Swin transformer

Swin Transformer 原理分析

Swin Transformer 提出了一种针对视觉任务的通用的 Transformer 架构，Transformer 架构在 NLP 任务中已经算得上一种通用的架构，但是如果想迁移到视觉任务中有一个比较大的困难就是处理数据的尺寸不一样。作者分析表明，Transformer 从 NLP 迁移到 CV 上没有大放异彩主要有两点原因：

1. 最主要的原因是两个领域涉及的scale不同，NLP 任务以 token 为单位，scale 是标准固定的，而 CV 中基本元素的 scale 变化范围非常大
2. CV 比起 NLP 需要更大的分辨率，而且 CV 中使用 Transformer 的计算复杂度是图像尺度的平方，这会导致计算量过于庞大， 例如语义分割，需要像素级的密集预测，这对于高分辨率图像上的Transformer来说是难以处理的

Swin Transformer 就是为了解决这两个问题所提出的一种通用的视觉架构。Swin Transformer 引入 CNN 中常用的层次化构建方式

Swin Transformer 具体步骤

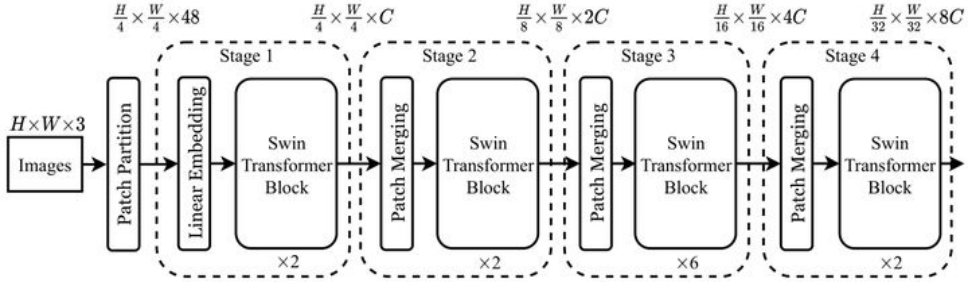
1 图片预处理：分块和降维 (Patch Partition)

首先把一张图片看作是一系列的展平的2D块的序列

这个序列中一共有 $N = HW / P^2$ 个展平的2D块，其中P是块大小

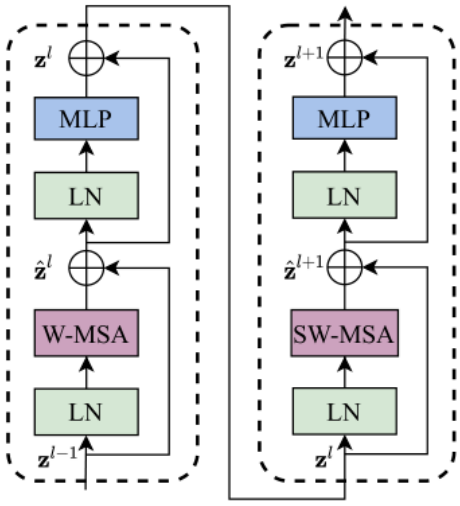
2 Stage 1：线性变换 (Linear Embedding)

假设现在得到的向量维度是：H/4 × W/4 × 48，需要做一步叫做Linear Embedding的步骤，对每个向量都做一个线性变换（即全连接层），变换后的维度为C，这里我们称其为 Linear Embedding。这一步之后得到的张量维度是：H/4 × W/4 × C，如下图



3 Stage 1：Swin Transformer Block

接下来H/4 × W/4 × C这个张量进入2个连续的 Swin Transformer Block 中，这被称作 Stage 1，在整个的 Stage 1 里面 token 的数量一直维持H/4 × W/4不变



Swin Transformer Block 的结构如上图2所示。上图是2个连续的 Swin Transformer Block。其中一个 Swin Transformer Block 由一个带两层 MLP 的 Shifted Window-based MSA 组成，另一个 Swin Transformer Block 由一个带两层 MLP 的 Window-based MSA 组成。在每个 MSA 模块和每个 MLP 之前使用 LayerNorm(LN) 层，并在每个 MSA 和 MLP之后使用残差连接

可以看到 Swin Transformer Block 和 ViT Block 的区别就在于将 ViT 的多头注意力机制 MSA 替换为了 Shifted Window-based MSA 和 Window-based MSA

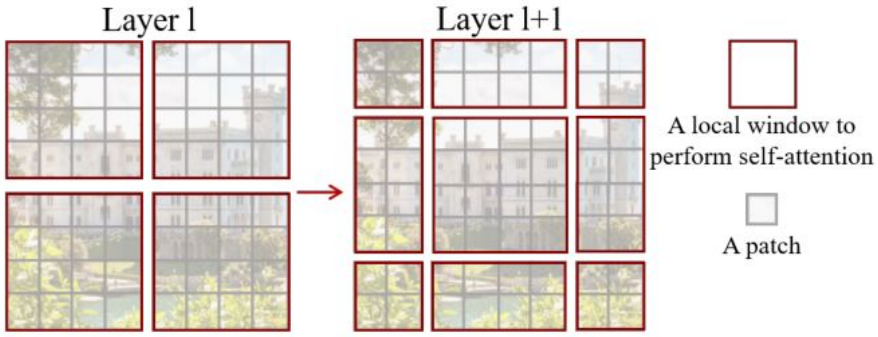
4 Stage 1：Swin Transformer Block：Window-based MSA

标准 ViT 的多头注意力机制 MSA 采用的是全局自注意力机制，即：计算每个 token 和所有其他 token 的 attention map

5 Stage 1：Swin Transformer Block：Shifted Window-based MSA

Window-based MSA 虽然大幅节约了计算量，但是牺牲了 windows 之间关系的建模，不重合的 Window 之间缺乏信息交流影响了模型的表征能力。Shifted Window-based MSA 就是为了解决这个问题，如下图3所示。在两个连续的Swin Transformer Block中交替使用W-MSA 和 SW-MSA。以上图为例，将前一层 Swin Transformer Block 的 8x8 尺寸feature map划分成 2x2 个patch，每个 patch 尺寸为 4x4，然后将下一层 Swin Transformer Block 的 Window 位置进行移动，得到 3x3 个不重合的 patch。移动 window 的划分方式使上一层相邻的不重合 window 之间引入连接，大大的增加了感受野

这样一来，在新的 window 里面做 self-attention 操作，就可以包括原有的 windows 的边界，实现 windows 之间关系的建模



6 Stage 2/3/4

从 Stage 2 到 Stage 4 的每个 stage 的初始阶段都会先做一步 Patch Merging 操作，Patch Merging 操作的目的是为了减少 tokens 的数量，它会把相邻的 2×2 个 tokens 给合并到一起，得到的 token 的维度是 4C 。Patch Merging 操作再通过一次线性变换把维度降为2C。至此，维度是H/4 × W/4 × C的张量经过Patch Merging 操作变成了维度是H/8 × W/8 × 2C的张量

同理，后面的每个 Stage 都会改变张量的维度，形成一种层次化的表征，最后变成H/32 × W/32 × 8C。因此，这种层次化的表征可以方便地替换为各种视觉任务的骨干网络

Swin Transformer 的结构

Swin Transformer 分为 Swin-T，Swin-S，Swin-B，Swin-L 这四种结构。使用的 window 的大小统一为 M=7，每个 head 的embedding dimension 都是 32，每个 stage 的层数如下：

Swin-T： C=96，layer number：{2, 2, 6, 2}
Swin-S： C=96，layer number： {2, 2, 18, 2}
Swin-B： C=128，layer number：{2, 2, 18, 2}
Swin-L： C=192，layer number：{2, 2, 18, 2}

实验及评估结果

1 图像分类

数据集：ImageNet

(a)表是直接在 ImageNet-1k 上训练，(b)表是先在 ImageNet-22k 上预训练，再在 ImageNet-1k 上微调

对标 88M 参数的 DeiT-B 模型，它在 ImageNet-1k 上训练的结果是83.1% Top1 Accuracy，Swin-B 模型的参数是80M，它在 ImageNet-1k 上训练的结果是83.5% Top1 Accuracy，优于 DeiT-B 模型

图像分类上比 ViT、DeiT等 Transformer 类型的网络效果更好，但是比不过 CNN 类型的 EfficientNet，猜测 Swin Transformer 还是更加适用于更加复杂、尺度变化更多的任务

(a) Regular ImageNet-1K trained models					
method	image size	#param.	FLOPs	throughput (image / s)	ImageNet top-1 acc.
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
EffNet-B3 [58]	300^2	12M	1.8G	732.1	81.6
EffNet-B4 [58]	380^2	19M	4.2G	349.4	82.9
EffNet-B5 [58]	456^2	30M	9.9G	169.1	83.6
EffNet-B6 [58]	528^2	43M	19.0G	96.9	84.0
EffNet-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-S [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
(b) ImageNet-22K pre-trained models					
method	image size	#param.	FLOPs	throughput (image / s)	ImageNet top-1 acc.
R-101x3 [38]	384^2	388M	204.6G	-	84.4
R-152x4 [38]	480^2	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

2 目标检测

数据集：COCO 2017 (118k Training, 5k validation, 20k test)

(a) 表是在 Cascade Mask R-CNN, ATSS, RepPoints v2, 和 Sparse RCNN 上对比 Swin-T 和 ResNet-50 作为 Backbone 的性能

(b) 表是使用 Cascade Mask R-CNN 模型的不同 Backbone 的性能对比

(c) 表是整体的目标检测系统的对比，在 COCO test-dev 上达到了 58.7 box AP 和 51.1 mask AP

(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	-	-	-
SpineNet-190 [21]	52.6	-	52.8	-	164M	1885G
ResNeSt-200* [78]	52.5	-	53.3	47.1	-	-
EfficientDet-D7 [59]	54.4	-	55.1	-	77M	410G
DetectoRS* [46]	-	-	55.7	48.5	-	-
YOLOv4 P7* [4]	-	-	55.8	-	-	-
Copy-paste [26]	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	-

3 语义分割

数据集：ADE20K (20k Training, 2k validation, 3k test)

下图13列出了不同方法/Backbone的mIoU、模型大小(#param)、FLOPs和FPS。从这些结果可以看出，Swin-S 比具有相似计算成本的 DeiT-S 高出+5.3 mIoU (49.3 vs . 44.0)。也比 ResNet-101 高+4.4 mIoU，比 ResNeSt-101 高 +2.4 mIoU

ADE20K Method	Backbone	val mIoU	test score	#param.	FLOPs	FPS
DANet [23]	ResNet-101	45.2	-	69M	1119G	15.2
DLab.v3+ [11]	ResNet-101	44.1	-	63M	1021G	16.0
ACNet [24]	ResNet-101	45.9	38.5	-		
DNL [71]	ResNet-101	46.0	56.2	69M	1249G	14.8
OCRNet [73]	ResNet-101	45.3	56.0	56M	923G	19.3
UperNet [69]	ResNet-101	44.9	-	86M	1029G	20.1
OCRNet [73]	HRNet-w48	45.7	-	71M	664G	12.5
DLab.v3+ [11]	ResNeSt-101	46.9	55.1	66M	1051G	11.9
DLab.v3+ [11]	ResNeSt-200	48.4	-	88M	1381G	8.1
SETR [81]	T-Large‡	50.3	61.7	308M	-	-
UperNet	DeiT-S†	44.0	-	52M	1099G	16.2
UperNet	Swin-T	46.1	-	60M	945G	18.5
UperNet	Swin-S	49.3	-	81M	1038G	15.2
UperNet	Swin-B‡	51.6	-	121M	1841G	8.7
UperNet	Swin-L‡	**53.5**	**62.8**	234M	3230G	6.2