SVTR: Scene Text Recognition with a Single Visual Model

甘其阳

摘要

主要场景文本识别模型一般包含两个构建模块，一个为可视化模型文本的特征提取和序列模型转录。这种混合架构精确，是复杂和低效的。在这个在此基础上，提出了一种场景的单一视觉模型在 patch-wise image toenization 框架内的文本识别，该框架完全免除了序列建模。该方法称为 SVTR 首先将图像文本分解成小的文本命名为字符组件的补丁。之后, 分层阶段由组件级混合、合并和/或组合。设计了全局和局部混合块感知角色间和角色内模式，导致多粒度的字符组件感知。因此，字符是可以识别的通过一个简单的线性预测。实验结果在中英文场景文本识别任务上验证了 SVTR 算法的有效性。SVTR-L(大型) 在英语中实现了极具竞争力的交流精度，并在中文中以很大的优势优于现有的 meth(大型) 得更快。此外，SVTR-T (Tiny) 是一种有效的药物更小的模型，这显示出吸引力推理速度。

关键词：SVTR；计算机视觉；模式识别

1 引言

场景文本识别的目的是将自然图像中的文本转录为数字字符序列，具有较高的文字层次语义学对于场景理解至关重要。这项任务很有挑战性，因为文字变形、字体、遮挡、杂乱的背景等等。在过去的几年里，许多努力已经提高了识别精度。现代的文本识别器除了精度，还考虑推理等因素由于实际需要考虑速度。

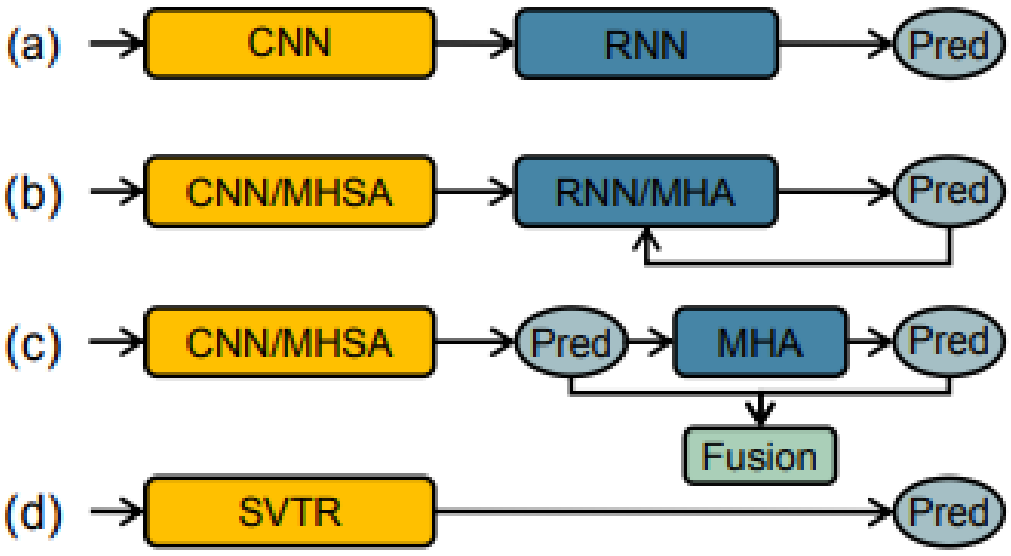

图 1: (a) 基于 CNN-RNN 的模型。(b) 编码器-解码器 mod。MHSA 和 MHA 分别表示多头自注意和多头注意。(c) 视觉-语言模型。(d) 我们 SVTR，它用单一的视觉模型识别场景文本语言高效、准确、跨语言通晓

从方法论上讲，场景文本识别可以看作从图像到字符序列的跨模态映射。通常，识别器由两个构建块组成，一个特征提取的可视化模型和序列模型文本转录。例如，基于 CNN-RNN 的模型首次使用 CNN 用于特征提取。然后将该特征重塑为序列，利用 BiLSTM 和 CTC 损失进行建模，得到特征值预

测 (图 1(a))。他们的特点是高效和仍然是一些商业识别器的选择。如何，重塑是敏感的文本干扰，如变形、闭塞等限制了它们的效果。后来，基于编码器-解码器的自回归方法开始流行，该方法将识别转换为 it 运算解码过程 (图 1(b))。因此，在上下文信息的情况下，获得了精确的证明考虑。然而，推理速度较慢，因为逐字符转录。管道被进一步扩展到基于视觉语言的框架，语言知识在哪里合并 (图 1(c))，并进行并行预测。然而，管道往往需要较大的容量模型或复杂的识别范式无法保证识别精度，限制了识别效率。

近年来，开发简化体系结构的努力得到了重视，以加快速度。例如，我们提供了复杂的训练范式，却提供了简单的推理模型。基于 CRNN-RNN 的求解方法在。它利用了注意机制和图形神经网络聚合与同一字符对应的序列特征。在推理中，为了平衡精度和速度，注意模分支被丢弃。PREN2D 进一步简化了识别通过聚合和解码 1D 子字符特征同时进行。提出了 VisionLAN。它介绍了字符型闭塞学习的赋予具有语言能力的视觉模型。在推断的时候，视觉模型的应用仅仅是为了加速。在视图基于单一可视化模型的体系结构的简单性，一些识别器是通过使用现成的来提出的 CNN 或 ViT 作为特征提取器。尽管效率很高，但准确性却很差与最先进的方法相比，竞争力较低。我们认为基于单一可视化模型的方案是可行的只有在字符特征具有鉴别性的情况下才有效提取。具体来说，该模型可以成功捕获无论是字符内的局部模式还是字符间的长期依赖。前者编码类似笔画的特征描述一个角色的细粒度特征区分字符的关键来源。而后者则记录了描述语言的类似知识人物从互补的角度出发。然而，这两个以前的特性提取器没有很好地对属性进行建模。例如，CNN 主干擅长建模局部相关，而不是全局依赖。与此同时, 而基于电流互感器的通用骨干则可以不给局部字符模式特权。

基于上述问题，本工作旨在通过强化 vi sual 模式，提高识别能力。为此，我们提出了一个基于视觉的模型 SVTR 准确，快速和跨语言多功能场景文本识别。受到 vision trans -最近成功的启发，SVTR 首先将图像文本分解为小的二维补丁字符组件，因为每个组件可能包含一个部分只是一个角色。在此基础上，采用自注意下的 patch-wise 图像标记技术捕获识别线索在字符组件中。具体来说，是一个自定义的文本体系结构就是为此目的而开发的。它分为三个阶段混合、合并和/或合并操作使脊柱高度逐渐降低。本地和全球混合每个阶段都设计并反复使用区块，与合并或合并经营一起进行的，收购局部组件级的亲缘关系，表示字符的笔画样特征，以及它们之间的长期依赖关系不同的字符。因此，主干提取不同距离和多尺度的成分特征，形成多粒度的人物特征感知。因此，通过简单的线性预测即可实现识别。在整个过程中只使用了一个可视化模型。我们构造四个具有不同能力的体系结构变体。对英语和英语进行了广泛的实验中文场景文本识别任务。结果表明，SVTR(大) 在英语和英语中取得了极具竞争力的成绩在中文方面，它的表现大大超过了最先进的模型，同时跑得更快。而 SVTR-T (Tiny) 则是这也是一个有效的，更小的模型，但有吸引力推理速度。这项工作的主要贡献可以是总结如下

• 我们第一次证明，一个单一的视觉模型可以达到具有竞争力甚至更高的精度作为场景文本识别中的高级视觉语言模型。由于它的高效性和跨语言通用性，在实际应用中具有广阔的应用前景。

• 我们提出了 SVTR，一种用于识别的文本定制模型。它介绍了局部和全局混合块分别提取笔画状特征和字间依赖数据，再加上多尺度的背骨数据，形成多粒度的特征描述。

• 对公共基准的实证研究表明 SVTR 的优势。SVTR-L 在识别英语和 Chi 中文场景文本方面实现了最先进的性能。SVTR-T 有效且高效，参数为 6.03M，耗时 4.5ms 在一个 NVIDIA 1080Ti GPU 中，

平均每个图像文本

2 相关工作

2.1 Overall Architecture

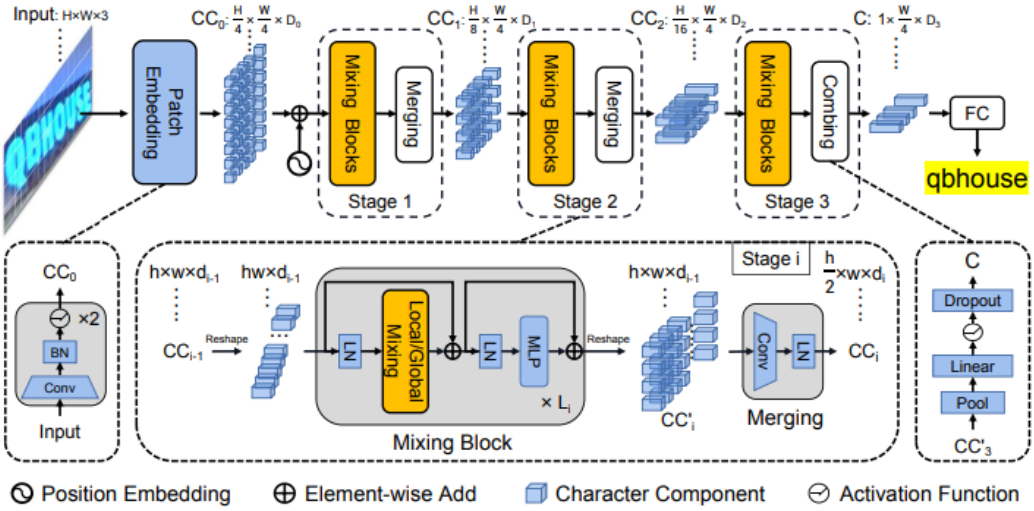

图 2: 建议的 SVTR 的总体架构。它是一个三级高度递减的网络。在每个阶段，都有一系列的进行混合块，然后进行合并或组合操作。最后，采用线性预测的方法进行识别

所提议的 SVTR 概述如图 2 所示。它是一种三级高度递减的网络，专门用于文本识别。对于大小为 H×W ×3 的图像文本，它首先转化为$\frac{H}{4}\times\frac{W}{4}\times D_0$ 维的补丁渐进式重叠贴片嵌入。这些补丁是称为字符组件，每个组件与图像中文本字符的一个片断相关联。然后，每个阶段有三个阶段由一系列混合块组成，然后进行合并或组合作业，均按不同规模进行特征提取。设计了局部和全局混合块用于类笔画局部图案提取和成分间捕获的依赖。具有主干、分量特征和不同距离、在多重的依赖性对刻度进行表征，生成引用的表示法大小为 C 的 $1\times\frac{W}{4}\times D_3$，感知多粒性格特性。最后，进行了并行线性预测进行重复数据删除得到字符序列。

2.2 Progressive Overlapping Patch Embedding

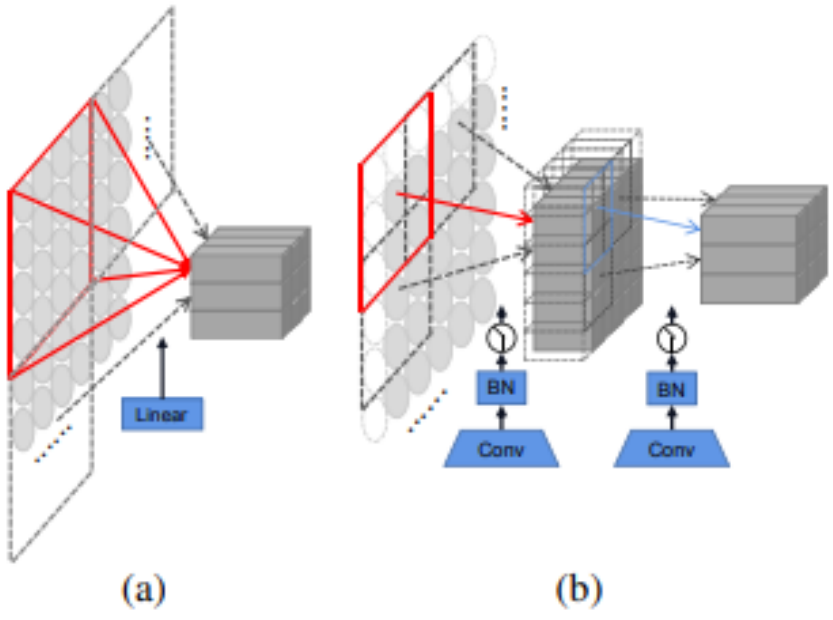

图 3: (a) ViT 中的线性投影 [Dosovitskiy 等人，2021]。(b) 渐进重叠补丁嵌入。

对于图像文本，首先要完成的任务是获取特征补丁，表示 $X \in R^{H \times W \times 3}$ 的字符分量到 $CC_0 \in R^{\frac{H}{4} \times \frac{W}{4} \times D_0}$。常见的一步法有两种用于此目的的投影，即 4×4 不相交线性投影 (参见图 3(a)) 和 stride 4 的 7 × 7 卷积。或者，我们通过使用两个连续的 3 × 3 卷积，步幅 2 和批处理如图 3(b) 所示。该计划，尽管增加了一点计算成本，增加了特性维数递增，有利于特征融合。

2.3 Mixing Block

由于两个字符可能略有不同，文本识别严重依赖于角色组件级别的功能。然而，现有的研究大多采用特征序列来分析表示图像文本。每个特征都对应于一个薄层的图像区域，该区域通常是有噪声的，特别是对于不规则的文本。它不是描述角色的最佳选择。视觉变压器的最新进展引入了 2D 功能表示法，但如何利用这种表示法文本的语境识别仍然是一个值得研究的问题。更具体地说，对于嵌入式组件，我们认为文本识别需要两种特征。的首先是局部成分模式，例如类似笔画的特征。它编码一个字符不同部分之间的形态特征和相关性。第二个是字符之间的依赖关系，例如不同字符之间或文本和非文本组件之间的相关性。因此，设计了两个混合块，通过使用不同接收场的自我注意来感知关联关系。

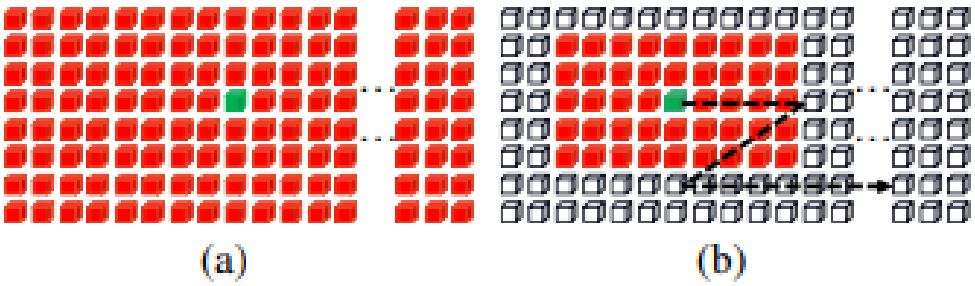

图 4: 说明 (a) 全局混合和 (b) 局部混合

Global Mixing. 如图 4(a) 所示，全局混合评估 (global mixed eval) 计算了所有字符分量之间的依赖关系。自文字和非文字是图像中的两个主要元素，例如通用混合可以在不同性质的组分之间建立长期依赖关系。同时，它也能削弱非文本的影响组件，同时增强文本组件的重要性。数学上，对于字符分量 CC_{i-1} 在前一阶段，它首先被重塑为一个特征序列。进料进入混合块时，层数为应用并跟随一个多头自注意进行依赖建模。在下面，一个层 norm 和一个 MLP 依次应用于特征融合。连同快捷连接，形成全局搅拌块。**Local Mixing.** 如图 4(b) 所示，局部混合评估 (local mixture eval) 用于计算预定义值中各个组件之间的相关性窗口。其目的是对特征形态进行编码特征化并建立组件之间的关联在字符内，它模拟用于字符识别的类笔画特征。与全球混合不同，局部混合为每个组件考虑一个邻域。与卷积类似，混合以滑动窗口的方式进行。窗口大小根据经验设置为 7 × 11。与全局混合相比，它实现了全局混合捕获局部模式的自注意机制。如前所述，这两个混合块旨在提取互补的不同特征。在 SVTR，这些块在每个阶段中被反复应用多次用于综合特征提取。两者的排列各种块稍后将被烧蚀。

2.4 Merging

保持空间不变需要大量的计算跨阶段的解析，这也会导致冗余表示。因此，我们在每个阶段 (除了最后一个阶段) 的混合块之后设计了一个合并操作一个)。根据上一个混合块输出的特征，我们首先将其重塑为大小为 h × w × d_{i-1} 的嵌入，分别表示当前的高度、宽度和通道。然后，我们使用 3 × 3 卷积，步幅 2 在高度维度，步幅 1 在宽度维度，然后是一个 LayerNorm，生成一个嵌入大小为 $\frac{h}{2} \times w \times d_i$。

合并操作将高度减半，同时保持宽度不变。它不仅降低了计算成本，而且还要构建一个文本自定义的层次结构。通常情况下, 大多数图像文本显示水平或接近水平。压缩高度维度可以为每个字符建立多尺度表示，而不影响补丁布局在宽度维度。因此，它不会类中编码相邻字符的机会增加跨阶段的相同组件。我们还增加了渠道维护补偿信息损失。

2.5 Combining and Prediction

在最后一个阶段，合并操作被合并操作取代。它首先将高度维度池化为 1，接下来是全连接层，非线性激活和辍学。通过这样做，角色组件得到了进一步的发展压缩到一个特征序列，其中每个元素由一个长度为 D_3 的特征表示。与合并相比运算时，合并运算可以避免将卷积应用于一个很小的嵌入高度为 2 的尺寸。结合这些特征，我们实现了识别通过一个简单的并行线性预测。具体地说，是线性的采用 N 节点分类器。它会生成一个转录本 $\frac{W}{4}$ 尺寸序列，理想情况下，相同的组件字符被转录为重复的字符、组件对非文本都转录为空白符号。序列自动压缩为最终结果。在实现中，英文的 N 设置为 37，中文的 N 设置为 6625。

3 本文方法

3.1 数据集

使用街景文本 (SVT) 有 647 个测试从谷歌街景裁剪的图像进行模型训练。

3.2 实现细节

SVTR 使用整流模块，其中图像文本被调整为 32 × 64 的失真校正。我们使用权重衰减为 0.05 的 AdamW 优化器培训。对于英语模型，初始学习率是设定好的到 $\frac{5}{104}\times\frac{batchsize}{2048}$。余弦学习率调度器与 2 epoch 的线性预热在所有 21 个 epoch 中都使用。旋转、透视失真、运动模糊等数据处理和高斯噪声，在训练过程中随机执行。字母表包括所有不区分大小写的字母数字。最大预测长度设置为 25。长度超过了绝大多数英语单词。

3.3 消融实验

为了更好地理解 SVTR，我们对 IC13(常规文本) 和 IC15(不规则文本) 进行了控制实验。在不同的设置下。为了效率，所有的实验采用 SVTR-T 进行无校正模组和数据增强。

3.4 可视化分析

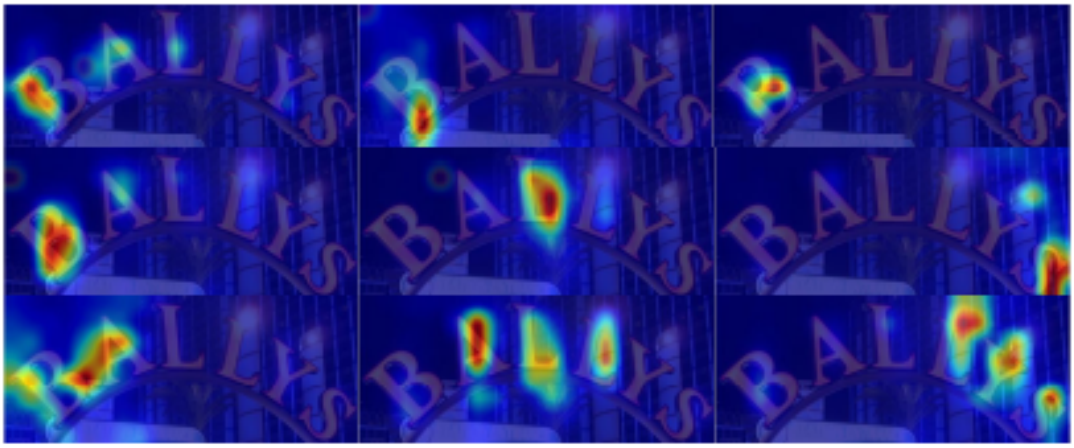

图 5: SVTR-T 注意图的可视化

我们已经可视化了 SVTR-T 在对不同字符组件进行编码时的注意图。每张地图都可以解释为在整个认知中扮演不同的角色。选取 9 张地图进行说明，如图 5 所示。第一行展示了三张凝视一小部分的地图字“B”，强调在左边，底部和中部，分别为。这表明不同的字符域对识别有贡献。第二个 Line 展示了三幅凝视不同人物的地图，即“B”，“L”和“S”。SVTR-T 还能够通过观察一个字符作为一个整体来了解它的特征。而第三行展品三个地图同时激活多个字符。它暗示了不同字符之间的依赖关系成功捕获。这三条线合在一起揭示了这一点子字符，字符级别和跨字符线索根据权利要求，所有被识别者捕获 SVTR 感知多粒度的字符组件特征。这再次解释了 SVTR 的有效性。通过研究结果和讨论，得出结论 SVTR-L 具有准确、快速、多功能等优点，是高精度应用领域极具竞争力的选择。而 SVTR-T 是一个有效且小得多的模型但速度相当快。它在资源有限的情况下很有吸引力

4 复现细节

4.1 与已有开源代码对比

代码主要参考开源源码和 GitHub 相关项目进行学习和模仿编写的，以下为网络模型的编写。

```
class ConvBNLayer(nn.Module):
    def __init__(self,
                 in_channels,
                 out_channels,
                 kernel_size=3,
                 stride=1,
                 padding=0,
                 bias_attr=False,
                 groups=1,
                 act=nn.GELU):
        super().__init__()

        self.conv = nn.Conv2d(
            in_channels=in_channels,
            out_channels=out_channels,
            kernel_size=kernel_size,
            stride=stride,
            padding=padding,
            groups=groups,
            bias=bias_attr
        )
        self.norm = nn.BatchNorm2d(out_channels)
        self.act = act()

    def forward(self, inputs):
        out = self.conv(inputs)
        out = self.norm(out)
        out = self.act(out)
        return out
```

```python
class DropPath(nn.Module):

    def __init__(self, drop_prob=None):
        super(DropPath, self).__init__()
        self.drop_prob = drop_prob

    def forward(self, x):
        return drop_path(x, self.drop_prob, self.training)


class Identity(nn.Module):
    def __init__(self):
        super(Identity, self).__init__()

    def forward(self, input):
        return input
```

```python
class Mlp(nn.Module):
    def __init__(self,
                 in_features,
                 hidden_features=None,
                 out_features=None,
                 act_layer=nn.GELU,
                 drop=0.):
        super().__init__()
        out_features = out_features or in_features
        hidden_features = hidden_features or in_features
        self.fc1 = nn.Linear(in_features, hidden_features)
        self.act = act_layer()
        self.fc2 = nn.Linear(hidden_features, out_features)
        self.drop = nn.Dropout(drop)

    def forward(self, x):
        x = self.fc1(x)
        x = self.act(x)
        x = self.drop(x)
        x = self.fc2(x)
        x = self.drop(x)
        return x
```

```python
class ConvMixer(nn.Module):
    def __init__(self,
                 dim,
                 num_heads=8,
                 HW=(8, 25),
                 local_k=(3, 3)):
        super().__init__()
        self.HW = HW
        self.dim = dim
        self.local_mixer = nn.Conv2d(
            dim,
            dim,
            local_k,
            1,
            (local_k[0] // 2, local_k[1] // 2),
            groups=num_heads
        )

    def forward(self, x):
        h = self.HW[0]
        w = self.HW[1]
        x = x.permute(0, 2, 1).reshape([-1, self.dim, h, w])
        x = self.local_mixer(x)
        x = torch.flatten(x, 2).permute(0, 2, 1)
        return x
```

```python
class Attention(nn.Module):
    def __init__(self,
                 dim,
                 num_heads=8,
                 mixer='Global',
                 HW=(8, 25),
                 local_k=[7,11],
                 qkv_bias=False,
                 qk_scale=None,
                 attn_drop=0.,
                 proj_drop=0.):
        super().__init__()
        self.num_heads = num_heads
        head_dim = dim // num_heads
        self.scale = qk_scale or head_dim**-0.5

        self.qkv = nn.Linear(dim, dim*3, bias=qkv_bias)
        self.attn_drop = nn.Dropout(attn_drop)
        self.proj = nn.Linear(dim, dim)
        self.proj_drop = nn.Dropout(proj_drop)
        self.HW = HW
        if HW is not None:
            H = HW[0]
            W = HW[1]
            self.N = H * W
            self.C = dim
        if mixer == 'Local' and HW is not None:
            hk = local_k[0]
            wk = local_k[1]
            mask = torch.ones([H * W, H + hk - 1, W + wk - 1], dtype=torch.float32).cuda()
            for h in range(0, H):
                for w in range(0, W):
                    mask[h * W + w, h:h + hk, w:w + wk] = 0.
            mask_torch = torch.flatten(mask[:, hk // 2:H + hk // 2, wk // 2:W + wk //
                                2], 1)
            mask_inf = torch.full([H * W, H * W], -np.inf, dtype=torch.float32).cuda()
```

```
            mask = torch.where(mask_torch < 1, mask_torch, mask_inf)
            self.mask = mask.unsqueeze(0)
            self.mask = self.mask.unsqueeze(0)
            # print(self.mask.size())

        self.mixer = mixer

    def forward(self, x):
        if self.HW is not None:
            N = self.N
            C = self.C
        else:
            _, N, C = x.size()
        qkv = self.qkv(x)
        qkv = qkv.reshape((-1, N, 3, self.num_heads, C // self.num_heads))
        qkv = qkv.permute(2, 0, 3, 1, 4)
        q, k, v = qkv[0] * self.scale, qkv[1], qkv[2]
        attn = (q.matmul(k.permute(0, 1, 3, 2)))
        if self.mixer == 'Local':
            attn += self.mask
        attn = nn.functional.softmax(attn, dim=-1)
        attn = self.attn_drop(attn)

        x = (attn.matmul(v)).permute(0, 2, 1, 3).reshape((-1, N, C))
        x = self.proj(x)
        x = self.proj_drop(x)
        return x
```

```python
class Block(nn.Module):
    def __init__(self,
                 dim,
                 num_heads,
                 mixer='Global',
                 local_mixer=[7, 11],
                 HW=[8, 25],
                 mlp_ratio=4.,
                 qkv_bias=False,
                 qk_scale=None,
                 drop=0,
                 attn_drop=0,
                 drop_path=0.,
                 act_layer=nn.GELU,
                 norm_layer='nn.LayerNorm',
                 epsilon=1e-6,
                 prenorm=True):
        super().__init__()
        if isinstance(norm_layer, str):
            self.norm1 = eval(norm_layer)(dim, eps=epsilon)
        else:
            self.norm1 = norm_layer(dim)
        if mixer == 'Global' or mixer == 'Local':
            self.mixer = Attention(
                dim,
                num_heads=num_heads,
                mixer=mixer,
                HW=HW,
                local_k=local_mixer,
                qkv_bias=qkv_bias,
                qk_scale=qk_scale,
                attn_drop=attn_drop,
                proj_drop=drop
            )
```

```
        elif mixer == 'Conv':
            self.mixer = ConvMixer(
                dim, num_heads=num_heads, HW=HW, local_k=local_mixer
            )
        else:
            raise TypeError('The mixer must be one of [Global, Local, Conv]')

        self.drop_path = DropPath(drop_path) if drop_path > 0. else Identity()
        if isinstance(norm_layer, str):
            self.norm2 = eval(norm_layer)(dim, eps=epsilon)
        else:
            self.norm2 = norm_layer(dim)
        mlp_hidden_dim = int(dim*mlp_ratio)
        self.mlp_ratio = mlp_ratio
        self.mlp = Mlp(
            in_features=dim,
            hidden_features=mlp_hidden_dim,
            act_layer=act_layer,
            drop=drop
        )
        self.prenorm = prenorm

    def forward(self, x):
        if self.prenorm:
            x = self.norm1(x + self.drop_path(self.mixer(x)))
            x = self.norm2(x + self.drop_path(self.mlp(x)))
        else:
            x = x + self.drop_path(self.mixer(self.norm1(x)))
            x = x + self.drop_path(self.mlp(self.norm2(x)))
        return x
```

```python
class PatchEmbed(nn.Module):

    def __init__(self,
                 img_size=[32, 100],
                 in_channels=3,
                 embed_dim=768,
                 sub_num=2):
        super().__init__()
        num_patches = (img_size[1] // (2 ** sub_num)) * \
                      (img_size[0] // (2 ** sub_num))
        self.img_size = img_size
        self.num_patches = num_patches
        self.embed_dim = embed_dim
        self.norm = None
        if sub_num == 2:
            self.proj = nn.Sequential(
                ConvBNLayer(
                    in_channels=in_channels,
                    out_channels=embed_dim // 2,
                    kernel_size=3,
                    stride=2,
                    padding=1,
                    act=nn.GELU,
                    bias_attr=False),
                ConvBNLayer(
                    in_channels=embed_dim // 2,
                    out_channels=embed_dim,
                    kernel_size=3,
                    stride=2,
                    padding=1,
                    act=nn.GELU,
                    bias_attr=False)
            )
```

```
        if sub_num == 3:
            self.proj = nn.Sequential(
                ConvBNLayer(
                    in_channels=in_channels,
                    out_channels=embed_dim // 4,
                    kernel_size=3,
                    stride=2,
                    padding=1,
                    act=nn.GELU,
                    bias_attr=False),
                ConvBNLayer(
                    in_channels=embed_dim // 4,
                    out_channels=embed_dim // 2,
                    kernel_size=3,
                    stride=2,
                    padding=1,
                    act=nn.GELU,
                    bias_attr=False),
                ConvBNLayer(
                    in_channels=embed_dim // 2,
                    out_channels=embed_dim,
                    kernel_size=3,
                    stride=2,
                    padding=1,
                    act=nn.GELU,
                    bias_attr=False))

    def forward(self, x):
        B, C, H, W = x.size()
        assert H == self.img_size[0] and W == self.img_size[1], \
            f"Input image size ({H}*{W}) doesn't match model ({self.img_size[0]}*{self.img_size[1]})."
        x = self.proj(x).flatten(2).permute(0, 2, 1)
        return x
```

```python
class SubSample(nn.Module):
    def __init__(self,
                 in_channels,
                 out_channels,
                 types='Pool',
                 stride=(2, 1),
                 sub_norm='nn.LayerNorm',
                 act=None):
        super().__init__()
        self.types = types
        if types == 'Pool':
            self.avgpool = nn.AvgPool2d(
                kernel_size=(3, 5), stride=stride, padding=(1, 2))
            self.maxpool = nn.MaxPool2d(
                kernel_size=(3, 5), stride=stride, padding=(1, 2))
            self.proj = nn.Linear(in_channels, out_channels)
        else:
            self.conv = nn.Conv2d(
                in_channels,
                out_channels,
                kernel_size=3,
                stride=stride,
                padding=1
            )
        self.norm = eval(sub_norm)(out_channels)
        if act is not None:
            self.act = act()
        else:
            self.act = None

    def forward(self, x):
        if self.types == 'Pool':
            x1 = self.avgpool(x)
            x2 = self.maxpool(x)
            x = (x1 + x2) * 0.5
            out = self.proj(x.flatten(2).permute(0, 2, 1))
        else:
            x = self.conv(x)
            out = x.flatten(2).permute(0, 2, 1)
        out = self.norm(out)
        if self.act is not None:
            out = self.act(out)

        return out
```

```python
def __init__(
        self,
        img_size=[32, 640],
        in_channels=3,
        embed_dim=[64, 128, 256],
        depth=[3, 6, 3],
        num_heads=[2, 4, 8],
        mixer=['Local'] * 6 + ['Global'] *
        6,  # Local atten, Global atten, Conv
        local_mixer=[[7, 11], [7, 11], [7, 11]],
        patch_merging='Conv',  # Conv, Pool, None
        mlp_ratio=4,
        qkv_bias=True,
        qk_scale=None,
        drop_rate=0.,
        last_drop=0.1,
        attn_drop_rate=0.,
        drop_path_rate=0.1,
        norm_layer='nn.LayerNorm',
        sub_norm='nn.LayerNorm',
        epsilon=1e-6,
        out_channels=192,
        out_char_num=100,
        block_unit='Block',
        act='nn.GELU',
        last_stage=True,
        sub_num=2,
        prenorm=True,
        use_lenhead=False,
        **kwargs):
    super().__init__()
    self.img_size = img_size
    self.embed_dim = embed_dim
    self.out_channels = out_channels
    self.prenorm = prenorm
    patch_merging = None if patch_merging != 'Conv' and patch_merging != 'Pool' else patch_merging

    self.patch_embed = PatchEmbed(
        img_size=img_size,
        in_channels=in_channels,
        embed_dim=embed_dim[0],
        sub_num=sub_num)
    num_patches = self.patch_embed.num_patches
    self.HW = [img_size[0] // (2**sub_num), img_size[1] // (2**sub_num)]
    # self.pos_embed = self.create_parameter(
    #     shape=[1, num_patches, embed_dim[0]], default_initializer=zeros_)
    # self.add_parameter("pos_embed", self.pos_embed)
    self.pos_embed = nn.Parameter(torch.zeros([1, num_patches, embed_dim[0]], dtype=torch.float32), requires_grad=True)
    self.pos_drop = nn.Dropout(p=drop_rate)
    Block_unit = eval(block_unit)

    dpr = np.linspace(0, drop_path_rate, sum(depth))

    self.blocks1 = nn.ModuleList([
        Block_unit(
            dim=embed_dim[0],
            num_heads=num_heads[0],
            mixer=mixer[0:depth[0]][i],
            HW=self.HW,
            local_mixer=local_mixer[0],
            mlp_ratio=mlp_ratio,
            qkv_bias=qkv_bias,
            qk_scale=qk_scale,
            drop=drop_rate,
            act_layer=eval(act),
            attn_drop=attn_drop_rate,
            drop_path=dpr[0:depth[0]][i],
            norm_layer=norm_layer,
            epsilon=epsilon,
            prenorm=prenorm) for i in range(depth[0])
    ])
```

```
if patch_merging is not None:
    self.sub_sample1 = SubSample(
        embed_dim[0],
        embed_dim[1],
        sub_norm=sub_norm,
        stride=[2, 1],
        types=patch_merging)
    HW = [self.HW[0] // 2, self.HW[1]]
else:
    HW = self.HW
self.patch_merging = patch_merging
self.blocks2 = nn.ModuleList([
    Block_unit(
        dim=embed_dim[1],
        num_heads=num_heads[1],
        mixer=mixer[depth[0]:depth[0] + depth[1]][i],
        HW=HW,
        local_mixer=local_mixer[1],
        mlp_ratio=mlp_ratio,
        qkv_bias=qkv_bias,
        qk_scale=qk_scale,
        drop=drop_rate,
        act_layer=eval(act),
        attn_drop=attn_drop_rate,
        drop_path=dpr[depth[0]:depth[0] + depth[1]][i],
        norm_layer=norm_layer,
        epsilon=epsilon,
        prenorm=prenorm) for i in range(depth[1])
])
```

```
if patch_merging is not None:
    self.sub_sample2 = SubSample(
        embed_dim[1],
        embed_dim[2],
        sub_norm=sub_norm,
        stride=[2, 1],
        types=patch_merging)
    HW = [self.HW[0] // 4, self.HW[1]]
else:
    HW = self.HW
self.blocks3 = nn.ModuleList([
    Block_unit(
        dim=embed_dim[2],
        num_heads=num_heads[2],
        mixer=mixer[depth[0] + depth[1]:][i],
        HW=HW,
        local_mixer=local_mixer[2],
        mlp_ratio=mlp_ratio,
        qkv_bias=qkv_bias,
        qk_scale=qk_scale,
        drop=drop_rate,
        act_layer=eval(act),
        attn_drop=attn_drop_rate,
        drop_path=dpr[depth[0] + depth[1]:][i],
        norm_layer=norm_layer,
        epsilon=epsilon,
        prenorm=prenorm) for i in range(depth[2])
])
self.last_stage = last_stage
```

```
        if last_stage:
            self.avg_pool = nn.AdaptiveAvgPool2d((1, out_char_num))
            self.last_conv = nn.Conv2d(
                in_channels=embed_dim[2],
                out_channels=self.out_channels,
                kernel_size=1,
                stride=1,
                padding=0,
                bias=False
            )
            self.hardswish = nn.Hardswish()
            self.dropout = nn.Dropout(p=last_drop)
        if not prenorm:
            self.norm = eval(norm_layer)(embed_dim[-1], epsilon=epsilon)
        self.use_lenhead = use_lenhead
        if use_lenhead:
            self.len_conv = nn.Linear(embed_dim[2], self.out_channels)
            self.hardswish_len = nn.Hardswish()
            self.dropout_len = nn.Dropout(p=last_drop)

        truncated_normal_(self.pos_embed)
        self.apply(self._init_weights)
        print('---------model weight inits-----------')

    def _init_weights(self, m):
        if isinstance(m, nn.Linear):
            truncated_normal_(m.weight)
            if isinstance(m, nn.Linear) and m.bias is not None:
                nn.init.zeros_(m.bias)
        if isinstance(m, nn.LayerNorm):
            nn.init.zeros_(m.bias)
            nn.init.ones_(m.weight)
        if isinstance(m, nn.Conv2d):
            nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='relu')
```

```
def forward_features(self, x):
    x = self.patch_embed(x)
    x = x + self.pos_embed
    x = self.pos_drop(x)
    for blk in self.blocks1:
        x = blk(x)
    if self.patch_merging is not None:
        x = self.sub_sample1(
            x.permute(0, 2, 1).reshape(
                [-1, self.embed_dim[0], self.HW[0], self.HW[1]]))
    for blk in self.blocks2:
        x = blk(x)
    if self.patch_merging is not None:
        x = self.sub_sample2(
            x.permute(0, 2, 1).reshape(
                [-1, self.embed_dim[1], self.HW[0] // 2, self.HW[1]]))
    for blk in self.blocks3:
        x = blk(x)
    if not self.prenorm:
        x = self.norm(x)
    return x

def forward(self, x):
    x = self.forward_features(x)
    if self.use_lenhead:
        len_x = self.len_conv(x.mean(1))
        len_x = self.dropout_len(self.hardswish_len(len_x))
    if self.last_stage:
        if self.patch_merging is not None:
            h = self.HW[0] // 4
        else:
            h = self.HW[0]
        x = self.avg_pool(
            x.permute(0, 2, 1).reshape(
                [-1, self.embed_dim[2], h, self.HW[1]]))
```

```
            x = self.last_conv(x)
            x = self.hardswish(x)
            x = self.dropout(x)
        if self.use_lenhead:
            return x, len_x
        return x


class Im2Seq(nn.Module):
    def __init__(self, **kwargs):
        super().__init__()
        # self.out_channels = in_channels

    def forward(self, x):
        B, C, H, W = x.size()
        assert H == 1
        x = x.squeeze(2)
        x = x.permute(0, 2, 1)  # (NTC)(batch, width, channels)
        return x
```

```python
class CTCHead(nn.Module):
    def __init__(self,
                 in_channels=192,
                 out_channels=6624,
                 fc_decay=0.0004,
                 mid_channels=None,
                 return_feats=False,
                 **kwargs):
        super(CTCHead, self).__init__()
        if mid_channels is None:
            self.fc = nn.Linear(
                in_channels,
                out_channels)
        else:
            self.fc1 = nn.Linear(
                in_channels,
                mid_channels)
            self.fc2 = nn.Linear(
                mid_channels,
                out_channels)
        self.in_channels = in_channels
        self.out_channels = out_channels
        self.mid_channels = mid_channels
        self.return_feats = return_feats
        self.apply(self._init_weights)
        print('---------model weight inits----------')

    def _init_weights(self, m):
        if isinstance(m, nn.Linear):
            stdv = 1.0 / math.sqrt(self.in_channels * 1.0)
            nn.init.uniform_(m.weight, -stdv, stdv)
            nn.init.uniform_(m.bias, -stdv, stdv)
```

```
        def forward(self, x):
            if self.mid_channels is None:
                predicts = self.fc(x)
            else:
                x = self.fc1(x)
                predicts = self.fc2(x)
            #   batchsize * T * C ---->  T * batchsize * C
            predicts = predicts.permute(1, 0, 2)
            predicts = predicts.log_softmax(2).requires_grad_()

            if self.return_feats:
                result = (predicts, x)
            else:
                result = (predicts, None)
            return result


    class SVTRArch(nn.Module):
        def __init__(self):
            super(SVTRArch, self).__init__()
            self.backbone = SVTRNet()
            self.neck = Im2Seq()
            self.head = CTCHead()

        def forward(self, x):
            x = self.backbone(x)
            x = self.neck(x)
            x = self.head(x)
            return x
```

4.2 实验环境搭建

安装 Python、Anaconda3，下载 PaddleOCR 代码

4.3 界面分析与使用说明

使用命令行执行程序，得到想要的文本识别结果

命令行格式：!python 调用工具使用配置训练模型测试图片是否使用 GPU

例子:!python tools/infer$_{r}ec.py-cconfigs/rec/rec_{s}vtrnet_{c}h.yml-oGlobal.pretrained_{m}odel = ./svtr_{t}rain_{n}$
$./doc/imgs_{w}ords/en/word_{6}.PNGGlobal.use_{g}pu = False$

5 实验结果分析

图 6: 数字的文本识别

图 7: 汉字的文本识别

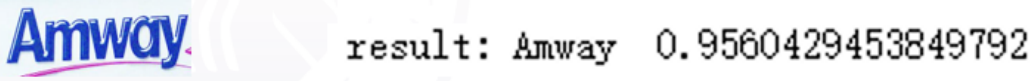

图 8: 英文的文本识别

6 总结与展望

整个文论复现参考了 paddle 的源码和 GitHub 上的项目进行参考、模仿编写，从一开始什么都不懂、什么都不会到看懂 transformer、Vit，到实现 svtr 网络模型框架，训练模型，将近三个月的时间，受益匪浅，明白了很多以前没有接触过的知识，但仍有很多不足，在复现过程中并没有任何创新点，只是实现论文了代码，在未来我将打好基础，学好机器学习、深度学习，并往机器视觉方向发展。

[1][2][3][4][5]

参考文献

[1] JADERBERG M, SIMONYAN K, VEDALDI A, et al. Reading Text in the Wild with Convolutional Neural Networks[Z]. 2016.

[2] BAEK J, KIM G, LEE J, et al. What Is Wrong With Scene Text Recognition Model Comparisons? Dataset and Model Analysis[J]. arXiv, 2019.

[3] WANG Y, XIE H, FANG S, et al. From Two to One: A New Scene Text Recognizer with Visual Language Modeling Network[J]., 2021.

[4] LIU Z, LIN Y, CAO Y, et al. Swin Transformer: Hierarchical Vision Transformer using Shifted Windows [J]., 2021.

[5] Atienza, Rowel. Vision Transformer for Fast and Efficient Scene Text Recognition[C]//. 2021.