A Comparative Study of CNNs and Swin Transformers on Traffic Sign Classification For CS 7150

An Analysis of "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows"

The Swin Transformer paper introduces a powerful and scalable vision architecture that extends the success of transformers to a wide range of computer vision tasks. Unlike standard Vision Transformers (ViT), which use global self-attention, Swin employs a hierarchical design with non-overlapping windows, making it more computationally efficient. The key innovation lies in the **shifted window mechanism**, which allows cross-window connections and improves information flow across local regions.

The architecture progresses through four stages, gradually reducing spatial dimensions via **Patch Merging** and increasing the number of channelsâ€”mirroring the design philosophy of CNNs. This allows the model to capture fine-grained features in earlier stages and more abstract representations in deeper layers.

The paper demonstrates strong results on benchmark datasets in image classification, object detection, and semantic segmentation, showing the Swin Transformerâ€™s potential as a general-purpose backbone. Its modular nature and competitive performance suggest a promising alternative to traditional convolutional networks.

Private Investigator

Ze Liu
Microsoft Research Asia (MSRA)
Focus: Vision, Hierarchical Modeling

Han Hu
MSRA
Focus: 3D Vision, Transformers

Yue Cao
MSRA
Focus: Vision Research Leadership

Yutong Lin
MSRA
Focus: Efficient Visual Learning

Swin Transformer was proposed by a group of leading researchers at Microsoft Research Asia (MSRA). Their goal was to design a general-purpose hierarchical vision transformer that scales effectively across classification, detection, and segmentation tasks. The baseline CNN, ResNet, was also originally developed at MSRA.

Diagrammer

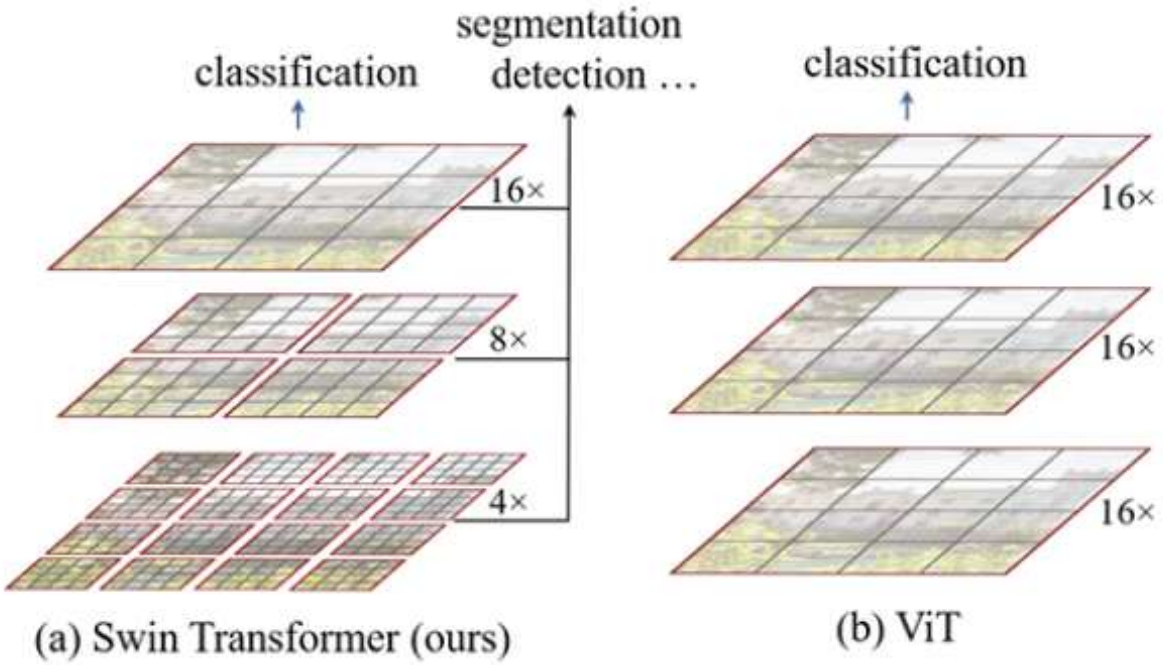


Figure 1: Comparison of hierarchical Swin Transformer vs. flat ViT.

Unlike ViT which applies global attention at a fixed scale, Swin Transformer processes images in a hierarchical fashionâ€”starting with small patches and gradually merging them. This enables Swin to support not only classification but also detection and segmentation.

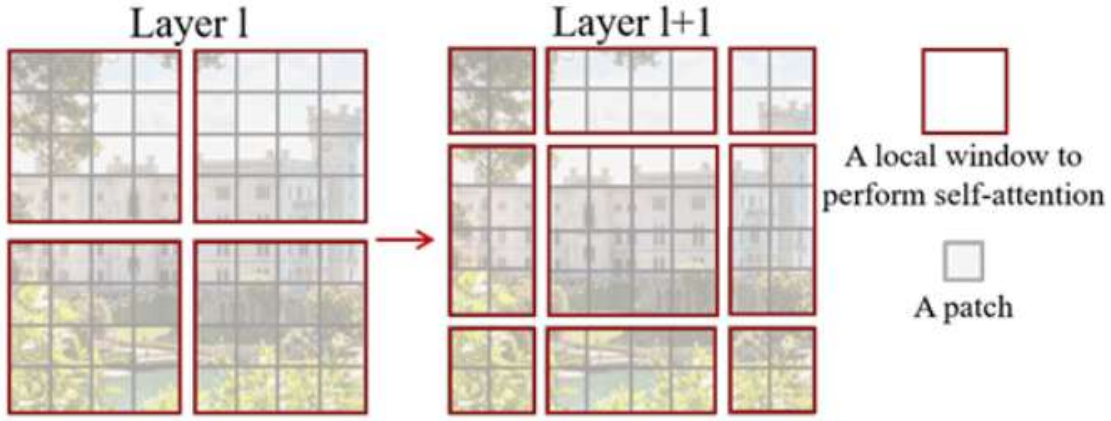


Figure 2: Shifted window self-attention across layers.

Swin Transformer performs local attention within windows in one layer, and then shifts the windows in the next. This allows information exchange between neighboring regions while keeping the computation efficient.

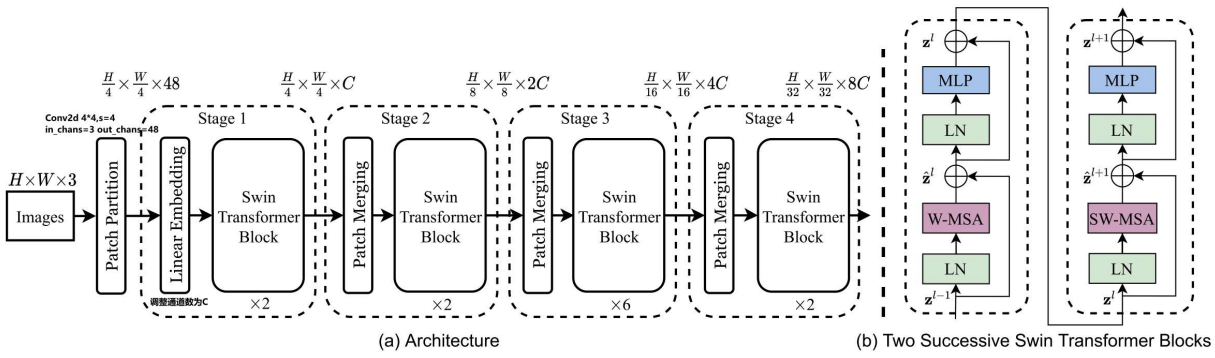


Figure 3. (a) The architecture of a Swin Transformer (Swin-T); (b) two successive Swin Transformer Blocks (notation presented with Eq. (3)). W-MSA and SW-MSA are multi-head self attention modules with regular and shifted windowing configurations, respectively.

Figure 3: The overall architecture of the Swin Transformer.

The model adopts a hierarchical design with four stages in total. Except for the first stage, each subsequent stage begins with a **Patch Merging** layer that downsamples the input feature map, reducing its resolution. This allows the model to progressively enlarge the receptive field - similar to how CNNs operate - enabling it to capture increasingly global information.

At the beginning of the model, a **Patch Partition** operation is applied - this is equivalent to the **Patch Embedding** in ViT. The input image of size $H \times W \times 3$ is divided into non-overlapping patches of size 4×4, producing $\frac{H}{4} \times \frac{W}{4}$ patches, each with a flattened vector of dimension $4 \times 4 \times 3 = 48$. Thus, the embedded feature sequence has shape:

$$\mathbf{X}_0 \in \mathbb{R}^{\frac{H}{4} \cdot \frac{W}{4} \times 48}$$

In the first stage, a **Linear Embedding** layer projects these 48-dimensional vectors into a hidden dimension C, where:

$$\mathbf{X}_1 = \mathbf{X}_0 \cdot \mathbf{W}_{\text{embed}} \in \mathbb{R}^{\frac{H}{4} \cdot \frac{W}{4} \times C}$$

Each subsequent stage (except the first) starts with **Patch Merging**, which concatenates features from a 2Ã—2 region and applies a linear projection:

$$\text{PatchMerging}(x) = \text{Linear}(\text{Concat}(x_{i,j}, x_{i,j+1}, x_{i+1,j}, x_{i+1,j+1}))$$

This reduces the resolution by a factor of 2 and doubles the channel dimension, producing a more abstract and compact representation for deeper stages.

Inside each stage, several **Swin Transformer Blocks** are applied. As shown in the diagram, each block contains:

- **Layer Normalization** before each major operation
- **Window-based Multi-head Self-Attention (W-MSA)** computing attention within non-overlapping windows
- **Shifted Window Attention (SW-MSA)** to enable cross-window connections
- **Multilayer Perceptron (MLP)** with two linear layers and a GELU activation

The W-MSA and SW-MSA operations can be formally described as:

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V$$

where Q, K, V are linear projections of the input X within each window and d is the dimensionality of the keys.

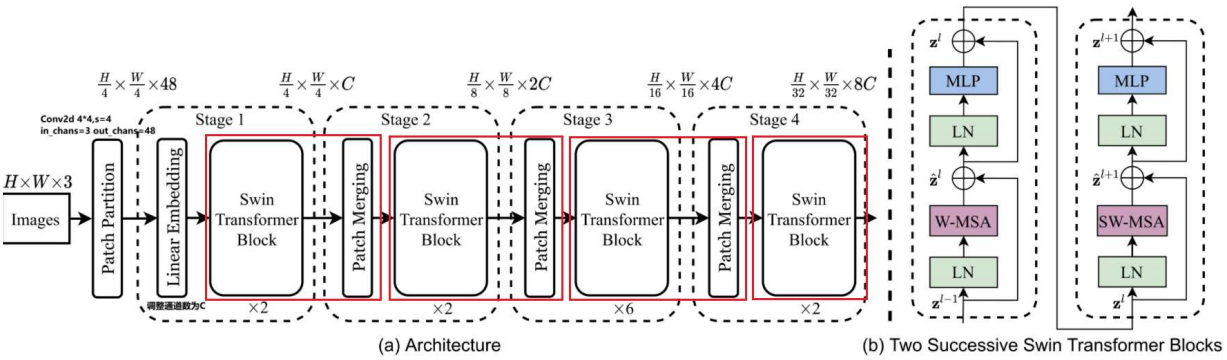


Figure 4: Understanding the framework from a code perspective.

In the Swin Transformer architecture, **Patch Merging** is placed at the end of each stage. The input first goes through a series of **Swin Transformer Blocks** for feature extraction, followed by downsampling via Patch Merging. The final stage does not perform any further downsampling. Instead, its output is passed directly to a fully connected layer, where it is used to compute the loss against the target labels.

```
            # window_size=7
            # input_batch_image.shape=[128,3,224,224]
            class SwinTransformer(nn.Module):
                def __init__(...):
                    super().__init__()
                    ...
                    # absolute position embedding
                    if self.ape:
                        self.absolute_pos_embed = nn.Parameter(torch.zeros(1, num_patches, embed_dim))

                    self.pos_drop = nn.Dropout(p=drop_rate)

                    # build layers
                    self.layers = nn.ModuleList()
                    for i_layer in range(self.num_layers):
                        layer = BasicLayer(...)
                        self.layers.append(layer)

                    self.norm = norm_layer(self.num_features)
                    self.avgpool = nn.AdaptiveAvgPool1d(1)
                    self.head = nn.Linear(self.num_features, num_classes) if num_classes > 0 else
nn.Identity()

                def forward_features(self, x):
                    x = self.patch_embed(x) # Patch Partition
                    if self.ape:
                        x = x + self.absolute_pos_embed
                    x = self.pos_drop(x)

                    for layer in self.layers:
                        x = layer(x)

                    x = self.norm(x)  # Batch_size Windows_num Channels
                    x = self.avgpool(x.transpose(1, 2))  # Batch_size Channels 1
                    x = torch.flatten(x, 1)
                    return x

                def forward(self, x):
                    x = self.forward_features(x)
                    x = self.head(x) # self.head => Linear(in=Channels,out=Classification_num)
                    return x
```

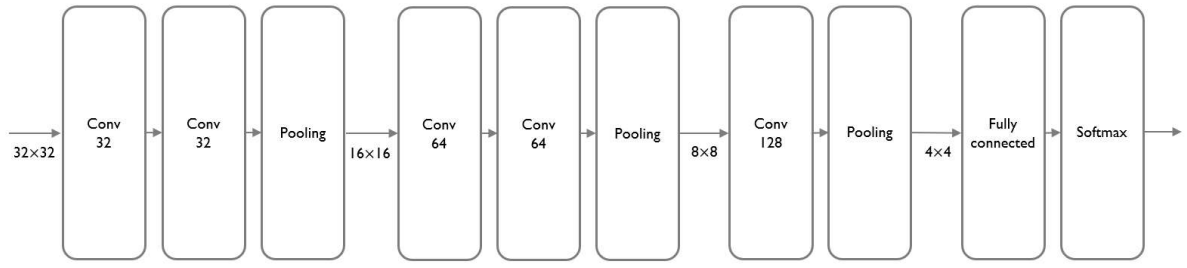


Figure 5: The overall architecture of the CNNs.

We implemented a Convolutional Neural Network (CNN) to classify traffic signs into 43 categories using the GTSRB dataset. The input is a 32 * 32 RGB image passed through five convolutional stages. Each stage uses 3 * 3 filters, ReLU activation, batch normalization, and dropout (25%) to prevent overfitting. Max pooling is applied to reduce spatial dimensions, eventually down to 4 * 4.

After feature extraction, the output is flattened and passed through a fully connected layer with 128 units and 50% dropout. The final layer outputs a 43-dimensional vector, transformed into class probabilities using softmax.

The model was trained for 15 epochs with a batch size of 32. It combines strong feature extraction, dimensionality reduction, and regularization, making it well-suited for robust traffic sign recognition.

Figures based on Liu et al., 2021 (ICCV): "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows".

Reviewer Comments

Overall Recommendation: Accept (7/10)

Swin Transformer presents a unified and scalable vision backbone, designed around hierarchical modeling and computationally efficient window-based self-attention.

Strengths

- **Shifted Window Attention** offers a clever balance between local and global context modeling, dramatically improving efficiency without sacrificing accuracy.
- Demonstrates **state-of-the-art performance** across diverse tasks - image classification, object detection, and semantic segmentation.
- The architecture is **modular and extensible**, making it well-suited for future advancements in vision-based AI systems.

Weaknesses

- The paper lacks a **deeper theoretical analysis** of the shifted window strategy and its limitations.
- Compared to other emerging transformer variants, Swin's conceptual novelty might be perceived as more incremental.

Experiment

Dataset: GTSRB

The **German Traffic Sign Recognition Benchmark (GTSRB)** is a widely-used dataset for multi-class image classification, designed for developing and evaluating traffic sign recognition models. It contains over **50,000 images** covering **43 traffic sign classes**, with variations in scale, lighting, rotation, and occlusionâ€”making it a challenging benchmark for real-world vision systems.

- **Image count:** 51,839
- **Classes:** 43 different traffic signs
- **Image size:** Varies, mostly resized to 32*32 or 64*64 in preprocessing
- **Applications:** Used in autonomous driving, traffic sign recognition, visual robustness testing

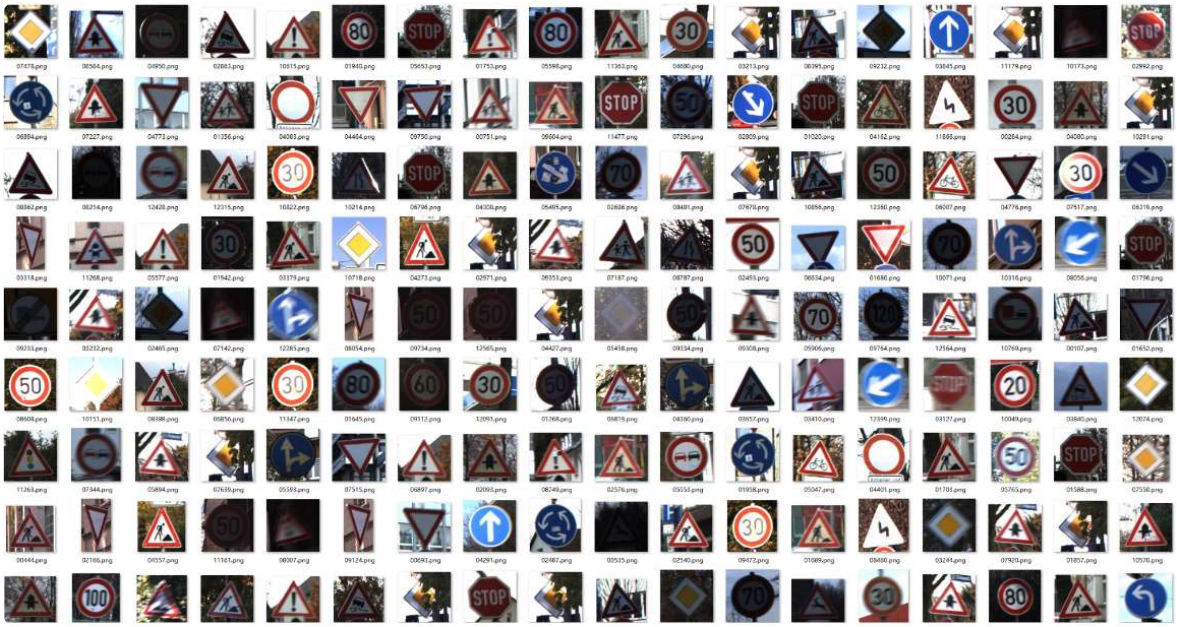
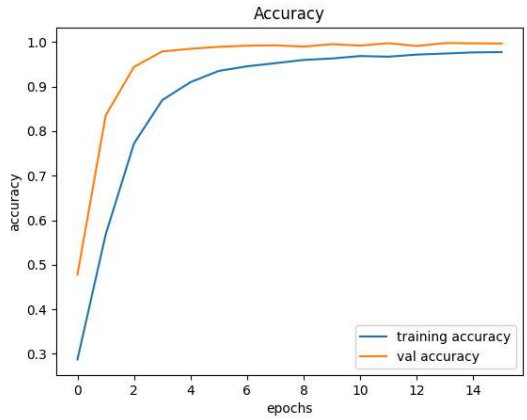


Figure: Sample images from the GTSRB dataset showing various traffic signs under different lighting and angles.

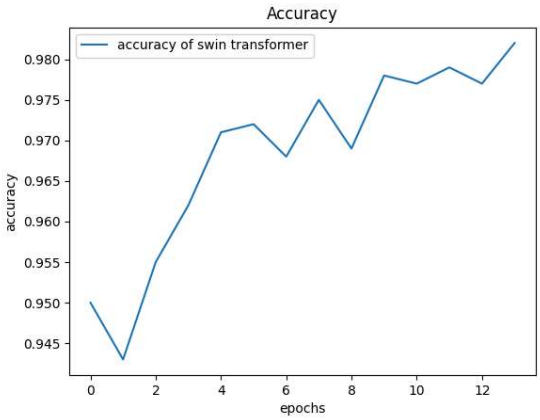
We implemented Swin-Tiny and ResNet-50 on the GTSRB dataset under the same conditions. The Swin Transformer outperformed ResNet in accuracy and convergence. We also investigated how changing the Swin window size affects model performance, with surprising insights on pattern sensitivity.

```
[2025-04-14 18:44:37 test](main.py 98): INFO number of params: 86937891
[2025-04-14 18:44:37 test](main.py 101): INFO number of GFLOPs: 11.75991808
All checkpoints founded in output/test/default: []
[2025-04-14 18:44:38 test](main.py 135): INFO no checkpoint found in output/test/default, ignoring auto resume
[2025-04-14 18:44:38 test](utils.py 19): INFO ==============> Resuming form output/the_best_result_98.16.pth.....................
[2025-04-14 18:44:38 test](utils.py 26): INFO <All keys matched successfully>
[2025-04-14 18:44:40 test](main.py 272): INFO Test: [0/99]      Time 1.109 (1.109)      Loss 0.1708 (0.1708)    Acc@1 97.656 (97.656)   Acc@5 100.000 (100.000) Mem 2815MB
[2025-04-14 18:44:43 test](main.py 272): INFO Test: [10/99]     Time 0.318 (0.405)      Loss 0.1639 (0.2159)    Acc@1 99.219 (97.656)   Acc@5 100.000 (99.716)  Mem 2816MB
[2025-04-14 18:44:46 test](main.py 272): INFO Test: [20/99]     Time 0.325 (0.367)      Loss 0.1992 (0.2044)    Acc@1 97.656 (98.065)   Acc@5 100.000 (99.740)  Mem 2816MB
[2025-04-14 18:44:50 test](main.py 272): INFO Test: [30/99]     Time 0.316 (0.354)      Loss 0.1600 (0.2065)    Acc@1 99.219 (98.110)   Acc@5 100.000 (99.672)  Mem 2816MB
[2025-04-14 18:44:53 test](main.py 272): INFO Test: [40/99]     Time 0.326 (0.348)      Loss 0.1958 (0.2031)    Acc@1 98.438 (98.152)   Acc@5 100.000 (99.695)  Mem 2816MB
[2025-04-14 18:44:56 test](main.py 272): INFO Test: [50/99]     Time 0.353 (0.345)      Loss 0.1733 (0.2002)    Acc@1 98.438 (98.238)   Acc@5 100.000 (99.740)  Mem 2816MB
[2025-04-14 18:45:00 test](main.py 272): INFO Test: [60/99]     Time 0.323 (0.346)      Loss 0.2202 (0.2005)    Acc@1 97.656 (98.207)   Acc@5 100.000 (99.744)  Mem 2816MB
[2025-04-14 18:45:03 test](main.py 272): INFO Test: [70/99]     Time 0.347 (0.345)      Loss 0.2142 (0.2030)    Acc@1 97.656 (98.118)   Acc@5 100.000 (99.714)  Mem 2816MB
[2025-04-14 18:45:07 test](main.py 272): INFO Test: [80/99]     Time 0.328 (0.345)      Loss 0.2192 (0.2035)    Acc@1 97.656 (98.139)   Acc@5 100.000 (99.720)  Mem 2816MB
[2025-04-14 18:45:10 test](main.py 272): INFO Test: [90/99]     Time 0.322 (0.344)      Loss 0.1517 (0.2030)    Acc@1 100.000 (98.189)  Acc@5 100.000 (99.717)  Mem 2816MB
[2025-04-14 18:45:12 test](main.py 278): INFO  * Acc@1 98.155 Acc@5 99.715
[2025-04-14 18:45:12 test](main.py 140): INFO Accuracy of the network on the 12630 test images: 98.2%
```

Accuracy Comparison

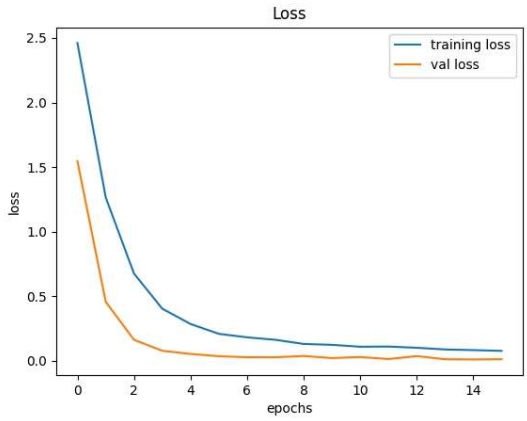


Training and validation accuracy of ResNet-50 (CNN). It shows fast convergence and high validation accuracy approaching 97% after just a few epochs.

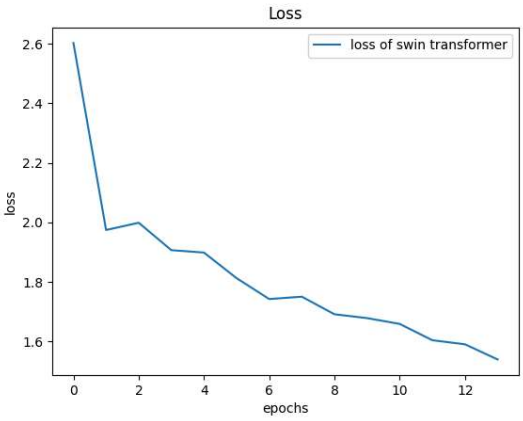


Accuracy curve of Swin Transformer shows stable improvements, reaching ~98.3% accuracy, with less variance after epoch 5. Suggests good generalization and robustness.

Loss Comparison



Loss curves of ResNet-50 show consistent decline, and low validation loss after epoch 3, indicating strong fit to data with low overfitting risk.



Loss curve of Swin Transformer also decreases steadily. While not as steep initially, it stabilizes well, suggesting a strong capacity for noise-tolerant optimization.

Overall, both models perform well, but Swin Transformer shows comparable or slightly better generalization ability. Its hierarchical architecture and shifted windows contribute to stable convergence with fewer parameters.

Download Our Trained Model

We provide the best-performing Swin Transformer model trained on the GTSRB dataset. If you'd like to run your own experiments or verify our results, you can download the model from the link below:

Download Swin - Tranformer Weight file (.pth) [with top 1 accuracy reaches 98.3 %]

After downloading, you can place the `.pth` file in Swin - Transformer code part at project folder /Swin - Transformer - main / output and load it using PyTorch for inference or further training.

References

[1] Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., ... & Guo, B. (2021). *Swin Transformer: Hierarchical Vision Transformer using Shifted Windows*. In Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), pp. 10012â€“10022.

[2] Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., & Zagoruyko, S. (2020, August). *End-to-end object detection with transformers*. In European conference on computer vision (pp. 213-229). Cham: Springer International Publishing.

[3] LeCun, Y., Bengio, Y., & Hinton, G. (2015). *Deep learning* . nature, 521(7553), 436-444.

[4] Simonyan, K., & Zisserman, A. (2014). *Very deep convolutional networks for large-scale image recognition*. arXiv preprint arXiv:1409.1556.

Team Members

Zefeng Zhao, Zuyu Guo.

Acknowledgment

We would like to express our sincere gratitude to **Professor David Bau** for his guidance throughout this course. His thoughtful lectures and insightful discussions helped us better understand modern deep learning models and inspired us to explore the capabilities of Swin Transformer in a real-world vision task.