Supplementary materials for CLTR

Dingkang Liang\textsuperscript{1}, Wei Xu\textsuperscript{2}, and Xiang Bai\textsuperscript{1,†}

\textsuperscript{1} Huazhong University of Science and Technology, Wuhan 430074, China
\{dkliang,xbai\}@hust.edu.cn
\textsuperscript{2} Beijing University of Posts and Telecommunications, Beijing 100876, China
xuwei2020@bupt.edu.cn
\textsuperscript{†}Corresponding Author

This file provides additional information from three aspects, including more architecture details, more experiments, and the inference code.

1 Architecture Details

1.1 Transformer-encoder

In our experiments, the transformer-encoder consists of 6 encoder layers. A standard encoder layer contains Multi-head self-attention (\textit{MSA}) and Feed-Forward (\textit{MLP}) layer\textsuperscript{1} and the output of \(l\)-th layer is defined as:

\[
Z'_l = \text{MSA}(\text{LN}(Z_{l-1})) + Z_{l-1},
\]

where \(Z_l\) represents the output of \(l\)-th layer. The \textit{MLP} has two linear layers, a GELU \textsuperscript{2} activation function is used. For each layer, layer normalization (\textit{LN}) and residual connections are adopted. \textit{MSA} is an extension with \(m\) (set as 8) independent self-attention (\textit{SA}) modules:

\[
\text{MSA}(Z_{l-1}) = [SA_1(Z_{l-1}); SA_2(Z_{l-1}); \cdots ; SA_m(Z_{l-1})]W_O,
\]

where \(W_O\) is a projection matrix. Each \textit{SA} consists of the query (\(Q\)), key (\(K\)), and value (\(V\)):

\[
Q = Z_{l-1}W_Q, \quad K = Z_{l-1}W_K, \quad V = Z_{l-1}W_V,
\]

\[
\text{SA}(Z_l) = \text{softmax}\left(\frac{QK^T}{\sqrt{D}}\right)V,
\]

where \(W_Q/W_K/W_V\) are three trainable matrices.

\textsuperscript{1} See the paper \textsuperscript{6} for more details
1.2 Transformer-decoder

The transformer-decoder is similar to the transformer-encoder. The main difference is that the decoder contains additional cross-attention layers, which take different inputs to generate \( Q \), \( K \), and \( V \). Following [4], we adopt the conditional cross-attention, i.e., the \( Q \) are concatenated by the trainable embeddings \( Q_h \) and content query (from decoder self-attention).

2 More Experiments

Table 1. Ablation study on different backbone on ShanghaiTech Part A dataset.

<table>
<thead>
<tr>
<th>Method</th>
<th>Backbone</th>
<th>Multi-scale</th>
<th>Feature size</th>
<th>Localization (( \sigma = 8 ))</th>
<th>Counting</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSDDN</td>
<td>ResNet-101</td>
<td>✔</td>
<td>1/4</td>
<td>-</td>
<td>65.9</td>
</tr>
<tr>
<td>LSC-CNN</td>
<td>VGG-16</td>
<td>✔</td>
<td>1/4</td>
<td>63.9% 61.0% 62.4%</td>
<td>66.4</td>
</tr>
<tr>
<td>TopoCount</td>
<td>VGG-16</td>
<td>✔</td>
<td>1/8</td>
<td>74.6% 72.7% 73.6%</td>
<td>61.2</td>
</tr>
<tr>
<td>GL</td>
<td>VGG-19</td>
<td>✗</td>
<td>1/8</td>
<td>-</td>
<td>61.3</td>
</tr>
<tr>
<td>CLTR</td>
<td>VGG-16</td>
<td>✗</td>
<td>1/8</td>
<td>74.1% 72.5% 73.3%</td>
<td>57.7</td>
</tr>
<tr>
<td>CLTR</td>
<td>VGG-19</td>
<td>✗</td>
<td>1/8</td>
<td>74.6% 72.8% 73.7%</td>
<td>57.4</td>
</tr>
<tr>
<td>CLTR (ours)</td>
<td>ResNet-50</td>
<td>✗</td>
<td>1/32</td>
<td>74.9% 73.5% 74.2%</td>
<td>56.9</td>
</tr>
</tbody>
</table>

We think making an absolutely fair comparison is almost impossible because different methods adopt different backbones, feature fusion mechanisms, and feature sizes. In this paper, to verify the effectiveness of the proposed method, we just extract the single-scale and low-resolution (\( \frac{1}{32} \) of the input image) feature maps from the backbone (i.e., ResNet-50). However, to the best of our knowledge, there are no crowd localization methods that adopt this experiment setting. Thus, we conduct more experiments on the ShanghaiTech Part A dataset to analyze the influences from the backbone, and feature size, as listed on Tab. 1.

We mainly want to introduce a new Transformer-based crowd localization paradigm. Although multi-scale and higher-resolution feature can bring improvement, it is heuristics. More importantly, these heuristic designs will decrease the objectivity of the evaluation of our method. In the future, we would like to design reasonable fusion mechanisms for the proposed CLTR.

3 Inference code

In this section, we provide the simple inference code of our CLTR.

```python
from torch import nn
class CLTR(nn.Module):
    def __init__(self, backbone, transformer, num_classes, num_queries=500):
        pass
```

```python
super().__init__()

self.num_queries = num_queries
self.transformer = transformer
hidden_dim = transformer.d_model

# two output heads
self.classification_head = nn.Linear(hidden_dim, num_classes)
self.regression_head = nn.Linear(hidden_dim, out_channel)

# trainable queries
self.query_embed = nn.Embedding(num_queries, hidden_dim)

# reducing the channel dimension from C to c
self.input_proj = nn.Conv2d(backbone.num_channels, hidden_dim, kernel_size=1)

# we use the resnet50 as the backbone
self.backbone = backbone

def forward(self, input_samples):
    features, pos = self.backbone(input_samples)
    hs = self.transformer(self.input_proj(features), self.query_embed.weight, pos[-1])

    outputs_class = self.classification_head(hs)
    outputs_coordinates = self.regression_head(hs)
    out = {'pred_logits': outputs_class, 'pred_points': outputs_coordinates}

    return out
```

References

