Swin Transformer

Group 7

Vinod Nithin Kumar Rachakonda

Sai Nithisha Marripelly

Gautam Mehta

How ViT works?

- Splits image into fixed-size patches, treated as input tokens
- Applies global self-attention to model patch interactions

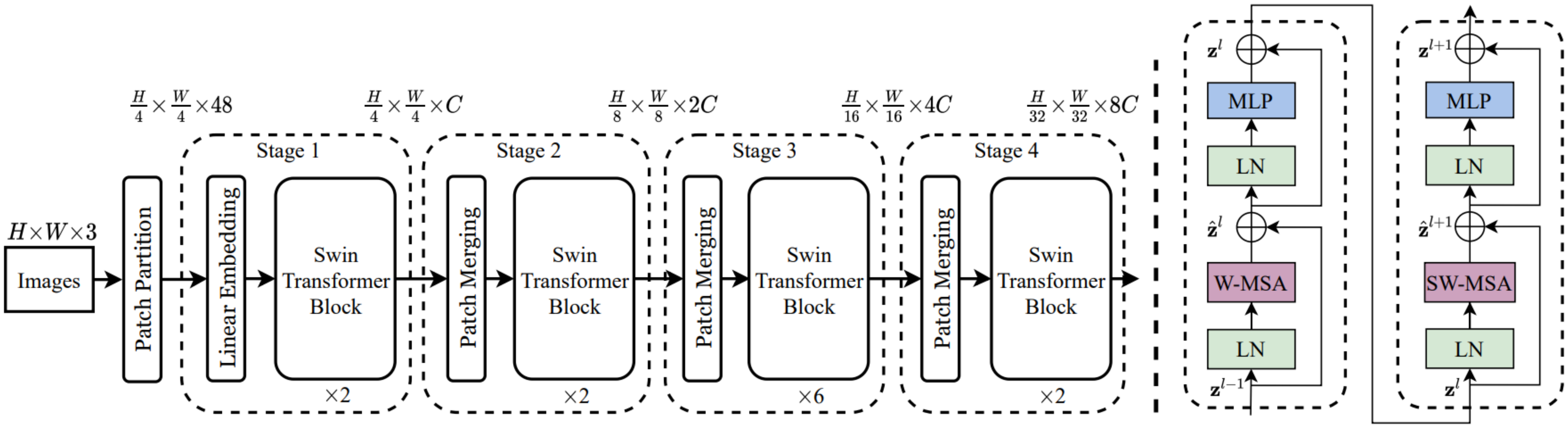
Challenges:

- Quadratic computational complexity
- Struggle with high-resolution images
- Loss of local spatial relationships

Swin Transformer

- Introduced:
  - Window based Multi-head Self Attention (W-MSA)
  - Shifted window based Multi-head Self Attention (SW-MSA)
  - Stage wise Patch merging
- Advantages:
  - Improved performance on visual tasks with fewer parameters
  - Linear computational complexity with input size
  - Captures both local and global features

Swin Transformer



1

Patch Partition
+
Linear Embedding

4

Patch Merging

2

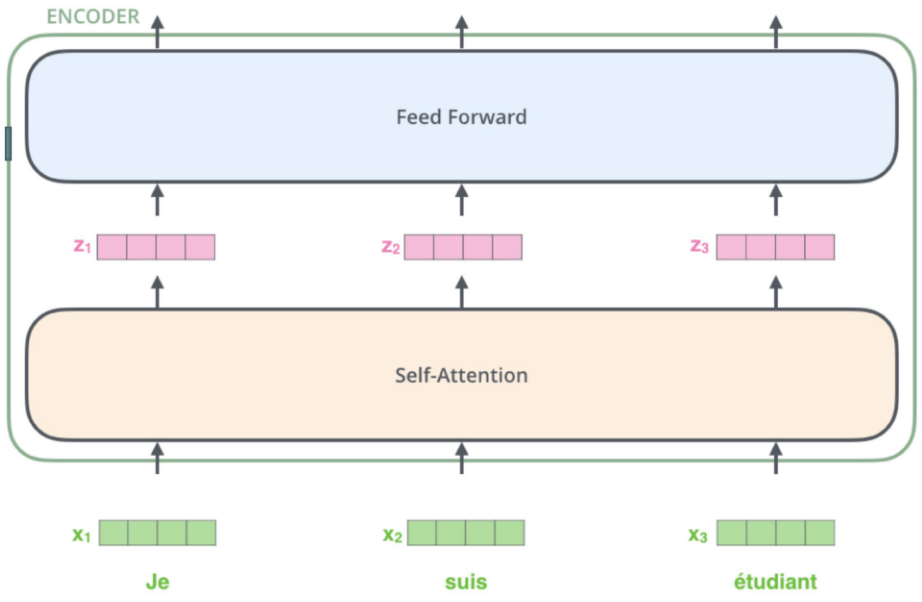
W-MSA

3

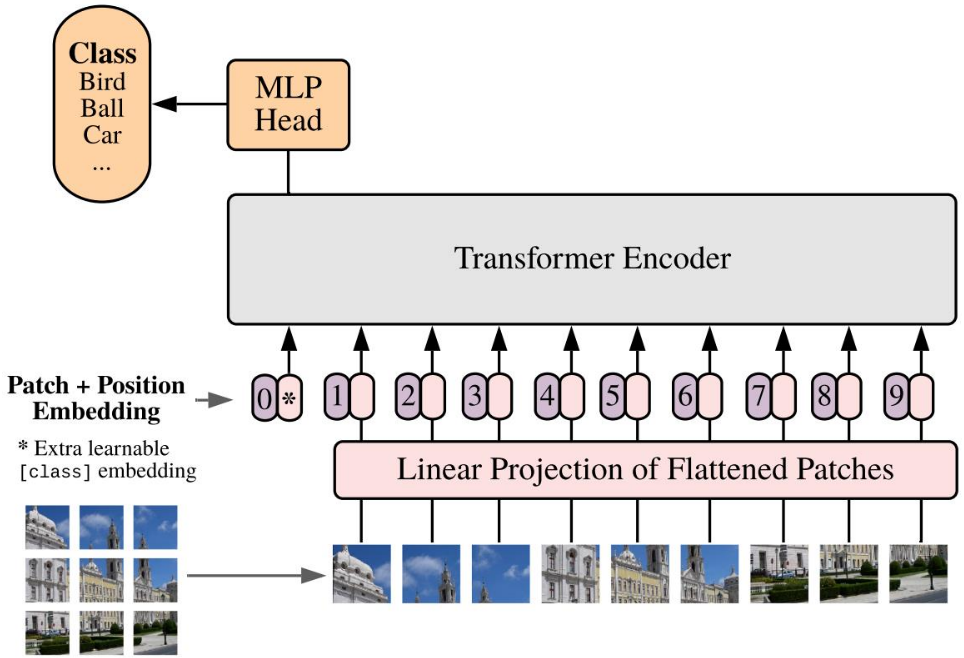
SW-MSA

1) Patch Partition and Linear embedding

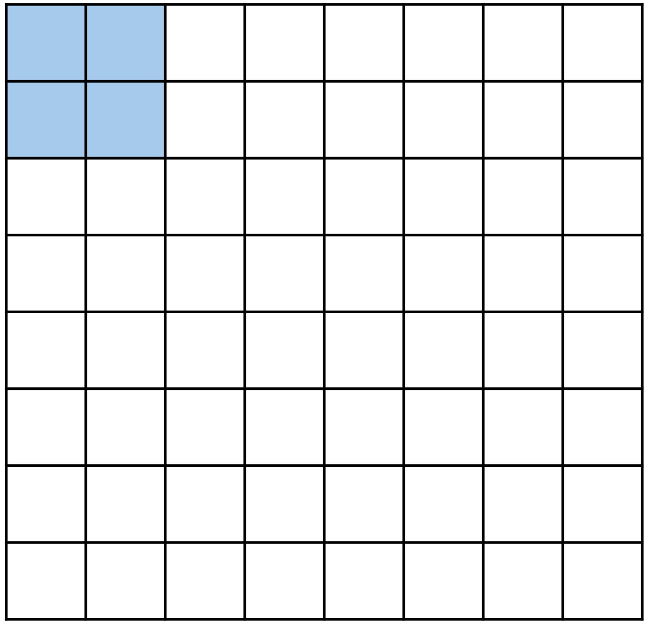
Inputs to a Transformer



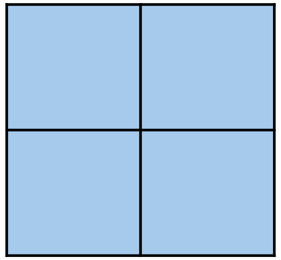
Inputs to a ViT



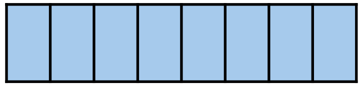
1) Patch Partition and Linear embedding



8x8 input

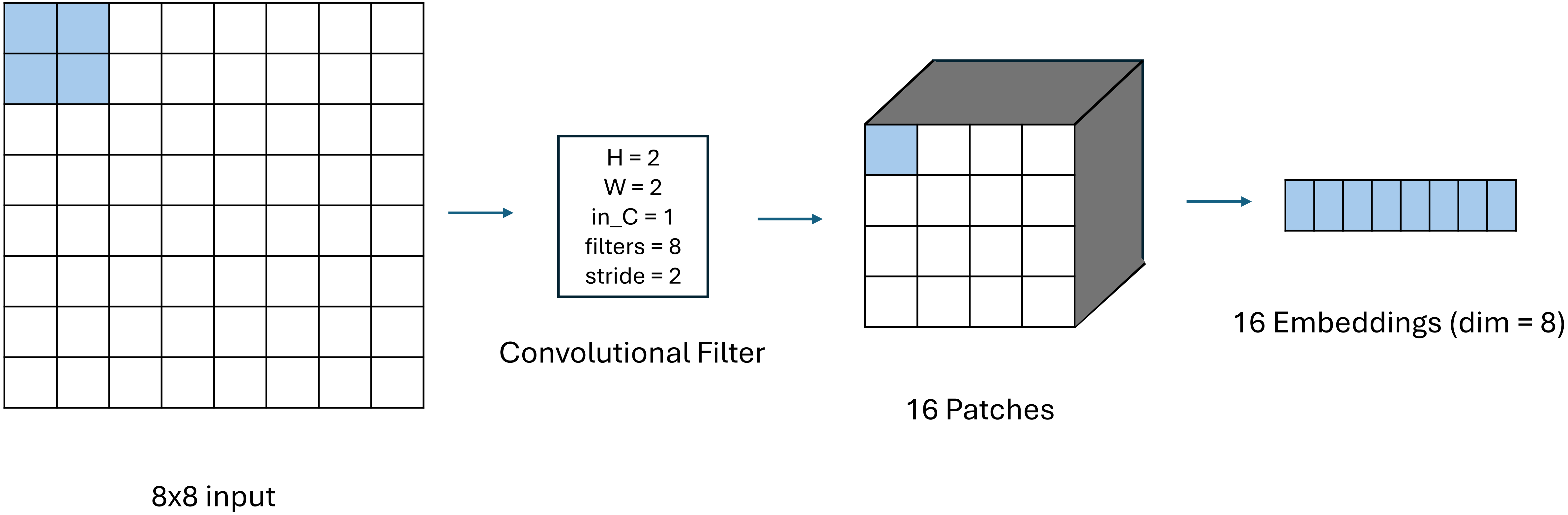


16 - 2x2 patches

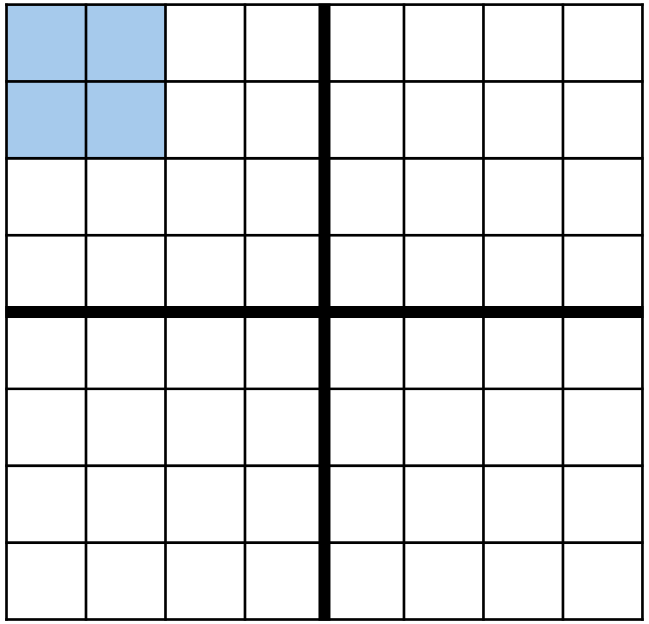


16 Embeddings (dim = 8)

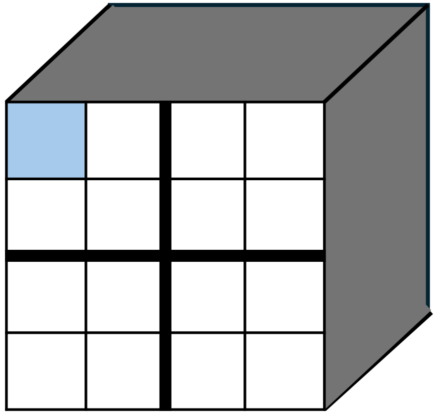
1) Patch Partition and Linear embedding



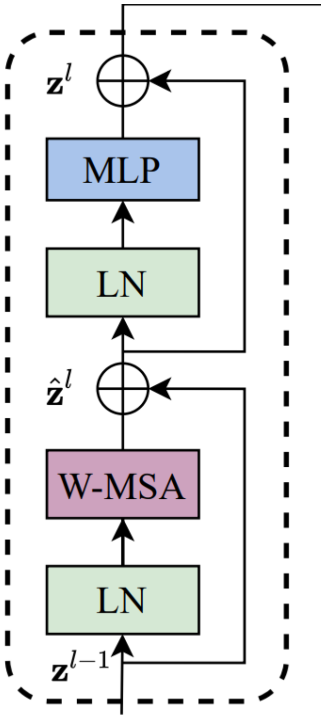
2) Window based Multi-head Self Attention (W-MSA)



8x8 input



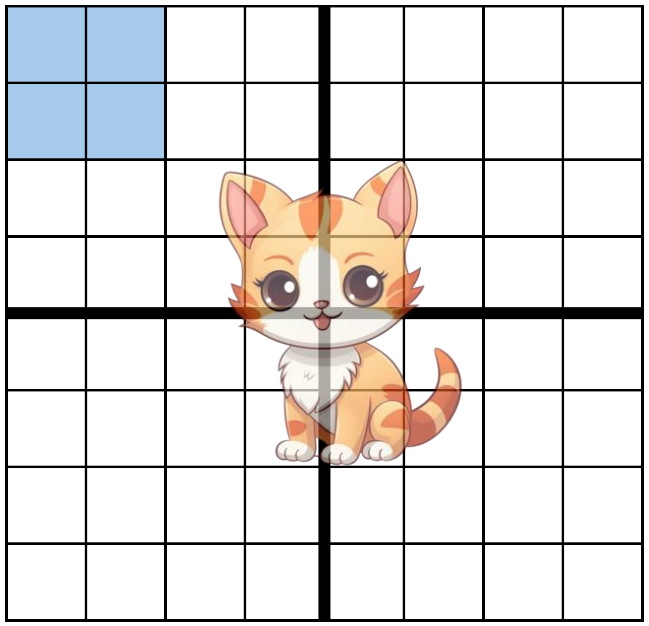
16 Patches



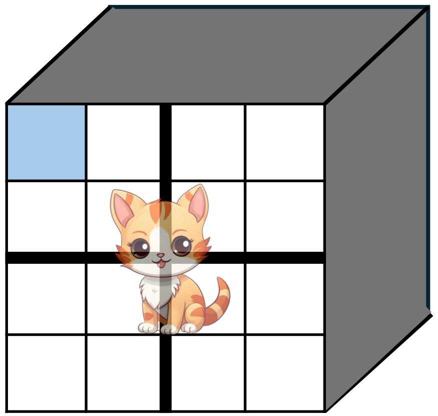
Advantage: Reduces complexity from quadratic to linear

2) Window based Multi-head Self Attention (W-MSA)

Challenge: What if an image is divided between the windows?



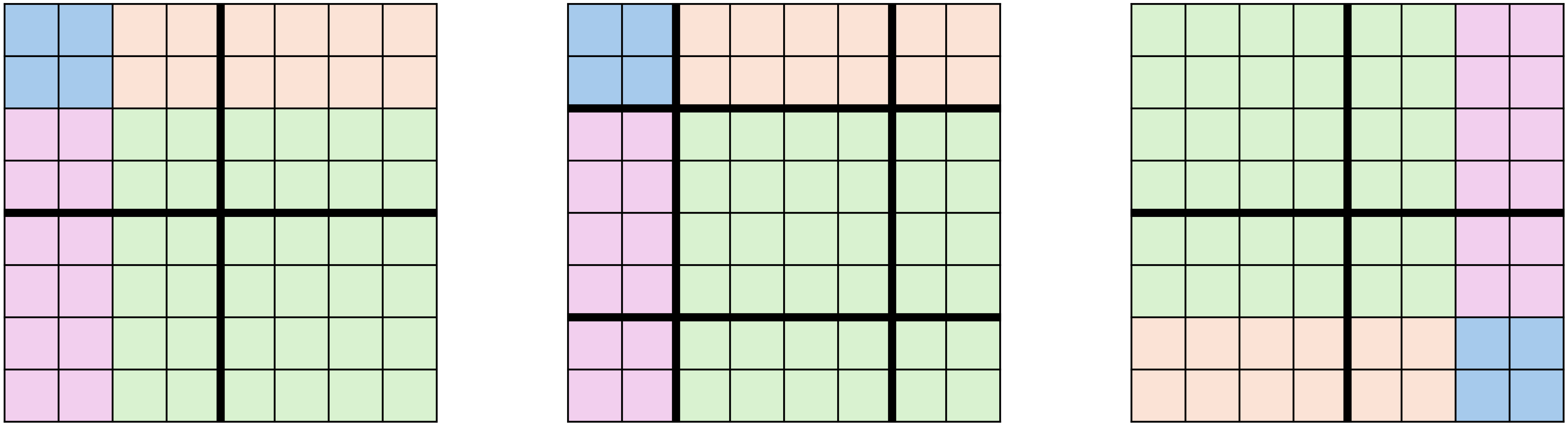
8x8 input



16 Patches

3) Shifted Window based Multi-head Self Attention (SW-MSA)

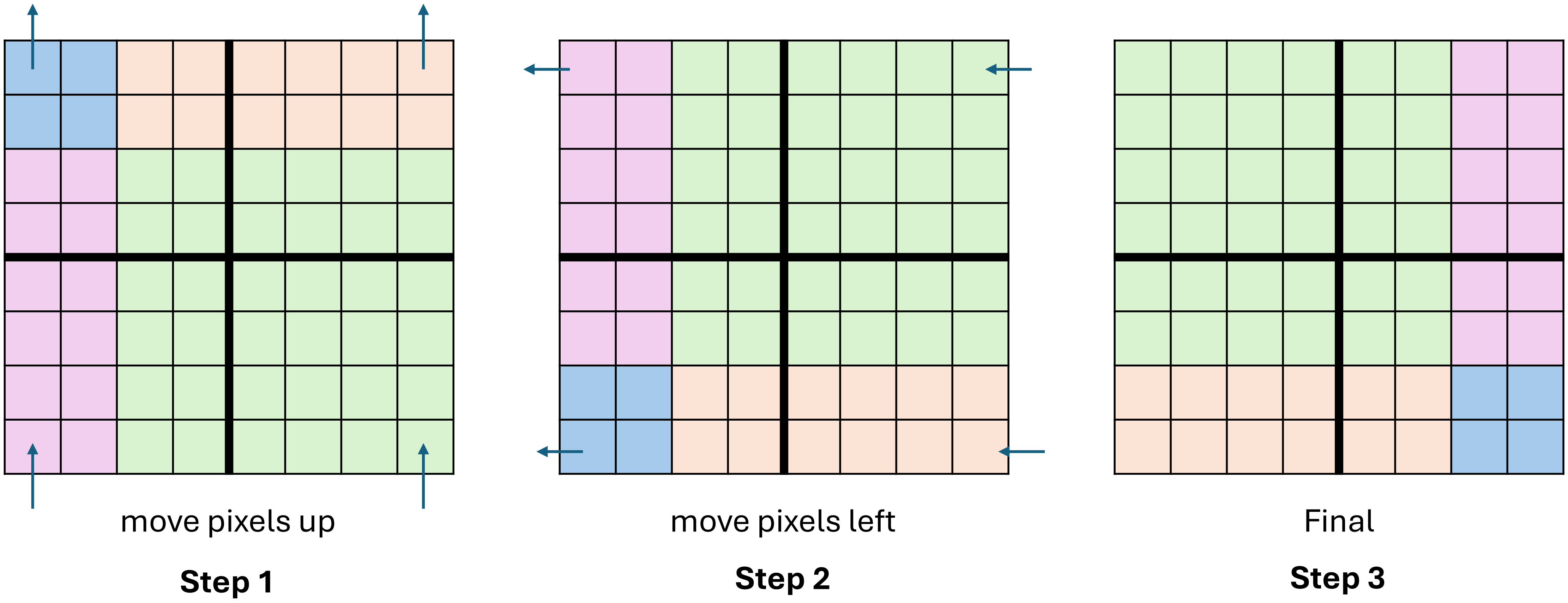
Solution: Shift the windows by half the window size in up and left directions



Before   After   Final

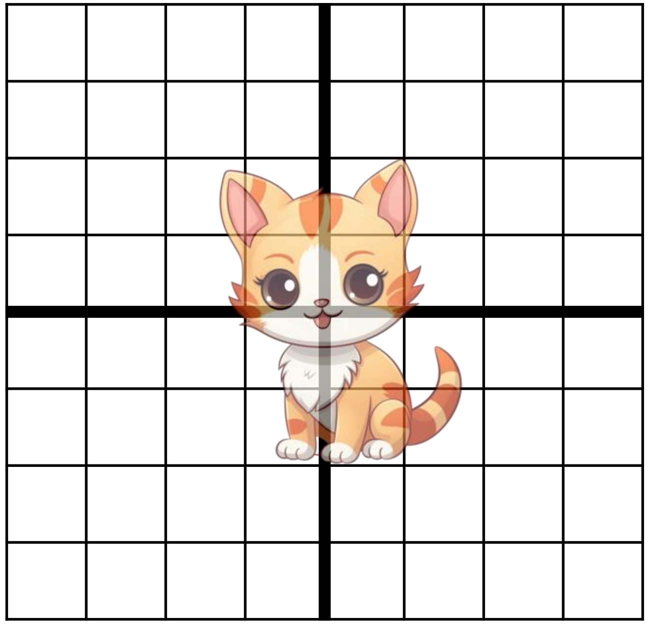
3) Shifted Window based Multi-head Self Attention (SW-MSA)

Simpler way: Move the pixels towards up and left

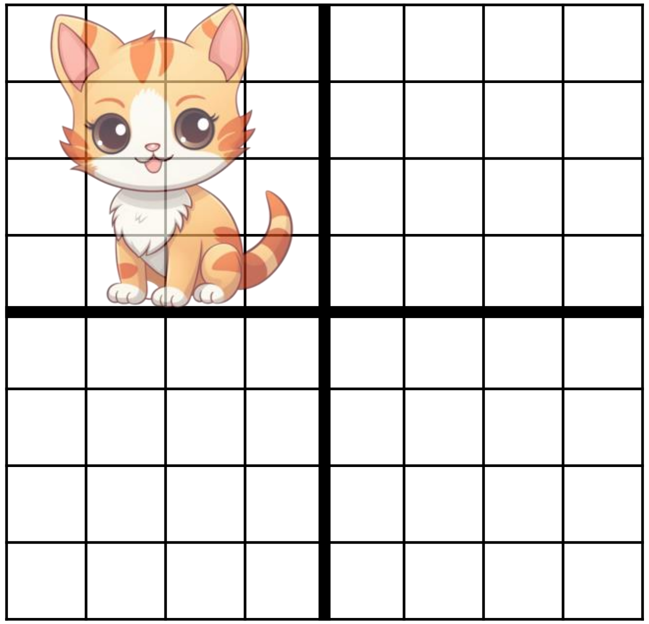


3) Shifted Window based Multi-head Self Attention (SW-MSA)

Solution: Can solve the problems that could occur when the windows are not shifted



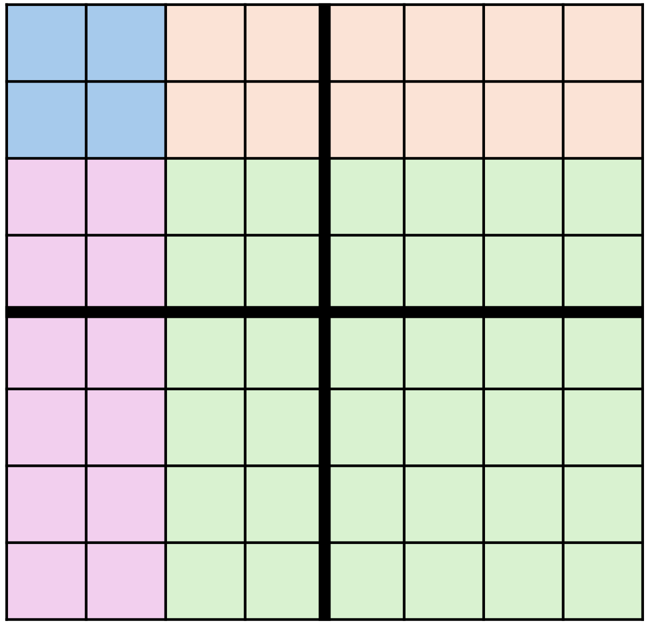
Before Shifted windows



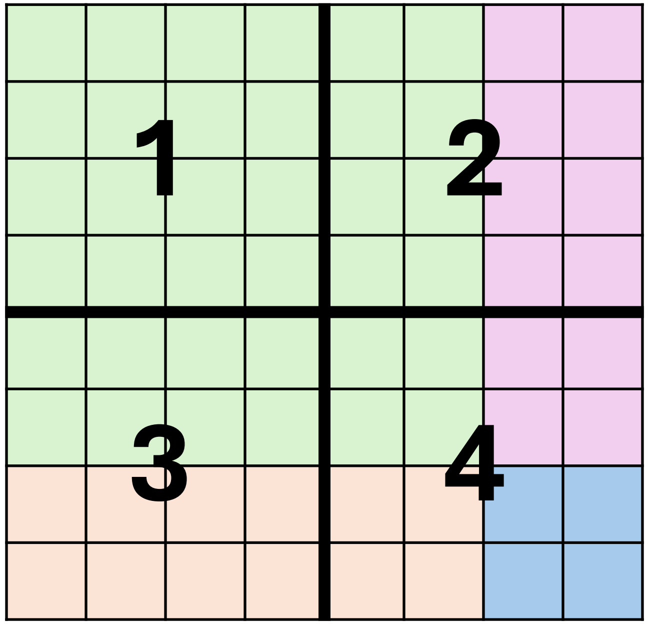
After Shifted windows

3) Shifted Window based Multi-head Self Attention (SW-MSA)

Challenge: After shifting the windows is it justifiable to do self attention within windows?



Initial

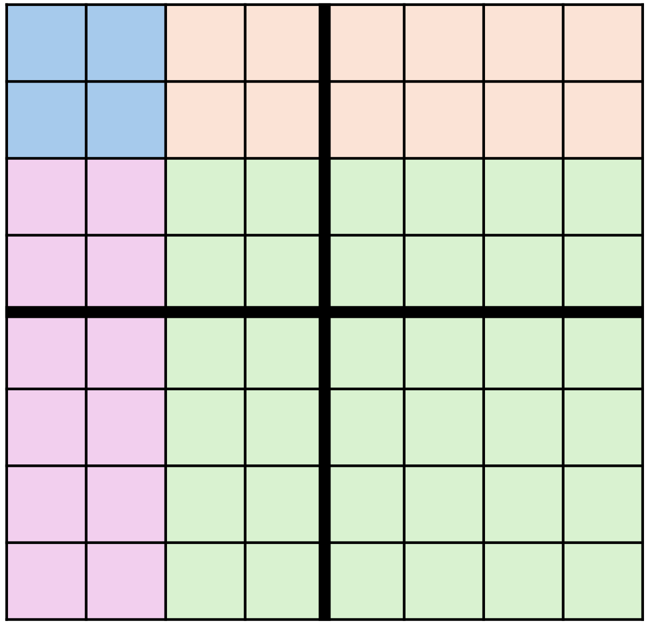


Final

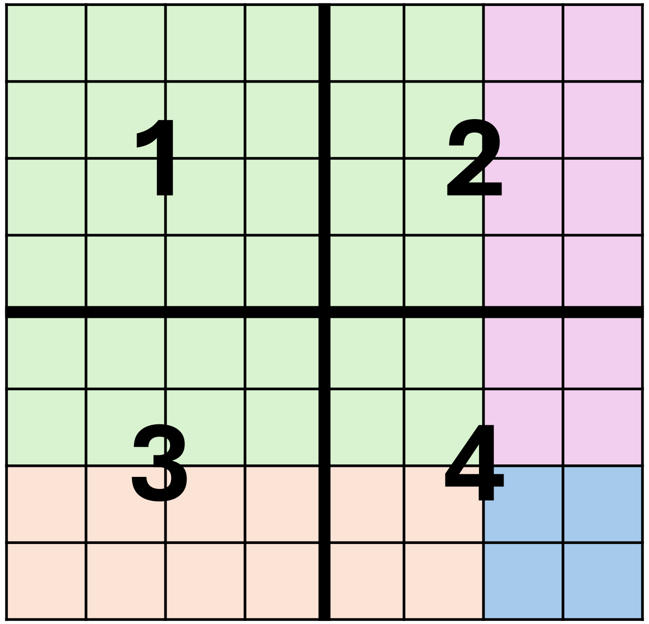
Although in window 1 all the pixels are adjacent, the pixels in window 2,3,4 were not adjacent previously

3) Shifted Window based Multi-head Self Attention (SW-MSA)

Solution: Masked Multi-head self attention



Initial

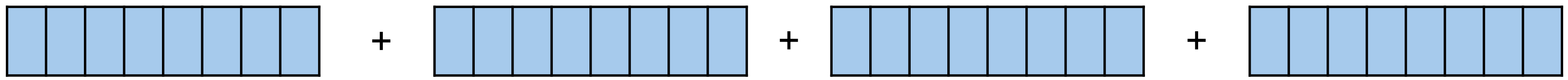


Final

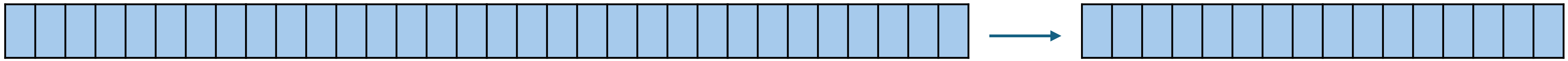
W1: No mask, W2: Mask between green & pink, W3: Mask between green & orange, W4: Mask between all colors

4) Patch Merging

Implementation: Concatenates 4 adjacent patches, each with dimension C, resulting in a vector of size 4C, and passes it through a linear layer to output a vector with dimension 2C.



Concatenation of 4 adjacent vectors



Concatenated vector (4C) given to linear layer

Output vector (2C)

Advantages:

- Learns gradually from local patterns to global patterns
- Fewer tokens passed to deeper layers, improving efficiency and lowering computation
- Concatenation increases feature dimensions, enabling more expressive embeddings

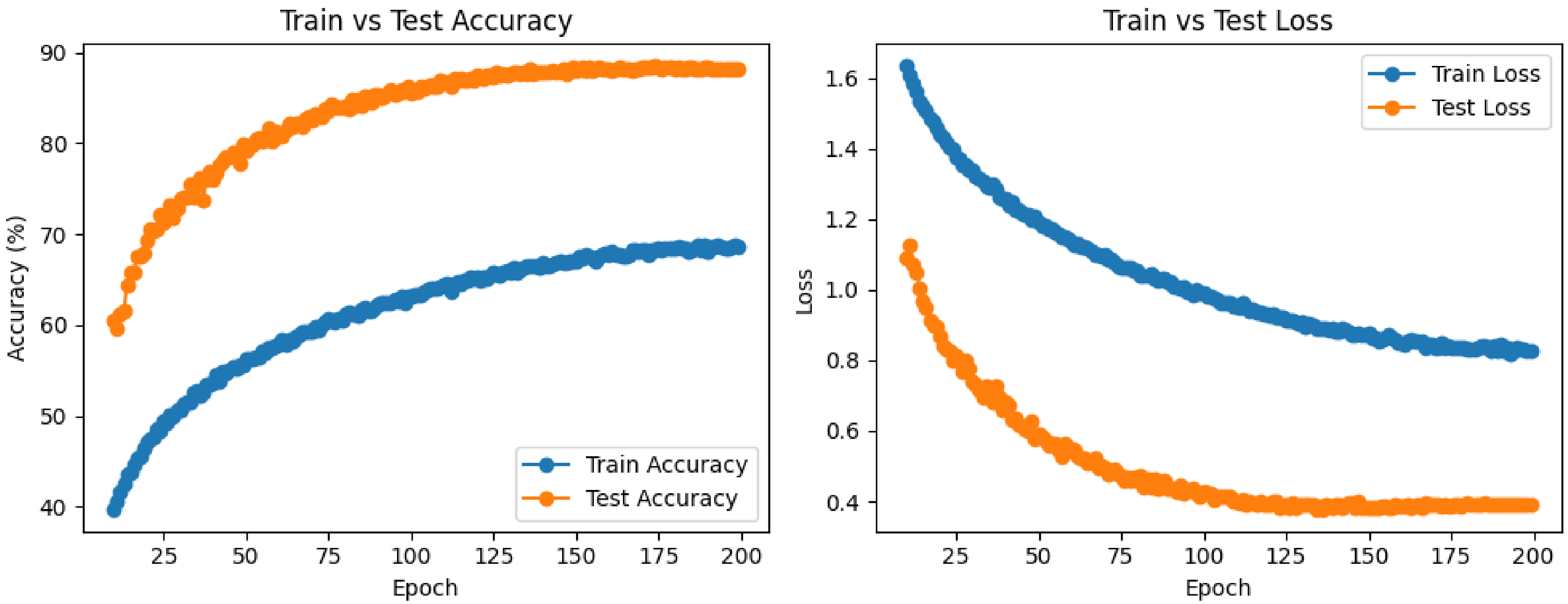
Differences in Implementation

1) **Parameters Flexibility**: In original Swin transformer code, the models are predefined. In Berniwal's implementation although there are predefined variants, it also allows custom configurations of layers, embedding dimensions, number of heads, etc.
2) **Attention masking**: The original Swin uses -100 to block attention, while Berniwal's version uses -inf for a stricter and simpler way to prevent attention across shifts.
3) **Stages**: The official Swin Transformer handles W-MSA and SW-MSA alternation automatically within modular blocks, while Berniwal's version simplifies this for clarity, often requiring manual alternation and customization.

Observation: Berniwal's lightweight and customizable design makes it more efficient and easier to adapt for small datasets like CIFAR

Results

Optimizer = Adam, downscaling factor =(2,2,2,1) with augmentation = True



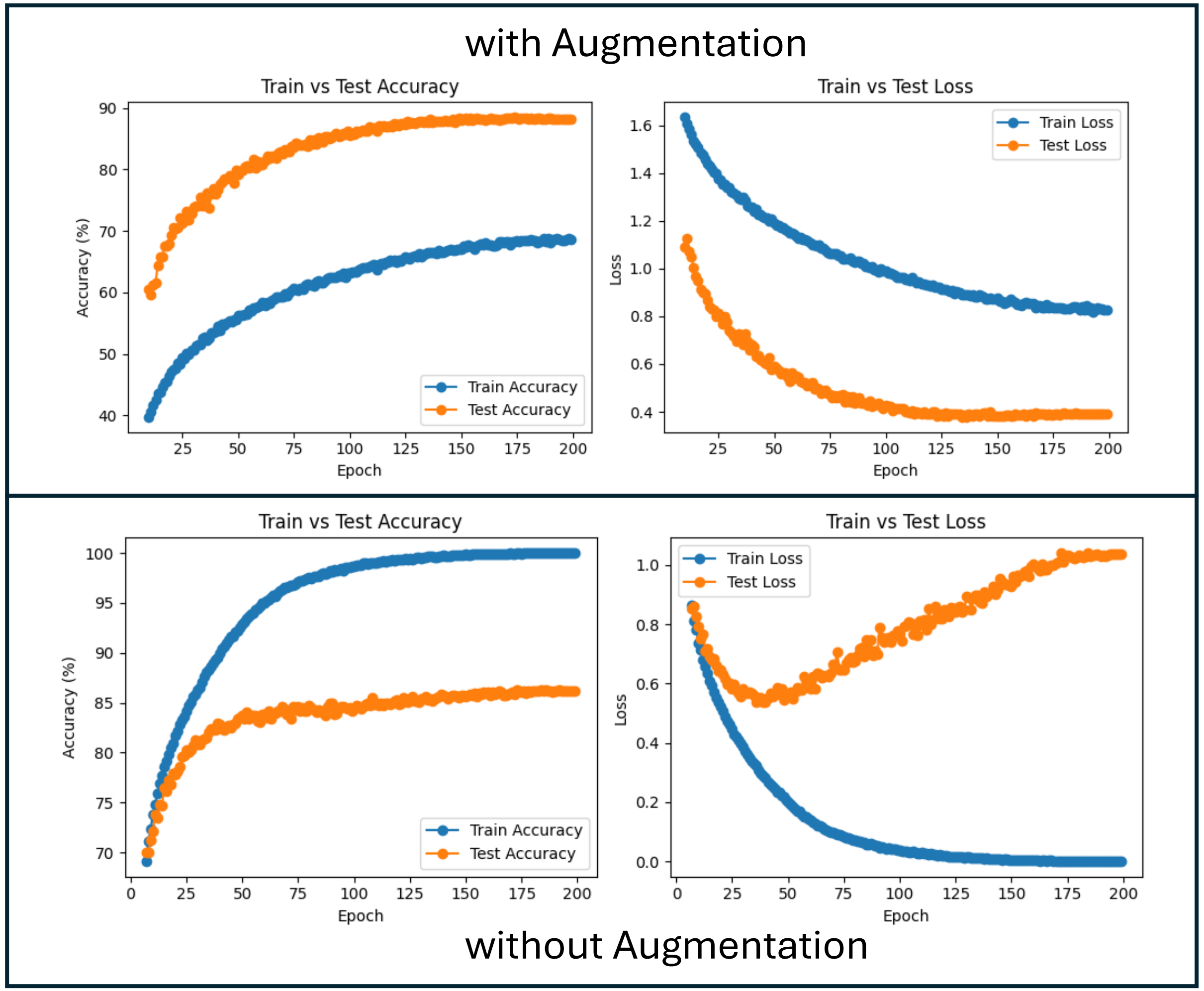
- The model shows better generalization, as indicated by higher test accuracy.
- This improvement may be attributed to robust training with a high degree of data augmentation (M = 14)

Results

Comparison of models **with** and **without augmentation** (Adam, downscaling factor = (2,2,2,1))

- With augmentation, the model generalizes better and achieves higher test accuracy
- Without augmentation, the model overfits
- Augmentation helps reduce the gap between train and test performance

Observation: Without augmentation, the model memorizes training data but performs worse on unseen data



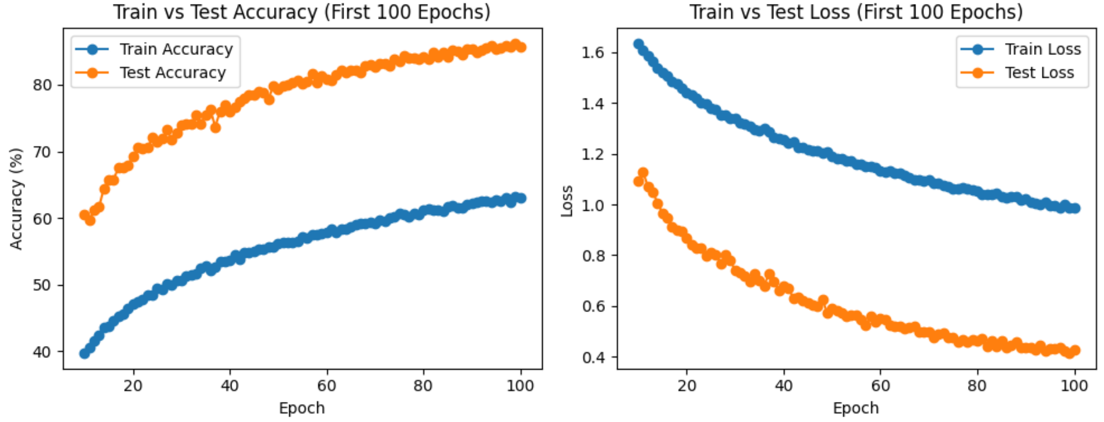
Results

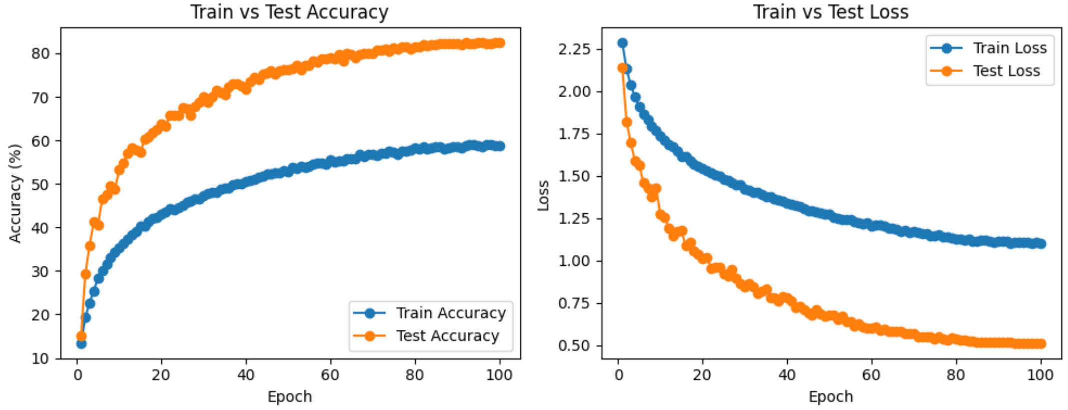
Comparison of models with downcaling factor (**2,2,2,1**) and (**2,2,1,1**) (Adam and with augmentation)

- Model 1 achieves higher test accuracy, indicating better learning capacity
- Model 2 converges faster but reaches lower accuracy, suggesting limited learning
- More aggressive downscaling in Model 1 allows capturing global features effectively

Observation: Progressive downscaling improves model performance by efficient hierarchical feature learning

Model 1- Downscaling factor: (2,2,2,1)





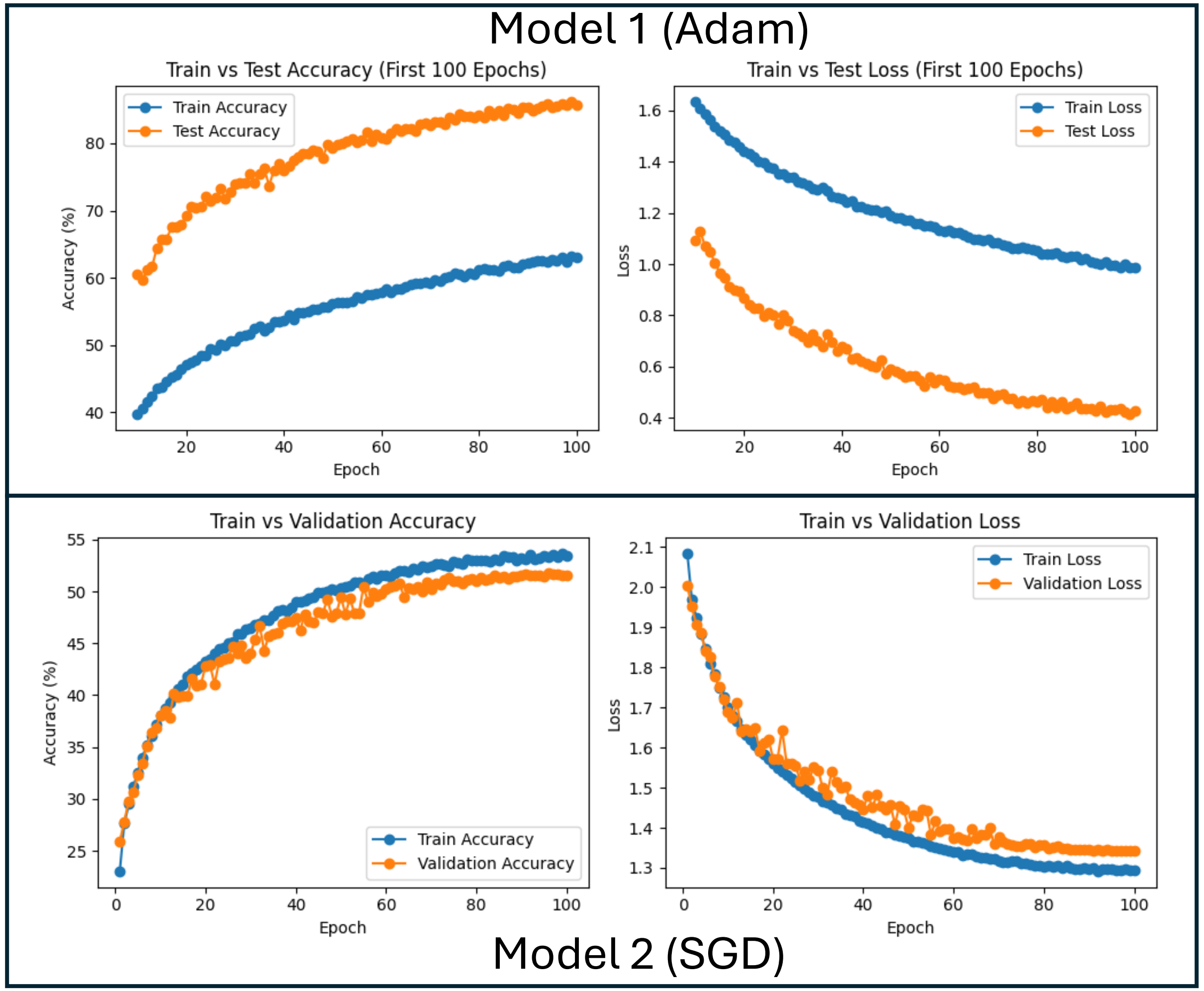
Model 2-Downscaling factor: (2,2,1,1)

Results

Comparison of models with different optimizers: **Adam** and **SGD**

- Model 1 achieves significantly higher test accuracy than Model 2, indicating better generalization.
- Model 2 has slower learning and lower accuracy, suggesting possible underfitting.
- Loss curves in Model 2 converges earlier, while Model 1 continues to improve steadily

Observation: The Adam optimizer helps the model learn better and faster than SGD in this case



Results:

Comparison of Different models for 100 epochs

S.No	Optimizer	Downscaling factor	Augmentation	Train_Acc	Test_Acc	Train_Loss	Test_Loss
1	Adam	(2,2,2,1)	Yes	63.09	85.64	0.99	0.42
2	Adam	(2,2,2,1)	No	98.64	84.5	0.03	0.76
3	Adam	(2,2,1,1)	Yes	58.7	82.37	1.1	0.5
4	SGD	(2,2,2,1)	No	53.37	51.15	1.29	1.34

Thank You