MaxViT: Multi-Axis Vision Transformer

김지환

Contents

1. Introduction

1. Introduction

Abstract

- 이미지 크기에 대한 self-attention의 scalability 부족으로 인해 트랜스포머가 최신 비전 백본에서 널리 사용되지 못하는 한계가 있는데 본 논문에서는 효율적이고 확장 가능한 어텐션 모델인 "multi-axis attention"을 소개.
- Blocked local attention, Dilated global attention 두 가지 측면으로 구성되어 임의의 입력 해상도에서 선형 복잡도만으로 global-local 공간 상호작용을 가능하게 한다.
- 이러한 attention model을 convolution과 효과적으로 결합한 새로운 구조적 요소 제안.
- MaxViT는 초기 고해상도 단계에서도 전체 네트워크에 걸쳐 globally 하게 볼 수 있다.

1. Introduction

ConvNets and ViT

ConvNets

AlexNet - Inception - Resnet - Mobilenet (efficient separable conv) -
Deeplab (Atrous conv) - Unet (enc-dec) - modern micro-design components (A ConvNet for the 2020s)

Transformer

ViT - Swin Transformer (self-attention on shifted non-overlapping windows)

1. Introduction

ConvNets and ViT

Swin Transformer 에서 처음으로 ConvNets의 ImageNet benchmark를 능가했다.
이러한 방법(self-attention on shifted non-overlapping windows)은 non-locality의 손실을 보완하기위해 hierarchical architecture를 다시 도입.
ViT의 full-attention 보다 exibility와 generalizability를 더 가졌지만 non-locality의 손실로 인해 모델 용량이 제한적이며 ImageNet-21K와 JFT 같은 대규모 데이터 체제에서 불리하게 확장되는 것을 관찰.

그러나 hierarchical network에서 초기 또는 고해상도 단계에서 full-attention을 통해 글로벌 상호작용을 하는 것은 2차 복잡도를 요구하기 때문에 계산량이 많이 소요된다. 모델 용량과 일반화 가능성의 균형을 맞추기 위해 글로벌 및 로컬 상호 작용을 효율적으로 통합하는 방법은 여전히 어려운 과제로 남아 있다.

1. Introduction

MaxViT

New type of Transformer module: Multi-axis self-attention (Max-SA)

단일 block에서 로컬 및 글로벌 공간 상호 작용을 모두 지원.

(shifted) window/local attention과는 달리, Max-SA는 global receptive field를 제안하여 모델 용량을 강화할 수 있다.

Max-SA는 네트워크의 모든 계층에서, 심지어 초기 고해상도 단계에서도 일반적인 독립형 attention 모듈로 사용할 수 있다.

이 모델의 효율성과 범용성을 입증하기 위해 Max-SA와 convolution으로 구성된 반복 블록을 계층적으로 적층하여

Multi-axis Vision Transformer(MaxViT)라는 간단하지만 효과적인 비전 백본을 설계했다.

2. Methods

2. Methods

Background - Vision Transformer

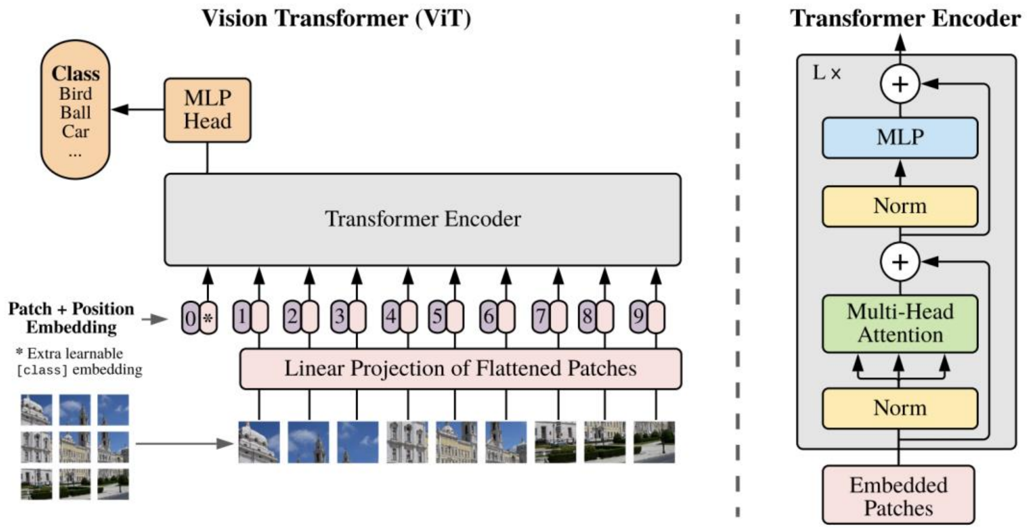
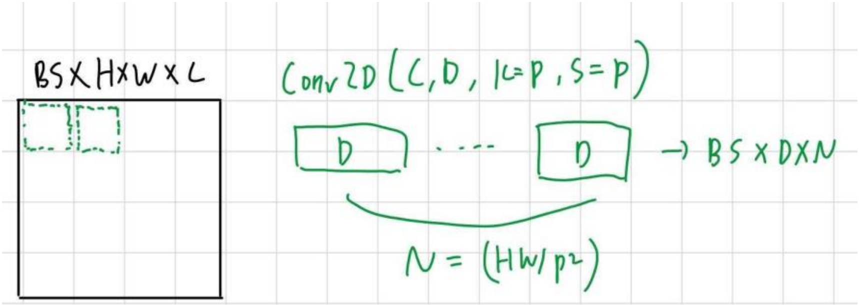


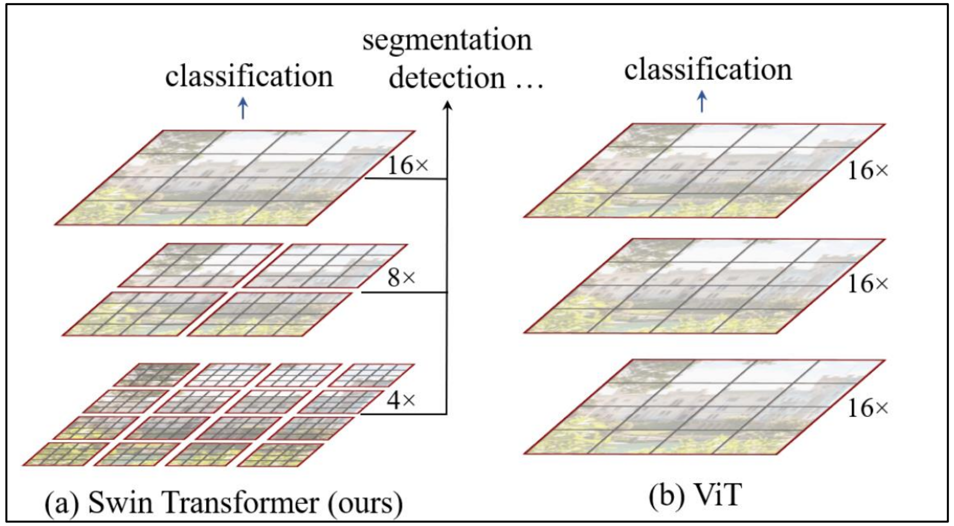
Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).



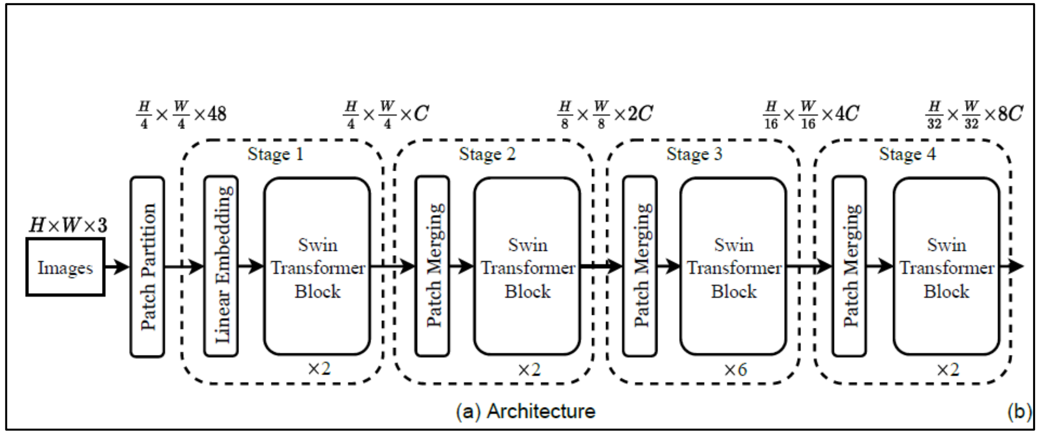
https://deepseow.tistory.com/34

2. Methods

Background - Swin Transformer

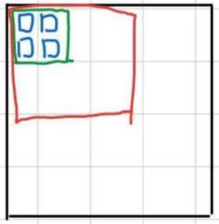


(a) Swin Transformer (ours) (b) ViT



(a) Architecture (b)

$H \times W \times 3$



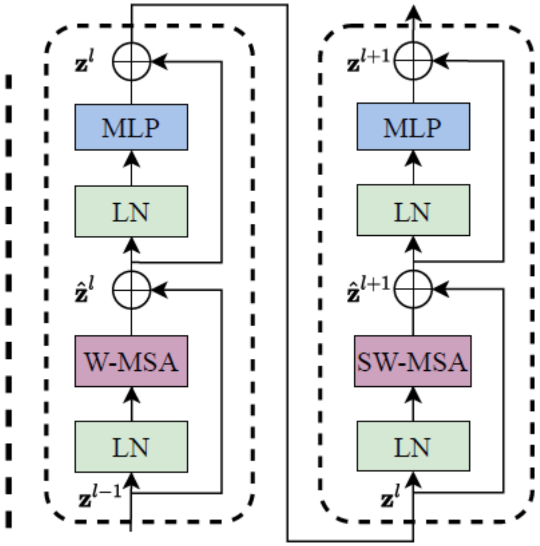
① 4×4 patch : $\frac{H}{4} \times \frac{W}{4} \times 48$ → embedding → $(\frac{H}{4} \times \frac{W}{4}) \times C$

② $(\frac{H}{8} \times \frac{W}{8}) \times 2C$ → $(\frac{H}{16}, \frac{W}{16}) \times 4C$ → $(\frac{H}{32}, \frac{W}{32}) \times 8C$

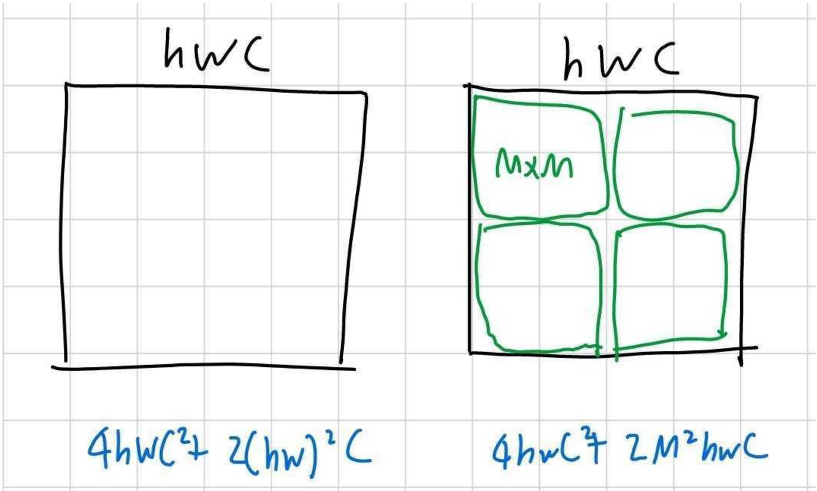
2. Methods

Background - Swin Transformer

Window-based self-attention 모듈은 윈도우간 connection이 부족하여 모델링 성능이 제한된다.
겹치지 않는 윈도우의 효율적인 계산을 유지하면서 윈도우 간 연결을 도입하기 위해 연속적인 Swin Transformer block에서
두 가지 partitioning 구성을 번갈아 사용하는 shifted window partitioning 접근 방식을 제안



Two Successive Swin Transformer Blocks



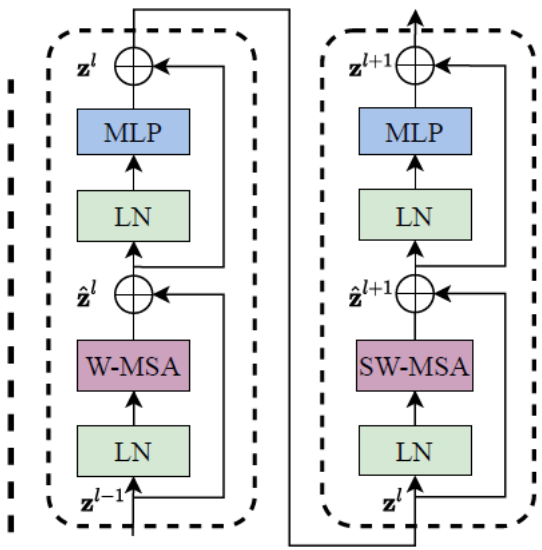
2. Methods

Background - Swin Transformer

Swin Transformer block

W-MSA: Window based multi-head self-attention using regular window partitioning

SW-MSA: Window based multi-head self-attention using shifted window partitioning



Two Successive Swin Transformer Blocks

$$
\begin{aligned}
\hat{\mathbf{z}}^{l} &= \text{W-MSA}\left(\text{LN}\left(\mathbf{z}^{l-1}\right)\right) + \mathbf{z}^{l-1}, \\
\mathbf{z}^{l} &= \text{MLP}\left(\text{LN}\left(\hat{\mathbf{z}}^{l}\right)\right) + \hat{\mathbf{z}}^{l}, \\
\hat{\mathbf{z}}^{l+1} &= \text{SW-MSA}\left(\text{LN}\left(\mathbf{z}^{l}\right)\right) + \mathbf{z}^{l}, \\
\mathbf{z}^{l+1} &= \text{MLP}\left(\text{LN}\left(\hat{\mathbf{z}}^{l+1}\right)\right) + \hat{\mathbf{z}}^{l+1},
\end{aligned}
$$

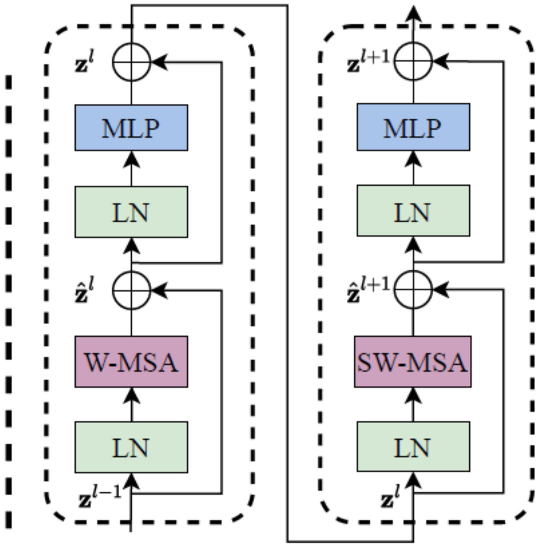
2. Methods

Background - Swin Transformer

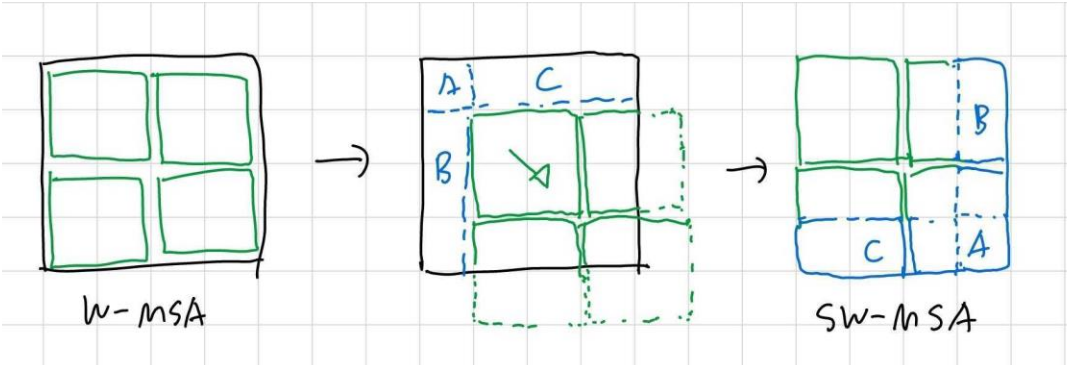
Swin Transformer block

W-MSA: Window based multi-head self-attention using regular window partitioning

SW-MSA: Window based multi-head self-attention using shifted window partitioning



Two Successive Swin Transformer Blocks

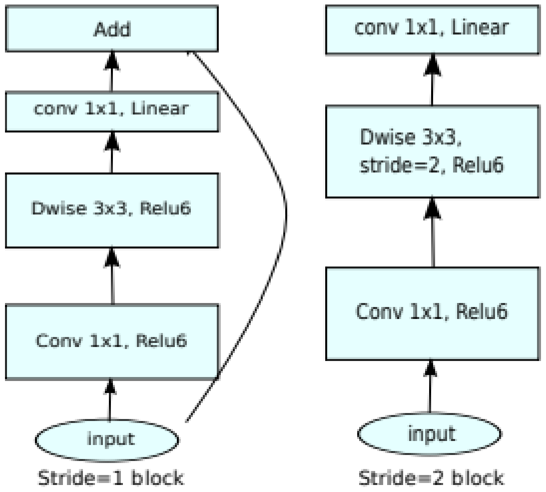


The shifted window partitioning approach introduces connections between neighboring non-overlapping windows in the previous layer and is found to be effective in image classification, object detection, and semantic segmentation

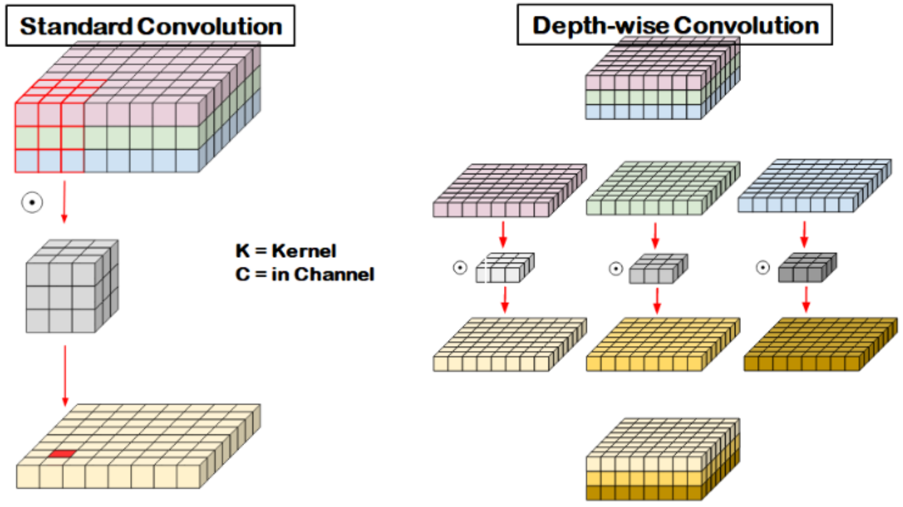
2. Methods

Background - MBConv

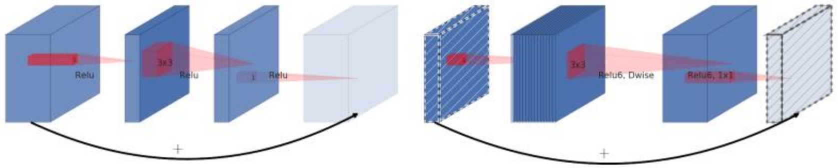
MobileNetv2



(d) Mobilenet V2



(a) Residual block (b) Inverted residual block



2. Methods

MaxViT

We introduce a new type of attention module, dubbed blocked multi-axis self-attention (Max-SA),
by decomposing the fully dense attention mechanisms into two sparse forms - window attention and grid attention -
which reduces the quadratic complexity of vanilla attention to linear, without any loss of non-locality.

2. Methods

MaxViT

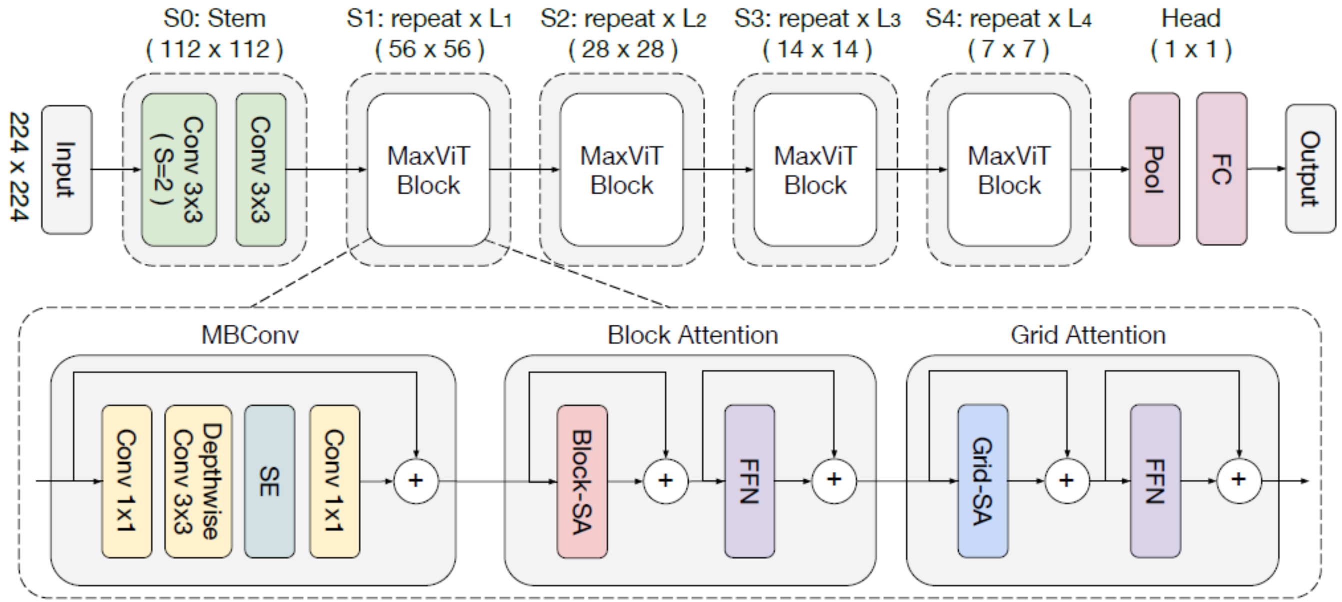
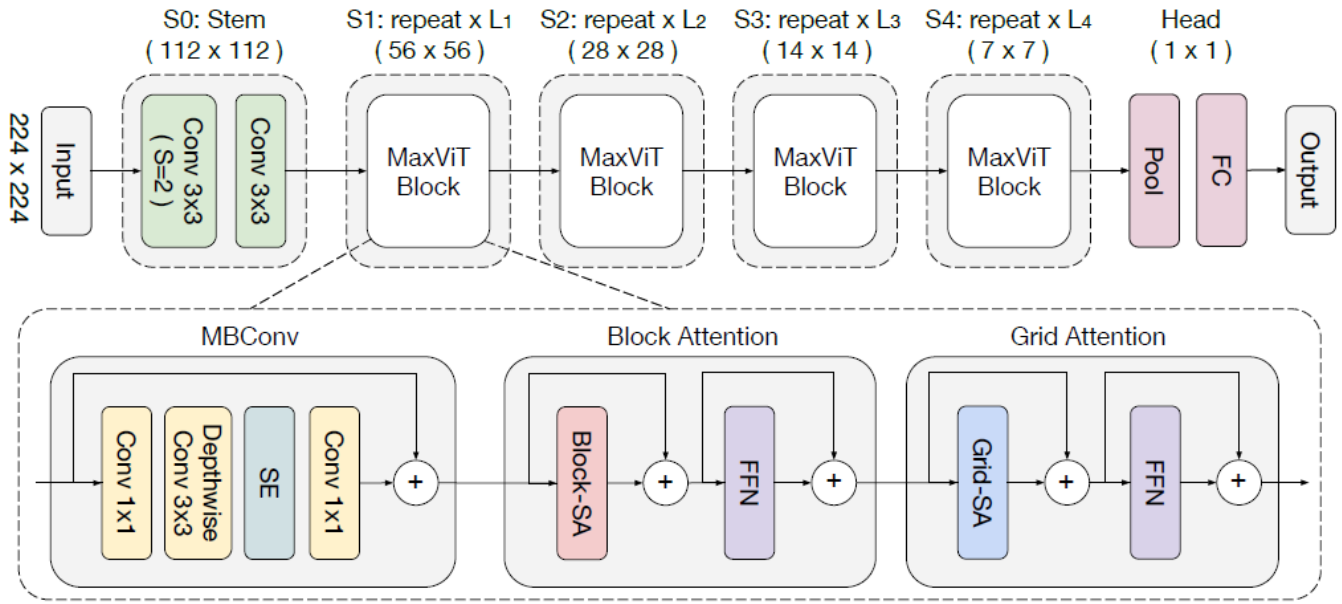


Fig. 2: **MaxViT architecture.** We follow a typical hierarchical design of ConvNet practices (e.g., ResNet) but instead build a new type of basic building block that unifies MBConv, block, and grid attention layers. Normalization and activation layers are omitted for simplicity.

2. Methods

MaxViT

We also add a MBConv block with squeeze-and-excitation (SE) module prior to the multi-axis attention, as we have observed that using MBConv together with attention further increases the generalization as well as the trainability of the network. Using MBConv layers prior to attention offers another advantage, in that depthwise convolutions can be regarded as conditional position encoding(CPE), making our model free of explicit positional encoding layers.



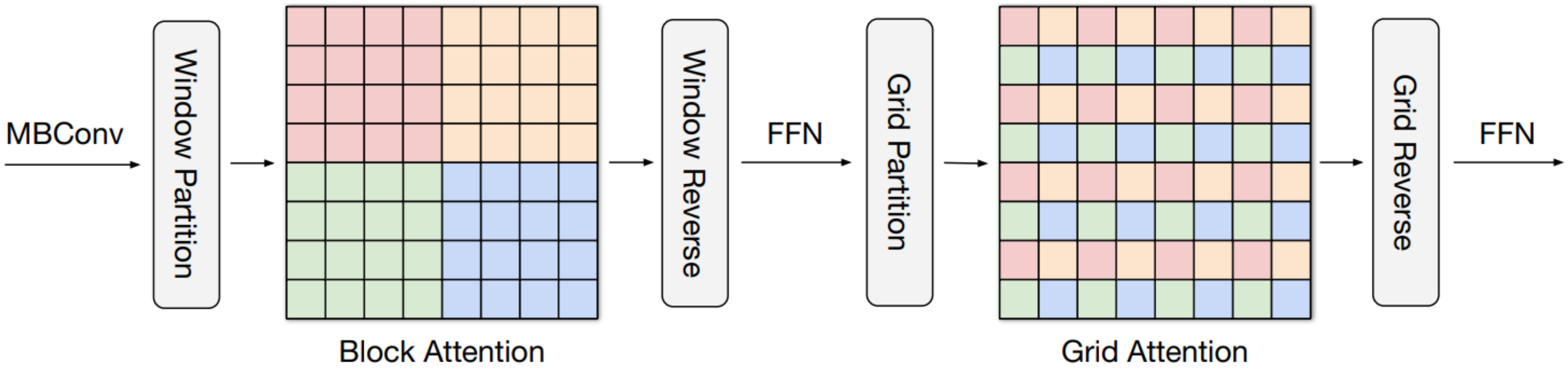
2. Methods

MaxViT

We present a multi-axis approach to decompose the full-size attention into two sparse forms - local and global - by simply decomposing the spatial axes

Block attention to conduct local interactions.

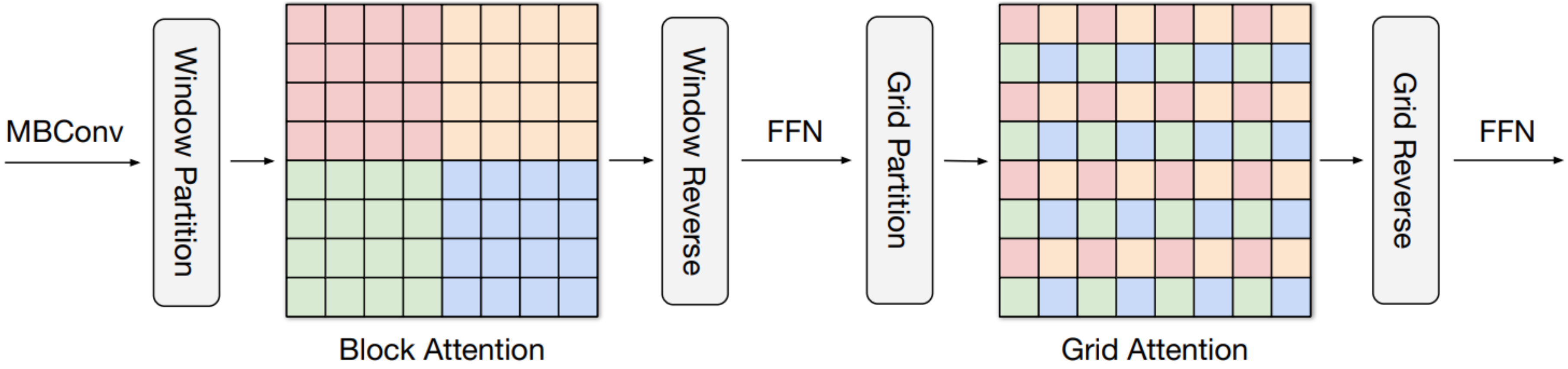
Employing self-attention on the decomposed grid axis corresponds to dilated, global spatial mixing of tokens.



2. Methods

MaxViT

Yet it enjoys global interaction capability without requiring masking, padding, or cyclic-shifting, making it more implementation friendly, preferable to the shifted window scheme



2. Methods

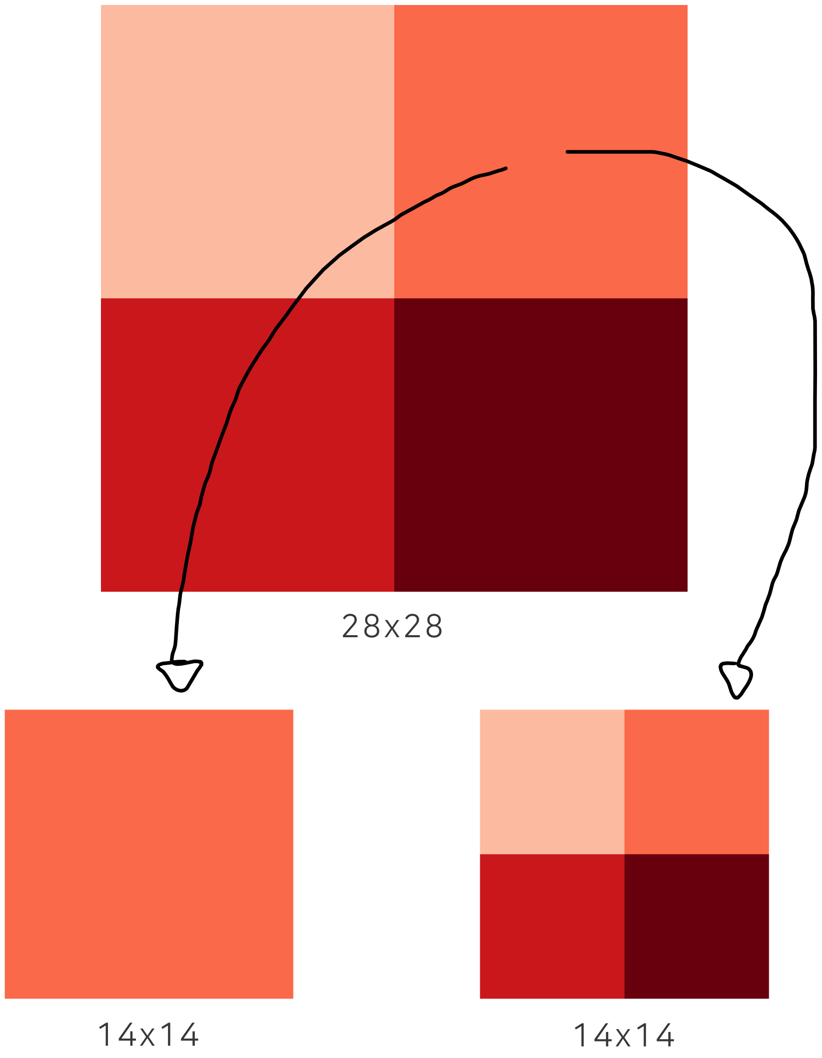
MaxViT

$$\text{Block}: (H, W, C) \rightarrow (\frac{H}{P} \times P, \frac{W}{P} \times P, C) \rightarrow (\frac{HW}{P^2}, P^2, C).$$

$$\text{Grid}: (H, W, C) \rightarrow (G \times \frac{H}{G}, G \times \frac{W}{G}, C) \rightarrow \underbrace{(G^2, \frac{HW}{G^2}, C) \rightarrow (\frac{HW}{G^2}, G^2, C)}_{\text{swapaxes(axis1=-2,axis2=-3)}}$$

```python
def window_partition_nchw(x, window_size: List[int]):
    B, C, H, W = x.shape
    _assert(H % window_size[0] == 0, f'height ({H}) must be divisible by window ({window_size[0]})')
    _assert(W % window_size[1] == 0, '')
    x = x.view(B, C, H // window_size[0], window_size[0], W // window_size[1], window_size[1])
    windows = x.permute(0, 2, 4, 1, 3, 5).contiguous().view(-1, C, window_size[0], window_size[1])
    return windows
```

```python
def grid_partition_nchw(x, grid_size: List[int]):
    B, C, H, W = x.shape
    _assert(H % grid_size[0] == 0, f'height {H} must be divisible by grid {grid_size[0]}')
    _assert(W % grid_size[1] == 0, '')
    x = x.view(B, C, grid_size[0], H // grid_size[0], grid_size[1], W // grid_size[1])
    windows = x.permute(0, 3, 5, 1, 2, 4).contiguous().view(-1, C, grid_size[0], grid_size[1])
    return windows
```



2. Methods

MaxViT

Algo. 1 Pseudocode of MaxViT Block

```
# input: features (b, h, w, c). Assume h==w; x/output: features (b, h, w, c).
# p/g: block/grid size. Use 7 by default.

def RelSelfAttn(x): return x # A self-attn function applied on the -2 axis

# Window/grid partition function
from einops import rearrange
def block(x,p):
  return rearrange(x,"b(hy)(wx)c->b(hw)(yx)c",h=x.shape[1]//p,w=x.shape[2]//p,y=p,x=p)

def unblock(x,g,p):
  return rearrange(x,"b(hw)(yx)c->b(hy)(wx)c",h=g,w=g,y=p,x=p)

x = MBConv(input) # MBConv layer

x = block(x,p) # window partition
x = RelSelfAttn(x) # Apply window-attention
x = unblock(x,x.shape[1]//p,p) # reverse

x = block(x,x.shape[1]//g) # grid partition
x = swapaxes(x,-2,-3) # move grid-axis to -2
x = RelSelfAttn(x) # Apply grid-attention
x = swapaxes(x,-2,-3) # reverse swapaxes
output = unblock(x,g,x.shape[1]//g) # reverse
```

2. Methods

MaxViT

Stage는 4개로 구성, block의 개수와 channel로 model의 size 결정.

Table 1: **MaxViT architecture variants.** B and C denotes number of blocks and number of channels for each stage. We set each attention head to 32 for all attention layers. For MBConv, we always use expansion rate 4 and shrinkage rate 0.25 in SE [36], following [19, 79, 80]. We use two Conv layers in the stem.

Stage	Size	MaxViT-T		MaxViT-S		MaxViT-B		MaxViT-L		MaxViT-XL	
S0: Conv-stem	1/2	B=2	C=64	B=2	C=64	B=2	C=64	B=2	C=128	B=2	C=192
S1: MaxViT-Block	1/4	B=2	C=64	B=2	C=96	B=2	C=96	B=2	C=128	B=2	C=192
S2: MaxViT-Block	1/8	B=2	C=128	B=2	C=192	B=6	C=192	B=6	C=256	B=6	C=384
S3: MaxViT-Block	1/16	B=5	C=256	B=5	C=384	B=14	C=384	B=14	C=512	B=14	C=768
S4: MaxViT-Block	1/32	B=2	C=512	B=2	C=768	B=2	C=768	B=2	C=1024	B=2	C=1536

3. Results

Thank you