IT Blog

MMOCR之多模态融合ABINET文字识别

admin 2个月前 (07-31) 18 次

MMCV系列之MMOCR

注：大家觉得博客好的话，别忘了点赞收藏呀，本人每周都会更新关于人工智能和大数据相关的内容
为原创，Python Java Scala SQL 代码，CV NLP 推荐系统等，Spark Flink Kafka Hbase Hive Flu
写的都是纯干货，各种顶会的论文解读，一起进步。
今天和大家分享一下MMOCR之多模态融合ABINET文字识别
论文地址：https://arxiv.org/pdf/2103.06495.pdf
代码地址：https://github.com/open-mmlab/mmocr
#博学谷IT学习技术支持#

文章目录

总结

前言

MMCV系列我会一直更新的，是CV很火很实用的一套框架，非常推荐做CV模型的小伙伴实用。

上一次是和大家分享MMOCR之DBNET文字检测。

https://blog.csdn.net/weixin_53280379/article/details/125995393?spm=1001.2014.3001.55(

今天是和大家继续分享MMOCR之ABINET文字识别。

下一次关键信息抽取。都是一个完整的系列。

先来看一下模型最终的输出效果。上次都框出来的基础上，这次是都能识别出里边文字的具体内容。

可以看到效果还是不错的。

一、ABINET文字识别模型的整体架构是什么?

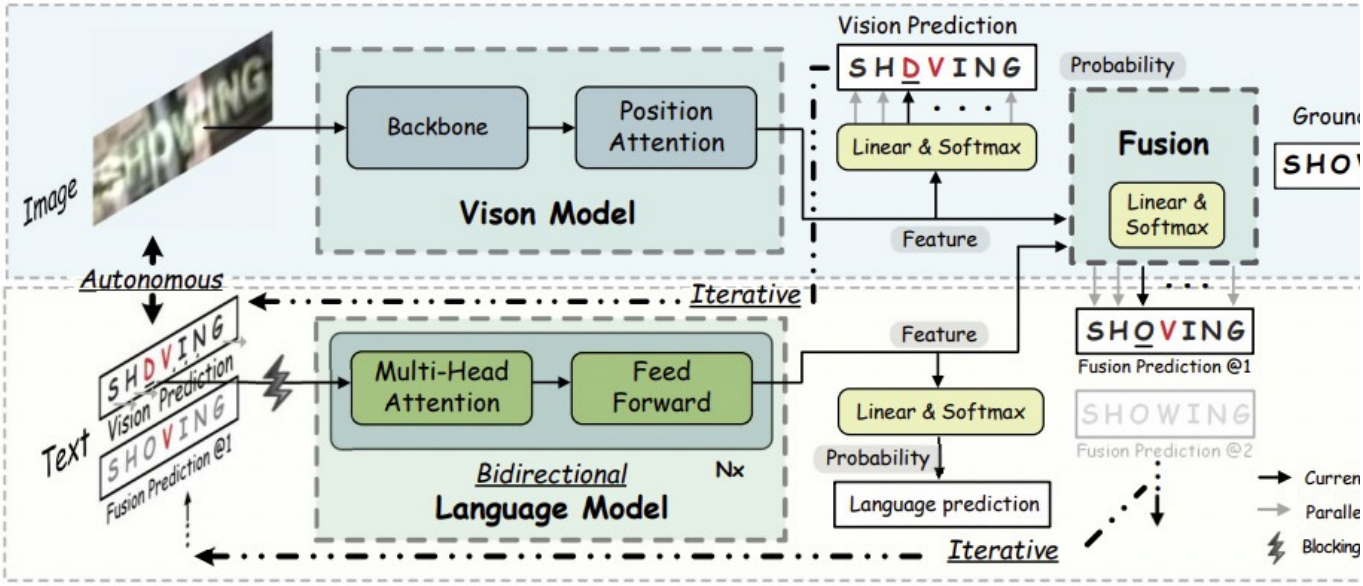

Figure 2. A schematic overview of ABINet.

代码如下（示例）：

```
num_chars = 38
max_seq_len = 26
```

```
label_convertor = dict(
    type='ABIConvertor',
    dict_type='DICT36',
    with_unknown=True,
    with_padding=False,
    lower=True)
model = dict(
    type='ABINet',
    backbone=dict(type='ResNetABI'),
    encoder=dict(
        type='ABIVisionModel',
        encoder=dict(
            type='TransformerEncoder',
            n_layers=3,
            n_head=8,
            d_model=512,
            d_inner=2048,
            dropout=0.1,
            max_len=256),
        decoder=dict(
            type='ABIVisionDecoder',
            in_channels=512,
            num_channels=64,
            attn_height=8,
            attn_width=32,
            attn_mode='nearest',
            use_result='feature',
            num_chars=38,
            max_seq_len=26,
            init_cfg=dict(type='Xavier', layer='Conv2d'))),
    decoder=dict(
        type='ABILanguageDecoder',
        d_model=512,
        n_head=8,
        d_inner=2048,
        n_layers=4,
        dropout=0.1,
        detach_tokens=True,
        use_self_attn=False,
        pad_idx=36,
        num_chars=38,
        max_seq_len=26,
        init_cfg=None),
    fuser=dict(
        type='ABIFuser',
        d_model=512,
        num_chars=38,
        init_cfg=None,
        max_seq_len=26),
```

```
        loss=dict(
            type='ABILoss',
            enc_weight=1.0,
            dec_weight=1.0,
            fusion_weight=1.0,
            num_classes=num_chars),
        label_convertor=dict(
            type='ABIConvertor',
            dict_type='DICT36',
            with_unknown=True,
            with_padding=False,
            lower=True),
        max_seq_len=26,
        iter_size=3)
```

可以看到这里几个超参的设置

num_chars = 38 表示一共有38分类分别是 DICT36 = tuple(‘0123456789abcdefghijklmnopqrz’)加上一个终止符和一个unknown。

max_seq_len = 26 表示每个单词最长不得超过26个字符。

由于ABINET是一个独立多模态模型，所以这里的encoder模型用到的是一个视觉模型ABIVisionMo

而decoder模型用到的是一个自然语言处理模型ABILanguageDecoder

最后将两种模型相融合得到ABIFuser，输出结果。

二、模型详解

1.模型的输入

模型的输入都是一张张这种经过文字检测模型输出的小图片。

标签为将每个文字转化为一一对应的38个数字，最长不超过26个，其中37标识终止符。

2.Encoder 视觉模型ABIVisionModel

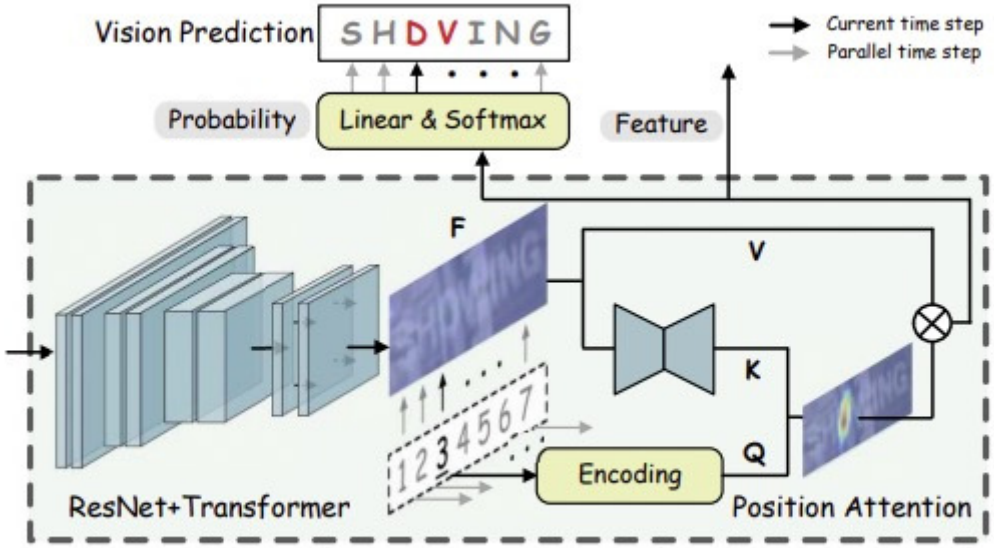

Figure 3. Architecture of vision model

第一步：Backbone这里先将输入的文字图片经过ResNet+Transformer提取特征都是非常非常常规得到的输出大小tensor(8,512,8,32)，其中8代表batch size，512代表特征图个数，8*32代表特征图

第二步：Position Attention 这里和传统的Self Attention不一样,是直接用的Attention。

1.Q代表26个位置编码，自己生成的，初始值类似于正余弦编码加一层线性转换。

2.K是通过Backbone的输出之后的特征图，加一个Unet网络，得到的K，这里没有直接用一般常见的换。

3.V也通过Backbone的输出之后的特征图，直接用的线性转换得到V。

ps：做这样的Position Attention是为了固定每个字母位置的信息，所以Q代表26个位置编码。

代码如下（示例）：

```
    def forward_train(self,
                      feat,
                      out_enc=None,
                      targets_dict=None,
                      img_metas=None):

        # Position Attention
        N, E, H, W = feat.size()
        # k, v这里直接是从特征图来的，而q不是，q是自己生成的，这个是最大的不同
        k, v = feat, feat  # (N, E, H, W)

        # Apply mini U-Net on k
        features = []
        for i in range(len(self.k_encoder)):
            k = self.k_encoder[i](k)
            features.append(k)
        for i in range(len(self.k_decoder) - 1):
            k = self.k_decoder[i](k)
            k = k + features[len(self.k_decoder) - 2 - i]
        k = self.k_decoder[-1](k)

        # q = positional encoding

        # 重点是这个q，这里q的初始值类似于正余弦编码加一层线性转换
        zeros = feat.new_zeros((N, self.max_seq_len, E))  # (N, T, E)
        q = self.pos_encoder(zeros)  # (N, T, E)
        q = self.project(q)  # (N, T, E)

        # Attention encoding
        attn_scores = torch.bmm(q, k.flatten(2, 3))  # (N, T, (H*W))
        attn_scores = attn_scores / (E**0.5)
        attn_scores = torch.softmax(attn_scores, dim=-1)
        v = v.permute(0, 2, 3, 1).view(N, -1, E)  # (N, (H*W), E)
        attn_vecs = torch.bmm(attn_scores, v)  # (N, T, E)

        logits = self.cls(attn_vecs)
        result = {
            'feature': attn_vecs,
            'logits': logits,
            'attn_scores': attn_scores.view(N, -1, H, W)
        }
        return result
```

最终得到logits的维度是tensor(8,26,38)，其中8还是代表batch size，26代表输出长度为26，每个
个38分类任务。

ps：这里的输出就表示视觉模块做完了，一般的ocr文字识别中，这里直接连多分类损失函数就可以
全没有问题，很多公司项目也这么落地的，但是效果没有加上文本模型，做多模态效果好。

3.Decoder 文本模型ABILanguageDecoder

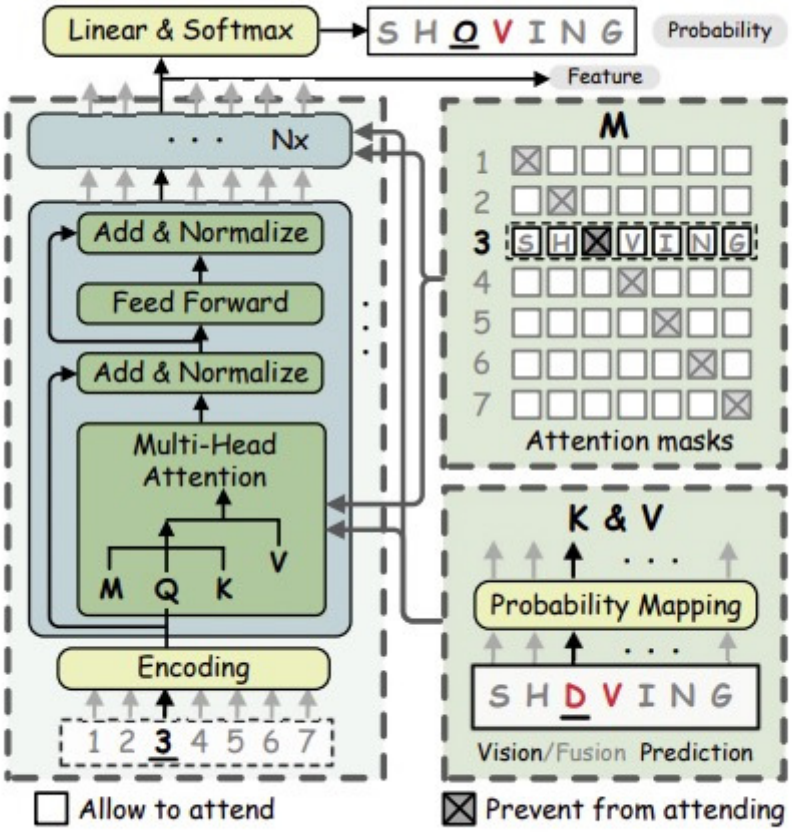

Figure 4. Architecture of language model (BCN).

文本模型的目的是为了做矫正，看看视觉模型的输出是否合理，每个字母逐一检查，迭代检查n遍（
可改），提高模型精度。

第一步：文本模型的输入就是视觉模型的输出结果，然后连一个softmax。维度还是tensor(8,26,38

第二步：升维操作，将38个结果升维成512，得到更多信息。

第三步：做location_mask，就是一个完形填空，根据上下文推测当前位置是什么，所以要用Mask在
tion的时候把自己给遮住。不能透题。Attention的q也是Position Attention做法一样。和BERT训练
。

ps：这里通过BERT完形填空对视觉模型的输入做更新，起到了一个再矫正的作用。

代码如下（示例）：

```
def forward_train(self, feat, logits, targets_dict, img_metas):

    lengths = self._get_length(logits)
    lengths.clamp_(2, self.max_seq_len)
    # 第一步：文本模型的输入就是视觉模型的输出结果，然后连一个softmax。维度还是tensor(8,26,
    tokens = torch.softmax(logits, dim=-1)
    if self.detach_tokens:
        tokens = tokens.detach()

    # 第二步：升维操作，将38个结果升维成512，得到更多信息。
    embed = self.proj(tokens)  # (N, T, E)
    embed = self.token_encoder(embed)  # (N, T, E)

    padding_mask = self._get_padding_mask(lengths, self.max_seq_len)
    zeros = embed.new_zeros(*embed.shape)
    query = self.pos_encoder(zeros)
    query = query.permute(1, 0, 2)  # (T, N, E)
    embed = embed.permute(1, 0, 2)
    # 第三步：做location_mask，就是一个完形填空，根据上下文推测当前位置是什么
    location_mask = self._get_location_mask(self.max_seq_len,
                                            tokens.device)
    output = query
    for m in self.decoder_layers:
        output = m(
            query=output,
            key=embed,
            value=embed,
            attn_masks=location_mask,
            key_padding_mask=padding_mask)
    output = output.permute(1, 0, 2)  # (N, T, E)

            # 最后做降维，重新变成tensor(8,26,38)
    logits = self.cls(output)  # (N, T, C)
    return {'feature': output, 'logits': logits}
```

反复迭代3次，反复矫正

4.融合操作ABIFuser

将Encoder视觉模型和Decoder 文本模型的结果拼接在一起。没啥好说的f = torch.cat一下完事了。一个权重值，看看视觉和文本哪个模型对最终预测更重要。

代码如下（示例）：

```
    def forward(self, l_feature, v_feature):

        f = torch.cat((l_feature, v_feature), dim=2)
        f_att = torch.sigmoid(self.w_att(f))
        output = f_att * v_feature + (1 - f_att) * l_feature

        logits = self.cls(output)  # (N, T, C)

        return {'logits': logits}
```

5.损失函数和Inference

损失函数非常简单，就是3个一般的多分类交叉墒损失。

1.视觉模型损失。

2.文本模型损失。

3.融合模型损失。

```
    def forward(self, outputs, targets_dict, img_metas=None):

        assert 'out_enc' in outputs or \
            'out_dec' in outputs or 'out_fusers' in outputs
        losses = {}

        target_lens = [len(t) for t in targets_dict['targets']]
        flatten_targets = torch.cat([t for t in targets_dict['targets']])
                # 1.视觉模型损失。
        if outputs.get('out_enc', None):
            enc_input = self._flatten(outputs['out_enc']['logits'],
                                      target_lens)
            enc_loss = self._ce_loss(enc_input,
                                     flatten_targets) * self.enc_weight
            losses['loss_visual'] = enc_loss

         # 2.文本模型损失。
        if outputs.get('out_decs', None):
            dec_logits = [
                self._flatten(o['logits'], target_lens)
                for o in outputs['out_decs']
            ]
            dec_loss = self._loss_over_iters(dec_logits,
                                             flatten_targets) * self.dec_weight
            losses['loss_lang'] = dec_loss

                3.融合模型损失。
        if outputs.get('out_fusers', None):
```

```
            fusion_logits = [
                self._flatten(o['logits'], target_lens)
                for o in outputs['out_fusers']
            ]
            fusion_loss = self._loss_over_iters(
                fusion_logits, flatten_targets) * self.fusion_weight
            losses['loss_fusion'] = fusion_loss
        return losses
```

最终Inference的时候只要拼接操作ABIFuser后的输出就行，不用Encoder视觉模型和Decoder 文本

果。

总结

今天是和大家继续分享MMOCR之ABINET文字识别。

主要是一个多模态融合的思想。用文本模型提升模型整体的精度。

视觉模型可以看做是先验信息，通过文本模型进行矫正。最后融合在一起，输出最终的结果，比较有

得一读。

下一次是和大家分享关键信息抽取，都是一个完成的系列。

相关推荐