**《图像分析基础》**

课程报告

**基于ControlNet与Stable Diffusion的**

**参考上色模型**

**姓 名 张一鸣**

**学 号 123103010658**

**授课教师 赵林 教授**

**日 期 2023.12.15**

目 录

[一、摘要 1](#_Toc21882)

[二、简介 1](#_Toc21626)

[三、相关工作 1](#_Toc962)

[3.1 扩散模型 1](#_Toc22462)

[3.2 隐含扩散模型 2](#_Toc16964)

[3.3 控制条件 3](#_Toc27510)

[四、改进方法 4](#_Toc8450)

[4.1 改进方法 4](#_Toc27300)

[4.2 问题以及解决方法 4](#_Toc5598)

[五、相关实验 7](#_Toc26367)

[5.1 数据集以及实验配置 7](#_Toc31141)

[5.2 版本测试 8](#_Toc29767)

[5.3 效果 9](#_Toc24212)

[5.4 递归生成和隔帧生成 10](#_Toc31550)

[六、实验结果分析 10](#_Toc5115)

**一、摘要**

本篇实验报告基于Adding conditional control to text-to-image diffusion models这篇工作的代码，将条件控制Stable Diffusion Model的思想运用到参考上色任务当中，即根据彩色的参考帧给与之结构相似的黑白图像进行上色。

Project webpage: https://github.com/unexpected0/ControlNet/tree/main

**二、简介**

在许多计算机视觉任务中，将黑白图像上色是一个具有挑战性但又非常有趣的问题。为了解决这个问题，我们借鉴了《Adding conditional control to text-to-image diffusion models》的思想，该工作通过引入条件控制来提高图像生成模型的灵活性和性能。我们希望能够利用参考彩色图像的信息，通过扩散模型为黑白图像添加真实感的颜色。

在第三节中，我们主要介绍了与本次课程设计相关的重要工作，包括了DDPM，LDM和Control Net的基本原理。第四节中介绍了本文针对基于ControlNet与Stable Diffusion的参考上色模型的改进思路。第五、第六节介绍了相关的图像上色的实验以及结果分析。

**三、相关工作**

在介绍我们的改进方案之前，我们简要介绍一下两项重要相关工作（DDPM和LDM）和Control Net。

**3.1 扩散模型**

在这里简要介绍一下扩散模型Denoising Diffusion Probabilistic Models，DDPM的训练过程可以分为两个过程，前向过程和后向过程。

前向过程（Forward Process）又称为扩散过程，对于数据集内部的每一张图片不断地加入相同均值和方差的高斯噪声，直到完全看不出来原来照片，我们认为这时的图像是纯净的高斯噪声，加入噪声的次数为T，往往选取1000或更大，前向过程的数学表达式和示意图如下所示：

其中，和代表0 1之间的系数，代表均值为0方差为I的高斯噪声，代表加入t次噪声之后的图像。

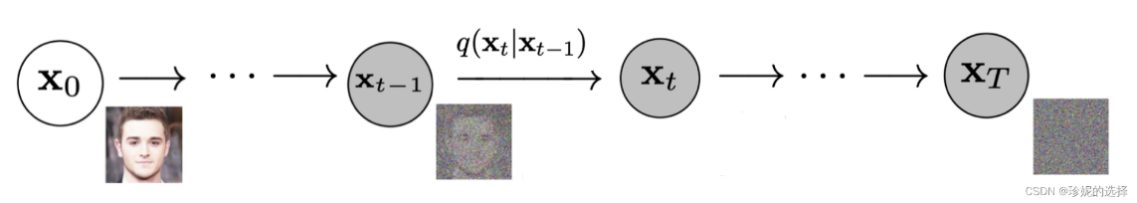


图1 扩散模型前向过程示意图

后向过程（Reverse Process）又称为去噪过程，主要是将刚刚得到的噪声慢慢去噪，数学表达式和示意图如下所示：

其中，表示去噪器的输出，采用U-Net作为模型的去噪器，输入上一次去噪的结果和当前步数的t，U-Net输入和输出尺寸完全相同，并且具有非常好的表征和学习能力。

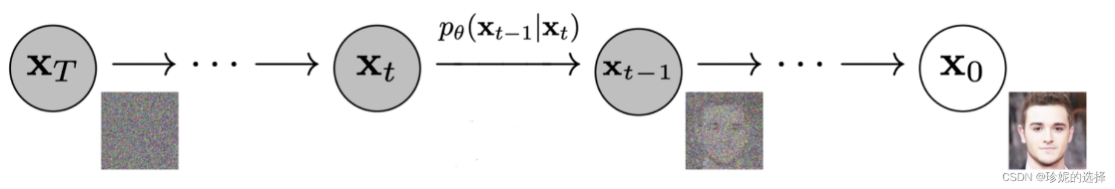


图2 扩散模型后向过程示意图

了解以上两个过程之后，DDPM将损失定义为：

由上式可知，去噪器模型的监督信号为，由上图可知，如果去噪器每一步都能够准确的预测出当前步加入的噪声的话，模型的还原结果应该和输入的图片完全一致。

**3.2 隐含扩散模型**

隐含扩散模型（Latent Diffusion Model）是扩散模型的升级版，也是著名的Stable Diffusion Model的原型，隐含扩散模型的结构示意图如下图3所示，它将DDPM扩展成两阶段模式，第一阶段单独训练自编码器，自编码器采用大名鼎鼎的VQGAN[19]，通过编码器将图片转换成较小的张量，再将小张量经过解码器还原成原来的图像，这样可大大降低扩散模型的训练负担；第二阶段基于预训练自编码器基础，在隐空间上训练扩散模型。另外一个亮点在于条件机制以指导图像去噪过程，最基础的条件是文本条件，利用CLIP模型[20]将文本信息编码成相同大小的张量，这是一个能实现文本信息和图片内容相对应的模型，再通过U-Net中每个残差模块之后加入的交叉注意力模块（Cross Attention Blocks），将文本张量加入去噪器，指导生成过程。

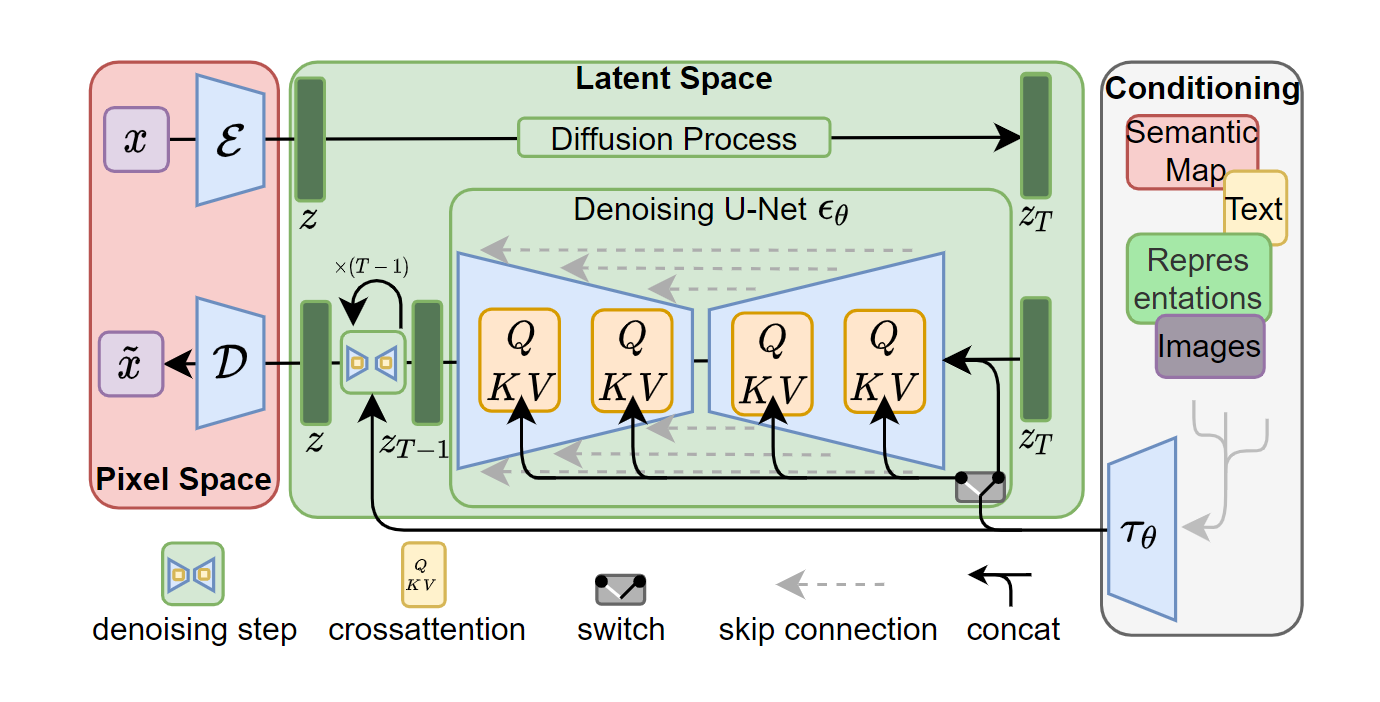


图3 隐含扩散模型结构示意图

**3.3 控制条件**

Stable Diffusion的效果当然很好，但是其训练成本是异常庞大的，目前公布出来的checkpoint经过1000张A100训练1个月得到的结果，因此控制网络Control Net的想法应运而生，其结构示意图如下图4所示。

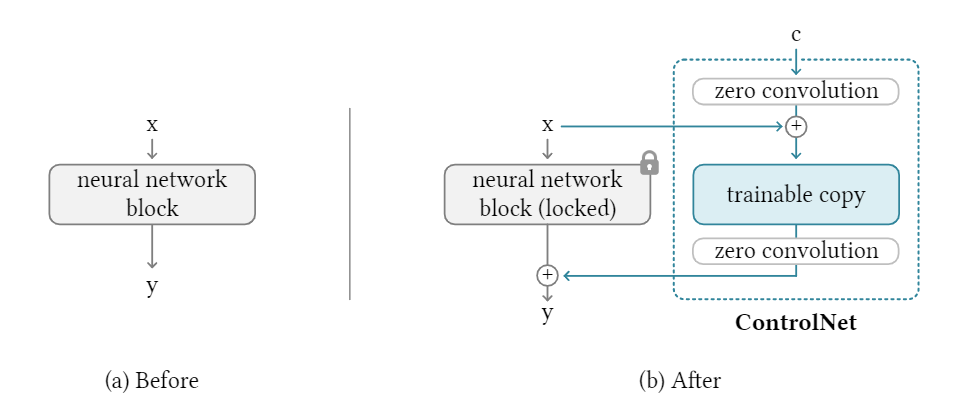


图4 控制网络示意图

Control Net旨在设计神经网络以微调已训练好的大模型，左侧(a)图表示Stable Diffusion模型的模块，我们希望可以在原Stable Diffusion架构中加入新的条件信息控制其生成图像；右侧(b)图表示控制网络改进后的示意图，控制网络先复制原来的模块架构和参数到可训练模块（trainable copy），对于加入的控制条件c以及复制模块，在他们之后添加零卷积（Zero Convolution），权重和偏置都被初始化为零的卷积，c的特征图经过对位加法加入噪声图x，同样控制网络输出通过对位加法加入原模块输出的噪声图。

训练过程中，Stable Diffusion部分的参数不改变，由于零卷积的存在，第一轮训练中不会将控制条件泄漏到可训练模块中，也不会将控制网络输出泄漏到Stable Diffusion中，但随着训练的深入，越来越多有用的信息会逐步传递到Stable diffusion当中。由于在Stable Diffusion的checkpoint上继续训练，大大减少了训练时间，官方给出的几个实例平均大约需要400个GPU小时（A100）。

**四、改进方法**

**4.1 改进方法**

参考了Stable Diffusion的技巧，在进行inpaint的过程中，给定原图片和掩码（mask），将原图片中的mask标注部分掩盖掉，然后让模型预测出被掩盖部分，预测过程中，会将待修复的原图片经过编码和掩码合并（concat）到噪声图当中。我们认为这项技巧对于本文的上色工作的启发性是很大的，这项工作也需要生成图像生成图片的整体色调靠近参考图片的。因此我们设计了三个版本，扩散模块的输入示意图如下图5所示。

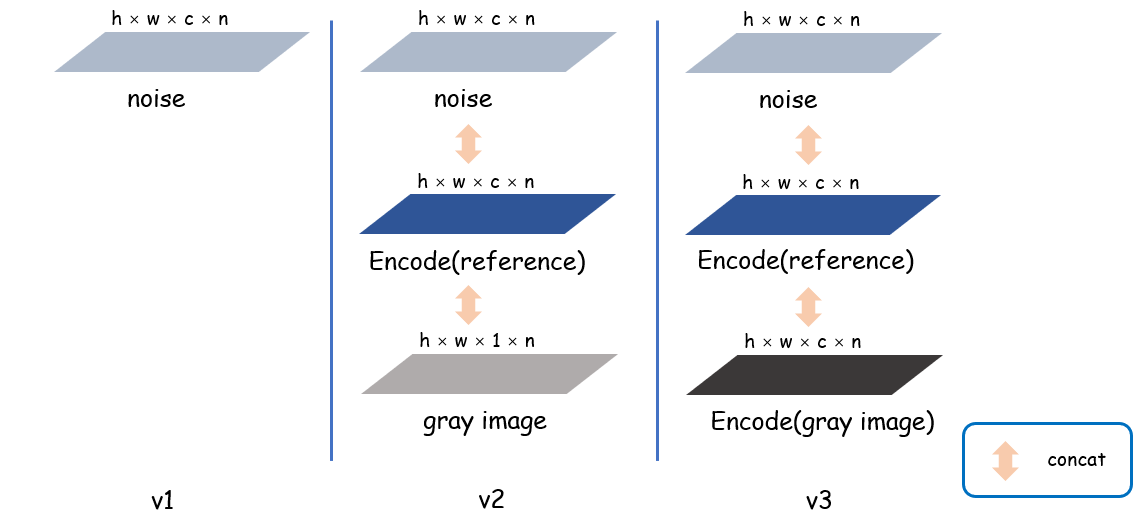


图5 改进版本示意图

其中，h、w和c分别表示图片的经过编码器之后的特征的长、宽和通道数（选取4），v1就是原版，扩散模型中去噪器以及控制网络中的可训练版去噪器的输入只有噪声图；v2在v1的基础上合并了参考彩色图的编码结果以及灰色图像的调整尺寸（resize）后的结果，因此v2的去噪器输入的通道数为9（4 + 4 + 1）；v3的思路和v2类似，区别在于v3我们将黑白图像扩展到3通道看成彩色图像，再经过VQGAN编码，因此v3的去噪器输入的通道数为12（4 + 4 + 4），关于以上三个版本相对比的具体实验和分析详见下文。

**4.2 问题以及解决方法**

首先，加载模型的时候，由于Control Net原来的噪声输入通道数是4，而我们改进的版本中v2（v3）对应的通道数为9（12），因此：

（1）将config文件中model-control\_stage\_config-params-in\_channels以及model-unet\_config-params-in\_channels的数值改为9（12）；

（2）加载模型时，原版本的输入层的卷积核尺寸为（320,4,3,3）与现在版本的输入尺寸不符合导致报错，我们需要将尺寸改为和输入通道一致。我们找到torch.nn.modules 文件line 1645，加入如下的代码：

# -------------------------------------------------------------------------------------

if hasattr(child,'in\_channels') and hasattr(child,'out\_channels') :

if child.in\_channels == 12 and child.out\_channels == 320:

for key, value in local\_state\_dict.items():

if key.startswith(child\_prefix) and key.endswith('weight'):

value = torch.cat((value,value,value), 1)

child\_state\_dict[key] = value

# -------------------------------------------------------------------------------------

以上是12通道的修改方案，9通道类似。

（3）在运用模型的时候我们还需要做进一步的修改，让噪声可以扩展成相应的通道，我们找到 ControlNet-main/cldm/cldm.py文件，找到ControlLDM的类下面的apply\_model函数，line 336 else语句之后写入如下代码：

# -------------------------------------------------------------------------------------

# -------------- 9 channels --------------

if self.control\_model.in\_channels == 9:

if concat\_part == None:

encoder\_posterior = self.encode\_first\_stage(torch.cat(cond['c\_concat'], 1)[:,0:3,:,:])

encode\_control = self.get\_first\_stage\_encoding(encoder\_posterior).detach()

gray = torch.cat(cond['c\_concat'], 1)[:,-1:,:,:]

goal\_shape = [1, 4, x\_noisy.shape[2], x\_noisy.shape[3]]

encode\_gray = torch.nn.functional.interpolate(gray, size=goal\_shape[-2:])

concat\_part = torch.cat((encode\_control,encode\_gray),1)

x\_noisy = torch.cat((x\_noisy,concat\_part),1)

# -------------- 12 channels --------------

elif self.control\_model.in\_channels == 12:

if concat\_part == None:

encoder\_posterior = self.encode\_first\_stage(torch.cat(cond['c\_concat'], 1)[:,0:3,:,:])

encode\_control = self.get\_first\_stage\_encoding(encoder\_posterior).detach()

gray = torch.cat(cond['c\_concat'], 1)[:,-1:,:,:]

gray = torch.cat((gray, gray, gray),1)

encoder\_posterior = self.encode\_first\_stage(gray)

encode\_gray = self.get\_first\_stage\_encoding(encoder\_posterior).detach()

oncat\_part = torch.cat((encode\_control,encode\_gray),1)

x\_noisy = torch.cat((x\_noisy,concat\_part),1)

# -------------------------------------------------------------------------------------

代码加入，无论v1v2v3都可以适用。

（4）在数据集传入时，我们需要将黑白图像和参考图像连接（concat）在一起作为hint，并实现算出每个电影相邻帧的相似度保存在txt文件当中，我们需要在tutrial\_dataset.py文件中加入以下类

class Movies\_Dataset(Dataset):

def \_\_init\_\_(self, path, file\_list, standard, frame):

self.path = path

self.data = []

with open (file\_list,'r',encoding='utf-8') as file:

for num, line in enumerate(file):

line = line.strip('\n').split()

pic\_list = []

for j in range(len(line)):

if float(line[j]) > standard:

index = num - 19 + j + int(j>=19)

if -frame > num + 1 - index or num + 1 - index > frame:

pic\_list.append(index)

if pic\_list != []:

file\_name = str(num + 1).zfill(6) + '.png'

item = {'file':file\_name,'list':pic\_list}

self.data.append(item)

def \_\_len\_\_(self):

return len(self.data)

def \_\_getitem\_\_(self, idx):

item = self.data[idx]

reference = cv2.imread(self.path + '/' + item['file'])

seed = np.random.randint(0, len(item['list']))

target = cv2.imread(self.path + '/' + str(item['list'][seed]).zfill(6) + '.png')

reference = cv2.cvtColor(reference, cv2.COLOR\_BGR2RGB)

target = cv2.cvtColor(target, cv2.COLOR\_BGR2RGB)

gray = cv2.cvtColor(target, cv2.COLOR\_BGR2GRAY)

reference = reference.astype(np.float32) / 255.0

gray = gray.astype(np.float32) / 255.0

gray = np.expand\_dims(gray,axis=2)

hint = np.concatenate((reference,gray), axis=2)

target = (target.astype(np.float32) / 127.5) - 1.0

return dict(jpg=target, txt=' ', hint=hint)

并且在训练文件tutrial\_train\_sd2.py文件中，引入该类，并运用。

from tutorial\_dataset import Movies\_Dataset

dataset = Movies\_Dataset\_v2(data\_path, txt\_path, 0.97, 3)

（5）为了提高实验效率，我们采用了分布式训练，如果写入如下代码

trainer = pl.Trainer(gpus=[1,2,3,4], precision=32, callbacks=[logger])

会报如下错误：

RuntimeError:

Attempt to start a new process before the current process has finished its bootstrapping phase. This probably means that you are on Windows and you have forgotten to use the proper idiom in the main module:

if \_\_name\_\_ == '\_\_main\_\_':

freeze\_support()

The "freeze\_support()" line can be omitted if the program is not going to be frozen to produce a Windows executable.

我们发现，如果修改加速策略，改为dp，代码如下，将不会报上述错误，但dp训练的资源利用率非常低。

trainer = pl.Trainer(gpus=[1,2,3,4], precision=32, accelerator='dp',callbacks=[logger])

我们希望采用ddp的分布式策略，训练代码改成如下，仍会报上述错误，经过查阅相关博客，我们发现在添加main函数后，将不会报错。

trainer = pl.Trainer(gpus=[1,2,3,4], precision=32, accelerator='ddp',callbacks=[logger])

**五、相关实验**

本节评估本文所提出方法在基于参考图的上色效果，首先介绍一下训练的数据集以及实验配置，然后进行方法的相关实验，包括三个版本的对比以及生成方式的实验。

**5.1 数据集以及实验配置**

我们的训练数据来自于5部电影，分别为《叶问》《芳华》《黑豹2》《好莱坞圣诞》《安静的女孩》，将所有的帧提取出来，一共超过12万张照片，测试集采用《西线无战事》中的片段。于每一帧计算相邻n帧之间的相似度，当相似度在比较高范围内，例如0.9到0.99之间，保留图片对作为一组数据，调用当前帧时我们在所有的可行的数据组里随机抽取一组。

若数据组相似度过高，模型趋于将参考图直接输出，对于训练过程的帮助比较小，因此我们会设置相似度的上限和最小间隔帧数。实验过程中，相似度范围为0.94到0.97，最小间隔帧数为3帧。

采用Stable Diffusion v21作为模型原型，实验过程中使用5张48G的A6000进行实验，主干网络完全锁住，batch size选取15，采用ddp分布式加速方式，采用Adam优化器优化。

**5.2 版本测试**

本小节我们对比一下所提出的三个版本，由于计算资源有限，三个版本在较小数据集（8000组数据）上，各训练100次，一个epoch花费12分钟左右，一共需要约18小时，由于训练时间非常短，实验数据的泛化能力比较差，对于数据集内部的数据进行测试，参考图采用第一帧，分别对第二三四帧上色，设置图片尺寸为512，DDIM步数为20，大致效果如下图6所示。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | Reference | Gray Image | Result | Ground Truth |
| v1 |  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| v2 |  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| v3 |  |  |  |  |
|  |  |  |  |
|  |  |  |  |

图6 v1v2v3效果对比图

由上图可知，v2的整体色调和整体的视觉效果会明显好于v1和v3版本，更加接近参考图和Ground Truth，我们认为v1版本未在噪声中混入参考图编码特征，可能学习效率较低，而v3版本中将黑白图像扩展成3通道，再进行VQGAN编码，可能会对模型产生一定程度的误导，而且自编码器本身就没有对3通道黑白图像进行编解码，他的编码解码过程并不可逆，所以也会大大降低训练效率。因此，为了提高训练效率，在大样本进行训练的时候我们选择v2版本进行训练。

**5.3 效果**

我们将v2版本在制作的5部电影的数据集上，训练成本实在太大，在5张A6000上以batch size为15需要约12小时每个epoch，我们训练了30个epoch，用时2周，模型在训练集和测试集上的效果分别如下图7、8所示。

|  |  |  |  |
| --- | --- | --- | --- |
| Reference | Frame 1 | Frame 2 | Frame 3 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

图7 训练集中图像效果展示图

从上图的效果可以看出，我们的方法在完成不改变灰度图像内容的情况下的上色工作，图片各部分的内容信息没有被破坏，上色之后边缘没有偏移或者消失。训练集上的效果相对好一些，电影《芳华》（第四行）效果最好，《圣诞》（第三行）和《黑豹2》（第二行）的效果也比较好，边缘准确而且图片整体色差比较小，进一步训练应该会更好，但在电影《叶问4》上的效果明显比较差，图像质量会大幅下降，主要原因在于边界变得模糊和颜色的轻微偏移，但从下图可以看出，在测试集上的颜色的准确性有待提升，边缘清晰但色差很严重。

|  |  |  |  |
| --- | --- | --- | --- |
| Reference | Frame 1 | Frame 2 | Frame 3 |
|  |  |  |  |
|  |  |  |  |

图8 测试集中图像效果展示图

我们认为我们模型表现不佳，可能的原因有以下三点：（1）自编码器会导致细致图片的细节丢失，例如比较小的人脸上色效果不好；（2）训练集数据的多样性不足，Stable Diffusion Model的训练集总共超过10T，而我们的仅仅100G左右

**5.4 递归生成和隔帧生成**

后续我们还探讨了不同的生成方式，我们假设一段黑白视频的第一帧的彩色图像是已知的，那我们的方法支持两种上色方式，一种是始终以第一帧作为参考帧，我们称为隔帧生成方式，也是我们之前在测试的时候用到的生成方式；另一种与之对应的声场方式是后续生成过程的参考帧都是上一次的生成结果，我们称之为递归生成，两种方法对比效果如下图9所示。

|  |  |  |  |
| --- | --- | --- | --- |
| Reference | Frame 1 | Frame 2 | Frame 3 |
|  |  |  |  |
|  |  |
|  |  |  |  |
|  |  |
|  |  |  |  |
|  |  |

图9 生成方式效果对比图

图9中前两列是参考图和第一帧的生成图像，对于两种生成模式是没有区别的，对于每一组图像上面一行对应隔帧生成方式，下面一行对应用递归生成方式。由上图我们可以清楚地看到，由于递归生成误差的传递和叠加，此种方式会明显增大颜色的偏差，在电影《黑豹2》（第二组图像）中体现尤为明显，第一帧只有极少部分的红色，第二帧第三帧大幅扩散。我们认为递归生成方式目前的效果明显弱于隔帧的生成效果。

**六、实验结果分析**

本次实验测试了基于ControlNet框架完成参考图像上色任务，在内容保持和边缘重合度上表现比较好，但Stable Diffusion无论有多么神奇，他仅仅是一个深度模型，需要学习更多的数据才能表现出更好的效果，我们的模型由于训练不足和数据集多样性的缺陷，上色准确率还有待提升，测试集的迁移上色效果和泛化能力极其不稳定。