

Topic: SwinTransformer

[12.1 Swin-Transformer 网络结构详解-哔哩哔哩] <https://b23.tv/jiZaBRH>

Swin Transformer: Hierarchical Vision Transformer using Shifted Windows

Ze Liu^{**} Yutong Lin^{**} Yue Cao^{*} Han Hu^{*‡} Yixuan Wei[†]
 Zheng Zhang Stephen Lin Baining Guo
 Microsoft Research Asia

{v-zeliul,v-yutlin,yuecao,hanhu,v-yixwe,zhez,stevelin,bainguo}@microsoft.com

2021 ICCV

ICCV 2021 best paper

val). Its performance surpasses the previous state-of-the-art by a large margin of +2.7 box AP and +2.6 mask AP on COCO, and +3.2 mIoU on ADE20K, demonstrating the potential of Transformer-based models as vision backbones. The hierarchical design and the shifted window approach also prove beneficial for all-MLP architectures. The code and models are publicly available at <https://github.com/microsoft/Swin-Transformer>.

论文地址: <https://arxiv.org/abs/2103.14030>

源码地址: <https://github.com/microsoft/Swin-Transformer>

博文地址: https://blog.csdn.net/qq_37541097/article/details/121119988

Swin Transformer

State of the Art	Object Detection on COCO test-dev (using additional training data)
State of the Art	Instance Segmentation on COCO test-dev (using additional training data)
State of the Art	Object Detection on COCO minival (using additional training data)
State of the Art	Instance Segmentation on COCO minival (using additional training data)
Ranked #8	Semantic Segmentation on ADE20K (using additional training data)
Ranked #9	Semantic Segmentation on ADE20K val
State of the Art	Action Recognition on Something-Something V2 (using additional training data)
Ranked #2	Action Classification on Kinetics-400 (using additional training data)
Ranked #2	Action Classification on Kinetics-600 (using additional training data)

(1) SwinTransformer paper

(2) MRA 微软亚洲研究院 作者 全华人

(3) 2021 ICCV , best paper

(4) SwinTransformer 有多好

} Abstract

github paper with code 链接

① Abstract } surpasses the previous SOTA

(左图)

} COCO 数据集上 } 目标检测提升 27 个点
实例分割提升 26 个点

ADE20K (图像分割数据集) 提升 3.2 个 mIoU

自己创新网络 1 个点即可发 paper 而 SwinT 不止 1 个

② 右图 github 官网 label (标签)

COCO 数据集上 4 个 SOTA

object counting

1	Soft Teacher + Swin-L (HTC++, multi-scale)	61.3					✓	End-to-End Semi-Supervised Object Detection with Soft Teacher	2021	 
2	DyHead (Swin-L, multi scale, self-training)	60.6	78.5	66.6	64.0	74.2	✓	Dynamic Head: Unifying Object Detection Heads with Attentions	2021	 
3	Dual-Swin-L (HTC, multi-scale)	60.1					✗	CBNetV2: A Composite Backbone Network Architecture for Object Detection	2021	 
4	Dual-Swin-L (HTC, single-scale)	59.4					✗	CBNetV2: A Composite Backbone Network Architecture for Object Detection	2021	

全都有 SwinTransformer 的影子

目录

- 1 网络整体框架
- 2 Patch Merging详解
- 3 W-MSA详解
 - MSA模块计算量
 - W-MSA模块计算量
- 4 SW-MSA详解
- 5 Relative Position Bias详解
- 6 模型详细配置参数

Swin Transformer

 State of the Art	Object Detection on COCO test-dev (using additional training data)
 State of the Art	Instance Segmentation on COCO test-dev (using additional training data)
 State of the Art	Object Detection on COCO minival (using additional training data)
 State of the Art	Instance Segmentation on COCO minival (using additional training data)
 Ranked #8	Semantic Segmentation on ADE20K (using additional training data)
 Ranked #9	Semantic Segmentation on ADE20K val
 State of the Art	Action Recognition on Something-Something V2 (using additional training data)
 Ranked #2	Action Classification on Kinetics-400 (using additional training data)
 Ranked #2	Action Classification on Kinetics-600 (using additional training data)

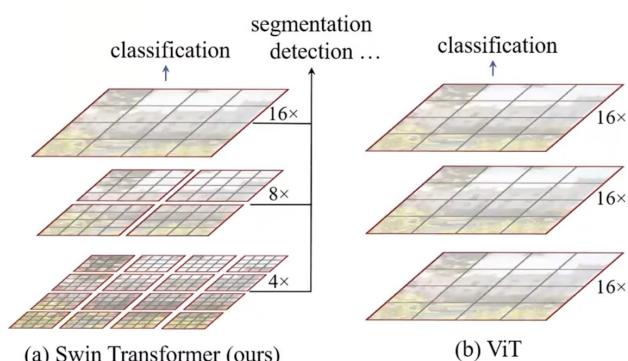
4. SW-MSA

较难理解

5. Relative Position Biases

网络整体框架

Swin Transformer与Vision Transformer对比



method	image size	#param.	FLOPs	ImageNet top-1 acc.	
				throughput (image / s)	ImageNet
RegNetY-4G [48]	224 ²	21M	4.0G	1156.7	80.0
RegNetY-8G [48]	224 ²	39M	8.0G	591.6	81.7
RegNetY-16G [48]	224 ²	84M	16.0G	334.7	82.9
EiFT-B3 [58]	300 ²	12M	1.8G	732.1	81.6
EiFT-B4 [58]	380 ²	19M	4.2G	349.4	82.9
EiFT-B5 [58]	456 ²	30M	9.9G	169.1	83.6
EiFT-B6 [58]	528 ²	43M	19.0G	96.9	84.0
EiFT-B7 [58]	600 ²	66M	37.0G	55.1	84.3
ViT-B/16 [20]	384 ²	86M	55.4G	85.9	77.9
ViT-L/16 [20]	384 ²	307M	190.7G	27.3	76.5
DeiT-T [63]	224 ²	22M	4.6G	940.4	79.8
DeiT-B [63]	224 ²	86M	17.5G	292.3	81.8
DeiT-B [63]	384 ²	86M	55.4G	85.9	83.1
Swin-T	224 ²	29M	4.5G	755.2	81.3
Swin-S	224 ²	50M	8.7G	436.9	83.0
Swin-B	224 ²	88M	15.4G	278.1	83.5
Swin-B	384 ²	88M	47.0G	84.7	84.5

method	image size	#param.	FLOPs	ImageNet top-1 acc.	
				throughput (image / s)	ImageNet
R-101x3 [38]	384 ²	388M	204.6G	-	84.4
R-152x4 [38]	384 ²	937M	840.5G	-	85.4
ViT-B/16 [20]	384 ²	86M	55.4G	85.9	84.0
ViT-L/16 [20]	384 ²	307M	190.7G	27.3	85.2
Swin-B	224 ²	88M	15.4G	278.1	85.2
Swin-B	384 ²	88M	47.0G	84.7	86.4
Swin-L	384 ²	197M	103.9G	42.1	87.3

Table 1. Comparison of different backbones on ImageNet-1K classification. Throughput is measured using the GitHub repository of [68] and a V100 GPU, following [63].

(1) SwinTransformer & VisionTransformer 对比

SwinTransformer

① 层次性
feature map 层次性
随着特征层的不断加深 size 越来越小

② 窗口分开 窗口与窗口之间不重叠

③ window 内部进行 Multi-Head self-attention 计算
window & window 之间 不进行信息的传递
好处：大大降低运算量

(2) 左图 table

前提 ImageNet-1K 上 pretrained

VIT-B/16 ImageNet 1K acc 77.9

(VIT 需要在大数据集 pretrain)

所以其 acc 甚至不如 EfficientNet V1

SwinTransformer Base 在相同 384x384 R5 下 达到准确率 84.5
相比 VITTransformer 非常大的提升

VIT-B/16 ImageNet 21K pretrained

之后在 ImageNet 1K fine-tuned

transfer learning

此时 VIT-B/16 在 ImageNet 1K acc = 84.0

Swin-B ~ acc = 86.4 lb VIT-B/16 效果好

Swin-L 达到 87.3%

网络整体框架

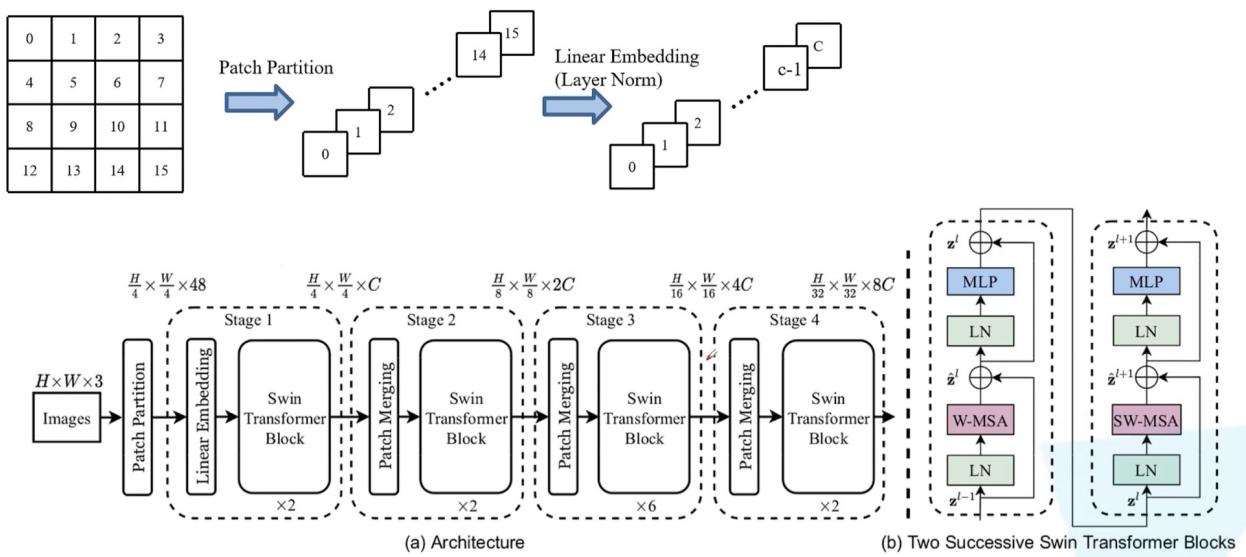
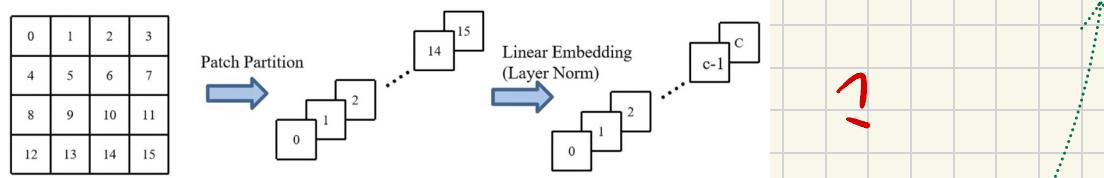


Figure 3. (a) The architecture of a Swin Transformer (Swin-T); (b) two successive Swin Transformer Blocks (notation presented with Eq. (3)). W-MSA and SW-MSA are multi-head self attention modules with regular and shifted windowing configurations, respectively.

- (1) 输入 Input Image $size = H \times W \times 3$ RGB 图片
- (2) 首先通过 patch partition 模块 经过这个模块之后
图片变成 $\frac{H}{4} \times \frac{W}{4} \times 48$ (channel 从 3 \rightarrow 48)
- (3) 接下来 依次通过 stage1、stage2、stage3、stage4
(很类似 ResNet 经过不同的 stage 对 feature map 下采样)
- (4) $\frac{H}{4} \times \frac{W}{4} \times 48 \xrightarrow{\text{stage1}} \frac{H}{4} \times \frac{W}{4} \times C \xrightarrow{\text{stage2}} \frac{H}{8} \times \frac{W}{8} \times 2C$
经过 stage2 对 stage1 输出的图片下采样了 2 倍 (图片尺寸减小)
- (5) $\xrightarrow{\text{stage2}} \frac{H}{8} \times \frac{W}{8} \times 2C \xrightarrow{\text{stage3}} \frac{H}{16} \times \frac{W}{16} \times 4C$
stage3 对 stage2 输出的图片继续下采样 2 倍
- (6) $\xrightarrow{\text{stage3}} \frac{H}{16} \times \frac{W}{16} \times 4C \xrightarrow{\text{stage4}} \frac{H}{32} \times \frac{W}{32} \times 8C$
stage4 对 stage3 的输出下采样 2 倍
- (3) ~ (6) 值得注意的是 每次下采样后 channel 是一个翻倍的操作
- (7) stage1 与 stage2、3、4 还有一点不同
stage1 是一个 Linear Embedding 层
stage2、3、4 的第一个模块是 patch merging 的结构
- (8) Patch Partition + Linear Embedding 的功能与 Patch Merging 的功能差不多的。

(9) Patch Partition 是什么?

$$me = c_1-0, c_2-0, c_3-0, c_1-1, c_2-1, c_3-1, \dots, 1$$



(1) 比如，我们输入一张图像 $4 \times 4 \times 3$ (画图省略)

(2) 通过 Patch Partition 4×4 大小的窗口对 Input Image 进行分割

(3) 分割之后 对每个窗口 channel 方向进行展平

展平以后，每个 pixel 沿深度方向拼接

(4) 每个 pixel RGB 3 通道 我们一共有 16 个值 $16 \times 3 = 48$

即 Patch Partition 之后 Image H, W 缩减为原来的 $\frac{1}{4}$ channel 变成 48

(5) 接下来通过 Linear Embedding 层

调整输入矩阵的 channel 原来的深度 48

调整之后 channel 变成 C

C 具体为多少？后面对于 Swin Transformer 的不同类型 T, S, B, L

采用的 C 有所不同

(6) 其中 Linear Embedding 层中还包含一个 Layer Norm

即调整 feature map 的 channel 之后

还对每个 channel 的 feature map 进行 Layer Norm 的处理

(7) 具体的实现过程和 ViT 是一样的

$$H \times W \times 3 \xrightarrow[k=4, s=4, p=0]{\text{Conv}} \frac{H}{4} \times \frac{W}{4} \times 48 \\ \# k = 48$$

process: Patch Partition 和 Linear Embedding 通过 1 个 conv 层即可实现

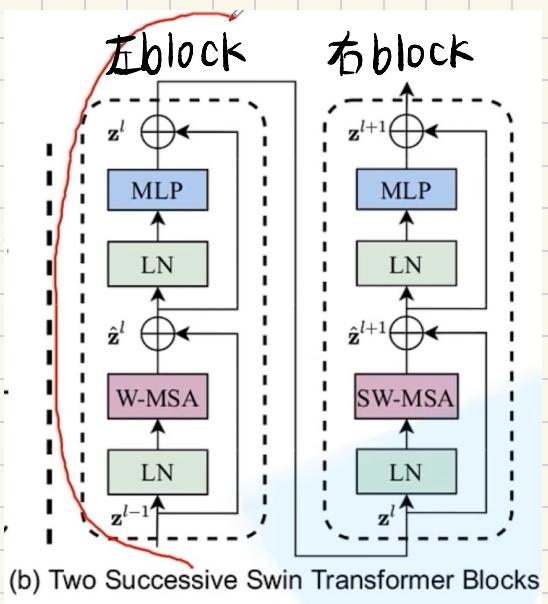
直接使用一个 kernel-size=4x4 48 个 kernel stride=4

实现 Patch Partition 和 Linear Embedding 的处理流程

后面再加上 Layer Norm

(10) 对于每个 stage 重复堆叠 SwinTransformer NVR 且都为偶数次

(11) Topic: 为什么重复的都是偶数次？



(1) 因为我们在使用堆叠 SwinTransformer，
是先使用 左 block、再使用 右 block

(2) 左 block

W-MSA

在 ViT 的这个部分是一个
Multi-Head Self-Attention 模块
在本文中是
Windows Multi-Head Self-Attention

(3) 通过 左 block 之后 下一个 block 采用的 SW-MSA 的自注意力模块
对应的名称 Shifted-Windows Multi-Head Self Attention

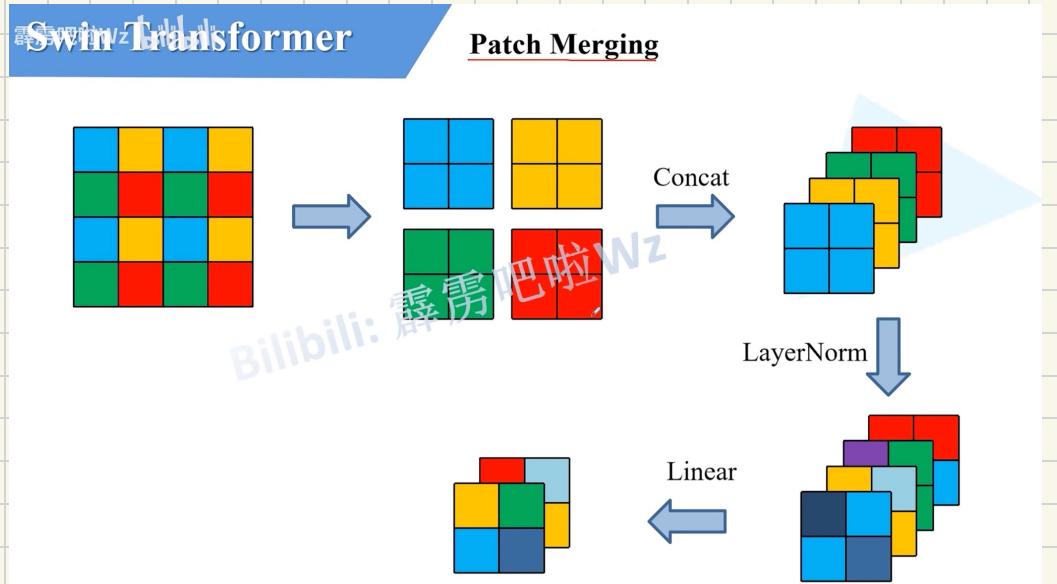
(4) 左 block & 右 block 成对使用
所以全部堆叠偶数次

(5) 关于 block 的其他结构

LN (Layer Norm)、MLP (多层感知机) 与 ViT 的对应部分是一样的
唯一不同的是 MSA 替换成了 W-MSA 和 SW-MSA

(12) Swin Transformer 以 N 作为 ViT backbone 小最后的回头看画

topic = Patch Merging



(1) Where use?

Patch Merging 是 stage 2, 3, 4 的初始模块

(2) Function?

Patch Merging 下采样

通过 Patch Merging 将 feature map 高、宽会缩减为原来的一半
channel 会翻倍

(3) How? (如何实现的 Function)

① 现假设 input image size = $4 \times 4 \times 1$ (以单通道为例)

② 以 2×2 大小为 1 个窗口 在 1 个窗口当中有 4 个像素

③ 每个窗口当中相同位置像素取出来
得到 4 个 feature map

④ 接着将 4 个 feature map 沿深度方向进行 concat

⑤ 然后在 channel 方向进行 layer norm 处理

⑥ 接着再通过一个 Linear (全连接层)

对每一个 pixel 深度方向 进行 1 对映射

图中 channel = 4 经过 Linear 之后 变成 2

此时得到的结果就是 Patch Merging 输出的 feature map

(4) 现在对比输入 & 输出

首先 特征图，高、宽缩减一半

其次 ~ 深度进行了翻倍

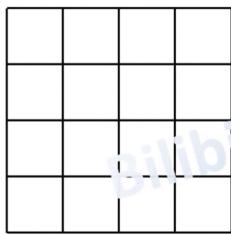
即 我们之前的 Height、Width / 2, channel * 2

topic: W-MSA

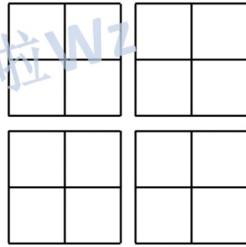
Swin Transformer

W-MSA

目的: 减少计算量
缺点: 窗口之间无法进行信息交互



Multi-head Self-Attention



Windows Multi-head Self-Attention

(1) MSA = Multi-Head Self - Attention

对于普通的MSA模块 对每一个像素求 q, k, v

对于每个pixel求得的 q 会特征图中的每一个pixel的 k 进行匹配

then进行一系列的操作

即 特征矩阵中的每一个pixel都会和特征矩阵当中所有的pixel进行信息的沟通

(2) 什么是 Windows Multi-Head Self - Attention ?

(1) 首先特征图分成一个一个 window

如图 特征窗口分成 2×2 特征图分成了4个window

(2) then 对每个window内部进行 multi-head self-attention

(3) but window & window之间没有通信

(4) 为什么这样设计?

(1) 目的: 减少计算量

(2) 缺点: 窗口之间无法进行信息交互

(4) 下面, 我们 discuss 计算量

topic: 计算量

Swin Transformer

W-MSA

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

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

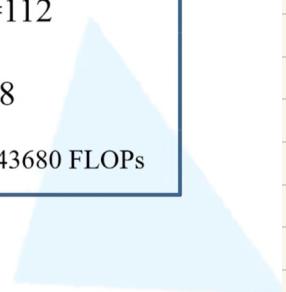
- h代表feature map的高度
- w代表feature map的宽度
- C代表feature map的深度
- M代表每个窗口(Windows)的大小

h=w=112

M=7

C=128

节省: 40124743680 FLOPs



(1) 首先明确使用 W-MSA 可以节省计算量

(2) 但具体可以省多少呢?

(3) 看式

$\Omega(\text{MSA})$

$\Omega(\text{W-MSA})$

meaning: h, w, C, M ↗

问: 式怎么来的? (next)

(4) 现设我们有输入的 feature map h, w = 112, 112

设在 W-MSA 中窗口大小 7×7 输入特征矩阵 channel = 128

将参数代入式中 相减 可以计算 = ... FLOPs

(5) explain 哪么来的?

topic = SW-MSA

SW-MSA 从哪来的？

这个公式是咋来的，原论文中并没有细讲，这里简单说下。首先回忆下单头Self-Attention的公式，如果对Self-Attention不了解的，请看下我之前写的文章。

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

MSA模块计算量

↓ 不明白看 blog

对于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(q), key(k)以及value(v)。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：

$$A^{hw \times C} \cdot W_q^{C \times C} = Q^{hw \times C}$$

- $A^{hw \times C}$ 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
- $Q^{hw \times C}$ 为所有像素通过 $W_q^{C \times C}$ 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 $hw \times C \times C$ ，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 $(hw)^2C$ ：

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

(1) 单头 self-attention 公式

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

以后，看 blog 整理 ↳ Batch Normalization

Layer Normalization ...

from: 太阳花的小绿豆

非重点 只要知道 W-MSA 的计算量即可

topic = SW-MSA

Swin Transformer

Shifted Window

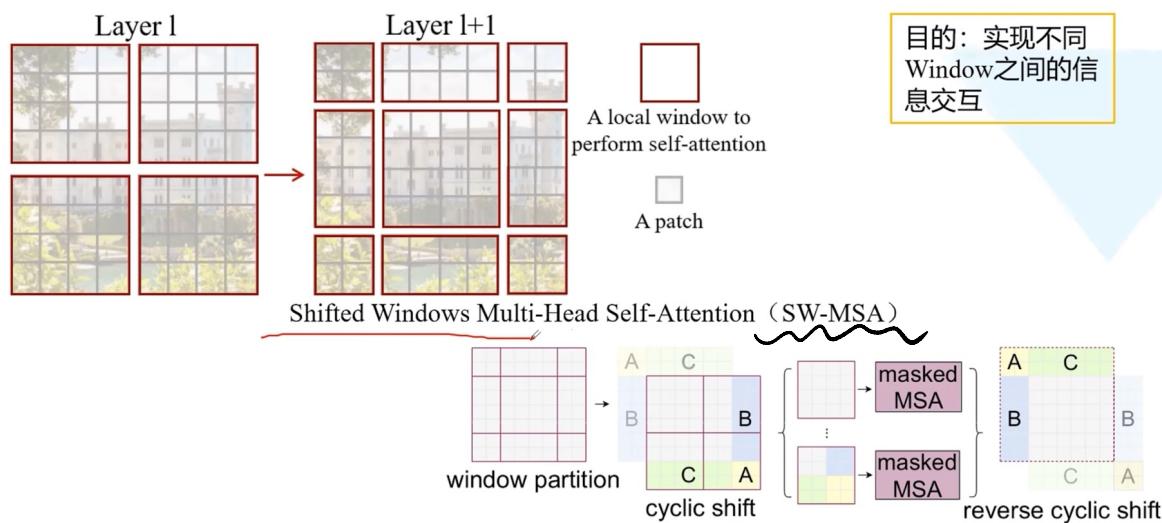


Figure 4. Illustration of an efficient batch computation approach for self-attention in shifted window partitioning.

(1) why SW-MSA?

解决 window & window 之间没有通讯的问题

so proposed: shifted Windows Multi-Head Self-Attention

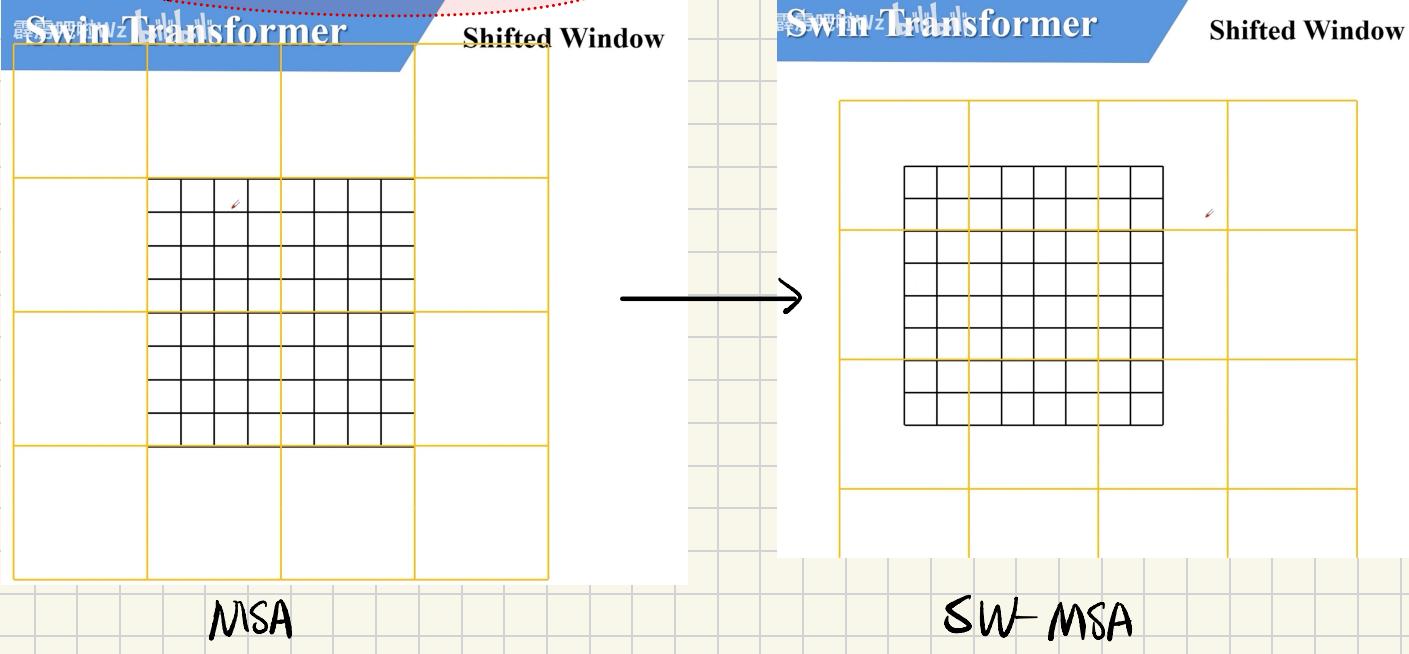
(2) 如在 l 层上采用 W-MSA Module

则在 layer l+1 上 使用 SW-MSA Module

(3) 在 SW-MSA Module 中 WINDOW 划分方式不一样

理解: 将每个 window 分别向右 & 向下移 2个 pixel (得到 layer l+1)

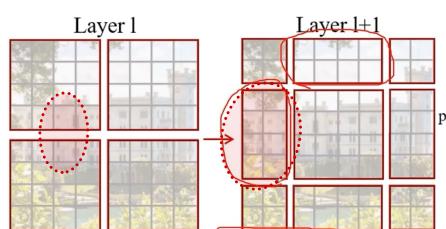
topic = 係如何理解 S ?



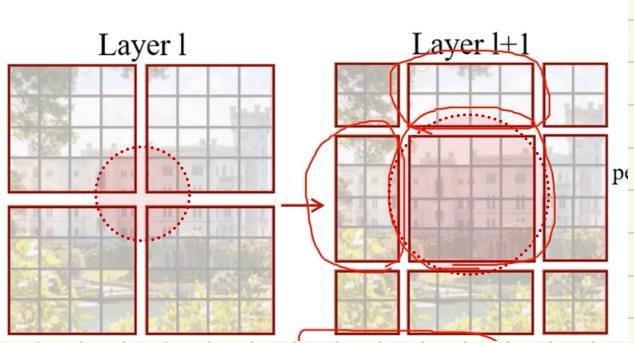
(4) topic: 怎么实现的信息交互?



对于这个 window 来说
若对其内部进行 MSA
则会融合 的信息



同理, 若计算这个 window 的 MSA
则会融合这两个 window 的信息



若对中间这个window进行 MSA
则会融入4个window的信息

这就实现了在不同 window 之间的信息交互

(5) A New question?

(1) 上一层 W-MSA 中 计算的是 4个 window

通过 shifted - window 之后 变成 9个 window

(2) 若要实现并行计算 则将 shifted window 中 8个 window 大小 $< 4 \times 4$
的填充到 $4 \times 4 N$

此时 相当于计算了 9个 window 计算量

In this way 计算量增加了

(3) Author proposed 一个更加高效计算方法

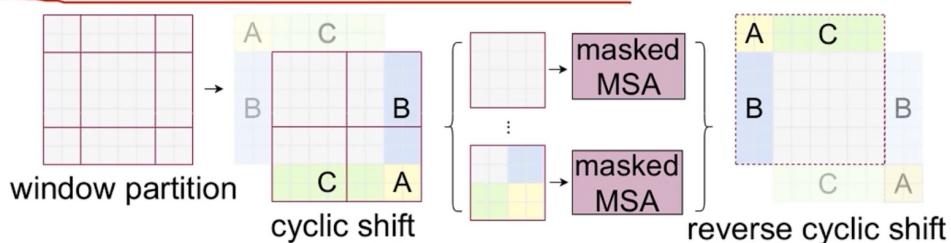
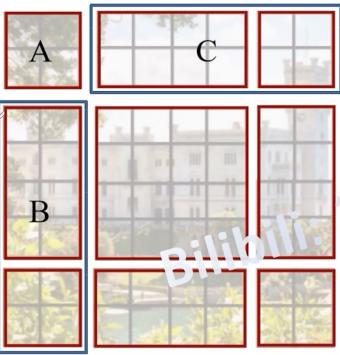


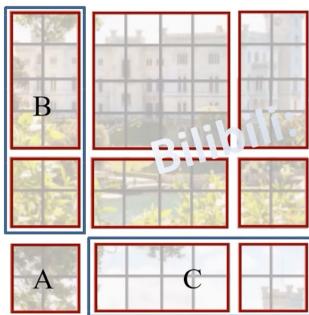
Figure 4. Illustration of an efficient batch computation approach for self-attention in shifted window partitioning.

大佬新画新解释：



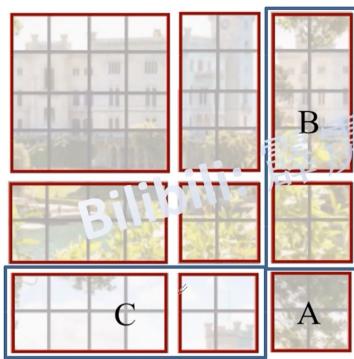
0	1	2
3	4	5
6	7	8

左上角 0号 标成 A
1号+2号 标成 C
3 & 6 标成 B



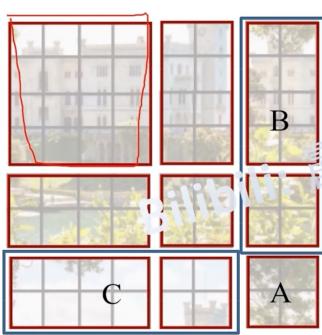
0	1	2
3	4	5
6	7	8

接下来 将 A & C 移到下面



3	4	5
6	7	8
0	1	2

再将 A & B 移到布边



4	5	3
7	8	6
1	2	0

现在可以重新划分
window 全部4x4
但要单独计算
区域5 & 区域3 MSA

$$\text{Attention}(Q, K, V) = \text{SoftMax}(QK^T / \sqrt{d} + B)V, \quad (4)$$

topic: 加盲板 分析 region 3 & regions 5

Swin Transformer

Shifted Window

$$\text{Attention}(Q, K, V) = \text{SoftMax}(QK^T / \sqrt{d} + B)V, \quad (4)$$

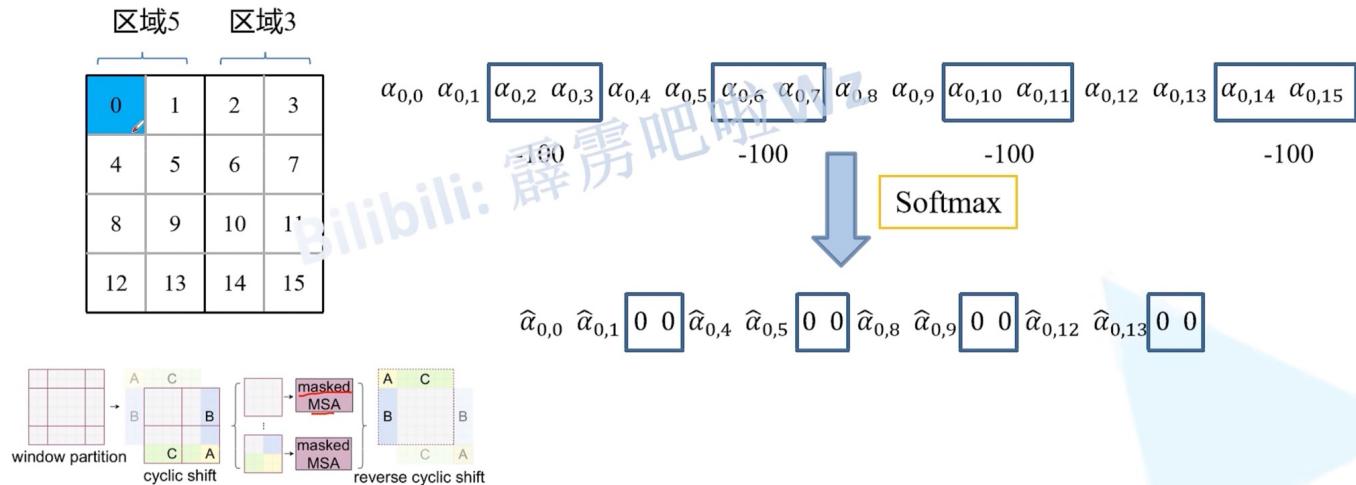


Figure 4. Illustration of an efficient batch computation approach for self-attention in shifted window partitioning.

注意，全部计算完后需要将数据挪回到原来的位置上

(1) 对于窗口中的每个 pixel 计算 q, k, v

针对每个 pixel 的 q 要和所有 pixel 的 k 进行 match 匹配

(2) 先用 0 这个位置的 pixel 求得了 q , 记成 q_0

q_0 与整个 window 中所有的 pixel 进行 match 会生成

$\alpha_{0,0} \alpha_{0,1} \alpha_{0,2} \alpha_{0,3} \dots \dots$ (对应 q_0, k 的过程)

(3) 根据 attention weight 除以 \sqrt{d} 再通过 softmax 处理

得到针对每个 pixel / token 的权重

(4) but 在计算 regions 5 内部的 MSA, 不希望引入 regions 3 的信息

$\alpha_{0,2} \alpha_{0,3} \alpha_{0,6} \alpha_{0,7} \dots \dots$ (框出来的)

将其减去 100

(因为实际计算权重都是很小的数 0, ...)

再由 softmax

再 region 3 pixel 权重 = 0

(5) 最后乘 v 加权求和

note:

(1) 最后计算量是不变的

因为加减法通常不计入计算量

这里使用 mask 减 100 且能同时计算 regions & regions 的 attention

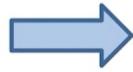
(2) 最后挪回原位

Swin Transformer

Shifted Window

$$\left\lceil \frac{M}{2} \right\rceil, \left\lceil \frac{M}{2} \right\rceil$$

1	2	3	4	5	6	7	8	9
10								
11								
12								
13								
14								
15								
16								
17								



10								
11								
12								
13								
14								
15								
16								
17								
1	2	3	4	5	6	7	8	9

topic: 举个 eg 继续讨论 SW-MSA

(1) 设上一层采用 W-MSA 模块输出特征矩阵 9×9 feature map

采用 window 大小 3×3

接下来下一层使用 SW-MSA

现在讨论：将哪几行 or 哪几列数据进行挪动？

(2) 根据论文中给的方法

首先 将 M 除以 2 然后向下取整

如这里 $M=3$ (window 大小)

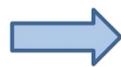
$3 \text{ 除 } 2$ 且向下取整之后等于 1

∴ 我们需要移动最上面一行以及最左边这一列

将最上面一行挪到下面，得到右图

(3) Then, 将最左边这一列 挪到右边，得到：(右图)

10									
11									
12									
13									
14									
15									
16									
17									
1	2	3	4	5	6	7	8	9	



									10
									11
									12
									13
									14
									15
									16
									17
2	3	4	5	6	7	8	9	1	

(1) 右图中，黑色分割线 对应上一层 window 对应的分割线

即时在挪动之后的 feature map

(2) 接着使用 3×3 的 window，进行分割，(下图)

1	2	3	4	5	6	7	8	9	
10									
11									
12									
13									
14									
15									
16									
17									
1	2	3	4	5	6	7	8	9	1

									10
									11
									12
									13
									14
									15
									16
									17
2	3	4	5	6	7	8	9	1	

for 橙色 window

(1) 直接进行 MSA

因为本来就是连乘的

(2) 且每个 window 都可以看成上一层中 4 个 window 的乘积

for 紫色 window

每个 window 内部信息并不连乘 $\xrightarrow{\text{so}}$ Masked MSA

关于如何实现？code！

topic: relative position bias 效果如何?

Swin Transformer

Relative position bias

$$\text{Attention}(Q, K, V) = \text{SoftMax}(QK^T / \sqrt{d} + B)V, \quad (4)$$

	ImageNet top-1 top-5		COCO AP ^{box} AP ^{mask}		ADE20k 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.	<u>80.1</u>	94.9	49.2	42.6	43.8
abs. pos.	<u>80.5</u>	95.2	<u>49.0</u>	<u>42.4</u>	<u>43.2</u>
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

Table 4. Ablation study on the *shifted windows* approach and different position embedding methods on three benchmarks, using the Swin-T architecture. w/o shifting: all self-attention modules adopt regular window partitioning, without *shifting*; abs. pos.: absolute position embedding term of ViT; rel. pos.: the default settings with an additional relative position bias term (see Eq. (4)); app.: the first scaled dot-product term in Eq. (4).

(1) 什么是 relative position bias? (相对位置 偏置?)

公式里的体现 $\text{Attention}(Q, K, V) = \text{SoftMax}(\frac{QK^T}{\sqrt{d}} + B)V$
如何来自的? (next)

Table 4 有 result & discussion

(2) case1: no pos 不使用任何位置参数和偏置

在 ImageNet top-1 上达到 80.1 的准确率

case2: abs. pos

(1) 若使用绝对位置偏置

也就是之前 ViT 中使用的 abs. pos

其 top1 acc = 80.5 相比 no pos 增加了一点 (80.1)

(2) 在 COCO 数据集 & ADE 数据集

(目标检测 task) (分割 task)

49.0 42.4 43.7 性能降低

(3) 使用绝对位置偏置 效果并不好

case3: rel. pos. 本文所采用的相对位置偏置

① 在 ImageNet1k top1 (top-one) acc = 81.3

相对 no pos (80.1) Improve 1.2 个点

② 在COCO dataset 和 ADE 20 K dataset

(no pos & abs. pos) 都有提升

49.2	42.6	43.8
49.0	42.4	43.2
50.5	43.7	46.1

③ 说明使用相对 position 偏置更加合理一些

(b) Now 看前两行

row1:

① 全部使用 W-MSA module ImageNet dataset ACC=80.2

② 若使用了 SW-MSA module 时 ImageNet 1k ACC=81.3

在COCO dataset & ADK 20 K ACC 提升都比较明显

(大于1个点)

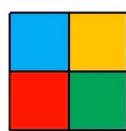
i) window & window 之间交互是十分有必要的

topic = 完成什么是 relative position bias

Swin Transformer

Relative position bias

feature map



蓝色q和所有k
匹配时相对位置索引

0, 0	0, -1
-1, 0	-1, -1

橙色q和所有k
匹配时相对位置索引

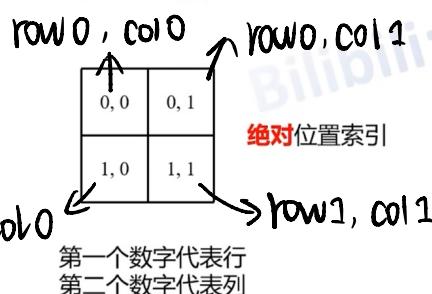
0, 1	0, 0
-1, 1	-1, 0

红色q和所有k
匹配时相对位置索引

1, 0	1, -1
0, 0	0, -1

绿色q和所有k
匹配时相对位置索引

1, 1	1, 0
0, 1	0, 0



0, 0	0, -1	-1, 0	-1, -1
0, 1	0, 0	-1, 1	-1, 0
1, 0	1, -1	0, 0	0, -1
1, 1	1, 0	0, 1	0, 0

(1) 假设 feature map 2x2

针对每个 pixel 标注 1 个 绝对位置索引

(2) 看相对位置索引

现假设 使用蓝色 pixel 对应的 q 和所有 pixel 对应的 k 进行匹配

现在以蓝色这个 pixel 为参考点

(3) 现在讨论 橙色 pixel 相对位置 index 如何 compute?

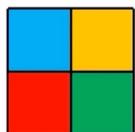
在 code 当中，用蓝色处的绝对位置索引减去 橙色绝对位置索引

依次得到 所有 pixel 的 相对位置索引

(4) 接下来 分别使用 橙色 pixel, 红色 pixel, 绿色 pixel 作为参考点

都能得到一个相对位置索引矩阵

feature map



蓝色 q 和所有 k
匹配时相对位置索引

0, 0	0, -1
-1, 0	-1, -1

橙色 q 和所有 k
匹配时相对位置索引

0, 1	0, 0
-1, 1	-1, 0

红色 q 和所有 k
匹配时相对位置索引

1, 0	1, -1
0, 0	0, -1

绿色 q 和所有 k
匹配时相对位置索引

1, 1	1, 0
0, 1	0, 0

(5) 观察

0, -1 黄色相对蓝色 (参考点)

0, -1 绿色相对红色 (参考点)

都是 0 和 -1

但都对应同一个相对位置

note

(1) 相对位置偏置 不是 相对位置索引 (是 2 个不同的概念)

(2) 我们要做的：

我们需要利用索引去 relative position bias table 中去取相应的参数

(什么是 relative position bias?) (next)

(6) 现在 将每个 相对位置索引矩阵 在行方向上展平

蓝色 q 和所有 k
匹配时相对位置索引

0, 0	0, -1
-1, 0	-1, -1

橙色 q 和所有 k
匹配时相对位置索引

0, 1	0, 0
-1, 1	-1, 0

红色 q 和所有 k
匹配时相对位置索引

1, 0	1, -1
0, 0	0, -1

绿色 q 和所有 k
匹配时相对位置索引

1, 1	1, 0
0, 1	0, 0

0, 0	0, -1	-1, 0	-1, -1
0, 1	0, 0	-1, 1	-1, 0
1, 0	1, -1	0, 0	0, -1
1, 1	1, 0	0, 1	0, 0

在原论文中作者使用的一元位置坐标 对索引对应的偏置 (how?)

① 首先，不能直接相加

因为有很多重复

$\begin{cases} (0, -1) \\ (-1, 0) \end{cases}$

是2个不同的相对 position

比如 2个 $0 + (-1) = -1$ 对应很多个不同的位置

对应的都是 $(0, -1)$ 在直接相加中对应同一个位置

② 作者(原论文) 处理：二元坐标简化成一元

0, 0	0, -1	-1, 0	-1, -1
0, 1	0, 0	-1, 1	-1, 0
1, 0	1, -1	0, 0	0, -1
1, 1	1, 0	0, 1	0, 0

偏移从0开始，
行、列标加上M-1

1, 1	1, 0	0, 1	0, 0
1, 2	1, 1	0, 2	0, 1
2, 1	2, 0	1, 1	1, 0
2, 2	2, 1	1, 2	1, 1

☆ $(0, -1)$ $(-1, 0)$ 是不同的位置 相加的行都减1 变成同一个位置

① 首先 偏移从0开始 行标、列标加上M-1

M、窗口大小 $M=2$ $M-1=1$

∴ 所有行标、列标加上1 得到矩阵

② 接下来 行标 $\times 2M-1$ $M=2$

$\therefore 2M-1=3$

那第1个元素 $\times 3$

1, 1	1, 0	0, 1	0, 0
1, 2	1, 1	0, 2	0, 1
2, 1	2, 0	1, 1	1, 0
2, 2	2, 1	1, 2	1, 1

行标乘上 $2M-1$

3, 1	3, 0	0, 1	0, 0
3, 2	3, 1	0, 2	0, 1
6, 1	6, 0	3, 1	3, 0
6, 2	6, 1	3, 2	3, 1

③ finally 行标、列标相加 得到一元原矩阵

3, 1	3, 0	0, 1	0, 0
3, 2	3, 1	0, 2	0, 1
6, 1	6, 0	3, 1	3, 0
6, 2	6, 1	3, 2	3, 1

行、列标相加

4	3	1	0
5	4	2	1
7	6	4	3
8	7	5	4

观察得到的一元相对位置索引矩阵

我们发现 之前对应相同索引的位置依旧相同 (-1,0)

对应 (0,-1) & (-1,0) 索引就不同了 变成 3,1

因此二元坐标 变成了一元坐标) \Rightarrow 得到了

relative position index

topic= 在得到了 relative position index 之后, 看 relative position bias.

Swin Transformer

Relative position bias

relative position bias table

$(2M-1) \times (2M-1)$

0.1	0.2	0.3	0.8	0.1	0.6	0.4	0.4	0.7
0	1	2	3	4	5	6	7	8

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

4	3	1	0
5	4	2	1
7	6	4	3
8	7	5	4

relative position index

0.1	0.8	0.2	0.1
0.6	0.1	0.3	0.2
0.4	0.4	0.1	0.8
0.7	0.4	0.6	0.1

relative position bias

(1) 我们根据 relative position index (索引) 在 relative position table 中取参数

(2) relative position bias table # elements (元素个数) = $(2M-1) \times (2M-1)$ (Why? next)

(3) 举例, 第 1 个 position 对应 index=4

在 relative position bias table 索引 4 的位置对应 0.1

Index=3 就取 3 所在的位置 0.8

最终得到 relative position bias = B

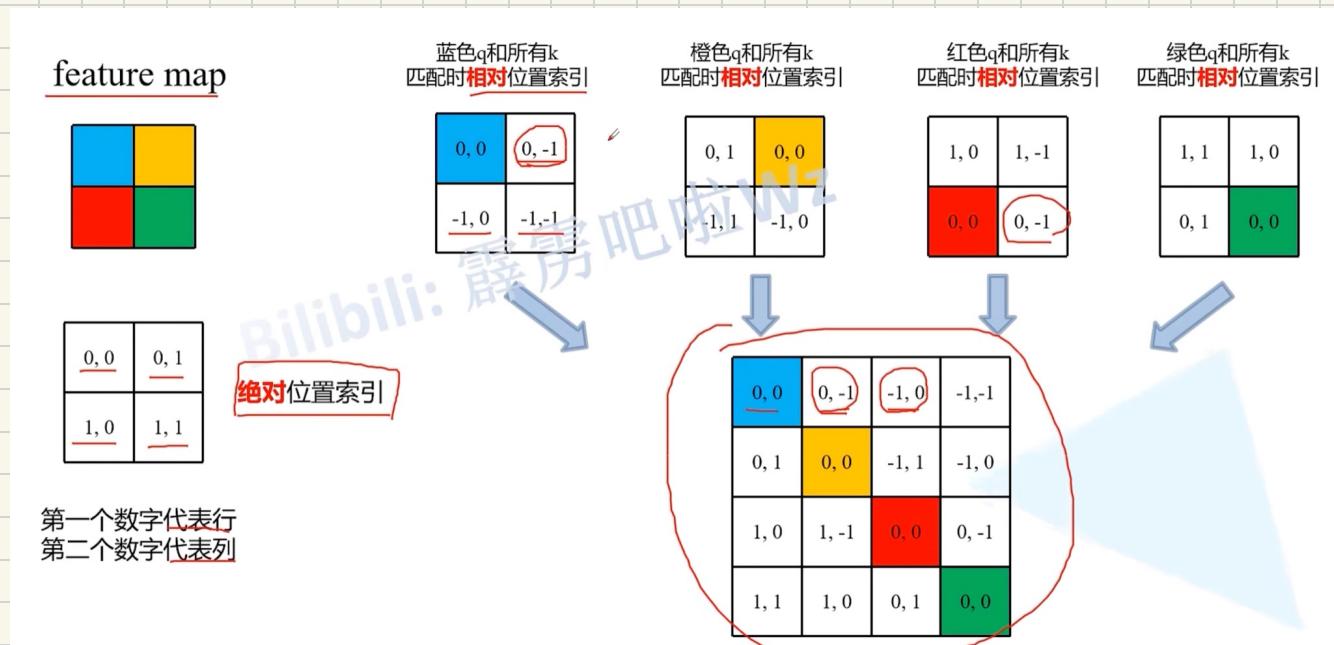
即在我们训练网络过程中训练的是 relative position bias table 中的参数 而 relative position index 是 fixed (固定的)

141 即我们的窗口大小固定

则 relative position index 固定 由观察可知 $0 \sim 8$ 共有 9 个数
且 relative position bias table 有 9 个数 = $(2M-1)(2M-1)$

151 Now tell why? $(2M-1)^2$ relative position bias table

从这儿开始：



首先看极端位置

① 左上角蓝色位置 以其为参考点 所能取到极端值 $(-1, -1)$

② 右下角绿色位置 pixel ~ ~ $(1, 1)$

⇒ 索引范围为 $[-M+1, M-1]$

□ 这里 $M=2$ $\therefore [-1, 1] \rightarrow [0, 1]$ 3 个数

行索引 3 种可能 列索引也 3 种可能

排列组合之后 总共 9 种可能

□ 设窗口 = M 则行索引所能取到的数值 $2M-1$ 个

即从 $-M+1 \sim M-1$ 共 $2M-1$ 个数 $(M-1+M-1+1) = 2M-1$

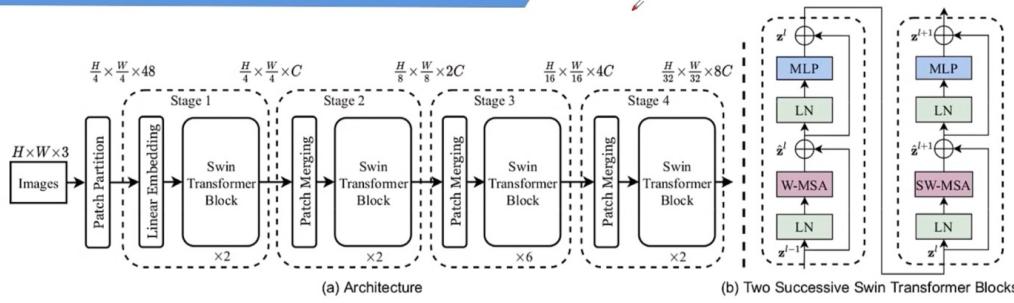
同样对列而言也是 $2M-1$ 个数

∴ $(2M-1)^2$

即 relative position bias table # = $(2M-1)^2$

Swin Transformer

模型详细配置参数



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

Table 7. Detailed architecture specifications.

topic = Swin transformer 模型详细配置参数

(1) (a) 是模型 architecture

table 7 是详细 T, S, B, L 参数

Tiny Small Base Large

(2) 在 Swin-T 讲解

① 首先 对于输入 经过 concat 4×4 , 96-d, LN

(对应于 Architecture 中的 patch partition & Linear Embedding)

patch partition & Linear Embedding 和 patch merging一样
都艮对我们的输入特征矩阵进行下采样 以及 调整特征矩阵
的 channel , 再通过 Layer Normal 进行输出

② concat 4×4 , 96-d, LN

4×4 = 把 Input Image 下采样 4 倍

96 : 通过 Linear Embedding 将 特征矩阵 channel 变成 96

LN = Layer Norm

stage 1	$4 \times$ (56×56)	concat 4×4 , 96-d, LN
		win. sz. 7×7 , dim 96, head 3 $\times 2$

③ 堆叠两个Swin Transformer Block

$\begin{bmatrix} \text{win. sz. } 7 \times 7 \\ \text{dim } 96, \text{ heads } 3 \end{bmatrix} \times 2$

meaning $\text{win. sz.} = \text{window size} = 7 \times 7$

$\text{dim } 96 = \text{通过 SwinTransformer Block 后输出}$
 $\text{feature map channel} = 96$

$\text{head } 3 = \text{在采用的多头注意力机制中 head} = 3$

④ 对于 stage 2

concat 2×2 192-d LN

stage 2	$8 \times (28 \times 28)$	concat 2×2 , 192-d, LN
		win. sz. 7×7 , dim 192, head 6 $\times 2$

通过 patch merging 下采样 高 & 宽 中下采样 2倍

channel 调整到 192 同样堆叠 2个 SwinTransformer Block

$\begin{bmatrix} \text{win. sz. } 7 \times 7 \\ \text{dim } 192, \text{ head } 6 \end{bmatrix}$

window size = 7×7 channel = 192 head = 6 (MSA Head)

⑤ 对于 stage 3

stage 3	$16 \times (14 \times 14)$	concat 2×2 , 384-d, LN
		win. sz. 7×7 , dim 384, head 12 $\times 6$

○ 首先通过 patch merging 调整高 & 宽 下采样 2倍

channel 调整成 384 通过 Layer Norm

○ 重复堆叠 6个 SwinTransformer Block

window size = 7×7 dim = 384 head = 12

⑥ 对于 stage 4

stage 4	$32 \times (7 \times 7)$	concat 2×2 , 768-d, LN
		win. sz. 7×7 , dim 768, head 24 $\times 2$

● 首先通过 patch merging 将 H & W 下采样 2倍

channel 调整成 768 通过 LN

● 堆叠 2个 SwinTransformer Block window size = 7×7

dimension = 768 head = 24

以上 Swin-T 详细参数

NOTE: SWIN Transformer Block 成对使用

一个是 W-MSA 一个 是 SW-MSA