Swin Transformer

Hierarchical Vision Transformer using Shifted Windows

2021

code:https://github.com/microsoft/Swin-Transformer

论文出发点或背景:

CNN在视觉任务上取得了非常好的效果，transformer模型在语言处理的巨大成功引起了研究人员对于transformer在视觉任务中使用的探索。

与作为语言transformer中处理基本元素的单词标记不同，视觉元素在规模上可能会有很大差异，这个问题在对象检测等任务中受到关注

与文本段落中的单词相比，图像中像素的分辨率要高得多。存在许多视觉任务，例如语义分割，需要在像素级别进行密集预测，这对于高分辨率图像上Transformer 来说是难以处理的，因为其自注意力的计算复杂度与图像大小成二次方。

之前的相关工作:

1.CNN与其变体:

VGG，GoogleNet，ResNet，DenseNet，HRNet，and EfficientNet

2.基于自注意力的backbone

曾经有人尝试将resnet中的卷积层替换为自注意力，虽然得到了略微优于resnet的结果，但是在延迟上远大于卷积网络

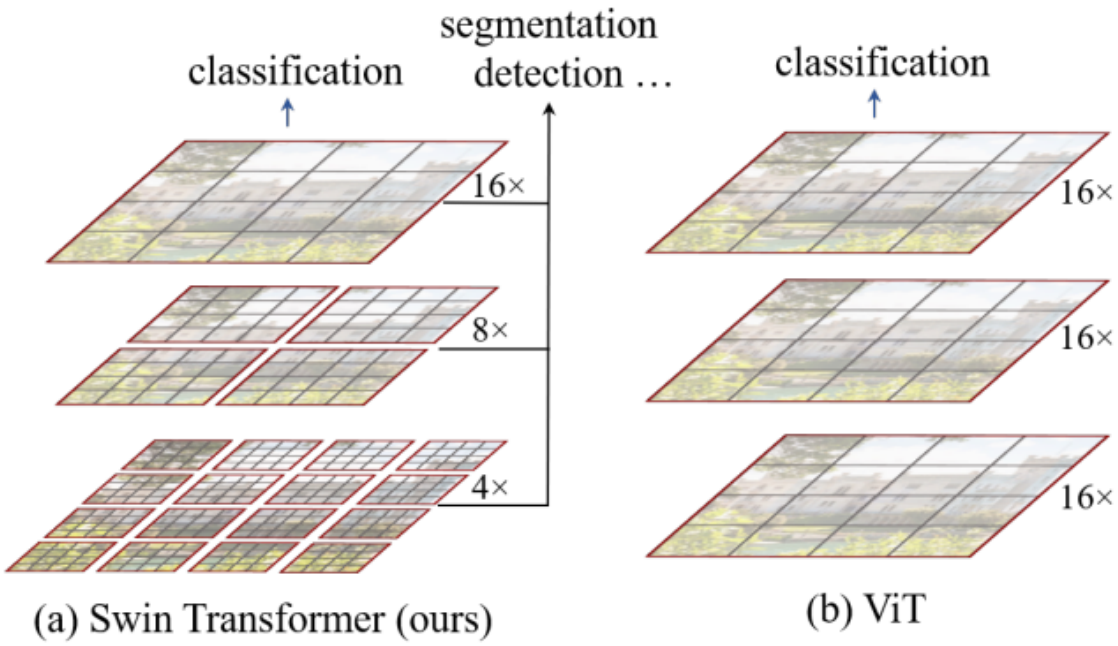
3.自注意力/transformer用以补充CNN

自注意力层可以通过提供编码远程依赖或异构交互的能力来补充主干 或头部网络 。最近，Transformer 中的编码器-解码器设计已应用于对象检测和实例分割任务

ViT 在图像分类上的结果令人鼓舞，但其架构不适合用作密集视觉任务或输入图像分辨率高时的通用骨干网络，因为它的低分辨率特征图和二次增加图像大小的复杂性。**有一些作品通过直接上采样或反卷积将 ViT 模型应用于目标检测和语义分割的密集视觉任务，但性能相对较低**

论文创新思路：

滑窗法通过限制非重叠局部窗口的自注意计算，同时还允许跨窗口连接，从而提高了效率



swin trans的自注意力模块复杂度呈线性，vit的呈二次复杂度

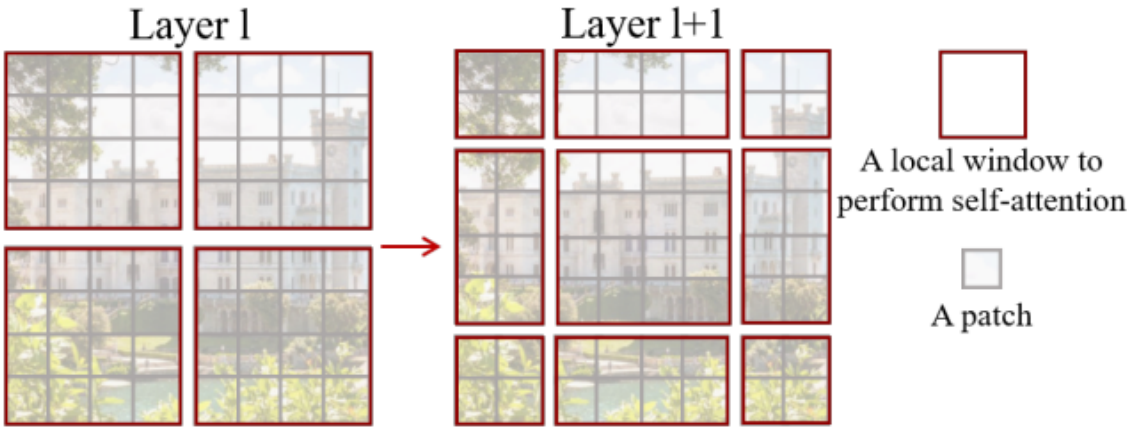
全局计算会导致与标记数量相关的二次复杂性，使其不适用于许多需要大量标记进行密集预测或表示高分辨率图像的视觉问题。

基于窗口的自注意力模块缺少跨窗口的连接这限制了它的模型能力。

论文方法介绍：

Swin Transformer 通过从小尺寸的补丁（灰色轮廓）开始并逐渐合并更深的 Transformer 层中的相邻补丁来构建分层表示。通过这些分层特征图，Swin Transformer 模型可以方便地利用先进的技术进行密集预测

Swin Transformer 的一个关键设计元素是它在连续自注意力层之间的窗口分区的移动



它首先通过补丁拆分模块（如 ViT）将输入的 RGB 图像拆分为不重叠的补丁。每个补丁都被视为一个“令牌”，其特征被设置为原始像素 RGB 值的串联。在我们的实现中，我们使用 4 × 4 的补丁大小，因此每个补丁的特征维度是 4 × 4 × 3 = 48。线性嵌入层应用于这个原始值特征以将其投影到任意维度（ 表示为 C)

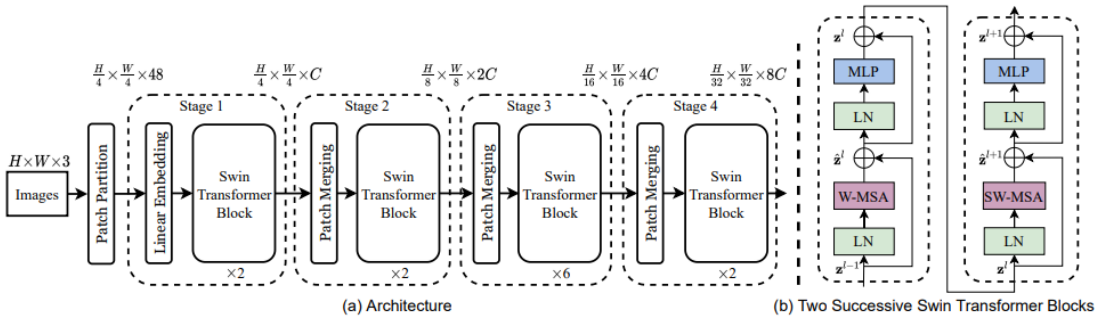


Figure 3. (a) The architecture of a Swin Transformer (Swin-T); (b) two successive Swin Transformer Blocks (notation presented with Eq. (3)). W-MSA and SW-MSA are multi-head self attention modules with regular and shifted windowing configurations, respectively.

为了产生分层表示，随着网络变得更深，通过补丁合并层来减少令牌的数量。第一个补丁合并层连接每组 2 × 2 相邻补丁的特征，并在 4C 维连接特征上应用线性层。这将令牌数量减少了 2×2 = 4 的倍数（分辨率的 2 倍下采样），并且输出维度设置为 2C。之后应用 Swin Transformer 块进行特征转换，分辨率保持在 。补丁合并和特

征转换的第一个块表示为“第 2 阶段”。该过程重复两次，分别为“第 3 阶段”和“第 4 阶段”，输出分辨率分别为$\frac{H}{16} \times \frac{W}{16}$ 和 $\frac{H}{32} \times \frac{W}{32}$。这些阶段共同产生一个分层表示，具有与典型卷积网络相同的特征图分辨率

Swin Transformer 模块由一个基于移动窗口的 MSA 模块组成，后跟一个 2 层 MLP，其间具有 GELU 非线性激活函数。在每个 MSA 模块和每个 MLP 之前应用一个 LayerNorm (LN) 层，在每个模块之后应用一个残差连接。

在非重叠窗口内的自注意力 为了高效地建模，我们提出在局部窗口内计算自注意力。窗口的设置是用非重叠的方式均匀的分割图片。假设每个窗口包含 M × M个图像块。一个全局的MSA模块和一个基于 h × w个图像块的图片的窗口的计算复杂度分别是

$$
\begin{aligned}
&\Omega(\text{ MSA }) = 4hwC^2 + 2(hw)^2C \\
&\Omega(\text{ W-MSA }) = 4hwC^2 + 2M^2hwC
\end{aligned}
$$

在连续的块中移动窗口 基于窗口的自注意力模块缺少跨窗口的连接这限制了它的模型能力。为了引入跨窗口连接同时保持对非重叠窗口的高效计算，我们提出了一个移动窗口分割方法，它能在连续的 Swin Transformer块中交替的使用两个分割配置。在移动窗口分割方法下，连续的 Swin Transformer块计算方法如下：

$$
\begin{aligned}
\hat{z}^1 &= \mathrm{WMSA}\left(\mathrm{LN}\left(\mathrm{z}^{1-1}\right)\right) + \mathrm{z}^{1-1} \\
\mathrm{z}^1 &= \mathrm{MLP}\left(\mathrm{LN}\left(\hat{z}^1\right)\right) + \hat{z}^1 \\
\hat{\mathrm{z}}^{1+1} &= \mathrm{SWMSA}\left(\mathrm{LN}\left(\mathrm{z}^1\right)\right) + \mathrm{z}^1 \\
\mathrm{z}^{1+1} &= \mathrm{MLP}\left(\mathrm{LN}\left(\hat{\mathrm{z}}^{1+1}\right)\right) + \hat{\mathrm{z}}^{1+1}
\end{aligned}
$$

移动窗口分割方法将上一层中的相邻的非重叠窗口连接起来，并且发现这种方法在图片分类，物体检测，语义分割中是有效的。

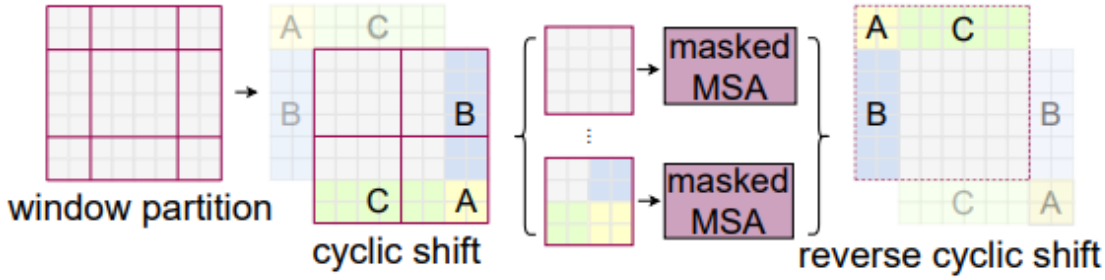


Figure 4. Illustration of an efficient batch computation approach for self-attention in shifted window partitioning.

沿着左上方向循环移动。如上图所述。移动之后，一个批量窗口由在特征图中不相连的几个子窗口组成，屏蔽机制是用来限制在每个子窗口中的自注意力计算。随着循环移动，批量窗口的数量保持和规则的窗口分割的数量一样。因此也是高效的。

相对位置偏置：

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

进一步的添加绝对位置嵌入到输入使得性能略降。因此在我们的实现中没有采用这种方法。

实际效果：

(a) Regular ImageNet-1K trained models					
method	image size	#param.	FLOPs	throughput (image / s)	ImageNet top-1 acc.
RegNetY-4G [44]	224^2	21M	4.0G	1156.7	80.0
RegNetY-8G [44]	224^2	39M	8.0G	591.6	81.7
RegNetY-16G [44]	224^2	84M	16.0G	334.7	82.9
ViT-B/16 [19]	384^2	86M	55.4G	85.9	77.9
ViT-L/16 [19]	384^2	307M	190.7G	27.3	76.5
DeiT-S [57]	224^2	22M	4.6G	940.4	79.8
DeiT-B [57]	224^2	86M	17.5G	292.3	81.8
DeiT-B [57]	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
(b) ImageNet-22K pre-trained models					
method	image size	#param.	FLOPs	throughput (image / s)	ImageNet top-1 acc.
R-101x3 [34]	384^2	388M	204.6G	-	84.4
R-152x4 [34]	480^2	937M	840.5G	-	85.4
ViT-B/16 [19]	384^2	86M	55.4G	85.9	84.0
ViT-L/16 [19]	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 [62] and a V100 GPU, following [57].

(a) Various frameworks

Method	Backbone	AP^{box}	AP^{box}_{50}	AP^{box}_{75}	#param.	FLOPs	FPS
Cascade Mask R-CNN	R-50	46.3	64.3	50.5	82M	739G	18.0
	Swin-T	**50.5**	**69.3**	**54.9**	86M	745G	15.3
ATSS	R-50	43.5	61.9	47.0	32M	205G	28.3
	Swin-T	**47.2**	**66.5**	**51.3**	36M	215G	22.3
RepPointsV2	R-50	46.5	64.6	50.3	42M	274G	13.6
	Swin-T	**50.0**	**68.5**	**54.2**	45M	283G	12.0
Sparse R-CNN	R-50	44.5	63.4	48.2	106M	166G	21.0
	Swin-T	**47.9**	**67.3**	**52.3**	110M	172G	18.4

(b) Various backbones w. Cascade Mask R-CNN

	AP^{box}	AP^{box}_{50}	AP^{box}_{75}	AP^{mask}	AP^{mask}_{50}	AP^{mask}_{75}	param	FLOPs	FPS
DeiT-S†	48.0	67.2	51.7	41.4	64.2	44.3	80M	889G	10.4
R50	46.3	64.3	50.5	40.1	61.7	43.4	82M	739G	18.0
Swin-T	**50.5**	**69.3**	**54.9**	**43.7**	**66.6**	**47.1**	86M	745G	15.3
X101-32	48.1	66.5	52.4	41.6	63.9	45.2	101M	819G	12.8
Swin-S	**51.8**	**70.4**	**56.3**	**44.7**	**67.9**	**48.5**	107M	838G	12.0
X101-64	48.3	66.4	52.3	41.7	64.0	45.1	140M	972G	10.4
Swin-B	**51.9**	**70.9**	**56.5**	**45.0**	**68.4**	**48.7**	145M	982G	11.6

(c) System-level Comparison

Method	mini-val		test-dev		#param.	FLOPs
	AP^{box}	AP^{mask}	AP^{box}	AP^{mask}		
RepPointsV2* [12]	-	-	52.1	-	-	-
GCNet* [7]	51.8	44.7	52.3	45.4	-	1041G
RelationNet++* [13]	-	-	52.7	-	-	-
DetectoRS* [42]	-	-	55.7	48.5	-	-
YOLOv4 P7* [4]	-	-	55.8	-	-	-
Copy-paste [23]	55.9	47.2	56.0	47.4	185M	1440G
X101-64 (HTC++)	52.3	46.0	-	-	155M	1033G
Swin-B (HTC++)	56.4	49.1	-	-	160M	1043G
Swin-L (HTC++)	57.1	49.5	57.7	50.2	284M	1470G
Swin-L (HTC++)*	**58.0**	**50.4**	**58.7**	**51.1**	284M	-

Table 2. Results on COCO object detection and instance segmentation. †denotes that additional decovolution layers are used to produce hierarchical feature maps. * indicates multi-scale testing.

	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**

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).

method	MSA in a stage (ms)				Arch. (FPS)		
	S1	S2	S3	S4	T	S	B
sliding window (naive)	122.5	38.3	12.1	7.6	183	109	77
sliding window (kernel)	7.6	4.7	2.7	1.8	488	283	187
Performer [14]	4.8	2.8	1.8	1.5	638	370	241
window (w/o shifting)	2.8	1.7	1.2	0.9	770	444	280
shifted window (padding)	3.3	2.3	1.9	2.2	670	371	236
shifted window (cyclic)	3.0	1.9	1.3	1.0	755	437	278

Table 5. Real speed of different self-attention computation methods and implementations on a V100 GPU.

	Backbone	ImageNet		COCO		ADE20k
		top-1	top-5	AP^{box}	AP^{mask}	mIoU
sliding window	Swin-T	81.4	95.6	50.2	43.5	45.8
Performer [14]	Swin-T	79.0	94.2	-	-	-
shifted window	Swin-T	81.3	95.6	50.5	43.7	46.1

Table 6. Accuracy of Swin Transformer using different methods for self-attention computation on three benchmarks.

个人理解

Swin transformer针对ViT中的一些问题做出了改进：

1.SA与输入个数呈二次复杂度

2.ViT在分类任务上表现较好，但是别的下游任务上表现仍不如CNN

PVT中指出ViT直筒型结构不适用于像素级任务的原因主要有以下两点：

（1）其输出分辨率比较低，且只有一个单一尺度，输出步幅为32或者16；

（2）输入尺寸即使增大一点，也会造成计算复杂度和内存消耗的大幅增加。

针对上述问题：Swin trans分别通过：

1：非重叠窗口内的自注意力

2：采用类似CNN中的逐渐收缩的金字塔结构随着网络深度增加，逐渐减小Transformer的序列长度，显著降低了计算量。这与PVT(Pyramid Vision Transformer)的思想是一致的

进行解决

其中为了引入跨窗口连接同时保持对非重叠窗口的高效计算，又提出了一个移动窗口分割方法

感觉整篇论文对于方法产生的思路比较完善，但是对于方法的具体实现细节还是不足。