Swin Transformer Paper Review

Hierarchical Vision Transformer Using Shifted Windows

Prof. John McDonald[1]
Eduardo A. Avila H.[1]

[1]Computer Sciences Department

CVS Meeting September 2021

Table of Contents

Table of Contents

CNN Backbone Architectures for Computer Vision

AlexNet

VGGNet

ResNet

DenseNet

Main Qualities and Advantages

First transformer based backbone architecture for C. V.

Image Classification

Object Detection

New State of the Art Performance

Vision Transformer (ViT)

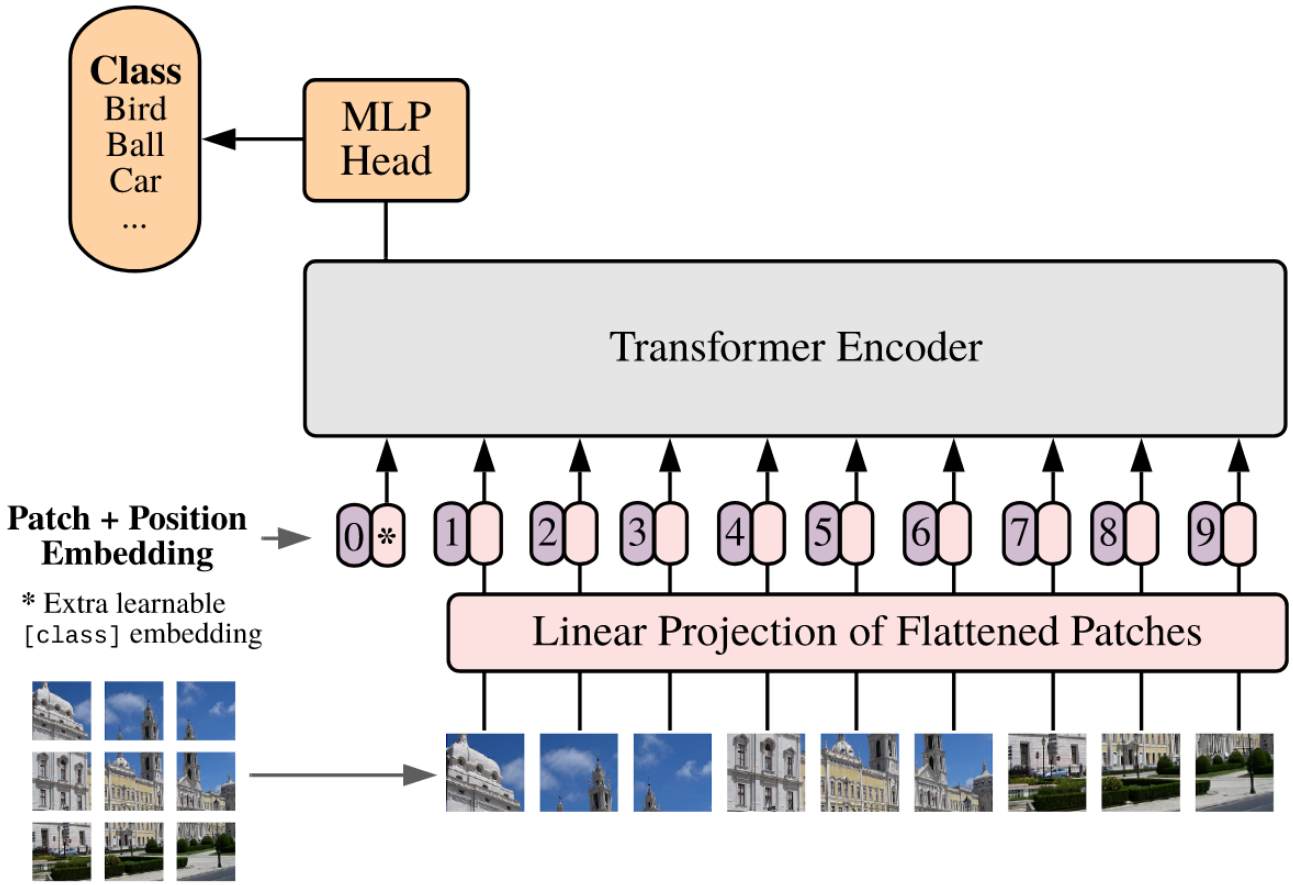

Figure: Vision Transformer [?]

Classic Transformer Encoder

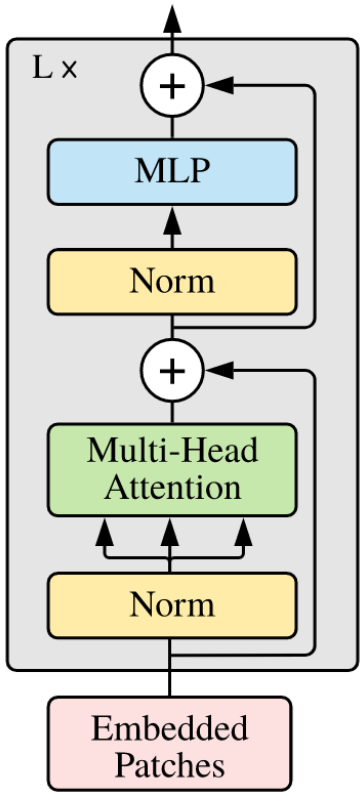

Figure: Classic Transformer Block [?]

Table of Contents

Architecture Main Components

Understanding the following four composing blocks is crucial for the understanding of Swin Transformers.

- Patch Partition
- Linear Embedding
- Swin Transformer Block
- Patch Merging

Architecture Main Components

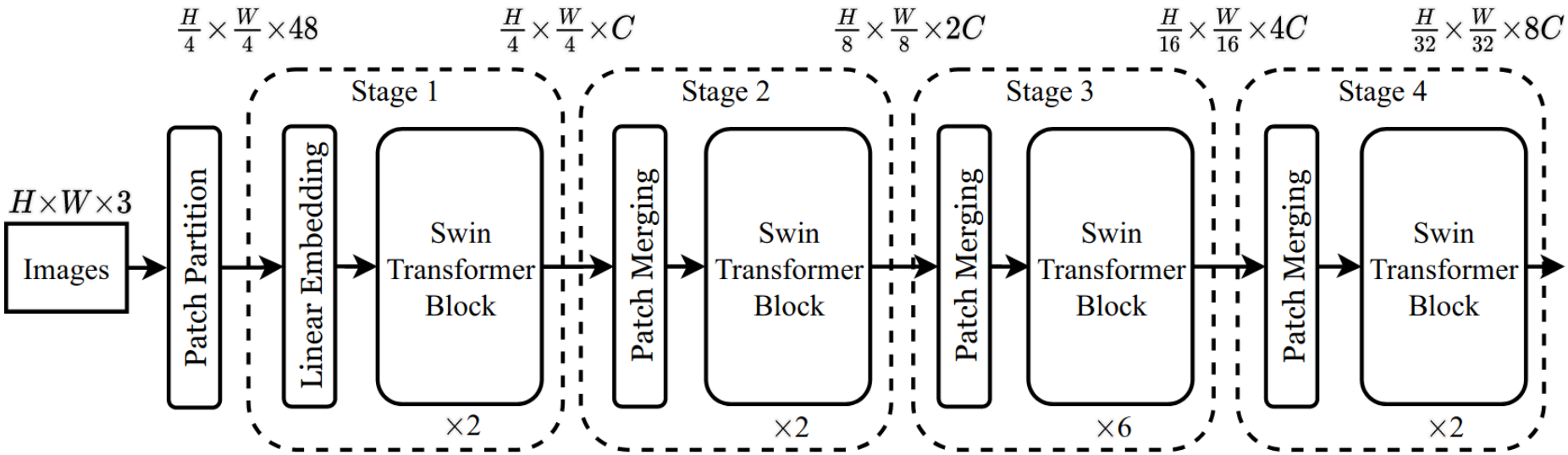

Figure: Swin Transformer Architecture. Composed of Patch Partition, Linear Embedding, Swin Transformer Block and Patch Merging modules. [?]

Patch Partition

An image with a height (H), width (W) and three channels as an input is divided into non overlapping patches or tokens. Using a patch size of 4 by 4 by 3 channels results in a 48 dimension patch.

Patch Partition

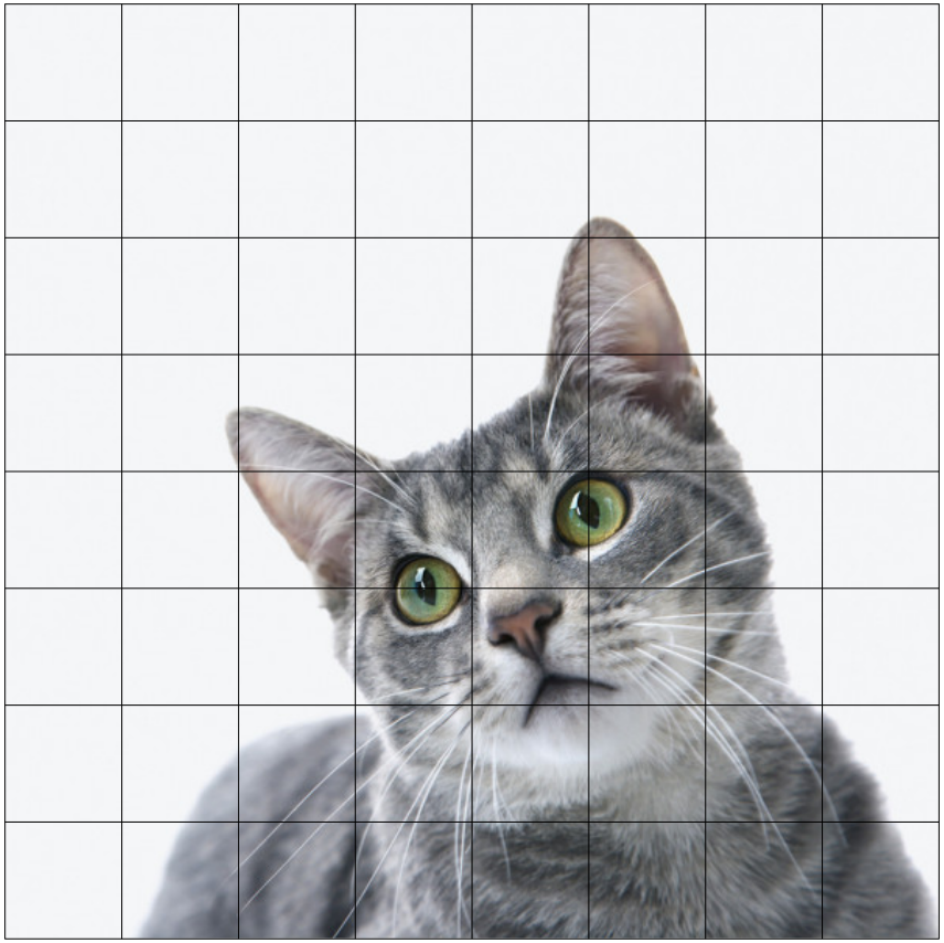

Figure: Imaged Partitioned Into 64 Patch Partitions

Linear Embedding Layer

This layers is composed of an artificial neural network taking an input of 48 dimension vector of each patch, then converting it into (C) dimension vector. C is defined as the channel number of the hidden layers and is given standard values of 96, 128 and 192.

Multi-Layer perceptron (MLP)

Multi-Layer Perceptron, also Known as Vanilla Neural Network, is a type of feed forward artificial neural network (ANN). They consist of at least three layers of nodes. Input layer, hidden layer and output layer. The MLP block is used with $tanh()$ as non-linear activation function.

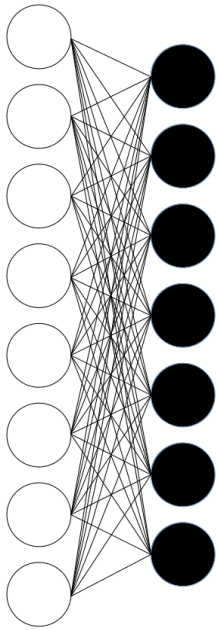

Figure: Multi-Layer Perceptron

Swin Transformer Block

A single transformer layer is replaced by two transformer layers. The Window Multi-Headed Self Attention (W-MSA) and the Shifted Window Multi-Headed Self Attention (SW-MSA).

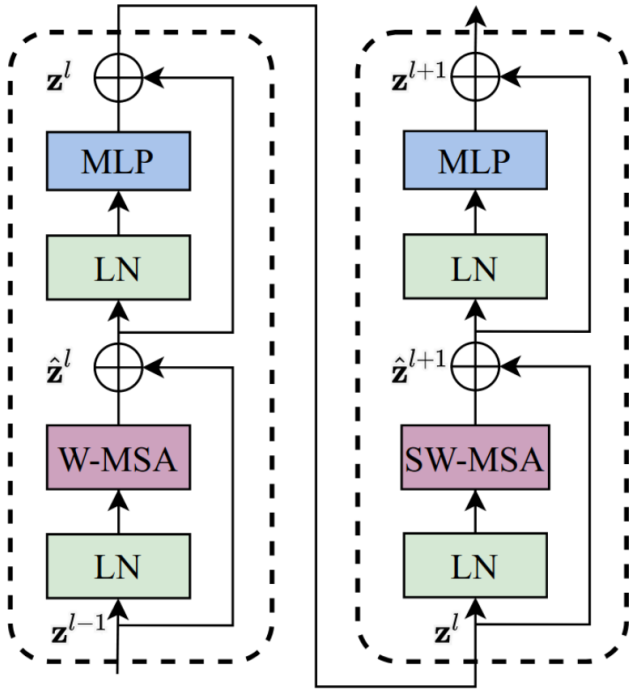

Figure: Swin Transformer Block. The block is composed of two transformer blocks, a W-MSA and SW-MSA. [?]

Patch Merging

Simplifying the problem to an image of 8px by 8px by 3ch. Divide the image into four patches resulting in 4 patches of size 4px by 4px by 3ch. In terms of patches there are 2 by 2 by 1 patches. For patch merging the two neighbouring patches are combined to result into a 1 by 1 patch. Utilizing a linear layer or artificial neural network the ability of deciding the size of the output is given. From the ANN output the channel number of the hidden layers C is doubled, becoming $2C$ and decreasing the number of patches by 2, increasing C by 2.

Taking the original value of $C = 96$ it can be said that $2C = 192$ making it the size of the output of the stage two of the swin transformer network.

Patch Merging

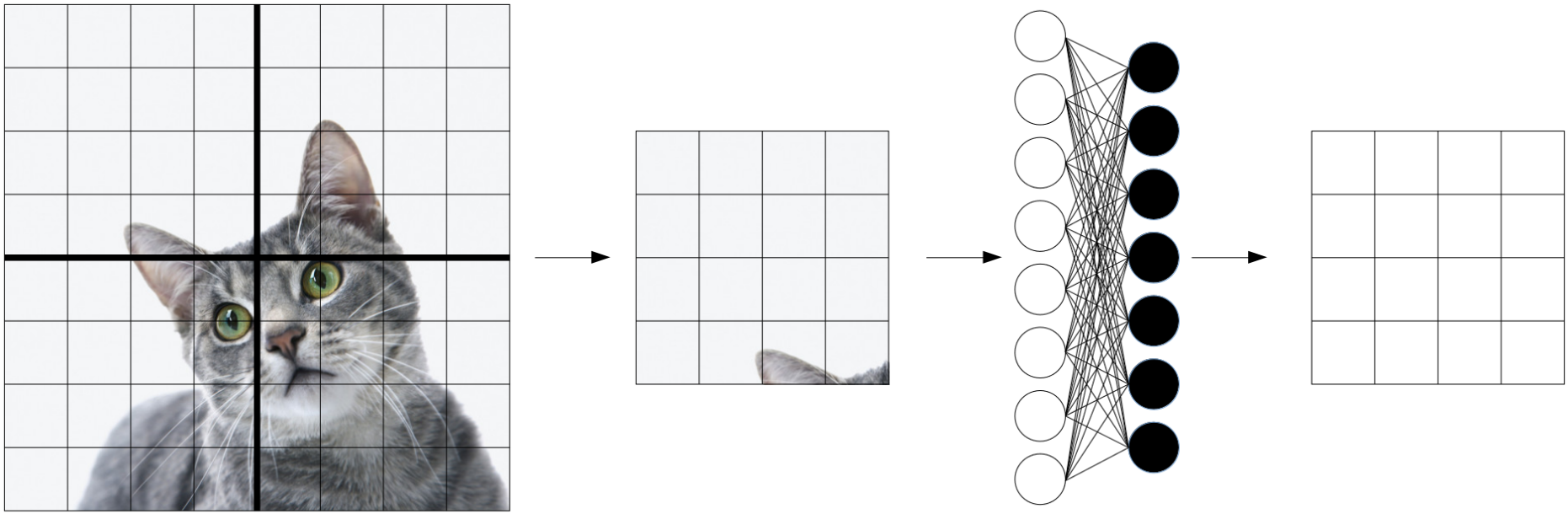

Figure: Patch Merging Module Example

Table of Contents

Vision Transformer (ViT)

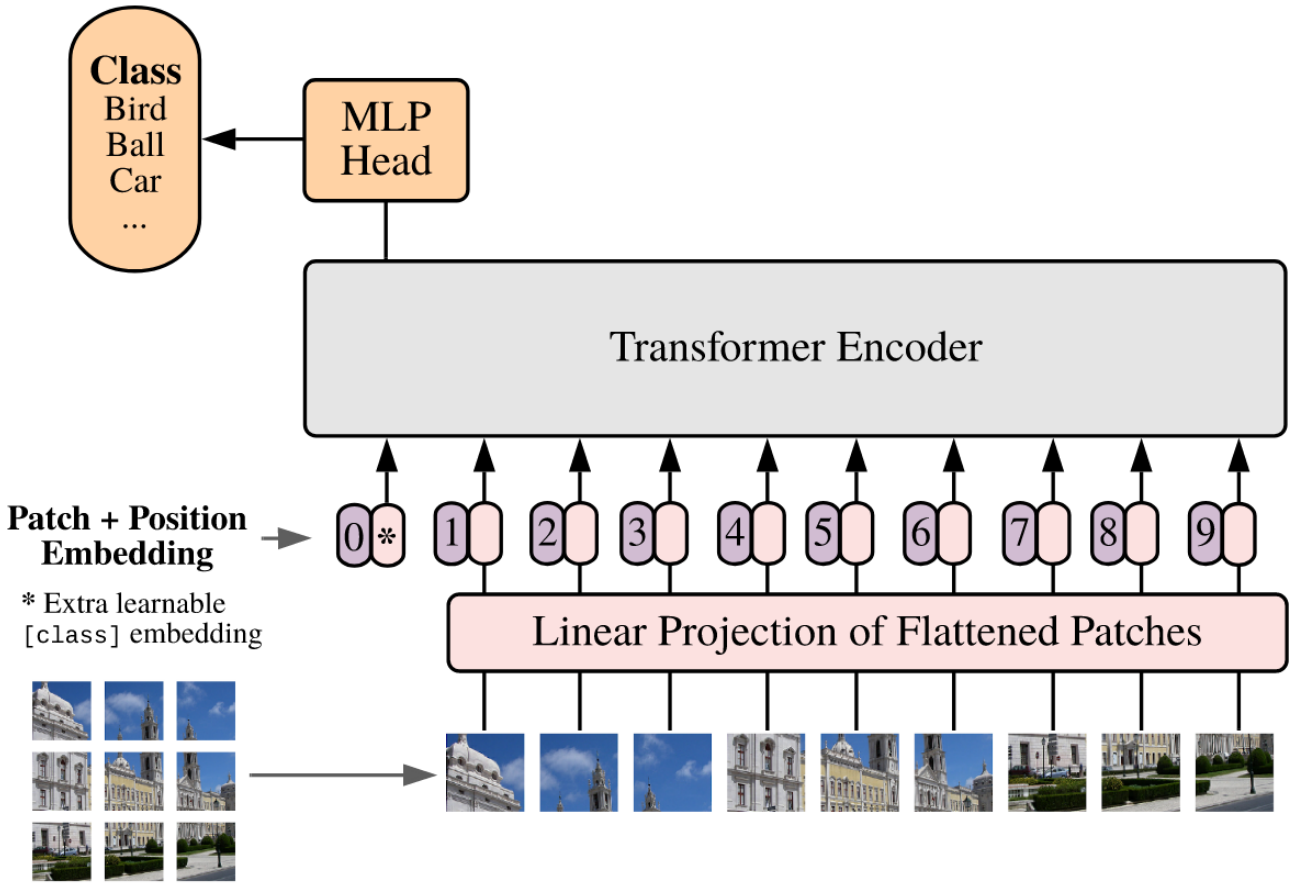

Figure: Vision Transformer [?]

Classic Transformer Block

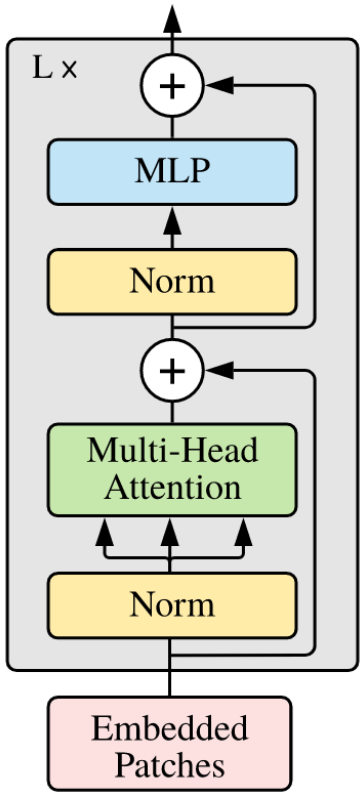

Figure: Classic Transformer Block [?]

Multi-Headed Self Attention (MSA)

Standard attention mechanism in transformer networks. Works well for natural language processing tasks. Computing the attention for every word allows to create the numerical relationship in between all the words in the given sample. However, the computation of attention is a quadratic operation. To calculate attention a pairwise inner product between each token is required.

In the case of images, the image have been divided into several non overlapping patches. Self attention still needs to be computed for every given patch, making multi-Headed Self Attention computationally intensive task for image processing.

Multi-Headed Self Attention (MSA)

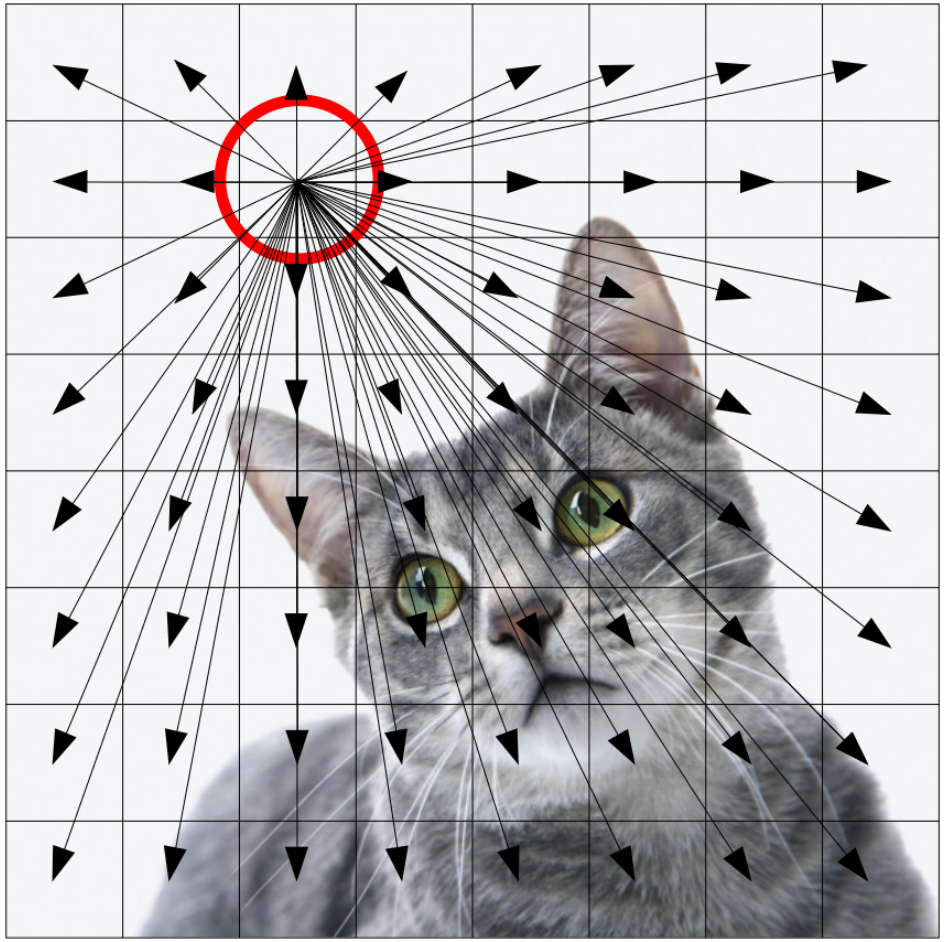

Figure: Multi-Headed Self Attention

Non-Ovelapped Windows

Self-attention algorithm is only applied on the bases of local windows in order to increase efficiency. Windows are arranged to evenly partition the input image in a non-overlapping manner.

Each window will contain $M \times M$ patches. in example $M = 4$

Non-Ovelapped Windows

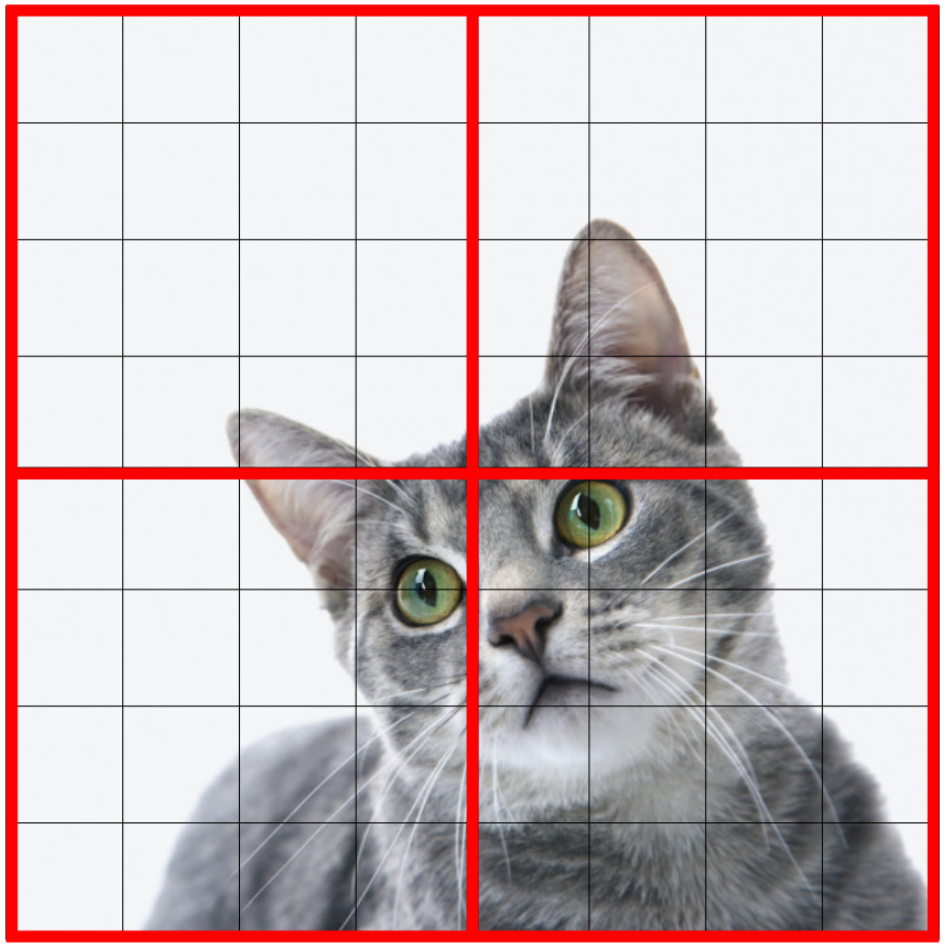

Figure: Image divided by Non Ovelapping Windows

Shifted Window Partitioning

Swin Transformers introduce a window partitioning and shifting Attention methodology. The first module divides the image into evenly partitioned into 2×2 windows of size 4×4 patches ($M = 4$). The second module displaces the window partitions by $(\frac{M}{2}, \frac{M}{2})$ pixels from the regularly partitioned windows.

Swin Transformer Block

A single transformer layer is replaced by two transformer layers. The Window Multi-Headed Self Attention (W-MSA) and the Shifted Window Multi-Headed Self Attention (SW-MSA).

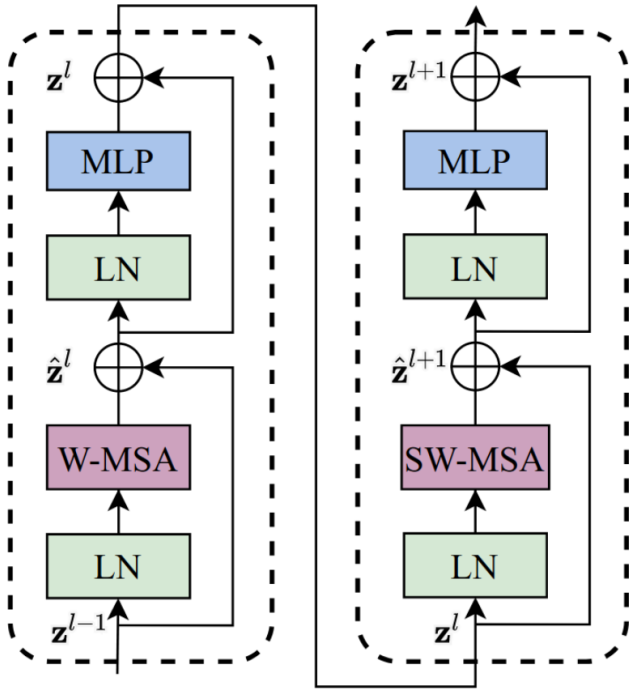

Figure: Swin Transformer Block. The block is composed of two transformer blocks, a W-MSA and SW-MSA. [?]

Window Multi-Headed Self Attention (W-MSA)

The image is divided into windows and the attention is computed in between the patches of the current analysed window. This occurs in the first layer of the swin transoformer block.

Window Multi-Headed Self Attention (W-MSA)

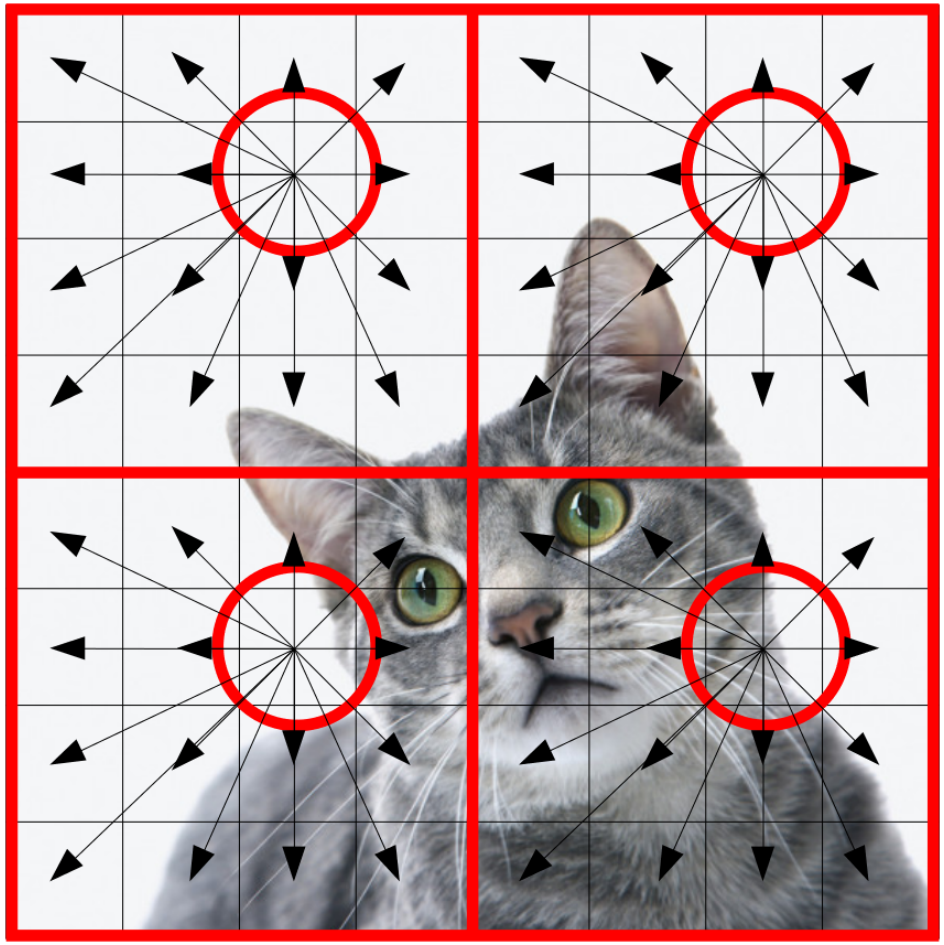

Figure: Window Multiheaded Self Attention Calculation

Computational complexity of (MSA) vs (W-MSA)

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

$$\Omega(W-MSA) = 4hwC^2 + 2M^2(hw)C \tag{2}$$

Global self attention computation is generally non affordable on a large hw. However, the window based self-attention is scale-able.

Shifted Window Multi-Headed Self Attention (SW-MSA)

Recollecting the CNN computation mechanism, a kernel slides across the image and computes the outputs of the layer. The equivalent of the sliding kernel in the transformer network is the window shift.

The windows are shifted diagonally by two patches. In other words, the second module displaces the window partitions by $(\frac{M}{2}, \frac{M}{2})$ pixels from the regularly partitioned windows. Then the attention within the windows is computed. An empty space is caused by the diagonal shift. To compensate for this, the space can be zero padded. A better approach is cycle shifting.

Efficient Batch Computation

Issues occur with the shifting window partitioning, as shifting will result in more windows of different patch sizes. Some of the windows will result to be smaller that $M \times M$. A naive solution is to pad the smaller windows to a size of $M \times M$. Doing this will increase the computation of the self-attention step.

To overcome this, cyclic-shifting is utilized as an efficient batch computation approach. Performing a masking mechanism on the different size window batches, the number of batched windows remains the same as the one of regular partitioning.

Shifted Window Multi-Headed Self Attention (SW-MSA)

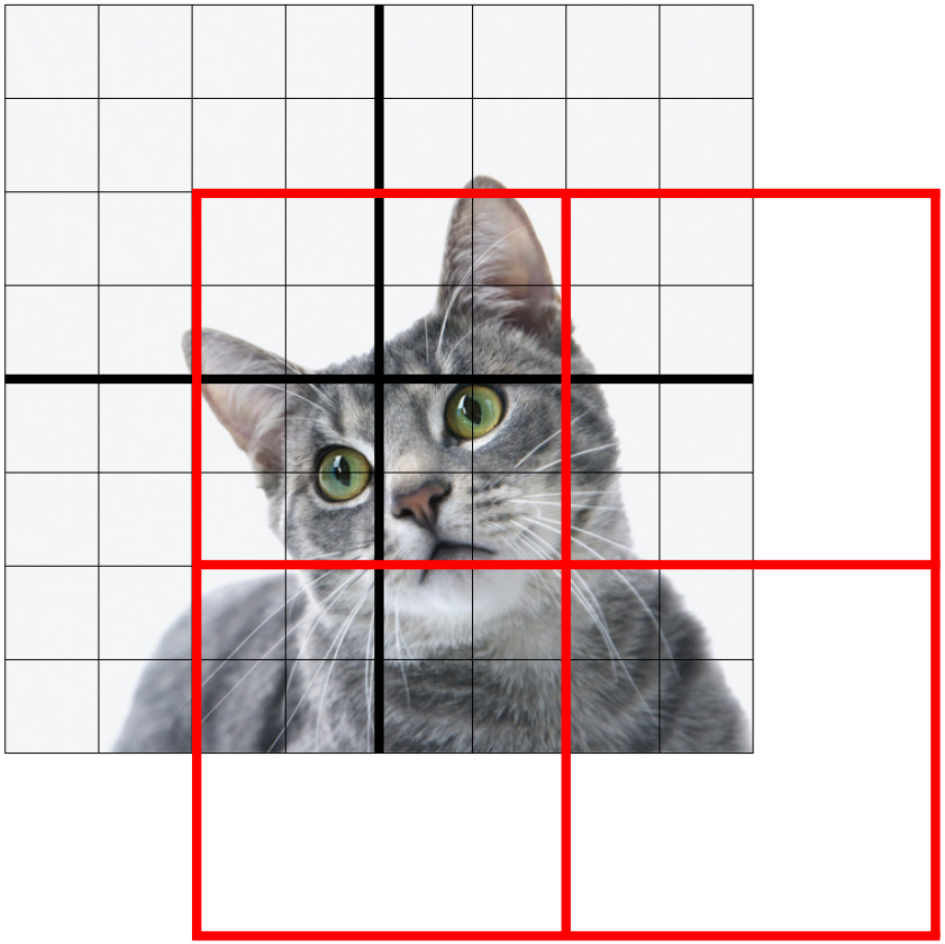

Figure: Window Shift Displacement

Shifted Window Multi-Headed Self Attention (SW-MSA)

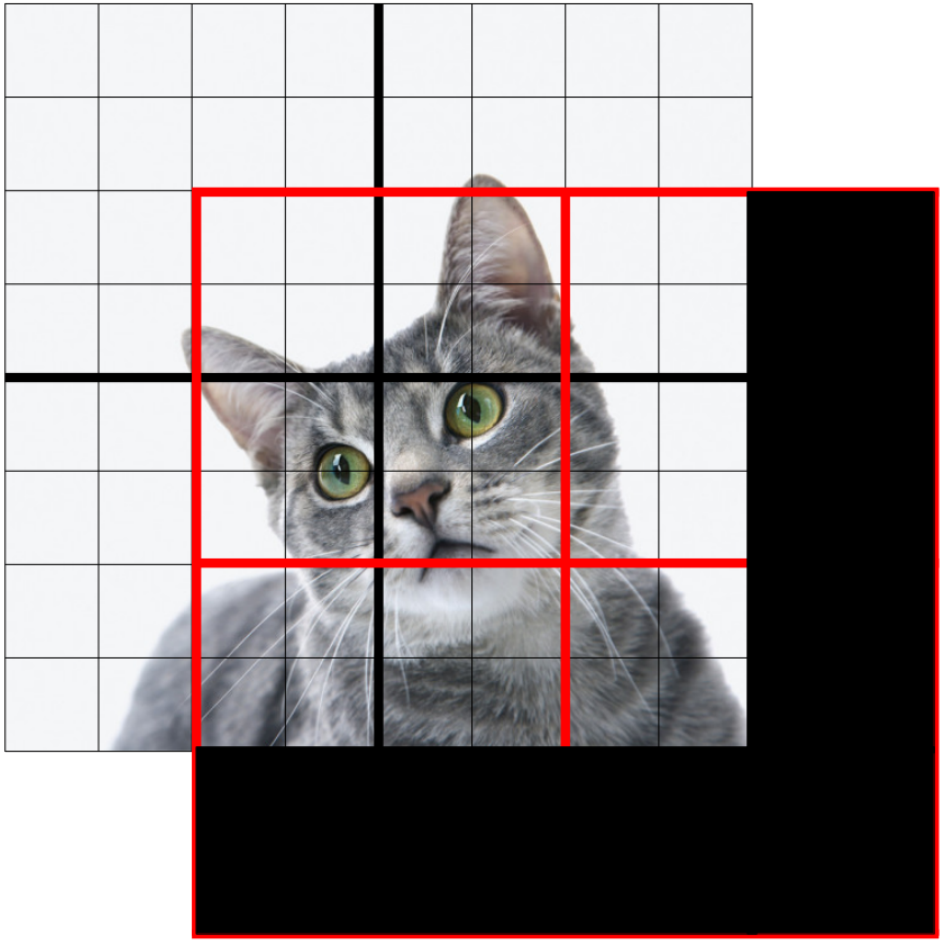

Figure: Window Shift Padding

Shifted Window Multi-Headed Self Attention (SW-MSA)

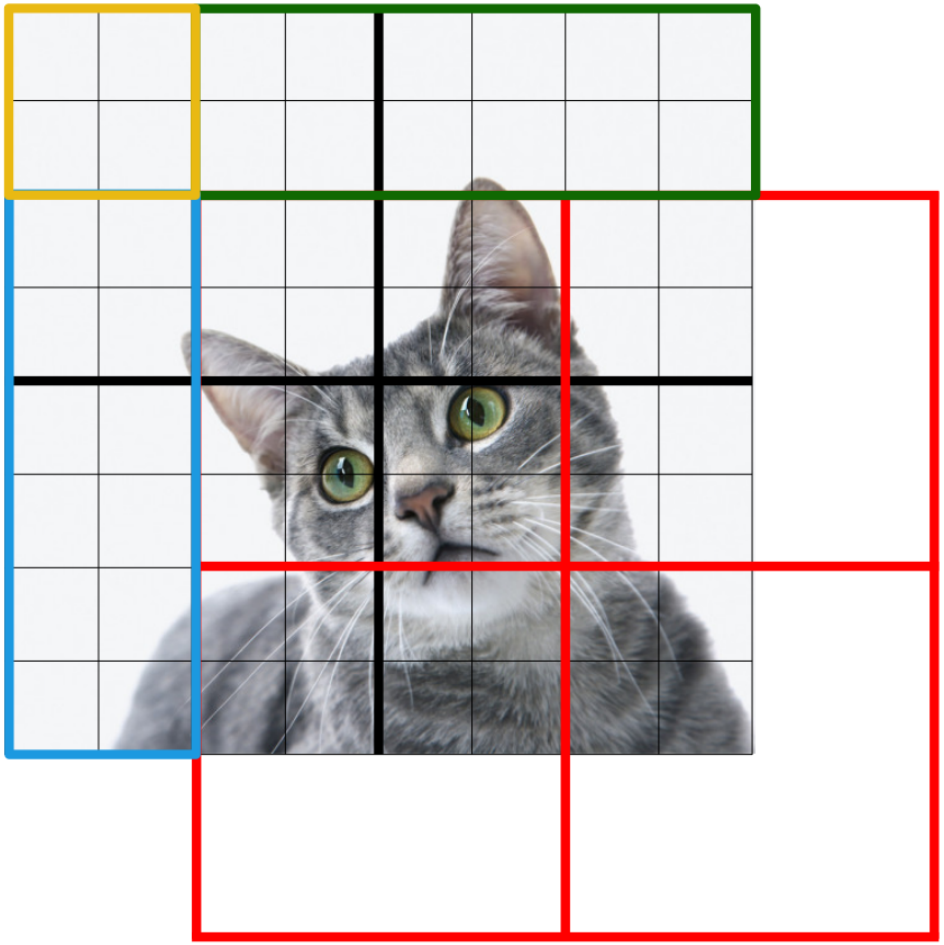

Figure: Cycle Shift

Shifted Window Multi-Headed Self Attention (SW-MSA)

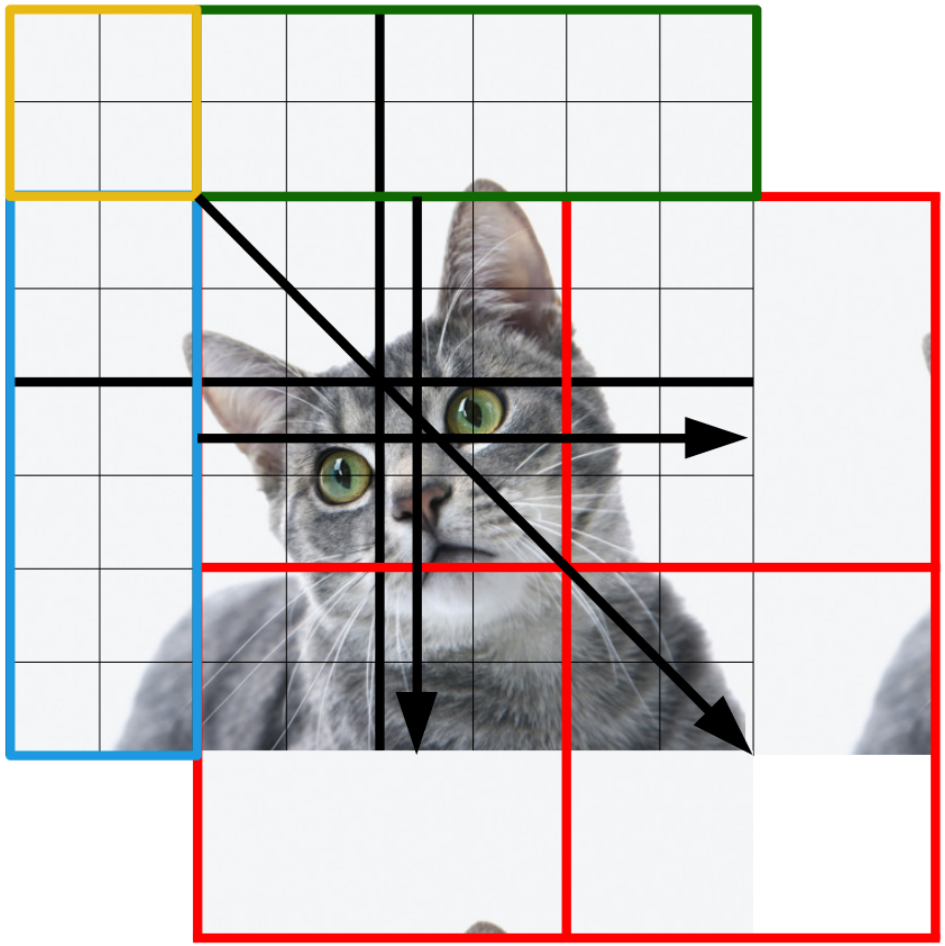

Figure: Cycle Shift

Shifted Window Multi-Headed Self Attention (SW-MSA)

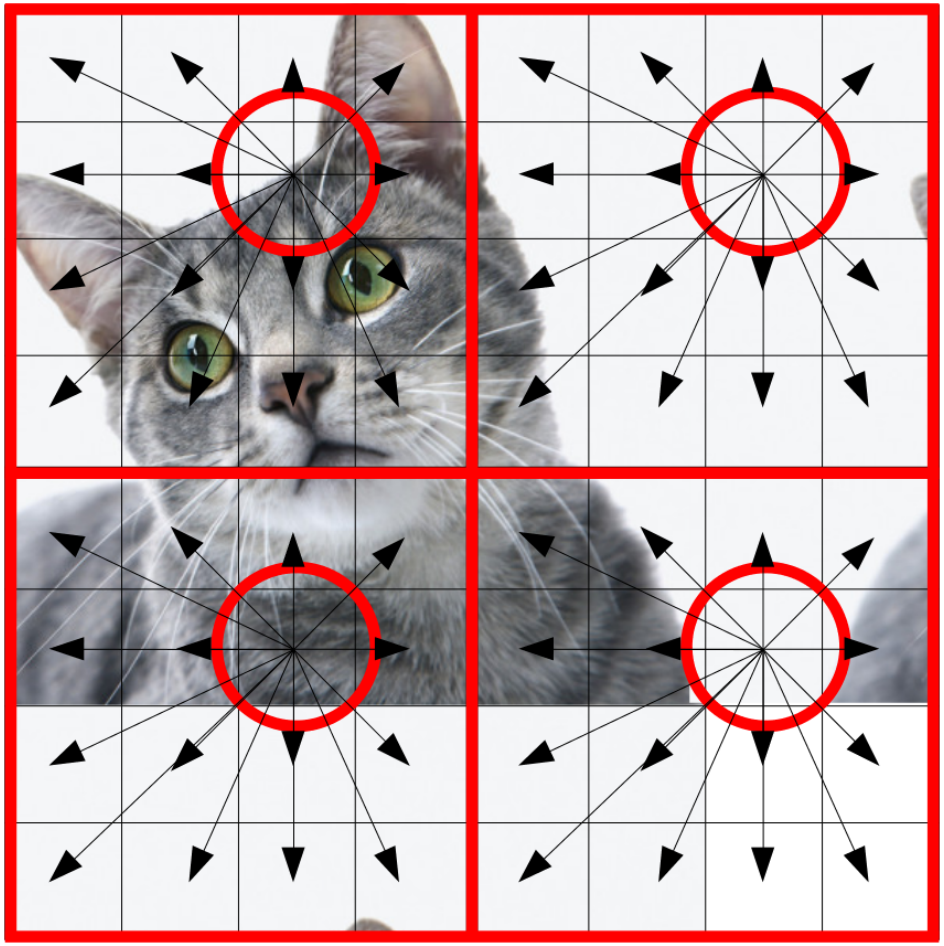

Figure: Attention Calculation on Cycle Shifted Features

Relative Position Bias

$$Attention(Q, K, V) = SoftMax(QK^T/\sqrt{d} + B)V \tag{3}$$

Where,

$Q \equiv$ query

$K \equiv$ key

$V \in \mathbb{R}^{M^2 \times d} \equiv$ value matrices

$d \equiv$ query/key dimension

$M^2 \equiv$ number of patches in a window

Since the relative position along each axis lies in the range $[-M+1, M-1]$ values in B are taken from $\hat{B}$

$$\hat{B} \in \mathbb{R}^{(2M-1)\times(2M-1)} \tag{4}$$

Table of Contents

Architecture Variants

The base model is called Swin-B was built with similar model size and complexity as ViT-B/DeiT-B.

Swin-T: $C = 96$, layer numbers $= \{2, 2, 6, 2\}$, size $0.25\times$

Swin-S: $C = 96$, layer numbers $=\{2, 2, 18, 2\}$, size $0.5\times$

Swin-B: $C = 128$, layer numbers $=\{2, 2, 18, 2\}$, size $1\times$

Swin-L: $C = 192$, layer numbers $=\{2, 2, 18, 2\}$, size $2\times$

Where,

C is the channel number of the hidden layers in the first stage

$M = 7$ as default window size

$d = 32$ as the query dimension of each head

$\alpha = 4$ as the expansion layer of each MLP

Table of Contents

Classification Results

(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

Figure: Results on ImageNet 1K

Classification Results

(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

Figure: Results on ImageNet 22K

Object Detection Results - COCO 2017 DS

(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

Figure: Object Detection Results

Object Detection Results - COCO 2017 DS

(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

Figure: Object Detection Results

Segmentation Results

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

Figure: Segmentation Results

References