CLIPBERT

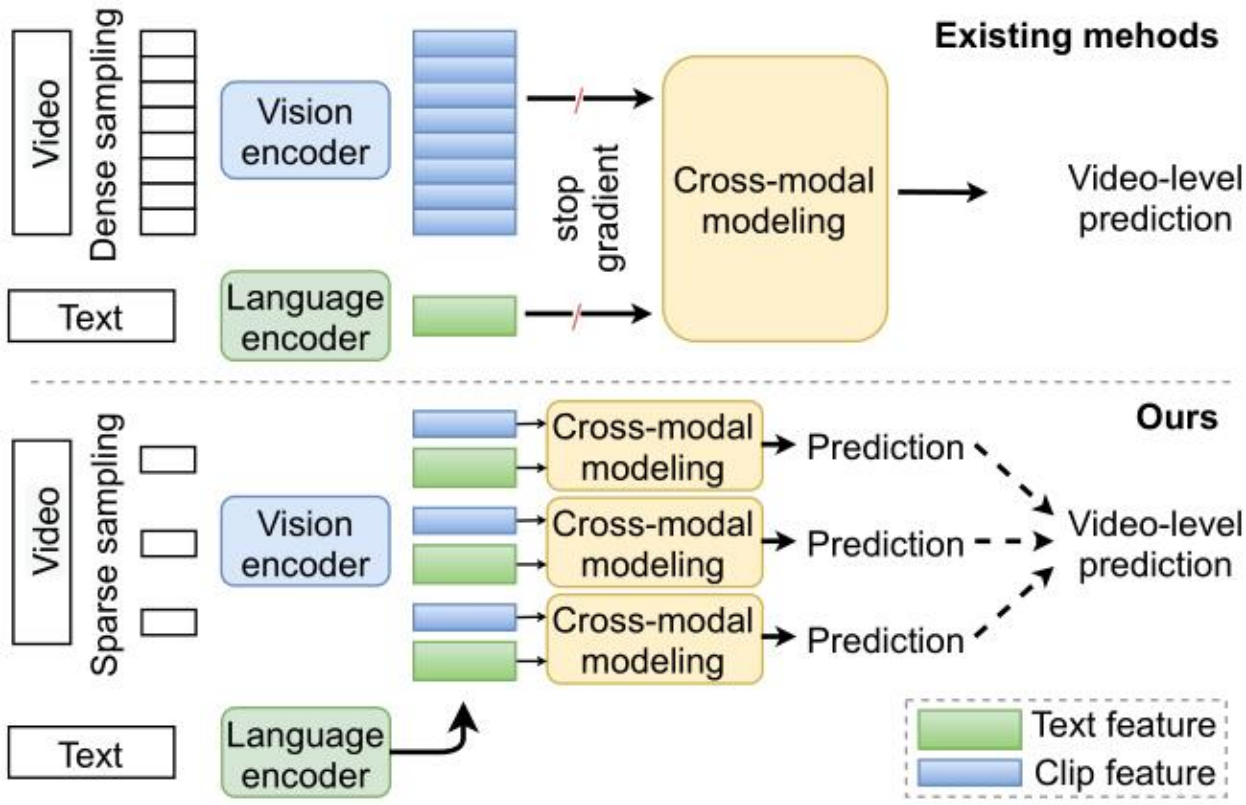
- CLIPBERT，一个通用而有效的端到端视频和语言学习框架

处理这些跨模式任务的实际范例是，首先从预训练的视觉模型中提取密集的视频特征，并从预训练的语言模型中提取文本特征，然后应用多模式融合在共享的嵌入空间中将这些固定表示形式结合在一起
遵循这一范式的现有方法取得了巨大成功，但存在两个主要缺点
tasks/domains中的断开连接：offline feature extractors通常针对不同于目标任务的tasks/domains进行训练。例如，从人类活动视频中学习的动作识别特征不一致地应用于generic-domain GIF视频上的下游视频问答。
多模式特征中的断开：从不同模式中学习的特征相互独立。例如，动作识别模型通常是从纯视频数据中训练出来的，没有文本输入，但应用于视频和语言任务。

端到端特定于任务的微调提供了一种缓解这些固有断开的方法。

然而，与大多数现有工作一样，从完整的视频帧序列中提取特征会对内存和计算造成过度需求，因此很难甚至不可能直接将特征提取器插入到视频+语言学习框架中以实现有效的端到端微调。

仅从视频中得到一个或者极少的clip用于训练（之前基本上是densely sampling）



SwimTransformer

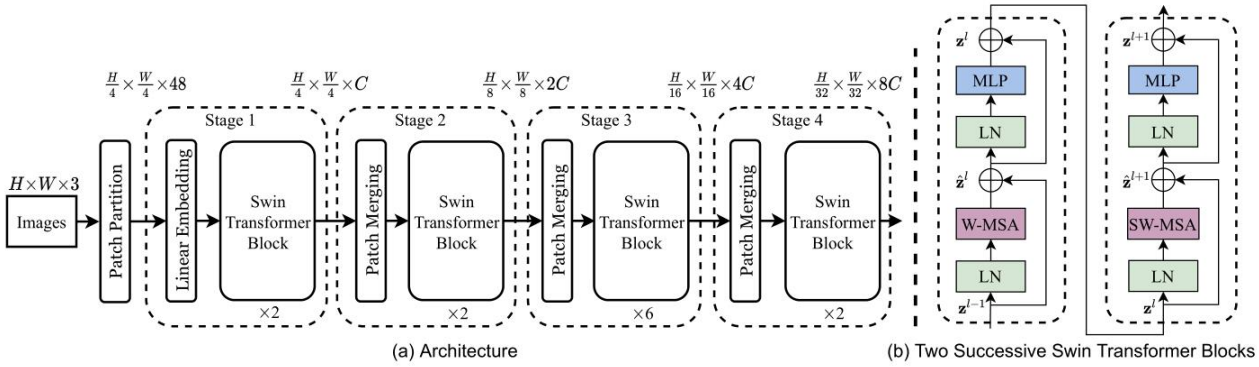


Figure 3. (a) The architecture of a Swin Transformer (Swin-T); (b) two successive Swin Transformer Blocks (notation presented with Eq. (3)). W-MSA and SW-MSA are multi-head self attention modules with regular and shifted windowing configurations, respectively.

```
# SwinTransformer总体设计
class SwinTransformer(nn.Module):
    def __init__(...):
        super().__init__()
        ...
        # absolute position embedding
        if self.ape:
            self.absolute_pos_embed = nn.Parameter(torch.zeros(1, num_patches,
embed_dim))

        self.pos_drop = nn.Dropout(p=drop_rate)

        # build layers
        self.layers = nn.ModuleList()
        for i_layer in range(self.num_layers):
            layer = BasicLayer(...)
            self.layers.append(layer)

        self.norm = norm_layer(self.num_features)
        self.avgpool = nn.AdaptiveAvgPool1d(1)
        self.head = nn.Linear(self.num_features, num_classes) if num_classes > 0
else nn.Identity()

    def forward_features(self, x):
        x = self.patch_embed(x)
        if self.ape:
            x = x + self.absolute_pos_embed
        x = self.pos_drop(x)

        for layer in self.layers:
            x = layer(x)

        x = self.norm(x)  # B L C
        x = self.avgpool(x.transpose(1, 2))  # B C 1
        x = torch.flatten(x, 1)
        return x
```

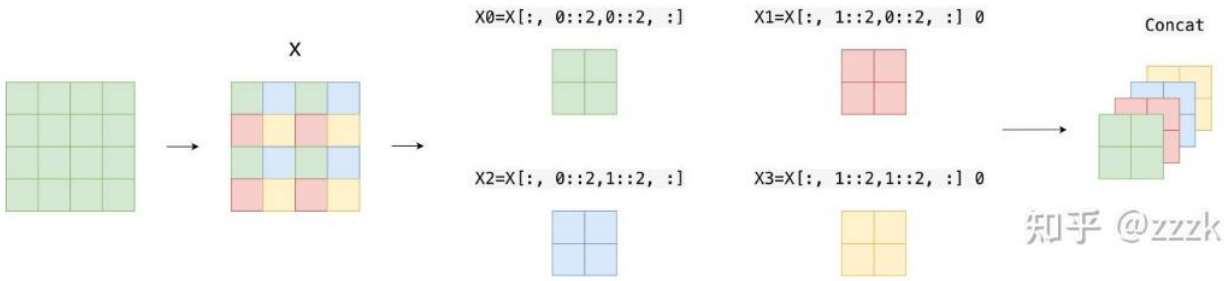
```
    def forward(self, x):
        x = self.forward_features(x)
        x = self.head(x)
        return x
```

images先做了一个Patch Embedding，将图片切成一个个图块，并嵌入到Embedding
再通过4个Stage，每个Stage里，由Patch Merging和多个Block组成。
其中Patch Merging模块主要在每个Stage一开始降低图片分辨率。

- Patch Merging

该模块的作用是在每个Stage开始前做降采样，用于缩小分辨率，调整通道数 进而形成**层次化**的设计，同时也能节省一定运算量。 每次降采样是两倍，因此在行方向和列方向上，间隔2选取元素。

然后拼接在一起作为一整个张量，最后展开。此时通道维度会变成原先的4倍（因为H,W各缩小2倍），此时再通过一个全连接层再调整通道维度为原来的两倍



$$
\begin{aligned}
\hat{\mathbf{z}}^{l} &= \text{W-MSA}\left(\text{LN}\left(\mathbf{z}^{l-1}\right)\right) + \mathbf{z}^{l-1}, \\
\mathbf{z}^{l} &= \text{MLP}\left(\text{LN}\left(\hat{\mathbf{z}}^{l}\right)\right) + \hat{\mathbf{z}}^{l}, \\
\hat{\mathbf{z}}^{l+1} &= \text{SW-MSA}\left(\text{LN}\left(\mathbf{z}^{l}\right)\right) + \mathbf{z}^{l}, \\
\mathbf{z}^{l+1} &= \text{MLP}\left(\text{LN}\left(\hat{\mathbf{z}}^{l+1}\right)\right) + \hat{\mathbf{z}}^{l+1},
\end{aligned}
$$

- Block的前向代码

```
def forward(self, x):
    H, W = self.input_resolution
    B, L, C = x.shape
    assert L == H * W, "input feature has wrong size"

    shortcut = x
    x = self.norm1(x)
    x = x.view(B, H, W, C)

    # cyclic shift
    if self.shift_size > 0:
```

```
        shifted_x = torch.roll(x, shifts=(-self.shift_size, -self.shift_size),
dims=(1, 2))
    else:
        shifted_x = x

    # partition windows
    x_windows = window_partition(shifted_x, self.window_size)  # nW*B,
window_size, window_size, C
    x_windows = x_windows.view(-1, self.window_size * self.window_size, C)  #
nW*B, window_size*window_size, C

    # W-MSA/SW-MSA
    attn_windows = self.attn(x_windows, mask=self.attn_mask)  # nW*B,
window_size*window_size, C

    # merge windows
    attn_windows = attn_windows.view(-1, self.window_size, self.window_size, C)
    shifted_x = window_reverse(attn_windows, self.window_size, H, W)  # B H' W' C

    # reverse cyclic shift
    if self.shift_size > 0:
        x = torch.roll(shifted_x, shifts=(self.shift_size, self.shift_size), dims=
(1, 2))
    else:
        x = shifted_x
    x = x.view(B, H * W, C)

    # FFN
    x = shortcut + self.drop_path(x)
    x = x + self.drop_path(self.mlp(self.norm2(x)))

    return x
```

为了更好的和其他window进行信息交互，Swin Transformer还引入了shifted window操作。

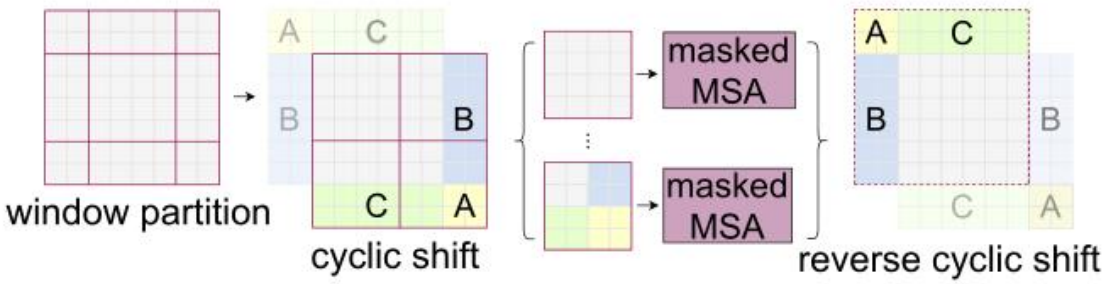


Figure 4. Illustration of an efficient batch computation approach for self-attention in shifted window partitioning.

先对特征图进行LayerNorm
通过self.shift_size决定是否需要对特征图进行shift
然后将特征图切成一个个窗口
计算Attention，通过self.attn_mask来区分Window Attention还是Shift Window Attention
将各个窗口合并回来
如果之前有做shift操作，此时进行reverse shift，把之前的shift操作恢复
做dropout和残差连接
再通过一层LayerNorm+全连接层，以及dropout和残差连接

引入window这一个概念，将CNN的局部性引入，还能控制模型整体计算量。在Shift Window Attention部分，用一个mask和移位操作，很巧妙的实现计算等价。