Supplementary materials for CLTR

Dingkang Liang[1], Wei Xu[2], and Xiang Bai[1,†]

[1] Huazhong University of Science and Technology, Wuhan 430074, China
{dkliang,xbai}@hust.edu.cn
[2] Beijing University of Posts and Telecommunications, Beijing 100876, China
xuwei2020@bupt.edu.cn
[†]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 (MSA) and Feed-Forward (MLP) layer[1], and the output of l-th layer is defined as:

$$Z'_l = MSA(LN(Z_{l-1})) + Z_{l-1}, \tag{1}$$

$$Z_l = MLP(LN(Z'_l)) + Z'_l, \tag{2}$$

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

$$MSA(Z_{l-1}) = [SA_1(Z_{l-1}); SA_2(Z_{l-1}); \cdots ; SA_m(Z_{l-1})]W_O, \tag{3}$$

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

$$Q = Z_{l-1}W_Q, \quad K = Z_{l-1}W_K, \quad V = Z_{l-1}W_V, \tag{4}$$

$$SA(Z_l) = softmax(\frac{QK^T}{\sqrt{D}})V, \tag{5}$$

where $W_Q/W_K/W_V$ are three trainable matrices.

[1] See the paper [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.

Method	Backbone	Multi-scale	Feature size	Localization ($\sigma = 8$)			Counting	
				P(%)	R(%)	F(%)	MAE	MSE
PSDDN [3]	ResNet-101	✔	1/4	-	-	-	65.9	112.3
LSC-CNN [5]	VGG-16	✔	1/4	63.9%	61.0%	62.4%	66.4	117.0
TopoCount [1]	VGG-16	✔	1/8	**74.6%**	**72.7%**	**73.6%**	**61.2**	104.6
GL [7]	VGG-19	✗	1/8	-	-	-	61.3	**95.4**
CLTR	VGG-16	✗	1/8	74.1%	72.5%	73.3%	57.7	97.9
CLTR	VGG-19	✗	1/8	74.6%	72.8%	73.7%	57.4	96.1
CLTR (**ours**)	ResNet-50	✗	1/32	**74.9%**	**73.5%**	**74.2%**	**56.9**	**95.2**

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.

```
from torch import nn

class CLTR(nn.Module):
    def __init__(self, backbone, transformer, num_classes,
        num_queries=500):
```

```
        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

1. Abousamra, S., Hoai, M., Samaras, D., Chen, C.: Localization in the crowd with topological constraints. In: Proc. of the AAAI Conf. on Artificial Intelligence (2021)
2. Hendrycks, D., Gimpel, K.: Bridging nonlinearities and stochastic regularizers with gaussian error linear units (2016)
3. Liu, Y., Shi, M., Zhao, Q., Wang, X.: Point in, box out: Beyond counting persons in crowds. In: Proc. of IEEE Intl. Conf. on Computer Vision and Pattern Recognition (2019)
4. Meng, D., Chen, X., Fan, Z., Zeng, G., Li, H., Yuan, Y., Sun, L., Wang, J.: Conditional detr for fast training convergence. In: Porc. of IEEE Intl. Conf. on Computer Vision. pp. 3651–3660 (2021)

5. Sam, D.B., Peri, S.V., Sundararaman, M.N., Kamath, A., Radhakrishnan, V.B.: Locate, size and count: Accurately resolving people in dense crowds via detection. IEEE Transactions on Pattern Analysis and Machine Intelligence (2020)
6. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Proc. of Advances in Neural Information Processing Systems. pp. 5998–6008 (2017)
7. Wan, J., Liu, Z., Chan, A.B.: A generalized loss function for crowd counting and localization. In: Proc. of IEEE Intl. Conf. on Computer Vision and Pattern Recognition. pp. 1974–1983 (2021)