型的实现过程。

GAN 对抗生成网络主要分为生成器 G 与辨别器 D 两部分，算法流程可以概括为在训练过程中轮流固定生成器 G，更新辨别器 D。固定辨别器 D，更新生成器 G。

第 5 章 训练过程与结果可视化

本章对训练过程中每一轮模型迭代进行了可视化分析，并且对于训练所得的模型性能进行了评估。将生成的最佳图片进行了展示。

§5.1 训练过程可视化

对于训练过程中，每一轮训练后，将随机简单向量输入到生成器 G 中，并观察生成结果，当 Epoch 为 1-4 时，分别进行了 1-4 轮训练，可以发现在训练不足时，动漫头像并不逼真，且并没有产生“分化”的效果。以第一轮训练结果为例，大部分头像均为紫红色，生成的头像之间较为相似。而随着训练轮数增多，头像之间开始出现分化的效果，并且动漫头像逐渐提高。

图 5- 1 Epoch 1-4 生成结果

而当 Epoch 为 13-16 时，分别进行了 13-16 轮训练，此时可以发现生成的动漫头像质量较高，较为逼真，其效果明显优于第 1-4 轮。并且随着轮数增多，每一轮变化情况逐渐减小，与真实图片相似程度提高，生成器 G 与辨别器 D 参数值接近最优解。

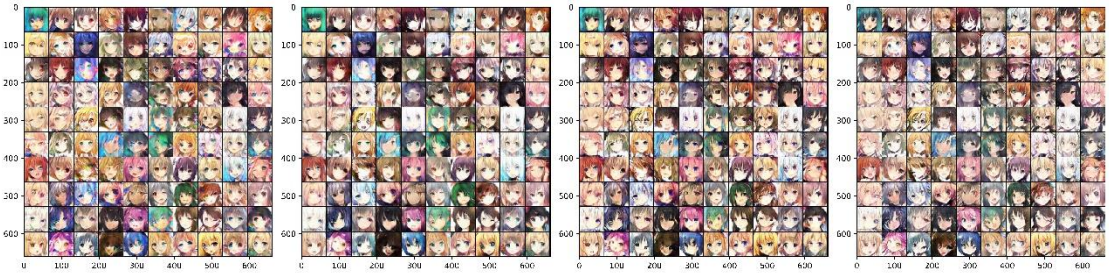

图 5- 2 Epoch 12-16 生成结果

§5.2 训练过程中的困难

在训练之中，由于生成器和辨别器不平衡，导致产生了局部最优解的情况。当训练到达第 17 轮时，生成的图像如下：

图 5-3 局部最优解图像情况

此时，生成器 G 的损失函数值明显大于正常情况，而辨别器 D 的损失函数值明显小于正常情况，生成的图像质量很差。

这是由于辨别器 D 为二分类分类器，由于对于真实图片输出 1，虚假图片输出 0，当迭代一定次数时，辨别器 D 网络层的参数逐步到达局部最优，参数值极大，这使得辨别器 D 的损失函数值极小，但由于 GAN 需要生成器 G 与辨别器 D 对抗进步，当辨别器 D 性能极强时，对于生成器 G 的训练没有帮助，因此生成器 G 的损失函数值升高，生成图片的质量很差。

表 5.1 局部最优与正常情况损失函数值对比

	正常情况	局部最优
Loss_G 数量级	10^{-2}	10^{-8}
Loss_D 数量级	10^{0}	10^{1}

§5.3 模型评估

评估和比较 GAN 并不容易，部分原因是缺乏明确的、在可比较概率模型中常用的似然方法。此动漫头像任务为无监督学习任务，对于生成问题而言，模型评估并非分类问题或回归问题，并无准确的标签或者真实值 Ground Truth，不可以直接计算误差值。

本文采用两种方式对于此 GAN 模型的结果进行评价,分别为主观观察评估与 FID 指标评估。

§5.3.1 主观观察评估

此处选取训练过程中，效果最好的一组结果进行展示：

图 5-4 生成结果可视化

观察生成结果，对于每一个生成的图片而言，由于其根源为服从高斯分布的随机向量，各个向量值并不相同，在训练前期由于神经网络参数并未调整至适合值，因此每个动漫头像之间区别并不明显，“分化”程度不高。而最终较好结果图片中，在未仔细观察时，生成的图像于真实图像之间差距不大，而在仔细观察时，可以发现，部分动漫头像脸部扭曲。不过生成器 G 学到了动漫人脸具有彩色头发，两只眼睛，一张嘴等特征。

§5.3.2 FID 指标

FID 是一种评价 GAN 的指标，于 2017 年提出，它的想法是这样的：分别把生成器生成的样本和判别器生成的样本送到分类器中（例如 Inception Net-V3 或者其他 CNN 等），抽取分类器的中间层的抽象特征，并假设该抽象特征符合多元高斯分布，估计生成样本高斯分布的均值和方差，以及训练样本和方差，计算两个高斯分布的弗雷歇距离，此距离值即 FID，公式如下：

$$|\mu_{data} - \mu_g| + \left(\Sigma_{data} + \Sigma_g - 2\left(\Sigma_{data}\Sigma_g\right)^{\frac{1}{2}}\right)$$

由于能力有限，本文并未直接训练 Inception Net 分类模型对于此模型进行评估，此处使用 Pytorch-FID 软件包中提供的函数，对于由生成器 G 生成的图片与训练集中真实数据集进行 FID 指标评估，得到模型的 Fid 分数为 140.9。

```
tcmalloc: large alloc 1168408576 bytes == 0xb91ee000 @  0x7fefalad31e7 0x7fef9f3640ce 0x7fef9f3bacf5 0x7fef9f3baf4f 0x7fef9f45d673 0x5936cc 0x548c51 0x5156
100% 1427/1427 [04:37<00:00,  5.15it/s]
100% 20/20 [00:04<00:00,  4.89it/s]
FID:  140.8437387218936
```

图 5-5 FID 评估结果

§5.4 本章小结

本章对于训练过程中生成器 G 的参数变化情况进行了分析，并且可视化展示了每一轮图片的生成结果。在训练过程中，生成器 G 在神经网络前向传播与反向传播更新参数的作用之下，学习到了动漫图像的泛化特点，例如头发为彩色、有两只眼睛、一张嘴等。并且最佳的图片尽管在细节上有所差距，但总体而言与真实图片差距较小。

第 6 章 总结与展望

本章对全文的主要工作进行总结，并提出需要进一步研究和改进之处。

§6.1 本文总结

本文对于数据集完成清洗与预处理工作后，输入到 GAN 对抗生成网络进行模型训练，并使用训练好的模型完成了动漫图像生成任务。

§6.1.1 本文的主要工作

本文主要研究的是基于 GAN 的动漫图像生成，主要工作内容有以下几个方面：

（1）数据集采集：采集来自 Kaggle 的数据集，并熟悉数据集提供者的数据清洗工作。

（2）数据预处理：使用图像标准化、归一化、图像增广等方法对采集到的数据进行预处理，便于后续训练。

（3）模型训练：训练 GAN 生成对抗网络，并分析与解释模型训练时遇到的局部最优困难。

（4）数据可视化：对于模型训练过程进行可视化。

（5）模型评估：对于训练好的模型分别进行定性与定量指标的评估。

§6.2 展望

虽然本文实现了基于 GAN 生成动漫图片，但仍存在不少需要改进的地方：

（1）生成的图片质量仍然不高。模型架构只选择了基础的 GAN 模型完成训练，并且由于算力不足没有长时间训练，并不知道模型是否已经完全收敛。

（2）本文对模型的定量评估指标不足。本文只选择了使用 FID 进行定量评估，并没有使用 IS 等其他对于生成模型的评估质量进行评估。

致谢

十分感谢在百忙之中抽出时间审阅本文的老师。也正有着老师的带领，为我打开了数据科学学习的大门。由于本人的学识和写作的水平有限，本文的写作中难免有僻陋，恳请老师多指教。

春季学期注定对我们来说是个难以忘记的学期，它有着两个学期的考试都堆在一起的紧张刺激，也有着疫情突如其来的措手不及。不知不觉，这也是我足不出校的整整第 90 天。查询了学校的健康之路我才发现，自 3 月 16 日以来，已经做过了 40 次核酸，以及不知道多少次抗原。

由于学校三学期制度，使得我们的节奏进度较快，导致自己没有充足的时间去细细探究每一个有趣的机器学习算法的的思想、理念、应用的延申和拓展，让自己可以熟练掌握其应用，并且举一反三，亲自动手实现。在疫情隔离期间，我们有更多的时间去了解课堂之外的事，比如在此期间我了解了很多深度学习模型，学习了开发技术，极大地提高了我的兴趣。

希望疫情早日结束，大家都能回到正常的生活轨迹之中。

参考文献

[1] 林野. 基于生成对抗网络的跨域人脸合成研究和应用[D].四川大学,2021.DOI:10.27342/d.cnki.gscdu.2021.000741.

[2] 梁俊杰,韦舰晶,蒋正锋.生成对抗网络 GAN 综述[J].计算机科学与探索,2020,14(01):1-17.

[3] 程显毅,谢璐,朱建新,胡彬,施佺.生成对抗网络 GAN 综述[J].计算机科学,2019,46(03):74-81.

[4] Goodfellow, Ian J., Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron C., and Bengio, Yoshua. Generative adversarial nets. NIPS, 2014.

附录：部分源程序清单

//1. 依赖模块导入

```
import os
import glob
import random
from datetime import datetime

import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
import torchvision.transforms as transforms
from torch import optim
from torch.autograd import Variable
from torch.utils.data import Dataset, DataLoader

import matplotlib.pyplot as plt
import numpy as np
import logging
from tqdm import tqdm
```

//2. 随机数种子设置

```
def same_seeds(seed):
    # Python built-in random module
    random.seed(seed)
    # Numpy
    np.random.seed(seed)
    # Torch
    torch.manual_seed(seed)
    if torch.cuda.is_available():
        torch.cuda.manual_seed(seed)
        torch.cuda.manual_seed_all(seed)
    torch.backends.cudnn.benchmark = False
    torch.backends.cudnn.deterministic = True

same_seeds(2022)
workspace_dir = '.'
```

//3. 数据集准备设置

```
class CrypkoDataset(Dataset):
    def __init__(self, fnames, transform):
        self.transform = transform
        self.fnames = fnames
        self.num_samples = len(self.fnames)

    def __getitem__(self,idx):
        fname = self.fnames[idx]
        img = torchvision.io.read_image(fname)
        img = self.transform(img)
        return img
```

```
    def __len__(self):
        return self.num_samples

def get_dataset(root):
    fnames = glob.glob(os.path.join(root, '*'))
    compose = [
        transforms.ToPILImage(),
        transforms.Resize((64, 64)),
        transforms.ToTensor(),
        transforms.Normalize(mean=(0.5, 0.5, 0.5), std=(0.5, 0.5, 0.5)),
    ]
    transform = transforms.Compose(compose)
    dataset = CrypkoDataset(fnames, transform)
    return dataset

def get_my_dataset(root):
    fnames = glob.glob(os.path.join(root, '*'))
    compose = [
        transforms.ToPILImage(),
        transforms.Resize((64, 64)),
        transforms.ToTensor(),
        transforms.Normalize(mean=(0.5, 0.5, 0.5), std=(0.5, 0.5, 0.5)),
        transforms.RandomHorizontalFlip(),
        transforms.ColorJitter(
    brightness=0.5, contrast=0, saturation=0, hue=0)
    ]
    transform = transforms.Compose(compose)
    dataset = CrypkoDataset(fnames, transform)
    return dataset
```

//4. 图像展示

```
temp_dataset = get_dataset(os.path.join("../data", 'faces'))
my_dataset = get_my_dataset(os.path.join("../data", 'faces'))
images = [temp_dataset[i] for i in range(4)]
my_images = [my_dataset[i] for i in range(4)]
images.extend(my_images)
grid_img = torchvision.utils.make_grid(images, nrow=4)
plt.figure(figsize=(10,10))
plt.imshow(grid_img.permute(1, 2, 0))
plt.savefig("change2.jpg",dpi=1080)
plt.show()
```

//5. 生成器构建

```
class Generator(nn.Module):
    """
    Input shape: (batch, in_dim)
    Output shape: (batch, 3, 64, 64)
    """
    def __init__(self, in_dim, feature_dim=64):
        super().__init__()

        #input: (batch, 100)
        self.l1 = nn.Sequential(
            nn.Linear(in_dim, feature_dim * 8 * 4 * 4, bias=False),
```

```
            nn.BatchNorm1d(feature_dim * 8 * 4 * 4),
            nn.ReLU()
        )
        self.l2 = nn.Sequential(
            self.dconv_bn_relu(feature_dim   *   8,   feature_dim   *   4),
#(batch, feature_dim * 16, 8, 8)
            self.dconv_bn_relu(feature_dim   *   4,   feature_dim   *   2),
#(batch, feature_dim * 16, 16, 16)
            self.dconv_bn_relu(feature_dim     *     2,     feature_dim),
#(batch, feature_dim * 16, 32, 32)
        )
        self.l3 = nn.Sequential(
            nn.ConvTranspose2d(feature_dim, 3, kernel_size=5, stride=2,
                               padding=2, output_padding=1, bias=False),
            nn.Tanh()
        )
        self.apply(weights_init)
    def dconv_bn_relu(self, in_dim, out_dim):
        return nn.Sequential(
            nn.ConvTranspose2d(in_dim, out_dim, kernel_size=5, stride=2,
                               padding=2,   output_padding=1,   bias=False),
#double height and width
            nn.BatchNorm2d(out_dim),
            nn.ReLU(True)
        )
    def forward(self, x):
        y = self.l1(x)
        y = y.view(y.size(0), -1, 4, 4)
        y = self.l2(y)
        y = self.l3(y)
        return y
```

//6. 辨别器构建

```
class Discriminator(nn.Module):
    """
    Input shape: (batch, 3, 64, 64)
    Output shape: (batch)
    """
    def __init__(self, in_dim, feature_dim=64):
        super(Discriminator, self).__init__()

        #input: (batch, 3, 64, 64)
        """
        NOTE FOR SETTING DISCRIMINATOR:

        Remove last sigmoid layer for WGAN
        """
        self.l1 = nn.Sequential(
            nn.Conv2d(in_dim,   feature_dim,   kernel_size=4,   stride=2,
padding=1), #(batch, 3, 32, 32)
            nn.LeakyReLU(0.2),
            self.conv_bn_lrelu(feature_dim,     feature_dim     *     2),
#(batch, 3, 16, 16)
            self.conv_bn_lrelu(feature_dim   *   2,   feature_dim   *   4),
```

```
#(batch, 3, 8, 8)
            self.conv_bn_lrelu(feature_dim   *   4,   feature_dim   *   8),
#(batch, 3, 4, 4)
            nn.Conv2d(feature_dim   *   8,   1,   kernel_size=4,   stride=1,
padding=0),
            nn.Sigmoid()
        )
        self.apply(weights_init)
    def conv_bn_lrelu(self, in_dim, out_dim):
        """
        NOTE FOR SETTING DISCRIMINATOR:

        You can't use nn.Batchnorm for WGAN-GP
        Use nn.InstanceNorm2d instead
        """

        return nn.Sequential(
            nn.Conv2d(in_dim, out_dim, 4, 2, 1),
            nn.BatchNorm2d(out_dim),
            nn.LeakyReLU(0.2),
        )
    def forward(self, x):
        y = self.l1(x)
        y = y.view(-1)
        return y
```

//7. 初始化模型参数

```
def weights_init(m):
    classname = m.__class__.__name__
    if classname.find('Conv') != -1:
        m.weight.data.normal_(0.0, 0.02)
    elif classname.find('BatchNorm') != -1:
        m.weight.data.normal_(1.0, 0.02)
        m.bias.data.fill_(0)
```

//8. 训练过程整合

```
class TrainerGAN():
    def __init__(self, config):
        self.config = config

        self.G = Generator(100)
        self.D = Discriminator(3)

        self.loss = nn.BCELoss()

        self.opt_D = torch.optim.Adam(self.D.parameters(),
lr=self.config["lr"], betas=(0.5, 0.999))
        self.opt_G = torch.optim.Adam(self.G.parameters(),
lr=self.config["lr"], betas=(0.5, 0.999))

        self.dataloader = None
        self.log_dir = os.path.join(self.config["workspace_dir"], 'logs')
        self.ckpt_dir = os.path.join(self.config["workspace_dir"],
'checkpoints')
```

```
        FORMAT = '%(asctime)s - %(levelname)s: %(message)s'
        logging.basicConfig(level=logging.INFO,
                            format=FORMAT,
                            datefmt='%Y-%m-%d %H:%M')

        self.steps = 0
        self.z_samples = Variable(torch.randn(100,
self.config["z_dim"])).cuda()

    def prepare_environment(self):
        """
        Use this funciton to prepare function
        """
        os.makedirs(self.log_dir, exist_ok=True)
        os.makedirs(self.ckpt_dir, exist_ok=True)

        # update dir by time
        time = datetime.now().strftime('%Y-%m-%d_%H-%M-%S')
        self.log_dir = os.path.join(self.log_dir,
time+f'_{self.config["model_type"]}')
        self.ckpt_dir = os.path.join(self.ckpt_dir,
time+f'_{self.config["model_type"]}')
        os.makedirs(self.log_dir)
        os.makedirs(self.ckpt_dir)

        # create dataset by the above function
        dataset = get_dataset(os.path.join("../data", 'faces'))
        self.dataloader = DataLoader(dataset,
batch_size=self.config["batch_size"], shuffle=True, num_workers=2)

        # model preparation
        self.G = self.G.cuda()
        self.D = self.D.cuda()
        self.G.train()
        self.D.train()
    def train(self):
        self.prepare_environment()

        for e, epoch in enumerate(range(self.config["n_epoch"])):
            progress_bar = tqdm(self.dataloader)
            progress_bar.set_description(f"Epoch {e+1}")
            for i, data in enumerate(progress_bar):
                imgs = data.cuda()
                bs = imgs.size(0)

                z = Variable(torch.randn(bs, self.config["z_dim"])).cuda()
                r_imgs = Variable(imgs).cuda() # real images
                f_imgs = self.G(z) # fake images
                r_label = torch.ones((bs)).cuda() # real labels (1)
                f_label = torch.zeros((bs)).cuda() # fake labels (0)


                # Discriminator forwarding
```

```
                r_logit = self.D(r_imgs)
                f_logit = self.D(f_imgs)


                # Loss for discriminator
                r_loss = self.loss(r_logit, r_label)
                f_loss = self.loss(f_logit, f_label)
                loss_D = (r_loss + f_loss) / 2

                # Discriminator backwarding
                self.D.zero_grad()
                loss_D.backward()
                self.opt_D.step()


                if self.steps % self.config["n_critic"] == 0:
                    # Generate some fake images.
                    z = Variable(torch.randn(bs,
self.config["z_dim"])).cuda()
                    f_imgs = self.G(z) # 将向量通过生成器，输入(batch, z_dim)
输出(batch, 3, 64, 64)大小的向量

                    # Generator forwarding
                    f_logit = self.D(f_imgs)

                    # Loss for the generator.
                    loss_G = self.loss(f_logit, r_label)

                    # Generator backwarding
                    self.G.zero_grad()
                    loss_G.backward()
                    self.opt_G.step()

                if self.steps % 10 == 0:
                    progress_bar.set_postfix(loss_G=loss_G.item(),
loss_D=loss_D.item())
                self.steps += 1

            self.G.eval()
            f_imgs_sample = (self.G(self.z_samples).data + 1) / 2.0
            filename = os.path.join(self.log_dir,
f'Epoch_{epoch+1:03d}.jpg')
            torchvision.utils.save_image(f_imgs_sample, filename, nrow=10)
            logging.info(f'Save some samples to {filename}.')

            # Show some images during training.
            grid_img = torchvision.utils.make_grid(f_imgs_sample.cpu(),
nrow=10)
            plt.figure(figsize=(10,10))
            plt.imshow(grid_img.permute(1, 2, 0))
            plt.show()

            self.G.train()
```

```
            if (e+1) % 5 == 0 or e == 0:
                # Save the checkpoints.
                torch.save(self.G.state_dict(),
os.path.join(self.ckpt_dir, f'G_{e}.pth'))
                torch.save(self.D.state_dict(),
os.path.join(self.ckpt_dir, f'D_{e}.pth'))

        logging.info('Finish training')

    def inference(self, G_path, n_generate=1000, n_output=30, show=False):

        self.G.load_state_dict(torch.load(G_path))
        self.G.cuda()
        self.G.eval()
        z = Variable(torch.randn(n_generate, self.config["z_dim"])).cuda()
        imgs = (self.G(z).data + 1) / 2.0

        os.makedirs('output', exist_ok=True)
        for i in range(n_generate):
            torchvision.utils.save_image(imgs[i], f'output/{i+1}.jpg')

        if show:
            row, col = n_output//10 + 1, 10
            grid_img = torchvision.utils.make_grid(imgs[:n_output].cpu(),
nrow=row)
            plt.figure(figsize=(row, col))
            plt.imshow(grid_img.permute(1, 2, 0))
            plt.show()
```

//9. 全局配置

```
config = {
    "model_type": "GAN",
    "batch_size": 64,
    "lr": 1e-4,
    "n_epoch": 100,
    "n_critic": 1,
    "z_dim": 100,
    "workspace_dir": workspace_dir,
}
```

//10. 开始训练

```
trainer = TrainerGAN(config)
trainer.train()
```