

Semantic Image Synthesis with Spatially-adaptive Normalization

林榕桢

摘要

在先前的语义控制生成方法之中，往往直接将语义布局作为输入输入到深度网络，但这种方式在 normalize 层会倾向于“洗掉”语义信息。SPADE 是一种空间自适应标准化层，能够解决上述丢失语义信息的问题，可用于合成给定输入语义的逼真图像。本次在复现 SPADE 过程中发现了几个问题：所给语义较为单一，难以生成复杂图片；使用 encoder 的多模态训练效果不明显。基于以上问题，本次复现中尝试利用预训练模型得到 CLIP 语义，并注入到 SPADE 模块中进行多模态训练，获得更丰富和统一的语义表达和更好的风格迁移结果。此外，还额外实现了实时的交互界面，能够让用户绘画语义内容并同时给出生成结果。

关键词：图像生成；条件图像合成；条件标准化层

1 引言

使用图像渲染技术来渲染一张逼真的图像，往往需要涉及到几何、材料和光线传输等等，构建和编辑虚拟环境既昂贵又耗时。如果我们能通过训练神经网络来渲染逼真的图像，将图形渲染转化为模型的学习和推理问题，就可以将创造的过程简化为在数据集上训练模型来创建新的虚拟环境。用户只需要简单地通过指定目标的语义结构，就可以轻松渲染出逼真的图像。

从给定的语义图中生成目标图片，往往会使用 pix2pix^[1]框架，但该框架可能存在训练不稳定的问题，难以生成具有细节的高清图片。在 pix2pixHD 中^[2]，提出了使用更加鲁棒对抗损失以及多尺度生成器和判别器架构，能够合成 2048×1024 分辨率的逼真图像。但在 SPADE 中论证了 pix2pix 框架将语义作为输入的方法会随着网络标准化而“洗去”语义信息。SPADE 是一种空间自适应标准化层，能够用来解决上述在标准化过程中丢失语义信息的问题，其不仅能合成高分辨率图片，且存在更加丰富的细节。

在复现 SPADE 的过程中，我发现由于给定的语义信息有限，比如人的语义并未详细标注头部、四肢、衣服等等，模型难以生成更加细化的内容；且在不同场景下，相同的语义可能会有截然不同的图像内容，比如在滑雪的人与打棒球的人可能十分不同。另外，原文中利用 encoder 实现多模态的训练结果，在本次实验中所使用的数据集中的效果并不明显。CLIP^[3]是一个基于图像和文本并行的多模态模型，通过计算图像和文本的特征向量相似度来构建训练目标。CLIP 在诸多多模态任务上取得了非常好的效果，例如图像检索，地理定位，视频动作识别等等。综上我尝试使用 CLIP 编码图像特征，并使用该特征为语义场景进行细分，以此来增强语义信息，实现了更加明显的风格迁移效果。

2 相关工作

2.1 条件图像合成

深度生成模型可以学习合成图像。常见的方法包括生成对抗网络(GANs)^[4]和变分自编码器(VAE)^[5]。条件图像合成以多种形式存在，根据输入数据类型不同，可以分为：类条件模型^[6]学习合成给定类别标签的图像；基于文本生成图像的模型^[7]；基于条件GAN的图像到图像的迁移，它的目标是将输入图像从一个域转换到另一个域^[8]。与早期的非参数方法相比^[9]，基于学习的方法通常在测试期间运行更快，产生更真实的结果。

2.2 标准化层

标准化层可以分为无条件标准化层和条件标准化层。无条件标准化层不依赖于外部数据，是现代深度网络的重要组成部分，可以在各种分类器中找到，包括AlexNet^[10]中的局部响应标准化和Inception-v2网络^[11]中的批处理标准化(BatchNorm)。其他流行的规范化层包括实例规范化(InstanceNorm)^[12]、层标准化(LayerNorm)^[13]、组标准化(GroupNorm)^[14]和权重标准化(WeightNorm)^[15]。条件标准化层包括条件批处理标准化(Conditional BatchNorm)^[16]和自适应实例标准化(AdaIN)^[17]。两者都首先用于风格转移任务，后来被用于各种视觉任务。与早期的标准化技术不同，条件标准化层需要外部数据，通常按以下方式操作。首先，会将该层标准化到均值为零方差为一。然后通过外部数据训练一组参数来学习仿射变换，以此调制标准化后的内容。对于风格迁移任务^[16-17]，仿射参数用于控制输出的全局风格，因此该变换在空间上是一致的。相比之下，SPADE的标准化层应用了空间变化的仿射变换，使其适合于从语义掩码合成图像。

3 本文方法

3.1 本文方法概述

给定一张语义图片 $m \in L^{W \times H}$ ，该语义图为一系列整数表示，W 和 H 分别为图片的宽和高，语义图中每个位置表示了对应位置像素的语义内容。我们的目标是学习一个映射函数，可以将输入语义图 m 转换为逼真的图像。具体的 Pipeline 如 1 所示：

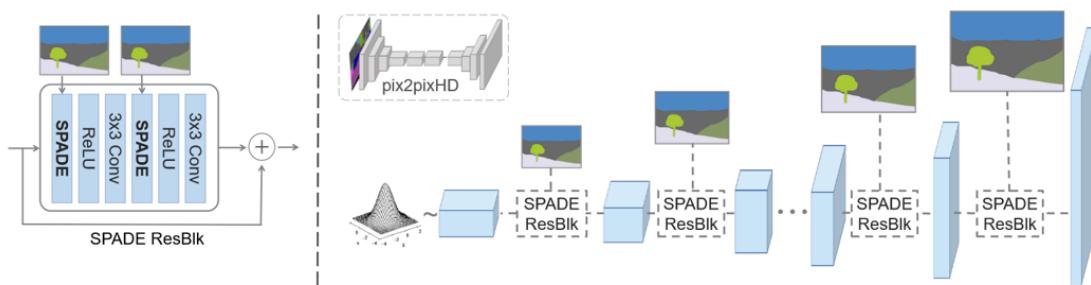


图 1: **Pipeline**. 左边为单个 SPADE ResBlk 的示意图；右边为网络框架

3.2 SPADE

设 h_i 表示一批 N 个样本对深度卷积网络第 i 层的激活。设 C_i 为层中的通道数。设 H_i 和 W_i 为层中激活映射的高度和宽度。SPADE 是一种新的条件归一化方法，称为空间自适应归一化。与批量归一化^[11]相似，将经过激活函数后的结果以通道方式归一化，然后用学习的均值和方差进行调制。2 说明了 SPADE 设计。

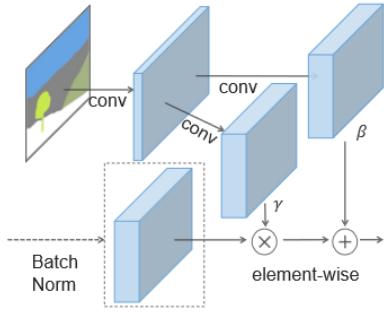


图 2: SPADE

该模块的计算函数如下:

$$\gamma_{c,y,x}^i(m) \frac{h_{n,c,y,x}^i - \mu_c^i}{\sigma_c^i} + \beta_{c,y,x}^i(m), \quad (n \in N, c \in C^i, y \in H^i, x \in W^i) \quad (1)$$

其中 $h_{n,c,y,x}^i$ 为 BatchNorm 前的输入内容, μ_c^i 和 σ_c^i 为 BatchNorm 中各通道 c 的均值和方差:

$$\mu_c^i = \frac{1}{N H^i W^i} \sum_{n,y,x} h_{n,c,y,x}^i \quad (2)$$

$$\sigma_c^i = \sqrt{\frac{1}{N H^i W^i} \sum_{n,y,x} ((h_{n,c,y,x}^i)^2 - (\mu_c^i)^2)} \quad (3)$$

而变量 $\gamma_{c,y,x}^i(m)$ 和 $\beta_{c,y,x}^i(m)$ 是可学习变量。与 InstanceNorm^[11]相比, 它们依赖于输入的语义图, 并随语义所在空间位置变化 (y, x) 而变化。使用符号 $\gamma_{c,y,x}^i$ 和 $\beta_{c,y,x}^i$ 来表示将语义图 m 第 i 层网络中对应 (c, y, x) 的缩放和偏置值的函数。

SPADE 与其他常见的标准化层相比, 可以更好地保护语义信息不被丢失。具体来说, 虽然像 InstanceNorm^[12]这样的归一化层常常被用在各种条件图像合成模型中, 但当应用于均匀或平面的语义图时, 它们往往会洗去语义信息。让我们假设一个简单的模型, 首先将卷积应用到带有单一标签的语义图上 (相同语义具有相同标签), 然后进行标准化。在此设置下, 卷积输出仍然是均匀的, 不同的标签具有不同的均匀值。现在, 在我们将 InstanceNorm 应用于输出之后, 此时无论给定的输入语义标签是什么, 标准化的结果都将变成全零, 语义信息完全丢失。相比之下, SPADE 中的语义图是通过空间自适应调制而不标准化的。只有来自前一层的输入是标准化的。因此, SPADE 可以更好地保存语义信息。它可以在不丢失语义输入信息的情况下享受标准化带来的好处。

3.3 网络架构

使用 SPADE, 不需要将语义图作为生成器的输入, 因为学习到的调制参数已经编码了关于标签布局的足够信息。这种简化产生了一个更轻量级的网络。此外, 生成器可以将随机向量作为输入, 从而实现简单自然的多模态合成。1展示了 SPADE 生成器的网络架构, 它使用了几个带有上采样层的 ResNetBlock。使用 SPADE 学习所有归一化层的调制参数。由于每个残差块都以不同的规模运行, 在不同尺度下会对语义图进行下采样以匹配空间分辨率。SPADE 基于 GAN 网络架构进行训练, 采用了 pix2pixHD^[2]中使用的多尺度鉴别器和损失函数来训练生成器。

3.4 损失函数定义

损失函数的定义参照了 pix2pixHD，其主要由三个方面组成：对抗生成损失、判别器特征损失以及基于 VGG 的感知损失^[18]。假设有 D_1, D_2, D_3 三个不同尺度的判别器，其对抗损失计算公式如下：

$$L_{GAN} = \max_{D_1, D_2, D_3} \sum_{k=1,2,3} L_{GAN}(G, D_k) \quad (4)$$

该对抗损失使用了合页损失 (hinge)，能够帮助网络更稳定的训练。判别器特征损失计算公式如下：

$$L_{FM} = \sum_{k=1,2,3} \mathbf{E}_{(seg, img)} \sum_{i=1}^T \frac{1}{N_i} [||D_k^{(i)}(seg, img) - D_k^{(i)}(seg, G(seg))||_1] \quad (5)$$

其中 (seg, img) 为语义图及其对应的目标图片， T 为判别器层数， N_i 为判别器第 i 层的元素数量。最后的感知损失计算公式如下：

$$L_{VGG} = \mathbf{E}_{(seg, img)} \sum_{i=1}^T \frac{1}{N_i} [||VGG^{(i)}(img) - VGG^{(i)}(G(seg))||_2^2] \quad (6)$$

最后的总体的损失函数如下：

$$L_{ALL} = L_{GAN} + \lambda_1 L_{FM} + \lambda_2 L_{VGG} \quad (7)$$

4 复现细节

4.1 与已有开源代码对比

本次复现主要是基于论文描述，在细节上参考了 SPADE 源码，大部分内容都已在实验中重新实现。以下将分为数据预处理、模型架构、训练过程以及可视化界面来简单分析。

4.1.1 数据预处理

CocoHandler: 本次使用的数据集为 cocostuff，拥有从不同场景捕获的 118,000 张训练图像和 5,000 张验证图像，总计 182 个语义类。需要使用 coco-api 从配置文件中读取对应分类的图片目录 (_get_img_ids)，而后获得对应图片和语义图的目录 (_gen_paths)。根据语义图生成对应的 instance map (gen_inst_map)，以及生成对应图片的 CLIP 特征 (gen_clip)。

```
class CocoHandler:
    def __init__(self, dataset_path, config_path, supercategory, is_train=True) -> None:
        self.coco = COCO(config_path)
        self.status = "train" if is_train else "val"
        self.img_path = os.path.join(dataset_path, f"{self.status}_img")
        self.label_path = os.path.join(dataset_path, f"{self.status}_label")
        self.inst_path = os.path.join(dataset_path, f"{self.status}_inst")
        self.img_ids = self._get_img_ids(supercategory)
        self.img_list, self.label_list, self.inst_list = self._gen_paths()

    def _get_img_ids(self, supercategory):
        img_ids = set()
        coco = self.coco
        if supercategory == "":
            cat_ids = coco.getCatIds()
        else:
            cat_ids = coco.getCatIds(supNms=[supercategory])
        for id in cat_ids:
            img_list = coco.getImgIds(catIds=[id])
            for img in img_list:
                img_ids.add(img)
        img_ids = sorted(list(img_ids))
        return img_ids

    def _gen_paths(self):
        img_list = []
        label_list = []
        inst_list = []
        img_ids = self.img_ids
        for id in img_ids:
            name = self.coco.loadImgs(id)[0]["file_name"].split(".")[0]
            img_list.append(os.path.join(self.img_path, f"{name}.jpg"))
            label_list.append(os.path.join(self.label_path, f"{name}.png"))
            inst_list.append(os.path.join(self.inst_path, f"{name}.png"))
        return img_list, label_list, inst_list

    def _gen_inst_maps(self):
        img_ids = self.img_ids
        def gen_inst_maps(self):
            img_ids = self.img_ids
            for id in img_ids:
                name = self.coco.loadImgs(id)[0]["file_name"].split(".")[0]
                cur_label_path = os.path.join(self.label_path, f"{name}.png")
                map = io.imread(cur_label_path, as_gray=True)
                ann_id = self.coco.getAnnIds(imgIds=id)
                anns = self.coco.loadAnns(ann_id)
                counter = 0
                for ann in anns:
                    if type(ann["segmentation"]) == list and "segmentation" in ann:
                        seg = np.array(ann["segmentation"])
                        poly = np.array(seg).reshape((int(len(seg) / 2), 2))
                        xs, ys = polygon(poly[:, 1] - 1, poly[:, 0] - 1)
                        map[xs, ys] = counter
                counter += 1
            io.imsave(os.path.join(self.inst_path, f"{name}.png"), map)

        def gen_clip(self, device="cpu"):
            import clip
            img_ids = self.img_ids
            clip_arr = []
            model, preprocess = clip.load("ViT-B/32", device=device)
            counter = 0
            for cur_im_path in self.img_list:
                img = Image.open(cur_im_path)
                img = preprocess(img).unsqueeze(0).to(device)
                with torch.no_grad():
                    img_feature = model.encode_image(img)
                    clip_arr.append(img_feature.cpu().numpy()[0])
                counter += 1
                print(f"\r{counter}/{len(img_ids)}", end="\r")
            clip_features = np.array(clip_arr)
            np.save("./clip_feature.npy", clip_features)

        def get_cur_img_ids(self):
            return self.img_ids

        def get_all_list(self):
            return self.img_list, self.label_list, self.inst_list
```

图 3: CocoHandler

CocoDataset: 在利用 CocoHandler 构建好一系列所需数据后，需要继续构建 pytorch 训练使用的 Dataset，需要完成在训练中对数据对进行的随机裁剪和翻转操作。

```

class CocoDataset(data.Dataset):
    def __init__(self, img_list, label_list, inst_list, feature_list=None, status="train") -> None:
        super().__init__()
        self.img_list = img_list
        self.label_list = label_list
        self.inst_list = inst_list
        self.feature_list = feature_list
        self.is_train = True if status == "train" else False
        self.resize_size = 284 if status == "train" else 256
        self.label_nc = 182
        self.transform_list = [
            transforms.Resize((self.resize_size, self.resize_size), interpolation=InterpolationMode.NEAREST),
            transforms.ToTensor(),
        ]
        self.normalize = transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])

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

    def __getitem__(self, index):
        img_path = self.img_list[index]
        label_path = self.label_list[index]
        inst_path = self.inst_list[index]

        transform_list = self.transform_list.copy()
        if self.is_train:
            x = random.randint(0, np.maximum(0, 284 - 256))
            y = random.randint(0, np.maximum(0, 284 - 256))
            transform_list.insert(-1, transforms.Lambda(lambda img: img.crop((x, y, x+256, y+256))))
            if random.random() > 0.5:
                transform_list.insert(-1, transforms.Lambda(lambda img: transforms.functional.vflip(img)))
        transform = transforms.Compose(transform_list)

        img = Image.open(img_path).convert("RGB")
        img = self.normalize(transform(img))
        label = Image.open(label_path)
        label = (transform(label)*255).long()
        label[label == 255] = self.label_nc
        inst = Image.open(inst_path)
        inst = (transform(inst)*255).long()

        if self.feature_list is not None:
            feature = torch.tensor(self.feature_list[index])
            return img, label, inst, feature
        else:
            return img, label, inst

```

```

def get_edge(rank, inst):
    edge = torch.zeros_like(inst).to(rank)
    edge[:, :, :, 1:] = edge[:, :, :, 1:] | (inst[:, :, :, 1:] != inst[:, :, :, :-1])
    edge[:, :, :, :-1] = edge[:, :, :, :-1] | (inst[:, :, :, :-1] != inst[:, :, :, 1:])
    edge[:, :, 1:, :] = edge[:, :, 1:, :] | (inst[:, :, 1:, :] != inst[:, :, 1:, :-1])
    edge[:, :, :-1, :] = edge[:, :, :-1, :] | (inst[:, :, :-1, :] != inst[:, :, 1:, :-1])
    return edge.float()

def get_seg(rank, label, inst=None):
    seg = torch.zeros([label.shape[0], 183, *label.shape[2:]]).to(rank)
    seg = seg.scatter_(1, label, 1.0)
    if inst is None:
        edge = get_edge(rank, inst)
        seg = torch.cat([seg, edge], dim=1)
    return seg

```

图 4: CocoDataset

4.1.2 模型架构

Encoder: Encoder 如 5 中右边的架构图所示，由一系列的下采样的卷积层堆叠而成，最后输出一个 256 维均值与一个 256 维的方差，用于调制后续采样标准高斯分布的噪声。Encoder 在后续使用 CLIP 注入特征时并不会使用，取而代之的将是语义图作为输入，并会使用 CLIP 特征进行调制。

```

class DownSampleBlock(torch.nn.Module):
    def __init__(self, in_c, out_c) -> None:
        super().__init__()
        self.conv = torch.nn.Conv2d(in_c, out_c, 3, 2, 1)
        self.norm = torch.nn.InstanceNorm2d(out_c)

    def forward(self, x):
        return self.norm(self.conv(x))

class Encoder(torch.nn.Module):
    def __init__(self, args) -> None:
        super().__init__()
        in_c = args.img_c
        start_c = args.start_c
        img_size = args.crop_size
        self.scale_time = args.en_downsample_times

        self.in_conv = torch.nn.Conv2d(in_c, start_c, 3, 2, 1)
        self.act_func = torch.nn.LeakyReLU(0.2, False)

        self.downsample_list = torch.nn.ModuleList()
        self.downsample_list.append(DownSampleBlock(start_c, start_c*2))
        self.downsample_list.append(DownSampleBlock(start_c*2, start_c*4))
        self.downsample_list.append(DownSampleBlock(start_c*4, start_c*8))
        self.downsample_list.append(DownSampleBlock(start_c*8, start_c*8))
        self.downsample_list.append(DownSampleBlock(start_c*8, start_c*8))

        self.mu = torch.nn.Linear(512*((img_size//(2**self.scale_time))**2), 256)
        self.sigma = torch.nn.Linear(512*((img_size//(2**self.scale_time))**2), 256)

    def forward(self, x):
        output = self.in_conv(x)
        for i in range(self.scale_time-1):
            output = self.downsample_list[i](output)
            output = self.act_func(output)

        output = output.view(x.shape[0], -1)
        mu = self.mu(output)
        log_s = self.sigma(output)

        return mu, log_s

```



图 5: Encoder

SPADE & SPADEResBlk: 即实现图 2 中的 SPADE 空间自适应标准化层以及图 1 中左侧的 SPADEResBlk。与原论文不同的是，在 SPADE 模块中加入了两个 MLP 层，该 MLP 是为了利用 CLIP 特征为语义图初次卷积后的各个通道分布预测一个仿射变换的均值与方差，以实现对语义进行增强的效果。

```

class SPADE(torch.nn.Module):
    def __init__(self, x_c, seg_c, hiden_c=128, is_clip=False) -> None:
        super().__init__()
        self.hiden_c = hiden_c
        self.is_clip = is_clip
        # self.normalize = torch.nn.BatchNorm2d(x_c, affine=False)
        self.normalize = torch.nn.SyncBatchNorm(x_c, affine=False)
        self.shared = torch.nn.Conv2d(seg_c, hiden_c, 3, 1, 1)
        self.act_func = torch.nn.ReLU(True)
        self.scale = torch.nn.Conv2d(hiden_c, x_c, 3, 1, 1)
        self.bias = torch.nn.Conv2d(hiden_c, x_c, 3, 1, 1)

        if self.is_clip:
            self.clip_mean = torch.nn.Sequential(
                torch.nn.Linear(512, 1024),
                torch.nn.Linear(1024, hiden_c),
            )

            self.clip_std = torch.nn.Sequential(
                torch.nn.Linear(512, 1024),
                torch.nn.Linear(1024, hiden_c),
            )

        def forward(self, x, seg_map, clip_feature=None):
            norm = self.normalize(x)
            seg_map = torch.nn.functional.interpolate(seg_map, x.size()[2:], mode="nearest")

            if self.is_clip:
                output = self.shared(seg_map)
                mean = self.clip_mean(clip_feature).unsqueeze(-1).unsqueeze(-1)
                std = self.clip_std(clip_feature).unsqueeze(-1).unsqueeze(-1)
                output = (1+std)*output + mean
            else:
                output = self.shared(seg_map)
                seg_act = self.act_func(output)
                scale = self.scale(seg_act)
                bias = self.bias(seg_act)
                output = norm * (1+scale) + bias

            return output

class SPADEResBlk(torch.nn.Module):
    def __init__(self, in_c, out_c, args) -> None:
        super().__init__()
        spade_hiden_c = args.spade_hiden_c
        mid_c = min(in_c, out_c)
        self.is_shortcut = not (in_c == out_c)
        self.is_clip = args.is_clip

        self.act_func = torch.nn.LeakyReLU(0.2, True)
        self.spade_0 = SPADE(in_c, args.seg_c, spade_hiden_c, self.is_clip)
        self.conv_0 = torch.nn.Conv2d(in_c, mid_c, 3, 1, 1)
        self.spade_1 = SPADE(mid_c, args.seg_c, spade_hiden_c, self.is_clip)
        self.conv_1 = torch.nn.Conv2d(mid_c, out_c, 3, 1, 1)
        if self.is_shortcut:
            self.spade_cut = SPADE(in_c, args.seg_c, spade_hiden_c)
            self.conv_cut = torch.nn.Conv2d(in_c, out_c, 3, 1, 1, bias=False)
        if args.spectral_norm:
            self.conv_0 = spectral_norm(self.conv_0)
            self.conv_1 = spectral_norm(self.conv_1)
            if self.is_shortcut:
                self.conv_cut = spectral_norm(self.conv_cut)

    def forward(self, x, seg_map, clip_feature=None):
        output = self.spade_0(x, seg_map, clip_feature)
        output = self.act_func(output)
        output = self.conv_0(output)
        output = self.spade_1(output, seg_map, clip_feature)
        output = self.act_func(output)
        output = self.conv_1(output)
        if self.is_shortcut:
            x = self.conv_cut(self.spade_cut(x, seg_map))
        return output + x

```

图 6: SPADE & SPADEResBlk

Generator: 即实现图 1 中右侧的生成器框架。当使用 CLIP 来进行语义增强时，会将语义图作为网络的输入，并使用两个 MLP 来用 CLIP 预测缩放的均值与方差，作用在语义图初次卷积结果的各个通道之上。

```

class Generator(torch.nn.Module):
    def __init__(self, args) -> None:
        super().__init__()
        self.args = args
        self.is_clip = args.is_clip
        self.start_size = self.args.crop_size // (2**self.args.upsample_times)
        self.start_c = self.args.last_spade_c * (2**self.args.c_downsample_times)
        if self.args.is_encode:
            self.in_layer = torch.nn.Linear(self.args.z_c, self.start_c*(self.start_size**2))
        else:
            self.in_layer = torch.nn.Conv2d(self.args.seg_c, self.start_c, 3, 1, 1)

        if self.is_clip:
            self.clip_mean = torch.nn.Sequential(
                torch.nn.Linear(512, 1024),
                torch.nn.Linear(1024, self.start_c),
            )
            self.clip_std = torch.nn.Sequential(
                torch.nn.Linear(512, 1024),
                torch.nn.Linear(1024, self.start_c),
            )

        self.spade_list = torch.nn.ModuleList()
        self.spade_list.append(SPADEResBlk(self.start_c, self.start_c, self.args))
        self.spade_list.append(SPADEResBlk(self.start_c, self.start_c, self.args))
        self.spade_list.append(SPADEResBlk(self.start_c, self.start_c, self.args))
        self.spade_list.append(SPADEResBlk(self.start_c, self.start_c//2, self.args))
        self.spade_list.append(SPADEResBlk(self.start_c//2, self.start_c//4, self.args))
        self.spade_list.append(SPADEResBlk(self.start_c//4, self.start_c//8, self.args))
        self.spade_list.append(SPADEResBlk(self.start_c//8, self.start_c//16, self.args))

        self.upsample = torch.nn.Upsample(scale_factor=2)
        self.conv_out = torch.nn.Conv2d(self.args.last_spade_c, 3, 3, 1, 1)
        self.l_relu = torch.nn.LeakyReLU(0.2, True)
        self.act_func = torch.nn.Tanh()

    def forward(self, x, seg_map, clip_feature=None):
        if self.is_clip:
            if clip_feature == None:
                clip_feature = torch.randn(x.shape[0], 512).cuda()
            output = self.in_layer(x)
            mean = self.clip_mean(clip_feature).unsqueeze(-1).unsqueeze(-1)
            std = self.clip_std(clip_feature).unsqueeze(-1).unsqueeze(-1)
            output = (1+std)*output + mean
        else:
            output = self.in_layer(x)
            if self.args.is_encode:
                output = output.view(-1, self.start_c, self.start_size, self.start_size)

        for i in range(self.args.upsample_times):
            output = self.spade_list[i](output, seg_map, clip_feature)
            output = self.upsample(output)
            output = self.spade_list[-1](output, seg_map, clip_feature)
            output = self.l_relu(output)
            output = self.act_func(self.conv_out(output))
        return output

```

图 7: Generator

Discriminator: Discriminator 如图 8 中右边的架构图所示，在将图片或生成结果与其对应的语义图在通道上进行拼接后，传入一系列的下采样的卷积层得到输出结果，该输出具有空间维度，能够使得训练更加稳定。不同于使用多个不同尺度的判别器，本次实验中只使用了一个判别器。

```

class DownSampleBlock(torch.nn.Module):
    def __init__(self, in_c, out_c, stride=2) -> None:
        super().__init__()
        self.conv = spectral_norm(torch.nn.Conv2d(in_c, out_c, 4, stride, 1))
        self.norm = torch.nn.InstanceNorm2d(out_c)

    def forward(self, x):
        return self.norm(self.conv(x))

class Discriminator(torch.nn.Module):
    def __init__(self, args) -> None:
        super().__init__()
        self.in_c = args.img_c + args.seg_c
        self.start_c = args.start_c
        self.act_func = torch.nn.LeakyReLU(0.2)
        self.conv_in = torch.nn.Conv2d(self.in_c, self.start_c, 4, 2, 1)

        self.downsample_list = torch.nn.ModuleList()
        self.downsample_list.append(DownSampleBlock(self.start_c, self.start_c*2))
        self.downsample_list.append(DownSampleBlock(self.start_c*2, self.start_c*4))
        self.downsample_list.append(DownSampleBlock(self.start_c*4, self.start_c*8, 1))

        self.conv_out = torch.nn.Conv2d(self.start_c*8, 1, 4)

    def forward(self, x):
        output = self.act_func(self.conv_in(x))
        res = [output]
        for i in range(3):
            output = self.downsample_list[i](output)
            output = self.act_func(output)
            res.append(output)
        output = self.conv_out(output)
        res.append(output)
        return res

```

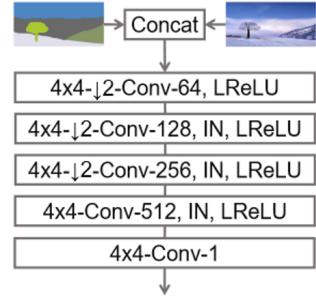


图 8: Discriminator

Model: 在 Model 类里将上述所有网络架构整合起来并构建成为完整的模型。实现了模型的保存读取，以及权重的初始化功能。

```

class Model(torch.nn.Module):
    def __init__(self, is_train=True) -> None:
        super().__init__()
        self.args = get_model_args()
        self.is_encode = self.args.is_encode
        self.is_clip = self.args.is_clip
        self.is_train = is_train

        self.net_g = Generator(self.args)
        if self.is_encode:
            self.encoder = Encoder(self.args)
        if self.is_train:
            self.net_d = Discriminator(self.args)
        self.init_weights('xavier')

    def encode_img(self, img):
        mu, log_s = self.encoder(img)
        return mu, log_s

    def get_noise(self, mu, log_s):
        std = (0.5*log_s).exp()
        eps = torch.randn_like(std)
        z = eps.mul(std) + mu
        return z

    def gen_img(self, z, seg, feature=None):
        return self.net_g(z, seg, feature)

    def det_img(self, img, seg):
        img_seg = torch.cat([img, seg], dim=1)
        score = self.net_d(img_seg)
        return score

    def forward(self, seg, rank=0, img=None, clip_feature=None):
        if self.is_encode:
            if not img is None:
                mu, log_s = self.encoder(img)
                z = self.get_noise(mu, log_s)
            else:
                z = torch.randn([seg.shape[0], 256]).to(rank)
        else:
            size = self.net_g.start_size
            z = torch.nn.functional.interpolate(seg, (size, size))
            if clip_feature is not None:
                return self.gen_img(z, seg, clip_feature)

        return self.gen_img(z, seg)

    def save(self, save_path):
        if not os.path.exists(save_path):
            os.makedirs(save_path)
        torch.save(self.net_g.state_dict(), save_path + "/net_g.pth")
        torch.save(self.net_d.state_dict(), save_path + "/net_d.pth")
        if self.is_encode:
            torch.save(self.encoder.state_dict(), save_path + "/encoder.pth")

    def load(self, save_path):
        self.net_g.load_state_dict(torch.load(save_path + "/net_g.pth"))
        if self.is_encode:
            self.encoder.load_state_dict(torch.load(save_path + "/encoder.pth"))
        if self.is_train:
            self.net_d.load_state_dict(torch.load(save_path + "/net_d.pth"))

    def init_weights(self, init_type='normal', gain=0.02):
        def init_func(m):
            classname = m.__class__.__name__
            if classname.find('BatchNorm2d') != -1 or classname.find('SyncBatchNorm') != -1:
                if hasattr(m, 'weight') and m.weight is not None:
                    init.normal_(m.weight.data, 1.0, gain)
                if hasattr(m, 'bias') and m.bias is not None:
                    init.constant_(m.bias.data, 0.0)
            elif classname.find('weight') and (classname.find('Conv') != -1 or classname.find('Linear') != -1):
                if init_type == 'normal':
                    init.normal_(m.weight.data, 0.0, gain)
                elif init_type == 'xavier':
                    init.xavier_normal_(m.weight.data, gain=gain)
                elif init_type == 'xavier_uniform':
                    init.xavier_uniform_(m.weight.data, gain=1.0)
                elif init_type == 'kaiming':
                    init.kaiming_normal_(m.weight.data, a=0, mode='fan_in')
                elif init_type == 'orthogonal':
                    init.orthogonal_(m.weight.data, gain=gain)
                elif init_type == 'none': # uses pytorch's default init method
                    m.reset_parameters()
                else:
                    raise NotImplementedError(f'initialization method [{init_type}] is not implemented')
            if hasattr(m, 'bias') and m.bias is not None:
                init.constant_(m.bias.data, 0.0)

        self.apply(init_func)

```

图 9: Model

4.1.3 训练设置

损失函数: 如章节 3.4 中介绍了损失函数一样，此处实现了所有损失函数的计算。此外还有 KL 散度的损失计算，被用于计算 Encoder 结果的损失，是一个正则项。

```

class GAN_Loss(torch.nn.Module):
    def __init__(self, gan_mode):
        super().__init__()
        self.gan_mode = gan_mode
        self.real_label = None
        self.fake_label = None

    def get_target_label(self, score, flag):
        if flag:
            if self.real_label is None or self.real_label.shape != score.shape:
                self.real_label = torch.ones_like(score)
            self.real_label.requires_grad_(False)
        return self.real_label
    else:
        if self.fake_label is None or self.fake_label.shape != score.shape:
            self.fake_label = torch.zeros_like(score)
            self.fake_label.requires_grad_(False)
        return self.fake_label

    def loss(self, score, flag, for_gan=False):
        cur_target = self.get_target_label(score, flag)
        loss = None
        if self.gan_mode == "bce":
            loss = torch.nn.functional.binary_cross_entropy_with_logits(score, cur_target)
        elif self.gan_mode == "mse":
            loss = torch.nn.functional.mse_loss(score, cur_target)
        elif self.gan_mode == "hinge":
            if for_gan:
                loss = -torch.mean(score)
            else:
                if flag:
                    minval = torch.min(score - 1, torch.zeros_like(score))
                    loss = -torch.mean(minval)
                else:
                    minval = torch.min(score + 1, torch.zeros_like(score))
                    loss = -torch.mean(minval)
        else:
            print("Unknown gan mode!")
        return loss

    def forward(self, input, flag, for_gan=False):
        if isinstance(input, list):
            loss = 0
            for pred_i in input:
                if isinstance(pred_i, list):
                    pred_i = pred_i[-1]
                loss_tensor = self.loss(pred_i, flag, for_gan)
                bs = 1 if len(loss_tensor.size()) == 0 else loss_tensor.size(0)
                new_loss = torch.mean(loss_tensor.view(bs, -1), dim=1)
                loss += new_loss
            loss /= len(input)
        else:
            return self.loss(input, flag, for_gan)

class KLD_Loss(torch.nn.Module):
    def __init__(self) -> None:
        super().__init__()
    def forward(self, mu, log_s):
        return -0.5 * torch.sum(1 + log_s - mu.pow(2) - log_s.exp())

class VGG_Loss(torch.nn.Module):
    def __init__(self) -> None:
        super().__init__()
        vgg_features = vgg19(weights='DEFAULT').features.eval()
        self.blocks = torch.nn.ModuleList()
        self.blocks.append(vgg_features[:2])
        self.blocks.append(vgg_features[2:7])
        self.blocks.append(vgg_features[7:12])
        self.blocks.append(vgg_features[12:21])
        self.blocks.append(vgg_features[21:30])
        for block in self.blocks:
            for p in block.parameters():
                p.requires_grad = False

        self.weights = [1.0/32, 1.0/16, 1.0/8, 1.0/4, 1.0]

    def forward(self, real, fake):
        x = fake
        y = real
        loss = 0.0
        cur = 0
        for block in self.blocks:
            x = block(x)
            y = block(y)
            loss += self.weights[cur]*torch.nn.functional.l1_loss(x, y.detach())
            cur += 1
        return loss

class GAN_FLoss(torch.nn.Module):
    def __init__(self) -> None:
        super().__init__()
        self.losser = torch.nn.L1Loss()
    def forward(self, real_score, fake_score):
        gan_floss = 0
        n = len(fake_score)-1
        for i in range(n):
            gan_floss += self.losser(fake_score[i], real_score[i].detach()) / n
        return gan_floss

```

图 10: Loss

训练过程: 具体的训练过程如下。采用了 DDP 分布式训练，生成器判别器交替训练。使用 CLIP 特征时需要对训练集的 CLIP 特征做一次 K 聚类，以供推理时使用。

```

setup(rank, world_size)
os.environ['MASTER_ADDR'] = '127.0.0.1'
os.environ['MASTER_PORT'] = '64087'
dist.init_process_group("nccl", rank=rank, world_size=world_size)
if (rank == 0):
    dist.destroy_process_group()

def train(rank, model, epoch, train_loader, sampler, vgg_losser, optimizer_g, optimizer_d):
    model.train()
    sampler.set_epoch(epoch)

    gen_loss = 0
    fake_loss = 0
    real_loss = 0

    for batch_idx, data in enumerate(train_loader):
        if model.module.is_encode:
            real_label, fake_label, inst, feature = data
            feature = feature.to(rank)
            inst = inst.to(rank)
            real_img, label, inst = data
            feature = None
            real_label = real_label.to(rank)
            fake_label = fake_label.to(rank)
            inst = inst.to(rank)
            seg = get_seg(fake_label, inst)

            # Encoders
            if model.module.is_encode:
                seg = model.module.module.encode_img(seg)
                seg = model.module.module.encode_img(real_img)
                seg = model.module.module.encode_img(fake_label)
            else:
                seg = model.module.net_g.start_size
                z = torch.nn.functional.interpolate(seg, (size, size))

            fake_img = model.module.gen_img(seg, feature)

            # Losses
            real_score = model.module.det_img(real_img, seg)
            fake_score = model.module.det_img(fake_img, seg)

            # Loss
            vgg_loss = vgg_losser(fake_img, fake_img)
            fake_loss += fake_score.item()
            real_loss += real_score.item()
            gen_loss += gen_loss.item()

            if model.module.is_encode:
                kid_losser = kid_losser_kid
                kid_loss = kid_losser_kid(seg, real_img, log)
                gen_loss = gen_loss + 0.05 * kid_loss + 10 * vgg_loss + 10 * gen_floss
            else:
                gen_loss = gen_loss + 10 * vgg_loss + 10 * gen_floss

            optimizer_g.zero_grad()
            gen_loss.backward()
            optimizer_g.step()

            if rank == 0:
                print("Epoch: %d, Iteration: %d, Loss: %f" % (epoch, batch_idx, gen_loss))

        else:
            real_score = model.module.det_img(real_img, seg)
            fake_score = model.module.det_img(fake_img, seg)

            # Loss
            vgg_loss = vgg_losser(fake_img, fake_img)
            fake_loss += fake_score.item()
            real_loss += real_score.item()
            gen_loss += gen_loss.item()

            if model.module.is_encode:
                kid_losser = kid_losser_kid
                kid_loss = kid_losser_kid(seg, real_img, log)
                gen_loss = gen_loss + 0.05 * kid_loss + 10 * vgg_loss + 10 * gen_floss
            else:
                gen_loss = gen_loss + 10 * vgg_loss + 10 * gen_floss

            optimizer_g.zero_grad()
            gen_loss.backward()
            optimizer_g.step()

            if rank == 0:
                print("Epoch: %d, Iteration: %d, Loss: %f" % (epoch, batch_idx, gen_loss))

    if rank == 0:
        print("Epoch: %d, Loss: %f" % (epoch, gen_loss))

def main(rank, world_size):
    setup(rank, world_size)

    if rank == 0:
        print("Rank: %d, World Size: %d" % (rank, world_size))
        model = Model()
        start_epoch = 0
        if os.path.exists(model.module.get_model_path()):
            print("Load pre-trained model!")
            model = torch.load(model.module.get_model_path())
        else:
            print("Train from scratch!")

        model = Model()
        model = model.to(rank)
        model = model.module.to(rank)
        model = model.module.to(device_id[rank])
        parameter_a = []
        parameter_b = []
        for parameter in model.named_parameters():
            if parameter[0].split('.')[0] == 'net_g':
                parameter_a.append(parameter[1])
            else:
                parameter_b.append(parameter[1])

        parameter_a.append(parameter_b)
        parameter_a.append(parameter_b)

        # Data
        supercategory = "sports"
        # Data config
        dataset_root = "/home/ubuntu/opendata/dataset"
        train_config = "%s/dataset/annotation/instances_train2017.json" % dataset_root
        img_list, label_list, test_list = CocoDataset(dataset_path, train_config, supercategory).get_all_list()
        if model.module.is_encode:
            train_dataset = CocoDataset(img_list, label_list, test_list, FeatureList=feature_list)
        else:
            train_dataset = CocoDataset(img_list, label_list, test_list, FeatureList=feature_list)

        train_sampler = DistributedSampler(train_dataset, rank=rank, num_replicas=world_size, shuffle=True)
        train_kwargs = {'batch_size': args.batch_size, 'sampler': train_sampler}
        collate_fn = lambda x: x[0]
        train_dataloader = torch.utils.data.DataLoader(train_dataset, **train_kwargs)
        train_dataloader = torch.utils.data.DataLoader(train_dataset, **train_kwargs)

        vgg_losser = VGGLoss(10).to(rank)

        os.environ['CUDA_VISIBLE_DEVICES'] = "4"
        if __name__ == '__main__':
            args = get_base_args()
            torch.manual_seed(args.seed)
            WORLD_SIZE = torch.cuda.device_count()
            np.random.seed(args.seed)
            args(WORLD_SIZE, args),
            nprocess=WORLD_SIZE,
            join=True)

```

图 11: Train

4.1.4 交互界面

PaintBoard: 该类完成了画板功能，包括变换笔画颜色与粗细、使用橡皮擦、清空画板、保存画板等操作。其中不同颜色代表了不同的语义标签。为了能够不借助其他文件生成语义图和 instance map，使用了一个隐藏的画板去额外辅助生成，利用 RGB 通道中的 R 记录语义标签，B 通道记录 instance map。

```

class PaintBoard(QWidget):
    def __init__(self, Parent=None):
        ...
        super().__init__(Parent)
        self.parent = Parent
        self._InitData()
        self._InitView()

    def _InitData(self):
        self._size = QSize(512,512)

        self._board = QPixmap(QSize(512,512))
        self._board.fill(QColor(255, 255, 255))
        self._hidden_board = QPixmap(QSize(512,512))
        self._hidden_board.fill(QColor(255, 0, 255))
        self._hidden_board.toImage().save("temp.png")

        self._IsEmpty = True
        self.EraseMode = False

        self._lastPos = QPoint(0,0)
        self._currentPos = QPoint(0,0)

        self._painter = QPainter()
        self._painter_1 = QPainter()

        self._thickness = 30
        self._colorList = QColor.colorNames()
        self._penColor = QColor(self._colorList[0])
        self._penColor_1 = QColor(1,0,0)

    def _InitView(self):
        self.setFixedSize(self._size)
        self.Clear(self):
            self._board.fill(QColor(255, 255, 255))
            self._hidden_board.fill(QColor(255, 0, 255))
            self._update()
            self._IsEmpty = True

        def ChangePenColor(self, color, color_1):
            self._penColor = color
            self._penColor_1 = color_1

        def ChangePenThickness(self, thickness=10):
            self._thickness = thickness

        def IsEmpty(self):
            return self._IsEmpty

        def GetContentAtQImage(self):
            image = self._hidden_board.toImage()
            return image

        def paintEvent(self, paintEvent):
            self._painter_1.begin(self)
            self._painter_1.drawPixmap(0,0,self._hidden_board)
            self._painter_1.end()
            self._painter.begin(self)
            self._painter.drawPixmap(0,0,self._board)
            self._painter.end()

        def mousePressEvent(self, mouseEvent):
            self._currentPos = mouseEvent.pos()
            self._lastPos = self._currentPos

        def mouseReleaseEvent(self, mouseEvent):
            self._IsEmpty = False
            self.parent.update_img()

    def mouseMoveEvent(self, mouseEvent):
        self._currentPos = mouseEvent.pos()
        self._lastPos = self._currentPos

        if self.EraseMode == False:
            self._painter_1.setPen(QPen(self._penColor, self._thickness))
        else:
            self._painter_1.setPen(QPen(QColor(255, 0, 255),self._thickness))

        self._painter_1.drawLine(self._lastPos, self._currentPos)
        self._painter.end()

        self._lastPos = self._currentPos
        self._update()
        self._hidden_board.toImage().save("temp.png")

```

图 12: PaintBoard

ImageHandler: 该类实现了交互界面与模型通讯的接口。具体有生成图片、生成 instance map、生成边界图等等。

```

class ImageHandler():
    def __init__(self, model, clip_model, clip_preprocess, k_means=None) -> None:
        self.model = model
        self.clip_model = clip_model
        self.clip_preprocess = clip_preprocess
        self.transforms = transforms.Compose([
            transforms.Resize([256, 256],
                            interpolation=transforms.InterpolationMode.NEAREST),
            transforms.ToTensor()
        ])

        def gen_img(self, label_path, img_feature):
            label = Image.open(label_path)
            seg = (self.transforms(label)*255).long()
            label = seg[0,:,:]
            label[label==255] = 182
            label = label.unsqueeze(0).unsqueeze(0).cuda()
            inst = seg[1,:,:]
            inst = inst.unsqueeze(0).unsqueeze(0).cuda()
            seg = get_seg(0, label, inst)
            # img = model(seg, img=img_feature)
            img = self.model(seg, clip_feature=img_feature)
            img = ((img+1)/2).clamp_(0, 1)
            transforms.ToPILImage()(img.squeeze(0).cpu()).save("gen_img.png")

        def get_inst(self, label):
            arr = []
            inst = torch.zeros_like(label)
            for i in range(label.size()[1]):
                for j in range(label.size()[2]):
                    cur_pix = label[0, i, j].item()
                    if cur_pix == 182:
                        continue
                    if cur_pix not in arr:
                        arr.append(cur_pix)
                        cur_index = arr.index(cur_pix)+1
                        inst[0, i, j] = cur_index
            return inst

        def get_edge(self, inst):
            edge = torch.zeros_like(inst)
            edge[:, :, :, 1:] = edge[:, :, :, :-1] | (inst[:, :, :, 1:] != inst[:, :, :, :-1])
            edge[:, :, :, :-1] = edge[:, :, :, :-1] | (inst[:, :, :, :-1] != inst[:, :, :, 1:])
            edge[:, :, 1:, :] = edge[:, :, 1:, :] | (inst[:, :, 1:, :] != inst[:, :, :-1, :])
            edge[:, :, :-1, :] = edge[:, :, :-1, :] | (inst[:, :, :-1, :] != inst[:, :, 1:, :])
            return edge.float()

        def get_feature(self, img_path):
            img = Image.open(img_path)
            img = self.clip_preprocess(img).unsqueeze(0)
            with torch.no_grad():
                clip_feature = self.clip_model.encode_image(img)

            return clip_feature.cuda()

        def gen_from_seg(self, label_path, img_feature):
            label = Image.open(label_path)
            label = (self.transforms(label)*255).long()
            label[label==255] = 182

            inst = self.get_inst(label)
            inst = inst.unsqueeze(0)
            edge = self.get_edge(inst)

            label = label.unsqueeze(0)
            seg = torch.zeros([label.shape[0], 183, *label.shape[2:]])
            seg = seg.scatter_(1, label, 1.0)
            seg = torch.cat([seg, edge], dim=1).cuda()

            self.model.eval()

            gen = self.model(seg, clip_feature=img_feature)
            gen = ((gen+1)/2).clamp_(0, 1)
            transforms.ToPILImage()(gen.squeeze(0).cpu()).save("gen_img.png")

```

图 13: ImageHandler

MainWidget: 该类使用 PyQt6 实现了交互界面，包括画板、生成结果展示、读取语义图并生成、选取参考图、保存成果等等功能。

图 14: MainWidget

4.2 实验环境搭建

本次实验基于 conda 环境搭建，具体如下：GPU: Nvidia P4000 两张；Python:3.10.4；Pytorch:11.4.1；CUDA:11.4；所使用的库可以在提供的代码中的 requirements 中获得。

4.3 界面分析与使用说明

4.3.1 模型训练

模型训练需要准备 coco-stuff 数据集。将数据集解压后需要在 train.py 中修改数据集位置以及数据集配置文件目录，如下图所示：

```
dataset_path = "/home/ubuntu/spade/dataset"
train_config = "/home/ubuntu/spade/dataset/annotations/instances_train2017.json"
```

图 15: Dataset Path

修改后可直接回到程序根目录下使用”python train.py”开始运行训练模型。

4.3.2 交互界面

本次复现提供了实时的交互界面，可供用户绘画任意语义图并实时展示生成结果。可在程序根目录下使用”python main.py”运行交互界面，界面如下图：。

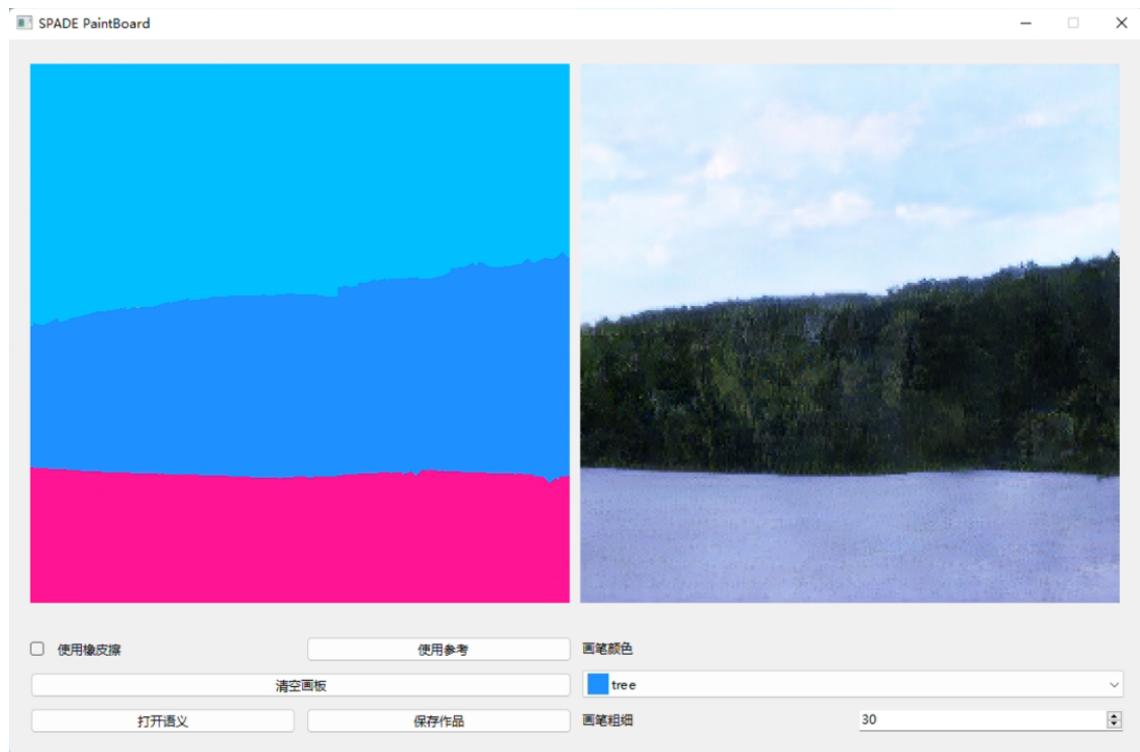


图 16: UI

左上方区域为语义画板，用户可任意画出心仪的语义图并在右上方实时生成结果。用户可通过右下角改变笔画颜色的同时改变语义标签，此外还提供了橡皮擦、笔画粗细调整以及一键清空画板的操作。左下角的“使用参考”按钮可以让用户选定任意参考图片，并使对该图片生成 CLIP 特征来指导图片的生成。”打开语义“按钮可以让用户选定已有的语义图进行生成。”保存作品“按钮可以保存生成的作品。

4.4 创新点

4.4.1 使用 CLIP 特征增强语义

在对 SPADE 进行复现过程中，发现了 SPADE 存在的几个问题：1) 相同语义标签可能拥有多种不同语义内容，比如不同场景下的人穿着完全不同的衣服；2) 语义标签不够精细，模型无法生成合理的内容，比如人的五官、楼宇的窗户等；3) 由于使用的数据集（coco-stuff）多模态内容不明显，导致其多模态结果不是特别明显。针对以上问题，本次复现中尝试使用 CLIP 特征值来对语义进行增强。由于 CLIP 是一个图像文本对特征，给定一张图像的 CLIP 特征，在一定程度上能够由几个关键词来描述这张图片的内容，这几个关键词就构成了图片的一个场景。利用 CLIP 特征值对语义进行增强，实际上是对数据集场景做了一次进一步的细分，在棒球场上的人应该穿棒球服，在雪地里的人应该穿滑雪服等等。CLIP 特征值的注入过程如下图所示：

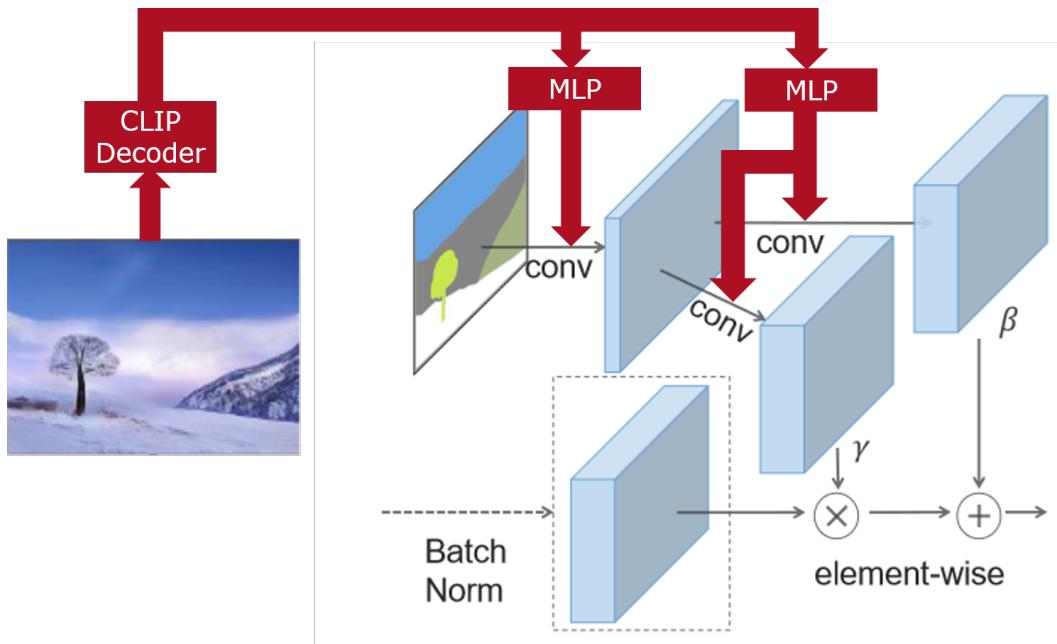


图 17: CLIP 注入

将图片输入预训练的 CLIP Decoder 之中获得 CLIP 特征，该特征值会被送入两个 MLP 分别得到两组均值与方差。该均值方差会被送到箭头所指的位置在通道维度上进行缩放，以达到增强语义的目的。

4.4.2 交互界面

如 4.3.2 中介绍，本次复现提供了功能丰富的实时交互界面，可供用户随心画出语义图并实时生成结果。

5 实验结果分析

本次实验共分为两个部分，第一部分为 SPADE 原文复现部分，包括了复现结果展示、与原文结果对比、多模态结果展示；第二部分为加入 CLIP 特征值进行语义图增强，包括了结果展示、与原文多模态的对比等等。图 18 展示了大量复现结果，可以看到大部分情况下，本次复现能够生成符合语义且较为真实的图片结果。在图 19 中展示了本次复现结果与 SPADE 原文以及其他方法的对比，从结果中可以得到，本次复现结果与原论文效果较为一致，比其他的方法，如 pix2pixHD，有着很大的优势。图 20 展示了多模态的生成结果。



图 18: SPADE 复现结果

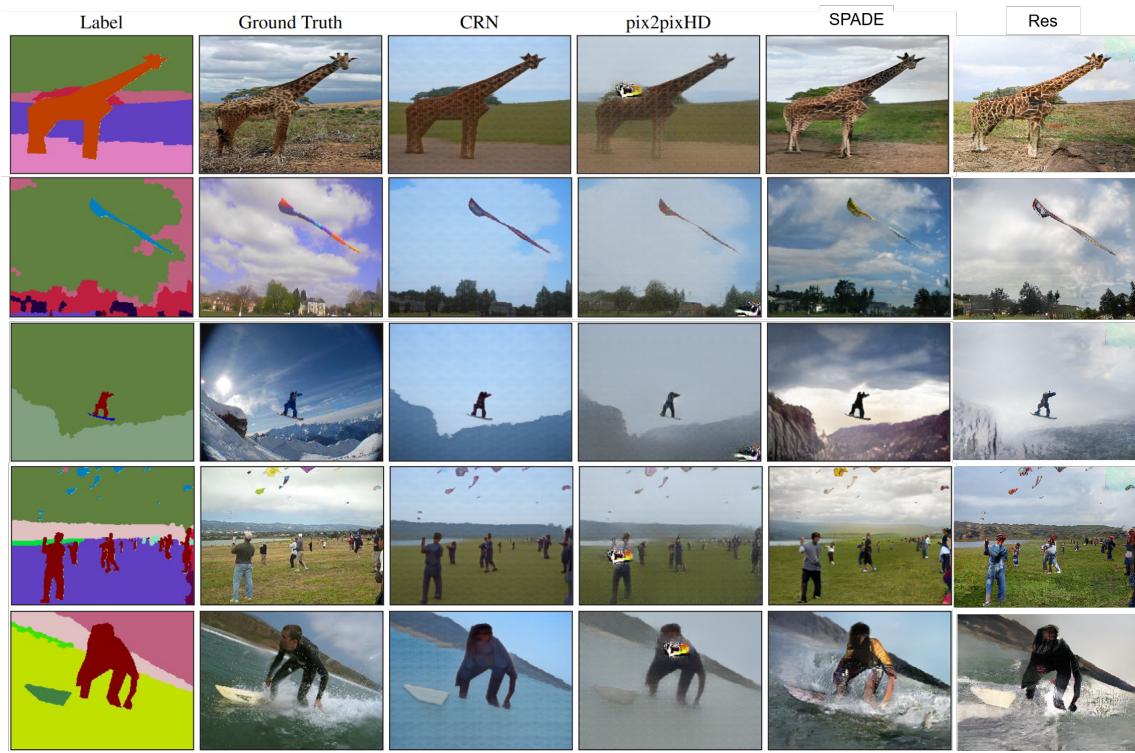


图 19: 方法对比



图 20: 多模态结果

接下来将会展示注入 CLIP 特征后的结果。图 21展示了注入 CLIP 特征进行语义增强的结果，可以看到，结果能够正确生成诸如人的躯干、船的船帆等等原本语义标签一致的内容，且生成效果十分真实。图 22展示了利用 CLIP 特征进行多模态学习的结果，对比原文多模态的效果，可以看到使用 CLIP 能够实现更加明显的风格化结果。图 23展示了更多使用 CLIP 特征进行风格化的结果。

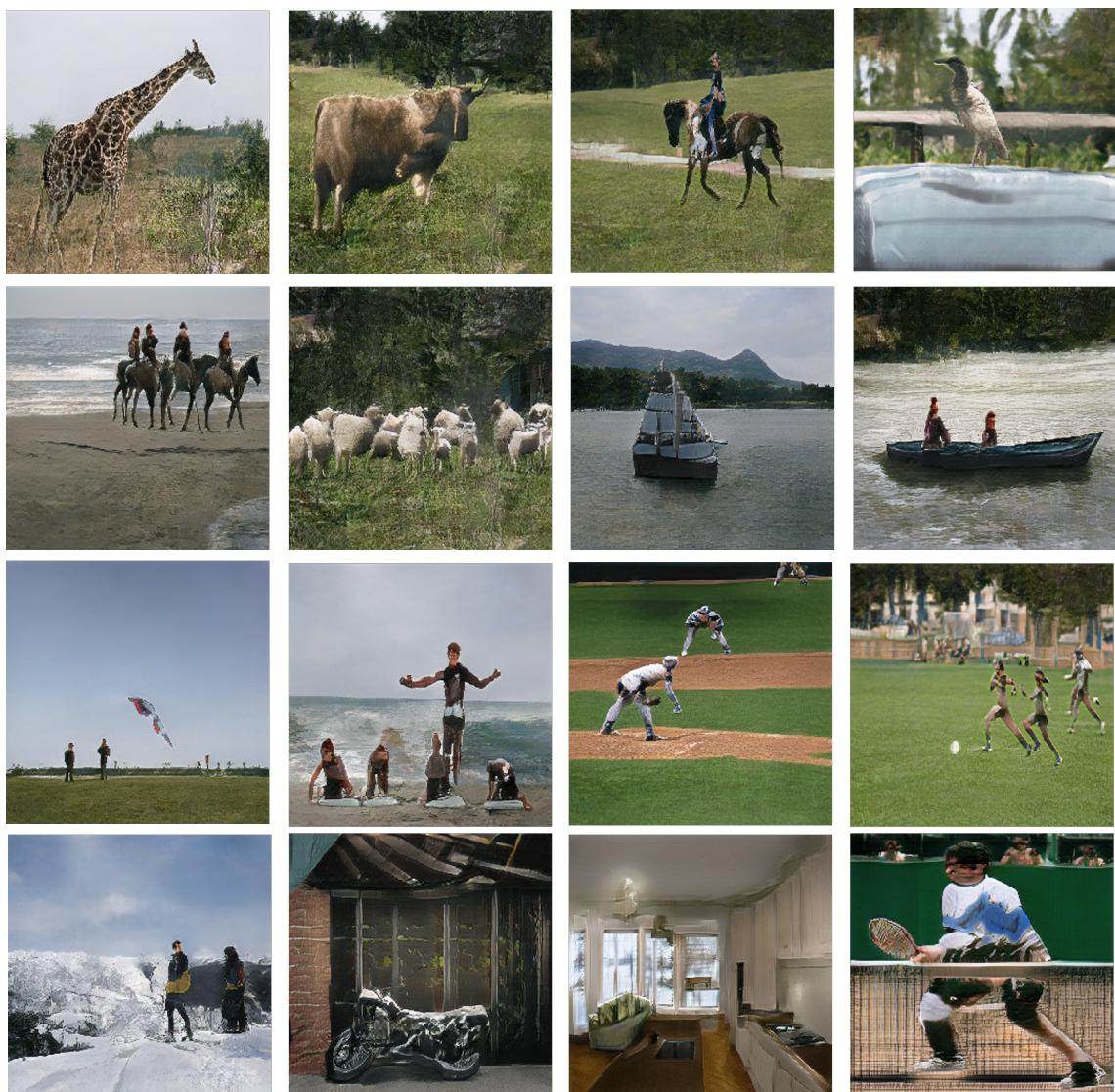


图 21: 注入 CLIP 的生成结果



图 22: 注入 CLIP 的多模态结果



图 23: 注入 CLIP 的风格化结果

6 总结与展望

本文简要介绍了本次复现论文 Semantic Image Synthesis with Spatially-Adaptive Normalization 的背景以及相关工作，并详细介绍了 SPADE 算法的工作原理。随后详细展示了本次复现工作的代码细节，并展示了大量实验结果。本次复现结果在效果上与原论文方法相近，并加入了 CLIP 特征值以及实时交互界面作为创新点。代码实现上，有些细节还需要进行修改。实验展示方面，由于时间关系，缺少加入 CLIP 特征后与原方法更多的对比，这方面可能会在后续工作中补足。

参考文献

- [1] ISOLA P, ZHU J Y, ZHOU T, et al. Image-to-image translation with conditional adversarial networks [C]// Proceedings of the IEEE conference on computer vision and pattern recognition. 2017: 1125-1134.
- [2] WANG T C, LIU M Y, ZHU J Y, et al. High-resolution image synthesis and semantic manipulation with conditional gans[C]// Proceedings of the IEEE conference on computer vision and pattern recognition. 2018: 8798-8807.
- [3] RADFORD A, KIM J W, HALLACY C, et al. Learning transferable visual models from natural language supervision[C]// International Conference on Machine Learning. 2021: 8748-8763.
- [4] GOODFELLOW I, POUGET-ABADIE J, MIRZA M, et al. Generative adversarial networks[J]. Communications of the ACM, 2020, 63(11): 139-144.
- [5] KINGMA D P, WELLING M. Auto-encoding variational bayes[J]. arXiv preprint arXiv:1312.6114, 2013.
- [6] BROCK A, DONAHUE J, SIMONYAN K. Large scale GAN training for high fidelity natural image synthesis[J]. arXiv preprint arXiv:1809.11096, 2018.
- [7] HONG S, YANG D, CHOI J, et al. Inferring semantic layout for hierarchical text-to-image synthesis[C]

- // Proceedings of the IEEE conference on computer vision and pattern recognition. 2018: 7986-7994.
- [8] HUANG X, LIU M Y, BELONGIE S, et al. Multimodal unsupervised image-to-image translation[C] // Proceedings of the European conference on computer vision (ECCV). 2018: 172-189.
- [9] CHEN T, CHENG M M, TAN P, et al. Sketch2photo: Internet image montage[J]. ACM transactions on graphics (TOG), 2009, 28(5): 1-10.
- [10] KRIZHEVSKY A, SUTSKEVER I, HINTON G E. Imagenet classification with deep convolutional neural networks[J]. Communications of the ACM, 2017, 60(6): 84-90.
- [11] IOFFE S, SZEGEDY C. Batch normalization: Accelerating deep network training by reducing internal covariate shift[C]// International conference on machine learning. 2015: 448-456.
- [12] ULYANOV D, VEDALDI A, LEMPITSKY V. Instance normalization: The missing ingredient for fast stylization[J]. arXiv preprint arXiv:1607.08022, 2016.
- [13] BA J L, KIROS J R, HINTON G E. Layer normalization[J]. arXiv preprint arXiv:1607.06450, 2016.
- [14] WU Y, HE K. Group normalization[C]// Proceedings of the European conference on computer vision (ECCV). 2018: 3-19.
- [15] SALIMANS T, KINGMA D P. Weight normalization: A simple reparameterization to accelerate training of deep neural networks[J]. Advances in neural information processing systems, 2016, 29.
- [16] DUMOULIN V, SHLENS J, KUDLUR M. A learned representation for artistic style[J]. arXiv preprint arXiv:1610.07629, 2016.
- [17] HUANG X, BELONGIE S. Arbitrary style transfer in real-time with adaptive instance normalization [C]// Proceedings of the IEEE international conference on computer vision. 2017: 1501-1510.
- [18] JOHNSON J, ALAHI A, FEI-FEI L. Perceptual losses for real-time style transfer and super-resolution [C]// European conference on computer vision. 2016: 694-711.