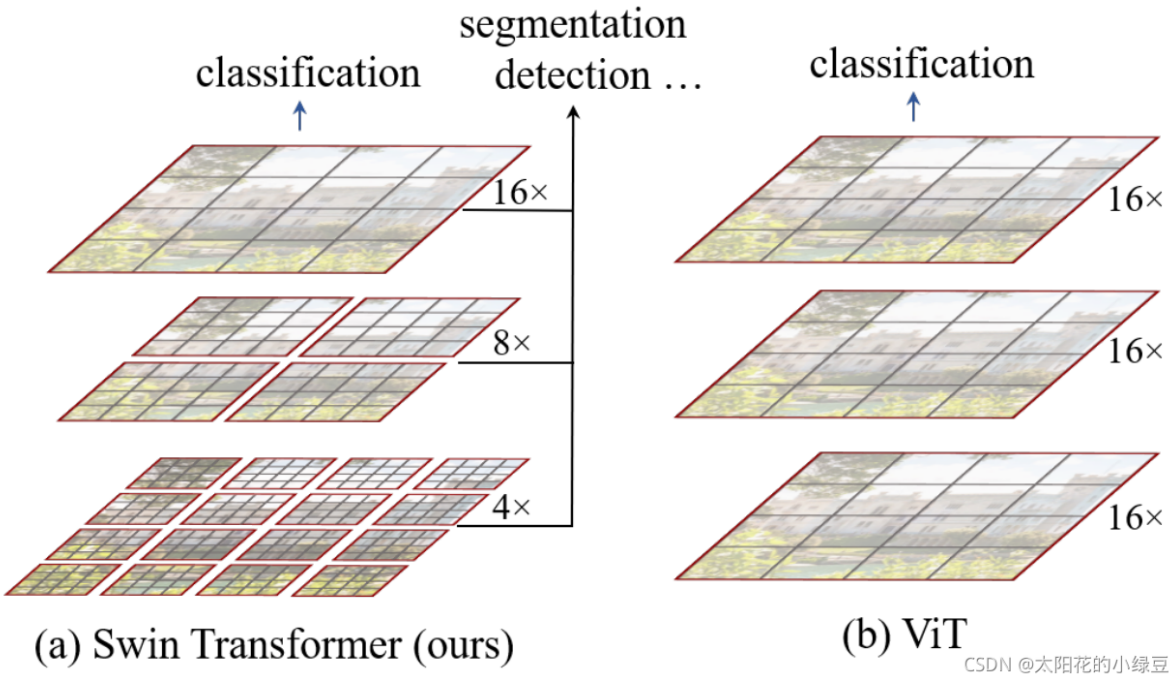
Swin Transformer【披着`Transformer`皮的卷积神经网络】

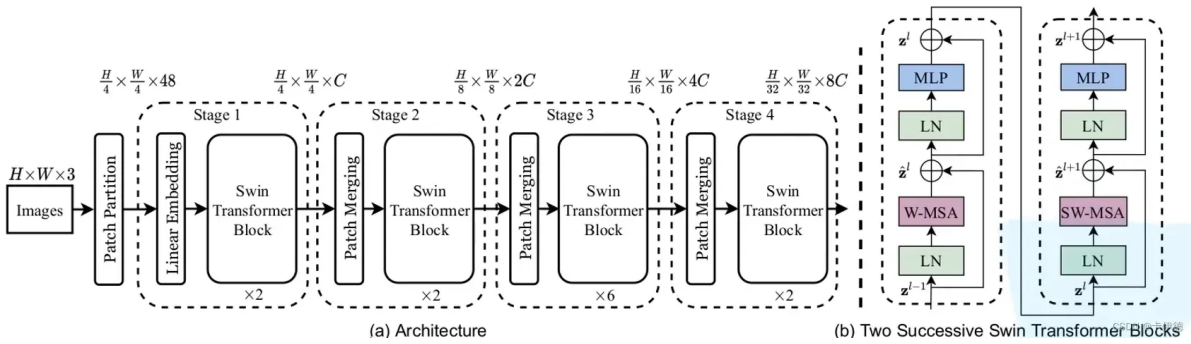
1 网络整体框架

- Swin Transformer使用了类似卷积神经网络中的层次化构建方法（Hierarchical feature maps），比如特征图尺寸中有对图像下采样4倍的，8倍的以及16倍的，这样的backbone有助于在此基础上构建目标检测，实例分割等任务。而在之前的Vision Transformer中是一开始就直接下采样16倍【可以理解划分成16*16的Patch】，后面的特征图也是维持这个下采样率不变。

`W-MSA（Windows Multi-Head Self-Attention）`

在下图的4倍下采样和8倍下采样中，将特征划分成了多个不相交的区域（Window）,并且Multi-Head Attention只在每个窗口（Window）内进行。而vit直接对整个（Global）特征图进行Multi-Head Self-Attention，这样可以减少计算量，特别是浅层特征图很大的时候【即输入图片很大时】。但是【note:]这么做虽然减少了计算量但是也隔绝了不同窗口之间的信息传递，所以后文又提出了Shifted Windows MultiHead Self- Attention（SW-MSA）的概念，通过此方法让信息在相邻的窗口中进行传递】。





- 首先将图片输入到Patch Partition模块中进行分块，即4*4相邻的像素为一个Patch，然后在channel方向展平。假设输入的是RGB三通道图片，那么每个Patch就有4*4=16个像素，然后每个像素有R、G、B三个值所以展平后是16×3=48，所以经过Patch Partition后图像shape由`[H,W,3]`变成了`[H/4,H/4,48]`。然后再通过Linear Embedding层对每个像素的channel数据做线性变换，由48变成C，即图像的shape再由`[H/4,H/4,48]`变成了`[H/4,H/4,C]`。跟Vision Transformer类似，源码中Patch Partition和LInear Embedding就是直接通过一个卷积层实现的。

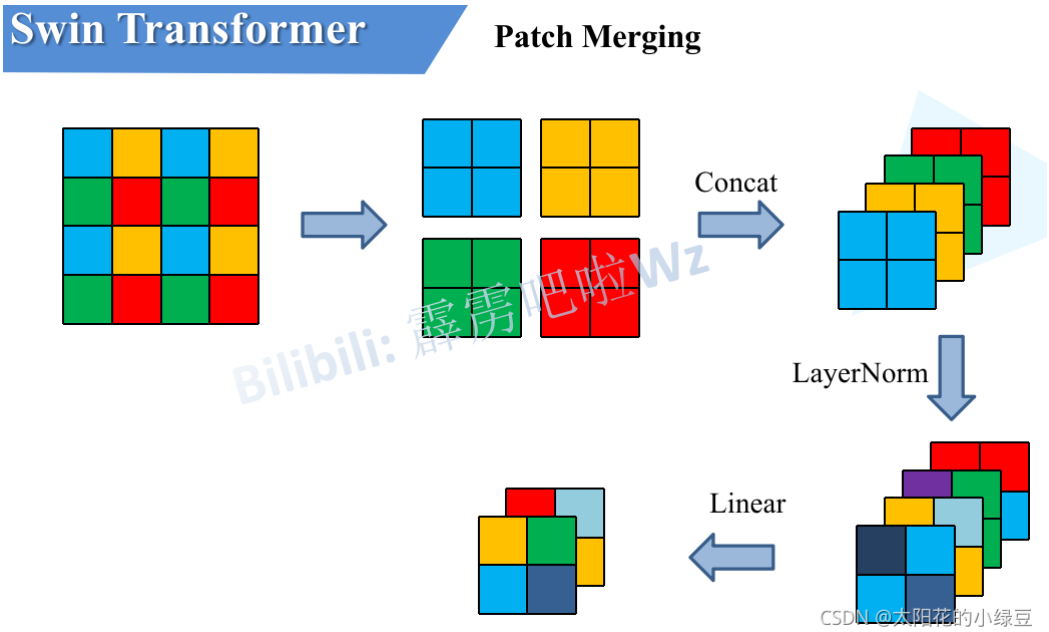
- 然后就是通过四个Stage构建不同大小的特征图，除了Stage1中先通过一个Linear Embeding层外，剩下三个stage都是先通过一个Patch Merging层进行下采样。然后都是重复堆叠Swin Transformer Block注意这里的Block其实有两种结构，如图(b)中所示，这两种结构的不同之处仅在于一个使用了W-MSA结构，一个使用了SW-MSA结构。而且这两个结构是成对使用的，先使用一个W-MSA结构再使用一个SW-MSA结构。所以你会发现堆叠Swin Transformer Block的次数都是偶数（因为成对使用）。
- 最后对于分类网络，后面还会接上一个Layer Norm层、全局池化层以及全连接层得到最终输出。图中没有画，但源码中是这样做的。

Swin Transformer Block中的`MLP`结构和Vision Transformer中的结构是一样的

2 Patch Merging详解

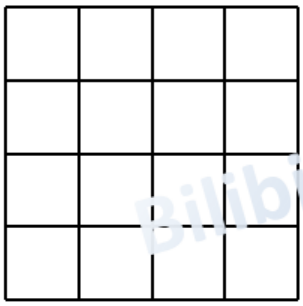
根据上文，在每个Stage中首先要通过一个Patch Merging层进行下采样（Stage1除外）。如下图所示，假设输入Patch Merging的是一个4×4的单通道特征图（feature map），Patch Merging会将每个2×2的相邻像素划分为一个Patch，然后将每个Patch中相同位置（同一颜色）像素给拼在一起就得到了4个feature map。接着将这四个feature map在深度方向【channel那个维度】进行concat拼接，然后通过一个Layernorm层。最后通过一个全连接层在feature map的深度方向做线性变化，将feature map的深度由C变成C/2。

通过Patch Merging层后，feature map的高和宽会减半，深度会翻倍。

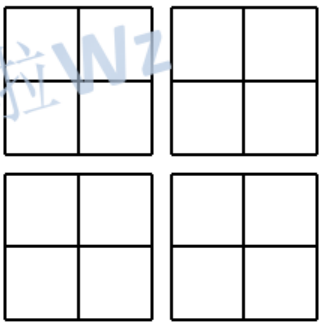


3 W-MSA详解

引入Windows Multi-head Self-Attention（W-MSA）模块是为了减少计算量。如下图所示，左侧使用的是普通MSA模块，对于feature map中的每个像素（或称作token，patch）在Self-Attention计算过程中需要和所有的像素去计算。但在图右侧，在使用W-MSA模块时，首先将feature map按照M×M（本例中M=2）大小划分为一个个Windows，然后单独对每个Windows内部进行Self-Attention。



Multi-head Self-Attention



Windows Multi-head Self-Attention

$$\Omega(\mathrm{MSA}) = 4\mathrm{hwC}^2 + 2(\mathrm{hw})^2\mathrm{C} \quad (1)$$

$$\Omega(\mathrm{W}\ -\mathrm{MSA}) = 4\mathrm{hwC}^2 + 2\mathrm{M}^2\mathrm{hwC} \quad (2)$$

- h代表feature map的高度
- w代表feature map的宽度
- C代表feature map的深度
- M代表每个窗口（Windows）的大小

MSA模块计算量

对于每个feature map中的每个像素（或称作token，patch），都要通过w_q, W_k, W_v生成对应的query，key以及value。这里假设q,k,v的向量长度与feature map的深度C保持一致。那么对应所有像素生成Q的过程如下式：

$$\mathrm{A}^{\mathrm{hw}\times\mathrm{C}} \cdot \mathrm{W}_{\mathrm{q}}^{\mathrm{C}\times\mathrm{C}} = \mathrm{Q}^{\mathrm{hw}\times\mathrm{C}}$$

- $A^{\mathrm{hw}\times\mathrm{C}}$为将所有像素 (token)拼接在一起得到的矩阵 (一共有hw个像素，每个像素的深度为C)
- $W_q^{C\times C}$为生成query的变换矩阵
- $Q^{\mathrm{hw}\times\mathrm{C}}$为所有像素通过$\mathrm{W}_{\mathrm{q}}^{\mathrm{C}\times\mathrm{C}}$得到的query拼接后的矩阵

根据矩阵运算的计算量公式可以得到生成Q的计算量为hw×C×C，生成K和V同理都是hwC^2，那么总共是$3hwC^2$。接下来Q和K^T相乘，对应计算量为$(hw)^2C$：

$$\mathrm{Q}^{\mathrm{hw}\times\mathrm{C}} \cdot \mathrm{K}^{\mathrm{T}(\mathrm{C}\times\mathrm{hw})} = \mathrm{X}^{\mathrm{hw}\times\mathrm{hw}}$$

接下来忽略除以$\sqrt{\mathrm{d}}$以及softmax的计算量，假设得到$\Lambda^{\mathrm{hw}\times\mathrm{hw}}$,最后还要乘以V，对应的计算量为(hw)$^2C$：

$$\Lambda^{\mathrm{hw}\times\mathrm{hw}} \cdot \mathrm{V}^{\mathrm{hw}\times\mathrm{C}} = \mathrm{B}^{\mathrm{hw}\times\mathrm{C}}$$

那么对应的单头Self-Attention模块，总共需要$3hwC^2 + (hw)^2C + (hw)^2C = 3hwC^2 + 2(hw)^2C$。而在实际使用过程中，使用的是多头的Mutli-head Self-Attention模块，多头注意力机制能够并行计算，所以多头注意力模块比单头注意力模块的计算量仅多了最后一个融合矩阵W_o的计算量hwC^2。

$$B^{hw\times C} . W_o^{C\times C} = O^{hw\times C}$$

所以总共加起来是：$4hwC^2 + 2(hw)^2C$

W-MSA模块计算量

对于W-MSA模块首先要将feature map划分到一个个窗口（Windows）中，假设每个窗口的宽高都是M，那么总共是得到$\frac{h}{M} \times \frac{w}{M}$个窗口，然后对每个窗口内使用多头注意力机制。刚刚计算高为h，宽为w，深度为C的feature map的计算量为$4hwC^2 + 2(hw)^2C$，这里每个窗口的高为M宽为M，带入公式可得：

$$4(MC)^2 + 2(M)^4C$$

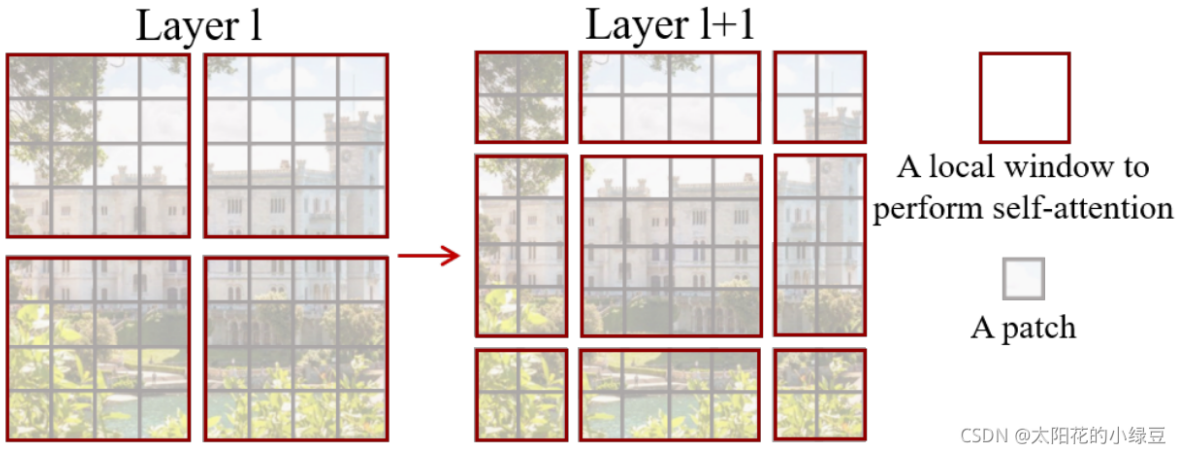
又因为有$\frac{h}{M} \times \frac{w}{M}$个窗口，则：

$$\frac{h}{M} \times \frac{w}{M} \times (4(MC)^2 + 2(M)^4C) = 4hwC^2 + 2M^2hwC$$

故使用W-MSA模块的计算量为：$4hwC^2 + 2M^2hwC$

4 SW-MSA详解

上文写道，采用W-MSA模块时，只会在每个窗口内进行自注意力计算，所以窗口与窗口之间是无法进行信息传递的。因此，作者提出了Shifted Windows Multi-Head Self-Attention（SW-MSA）模块，即进行偏移的W-MSA。如下图所示，左图使用的是W-MSA（假设是第l层），那么第l+1层使用的就是SW-MSA。对比两张图可知，窗口（Windows）发生了偏移【可以理解为窗口从左上角分别向右侧和下方各偏移了$\lfloor \frac{M}{2} \rfloor$个像素，由左图可以知道这张图片被划分成4块，每块的大小是4*4，所以M是4】。



根据上图，可以发现通过将窗口进行偏移后，由原来的4个窗口变成9个窗口了。后面又要对每个窗口内部进行MSA，这样做无疑增大了计算量。为了解决这个这个麻烦，作者又提出了Efficient batch computation for shifted configuration

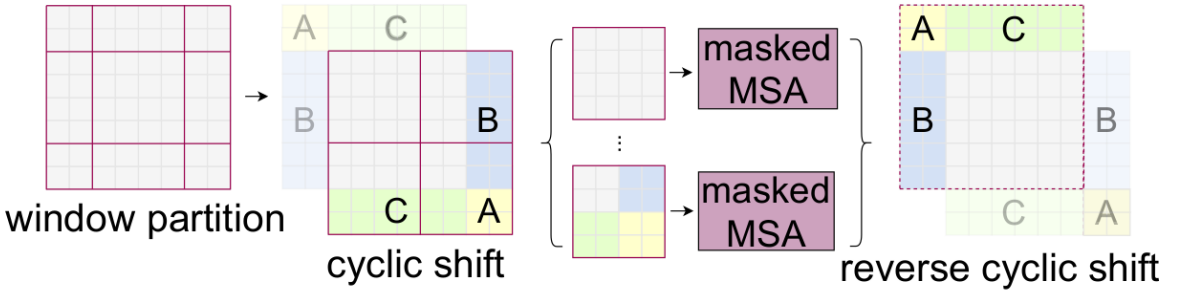
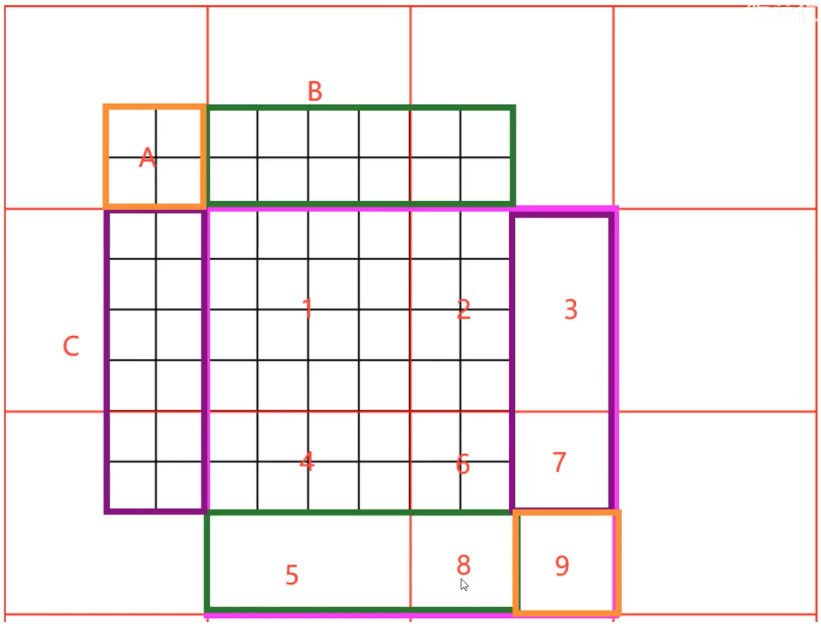
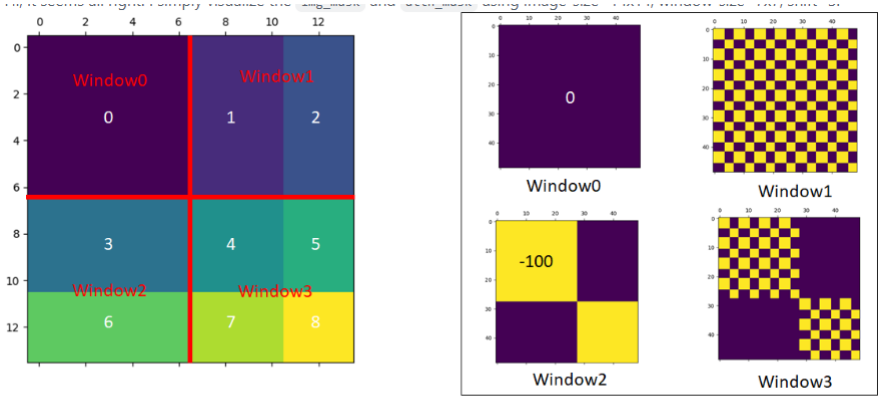


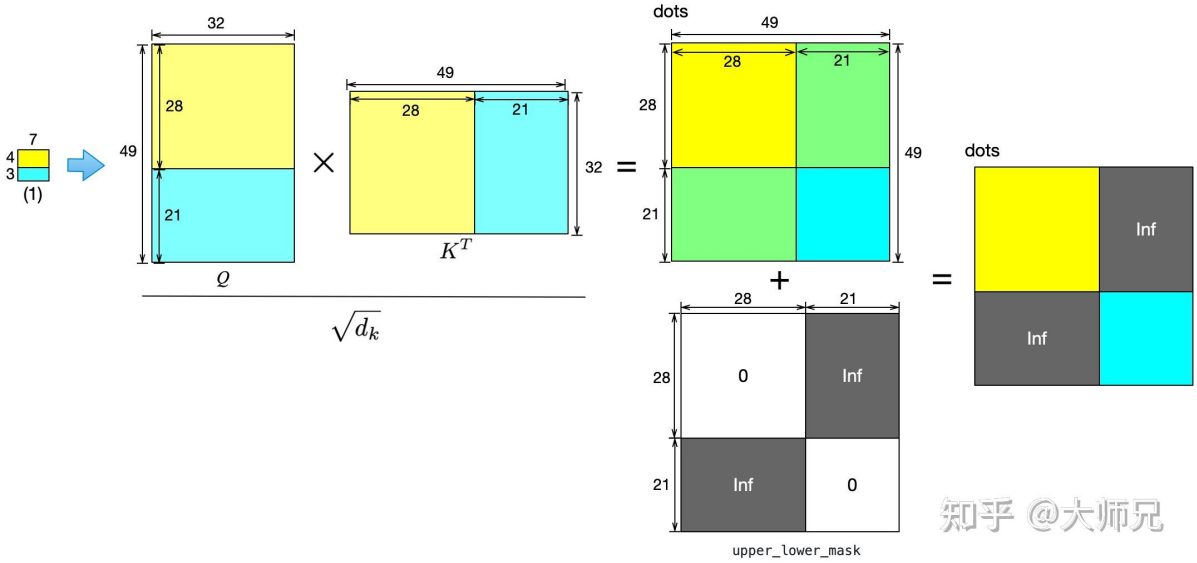
Figure 4. Illustration of an efficient batch computation approach for self-attention in shifted window partitioning.

用下面这个图更加清晰明了



这里计算Attention是在每一个4*4 的窗口中进行计算的，4与5是不连续的，那么就不能做注意力计算





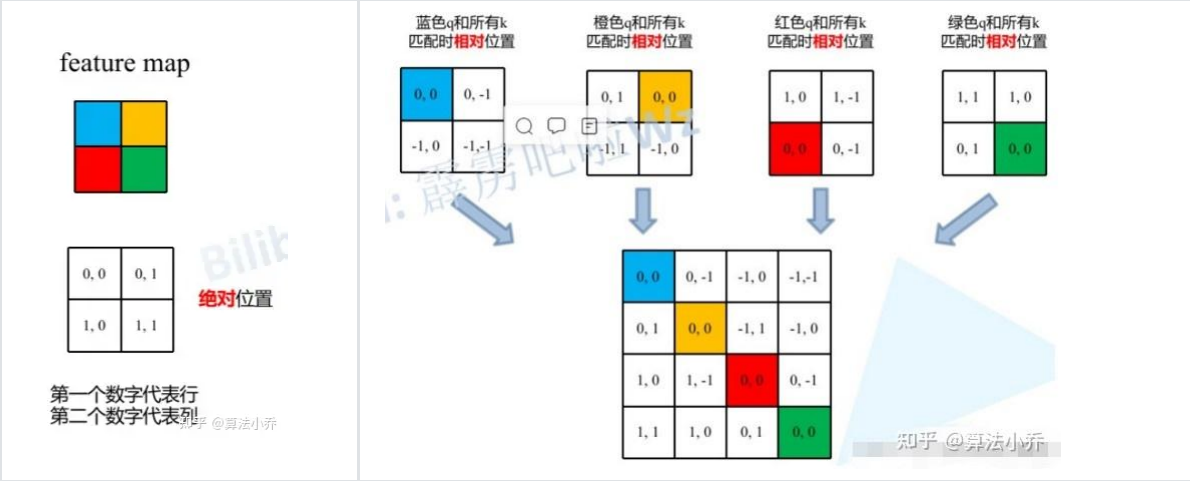
接着便是计算QK^T，在图中相同颜色区域的相互计算后会依旧保持颜色，而黄色和蓝色区域计算后会变成绿色，而绿色的部分便是无意义的相似度。这部分就是掩码，最后得到 0 和-100组成的二值矩阵。

5 Relative Position Bias详解

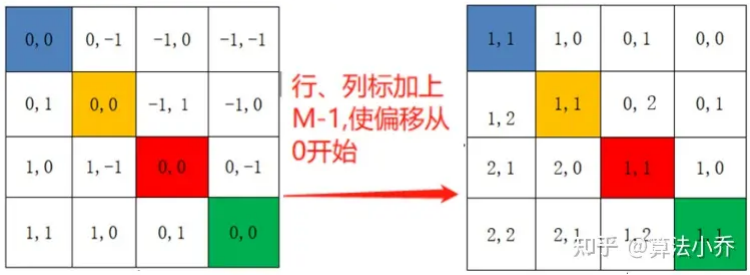
	ImageNet		COCO		ADE20k
	top-1	top-5	AP^{box}	AP^{mask}	mIoU
w/o shifting	80.2	95.1	47.7	41.5	43.3
shifted windows	**81.3**	**95.6**	**50.5**	**43.7**	**46.1**
no pos.	80.1	94.9	49.2	42.6	43.8
abs. pos.	80.5	95.2	49.0	42.4	43.2
abs.+rel. pos.	81.3	95.6	50.2	43.4	44.0
rel. pos. w/o app.	79.3	94.7	48.2	41.9	44.1
rel. pos.	**81.3**	**95.6**	**50.5**	**43.7**	**46.1**

$$\text{Attention}(Q,K,V)=\text{SoftMax}(\frac{QK^T}{\sqrt{d}}+B)V$$

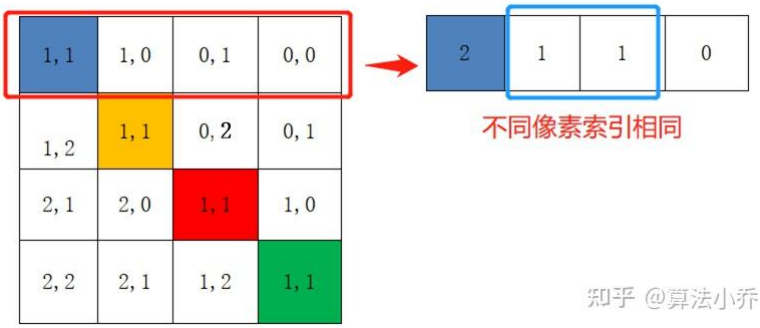
- 自注意力机制本身并不包含任何关于元素位置的信息。这意味着，如果没有位置编码，模型将无法区分序列中不同位置的元素，即使它们在内容上是相同的。因此，位置编码的引入是为了弥补这一缺陷，它通过向模型提供关于元素位置的信息，使得模型能够更好地理解和处理序列数据。



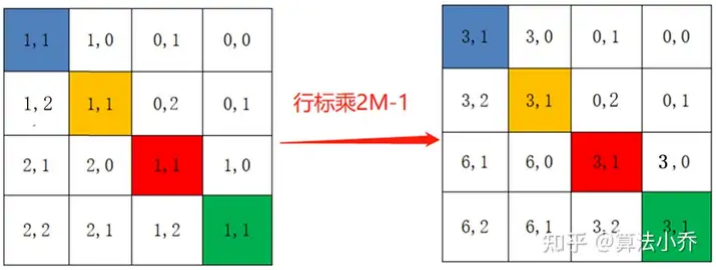
- 这里描述的一直是相对位置索引，并不是相对位置偏执参数，相对位置偏执参数是我们要求的B。但是上面的结果是二维的，而最终获取的**位置参数表对于每个Head来说是一维的**，故需要将上面的这个结果转换为一维的形式。由于索引值的范围为[-M+1,M-1],这里是[-1,1]，原始的相对位置索引加上M-1,使得索引值大于等于0，变为[0,2M-2]。



- 对于每行，即不同像素间，希望得到的索引位置是不同的，但是如果直接横纵坐标相加的话，往往会出现像素不同，索引相同的情况，如下所示：



所以最后将所有横坐标都乘上2M-1,最后再将横坐标和纵坐标求和，这样**每行不同像素间**得到的索引就具有独一性。

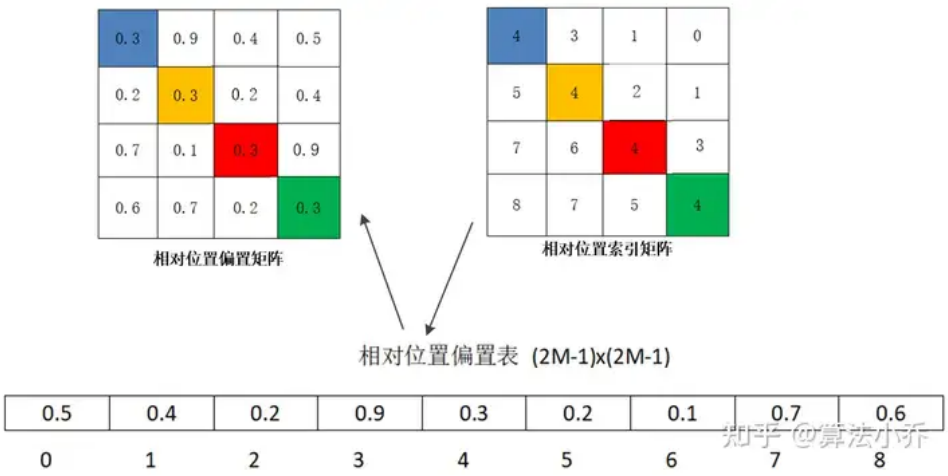


最后将行标和列标进行相加，得到独一的一维的索引，这样即保证了相对位置关系，而且不会出现上述0+1 = 1 + 0 的问题了。

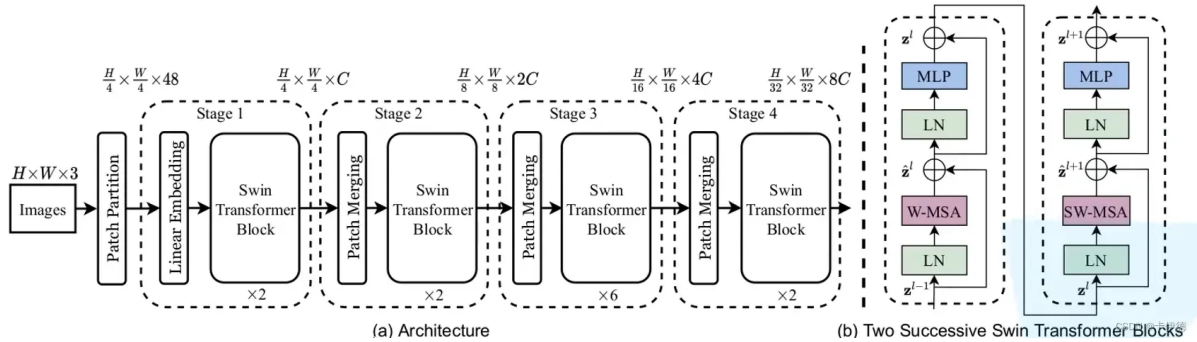
至此就计算出了相对位置的索引，其并不是公式中的位置偏置参数。

真正使用到的可训练参数使保存在**相对位置偏置表 relative position bias table**中的，这个表的size为9，因为上面矩阵中索引值为0到8 是9个数。

即`N = （2M-1）* (2M-1) = (4-1) * (4-1) = 9`，其是可训练的，随着训练过程，其内部的数值是不断优化更新的。



6 模型详细配置参数



上图是原论文中给出的关于不同Swin Transformer的配置，T(Tiny)，S(Small)，B(Base)，L(Large)，其中：

- win. sz. 7x7表示使用的窗口（Windows）的大小
- dim表示feature map的channel深度（或者说token的向量长度）
- head表示多头注意力模块中head的个数

	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.