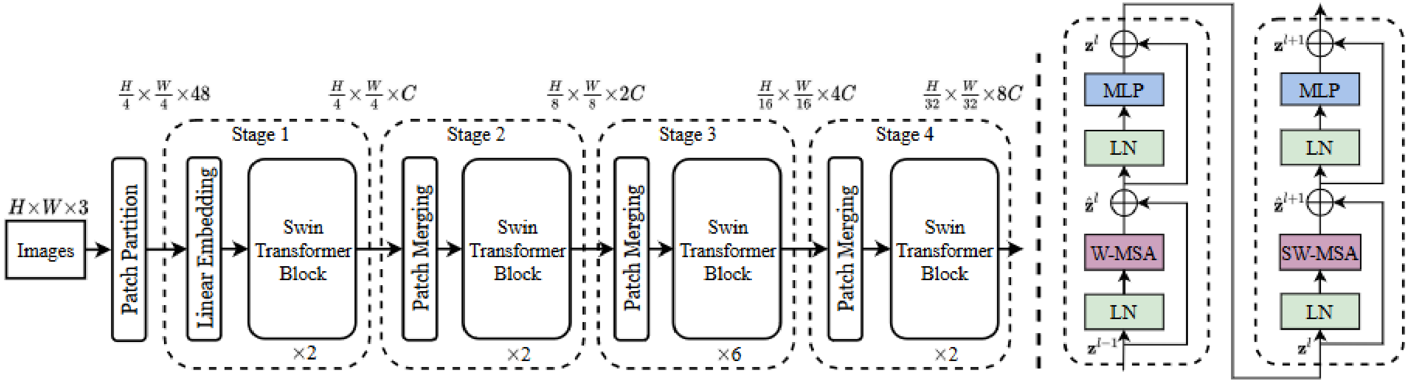
Swin Transformer

汇报人：王浩宇

https://arxiv.org/pdf/2103.14030
https://github.com/microsoft/Swin-Transformer
https://github.com/zhuomo-1/deep-learning-for-image-processing
https://www.bilibili.com/video/BV1yg411K7Yc/?spm_id_from=pageDriver&vd_source=2d0dfe19186ab4fcdb0dfc3bf9953784

Architecture



(a) Architecture

(b) Two Successive Swin Transformer Blocks

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.

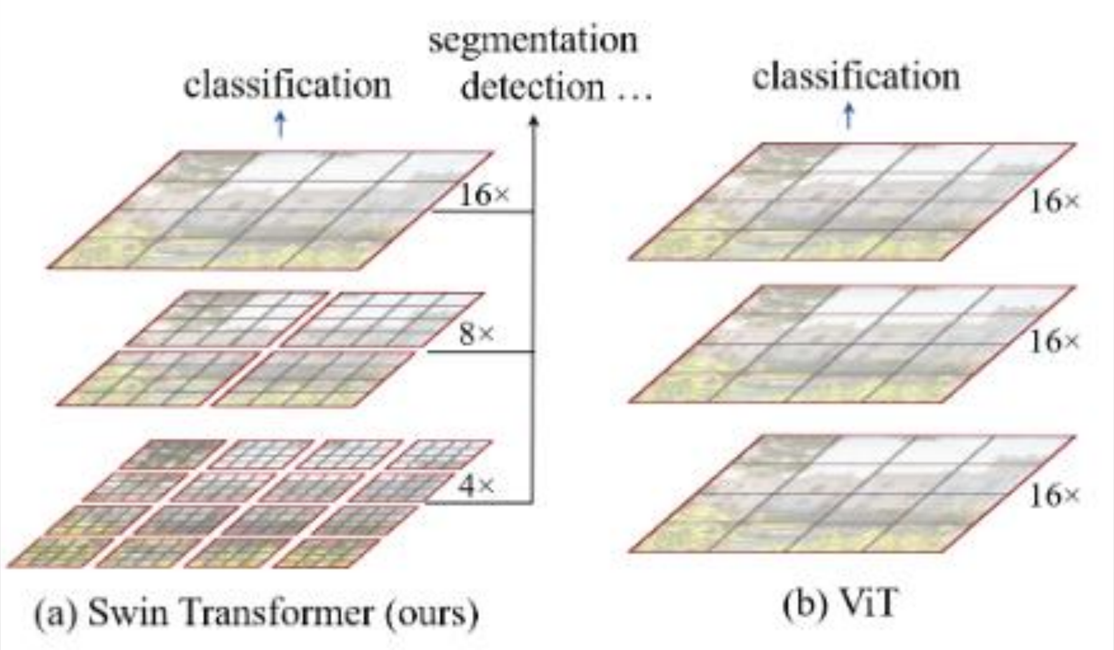
	downsp. rate (output size)	Swin-T	Swin-S	Swin-B	Swin-L
stage 1	4× (56×56)	concat 4×4, 96-d, LN	concat 4×4, 96-d, LN	concat 4×4, 128-d, LN	concat 4×4, 192-d, LN
		[win. sz. 7×7, dim 96, head 3] × 2	[win. sz. 7×7, dim 96, head 3] × 2	[win. sz. 7×7, dim 128, head 4] × 2	[win. sz. 7×7, dim 192, head 6] × 2
stage 2	8× (28×28)	concat 2×2, 192-d , LN	concat 2×2, 192-d , LN	concat 2×2, 256-d , LN	concat 2×2, 384-d , LN
		[win. sz. 7×7, dim 192, head 6] × 2	[win. sz. 7×7, dim 192, head 6] × 2	[win. sz. 7×7, dim 256, head 8] × 2	[win. sz. 7×7, dim 384, head 12] × 2
stage 3	16× (14×14)	concat 2×2, 384-d , LN	concat 2×2, 384-d , LN	concat 2×2, 512-d , LN	concat 2×2, 768-d , LN
		[win. sz. 7×7, dim 384, head 12] × 6	[win. sz. 7×7, dim 384, head 12] × 18	[win. sz. 7×7, dim 512, head 16] × 18	[win. sz. 7×7, dim 768, head 24] × 18
stage 4	32× (7×7)	concat 2×2, 768-d , LN	concat 2×2, 768-d , LN	concat 2×2, 1024-d , LN	concat 2×2, 1536-d , LN
		[win. sz. 7×7, dim 768, head 24] × 2	[win. sz. 7×7, dim 768, head 24] × 2	[win. sz. 7×7, dim 1024, head 32] × 2	[win. sz. 7×7, dim 1536, head 48] × 2

Table 7. Detailed architecture specifications.

Parameters

```
def __init__(self,
             patch_size=4,#卷积核大小，也是每个最小的ceil的尺寸，所以也是卷积的步长
             in_chans=3, #输入通道
             num_classes=1000,#输出通道
             embed_dim=96,#论文中的C，原始dimension
             depths=(2, 2, 6, 2),#block数量
             num_heads=(3, 6, 12, 24),#Muti-head head个数
             window_size=7,#window的尺寸
             mlp_ratio=4.,#FC隐藏层上采样倍数
             qkv_bias=True,#是否使用偏置
             drop_rate=0.,
             attn_drop_rate=0.,
             drop_path_rate=0.1,#每一个block中的dpr
             norm_layer=nn.LayerNorm,
             patch_norm=True,
             use_checkpoint=False, **kwargs):
    super().__init__()
```

Main idea



segmentation
classification
detection ...
classification
16×
8×
4×
16×
16×
16×
(a) Swin Transformer (ours)
(b) ViT

How windows work

```python
#将大的特征图划分成窗口
def window_partition(x, window_size: int):
    """
    将feature map按照window_size划分成一个个没有重叠的window
    Args:
        x: (B, H, W, C)
        window_size (int): window size(M)

    Returns:
        windows: (num_windows*B, window_size, window_size, C)
    """
    B, H, W, C = x.shape
    x = x.view(B, H // window_size, window_size, W // window_size, window_size, C)
    # permute: [B, H//Mh, Mh, W//Mw, Mw, C] -> [B, H//Mh, W//Mh, Mw, Mw, C]
    # view: [B, H//Mh, W//Mw, Mh, Mw, C] -> [B*num_windows, Mh, Mw, C]
    windows = x.permute(0, 1, 3, 2, 4, 5).contiguous().view(-1, window_size, window_size, C)#permute后的数据会变成内存上不连续的数据，需要用contiguous继续连接起来
    return windows

#将窗口化的特征图还原成正常尺寸
def window_reverse(windows, window_size: int, H: int, W: int):#对应分割前的高宽
    """
    将一个个window还原成一个feature map
    Args:
        windows: (num_windows*B, window_size, window_size, C)
        window_size (int): Window size(M)
        H (int): Height of image
        W (int): Width of image

    Returns:
        x: (B, H, W, C)
    """
    B = int(windows.shape[0] / (H * W / window_size / window_size))#还原第一个维度为batch，就是把windows后的第一个维度的数量，除以windows的总数量
    # view: [B*num_windows, Mh, Mw, C] -> [B, H//Mh, W//Mw, Mh, Mw, C]
    x = windows.view(B, H // window_size, W // window_size, window_size, window_size, -1)
    # permute: [B, H//Mh, W//Mw, Mh, Mw, C] -> [B, H//Mh, Mh, W//Mw, Mw, C]
    # view: [B, H//Mh, Mh, W//Mw, Mw, C] -> [B, H, W, C]
    x = x.permute(0, 1, 3, 2, 4, 5).contiguous().view(B, H, W, -1)
    return x
```

PatchEmbed

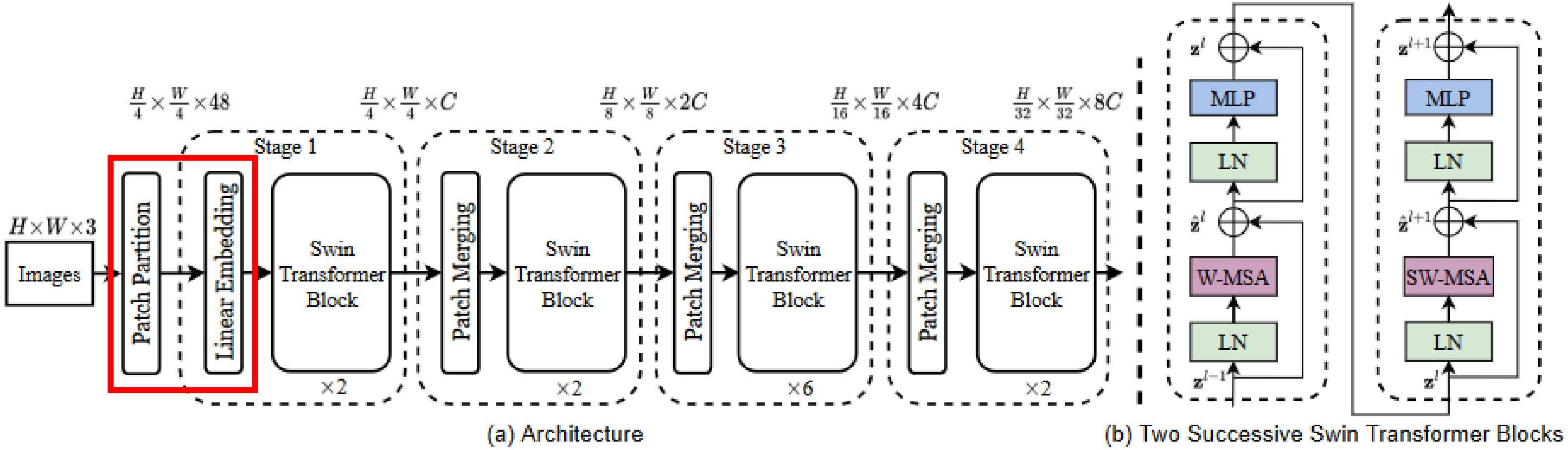


Figure 3. (a) The architecture of a Swin Transformer (Swin-T); (b) two successive Swin Transformer Blocks (notation presented with Eq. (3)). W-MSA and SW-MSA are multi-head self attention modules with regular and shifted windowing configurations, respectively.

PatchEmbed

```python
class PatchEmbed(nn.Module):
    """
    2D Image to Patch Embedding
    """
    def __init__(self,
                 patch_size=4,
                 in_c=3,
                 embed_dim=96,
                 norm_layer=None):
        super().__init__()
        patch_size = (patch_size, patch_size)
        self.patch_size = patch_size
        self.in_chans = in_c
        self.embed_dim = embed_dim
        self.proj = nn.Conv2d(in_c, embed_dim, kernel_size=patch_size, stride=patch_size)#定义一个实现下采样的卷积层，步长和k size 都是4
        self.norm = norm_layer(embed_dim) if norm_layer else nn.Identity()

    def forward(self, x):
        _, _, H, W = x.shape

        # padding
        # 如果输入图片的H，W不是patch_size的整数倍，需要进行padding
        pad_input = (H % self.patch_size[0] != 0) or (W % self.patch_size[1] != 0)
        if pad_input:
            # to pad the last 3 dimensions,
            # (W_left, W_right, H_top,H_bottom, C_front, C_back)
            x = F.pad(x, (0, self.patch_size[1] - W % self.patch_size[1],
                          0, self.patch_size[0] - H % self.patch_size[0],
                          0, 0))

        # 下采样patch_size倍
        x = self.proj(x)
        _, _, H, W = x.shape
        # flatten: [B, C, H, W] -> [B, C, HW]
        # transpose: [B, C, HW] -> [B, HW, C]
        x = x.flatten(2).transpose(1, 2)
        x = self.norm(x)
        return x, H, W
```

PatchMerging

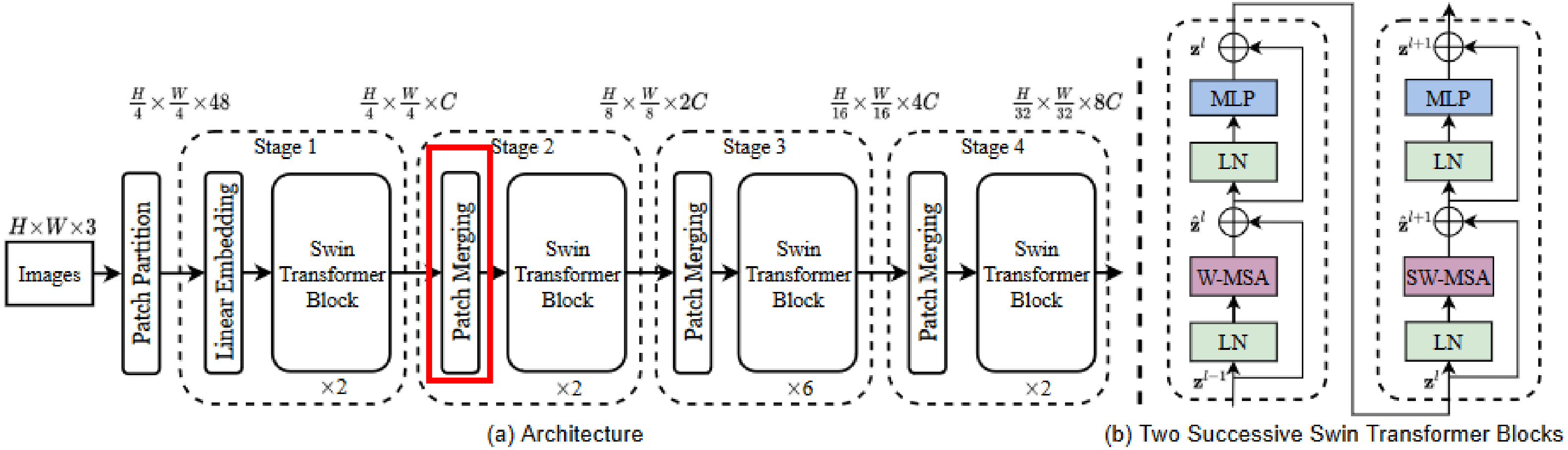
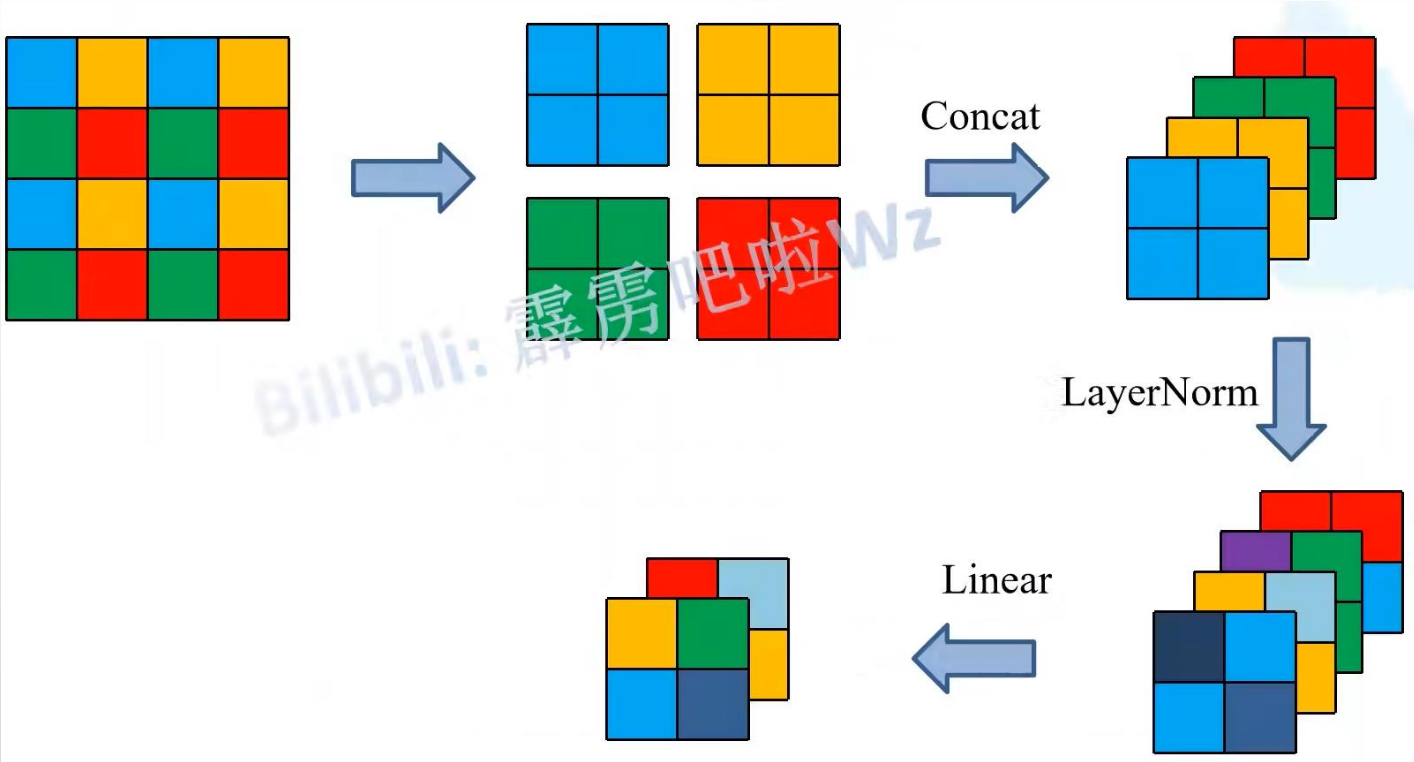


Figure 3. (a) The architecture of a Swin Transformer (Swin-T); (b) two successive Swin Transformer Blocks (notation presented with Eq. (3)). W-MSA and SW-MSA are multi-head self attention modules with regular and shifted windowing configurations, respectively.

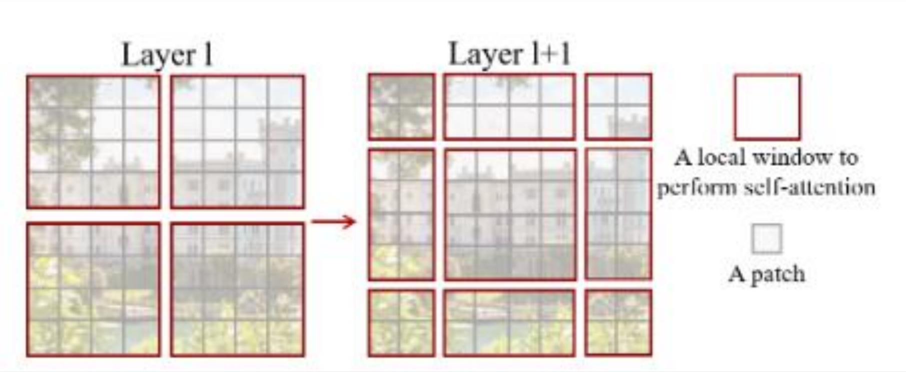
PatchMerging

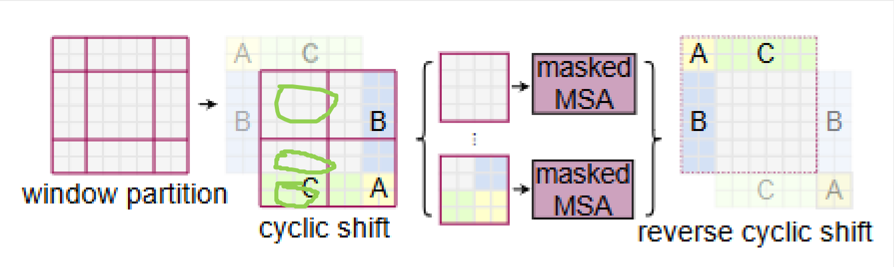


PatchMerging

```python
class PatchMerging(nn.Module):
    r""" Patch Merging Layer.

    Args:
        dim (int): Number of input channels.
        norm_layer (nn.Module, optional): Normalization layer.  Default: nn.LayerNorm
    """

    def __init__(self, dim, norm_layer=nn.LayerNorm):
        super().__init__()
        self.dim = dim
        self.reduction = nn.Linear(4 * dim, 2 * dim, bias=False)#使用FC结尾输出新的特征，下采样两倍，idm增加一倍，总体特征量为原来的一半
        self.norm = norm_layer(4 * dim)#linear前norm

    def forward(self, x, H, W):#特征，高，宽
        """
        x: B, H*W, C
        """
        B, L, C = x.shape
        assert L == H * W, "input feature has wrong size"#检验输入特征尺度是否正确

        x = x.view(B, H, W, C)#view成新的尺度

        # padding 如果输入feature map的H，W不是2的整数倍，需要进行padding
        pad_input = (H % 2 == 1) or (W % 2 == 1)
        if pad_input:
            # to pad the last 3 dimensions, starting from the last dimension and moving forward.
            # (C_front, C_back, W_left, W_right, H_top, H_bottom)
            # 注意这里的Tensor通道是[B, H, W, C]，所以会和官方文档有些不同
            x = F.pad(x, (0, 0, 0, W % 2, 0, H % 2))#padding顺序是从后往前，所以从C，W，H

        #开始patch merging操作，一张二维特征图采样出四张特征图然后拼接
        x0 = x[:, 0::2, 0::2, :]  # [B, H/2, W/2, C]
        x1 = x[:, 1::2, 0::2, :]  # [B, H/2, W/2, C]
        x2 = x[:, 0::2, 1::2, :]  # [B, H/2, W/2, C]
        x3 = x[:, 1::2, 1::2, :]  # [B, H/2, W/2, C]
        x = torch.cat([x0, x1, x2, x3], -1)  # [B, H/2, W/2, 4*C]
        x = x.view(B, -1, 4 * C)  # [B, H/2*W/2, 4*C],形成新的下采样后的特征图
        x = self.norm(x)#layernorm处理
        x = self.reduction(x)  # [B, H/2*W/2, 2*C]，实现通道数翻倍

        return x
```

Shift windows





Shift windows & Mask

```python
def create_mask(self, x, H, W):
    # calculate attention mask for SW-MSA
    # 保证Hp和Wp是window_size的整数倍，因为就是针对每一个window的操作
    Hp = int(np.ceil(H / self.window_size)) * self.window_size
    Wp = int(np.ceil(W / self.window_size)) * self.window_size
    # 拥有和feature map一样的通道排列顺序，方便后续window_partition
    img_mask = torch.zeros((1, Hp, Wp, 1), device=x.device)  # [1, Hp, Wp, 1]，初始化mask
    h_slices = (slice(0, -self.window_size),#0到倒数window size
                slice(-self.window_size, -self.shift_size),#倒数window size 到倒数shift size
                slice(-self.shift_size, None))#剩余的部分
    w_slices = (slice(0, -self.window_size),
                slice(-self.window_size, -self.shift_size),
                slice(-self.shift_size, None))

    #遍历每一个新的window的元素，给每个cell编码
    cnt = 0
    for h in h_slices:
        for w in w_slices:
            img_mask[:, h, w, :] = cnt
            cnt += 1

    mask_windows = window_partition(img_mask, self.window_size)  # [nW, Mh, Mw, 1] 窗口化 mask
    mask_windows = mask_windows.view(-1, self.window_size * self.window_size)  # [nW, Mh*Mw] 每一个窗口初始化一个mask
    attn_mask = mask_windows.unsqueeze(1) - mask_windows.unsqueeze(2)  # [nW, 1, Mh*Mw] - [nW, Mh*Mw, 1] 在指定位置插入一个新的维度，通过广播机制做减法
    # [nW, Mh*Mw, Mh*Mw]

    #做差为0的部分代表是同一编号的区域(也就是应当attention的区域)，不等于0的区域说明不在统一区域，于是就上一个“负无穷”的掩码，使该部分失效
    attn_mask = attn_mask.masked_fill(attn_mask != 0, float(-100.0)).masked_fill(attn_mask == 0, float(0.0))
    return attn_mask
```

Main forward

```python
def forward(self, x):
    # x: [B, L, C]
    x, H, W = self.patch_embed(x)#前两层
    x = self.pos_drop(x)

    #搭建各个stage
    for layer in self.layers:
        x, H, W = layer(x, H, W)

    x = self.norm(x)  # [B, L, C] 最后再norm一次
    x = self.avgpool(x.transpose(1, 2))  # [B, C, 1]，通过池化进行最终下采样
    x = torch.flatten(x, 1)#从C这个维度展开，相当于每个batch化为一维向量准备输出
    x = self.head(x)#FC层分类
    return x
```

Thank you！