项目报告

项目名称	基于 ControlNet 引导的动漫线稿上色	
完成时间 2023 年 6 月 10 日	**总代码量** 400	**项目组长** 陈实
成员	**分工**	
姓名、学号 陈实， 20202251201	课题主导和任务分配、ControlNet、Stable Diffusion 基本理解、服务器环境搭建及项目运行、实现动漫线稿上色、建立数据集并进行训练、测试和分析、报告撰写及所有内容进行整合修改	
姓名、学号 刘浩宇， 20202251193	ControlNet、Stable Diffusion 基本原理和模型结构的理解、DragGAN 与 ControlNet 的区别总结、报告撰写	
姓名、学号 高晨超， 20202251059	学习 Attention 机制、分析 Diffusion 模型数学原理、了解 Text-to-Image 结构、报告撰写	
姓名、学号 陈戒， 20202251050	搜集 Fine-Tuning 相关方法的论文并进行学习与讲解、整理总结小组对于上色任务的见解、报告撰写	
姓名、学号 杨铠宁， 20202251202	DragGAN 与 ControlNet 的区别总结、研究源代码整体流程以及参数含义、实现动漫线稿上色、报告撰写	

目录

1 需求分析

1.1 背景

随着科技的发展，人工智能在各个领域中的应用越来越广泛，其中包括图像处理和计算机视觉。在这些领域中，线稿上色是一个重要的研究课题。线稿上色是指在黑白线条草图上填充适当颜色，以使图像更加生动和真实。这是一个在动画、漫画、游戏设计和其他视觉艺术领域中常见的任务。然而，这个任务通常需要大量的时间和专业知识，因此，开发能够自动进行线稿上色的算法和工具具有重要的实际意义。

近年来，深度学习技术在图像处理领域取得了显著的进展，其中一种被称为 ControlNet 的网络结构在线稿上色任务中表现出了优秀的性能。

ControlNet 是一种基于 Stable Diffusion 的神经网络结构，它可以根据输入的线稿生成具有丰富色彩和细节的彩色图像。ControlNet 的关键在于它的控制策略，它可以通过精细地控制生成过程，使得生成的彩色图像能够更好地符合线稿的结构和细节。

1.2 拟解决问题

1. 线稿上色的自动化：传统的线稿上色方法通常需要大量的人工操作，这使得这个过程既耗时又耗力。我们希望通过引入 ControlNet，实现线稿上色的自动化，从而大大提高上色的效率；

2. 线稿上色的质量提升：由于传统的线稿上色方法受到人工操作水平的限制，处理复杂线稿时，上色质量可能会受到影响。使用 ControlNet 可以提高线稿上色的质量，使生成的图像更加真实和生动；

3. 未提示区域的自主生成：在传统的线稿上色方法中，未提示区域的上色通常需要人工进行。ControlNet 有能力实现未提示区域的自主生成，从而使生成的图像更加完整和准确；

为解决以上几个问题，我们的课题目标主要包括以下几点：首先，理解动漫上色任务以及此论文研究的动机；其次，阅读并理解至少六篇与此课题相关的学术文章，学习文章的核心思想方法、总结这些文章的主要观点，并提出和总结我们小组关于上色任务的见解；最后，理解 ControlNet 论文的方法，理解 ControlNet 代码并复现，实现动漫线稿上色。

2 概要设计

为完成此次课题，我们小组计划分为以下几个阶段展开课题内容的研究：

阶段一：学习课题内容相关前置知识，在这些知识基础上，理解 ControlNet 论文动机和方法；

阶段二：学习 8 篇 ControlNet 论文中相关工作章节提及的学术文章，理解这些文章内容，提出并总结我们小组关于上色任务的见解；

阶段三：分析 ControlNet 源代码、理清逻辑结构，服务器上搭建 ControlNet 模型，并实现动漫线稿上色；

阶段四：尝试自己构建动漫线稿数据集、训练线稿模型，并进行模型测试以及评估分析。

2.1 相关工作

本节中，我们对 ControlNet 相关工作的学术文章的核心内容进行了理解。

2.1.1 Attention 机制

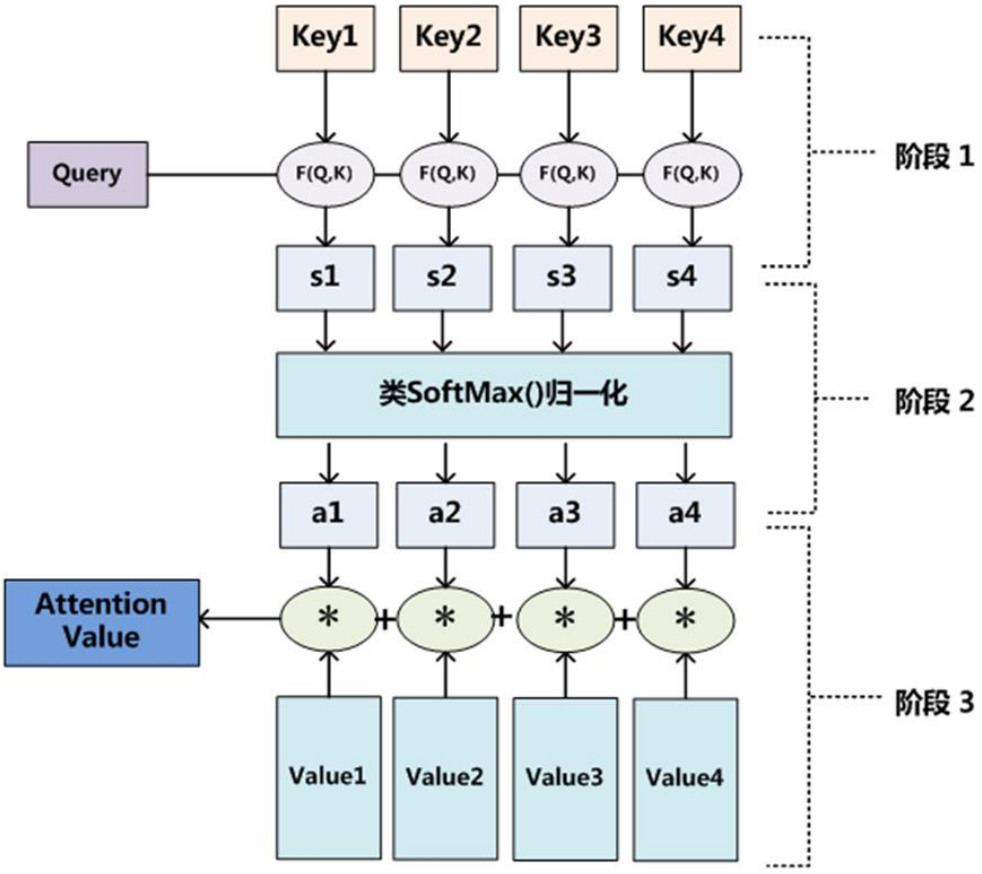

Fig. 2.1.1

在传统序列模型中，比如循环神经网络 RNN，模型会按照的固定方式处理输入序列中的每个元素。然而这种方式无法处理长序列或复杂的语义结构，而且会忽略一些与当前任务相关但位置较远的信息。而 Attention 机制的出现就是为了解决这个问题。Attention 机制通过计算输入序列中每个元素与当前时刻的注意力权重，然后根据这些权重对输入序列进行加权求和，从而得到一个上下文向量。这个上下文向量会被用于当前时刻的输出计算，使得模型能够更加关注与当前任务相关的信息。

Attention 机制包括三个关键步骤：计算注意力权重、加权求和并进行归一化处理和上下文向量的计算。

首先，计算注意力权重。通常使用“注意力打分函数”的方式来度量输入序列中每个元素与当前时刻的相关性。打分函数的输入是当前时刻的隐藏状态和输入序列中每个元素的表示，输出是一个注意力权重向量。

然后，根据注意力权重对输入序列进行加权求和。将注意力权重向量与输入序列中的每个元素进行按元素乘法，然后将乘积相加得到一个加权求和向量。这个加权求和向量反映了模型关注的信息。其中引入类似 SoftMax 的计算方法进行数值转换，一方面是为了归一化，将原始计算分值整理成权重和为 1 的概率分布；另一方面，通过 SoftMax 可以更加突出重要元素的权重。

最后，计算上下文向量。通过将加权求和向量经过一些变换如线性变换，得到上下文向量，该向量会作为当前时刻的上下文信息输入到模型的输出计算中。

通过以上三个步骤，Attention 机制能够根据输入序列的内容和当前任务的需要动态地调整注意力权重，使模型能够更加准确地捕捉到关键信息。优点在于它能够处理变长序列，关注与当前任务相关的信息，并且允许模型学习到不同位置之间的依赖关系。

2.1.2 Text-to-Image

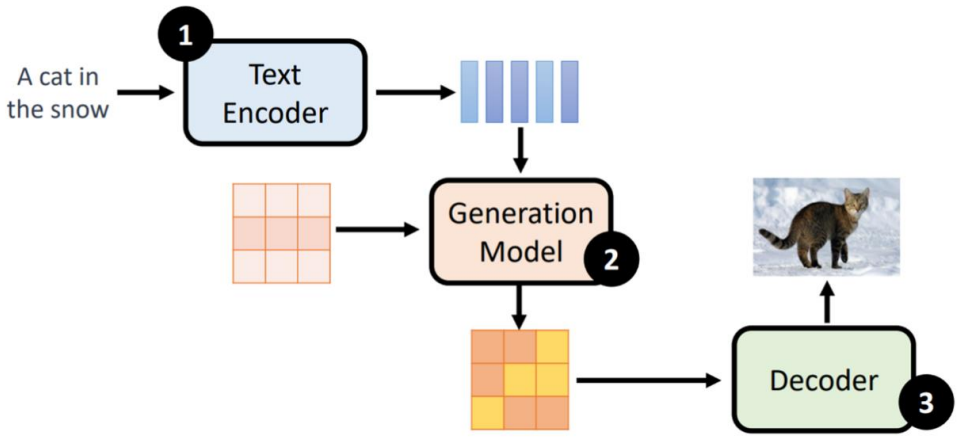

Fig. 2.1.2

ControlNet 整体的一个重要的功能是文生图。大框架由三个部分组成：Text Encoder，负责将非结构化、不可计算的文本内容，变成结构化可计算的向量也就是 Text Embedding；Generation Mode，ControlNet 使用的是 Latent Diffusion Model。如果扩散过程都是在像素空间进行的，它的计算量将会非常巨大，如果有一张 512x512 大小的三通道彩色图片，这将是一个将近 77 万维度的空间；Image Decoder，它负责将 Latent Representation 还原为原始图片，也就是最终产物。这需要用到 Auto-Encoder。通常这 3 个部分是分开训练，最后再组合起来。

2.1.3 CLIP 模型

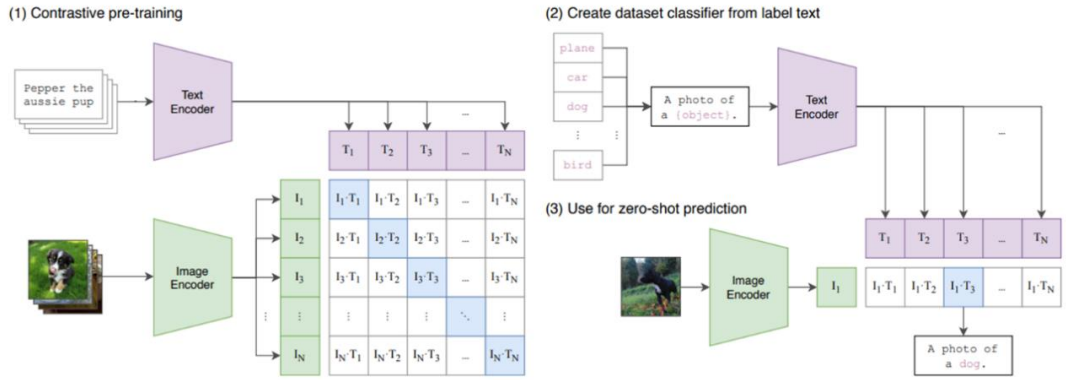

Fig. 2.1.3

CLIP 模型是用于联合处理图像和文本数据的多模态学习模型。CLIP 的目标是让模型能够理解图像和文本之间的语义关系，并在这种理解的基础上执行各种任务，如图像分类、图像生成描述和图像检索等。

CLIP 模型的核心思想是将图像和文本编码到同一个嵌入空间中，使得相似的图像和文本在嵌入空间中的距离更近，而不相似的图像和文本的距离更远。

在 CLIP 的训练中，模型同时接收图像和文本输入，并通过多层神经网络将它们映射到共享的嵌入空间中。通过调整模型的参数，CLIP 通过最小化相似图像和文本的嵌入向量之间的距离、最大化不相似图像和文本的距离，来学习这个嵌入空间。

CLIP 在文生成图任务中可以用于两个方面：

一，评估图像与文本之间的匹配度。通过计算给定图像和文本描述的嵌入向量在共享的嵌入空间中的相似度，可以衡量它们的语义关联程度。

二，生成图像的优化。文生成图任务中，生成的图像可能不完全符合输入的文本描述或者存在细节不准确的问题。通过 CLIP 可以将生成的图像与目标文本描述计算在嵌入空间中的距离，并使用这个距离作为损失函数进行优化。最小化这个损失函数，使生成的图像更贴合文本描述。

具体操作上，可以使用优化算法如梯度下降算法来调整生成的图像，使其在嵌入空间中与目标文本描述的距离逐渐减小。通过迭代优化过程，生成的图像会逐渐趋近于最优解，与输入的文本描述更匹配。

2.1.4 Diffusion 模型原理

Diffusion 模型的核心思想是通过不断迭代的过程，逐渐将简单的初始分布转化为目标分布。目标分布通常是我们希望模拟的真实数据分布。为了实现这一目标，Diffusion 模型引入了时间步长的概念。

假设我们有一个简单的初始分布 q(x)，通过迭代逼近目标分布 p(x)。在 Diffusion 模型的迭代过程中，每个时间步长都会对当前样本进行一次转移，转移的目标是通过马尔可夫链中的转移核函数来从当前状态采样下一个状态。

在每个时间步长中，Diffusion 模型的转移核函数可以使用扩散方程来描述。扩散方程是一个偏微分方程，表示样本在时间步长内根据当前梯度进行微小的扩散和漂移。扩散方程可以用来计算下一个时间步长的样本。Diffusion 模型使用以下形式的扩散方程：

$$dx(t) = \sigma(t) + dW(t) + \nabla \log p(x(t), t)dt$$

dx(t) 表示样本在时间 t 的微小变化量，σ(t) 是一个标准差函数，dW(t) 是布朗运动的增量，∇log p(x(t), t) 是目标分布 p(x, t)的梯度。这个方程包含了两部分：扩散项和漂移项。扩散项通过随机游走来模拟样本的扩散，漂移项通过梯度信息引导样本向着目标分布移动。每个时间步长内，通过求解扩散方程，我们可以得到下一个时间步长的样本。通过不断迭代，样本逐渐从初始分布 q(x) 向目标分布 p(x) 转变。迭代过程中，时间步长的选择对模型的性能有着重要的影响。较小的时间步长可以提高模型的精度，但会增加计算成本；较大的时间步长可以加快迭代速度，但可能会导致模型的不稳定性。

2.1.5 Auto-Encoder

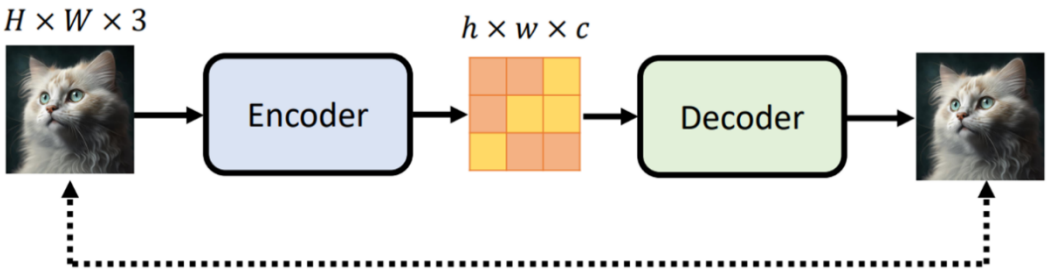

Fig. 2.1.5

Auto-Encoder 在 Text-to-Image 任务中可以用于学习文本的低维表示，并通过解码器生成相应的图像。首先，使用编码器部分将文本描述编码为低维表示。

编码器可以是一个神经网络，接受文本输入并将其映射到潜在空间中的低维向量。这个低维向量表示了文本描述的语义特征；接下来，使用解码器部分将编码表示解码为图像。解码器也可以是一个神经网络，接收编码表示并尝试生成与文本描述相匹配的图像。解码器的目标是尽可能地将编码表示解码为高质量、真实性的图像。在训练过程中，Auto-Encoder 的目标是最小化生成图像与目标图像之间的差异。通过计算重构图像与目标图像之间的重构误差，并使用反向传播算法和优化方法（如梯度下降），调整编码器和解码器的参数，使得重构误差尽可能地减小。训练完成后，可以使用编码器将文本描述编码为低维向量，然后使用解码器将低维向量解码为生成的图像。这样就可以通过输入文本描述，生成匹配的图像。

Auto-Encoder 在 Text-to-Image 任务中的应用可以使得文本描述与图像之间建立联系，并进行图像的生成和重构。它能够学习到文本描述的语义特征，并尝试生成与文本描述相匹配的图像。通过合理设计编码器和解码器的结构，以及优化训练过程，可以获得更好质量的生成图像。

2.1.6 Textual Inversion

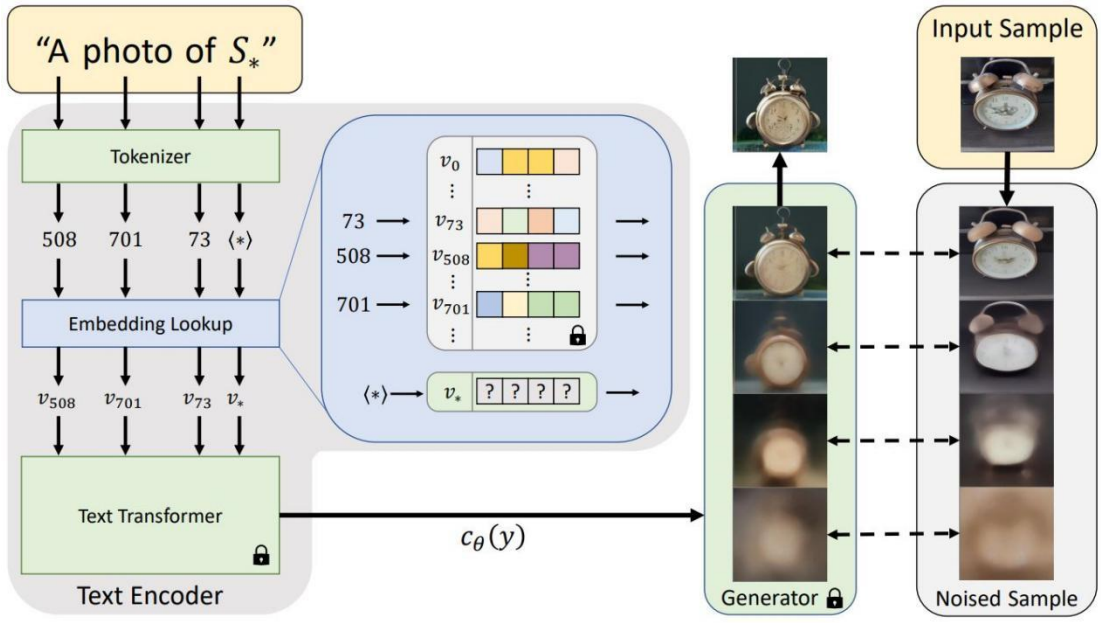

Fig. 2.1.6

Textual Inversion 是一种从少量示例图像中捕捉新概念的技术，它的核心思想是在文本编码器的 Embedding Space 中学习新的“伪词”。这些伪词并不是语言中真实存在的有意义的单词，而是模仿真实单词的一串字母。这些伪词可以在个性化图像生成的文本提示中使用，以实现对结果图像的精细控制。在扩散模型中使用文本提示之前，必须先将其处理为数值表示。这通常是通过将单词转换为 tokens 来完成的，每个 token 相当于模型字典中的一个条目。然后将这些条目转

换为 Embedding，一个特定的 token 的连续向量表示。最后，Embeddings 被转换为单个 Conditioning Code，用于指导生成模型。通过训练，Textual Inversion 为伪词找到了一个最合适的 Embedding，使用该 Embedding 即可指导原始的扩散模型输出与伪词所匹配的图像。Textual Inversion 的优点在于其轻量化，但总体效果上可能略逊于 DreamBooth。

2.1.7 DreamBooth

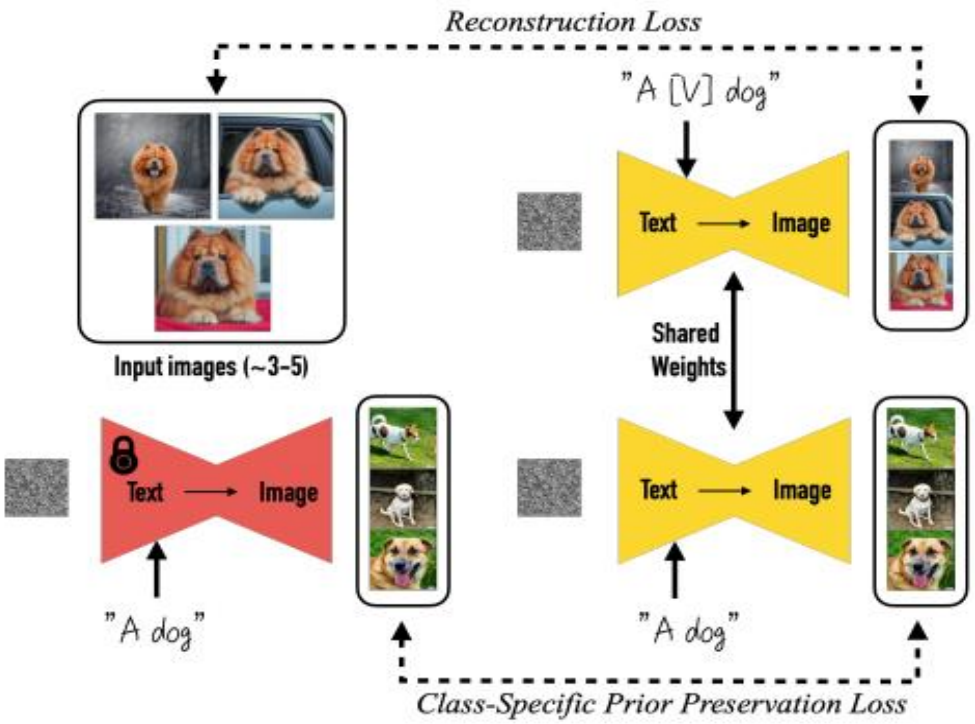

Fig. 2.1.7

DreamBooth 是一种以特定主题为驱动的微调方法，它可以生成与特定主题相关的各种场景，从而实现高质量的个性化图像生成。DreamBooth 的工作原理是通过训练，使模型能够生成与特定主题相关的各种场景，从而实现高质量的个性化图像生成。为了解决训练过程中可能出现的语言漂移和过拟合问题，DreamBooth 引入了类别特定的先验保护损失。这种损失函数可以让模型在训练过程中保持类别的多样性，并将各种姿势或环境连接到特定的主体上。这种方法的优点在于其生成的图像具有高度的保真度，能够保持主体的主要特征，但是由于它需要训练出一个完整的新模型，因此会占据相当大的存储空间。

2.1.8 Low-Rank Adaptation（LoRA）

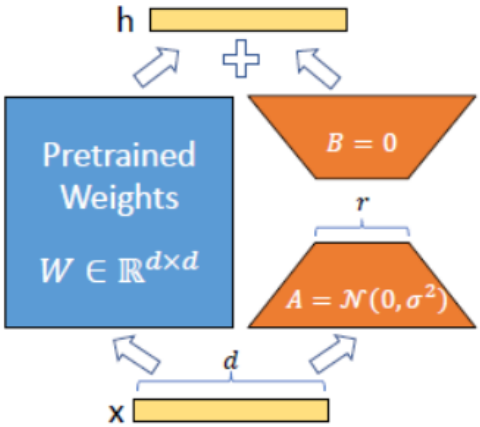

Fig. 2.1.8

LoRA 是一种高效的微调方法，它的核心思想是冻结预训练模型的权重，并添加一些小型网络，以实现高效的微调。LoRA 的关键在于，它并不直接修改预训练模型的权重，而是添加了一个新的网络部分，这个部分的输出与预训练模型的输出相加，得到最后经过调整后的输出。这种方法降低了微调过程中的可训练参数量，使得训练更高效。在 LoRA 的架构中，预训练模型的权重被冻结，不参与训练，而新添加的网络部分则是可训练的。这个新的网络部分通常比预训练模型的网络部分要小得多，因此可以大大降低训练的复杂性和计算需求。最后，预训练模型的输出和新添加网络部分的输出被相加，得到最后的输出。这种方法的优点在于它可以有效地微调模型，而不需要大量的计算资源，同时也不会对预训练模型的权重造成太大的影响，从而保持了预训练模型的优点。

2.2 方法实现

2.2.1 Stable Diffusion

在论文中，作者基于 Stable Diffusion 这一大模型进行微调。相较于传统的扩散模型，Stable Diffusion 主要解决了扩散模型速度的问题。传统扩散模型在反向扩散过程中需要输入完整尺寸的图片，导致在图片尺寸和时间步长 t 较大时，扩散速度非常缓慢。为了解决这一问题，Stable Diffusion 利用预训练好的自编码器将图像压缩到潜空间中，其中编码器负责压缩图像，解码器用于将潜空间数据还原为原始尺寸的图像。在潜空间中，经过压缩的图像进行扩散。在反向扩散过程中，作者改进了去噪过程中的 U-Net 结构，并引入了 Cross-Attention 注意力机制，以处理文本向量与当前图像的相关性。每次使用注意力机制，即一次图片信息和语义信息的耦合。此外，条件输入包括语义生成图像、文字生成图像、语言描述生成图像以及图像生成图像。这些输入通过不同的编码器模型转换为可用于

计算的向量，如文本通过文本编码器生成对应的嵌入。还引入了 Skip Connection 的概念，即通过从编码器阶段的每个编码器层复制一部分特征，并连接到解码器阶段的相应解码器层，以帮助解码器恢复丢失的低级细节和上下文信息，从而提高模型性能，为了融合条件输入。

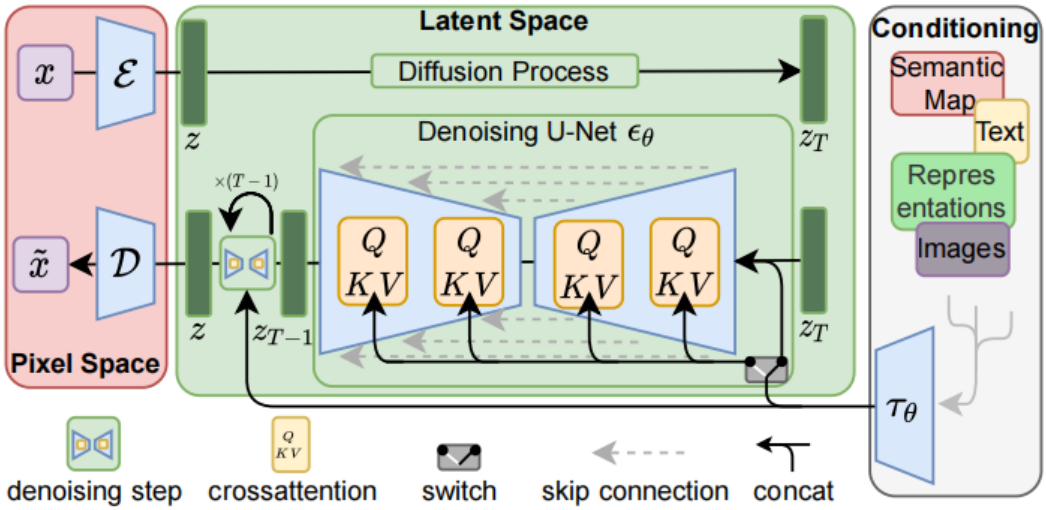

Fig. 2.2.1

2.2.2 ControlNet 基本认识

ControlNet 通过控制扩散的生成过程，来实现对线稿进行上色。它可以给扩散模型添加控制条件的神经网络结构，选择通过不同数据集训练出来的模型来控制图像生成。主要步骤就是输入原始图像以及文字提示 Prompt，首先模型将图像进行预处理得到特征信息，之后 Stable Diffusion 会根据输入 Prompt 进行图像生成，而 ControlNet 会根据预处理得到的特征信息，对 Stable Diffusion 生成过程进行控制。如果输入的图像可以直接作为特征信息，如本课题中的线稿图，则不需要经过预处理器。实际上通过 Canny 边缘检测提取出来的特征图是黑底白线，而线稿图是白底黑线，计算机中黑色部分是作为需要填充部分，白色部分作为边缘信息，因此我们输入线稿图后还需要对其进行反色处理。

2.2.3 ControlNet 原理

Stable Diffusion 是一个大型扩散模型，需要大量数据集和资源进行长时间训练才可能获得优秀的性能。在某些领域中，数据集可能很小且资源有限，因此需要利用训练好的 Stable Diffusion 模型进行微调。为此，作者提出了 ControlNet 这一小型模型用于控制大型模型。ControlNet 对 Stable Diffusion 进行部分复制得到 Locked Copy 和 Trainable Copy 两部分，其中 Locked Copy 的参数在训练过程中保持不变，而 Trainable Copy 的参数会被调整。实现的思路是利用 Locked Copy 保留原始网络的能力，同时训练 Trainable Copy，从小样本中学习生成的控制方

法。为了连接两个部分，引入了零卷积层，它的权重和偏差被初始化为零，保持零的状态，并在训练过程中开始更新。零卷积层的目的是确保在训练初期，模型的输入输出与未引入 ControlNet 的结果相同。因此，通过将原模型的输出与 ControlNet 的输出相加，即可得到该模型的输出。相对于从头开始训练网络，这种方法可以加快训练速度。如 Fig. 2.2.3.1 所示。ControlNet 部分的输入 c 可以是预处理之后得到的图像特征信息。

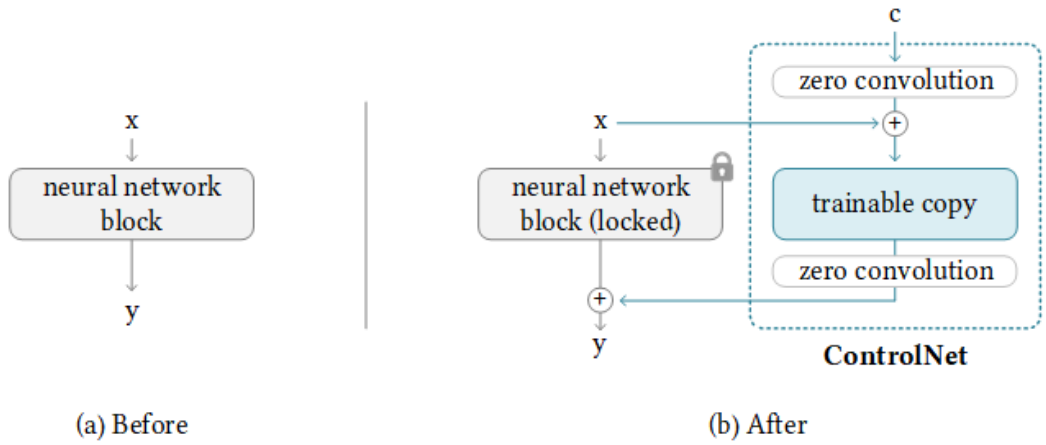

Fig. 2.2.3.1

施加控制条件之后，最后将原始网络的输出修改为：

$$\boldsymbol{yc} = F(\boldsymbol{x};\Theta) + Z(F(\boldsymbol{x} + Z(\boldsymbol{c};\Theta z_1);\Theta c);\Theta z_2)$$

其中零卷积层 Z 是初始化权重和偏差为 0，两层零卷积的参数为 $\{\Theta z_1, \Theta z_2\}$

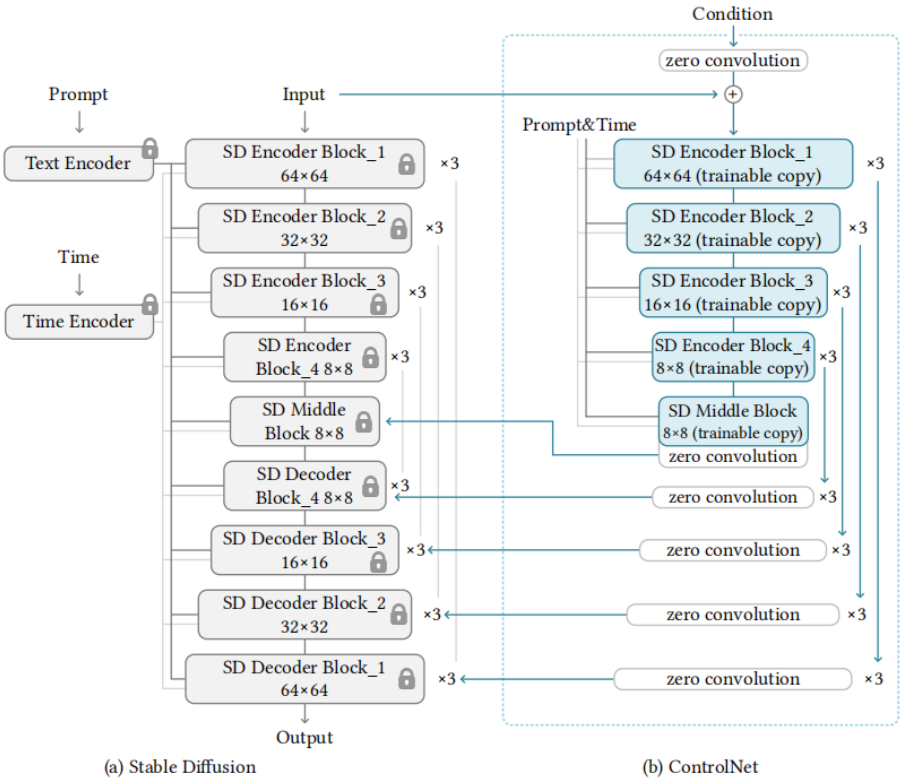

Fig. 2.2.3.2

如上图 Fig.2.2.3.2 所示，左侧部分为 Stable Diffusion，通过不断下采样和上采样实现，中间的连接层连接了相对应的上采样或下采样层；右侧部分为 ControlNet 新增的部分，编码器部分复制了左侧模型的结构和参数，但没有解码器部分。利用零卷积层将编码器部分的输出与左侧解码器部分相加。每个块包含了 ResNet 和 Transformer 的结构。对于较小的数据集，只需保留与中间块相连的零卷积层，并删除其他部分，以确保高效训练和减少参数更新；对于大型数据集，可以在训练过程中逐步解锁 Stable Diffusion 后半部分的解码器部分，即不再锁定该部分，以获得更好的学习效果。

初始化之后还未开始训练的 ControlNet 参数为：

$$Z(\boldsymbol{c};\Theta z_1) = 0$$
$$F(x + Z(\boldsymbol{c};\Theta z_1);\Theta c) = F(x;\Theta c) = F(x;\Theta)$$
$$Z(F(x + Z(\boldsymbol{c};\Theta z_1);\Theta c);\Theta z_2) = Z(F(x;\Theta c);\Theta z_2) = 0$$

从上式可以看出，ControlNet 未经训练的时候，输出为 0，故对原始网络是没有产生影响，保存了原始网络的性能。当开始训练后，ControlNet 的 Trainable Copy 部分的参数会被调整优化，从而实现对扩散模型的微调。

ControlNet 的训练目标函数为：

$$L = E_{z_0,t,c_t,c_r,\epsilon\sim N(0,1)}[\| \epsilon - \epsilon_\theta(z_t,t,c_t,c_f)) \|_2^2]$$

该目标函数与 Stable Diffusion 目标函数基本一致，将采样 z_t 使用网络 ϵ_θ 进行去噪之后，和原始图像经过网络 ϵ 后获得的潜向量进行 L_2 损失函数的计算。其中 E 是最大似然估计，因为 Stable Diffusion 模型是非线性模型，它的参数需要通过最大似然估计贝叶斯推断得到。目标函数中 c_t，c_f 是 ControlNet 添加的两个控制条件：文字提示 Prompt 以及任务相关的提示 Prompt（包括正向提示词和反向提示词）。

3　功能测试

3.1　参数介绍

ControlNet 复现完成后的 ui 界面中，包含如下图 Fig.3.1.1 所示的几个参数：

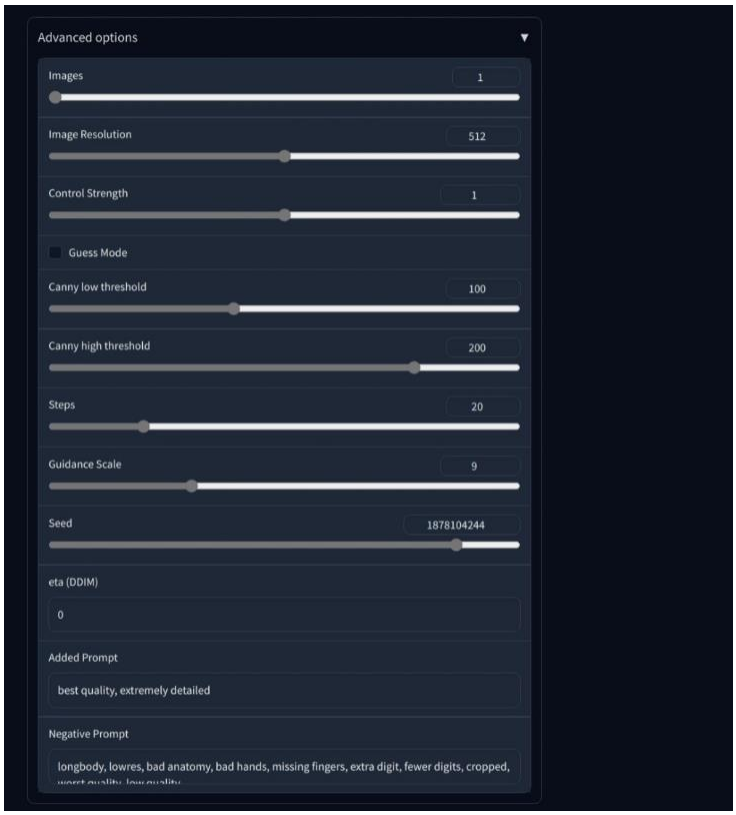

Fig. 3.1.1

（后文演示的图片均为我们小组自己进行调试生成，算作线稿上色复现）

采样迭代步数 Steps：对图像进行降噪的次数。如果降噪次数过少，可能噪声还未去掉就把图输出，导致最终图像非常模糊；如果降噪次数过多，则会计算时间会变长。

Fig. 3.1.2

分辨率 Resolution：最终图像的分辨率。

提示词相关性 Guidance Scale：与根据提示词生成的特征相乘的一个系数。当提示词数量过多时，可能存在提示词的特征无法表现，此时可通过调高这个参数解决。如果这个参数设置过高，就会出现颜色过饱和、图像模糊、线条过粗、形状扭曲等现象，类似于训练的时候的过拟合。

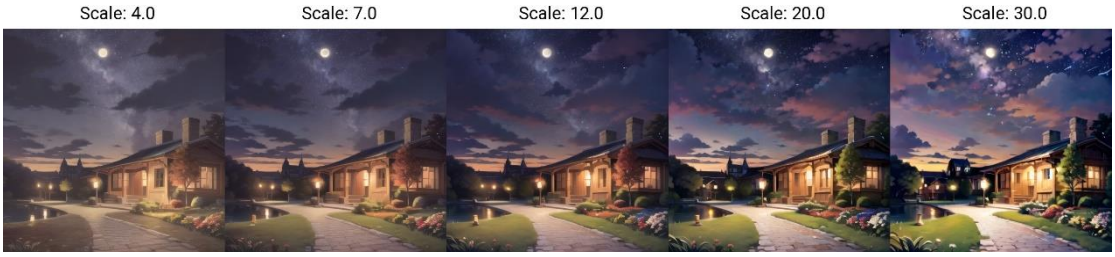

Fig. 3.1.3

随机噪声系数 eta：该参数设置为 0 时表示没有添加随机噪声，设置为 1 时表示在生成去噪的过程中会添加随机的噪声，使最终的结果更多样化，同时降低可能陷入局部最优的概率。该参数对最终生成结果的影响较小。

Fig. 3.1.4

边缘阈值 low/high threshold：仅在 Canny 预处理过程生效的参数，用于判断是否将该像素点作为图像边缘。不使用预处理器时，该参数不起作用。

无提示词模式 Guess Mode：开启后 ControlNet 可以在无提示词条件下对线稿进行上色。启用后，反向提示词将只作用于 Stable Diffusion，不作用于 ControlNet。同时 ControlNet 每层的权重逐渐递增。关闭后，ControlNet 每层的权重相同。

权重 Weight（即 Fig.3.1.1 中的 Control Strength）：影响最终图像与原图的契合程度。权重低，则最终图像与原图相似度低；权重高，则最终图像的边缘能很好的契合原图，同时也可能因为过于契合原图，导致部分物体的颜色和形状失真。

Fig. 3.1.5

3.2 ControlNet 代码逻辑结构

本节以 Canny 模型为例，代码的基本流程包括：将图像缩放到设定大小、图像预处理、处理提示词和权重、进行降噪、解码最终生成结果等。

将图像缩放到设定的大小。image_resolution 变量为设定的图像分辨率，输入的图像会经过 HWC3() 函数，将图像处理为标准的 RGB 三通道图像，且输入包含透明像素的图像的透明度会被去除。之后通过 resize_image() 函数缩放图像大小。图像的大小不能为任意数值，必须为 Stable Diffusion 潜空间大小的整数倍。

```
def process(input_image, prompt, a_prompt, n_prompt, num_samples, image_resolution, ddim_steps,
    with torch.no_grad():
        ## 缩放图像到目标分辨率
        img = resize_image(HWC3(input_image), image_resolution)
        H, W, C = img.shape
```

Fig. 3.2.1

```
def HWC3(x):
    assert x.dtype == np.uint8
    if x.ndim == 2:
        x = x[:, :, None]
    assert x.ndim == 3
    H, W, C = x.shape
    assert C == 1 or C == 3 or C == 4
    if C == 3:
        return x
    if C == 1:
        return np.concatenate([x, x, x], axis=2)
    if C == 4:
        color = x[:, :, 0:3].astype(np.float32)
        alpha = x[:, :, 3:4].astype(np.float32) / 255.0
        y = color * alpha + 255.0 * (1.0 - alpha)
        y = y.clip(0, 255).astype(np.uint8)
        return y


def resize_image(input_image, resolution):
    H, W, C = input_image.shape
    H = float(H)
    W = float(W)
    k = float(resolution) / min(H, W)
    H *= k
    W *= k
    H = int(np.round(H / 64.0)) * 64
    W = int(np.round(W / 64.0)) * 64
    img = cv2.resize(input_image, (W, H), interpolation=cv2.INTER_LANCZOS4 if k > 1 else cv2.INTER_AREA)
    return img
```

Fig. 3.2.2

图像预处理。程序直接调用 cv2 库函数来提取图像边缘进行预处理。线稿上色时，由于线稿可以看作是边缘图，因此可以选择不进行预处理。但仍然需要对图片进行反色操作。

```
        H, W, C = img.shape

        ## 检测图像边缘
        detected_map = apply_canny(img, low_threshold, high_threshold)
        detected_map = HWC3(detected_map)

        control = torch.from_numpy(detected_map.copy()).float().cuda() / 255.0
        control = torch.stack([control for _ in range(num_samples)], dim=0)
        control = einops.rearrange(control, 'b h w c -> b c h w').clone()

        if seed == -1:
            seed = random.randint(0, 65535)
        seed_everything(seed)

        if config.save_memory:
```

Fig. 3.2.3

```
import cv2


class CannyDetector:
    def __call__(self, img, low_threshold, high_threshold):
        return cv2.Canny(img, low_threshold, high_threshold)

```

Fig. 3.2.4

处理提示词和权重。cond 变量相当于正向提示词，un_cond 变量相当于反向提示词，guess_mode 表示无提示词模式。作用前文已阐述。

```
if seed == -1:
    seed = random.randint(0, 65535)
seed_everything(seed)

if config.save_memory:
    model.low_vram_shift(is_diffusing=False)

cond = {"c_concat": [control], "c_crossattn": [model.get_learned_conditioning([prompt + ', ' + a_prompt] * num_samples)]}
un_cond = {"c_concat": None if guess_mode else [control], "c_crossattn": [model.get_learned_conditioning([n_prompt] * num_samples)]}
shape = (4, H // 8, W // 8)

if config.save_memory:
    model.low_vram_shift(is_diffusing=True)

model.control_scales = [strength * (0.825 ** float(12 - i)) for i in range(13)] if guess_mode else ([strength] * 13)  # Magic number.
samples, intermediates = ddim_sampler.sample(ddim_steps, num_samples,
```

Fig. 3.2.5

```
    def apply_model(self, x_noisy, t, cond, *args, **kwargs):
        assert isinstance(cond, dict)
        diffusion_model = self.model.diffusion_model

        cond_txt = torch.cat(cond['c_crossattn'], 1)

        if cond['c_concat'] is None:
            eps = diffusion_model(x=x_noisy, timesteps=t, context=cond_txt, control=None, only_mid_control=self.only_mid_control)
        else:
            control = self.control_model(x=x_noisy, hint=torch.cat(cond['c_concat'], 1), timesteps=t, context=cond_txt)
            control = [c * scale for c, scale in zip(control, self.control_scales)]
            eps = diffusion_model(x=x_noisy, timesteps=t, context=cond_txt, control=control, only_mid_control=self.only_mid_control)

        return eps
```

Fig. 3.2.6

降噪采样。使用 DDIM 采样器，根据设定的步骤从一张噪声图中不断的根据设定的条件去掉噪声最终得到图像，然后解码生成的图像并返回，完成上色过程。

```
    model.control_scales = [strength * (0.825 ** float(12 - i)) for i in range(13)] if guess_mode else ([strength] * 13)  # Magic number. IDK why.
    samples, intermediates = ddim_sampler.sample(ddim_steps, num_samples,
                                                 shape, cond, verbose=False, eta=eta,
                                                 unconditional_guidance_scale=scale,
                                                 unconditional_conditioning=un_cond)

    if config.save_memory:
        model.low_vram_shift(is_diffusing=False)

    x_samples = model.decode_first_stage(samples)
    x_samples = (einops.rearrange(x_samples, 'b c h w -> b h w c') * 127.5 + 127.5).cpu().numpy().clip(0, 255).astype(np.uint8)

    results = [x_samples[i] for i in range(num_samples)]
return [255 - detected_map] + results
```

Fig. 3.2.7

在采样过程中，conditioning 变量表示生成的条件，包含我们输入的提示词和一些设定的参数。make_schedule() 函数是用来构造初始化采样器，构造完成后开始具体的采样操作。

```
if conditioning is not None:
    if isinstance(conditioning, dict):
        ctmp = conditioning[list(conditioning.keys())[0]]
        while isinstance(ctmp, list): ctmp = ctmp[0]
        cbs = ctmp.shape[0]
        if cbs != batch_size:
            print(f"Warning: Got {cbs} conditionings but batch-size is {batch_size}")

    elif isinstance(conditioning, list):
        for ctmp in conditioning:
            if ctmp.shape[0] != batch_size:
                print(f"Warning: Got {cbs} conditionings but batch-size is {batch_size}")

    else:
        if conditioning.shape[0] != batch_size:
            print(f"Warning: Got {conditioning.shape[0]} conditionings but batch-size is {batch_size}")

self.make_schedule(ddim_num_steps=S, ddim_eta=eta, verbose=verbose)
# sampling
C, H, W = shape
size = (batch_size, C, H, W)
print(f'Data shape for DDIM sampling is {size}, eta {eta}')

samples, intermediates = self.ddim_sampling(conditioning, size,
                                            callback=callback,
```

Fig. 3.2.8

```
                    ucg_schedule=None):
    device = self.model.betas.device
    b = shape[0]
    if x_T is None:
        img = torch.randn(shape, device=device)
    else:
        img = x_T

    if timesteps is None:
        timesteps = self.ddpm_num_timesteps if ddim_use_original_steps else self.ddim_timesteps
    elif timesteps is not None and not ddim_use_original_steps:
        subset_end = int(min(timesteps / self.ddim_timesteps.shape[0], 1) * self.ddim_timesteps.shape[0]) - 1
        timesteps = self.ddim_timesteps[:subset_end]

    intermediates = {'x_inter': [img], 'pred_x0': [img]}
    time_range = reversed(range(0,timesteps)) if ddim_use_original_steps else np.flip(timesteps)
    total_steps = timesteps if ddim_use_original_steps else timesteps.shape[0]
    print(f"Running DDIM Sampling with {total_steps} timesteps")

    iterator = tqdm(time_range, desc='DDIM Sampler', total=total_steps)

    for i, step in enumerate(iterator):
        index = total_steps - i - 1
        ts = torch.full((b,), step, device=device, dtype=torch.long)
```

Fig. 3.2.9

采样前，需要生成一张随机的噪声图，该图就作为初始图像来进行降噪生成。然后将时间步转化为向量后作为条件输入模型，接着开始循环降噪。

变量 x 表示每次处理的图像，c 表示生成的条件，t 表示迭代步数，model_t 表示根据提示词和条件生成的特征，model_uncond 表示根据反向提示词生成的特征。使用公式：降噪结果 = 反向提示词特征 + 提示词相关性 *（提示词特征 – 反向提示词特征）。通过这个公式，图像将朝着设定的条件进行降噪，同时尽可能避免反向提示词包含的内容。不断降噪指定步数后将最终得到结果返回，完成降噪过程。

```
@torch.no_grad()
def p_sample_ddim(self, x, c, t, index, repeat_noise=False, use_original_steps=False, quantize_denoised=False,
                  temperature=1., noise_dropout=0., score_corrector=None, corrector_kwargs=None,
                  unconditional_guidance_scale=1., unconditional_conditioning=None,
                  dynamic_threshold=None):
    b, *_, device = *x.shape, x.device

    if unconditional_conditioning is None or unconditional_guidance_scale == 1.:
        model_output = self.model.apply_model(x, t, c)
    else:
        model_t = self.model.apply_model(x, t, c)
        model_uncond = self.model.apply_model(x, t, unconditional_conditioning)
        model_output = model_uncond + unconditional_guidance_scale * (model_t - model_uncond)
```

Fig. 3.2.10

最终线稿上色效果：（调整参数部分前文已经展示）

输入线稿测试图：

Fig. 3.2.11

无提示词条件下：

Fig. 3.2.12

有提示词 red dress 条件下：

Fig. 3.2.13

3.3 实验测试

我们小组一共进行了 3 次训练实验测试。每次训练我们采用相同的训练参数，训练的 batch_size = 4，learning_rate = e^{-5}，Stable Diffusion 模型部分锁定，训练 ControlNet 部分。训练基本流程如下：

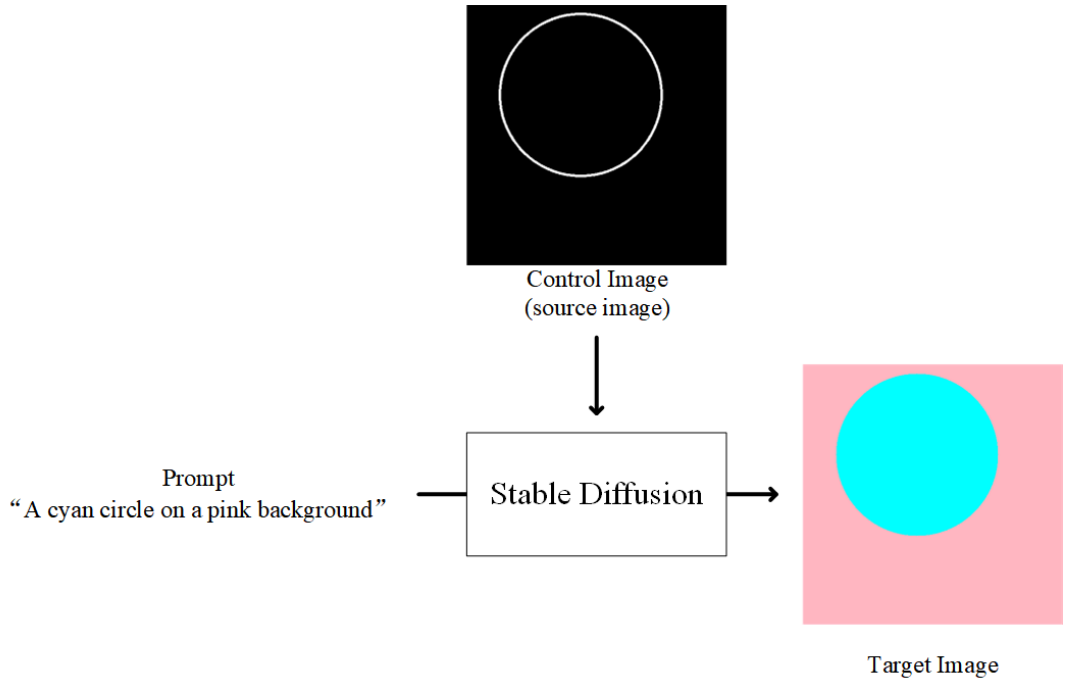

Fig. 3.3

我们想控制 Stable Diffusion 模型用颜色填充 circle，prompt 包含我们 target 图片的文字描述。Stable Diffusion 模型是已经预训练好的，它已经知道提示文字的含义，但不知道“控制图像”的含义。我们的目标就是让他知道控制条件的含义，然后生成期望的图片。

3.3.1 第一次训练实验

我们使用作者提供的数据集 fill50k 进行训练。数据集的结构包括：控制图 source/ 文件夹、原图 target/ 文件夹、提示词 prompt.json 文件。这个数据集不是专门针对线稿上色模型的，因此对线稿上色效果较差。此次实验的目的一是摸清实验整体流程，二是了解从哪些方面进行评估分析。我们训练了 3 个 epoch，根据训练日志得到的曲线走势图如下：

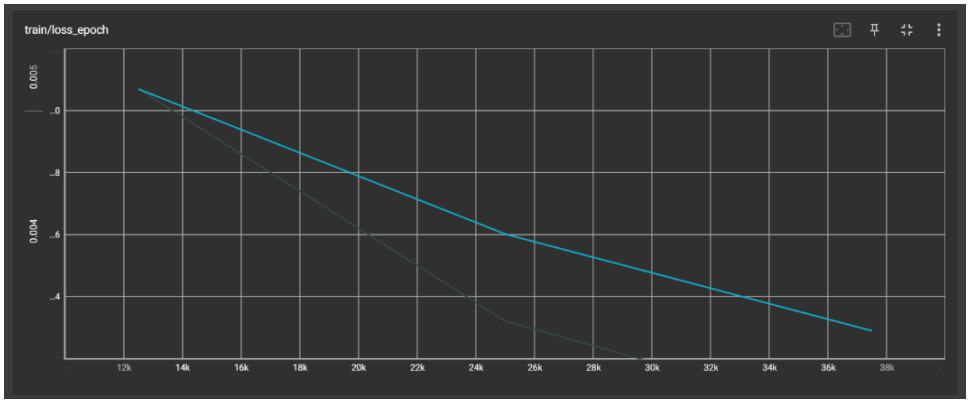

Fig. 3.3.1.1

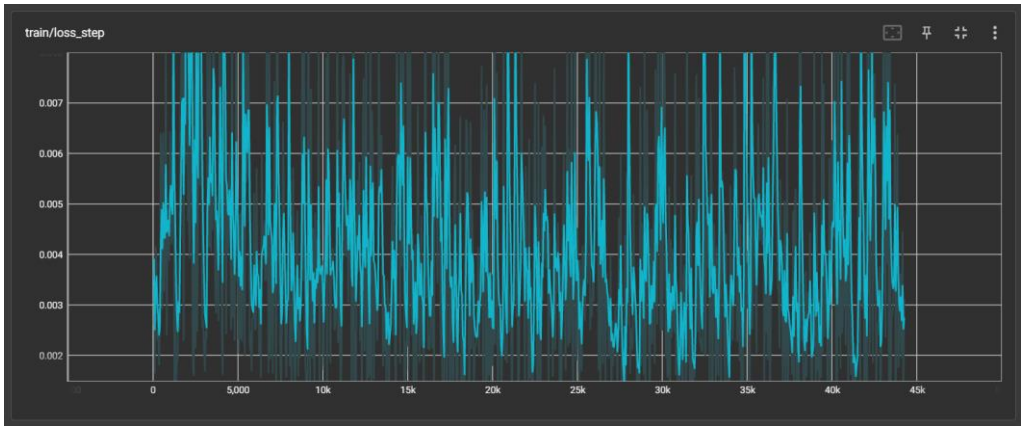

Fig. 3.3.1.2

整体上 loss 成下降趋势，说明真实图片和预测图片的误差在减小，模型的性能逐渐向好。我们随机选取了训练过程中模型对训练集图像的预测图片，结果如下：

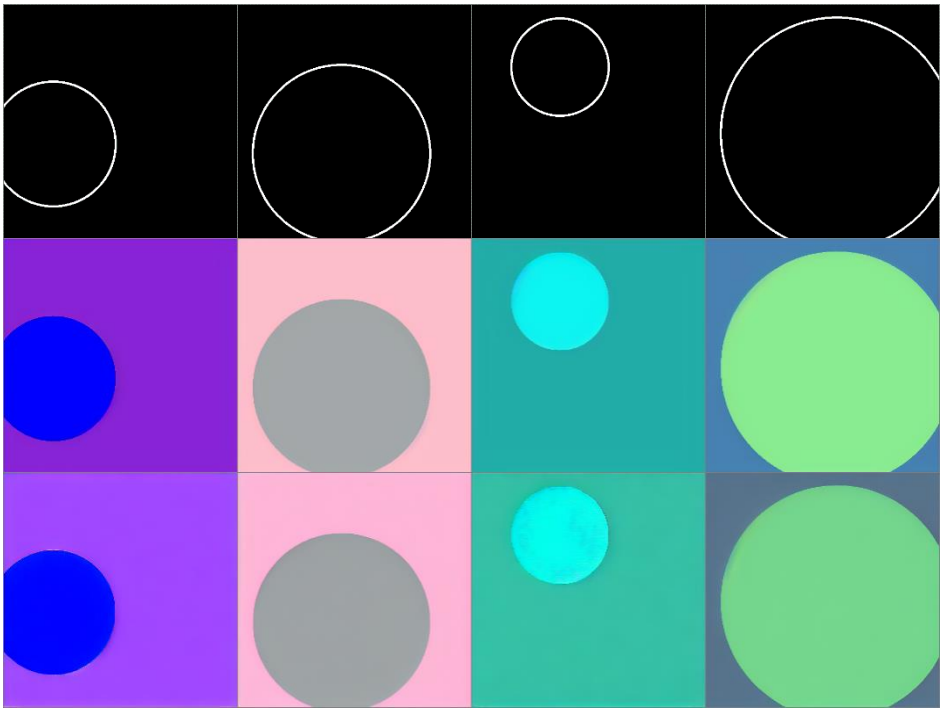

Fig. 3.3.1.3

第一排是控制图像，第二排是真实图像，第三排是训练模型的预测图像。可以发现预测图像和真实图像的误差不算大，模型对 circle 的边缘信息理解较好，对图像填充的颜色信息理解稍差。可能由于训练集都是 circle 的原因，测试时选用非 circle 效果并不理想，于是我们选择训练集中的图进行测试。测试结果如下：

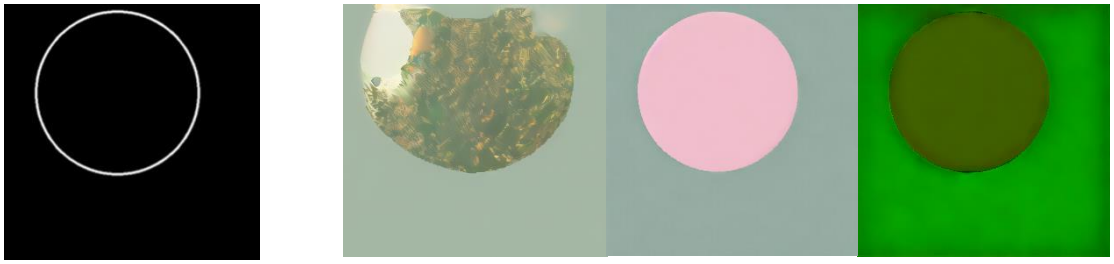

Fig. 3.3.1.4

第一张图是输入的控制图，第二张图在没有提示词条件下的生成图像，后两张图是在有提示词条件下的生成图像。可以看到模型对控制图的信息理解较好，但是在没有提示词条件和有提示词条件时相比，对图像边缘信息的理解较差，我们认为在迭代次数较少的情况下，模型还是比较依赖 prompt 描述才能有较好的生成效果。

3.3.2 第二次训练实验

我们在 danbooru 数据集基础上建立自己的数据集。我们从中随机选取 50k 对原图和线稿图片对，通过提示词反推工具对原图生成提示词，编写 python 脚本将提示词进行合成和结构化，并仿照作者给出的 fill50k 数据集的结构，建立自己的数据集。此次实验我们训练了 11 个 epoch，得到的曲线走势图如下：

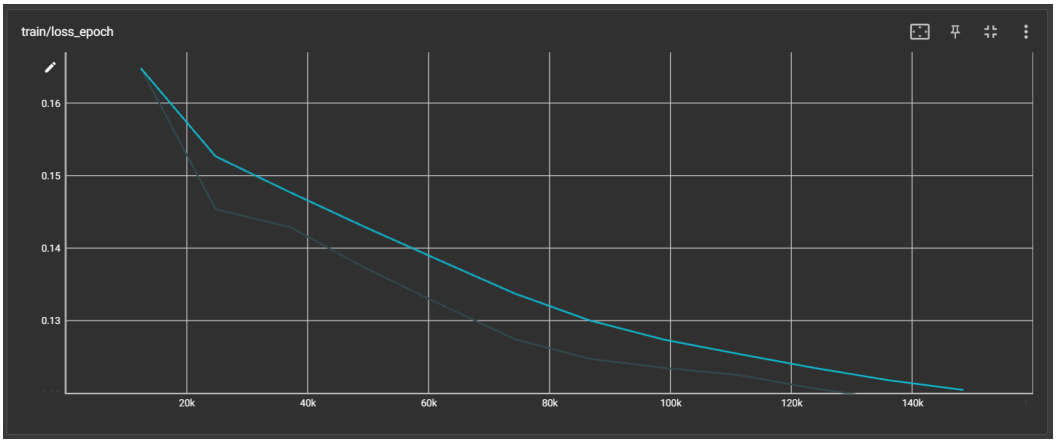

Fig. 3.3.2.1

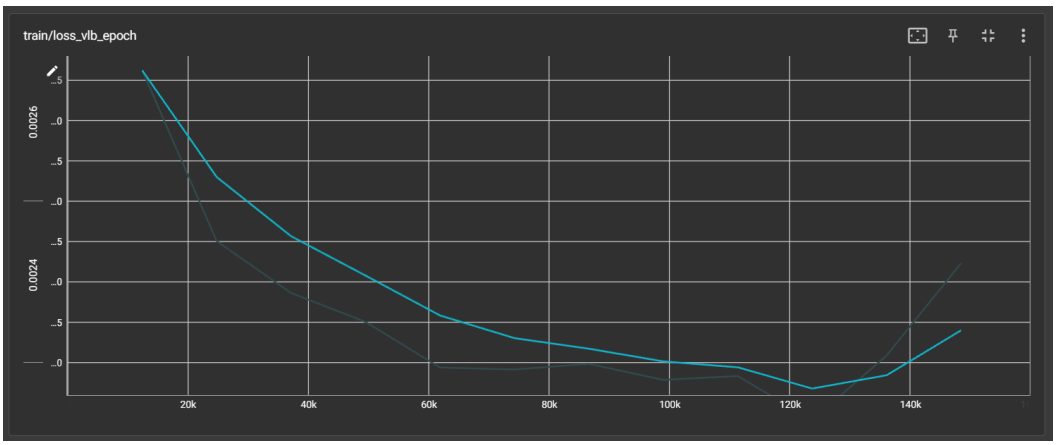

Fig. 3.3.2.2

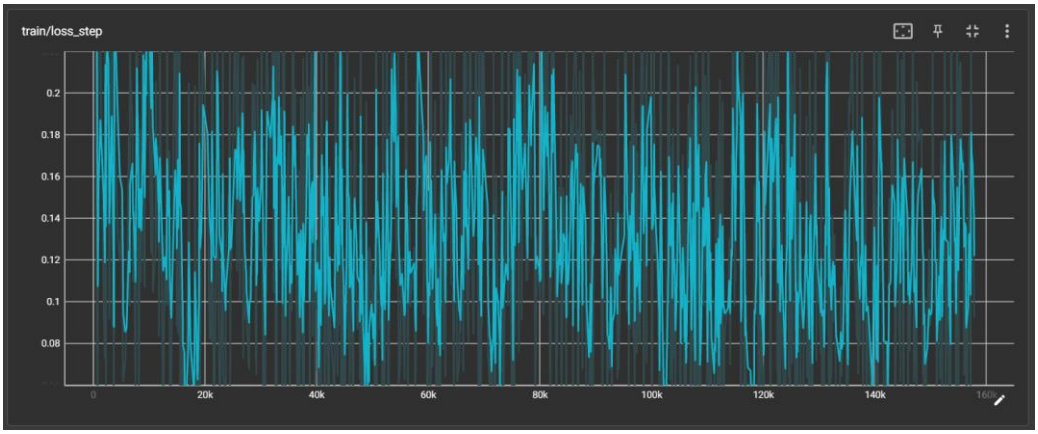

Fig. 3.3.2.3

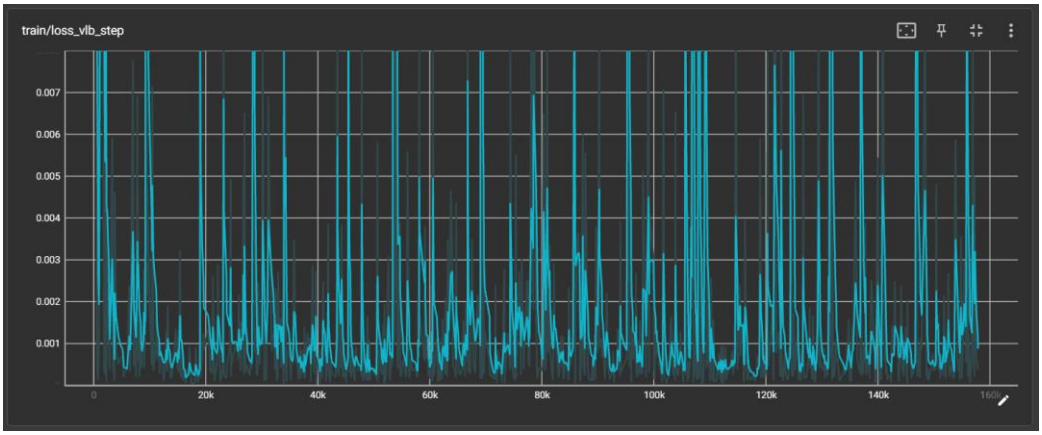

Fig. 3.3.2.4

其中一个指标是 loss_vlb_epoch，它反映的是扩散模型的变分推断中近似分布和真实分布的差异。vlb 是变分下界，用于计算近似分布。训练过程中模型对训练样本的预测如下：

Fig. 3.3.2.5

从上到下分别是线稿图、真实图和预测图。可以看到模型对人物边缘信息理解还是较好的，但是细节部分还是不容乐观，譬如人物的眼睛、脸、发型等，并且人物的颜色上色差距很大。之后测试的结果也不理想。我们将这个数据集和 fill50k 数据集进行对比后，发现我们的数据集控制图是白底黑线，而 fill50k 数据集是黑底白线，我们了解到计算机是将黑像素值为 0，白像素值为 255，把黑色当作 mask 掩膜也就是不需要处理的部分，因此我们决定将线稿图进行反色处理后，作为控制图进行训练。

3.3.3 第三次训练实验

此次训练实验中，我们将输入图像进行反色，并随机选取更少的 10k 对图片对，用更多的迭代次数进行训练。最终共迭代了 30 个 epoch，得到的曲线走势图如下：

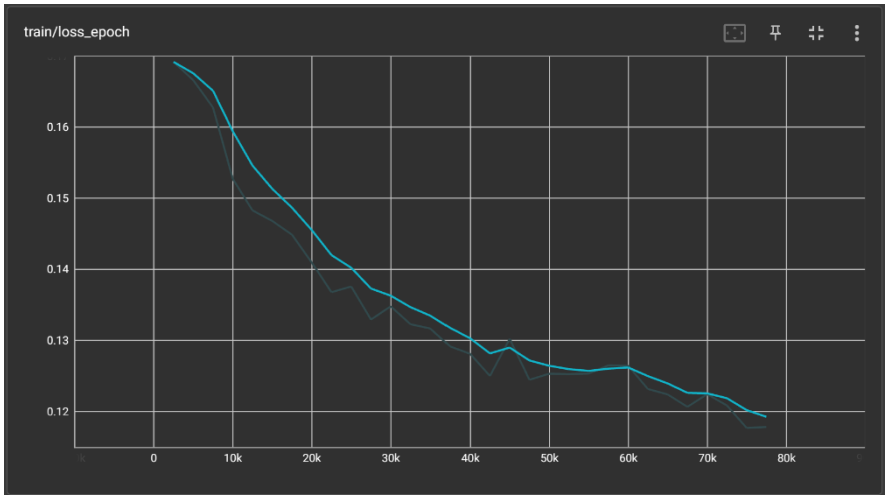

Fig. 3.3.3.1

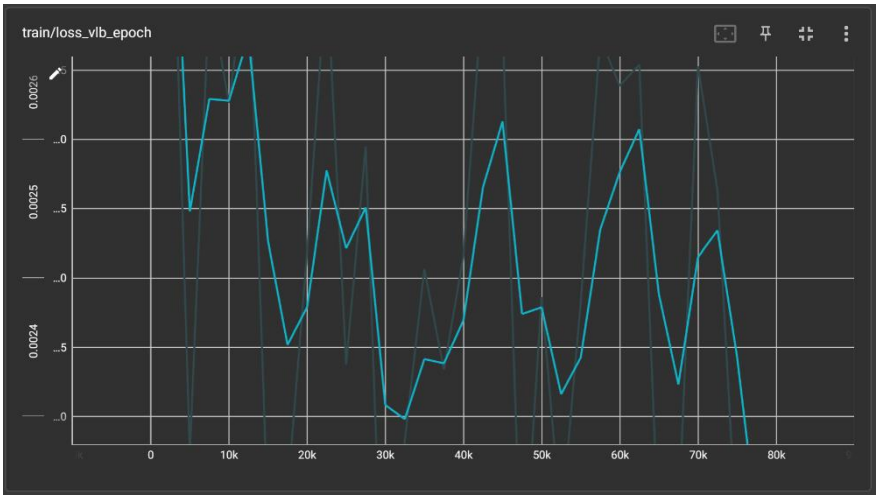

Fig. 3.3.3.2

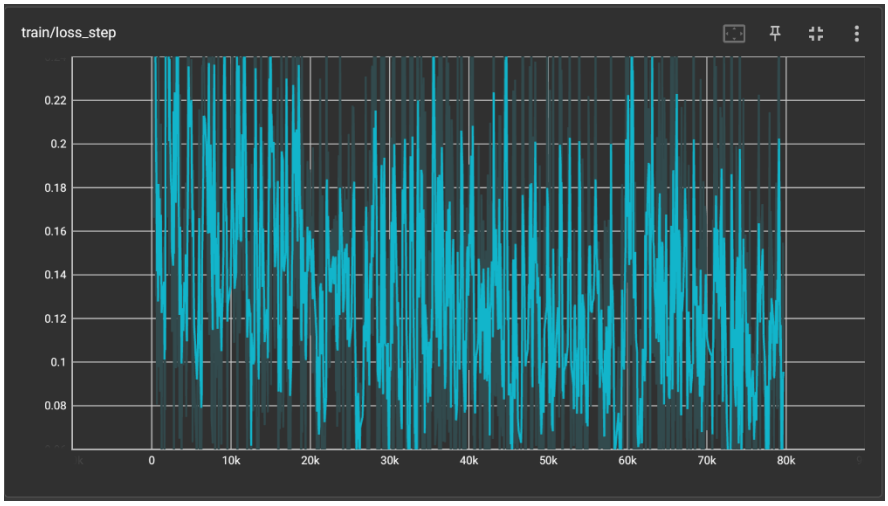

Fig. 3.3.3.3

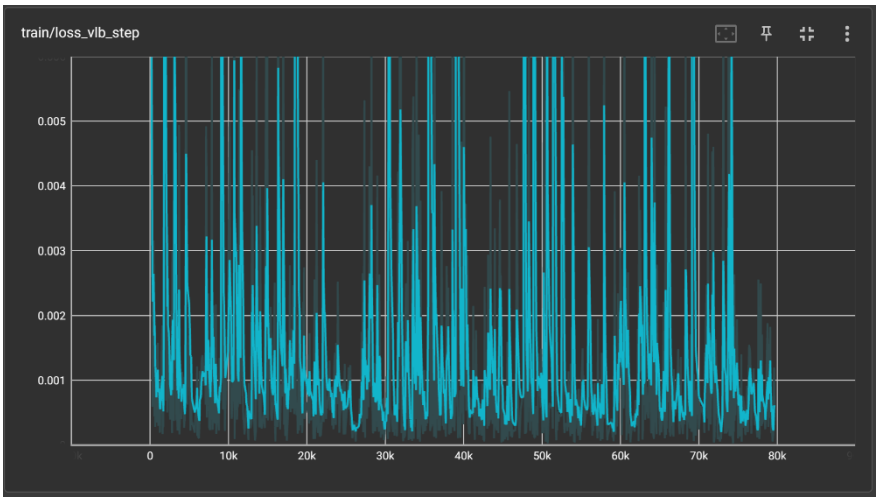

Fig. 3.3.3.4

四张曲线图的整体趋势都是下降的。而在第二张图中，可以发现 loss_vlb_epoch 的值发生了抖动现象。我们小组讨论研究后认为，出现抖动情况可能是因为每个迭代步骤只利用了一个小批量数据，在计算变分下界损失函数时，需要对整个训练集进行求和，计算出训练集的损失函数。如果样本数据较少，那么每次迭代计算得到的损失函数受到该少量数据中的噪声等因素的影响就会相对更大，从而导致损失函数值抖动，并且可能会导致过拟合现象的发生。解决抖动现象就需要更大批量的数据，或者使用更优的算法。以下是随机选取的训练过程中模型的预测图像：

Fig. 3.3.3.5

可以看到，此次训练实验与第二次训练实验相比，对图像的边缘信息理解更好，误差更小。但仍然存在一个问题，也就是预测图像的颜色问题。我们发现在自己建立的数据集中，prompt 对图像的内容信息有较好的描述，但是颜色信息却存在较大误差甚至是缺失。我们尝试解决该问题，但解决方案主要还是人工筛选优化，对于样本较多的数据集来说，这项工作非常繁杂。该模型的测试结果如下：

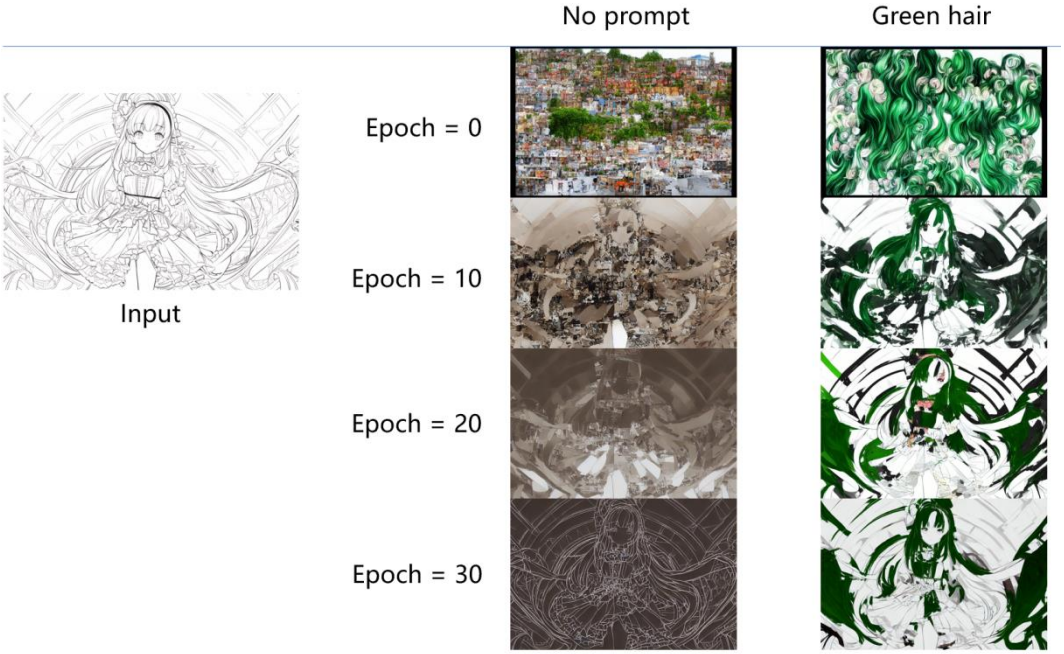

Fig. 3.3.3.6

我们用非训练集中的图像，在 epoch = 0，10，20，30 及分别有提示词和无提示词条件下进行了测试。可以看到模型生成结果整体趋势是向好的，ControlNet 的作用部分相对来说性能在逐渐提升。无提示词条件下，从最初的毫无关系的图像，逐渐生成边缘较清晰的图片，但是在对图像上色的颜色单一，这可能是由于

模型训练不足，导致对 prompt 的依赖，如果没有 prompt 提示，就只会对整张图片填充一种颜色；有提示词条件下，ControlNet 的作用部分对人物头发的理解逐渐提升，从最初的大范围上色，到最后的上色范围缩小，但缩小后，部分本应该上色的地方没有进行上色。同时，由于对 prompt 的依赖，当只输入一个 prompt 时，其他部位几乎没有进行颜色的填充。

3.3.4 第四次训练实验

本次实验中，我们使用第二次实验中的 50k 对图片对的数据集，将输入的线稿图像进行了反色后进行训练，最终一共训练了 32 个 epoch，400k 个 steps，loss_epoch 最终截止于 0.099。由于最终得到的各个指标的曲线图与第二、三次训练中的曲线走势大致相同，且训练过程中的预测图像较好于第三次训练中的预测图像，因此本节中不再赘述，仅展示纵向对比结果图。

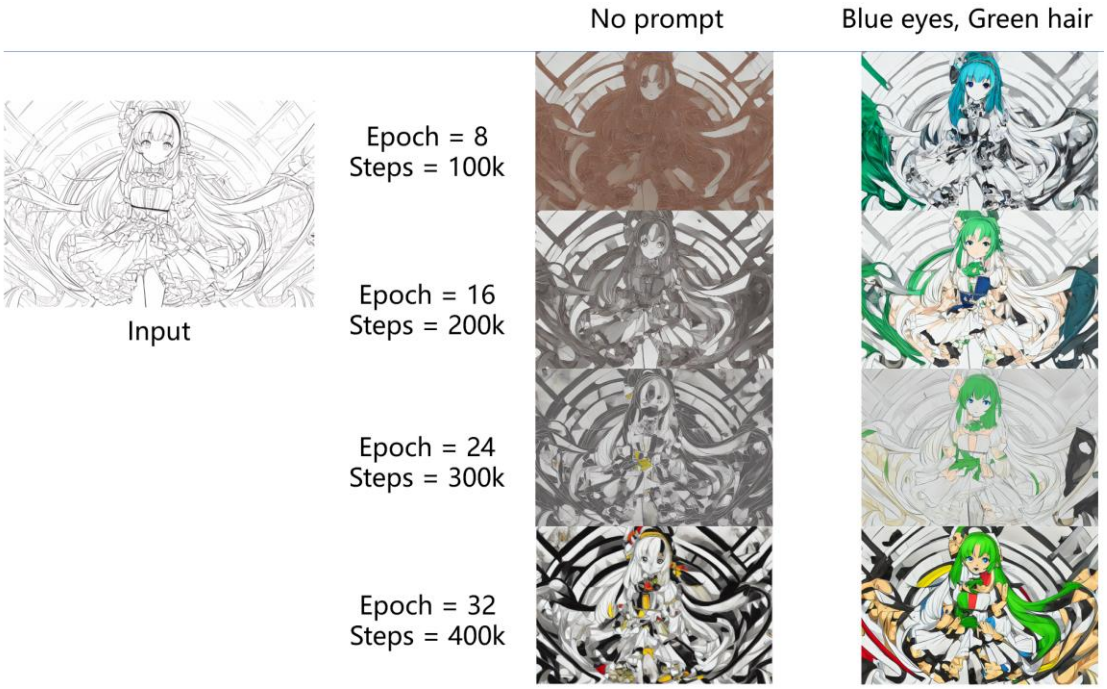

Fig. 3.3.4.1

可以看到模型在无提示词和有提示词的条件下都呈现逐渐向好的趋势，与第二次、第三次实验对比可以发现，模型对图像的边缘理解更加准确，在无提示词条件下，从最初只有单个颜色填充且边缘不明显的情况下，随着 Steps 的增加，边缘逐渐清晰，填充颜色数量增加，虽然仍存在很大误差；在有提示词条件下，对图像信息的理解也更加准确，头发部分的上色从最初的很少部分甚至上错位置，到最后头发基本上色准确。综上可以说明训练结果初见成效，相信在优化了 Prompt 提示词和训练更多 Steps 之后，能取得更满意的结果。

4 问题分析

在本文中的几个训练实验测试中，我们发现几个模型的训练与测试都有一定的效果，但是效果并不是很好，在对图像的特征信息的理解上还存在着明显的偏差，譬如将人物背景识别成人物头发、将人物颈部的装饰物识别为眼睛等，并且在对线稿进行上色时可能出现颜色单调或者没有成功上色的情况。我们小组对几个训练实验测试进行了思考总结，认为在以下方面存在不足和可以改进：

训练过程中迭代的次数以及训练的步数。当训练的迭代次数越多，模型越能够充分学习训练集中的模式和特征，提高模型的准确性。但是当迭代次数过多时，会出现过拟合的现象，导致在训练集上的效果好，在测试集上效果变差。因此我们尝试根据训练过程中 loss_epoch 的变化，以及在每进行 n 个左右的 epoch 后，保存 checkpoint 并检查当前的模型性能是否有所提升即上色效果变好，如果没有提升，则终止训练。

训练集中的图像质量。图像的分辨率低会导致模型无法正确识别图像特征，图像的噪声过多会导致模型的准确性受到影响，而样本图像之间的差异过大，也可能导致模型过度学习这些样本图像，从而导致在新的测试集的表现不佳的过拟合现象。fill50k 数据集图像与我们的数据集图像相比，图像分辨率更高，并且样本图像之间的差异较小，因此在相同的训练参数下，理论上前者的训练效果较好。较好的解决方案就是搜集分辨率更好样本差异较小的数据集进行训练。

训练集中的提示词质量。与 fill50k 数据集中的提示词相比，我们自己建立的数据集的提示词存在问题主要是在对图像的描述的准确度方面。我们的提示词是通过当前已存在的提示词反推模型推导而来，因此在提示词的准确度方面欠佳，譬如没有对颜色进行正确识别、对图像一些细节存在识别错误等。解决该问题的方法主要还是需要通过人工筛选优化。之后我们可以尝试使用更小的训练集，人工对提示词进行筛选优化后进行更多的 epoch 训练，对训练后的模型进行对比和分析，判断提示词的优劣对模型的影响的强弱。

收获与体会

在过去的两个多月中，我们小组有幸参与到一项论文复现课题《基于 ControlNet 引导的线稿上色》的研究中。我们小组的成员都对动漫有一定了解和兴趣，便希望借此机会学习当前领域的前沿知识和技术，积累相关学习经验。

我们小组成员都是初次接触与科研有关的论文复现课题，因此在进行课题研究的过程中，我们一直学习如何进行科研相关工作，如查找相关文献等；同时组

内经常进行探讨和交流，分享自己的学习成果，最终完成了论文复现课题。这为我们今后的研究生阶段的学习奠定了经验基础。

我们从兴趣出发，探索目前 AI 绘画领域的前沿知识和技术，比如 Attention 机制、Text-to-Image、扩散模型等。我们从 ControlNet 论文出发，了解了构建 ControlNet 的动机：为了在有限资源下也能产生较好的生成效果，对预训练好的大模型进行微调；理解了 ControlNet 的实现原理，通过复制 Stable Diffusion 的网络结构，锁定原来的网络以保留原网络能力，训练复制网络来进行性能优化等。在此基础上，我们拓展学习了相关工作的论文，如 Attention 机制，CLIP 文图匹配，Fine-Tuning 的其他模型等，并与 ControlNet 进行联系或对比，最后总结出小组关于上色任务的见解。

在进行论文复现课题的过程中，我们遇到了一些困难和挑战，比如本地设备性能不足、运行环境的配置和搭建、python 包的版本的不一致导致各种报错、模型的训练效果不理想、论文中相关概念不理解或缺少相关资料等。我们对出现的问题进行网络检索或是请教学姐，不断探索和尝试，最终一一解决了这些问题。从中我们获得了一些与工程相关的技术能力以及对论文的阅读理解能力，希望在将来我们能够将这些能力运用到实践中去。在解决这些困难和挑战的过程中，我们还通过定期汇报和组内交流探讨等形式，提高了团队合作和沟通交流的能力，明白了科研工作不仅仅是依靠个人的努力，它更需要团队的协作、集多人的智慧，每个人的专业知识和独特视角都是团队研究成功的关键。

最后我们感谢老师和学姐为我们提供指引和帮助，使我们在这次课题中收获良多。

参考文献

[1] L. Zhang, M. Agrawala. Adding Conditional Control to Text-to-Image Diffusion Models. *arXiv preprint arXiv: 2302.05543*, 2023.

[2] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. High-resolution image synthesis with latent diffusion models, 2021.

[3] A. Hertz, R. Mokady, J. Tenenbaum, K. Aberman, Y. Pritch, and D. Cohen-Or. Prompt-to-prompt image editing with cross attention control. *arXiv preprint arXiv:2208.01626*, 2022.

[4] V. Mnih, N. Heess, A. Graves, K. Kavukcuoglu. Recurrent Models of Visual Attention. *arXiv preprint arXiv:1406.6247*, 2014.

[5] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.

[6] R. Gal, Y. Alaluf, Y. Atzmon, O. Patashnik, A. H. Bermano, G. Chechik, D. Cohen-Or. An Image is Worth One Word: Personalizing Text-to-Image Generation using Textual Inversion. *arXiv preprint arXiv:* 2*208.01618*, 2022.

[7] N. Ruiz, Y. Li, V, Jampani, Y. Pritch, M. Rubinstein, K. Aberman. DreamBooth: Fine Tuning Text-to-Image Diffusion Models for Subject-Driven Generation. *arXiv preprint arXiv: 2208.12242*, 2023.

[8] E. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen. LORA: LOW-RANK ADAPTATION OF LARGE LANGUAGE MODEL. *arXiv preprint arXiv: 2106.09685*, 2021.

[9] X. Pan, A. Tewari, T. Leimkuhler, L. Liu, A. Meka, C. Theobalt. Drag Your GAN: Interactive Point-based Manipulation on the Generative Image Manifold.

附录

该附录章节包含在复现课题中自己编写的 python 脚本以及对源代码的部分修改。

get_prompt.py

```
import os

# {"source": "source/0.png", "target": "target/0.png", "prompt": "pale golden rod circle with old lace background"}
# source: 线稿图（sketch）
# target: 原图（src）

# 需要拼接的语句
concat1 = "{\"source\": \"source/"
concat2 = ".png\", \"target\": \"target/"
concat3 = ".png\", \"prompt\": \""
concat4 = "\"}"

# output file
output_file_path = "./training/danbooruSmall/prompt.json"
# open file path
open_file_path = "./training/danbooruSmall/prompts/"
# pictures file path
pic_file_path = "./training/danbooruSmall/source/"

"""
遍历 prompts 文件夹内文件
因为文件名相同，所以直接获取文件名（除去后缀）即可
然后写入 output_file 文件中
"""
def get_prompt():
    # open the output temp file
    output_file = open(output_file_path, 'w', encoding = 'utf-8')

    # 获取文件夹中存在的 pic 的文件名（共 10000 个）
    pic_filenames = []
```

```
    for pic_file in os.listdir(pic_file_path):
        (pic_filename, file_extension_name) = os.path.splitext(pic_file)
        pic_filenames.append(pic_filename)
    # print(pic_filenames)
    # print(len(pic_filenames))

    for pic_filename in pic_filenames:
        # open prompt file
        prompt_file = open(open_file_path + pic_filename + ".txt")
        # get the one line of the file
        line = prompt_file.readline()
        #  去除逗号，去除反斜杠，去除  ""
        # line = line.replace(",", "")
        line = line.replace("\\", "")
        line = line.replace("\"", "\'")
        # print(open_file_path + file + line)
        # write into the output file in format
        output_file.write(concat1 + pic_filename + concat2 + pic_filename + concat3
+ line + concat4 + "\n")
        # close
        prompt_file.close()
    output_file.close()

get_prompt()
```

gradio_canny2image.py

```
from share import *
import config

import cv2
import einops
import gradio as gr
import numpy as np
import torch
import random

from pytorch_lightning import seed_everything
from annotator.util import resize_image, HWC3
from annotator.canny import CannyDetector
```

```
from cldm.model import create_model, load_state_dict
from cldm.ddim_hacked import DDIMSampler

apply_canny = CannyDetector()

model = create_model('./models/cldm_v15.yaml').cpu()
# 加载预训练模型
model.load_state_dict(load_state_dict('./models/control_sd15_canny.pth',
location='cuda'))
# 加载自训练模型
#    model.load_state_dict(load_state_dict('./models/control_sd15_fill50k.ckpt',
location='cuda'))
#
model.load_state_dict(load_state_dict('./models/control_sd15_danbooru_epoch9.ckpt'
, location='cuda'))
model = model.cuda()
ddim_sampler = DDIMSampler(model)


def process(input_image, prompt, a_prompt, n_prompt, num_samples,
image_resolution, ddim_steps, guess_mode, strength, scale, seed, eta, low_threshold,
high_threshold):
with torch.no_grad():
img = resize_image(HWC3(input_image), image_resolution)
H, W, C = img.shape

# TODO 无需进行预处理
# detected_map = apply_canny(img, low_threshold, high_threshold)
# detected_map = HWC3(detected_map)

# TODO 传进来的线稿是黑线，需要进行反色为白线
input_image = HWC3(input_image)
detected_map = input_image.copy()
detected_map = 255 - detected_map

# TODO 这里与 lineart_anime 不一致
control = torch.from_numpy(detected_map.copy()).float().cuda() / 255.0
control = torch.stack([control for _ in range(num_samples)], dim=0)
control = einops.rearrange(control, 'b h w c -> b c h w').clone()
```

```
if seed == -1:
seed = random.randint(0, 65535)
seed_everything(seed)

if config.save_memory:
model.low_vram_shift(is_diffusing=False)

cond = {"c_concat": [control], "c_crossattn":
[model.get_learned_conditioning([prompt + ', ' + a_prompt] * num_samples)]}
un_cond = {"c_concat": None if guess_mode else [control], "c_crossattn":
[model.get_learned_conditioning([n_prompt] * num_samples)]}
shape = (4, H // 8, W // 8)

if config.save_memory:
model.low_vram_shift(is_diffusing=True)

# TODO 这里与 lineart_anime 不一致
model.control_scales = [strength * (0.825 ** float(12 - i)) for i in range(13)] if
guess_mode else ([strength] * 13)  # Magic number. IDK why. Perhaps because
0.825**12<0.01 but 0.826**12>0.01
samples, intermediates = ddim_sampler.sample(ddim_steps, num_samples,
shape, cond, verbose=False, eta=eta,
unconditional_guidance_scale=scale,
unconditional_conditioning=un_cond)

if config.save_memory:
model.low_vram_shift(is_diffusing=False)

x_samples = model.decode_first_stage(samples)
x_samples = (einops.rearrange(x_samples, 'b c h w -> b h w c') * 127.5 +
127.5).cpu().numpy().clip(0, 255).astype(np.uint8)

results = [x_samples[i] for i in range(num_samples)]
# return [255 - detected_map] + results
return [255 - detected_map] + results


block = gr.Blocks().queue()
with block:
```

```
with gr.Row():
gr.Markdown("## Control Stable Diffusion with Canny Edge Maps")
with gr.Row():
with gr.Column():
input_image = gr.Image(source='upload', type="numpy")
prompt = gr.Textbox(label="Prompt")
run_button = gr.Button(label="Run")
with gr.Accordion("Advanced options", open=False):
num_samples = gr.Slider(label="Images", minimum=1, maximum=12, value=1,
step=1)
image_resolution = gr.Slider(label="Image Resolution", minimum=256,
maximum=768, value=512, step=64)
strength = gr.Slider(label="Control Strength", minimum=0.0, maximum=2.0,
value=1.0, step=0.01)
guess_mode = gr.Checkbox(label='Guess Mode', value=False)
low_threshold = gr.Slider(label="Canny low threshold", minimum=1, maximum=255,
value=100, step=1)
high_threshold = gr.Slider(label="Canny high threshold", minimum=1,
maximum=255, value=200, step=1)
ddim_steps = gr.Slider(label="Steps", minimum=1, maximum=100, value=20,
step=1)
scale = gr.Slider(label="Guidance Scale", minimum=0.1, maximum=30.0, value=9.0,
step=0.1)
seed = gr.Slider(label="Seed", minimum=-1, maximum=2147483647, step=1,
randomize=True)
eta = gr.Number(label="eta (DDIM)", value=0.0)
a_prompt = gr.Textbox(label="Added Prompt", value='best quality, extremely
detailed')
n_prompt = gr.Textbox(label="Negative Prompt",
value='longbody, lowres, bad anatomy, bad hands, missing fingers, extra digit, fewer
digits, cropped, worst quality, low quality')
with gr.Column():
result_gallery = gr.Gallery(label='Output', show_label=False,
elem_id="gallery").style(grid=2, height='auto')
ips = [input_image, prompt, a_prompt, n_prompt, num_samples, image_resolution,
ddim_steps, guess_mode, strength, scale, seed, eta, low_threshold, high_threshold]
run_button.click(fn=process, inputs=ips, outputs=[result_gallery])


block.launch(server_name='0.0.0.0')
```

tutorial_dataset.py

```
import json
import cv2
import numpy as np

print("starting...")
from torch.utils.data import Dataset
print("ending...")

dataset_name = "danbooru/"

class MyDataset(Dataset):
    def __init__(self):
        self.data = []
        with open('./training/' + dataset_name + 'prompt.json', 'rt') as f:
            # for line in f:
            #     self.data.append(json.loads(line))
            lines = f.readlines()
            for line in lines:
                self.data.append(json.loads(line))

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        item = self.data[idx]

        source_filename = item['source']
        target_filename = item['target']
        prompt = item['prompt']

        source = cv2.imread('./training/' + dataset_name + source_filename)
        target = cv2.imread('./training/' + dataset_name + target_filename)

        # Do not forget that OpenCV read images in BGR order.
        # TODO 如果是 danbooru 数据集，则将控制图（线稿）进行反色处
理
        if ("danbooru" in dataset_name):
            source = 255 - cv2.cvtColor(source, cv2.COLOR_BGR2RGB)
```

```
        else:
            source = cv2.cvtColor(source, cv2.COLOR_BGR2RGB)

        target = cv2.cvtColor(target, cv2.COLOR_BGR2RGB)

        # Normalize source images to [0, 1].
        source = source.astype(np.float32) / 255.0

        # Normalize target images to [-1, 1].
        target = (target.astype(np.float32) / 127.5) - 1.0

        return dict(jpg=target, txt=prompt, hint=source)
```

tutorial_train.py

```
from share import *

import pytorch_lightning as pl
from pytorch_lightning.callbacks import ModelCheckpoint
from torch.utils.data import DataLoader
print("before import dataset")
from tutorial_dataset import MyDataset
print("after import dataset")
from cldm.logger import ImageLogger
from cldm.model import create_model, load_state_dict


# Configs
resume_path = './models/control_sd15_danbooru.ckpt'
batch_size = 4
logger_freq = 300
learning_rate = 1e-5
sd_locked = True
only_mid_control = False


# First use cpu to load models. Pytorch Lightning will automatically move it to GPUs.
model = create_model('./models/cldm_v15.yaml').cpu()
model.load_state_dict(load_state_dict(resume_path, location='cpu'))
```

```
model.learning_rate = learning_rate
model.sd_locked = sd_locked
model.only_mid_control = only_mid_control


# Misc
print("before dataset instance")
dataset = MyDataset()
print("dataset length: ")
print(len(dataset))
print("after dataset instance")
dataloader = DataLoader(dataset, num_workers=0, batch_size=batch_size,
shuffle=True)
logger = ImageLogger(batch_frequency=logger_freq)

# TODO add checkpoint_callback
checkpoint_callback = ModelCheckpoint(
                        dirpath="models_checkpoints/danbooruSmall/",
                        every_n_train_steps=2500,
                        save_weights_only=False,
                        save_top_k = -1
                    )

trainer = pl.Trainer(gpus=1, precision=32, callbacks=[logger,checkpoint_callback])


# Train!
trainer.fit(model, dataloader)
```