CoCosNet v2: Full-Resolution Correspondence Learning for Image Translation

Xingran Zhou

摘要

CoCosNet v2 是用与图像翻译任务的一种技术，目的是在给定纹理示例的前提下，建立语义或其他能表示图像信息的数据类型与图像类型数据之间的映射，而图像数据所需的纹理信息则从示例中提取。论文中完成图像翻译需要三个必需的步骤：域对齐，特征匹配以及图像合成。域对齐部分使用 U-net 结构完成数据特征的编码与上采样，特征匹配部分选择使用 patch-match 技术在特征空间中找到最相似的特征，图像合成部分则选择使用 SPADE 支持的 GAN 进行对抗生成，最终使用分层策略，在庞大数据集的支持下，可以以高质量完成更高分辨率下的图像合成任务。

关键词：图像翻译；U-net；Patch-match；GRU；SPADE

1 引言

图像到图像的翻译任务学习了图像域的映射关系并且已经取得了广泛应用。基于示例的图像翻译任务可以提供给用户更灵活的控制，但是如何生成高质量的图像又成为了新的难题。早期的研究利用对抗生成网络学习直接映射关系，但是他们没有充分利用示例中的信息。随后，一系列方法提出，通过调制特征的标准化操作，在翻译过程中参考示例。然而在调制被应用后，只有全局风格被成功转移而纹理细节在最终输出丢失了。

此前，CoCosNet 的第一代版本成功利用域对齐网络建立了不同域之间的密集对应关系，并可以通过这种对应关系将示例中的纹理细节注入到输出图像中。由于低分辨率的任务无法利用图像中的纹理细节，且高分辨率任务内存消耗过大，所以无法完成高质量图像生成任务。

本文提出了一种跨域对应学习，可以以照片质量完成高分辨的图像翻译任务，并且网络可以利用示例中精细的纹理细节。为达成这个目的，文章利用了 patch-match 这种高效搜索相似特征的技术，他利用邻居之间的高相似度以及迭代计算的方法可以近似达到全局搜索的效果，并可以有效减少内存消耗。但是直接使用 patch-match 会带来很多问题：首先，当搜索的区域被随机初始化后，很难建立高分辨率下的全局对应关系；其次，由于初期不正确的对应关系，回传的梯度会影响不正确位置的特征生成，导致特征编码网络很难训练；最后，patch-match 的传播阶段很难考虑更大范围的上下文关系，且需要很多轮迭代才能收敛。

为解决上述限制，文章提出了如下技术来建立全分辨率下的对应关系。1) 使用分层策略，从低分辨率层，在简单难度下建立初步搜索对应并作为后续高分辨率的初始化。2) 收循环调制的启发，使用 GRU 模块在 patch-match 的迭代过程中对搜索位置进行优化，并且可以建立梯度流，帮助梯度回传。3) 基于分层策略的 GRU 辅助的 patch-match 模块是全部可微的，这对于以无监督的方式学习跨域对应很有意义。

总结一下论文的主要工作：1) 对图像翻译任务来说，提出了一种全分辨率的跨域对应网络来辅助捕获示例中的纹理细节。2) 提出了基于分层策略的 GRU 辅助的 patch-match 模块，可以在低内存消耗的工作环境下有效的进行特征匹配计算。3) 展示了全分辨率的跨域翻译结果，展示了论文提出的方法的确有能力在高分辨率图像中学习精细的纹理细节。

2 相关工作

2.1 PatchMatch

在计算机视觉领域，相关性匹配是一个基本问题。Patch-match 已经很大程度上完成了对高计算消耗的挑战，而 patch-match 的关键性见解基于两个原则：1) 好的特征匹配可以通过随机采样的方法获取;2) 图像是连续的，所以好的匹配可以传播到它的邻居中去。然后传统的 patch-match 算法只能应用于图像之间的匹配，而无法应用于深度卷积网络中。最近,[1]提出可以使整个匹配过程可微并且可以使特征编码与对应进行端到端的学习。然而这种方法在高分辨率对应中仍然具有很高的计算量，作为对比，文章提出了分层策略，并提出了新型的 GRU 辅助的精调结构来帮助找到更大的上下文，并在此情况下可以更快的收敛。

2.2 图像到图像的翻译任务

典型的图像翻译任务通过利用条件对抗生成网络并且通过配对的数据进行有明确监督的学习，或者使用未配对的数据利用循环一致性进行学习。最近，基于示例的图像翻译任务由于其有良好的可交互性以及生成质量，吸引了研究者们的注意力，但很多工作只是捕捉到了示例中的全局性结构而忽略的示例中的局部细节。本文旨在计算全分辨率上的密集对应关系从而可以获取一个小规模上的匹配关系，并且由于高分辨率的支持，使得本文提出的方法可以捕捉示例中的纹理细节，并因此得到高质量的输出图像。

3 本文方法

3.1 本文方法概述

此部分对本文将要复现的工作进行概述，主要框架如图 1所示：

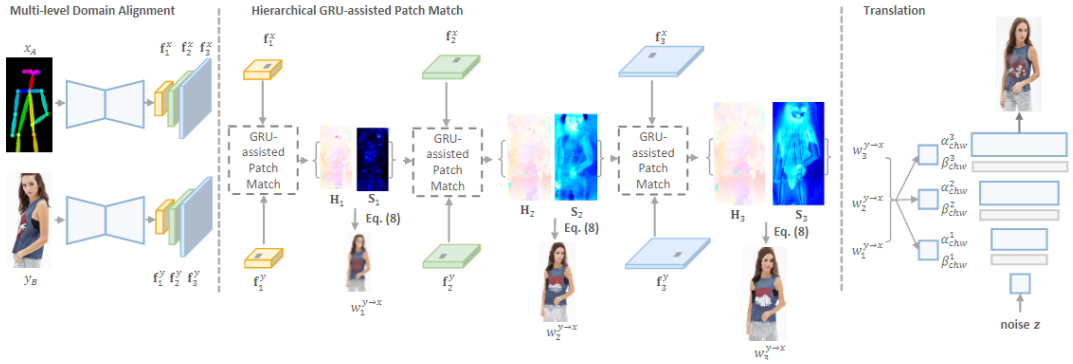

图 1: 网络结构图

文章提出的网络结构可以按功能分成三个部分，分别是：域对齐网络，特征匹配网络，图像合成网络。

3.2 域对齐网络

域对齐网络使用 U-net 网络结构，在下采样的过程中保留中间值，并在上采样的过程中将中间值累加至上采样卷积层的输入端，由此建立层与层之间的连接，不仅可以有效缓解梯度爆炸或梯度下降的问题，而且可以在上采样层中注入靠近输入端的结构特征，来保证上采样获取高分辨率特征的准确性。域对齐网络中为了保证语义信息不被冲散，并没有使用我们熟知的 BatchNormal 对每层的输入进行标准化，而是使用能保留更多语义信息的 SPADE 模块配合 SpectralNormal 进行标准化，并且效果显著。域对齐网络使用了两个 U-net 分别进行示例的特征提取以及语义的特征提取。本次复现中，语义特征提取网络输入端为 RBG 三通道的姿态图与姿态图中包括连接线与链接点在内的 17 种语义展开成的 17 通道的语义图拼接在一起，形成的 20 通道的语义图；图像特征提取网络的输入端为 RGB 三通道的人物图像与其对应的 20 通道的语义图拼接在一起，形成的 23 通道的图像输入。虽然输入端的通道数增加，并微量增加了网络规模，看似解空间变大了，会导致网络训练变得困难；但其实由于拼接的语义部分为网络输入提供了更好的初始化，在有监督的训练中会使网络收敛的更快，在卷积时提取到有效特征的能力也会变强。网络的输出端则会使用两个 U-net 中四个上采样卷积层的输出，分别输出 4 个分辨率的图像特征以及语义特征。

3.3 特征匹配网络

特征匹配网络则使用基于分层策略的 GRU 辅助的 patch-match 模块。分层策略指，在低分辨率下找到的最匹配的相似特征在高一级的分辨率中也可能是匹配的相似特征；基于这种策略，逐层构建搜索的索引，会比直接从最高分辨率使用 patch-match 进行全局特征匹配更简单。并且逐层匹配并输出根据特征相似度匹配获取的图像可以逐层注入梯度信息，使网络更快收敛。同时，原 patch-match 算法取单个最优的特征对应的像素进行生成图像对应位置像素的填充这个过程中，生成图像的每个像素都直接来源于示例图像，如果给生成图像添加损失，则无法产生梯度影响特征提取网络，所以文章在比较特征相似度时保存前 16 个最相似的特征对应的位置索引，并在输出图像时基于 16 个特征相似度分别计算权重，最后使用这个权重从示例图中采样像素并按权重分配，由此生成图像中的每个像素不仅与示例图有关，还跟 16 对特征之间的相似度有关，所以可以产生影响特征提取网络的梯度信息。在这个过程中，没有任何梯度信息会影响到索引部分，而对应的 16 个特征的索引则由 GRU 网络直接给出，那么 GRU 网络的梯度从何处获取？这个问题留在后面讨论。patch-match 经典算法中，搜索相似的特征的策略是：首先为每个像素位置随机分配一个位置索引，然后按照从上往下，从左往右的顺序在邻居对应的最佳索引周围，以及自身当前最优索引进行比较，取最优的匹配进行存储。这个策略在低分辨率情况下表现良好，当分辨率变大时，如果获得了一个并不是很良好的初始化则由于搜索范围的局限性，传播需要的迭代轮次将会线性增加。GRU 模块则是为解决这个问题而提出的策略，在每次搜索完毕后，将获得的索引送入 GRU 模块，预测输出一个更好的匹配索引，从而降低迭代轮次，增加收敛速度。

3.4 图像合成网络

图像合成网络的灵感来源于 SPADE 提供的语义保留的功能。如果在图像生成网络中，逐层加入 SPADE 模块用于提取示例图像中的概率分布性质，如：均值与方差，并将示例图像中的均值与方差应用于图像生成网络的标准化中，则图像生成网络的每一步结果都会尽可能地向示例图像的概率统计分

布靠近，从而提取到一些示例图像中的有用的性质，在本文中我们需要的信息是示例图像中的语义信息，以及相似但不完全准确的纹理信息。

3.5 损失函数定义

3.5.1 域对齐损失

为了正确得到域对齐网络的特征提取能力，提出训练策略：使用一张真实图片与图片对应的语义，分别输入网络，得到的输出特征应当相同。

这里用特征映射之间的 L1 损失来定义域对齐损失。

$$L_{align} = \|M_A(x_A;\theta_{M_A}) - M_{As}(x_{As};\theta_{M_{As}})\| \tag{1}$$

3.5.2 相关性损失

通过使用配对的数据进行训练，我们认为，经过特征匹配后，根据特征相似度生成的新的图像应当跟与输入语义配对的图像一致。

$$L_{corr} = \sum_l \|w_l^{\bar{y}_B - x_A} - x_B \downarrow\| \tag{2}$$

3.5.3 映射损失

我们希望跨域输入可以从其潜在表示映射回其本身，这将会帮助在潜在空间中保存语义信息。

$$L_{map} = \|R(M_A(x_A;\theta_{M_A})) - x_B\| + \|R(M_B(y_B;\theta_{M_B})) - y_B\| \tag{3}$$

3.5.4 翻译损失

图像生成网络的最终输出应该的外观表现应该与示例相似，我们设置了两个损失函数分别针对两个目标进行约束。

其一是感知损失，用于约束两个图像之间的语义差距。

$$L_{scm} = \|\phi_m(\hat{x}_B) - \phi_m(x_B)\| \tag{4}$$

其中 ϕ_m 指高层次的 VGG 损失。

另一个是外观损失，用于约束两个图像之间的外观差距，这里使用 contextual loss(CX)。

$$L_{app} = \sum_m u_m[-log(CX(\phi_m(\hat{x}_B), \phi_m(x_B)))] \tag{5}$$

3.5.5 对抗损失

最后一个损失对抗损失，也就是我们熟悉的对抗生成中的常用损失。

$$L_{adv}^{\mathcal{D}} = -\mathbb{E}[h(\mathcal{D}(y_B))] - \mathbb{E}[h(-\mathcal{D}(\mathcal{G}(x_A, y_B)))] \tag{6}$$

$$L_{adv}^{\mathcal{G}} = -\mathbb{E}[\mathcal{D}(\mathcal{G}(x_A, y_B))] \tag{7}$$

3.5.6 全部损失

将上述定义的全部损失整合在一起：

$$L = \min_{\mathcal{M},\mathcal{N},\mathcal{G},\mathcal{R}} \max_{\mathcal{D}} \lambda_1 L_{align} + \lambda_2 L_{corr} + \lambda_3 L_{map} + \lambda_4 (L_{scm} + L_{app}) + \lambda_5 (L_{adv}^{D} + L_{adv}^{G}) \tag{8}$$

4 复现细节

4.1 与已有开源代码对比

全部项目代码量较大，此部分会从数据预处理、域对齐网络搭建、特征匹配网络搭建、图像生成网络搭建、模型整合、训练器搭建六个部分做详细介绍。

4.1.1 数据预处理

原文使用的数据集为非公开数据集，在申请到数据集的使用权限后，发现姿态语义是以 txt 格式的文件给出，所以想要借鉴源码中对 txt 格式文件的处理方法来将他们处理为 RGB 格式，并按照原先的存储格式保存在指定文件夹中。在没有深入解读源码前，认为网络的输入是简单的 RGB 三通道图像和 RGB 三通道语义，所以预处理也按照这个方向来做。论文源码同时给出了能提供有监督学习的配对数据的 txt 文件类型，所以可以利用这个 txt 文件寻找配对训练的数据样本。

那么数据预处理的流程是首先使用 2将 txt 格式语义处理成 RGB 格式:

```
def get_label_tensor(self, path):
    candidate = np.loadtxt(path.format('candidate'))
    subset = np.loadtxt(path.format('subset'))
    stickwidth = 20
    limbSeq = [[2, 3], [2, 6], [3, 4], [4, 5], [6, 7], [7, 8], [2, 9], [9, 10], \
            [10, 11], [2, 12], [12, 13], [13, 14], [2, 1], [1, 15], [15, 17], \
            [1, 16], [16, 18], [3, 17], [6, 18]]
    colors = [[255, 0, 0], [255, 85, 0], [255, 170, 0], [255, 255, 0], [170, 255, 0], [85, 255, 0], [0, 255, 0], \
            [0, 255, 85], [0, 255, 170], [0, 255, 255], [0, 170, 255], [0, 85, 255], [0, 0, 255], [85, 0, 255], \
            [170, 0, 255], [255, 0, 255], [255, 0, 170], [255, 0, 85]]
    canvas = np.zeros((1024, 1024, 3), dtype=np.uint8)
    cycle_radius = 20
    for i in range(18):
        index = int(subset[i])
        if index == -1:
            continue
        x, y = candidate[index][0:2]
        cv2.circle(canvas, (int(x), int(y)), cycle_radius, colors[i], thickness=-1)
    joints = []
    for i in range(17):
        index = subset[np.array(limbSeq[i]) - 1]
        cur_canvas = canvas.copy()
        if -1 in index:
            joints.append(np.zeros_like(cur_canvas[:, :, 0]))
            continue
        Y = candidate[index.astype(int), 0]
        X = candidate[index.astype(int), 1]
        mX = np.mean(X)
        mY = np.mean(Y)
        length = ((X[0] - X[1]) ** 2 + (Y[0] - Y[1]) ** 2) ** 0.5
        angle = math.degrees(math.atan2(X[0] - X[1], Y[0] - Y[1]))
        polygon = cv2.ellipse2Poly((int(mY), int(mX)), (int(length / 2), stickwidth), int(angle), 0, 360, 1)
        cv2.fillConvexPoly(cur_canvas, polygon, colors[i])
        canvas = cv2.addWeighted(canvas, 0.4, cur_canvas, 0.6, 0)
        joint = np.zeros_like(cur_canvas[:, :, 0])
        cv2.fillConvexPoly(joint, polygon, 255)
        joint = cv2.addWeighted(joint, 0.4, joint, 0.6, 0)
        joints.append(joint)
    pose = Image.fromarray(cv2.cvtColor(canvas, cv2.COLOR_BGR2RGB)).resize((self.opt.load_size, self.opt.load_size), resample=Image.NEAREST)
    params = get_params(self.opt, pose.size)
    transform_label = get_transform(self.opt, params, method=Image.NEAREST, normalize=False)
    transform_img = get_transform(self.opt, params, method=Image.BILINEAR, normalize=False)
    tensors_dist = 0
    e = 1
    for i in range(len(joints)):
        im_dist = cv2.distanceTransform(255-joints[i], cv2.DIST_L1, 3)
        im_dist = np.clip((im_dist/3), 0, 255).astype(np.uint8)
        tensor_dist = transform_img(Image.fromarray(im_dist))
        tensors_dist = tensor_dist if e == 1 else torch.cat([tensors_dist, tensor_dist])
        e += 1
    tensor_pose = transform_label(pose)
    label_tensor = torch.cat((tensor_pose, tensors_dist), dim=0)
    return label_tensor, params
```

图 2: 语义样本处理

原文使用的代码如上图所示，其逻辑是在 getitem() 函数中, 当我们随机选取到了一组文件路径时，才会根据路径对指定的 txt 进行处理，并且会以 tensor 的数据类型返回我们需要的数据。所以我只用到了其前半部分，将其处理为 RGB 图像并按路径存储为止。

接下来要构建属于自己的数据集，构建之前，我们先根据原文提供的训练集列表来构建出自己想要的方便读取图像的 txt 文件，文件的每一行都保存了示例图像的存储路径以及语义图像的存储路径，当我们按使用字典存储这个 txt 文件中的数据时，可以很方便的取一对我们需要使用的数据。

当有了这个 txt 文件后，就可以构建自己的 getitem() 函数，在我们读取到数据后，避免不了的要对其尺寸进行裁剪，且为了进行一定程度上的数据增强，还要进行随机裁剪。这里我发现全部图像数据的尺寸是 750x1101, 且语义数据的尺寸是 1024x1024，为避免不必要的计算量，且能适配自己的机器运行，我决定把它裁至 128x128 的分辨率大小。首先 750x1101 的短边等比例放缩至 128，然后计算等比例下其长边的长度 h，然后将 1024x1024 放缩至 hxh，这样两幅图片就在长边上统一比例了。接下来对 hxh 的图像进行中心裁剪，裁剪的窗口大小为 128xh，裁剪的左下角坐标为 (h/2 - 64,0)，由此我们得到了尺寸完全相同的两张图像——128xh。接下来我们为了将图像的尺寸约束到 128x128，继续对这两张图进行相同的中心裁剪，尺寸已经知道，我们需要确定一个裁剪的坐标由于时随机裁剪，所以需要确定一个范围，经过计算，左下角坐标的变化范围为 (0,0) (0,h-128), 所以我们在这个范围内随机采样一个数字作为中心裁剪的坐标，即可得到我们想要的指定尺寸的输入。

处理完毕后的数据如图 3所示：

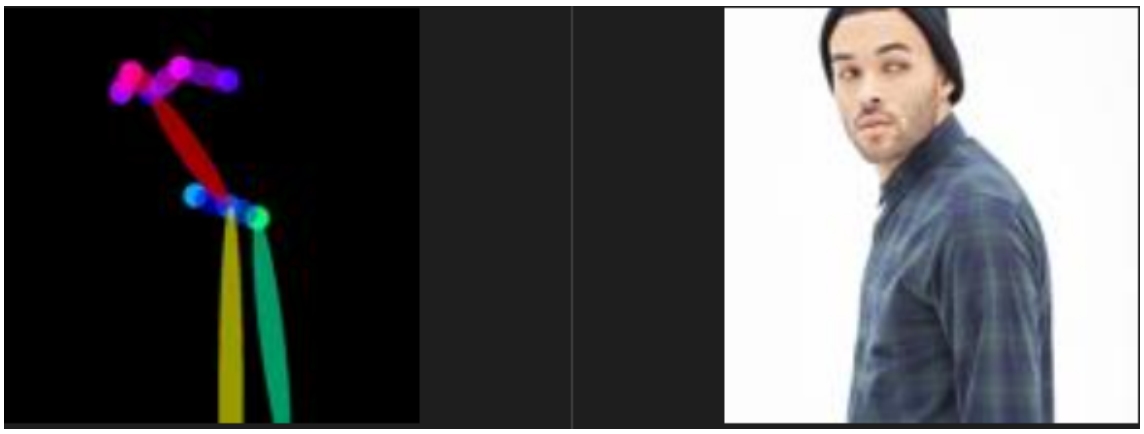

图 3: 输入语义以及图像

4.1.2 域对齐网络搭建

域对齐网络需要两个 U-net 来对两个输入分别进行编码，所以我们首先需要实现的模块是 U-net。根据论文中给出的网络架构，开始尝试使用卷积层以及残差网络模块对进行搭建，首先定义四个下采样的卷积模块，然后定义四个上采样的卷积模块，最后定义一个上采样操作，用于对每次上采样的输出进行下一次上采样。

整个网络中需要使用到残差模块，以及 SPADE 辅助的残差模块。残差模块用于下采样卷积网络，目的是在输入与输出之间建立链接，能够有效的传导梯度，加快收敛速度；SPADE 辅助的残差模块则用于上采样网络，SPADE 辅助的原因是为了将图像的语义信息注入到提取的特征映射中，最终我们会获得 4 层分辨率的特征映射，作为下一级的输入。这里借鉴了源代码中残差模块、SPADE 模块、以及 SPADE 残差模块的实现：

```python
class SPADEResnetBlock(nn.Module):
    def __init__(self, fin, fout, opt, use_se=False, dilation=1):
        super().__init__()
        # Attributes
        self.learned_shortcut = (fin != fout)
        fmiddle = min(fin, fout)
        self.opt = opt
        self.pad_type = 'nozero'
        self.use_se = use_se
        # create conv layers
        if self.pad_type != 'zero':
            self.pad = nn.ReflectionPad2d(dilation)
            self.conv_0 = nn.Conv2d(fin, fmiddle, kernel_size=3, padding=0, dilation=dilation)
            self.conv_1 = nn.Conv2d(fmiddle, fout, kernel_size=3, padding=0, dilation=dilation)
        else:
            self.conv_0 = nn.Conv2d(fin, fmiddle, kernel_size=3, padding=dilation, dilation=dilation)
            self.conv_1 = nn.Conv2d(fmiddle, fout, kernel_size=3, padding=dilation, dilation=dilation)
        if self.learned_shortcut:
            self.conv_s = nn.Conv2d(fin, fout, kernel_size=1, bias=False)
        # apply spectral norm if specified
        if 'spectral' in opt.norm_G:
            self.conv_0 = spectral_norm(self.conv_0)
            self.conv_1 = spectral_norm(self.conv_1)
            if self.learned_shortcut:
                self.conv_s = spectral_norm(self.conv_s)
        # define normalization layers
        spade_config_str = opt.norm_G.replace('spectral', '')
        if 'spade_ic' in opt:
            ic = opt.spade_ic #input channels of sematicmap
        else:
            ic = 4*3+opt.label_nc
        self.norm_0 = SPADE(spade_config_str, fin, ic, PONO=opt.PONO)
        self.norm_1 = SPADE(spade_config_str, fmiddle, ic, PONO=opt.PONO)
        if self.learned_shortcut:
            self.norm_s = SPADE(spade_config_str, fin, ic, PONO=opt.PONO)

    def forward(self, x, seg1):
        x_s = self.shortcut(x, seg1)
        if self.pad_type != 'zero':
            dx = self.conv_0(self.pad(self.actvn(self.norm_0(x, seg1))))
            dx = self.conv_1(self.pad(self.actvn(self.norm_1(dx, seg1))))
        else:
            dx = self.conv_0(self.actvn(self.norm_0(x, seg1)))
            dx = self.conv_1(self.actvn(self.norm_1(dx, seg1)))
        out = x_s + dx
        return out

    def shortcut(self, x, seg1):
        if self.learned_shortcut:
            x_s = self.conv_s(self.norm_s(x, seg1))
        else:
            x_s = x
        return x_s

    def actvn(self, x):
        return F.leaky_relu(x, 2e-1)
```

图 4: SPADE 辅助的残差模块

源代码中实现了多种方式的标准化操作用于消融实验，由于我们的目标是复现它的工作，所以在源码的基础上删除了这些不需要的标准化操作，只保留默认的标准化操作，由此，当我们将图像和语义图像分别输入 U-net 后，会返回多元数据，用于保存四个不同层级的特征映射输出。其中 U-net 的代码如下图所示：

```python
class U_Net(BaseNetwork):

    def __init__(self,input_channels = 3):
        super().__init__()
        kw = 4
        pw = int((kw -1)//2)
        nf = 64
        self.input_channels = input_channels

        norm_layer = get_nospade_norm_layer('spectralinstance') #return a function --first SN,and then IN
        self.layer1 = norm_layer(nn.Conv2d(self.input_channels,nf,3,stride=1,padding=pw)) #3 - 128
        self.layer2 = nn.Sequential(
            norm_layer(nn.Conv2d(nf *1,nf*2,3,stride=1,padding=1)),    # 128 - 64
            ResidualBlock(nf*2,nf*2)
        )
        self.layer3 = nn.Sequential(
            norm_layer(nn.Conv2d(nf*2,nf*4,kw,stride=2,padding=pw)),     #64- 32
            ResidualBlock(nf*4,nf*4)
        )
        self.layer4 = nn.Sequential(
            norm_layer(nn.Conv2d(nf*4,nf*4,kw,stride=2,padding=pw)),     #32 - 16
            ResidualBlock(nf*4,nf*4)
        )
        self.layer5 = nn.Sequential(
            norm_layer(nn.Conv2d(nf*4,nf*4,kw,stride=2,padding=pw)),     #16 - 8
            ResidualBlock(nf*4,nf*4)
        )
        self.head_0 = SPADEResnetBlock(nf*4,nf*4,ic=20)
        self.G_middle_0 = SPADEResnetBlock(nf*4,nf*4,ic = 20)
        self.G_middle_1 = SPADEResnetBlock(nf*4,nf*2,ic = 20)
        self.G_middle_2 = SPADEResnetBlock(nf*2,nf,ic = 20)
        self.up = nn.Upsample(scale_factor=2,mode='bilinear',align_corners=True)

    def forward(self,input,seg):
        #128 - 128   c 3-64
        x1 = self.layer1(input)
        #128 - 128   c 64-128
        x2 = self.layer2(self.actvn(x1))
        #128 - 64    c 128-256
        x3 = self.layer3(self.actvn(x2))
        #64 - 32     c 256-256
        x4 = self.layer4(self.actvn(x3))
        #32 - 16     c 256-256
        x5 = self.layer5(self.actvn(x4))
        #16 - 16 bottomleck c 256-256
        x6 = self.head_0(x5,seg)
        #16 - 32 c 256-256
        x7 = self.G_middle_0(self.up(x6) + x4 ,  seg)
        #32 - 64
        x8 = self.G_middle_1(self.up(x7) + x3,   seg)
        #64 - 128
        x9 = self.G_middle_2(self.up(x8) + x2,   seg)
        return [x6,x7,x8,x9]
        pass
    def actvn(self, x):
        return F.leaky_relu(x, 2e-1)
```

图 5: U-net

4.1.3 特征匹配网络搭建

特征匹配网络以 patch-match 为基础构建而成，这里以单个层级的特征匹配网络为例，进行详细介绍。

前文中提到，patch-match 本身并不具有梯度传导的能力，所以这里采用一种利用域对齐网络中提取到的特征附加权重的方式来创造可回传的梯度。首先我们会拿到示例图像和语义的两份特征映射，以及示例图像在当前分辨率层级下对应的副本，接下来我们会根据特征的相似度来选取示例图像对应位置的像素以及权重，并将这些像素的加权平均和作为语义对应位置的像素值。

现在确定搜索策略，我们需要维护一张表，表的宽高等同于特征映射的宽高，表中任意位置内容的功能是该位置的语义特征映射与图像特征映射中的哪些位置的特征最相似。由于我们需要计算权重，所以我们至少需要两个位置的特征来计算这个权重分配比例，在代码中，设置了保存前十六个最匹配的特征的位置来计算权重，所以我们维护的这个表应当是 16 通道的。

随机初始化一张 16 通道的索引表，基于 patch-match 的策略，对于特征映射中的每个特征，我们应当取当前特征在索引表中对应的索引以及这个索引的 16 个邻居进行比较，以及当前特征两个邻居在索引表中对应的索引以及 16 个邻居进行比较，共需比较 48 个位置，从中选取 16 个最近的索引进

行存储。这对应于 patch-match 的匹配阶段。原本 patch-match 的传播阶段会在当前最优位置周围指定半径内随机选若干的位置进行匹配，以此来随机的扩大匹配过程的感受野，但是由于对 16 个索引一一进行传播代码会更复杂，并且在分辨率过高的情况下，这种方法未必有效，所以采取 GRU 网络模块对维护的索引表进行重新分配。GRU 网络是一种 RNN 网络，他借助于前面时序的信息对当前时许的输出进行推断，这里使用 GRU 目的是将索引表中的索引预测到更合适的位置，本质也是在扩大感受野。

关于文中提到的 GRU 环节，在这里值得详细讨论一下。如上文所述，GRU 是为了将索引表中保存的索引“预测”到更合适匹配的位置，并且如原文所述：GRU 这个 RNN 模块应当是可学习的。但如果我们尝试去分解损失函数计算公式，我们无法找到任意一项可以与 GRU 对应的输出匹配。也就是说，这个 GRU 模块实际上是不可学习的，那么它的作用就并没有文章提到的那么大，它会将原本已经搜索好的索引随机扩散到其他 16 个区域，相当于每一轮都在重新初始化，并且在实验过程中发现，如果没有正确对这个网络模块进行初始化，则被“预测”后的索引将集中于某个区域或边缘，导致收敛速度极慢。

由于复现任务量大，时间短，设备性能要求高，没有多余的时间用作更多 GRU 模块的训练，故复现时选择使用另一种高内存占用的方法来代替 patch-match，代价是生成图像的分辨率将会很低，对于纹理细节的捕捉可能没有那么明显。这里选择替换使用的方法很粗暴，不再基于 16 个最优匹配去计算权重，而是直接基于全图去计算所有位置的权重，这样做会极大的消耗算力，并且会产生很多不必要的计算：比如当两个特征其实相差很大时，它所对应的权重其实非常小，最终提供到的梯度也是微乎其微，所以其实没有必要去计算这些梯度，尤其是最后在收敛时；但是在前期收敛难度大时，这种策略会将收敛速度变得非常高。

本部分代码量较大，这里只展示源代码主体部分的前两层实现：

```python
def multi_scale_patch_match(self, f1, f2, ref, hierarchical_scale, pre=None, real_img=None):
    if hierarchical_scale == 0:
        y_cycle = None
        scale = 64
        batch_size, channel, feature_height, feature_width = f1.size()
        ref = F.avg_pool2d(ref, 8, stride=8)
        ref = ref.view(batch_size, 3, scale * scale)
        f1 = f1.view(batch_size, channel, scale * scale)
        f2 = f2.view(batch_size, channel, scale * scale)
        matmul_result = torch.matmul(f1.permute(0, 2, 1), f2)/self.opt.temperature
        mat = F.softmax(matmul_result, dim=-1)
        y = torch.matmul(mat, ref.permute(0, 2, 1))
        if self.opt.phase is 'train' and self.opt.weight_warp_cycle > 0:
            mat_cycle = F.softmax(matmul_result.transpose(1, 2), dim=-1)
            y_cycle = torch.matmul(mat_cycle, y)
            y_cycle = y_cycle.permute(0, 2, 1).view(batch_size, 3, scale, scale)
        y = y.permute(0, 2, 1).view(batch_size, 3, scale, scale)
        return mat, y, y_cycle
    if hierarchical_scale == 1:
        scale = 128
        with torch.no_grad():
            batch_size, channel, feature_height, feature_width = f1.size()
            topk_num = 1
            search_window = 4
            centering = 1
            dilation = 2
            total_candidate_num = topk_num * (search_window ** 2)
            topk_inds = torch.topk(pre, topk_num, dim=-1)[-1]
            inds = topk_inds.permute(0, 2, 1).view(batch_size, topk_num, (scale//2), (scale//2)).float()
            offset_x, offset_y = inds_to_offset(inds)
            dx = torch.arange(search_window, dtype=topk_inds.dtype, device=topk_inds.device).unsqueeze_(dim=1).expand(-1, search_window).contiguous().view(-1) - centering
            dy = torch.arange(search_window, dtype=topk_inds.dtype, device=topk_inds.device).unsqueeze_(dim=0).expand(search_window, -1).contiguous().view(-1) - centering
            dx = dx.view(1, search_window ** 2, 1, 1) * dilation
            dy = dy.view(1, search_window ** 2, 1, 1) * dilation
            offset_x_up = F.interpolate((2 * offset_x + dx), scale_factor=2)
            offset_y_up = F.interpolate((2 * offset_y + dy), scale_factor=2)
        ref = F.avg_pool2d(ref, 4, stride=4)
        ref = ref.view(batch_size, 3, scale * scale)
        mat, y = self.patch_match(f1, f2, ref, offset_x_up, offset_y_up)
        y = y.view(batch_size, 3, scale, scale)
        return mat, y
```

图 6: patch match

4.1.4 图像生成网络搭建

图像生成网络依托于对抗生成模型 GAN，并利用 SPADE 模块不断提取特征匹配网络的输出语义，也就是概率统计特性，注入到 GAN 的生成过程中。模块并不复杂，关键是为了不在生成过程中冲刷掉语义信息，我们需要使用自己实现的 PN(位置归一化) 模块来代替 BN(批量归一化)。这里先给出源码中 PN 的实现代码：

```python
def PositionalNorm2d(x, epsilon=1e-8):
    # x: B*C*W*H normalize in C dim
    mean = x.mean(dim=1, keepdim=True)
    std = x.var(dim=1, keepdim=True).add(epsilon).sqrt()
    output = (x - mean) / std
    return output
```

图 7: positional Normalize

为了保证收敛相率和生成质量，源码并没有以噪声图作为输出，而是将四张特征匹配网络产生的输出插值并拼接其目标语义输入作为生成器的输入。也就是说，生成器的作用是对上个阶段产生的模糊输出结果进行精调，使之具备完整的纹理细节。最终，生成器将生成一张虚假图片送入判别器。生成网络代码实现如下：

```python
class SPADEGenerator(BaseNetwork):
    @staticmethod
    def modify_commandline_options(parser, is_train):
        parser.set_defaults(norm_G='spectralspadesyncbatch3x3')
        return parser

    def __init__(self, opt):
        super().__init__()
        self.opt = opt
        nf = opt.ngf
        self.sw, self.sh = self.compute_latent_vector_size(opt)
        ic = 4*3+opt.label_nc
        self.fc = nn.Conv2d(ic, 8 * nf, 3, padding=1)
        self.head_0 = SPADEResnetBlock(8 * nf, 8 * nf, opt)
        self.G_middle_0 = SPADEResnetBlock(8 * nf, 8 * nf, opt)
        self.G_middle_1 = SPADEResnetBlock(8 * nf, 8 * nf, opt)
        self.up_0 = SPADEResnetBlock(8 * nf, 8 * nf, opt)
        self.up_1 = SPADEResnetBlock(8 * nf, 4 * nf, opt)
        self.up_2 = SPADEResnetBlock(4 * nf, 2 * nf, opt)
        self.up_3 = SPADEResnetBlock(2 * nf, 1 * nf, opt)
        final_nc = nf
        self.conv_img = nn.Conv2d(final_nc, 3, 3, padding=1)
        self.up = nn.Upsample(scale_factor=2)

    def compute_latent_vector_size(self, opt):
        num_up_layers = 5
        sw = opt.crop_size // (2**num_up_layers)
        sh = round(sw / opt.aspect_ratio)
        return sw, sh

    def forward(self, input, warp_out=None):
        seg = torch.cat((F.interpolate(warp_out[0], size=(512, 512)), F.interpolate(warp_out[1], size=(512, 512)), F.interpolate(warp_out[2], size=(512, 512)), warp_out[3], input), dim=1)
        x = F.interpolate(seg, size=(self.sh, self.sw))
        x = self.fc(x)
        x = self.head_0(x, seg)
        x = self.up(x)
        x = self.G_middle_0(x, seg)
        x = self.G_middle_1(x, seg)
        x = self.up(x)
        x = self.up_0(x, seg)
        x = self.up(x)
        x = self.up_1(x, seg)
        x = self.up(x)
        x = self.up_2(x, seg)
        x = self.up(x)
        x = self.up_3(x, seg)
        x = self.conv_img(F.leaky_relu(x, 2e-1))
        x = torch.tanh(x)
        return x
```

图 8: generator

4.1.5 模型整合

在模型整合部分会创建域对齐网络，生成器网络，判别器网络等。并且会实现网络存储，网络加载，优化器创建，损失函数计算等在训练中会用到的模块。

这里为了保证能满足我自己想要的预训练验证需求，在里面添加了一些与训练中会用到的功能。

4.1.6 训练器搭建

初始化训练器时，首先要初始化一个模型，并且按照训练入口提供的参数选择判断是创建一套全新的模型，还是使用之前保存的模型副本继续训练。为了保证生成器与判别器的能力不会失衡，将判别器和生成器的一次梯度下降实现分开。并且在训练器中实现随训练轮次进行更新学习率的函数。这里的实现参考了源码中的部分片段，具体实现请参照提交代码。

4.2 实验环境搭建

本次复现源码使用主要语言为 python 与 pytorch，代码所需运行库请视情况自行安装。

本文使用数据集为非公开数据集,如有需求,请访问该网站:https://drive.google.com/file/d/1bByKH1ciLXYHd4pe_i/view?usp=sharing 自行申请获取。

获取压缩文件后，将其解压并以

```
deepfashionHD
│
└─── img
│ │
│ └─── MEN
│ │ │ ...
│ │
│ └─── WOMEN
│ │ ...
│
└─── pose
│ │
│ └─── MEN
│ │ │ ...
│ │
│ └─── WOMEN
│ │ ...
```

此种文件存储格式以及命名方式将其存放在 data 文件夹内，并使用 python 编译器，依次运行代码内提供的 remodify.py 和 preprocess.py 文件对数据进行预处理。

代码没有给出预训练时使用的相关代码，但是给出了预训练得到的模型，读者可从 train.py 文件，修改 resume_epoch,continue_train 和 pre_train 参数来满足自己对于功能上的需求。

4.3 创新点

1) 省时但占用存储空间更高的数据预处理方法。

2)GRU 模块的替换。

3) 将整个过程拆分为预训练和训练过程，便于参数拟合。

5 实验结果分析

由于时间和设备限制，未能将整个训练过程完整跑完。原文使用 8 块 32G 的英伟达显卡需要训练 200 小时，而我每天只能训练 1 个 epoch，目标 epoch 总数为 100，所以最终未能产生比较好的实验结果。

实验过程中还是遇到了很多问题的，现在开始逐一说明。

初次进行训练时，认为 3 通道的输入包含足够的信息，可以完成预训练任务，但结果并不尽如人意。训练一个 epoch 后的结果如下：

图 9: warp0_3channels

然后观察源码才发现，需要更多通道来概括语义，否则网络无法对语义进行准确识别，加重网络负担。于是修改数据加载部分的代码，将语义分割成 17 个通道，并返回给数据加载器，训练 1 个 epoch 后的结果如下：

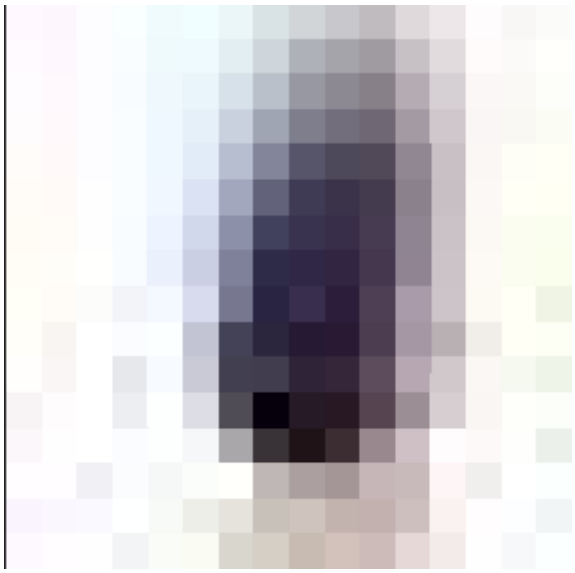

图 10: warp0_20channels

发现还是过于模糊，结果不尽如人意。于是思考可能会有问题发生的点，发现自己在最开始写数据预处理的代码时，并没有仔细分析源码给出的配队数据的 txt 文件，也就是说对配对数据的识别并不准确，在训练时，当数据对被识别为未配对数据时，将会丢失一个损失项，可能会对训练结果产生

重大影响。于是重新改写代码，使其能准确识别配对的图像，并增加这项损失的权重，训练 1 个 epoch 后得到如下结果：

图 11: warp0_pair

可以看到第一层级的特征匹配结果已经比较好了，说明此处修改是有效的。但是令人头痛的是，二、三、四层的特征匹配结果确实完全与预期不匹配，其效果如下：

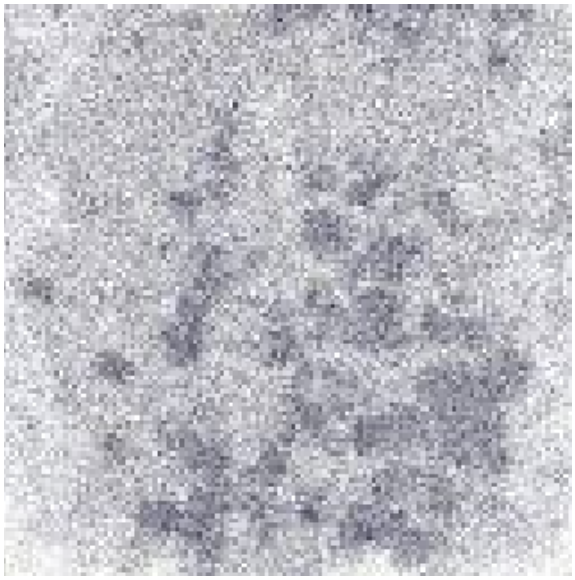

图 12: warp3_pair

而二三四层与一层的区别仅在于使用了 patch-match 模块，于是开始调查其梯度流向，发现在高层与 patch-match 的接口处是有梯度回传的，但是大多数都集中在较边缘区域，同时 GRU 模块上则完全没有梯度产生。由于此时已经没有时间可以浪费了，故直接抛弃掉所有 patch-match 模块，使用暴力的算法对整个特征映射赋予权重，预训练 1 个 epoch 后的结果如下：

图 13: warp3_without_patchmatch

此时已经可以从第四层得到相对比较好的特征匹配结果，于是在 5 个 epoch 的预训练后，将预训练好的模型接入正式训练的流程，耗时 8 天，训练了 8 个 epoch 的结果如下图所示：

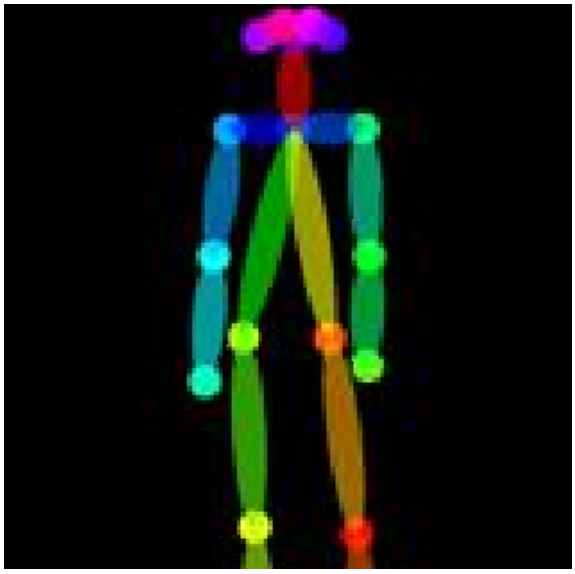

图 14: test_seg

图 15: test_exampler

图 16: test_out

可以看到在加入预训练模块后，网络在仅 8 次 epoch 后就取得了如此令人震惊的效果。

6 总结与展望

本次实验虽然没有完美的完成，但是无论是在编写代码的过程中，还是在训练或预训练的调参过程中都收获了很多经验，并且最终加深了对深度学习这门知识的理解程度。本次实验将成为我未来研究道路上的一个转折点，即使是在有源码参考的情况下，能完成这项可以称为巨量的工作任务已经使我积累了很多经验，以后的学习道路上再遇见如此规模的项目时，将不会再显得手足无措。谢谢各位老师以及助教给我这次学习的机会，非常感激!!

参考文献

[1] DUGGAL S, WANG S, MA W C, et al. DeepPruner: Learning Efficient Stereo Matching via Differentiable PatchMatch[J]. international conference on computer vision, 2019.