VOLO: Vision Outlooker for Visual Recognition笔记

- Paper: VOLO: Vision Outlooker for Visual Recognition
- Code: sail-sg/volo

0. Summary Keywords

- **finer-level** features
- a new **simple** and **light-weight** attention mechanism (**Outlooker**)
- **two-stage** (Outlooker+Transformer)

1. Introduction

1.1 Why

- 视觉Transformer模型在将细粒度特征和上下文方面编码到token表征方面的功效较低（low efficacy in encoding fine-level features and contexts into token representations）。The main factor limiting the performance of ViTs for ImageNet classification is their **low efficacy in encoding fine-level features into the token representations**.
- 直接采用细粒度图像token表征，自注意力机制的计算复杂度随token序列的长度呈平方递增。

1.2 What

- **VOLO (Vision Outlooker)**: Unlike self-attention that focuses on global dependency modeling at a coarse level (以224的图片为例，patch分辨率为16x16，token序列长度为14x14), the outlook attention aims to efficiently encode finer-level (以224的图片为例，patch分辨率为8x8，token序列长度为28x28) features and contexts into tokens.
- **Outlooker**是一种新的、简单的、轻量级的注意力机制。
- **VOLO**采用两阶段架构设计的方式同时考虑细粒度token表征和全局信息集合。

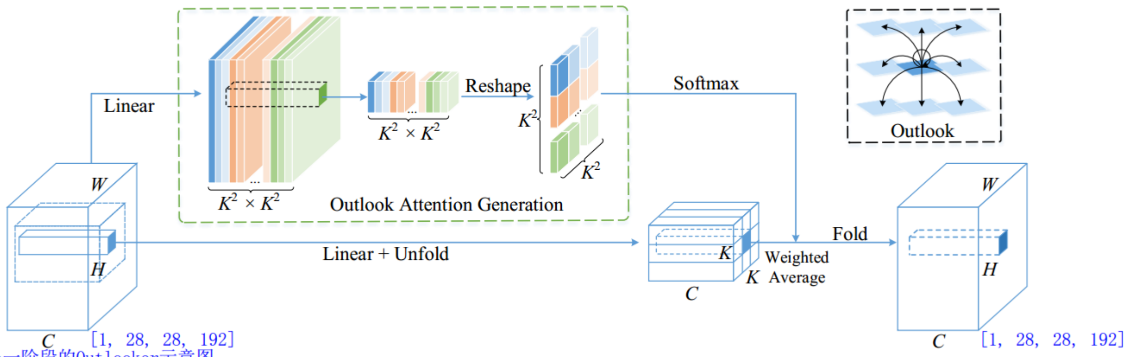
1.3 How

两阶段架构

- 堆叠的**Outlooker**：生成细粒度的token表征。以224的图片为例，在使用自注意力机制在粗粒度(token序列长度为14x14)特征上建立全局依赖关系之前，VOLO先将图像划分为分辨率较小(8x8)的patch，并使用多个Outlookers来在这种细粒度(28x28)尺度上编码token表征。所获得的token表征具有更强的表现力。
- 序列的**Transformer**：聚合全局信息。

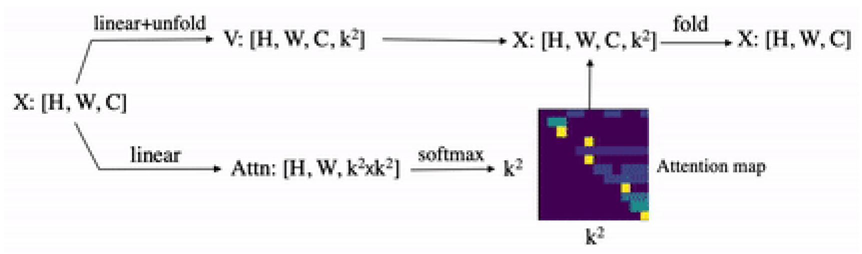
2. Method

- **Outlooker** + **Transformer**



这只是一阶段的Outlooker示意图

Figure 2. Illustration of outlook attention. The outlook attention matrix for a local window of size $K \times K$ can be simply generated from the center token with a linear layer followed by a reshape operation (highlighted by the green dash box). As the attention weights are generated from the center token within the window and act on the neighbor tokens and itself (as demonstrated in the black dash block), we name these operations as outlook attention.



Formally, given the input $\mathbf{X}$, each C-dim token is first projected, using two linear layers of weights $\mathbf{W}_A \in \mathbb{R}^{C\times K^4}$ and $\mathbf{W}_V \in \mathbb{R}^{C\times C}$, into outlook weights $\mathbf{A} \in \mathbb{R}^{H\times W\times K^4}$, and value representation $\mathbf{V} \in \mathbb{R}^{H\times W\times C}$, respectively. Let $\mathbf{V}_{\Delta_{i,j}} \in \mathbb{R}^{C\times K^2}$ denote all the values within the local window centered at (i, j), *i.e.*,

$$\mathbf{V}_{\Delta_{i,j}} = \{\mathbf{V}_{i+p-\lfloor\frac{K}{2}\rfloor, j+q-\lfloor\frac{K}{2}\rfloor}\}, \quad 0 \le p, q < K. \quad (3)$$

```
################# initialization ####################
##对应文中Figure2中下面的Linear
self.v = nn.Linear(dim, dim, bias=qkv_bias)

################# code in forward ####################
##h, w = 14, 14
h, w = math.ceil(H / self.stride), math.ceil(W / self.stride)
##对应文中的公式(3), 及Figure2中的Unfold
##1728=192*9; V-delta_ij包含9个元素，由于是通道内信息交互，所以是通道数192*元素个数9
##196=14*14; (28-3+1)/2 + 1 = 14
##[1, 192, 28, 28] -> [1, 1728, 196] -> [1, 6, 32, 9, 196] -> [1, 6, 196, 9, 32]
v = self.unfold(v).reshape(B, self.num_heads, C // self.num_heads,
                           self.kernel_size * self.kernel_size,
                           h * w).permute(0, 1, 4, 3, 2)  # B,H,N,kxk,C/H
```

- **Outlook Attention Generation**: The outlook weight at location (i; j) is directly used as attention weight for value aggregation, by reshaping it to A^i,j属于R(KxK x KxK), followed by a Softmax function. （对应上图的上面分支）

$$\mathbf{Y}_{\Delta_{i,j}} = \mathrm{MatMul}(\mathrm{Softmax}(\hat{\mathbf{A}}_{i,j}), \mathbf{V}_{\Delta_{i,j}}). \quad (4)$$

```
################# initialization ####################
##对应文中Figure2中上面的Linear
##3**4 * 6 = 486
self.attn = nn.Linear(dim, kernel_size**4 * num_heads)

################# code in forward ####################
##对应文中Figure2中的Outlook Attention Generation
```

```
##[1, 28, 28, 192] -> [1, 192, 28, 28] -> [1, 192, 14, 14] -> [1, 14, 14,
192]
attn = self.pool(x.permute(0, 3, 1, 2)).permute(0, 2, 3, 1)
##[1, 14, 14, 192] -> [1, 14, 14, 486] -> [1, 196, 6, 9, 9] -> [1, 6, 196,
9, 9]
##对应文中Figure2中上面的Linear、Reshape
attn = self.attn(attn).reshape(
    B, h * w, self.num_heads, self.kernel_size * self.kernel_size,
    self.kernel_size * self.kernel_size).permute(0, 2, 1, 3, 4)  #
B,H,N,kxk,kxk
attn = attn * self.scale
##对应文中的公式(4)中的Softmax(A^i,j)
attn = attn.softmax(dim=-1)
attn = self.attn_drop(attn)
```

- **Dense aggregation**: Outlook attention aggregates the projected value representations densely. Summing up the different weighted values at the same location yields the output

$$\tilde{\mathbf{Y}}_{i,j} = \sum_{0 \le m,n < K} \mathbf{Y}^{i,j}_{\Delta_{i+m-\lfloor \frac{K}{2} \rfloor, j+n-\lfloor \frac{K}{2} \rfloor}}. \qquad (5)$$

```
##对应文中的公式(4)
##      attn.shape为[1, 6, 196, 9,  9]
##         v.shape为[1, 6, 196, 9, 32]
##(attn @ v).shape为[1, 6, 196, 9, 32]
##[1, 6, 196, 9, 32] -> [1, 6, 32, 9, 196] -> [1, 1728, 196]
x = (attn @ v).permute(0, 1, 4, 3, 2).reshape(
    B, C * self.kernel_size * self.kernel_size, h * w)
##对应文中的公式(5)，及Figure2中的Fold
##[1, 1728, 196] -> [1, 192, 28, 28]
x = F.fold(x, output_size=(H, W), kernel_size=self.kernel_size,
           padding=self.padding, stride=self.stride)

##After outlook attention, a linear layer is often adopted as done in self-
attention.
##[1, 192, 28, 28] -> [1, 28, 28, 192] -> [1, 28, 28, 192]
x = self.proj(x.permute(0, 2, 3, 1))
##[1, 28, 28, 192] -> [1, 28, 28, 192]
x = self.proj_drop(x)
```

更新于2021-06-30