Friedrich-Alexander-Universität

Pattern Recognition Lab

Evaluation of Single-Head Vision Transformer (SHViT) for Volumetric Semantic Segmentation in Industrial CT Scans

Bachelor's Thesis in Data Science

submitted
by

ChangGeng Drewes

born 16.02.2000 in Yokohama, Japan

Written at

Lehrstuhl für Mustererkennung (Informatik 5)
Department Informatik
Friedrich-Alexander-Universität Erlangen-Nürnberg.

Advisor: Martin Leipert

Started: 18.06.2025

Finished: 18.08.2025

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

ChangGeng Drewes

Übersicht

Diese Bachelorarbeit befasst sich mit der Anpassung und Erweiterung von 2D-Vision-Transformer-Modellen, insbesondere SegFormer mit Mix-Vision Transformer (MVT) und Single-Head Vision Transfomer(SHViT), für die volumetrische 3D-Segmentierung. Der Fokus liegt auf der präzisen Segmentierung von 3D-Schuhscans. Als zentrales Qualitätsmaß wurde die »F1/Dice«-Metrik verwendet.

Die Experimente zeigen, dass die Anzahl der Convolution-Layer im SHViT-Modell erheblichen Einfluss auf die Segmentierungsgenauigkeit und den GPU-Speicherverbrauch hat. Die beste Balance wurde mit zwei Convolution-Layern im PatchEmbed-Block erzielt. Um die Speicherbelastung durch klassische Attention-Mechanismen mit $O(n^2)$-Komplexität zu verringern, wurde ein Central Self-Attention-Ansatz auf Basis eines Sliding-Window-Verfahrens implementiert. Dieser reduziert den Speicherbedarf je nach Konfiguration bis zu 54%, ohne die Modellleistung wesentlich zu beeinträchtigen. Zusätzlich konnten kleinere Kernelgrößen im Attention-Block die Genauigkeit weiter steigern, da damit feine Strukturen besser erfasst werden können.

Die besten Ergebnisse wurden mit einem SHViT-Modell mit Central Self-Attention und optimierter Parametrierung erzielt, bei dem der Speicherbedarf bei unter 24 GB GPU-Speicherbedarf liegt. Abschließend wird angeregt, sich in zukünftigen Arbeiten weiter mit der Optimierung von Attention-Mechanismen zur Reduzierung des Speicherbedarfs zu befassen und weitere Vision-Transformer-Backbones zur Verbesserung der Segmentierungsgenauigkeit auszuprobieren.

Abstract

This bachelor's thesis examines the adaptation and extension of 2D Vision Transformer models, specifically SegFormer with Mix-Vision Transformer (MVT) and Single-Head Vision Transfomer (SHViT), for volumetric 3D segmentation. The main emphasis is on accurately segmenting 3D shoe scans. The »F1/Dice«-metric was used as the primary performance measure.

Experiments show that the number of convolutional layers in the SHViT model significantly affects both segmentation accuracy and GPU memory usage. The best trade-off was achieved with two convolutional layers in the PatchEmbed block. To mitigate the high memory demands of conventional attention mechanisms with $O(n^2)$ complexity, a central self-attention approach based on a sliding-window method was implemented. Depending on the configuration, this reduces memory requirements by up to 54% without significantly affecting model performance. Furthermore, smaller kernel sizes in the attention block led to improved accuracy, as they allowed for better capture of fine structural details.

The best results were obtained with a SHViT model using central self-attention (sliding-windows) with optimized backbone and decoder parametrization, while keeping GPU memory usage below 24 GB. Finally, the thesis suggests that future work should further investigate the optimization of attention mechanisms to reduce memory requirements and explore additional Vision Transformer backbones to enhance segmentation accuracy.

Acknowledgments

I would like to express my gratitude towards everyone who supported me during my thesis project.

First, I would like to thank my advisor Martin Leipert for giving me this interesting topic, and the precious opportunity to do my bachelor's thesis in the Pattern Recognition Lab at Friedrich-Alexander-University. His mentorship and suggestions during this research project have been instrumental in shaping the direction and quality of my work. His encouragement and insightful advice have been a constant source of motivation and confidence. A special mention and heartfelt gratitude to Prof. Dr.-Ing. Maier for generating this opportunity which allowed me to grow in the professional field.

Lastly and most importantly, I thank my family for their constant encouragement without this assignment would not be possible.

Thank you all for making this thesis possible.

ChangGeng Drewes

Contents

Chapter 1

Introduction

Deep learning has dramatically advanced image analysis, particularly through Convolutional Neural Networkss (CNNs), but recently Vision Transformers (ViTs) have emerged as a powerful alternative by modeling long-range dependencies via self-attention. Dosovitskiy et al. [Dos+21] showed that a pure Vision Transformer operating on sequences of image patches can achieve state-of-the-art image classification performance, rivaling CNNs when pre-trained on large datasets. This success has spurred the adaptation of Transformer-based architectures to image segmentation tasks.

Transformers have been applied to three-dimensional (3D) medical image segmentation, often in U-Net-style architectures to capture global volumetric context. For example, Hatamizadeh et al. [Hat+22] introduced UNEt TRansformers (UNETR), which treats a 3D volume as a sequence of patches fed into a Transformer encoder; these global features are then integrated into a CNN decoder via skip connections in a U-shaped design. This increases the Transformers ability to learn rich, multiscale representations of the entire volume. Tang et al. [Tan+22] similarly proposed Swin UNETR, employing a hierarchical Swin Transformer encoder for 3D segmentation of medical scans. Both approaches attained competitive results on benchmark datasets by combining Transformers' global attention with CNN decoders for fine localization.

Perera et al. [Per+24] presented SegFormer3D, a lightweight volumetric Transformer that computes attention across multiscale features in a purely attention-based hierarchy. SegFormer3D uses an all-Multi-Layer Perceptron (MLP) decoder and achieves performance on par with much larger models on tasks like multi-organ and brain tumor segmentation. These studies illustrate that Vision Transformer architectures can effectively capture 3D

contextual information, often with far fewer parameters or less computation than traditional CNN models.

1.1 Background and Motivation

The growing complexity and volume of 3D data underscore the need for efficient segmentation methods. Conventional two-dimensional (2D) approaches often fall short, unable to fully utilize depth information critical for tasks like anatomical mapping or obstacle detection. Recently, several studies have been published that explore the use of Vision Transformers for 3D image segmentation, with a particular focus on applications in the medical domain [Das+25; Gan+25; Jol+24].

This thesis aims to extend the capabilities of ViTs to address unique challenges of 3D image analysis, with a particular emphasis on reducing memory consumption. In doing so, it seeks to bridge the gap between algorithmic efficiency and practical applicability in real-world segmentation tasks.

1.2 Objectives and Scope

The primary objective of this thesis is to optimize Vision Transformer architectures for high-resolution 3D image segmentation, with a focus on reducing their substantial memory requirements that hinder scalability. To counter this, the work introduces architectural modifications, most notably, a central self-based attention mechanism that improves computational efficiency.

These enhancements are evaluated using 3D scans of shoes, providing a practical case study to assess both the effectiveness and feasibility of the proposed approach in real-world scenarios.

While the primary application is footwear segmentation, the methods developed are broadly applicable to other domains where accurate 3D data processing is essential, such as medical imaging, autonomous driving, industrial inspection, and others.

The scope of this work includes a comparative analysis of various ViT configurations, highlighting the trade-offs between segmentation accuracy and memory efficiency. The findings offer generalizable insights into adapting Transformer-based models for volumetric data, contributing to the advancement of machine learning techniques for 3D analysis.

1.3 Structure of this Thesis

This thesis is structured into several chapters, each addressing a key part of the work. The organization is designed to follow a logical sequence, starting from the methodological background to the evaluation and conclusion. The overview of the structure is as follows:

- Methodology: This section delves into the technicalities of ViT architectures that are evaluated. It elaborates on the modifications made, specifically focusing on novel adaptations such as central self-attention, and provides comprehensive implementation details to convey the methodological framework employed.
- Data Preparation: In this context, the thesis provides in-depth coverage of the data collection processes, preparation steps, and augmentation techniques that are applied to use 3D scans effectively. This chapter underscores the importance of data integrity and variability in ensuring robust model evaluations.
- Results and Discussion: This chapter analyzes model performance, providing insights into the trade-offs between segmentation accuracy and memory efficiency. It evaluates the effectiveness of the proposed adaptations, offering both quantitative comparisons and qualitative discussions of the findings.
- Summary, Conclusion, and Future Work: In this closing section, the thesis summarizes the key research contributions and draws meaningful conclusions from the study. It also discusses potential implications and suggests ways for future research to build upon the basis laid by this thesis.
- Appendices: This portion of the thesis includes additional resources, offering practical insights into the computational environments set up for experiments and access to the source codes of models used. Such resources serve to enhance reproducibility and allow further exploration based on this thesis.

Chapter 2

Methodology

This chapter begins with an overview of different segmentation types, highlighting their characteristics and differences. Then, various Vision Transformer (ViT) architectures are introduced and discussed in detail. The model selected for this thesis is then presented, along with a comprehensive explanation of the modifications made to the attention mechanism, particularly the implementation of central self-attention and the reasoning behind its adoption. Afterward, the evaluation metrics used to assess model performance are described. The chapter concludes with implementation details, describing the model's construction and deployment using Python.

2.1 Image Segmentation

Image segmentation is a subfield of computer vision that focuses on interpreting visual content by grouping regions in images or videos based on their underlying classes, with labels providing the semantic meaning of those classes. Depending on the required level of detail, segmentation is typically categorized into three main types: semantic segmentation, instance segmentation, and panoptic segmentation. It plays a vital role in applications such as autonomous driving, medical image analysis, video surveillance, and image editing [Zho+24]. The following section presents the key differences between semantic, instance, and panoptic segmentation.

2.1.1 Semantic Segmentation

Semantic segmentation refers to the task of assigning a category label to each pixel in an image. This process allows for a high-level understanding of visual scenes by identifying

and localizing various object classes and regions within the image [Zho+18a]. The goal is to produce a dense, per-pixel classification map in which pixels belonging to the same category (e.g., sky, road, person) are grouped together, regardless of object instances. Semantic segmentation is critical for interpreting scene layout and contributes significantly to applications that require detailed comprehension of visual data.

2.1.2 Instance Segmentation

Instance segmentation is the task of detecting and delineating each distinct object instance in an image [Ke+21]. Unlike semantic segmentation, which assigns a category label to every pixel without distinguishing between different objects of the same class, instance segmentation treats each object occurrence as a separate entity. This means it not only classifies each pixel but also associates it with a specific instance of an object. Instance segmentation combines object detection and pixel-wise segmentation to generate high-quality masks that accurately capture the shapes and boundaries of individual objects, even when they overlap. The latter is called mask prediction. Compared to semantic segmentation it is significantly more complex due to the need to handle varying numbers of objects per image and to clearly differentiate between closely located or overlapping instances of the same class.

First, object detectors identify candidate object regions, typically using a region proposal network, and then a segmentation branch generates binary masks that accurately describe the shape of each object instance within its bounding box. The Mask Transformer approach improves upon previous methods by refining coarse mask predictions at the object boundaries using a Transformer-based architecture, enhancing accuracy particularly at fine-grained edges.

To sum up the difference from semantic segmentation is that instance segmentation has to separate and label each individual object, making it inherently more complex and computationally demanding while semantic segmentation treats all objects of the same category as a single class without distinction.

2.1.3 Panoptics Segmentation

Panoptic segmentation is a unified computer vision task that combines the goals of semantic and instance segmentation [Kir+19]. Specifically, it is about assigning a semantic label to each pixel in an image (e.g. road, sky, person, animal) and at the same time distinguishing

between individual object instances for »thing« classes (e.g. separating multiple persons or animals) and treating »stuff« classes (e.g. sky or grass) as amorphous regions without distinct instances.

Panoptic segmentation works by simultaneously generating two outputs: one for instance segmentation and one for semantic segmentation. In the Panoptic Feature Pyramid Network (FPN) model described in the paper [Kir+19], a shared FPN backbone is used to extract multi-scale features. On top of this backbone, a region-based branch (as in Mask R-CNN) is used for instance segmentation, while a lightweight dense-prediction branch is added for semantic segmentation. These two outputs are then merged through a post-processing step to ensure each pixel is assigned exactly one label and, if applicable, an instance ID.

The key difference from instance segmentation is that panoptic segmentation also includes semantic segmentation of stuff classes, thereby providing a complete pixel-level understanding of the entire scene and not just individual objects. In contrast, instance segmentation focuses only on distinguishing individual object instances, typically omitting the background or amorphous regions. Panoptic segmentation addresses both objectives.

2.1.4 Choice of Segmentation Method

Given the availability of annotated 3D training data of shoe scans (see also chapter 3), semantic segmentation is employed in this thesis to enable accurate classification of components within a given volume.

2.2 Survey of Evaluated Vision Transformer Models

Since this work focuses on the implementation and evaluation of a model for 3D semantic segmentation of high-resolution Computed Tomography (CT) scans of packed shoes, the following section presents ViT approaches suitable for this segmentation task.

Selecting an appropriate model architecture is a important step in the design of a efficient deep learning pipeline, particularly when working with limited but memory-intensive volumetric data. Unlike conventional vision tasks with dense and uniformly sized 2D images, the high-resolution voxel-based inputs in this project impose unique computational and architectural constraints. These include a cubic input space, sparse meaningful content, and extremely high memory requirements, which render many established 2D or video-based Transformer models unsuitable in their default configurations.

To evaluate architectural suitability, a detailed analysis of existing Transformer variants, such as Adapting Vision Transformer (AViT) [Du+23], Efficient Vision Transformer (EfficientViT) [Liu+23], Single-Head Vision Transformer (SHViT) [Yun+24], and Temporally Efficient Vision Transformer (TeViT) [Yan+22] are conducted. SHViT was finally selected and analysed in detail, focusing on their structural design, attention mechanisms, and scalability with respect to large 3D inputs.

An important criterion for inclusion or exclusion in the model selection process is not only the runtime performance but also memory efficiency. Equally relevant are the simplicity and architectural transparency of the model, which directly impact the ease of adaptability and extensibility. For instance, the ability to integrate additional components for semantic segmentation with minimal overhead and without entangling core components is essential for adapting a model to specialized tasks. Therefore, models that demonstrate memory scaling and allow for clean architectural structure are prioritized in the subsequent evaluation.

2.2.1 Semantic Segmentation Transformer (SegFormer)

SegFormer is another Vision Transformer (ViT) architecture that employs a series of efficient self-attention blocks within its encoder [Xie+21; Per+24]. The encoder is organized into four hierarchical stages, each designed to progressively reduce spatial resolution while enriching feature representations. The process begins with an overlapping patch embedding mechanism, in which small, overlapping regions are extracted from the input image and linearly projected into feature vectors. In the first stage, the patch kernel size is set to 7 with a stride of 4 and padding of 3; in subsequent stages, the patch kernel size is reduced to 3 with a stride of 2 and padding of 1. This design halves the spatial resolution at each stage while simultaneously increasing the number of feature channels.

Within each stage, the feature map passes through multiple transformer blocks. Each block starts with an efficient self-attention mechanism that reduces the sequence length for keys and values by a fixed factor to drastically cut computational cost. This is immediately followed by a Mix-Feed-Forward Network (FFN): a feed-forward network that combines a linear MLP with a lightweight, data-driven convolution, preserving spatial information without relying on fixed positional embeddings. At the end of each stage, patches are merged again via »Overlap Patch Merging« to produce the next deeper feature level. The

result is a multi-level feature hierarchy that seamlessly integrates local detail with global context (see also figure 2.1).

SegFormer's decoder follows an intentionally lightweight design, avoiding the use of complex convolutional operations. To begin, each of the four multi-scale feature maps generated by the encoder is linearly projected to ensure a consistent number of channels across all stages. These feature maps are then resized via bilinear interpolation to the same spatial resolution, typically one-fourth of the original input size.

After alignment, the features are concatenated and passed through a simple MLP, which fuses them into a unified representation. A final linear layer then produces the pixel-wise segmentation map, assigning class probabilities to each location.

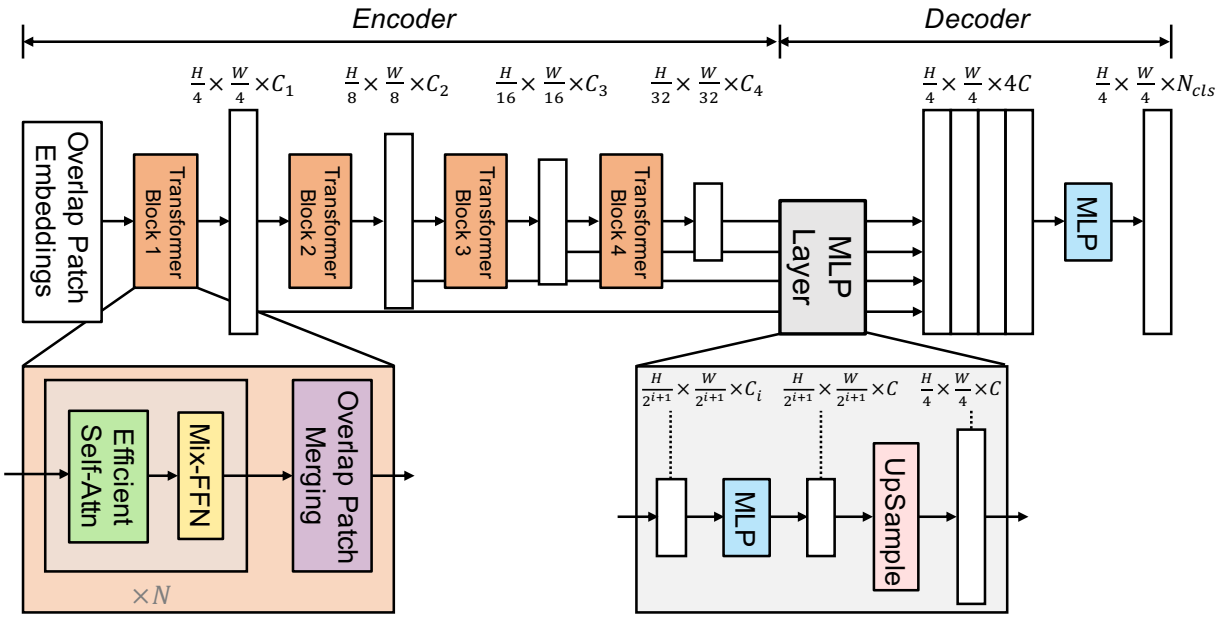

Figure 2.1: Architecture of SegFormer. Image taken from [Xie+21].

2.2.2 Adapting Vision Transformer (AViT)

AViT is a vision-transformer-based model specifically designed for skin lesion segmentation on small dermoscopic datasets. The idea behind AViT is that a large pre-trained Vision Transformer backbone already encodes the essential visual representations, e.g. spatial details, boundaries, and texture [Du+23]. Only a small subset of the model's parameters needs to be adapted to address new, specialized tasks. Instead of fine-tuning the entire ViT model for each new task, AViT keeps the backbone's weights completely frozen to preserve

the learned representations and reduce memory usage, as these parameters do not need to be duplicated across different datasets. This is achieved by inserting two lightweight modules: a prompt generator and an adapter (see figure 2.2).

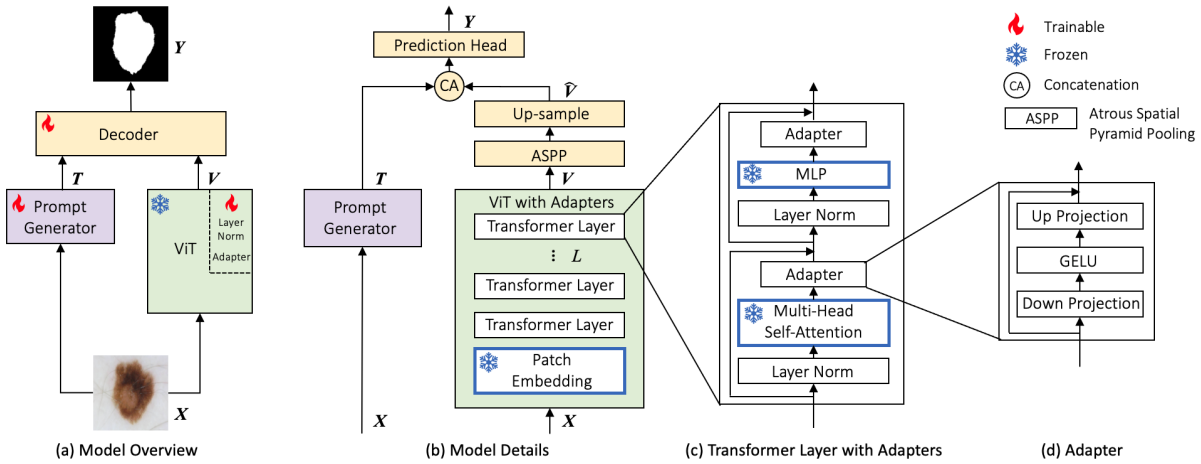

Figure 2.2: Architecture of AViT: (a) Model overview with prompt generator and a large pre-trained ViT backbone with adapters, and a compact decoder. (b to d) Model details. During training, all modules in (b,c,d) contoured with blue borders are frozen. Image and text taken from [Du+23].

The prompt generator extracts information from images by leveraging the inductive biases of CNNs, which provide the ability to capture spatial details, boundaries, and textures even with limited training data. It generates a set of prompt embeddings that consits seven convolutional layers. These prompt embeddings are prepended to the patch embeddings before being passed into the ViT's Multi-Head Self-Attention (MHSA) and remain present throughout all transformer stages. During training, only the parameters of the prompt generator and the adapter modules are updated, while the weights of the pre-trained backbone remain frozen. This design enables domain- and context-specific conditioning without increasing memory consumption or duplicating the frozen backbone for each new dataset.

Adapters are integrated into skip connections of each transformer block, following a bottleneck architecture composed of a down-projection to reduce hidden dimensionality and an up-projection to restore it. Positioned between the MHSA layers and the FFN, the adapters are the only components updated during training, while the parameterized pre-trained backbone remains frozen. This configuration reduces memory consumption by

over 60%[1] by avoiding redundancies of the backbone across datasets. Moreover, the compact parameterization of the adapters minimizes redundancy without significantly limiting the model's representational capacity, enabling efficient adaptation under constrained data conditions.

During training, AViT benefits not only from a significantly reduced number of trainable parameters, but also from a substantially lower peak Graphics Processing Unit (GPU) memory footprint. Classical MHSA scales quadratically with the input sequence length, which becomes computationally prohibitive for high-resolution volumetric inputs such as 256^3 voxels (resulting in approximately 4,096 patches at a patch size of 16^3). AViT addresses this challenge by freezing the large Vision Transformer backbone and introducing only a small number of additional prompt tokens. In practice, a patch size of 16^3 voxels reduces the sequence length to around 4,096 tokens, making the use of transformer-based models on volumetric data feasible without exceeding memory constraints[2].

2.2.3 Efficient Vision Transformer (EfficientViT)

EfficientViT was designed based on a so-called sandwich layout to achieve high inference speed [Liu+23]. The bottleneck arises from memory intensive and computationally inefficient operations such as tensor reshaping or elementwise functions in the MHSA block. Instead of the alternating arrangement of self attention and feedforward layers, this bottleneck can be reduced by adopting a sandwich layout in the EfficientViT block. In this design, a cascaded group attention block is placed between two blocks, each consisting of a token interaction layer and an FFN layer.

Acceleration in the cascaded group attention block is achieved by assigning each attention head a different split of the input. This reduces the computational load by a factor equal to the number of heads, while simultaneously increasing the diversity of the attention maps, without introducing any additional parameters.

[1] The fully fine-tuned base model with 367.2 M parameters (91.8 M×4), AViT requires only 140.2 M parameters, consisting of a single shared ViT backbone (85.8 M) and 13.6 M per dataset for adapters and the prompt generator, resulting in a memory reduction of approximately 60%.

[2] The number 4,096 results from dividing the volume size 256^3 by the patch volume 16^3 (i.e., $256^3/16^3 = 4,096$ patches). The number 4,096 is an example of a smaller volume such as 256^3 voxels: $256^3/16^3 = 16^3 = 4{,}096$. Adjust numbers based on actual use cases.

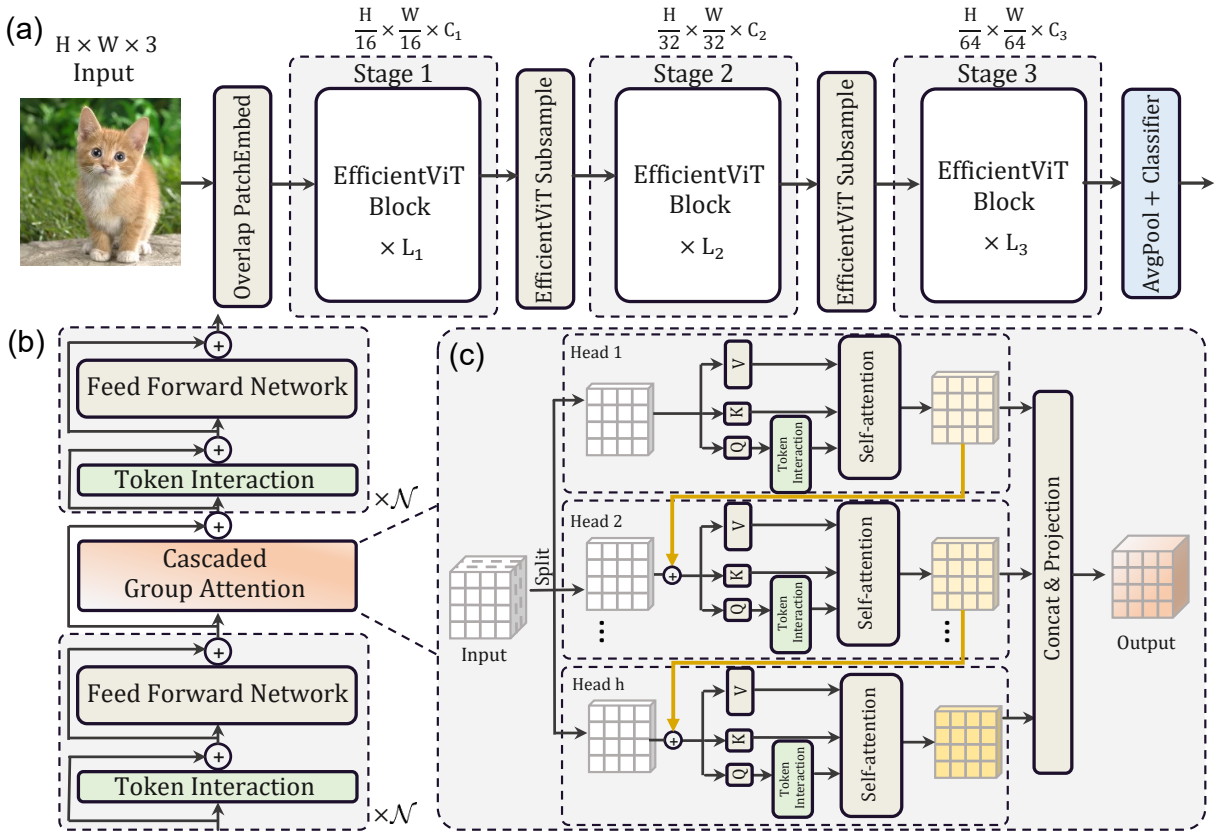

Figure 2.3: Architecture of EfficientViT: (a) Overview; (b) Sandwich Layout block; (c) Cascaded Group Attention. Image and text taken from [Liu+23].

2.2.4 Temporally Efficient Vision Transformer (TeViT)

TeViT was developed for Video Instance Segmentation (VIS), combining a hierarchical transformer backbone with multiple query-based heads for instance segmentation and temporal association (tracking) [Yan+22]. The backbone is built on a Pyramid Vision Transformer (PVT) variant that reduces the number of patch tokens to lower memory requirements. The Messenger Shift mechanism injects temporal context by shifting lightweight messenger tokens between successive frames, without any trainable parameters, thereby enabling efficient temporal fusion with minimal memory and compute overhead.

Given a sequence of video frames, Patch Merging successively generates multi-scale pyramid feature maps (with strides of 4, 8, 16, and 32), reducing spatial resolution at each stage. In parallel, messenger tokens are appended to each frame and then shifted across adjacent frames via the Messenger Shift mechanism, facilitating targeted temporal context exchange while the majority of tokens remain frame-local. On these spatiotemporal features,

the query-based head uses MHSA and »Dynamic Convolution« to produce instance masks and maintain identity consistency over time (see figure 2.4).

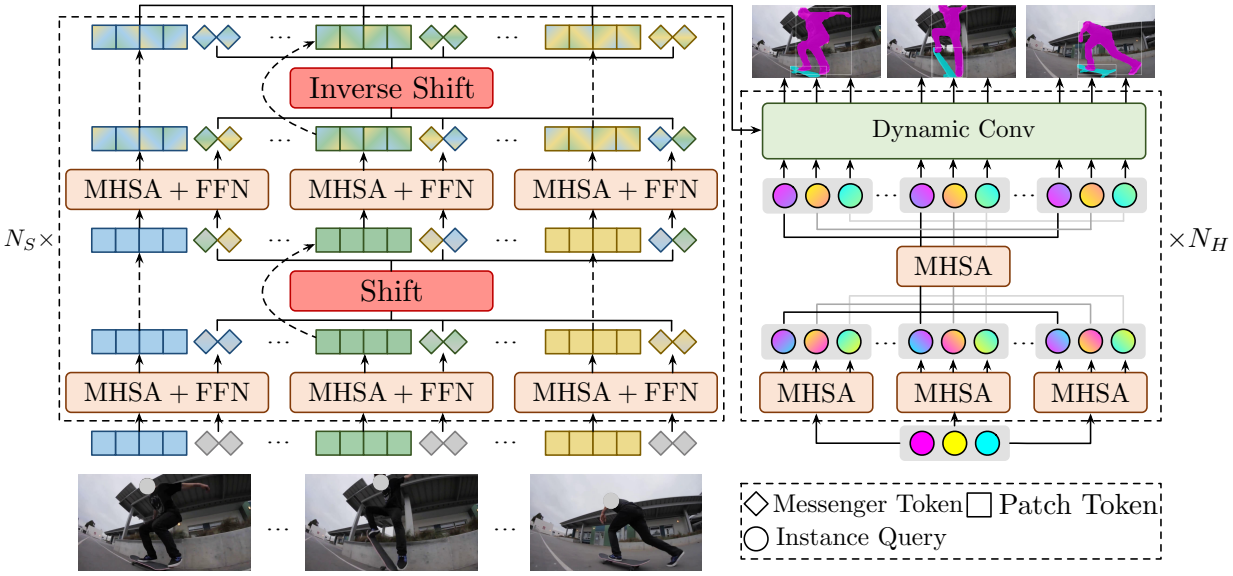

Figure 2.4: Architecture of TeViT: The overall illustration of our TeViT framework. TeViT contains a messenger shift transformer backbone and a series of spatiotemporal query-driven instance heads. The messenger shift mechanism performs efficient frame-level temporal modeling by simply shifting messenger tokens along the temporal axis. Spatiotemporal query interaction conducts two successive and parameter-shared multi-head self-attention (MHSA) with feed forward network (FFN) upon video instance queries. Image and text taken from [Yan+22].

2.2.5 Single-Head Vision-Transformer (SHViT)

Conventional Vision Transformers typically use 4×4 patch embeddings, a four-stage backbone architecture, and a multi-head attention mechanism [Liu+22; Vas+23a; Li+22b]. In contrast, SHViT was selected for its memory efficiency in segmentation tasks. This efficiency is partly achieved through the use of a larger-stride patchify stem, employing 16×16 patch embeddings without sacrificing performance. Additionally, researchers have observed that attention mechanisms in the early stages of the network lead to computational redundancies which is mitigated by SHViT by incorporating a single-head attention module [Yun+24].

The SHViT architecture is composed of a backbone and a decoder. The backbone begins with a 16×16 overlapping patch embedding block, followed by three SHViT blocks, each separated by a downsampling layer. The patch embedding is implemented using a

sequence of four 3×3 strided convolutional layers[3], each configured with a kernel size of 3×3, a stride of 2, Rectified Linear Unit (ReLU) activation, and Batch Normalization. This setup effectively reduces the spatial dimensions of the input by a factor of 16, resulting in a compact and efficient feature representation as shown in figure 2.5.

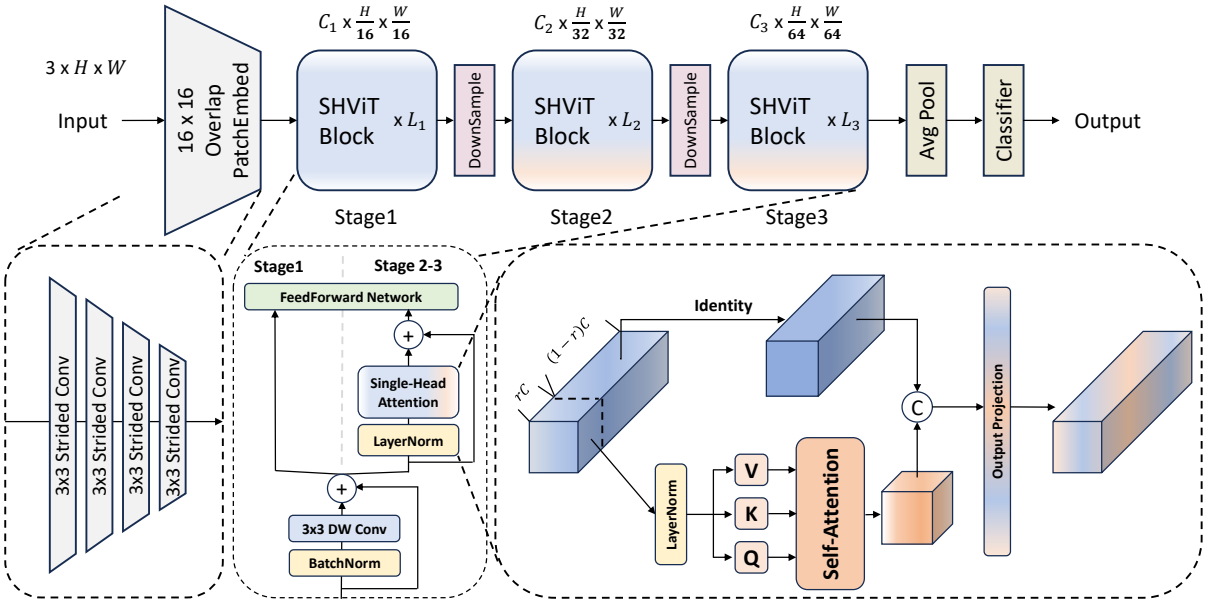

Figure 2.5: Architecture of SHViT. Image taken from [Yun+24].

To optimize computational efficiency, single-head self-attention is intentionally omitted in the early stages of the network and instead introduced in the later stages, where global context modeling is more beneficial. Consequently, the first stage of the SHViT backbone is designed without attention layers. It comprises Batch Normalization, a 3×3 depthwise convolution with stride 1 and ReLU activation, and a feedforward network composed of fully connected layers with fused Batch Normalization and ReLU activation. The spatial resolution remains unchanged throughout each individual stage, preserving feature map dimensions for consistent processing.

Given that SHViT is structured with only three stages, the single-head self-attention mechanism is employed exclusively in the second and third stages. It is placed between the depthwise convolution and the feedforward network within each stage. This design enables

[3]It was observed that using a 16×16 patch embedding block led to an excessive reduction in spatial resolution. Therefore, in this thesis a 4×4 block (2 convolutional layers instead of 4 convolutional layers) was used instead to preserve more spatial detail. However, the number of layers is parameterizable.

efficient capture of long-range dependencies and enhances the network's ability to model global context in deeper layers, where such information becomes more relevant.

The output of the backbone is then forwarded to a lightweight decoder, which consists of an average pooling layer followed by a classifier. This final component generates the model's predictions, thereby completing the SHViT pipeline as illustrated in figure 2.5. It is important to note that the final classification layer of the SHViT model is omitted, as only the outputs from the three backbone stages are required and subsequently passed to the MLP layer of the SegFormer decoder [Yun+24] (see also section 2.2.1).

The version of SHViT used in this thesis is based on the code from the publicly available GitHub repository[4], with minor modifications detailed in chapter B, section B.1.7. It should be noted that the »Block« shown in figure 2.5 differs from the »Block« defined in the code. In the code, the »Block« also includes the downsampling operation, whereas in the diagram, these components are depicted separately.

2.2.6 Choosing the right ViT for 3D volumetric data

In the previous sections various ViTs such as TeViT [Yan+22], EfficientViT [Liu+23] AViT [Du+23], and SegFormer [Xie+21], and SHViT [Yun+24] are described. As noted at the beginning, choosing the right ViT is critical given the enormous computational and memory demands of processing 3D data. An equally important criterion is the simplicity and adaptability of the architecture. Unlike 2D images, volumetric inputs generate a token number that cubically increases with each spatial dimension, rendering standard multi-head self-attention practically useless at comparable resolutions.

AViT, for example, freezes an pre-trained ViT backbone and uses lightweight prompt embeddings and adapters to adapt to new domains. Nevertheless, AViT still relies on self-attention across thousands of tokens, resulting in large-scale GPU memory requirements even when the volume is scaled down.

EfficientViT employs cascaded group attention, which helps reduce memory access overhead to some extent and saves computation overhead, but still uses attention mechanism in each stages that leads to computational redundancies that is solved by SHViT applying a subset of feature channels (partial ratio). Consequently, in practice, batch sizes must be drastically reduced to satisfy hardware constraints, which in turn destabilizes the normalization layers and reduces training throughput.

[4] https://github.com/ysj9909/SHViT/blob/main/model/shvit.py

TeViT is not suitable for 3D voxel data, as the efficiency advantages are limited exclusively to two spatial dimensions. The PVT backbone reduces the number of tokens by patch merging only along the height and width, leaving the depth dimension completely untouched. As a result, the cubic growth of the token length (depth × height × width) remains across all MHSA layers, which leads to high memory consumption even after downsampling.

SegFormer utilizes four Transformer blocks in its encoder and employs a multi-layer perceptron (MLP) layer in the decoder to fuse the outputs from each encoder stage and produce the final prediction. A major drawback of this design is the large number of trainable parameters, which can reach up to approximately 120 million, significantly increasing the model's computational and memory demands.

Unlike other Vision Transformer models, SHViT does not employ Multi-Head Self-Attention. Instead, it relies on a Single-Head Self-Attention (SHSA) mechanism, which reduces memory consumption and number of trainable parameters compared to multi-head architectures. By replacing multiple parallel attention heads with a single head, SHViT not only lowers computational overhead but also limits memory access. In particular, the partial-channel approach illustrated in figure 6 of the original paper [Yun+24] directs only a subset of feature channels through the SHSA modules, while preceding convolutional layers extract local detail. This design achieves »the best speed-accuracy trade-off«[Yun+24] by unifying the complementary strengths of convolution and attention in a single token mixer. At the same time, memory-bound operations such as tensor reshaping and layer normalization are confined to a limited number of channels. As shown in figure 7 of the SHViT-paper [Yun+24], this allows the SHSA module to uses GPU and CPU compute resources far more efficiently than conventional multi-head mechanisms. By reducing memory-intensive steps, such as reshaping and normalization, the peak memory footprint drops substantially, enabling larger batch sizes and higher image resolutions in volumetric 3D scenarios while preserving training stability.

2.2.7 Architecture of 3D SHViT-SegFormer

The decoder portion of SHViT is designed for instance or object detection rather than for semantic segmentation, that's why a different head is required. For this purpose, the head from the SegFormer model [Xie+21; Per+24] was adopted. Like most vision transformers, SegFormer also consists of an encoder and a decoder (see figure 2.1). The SegFormer model with SHViT as its backbone represents a straightforward combination of

both individual models. However, since SHViT consists of only three stages, only three feature maps are forwarded to the MLP decoder. The number of convolutional layers N in the Overlap Patch Embedding block is a configurable parameter and can range from 1 to 4. As will be demonstrated in chapter 4, setting `N=2` yields the best trade-off between F1/Dice-performance and GPU memory usage.

Figure 2.6 shows the points at which the SHVIT outputs is extracted and transferred to the MLP block of the SegFormer decoder.

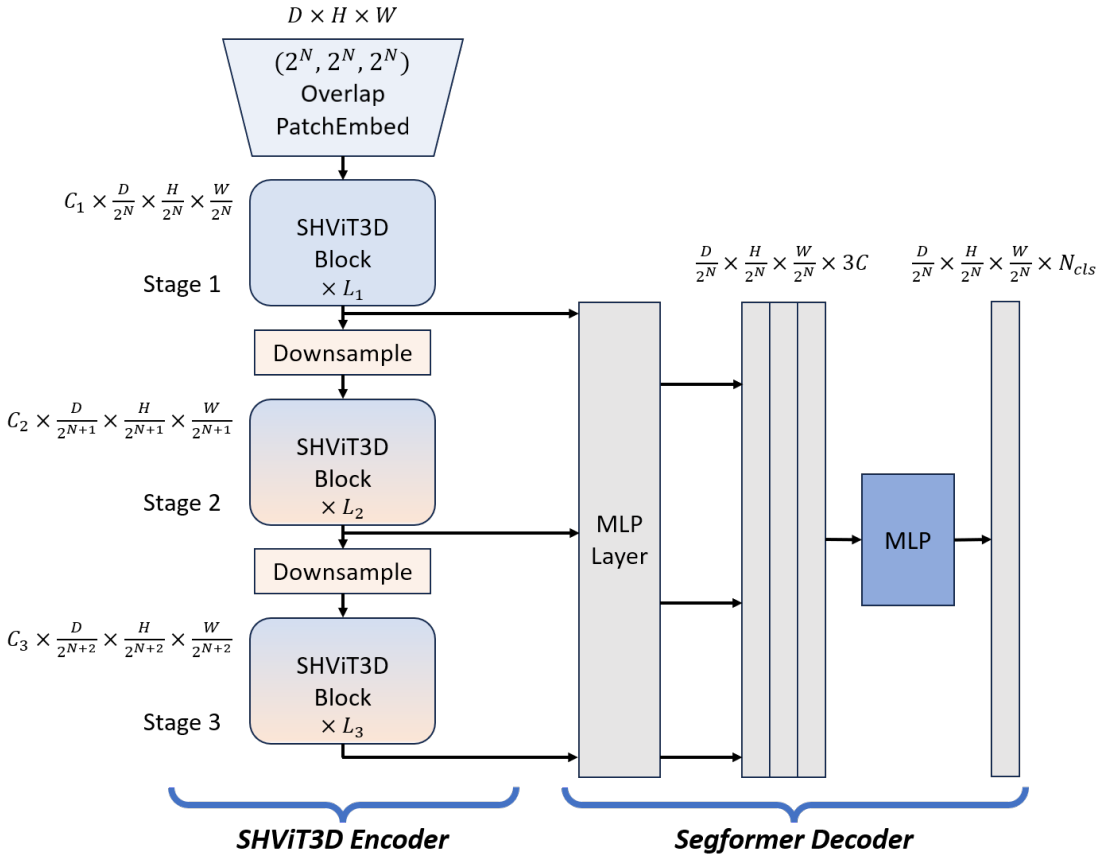

Figure 2.6: Architecture of SegFormer with 3D SHViT-backbone: N is the number of convolutional layers in PatchEmbedding-Block (SHViT3D-head). Because of `stride=2` the size after each downsampling block is decreased by a factor of 2.

According to [Xie+21], Section 4.2 (Ablation Studies: Influence of `C`, the MLP decoder channel dimension), the SegFormer decoder uses an `embed_dim` of `256` for variants B0 and B1, and `768` for B2 through B5. The parameter settings for the 3D SHViT component of the model are presented in table 2.1, which is adapted from [Yun+24; IMv24].

Model variants	Depth	Emb. dim.	Partial ratio	Exp. ratio	B0/B1	B2-B5
SHViT-S1	[2, 4, 5]	[128, 224, 320]	1 / 4.67	2	256	768
SHViT-S2	[2, 4, 5]	[128, 308, 448]				
SHViT-S3	[3, 5, 5]	[192, 352, 448]				
SHViT-S4	[4, 7, 6]	[224, 336, 448]				

Table 2.1: Architecture details of 3D SHViT variants: table taken from [Yun+24]. Parameters B0 to B5 are used in `embed_dim` of SegFormer decoder, values are taken from [IMv24].

2.3 Reducing Memory in Transformer Attention Mechanisms

The self-attention mechanism[5], particularly the computation of

$$\text{Attention}(q, k, v) = \text{softmax}\left(\frac{q^T k}{\sqrt{d_k}}\right) v \tag{2.1}$$

can be highly memory-intensive, especially when processing long sequences or high-dimensional embeddings [Vas+23b]. The high memory consumption comes from the matrix multiplication between the transposed query tensor (`q`) and the key tensor (`k`), which is of order $O(n^2)$. The term d_k, representing the dimensionality of the queries and keys, will later be denoted as `qk_dim`. To understand the impact, it is important to consider the shapes and scaling behavior of these tensors. The following explanation, based on the SHViT architecture with the S4 configuration, illustrates the reasons behind the large sizes of the query, key, and value matrices `q`, `k`, and `v`. In the following, the operations that take place within the SHSA block, are described. The corresponding code implementation can be found in section B.1.7 in the `SHSA3D` class.

2.3.1 Memory Bottleneck in 3D Attention Block

Starting with an input volume of shape `(1,224,224,224)`, where the first dimension indicates the batch size, and using for the calculation two `Conv3D` layers in the `PatchEmbed`

[5] In the original Transformer paper by Vaswani et al. [Vas+23b], the attention score is written as qk^T, since the query and key matrices are typically shaped (n, d_k), placing the sequence length n first. However, in the codebase for SHViT, the tensor shape is just opposite: (d_k, n). This reverses the usual convention, and to be consistent with the code $q^T k$ is used in this thesis.

head (see figure 2.6), the spatial dimensions are downsampled by a factor of 2 at each layer, resulting in an intermediate feature map of shape `(1,56,56,56)`. This tensor is then passed into the first `SHViT3D` block, where another spatial downsampling by a factor of 2 is performed, reducing the shape to `(28,28,28)`. At this point, the channel dimension is expanded to `336`, yielding a tensor of shape `(1,336,28,28,28)` as the input to the forward function of the `SHSA3D` module.

Within the forward function, the input tensor `x` has the shape `(1,336,28,28,28)`. The following configuration parameters are used:

- `dim = 336` (input channels),
- `qk_dim = 16` (dimension of queries and keys),
- `pdim = 72`, which corresponds to a fraction of approximately 1/4.7 of the total channels of 336.

The tensor `x` is then split along the channel dimension into two parts:

- `x1` with shape `(1,72,28,28,28)` (used for computing attention),
- `x2` with shape `(1,264,28,28,28)` (bypasses attention and is later concatenated back).

Next, the tensor `x1` is passed through the `qkv` layer, which is essentially a `Conv3D` with input channels of `72` and output channels of `2*qk_dim + pdim = 2*16 + 72 = 104`. This produces a tensor of shape `(1,104,28,28,28)`, which is then split into:

- query tensor `q` with shape `(1,16,28,28,28)`,
- key tensor `k` with shape `(1,16,28,28,28)`,
- value tensor `v` with shape `(1,72,28,28,28)`.

After flattening the spatial dimensions, the shapes become:

- `q` with shape `(1,16,21952)`,
- `k` with shape `(1,16,21952)`,
- `v` with shape `(1,72,21952)`.

where `21952` is simply 28^3, the total number of voxels of this stage.

The critical and memory-intensive operation which follows now, performs matrix multiplication between tensors $\mathtt{q}^\mathtt{T}$ and `k` of shapes `(1,21952,16)` and `(1,16,21952)`, respectively, producing an attention matrix of shape `(1,21952,21952)`, containing over 480 million elements. This quadratic growth in memory usage with respect to spatial size is the primary source of computational bottleneck in the model.

Subsequently, the final attention-weighted values are computed by multiplying the value tensor `v` to the matrix which reduces the tensor back to shape `(1,72,21952)`, and after reshaping, results in its final size of `(1,72,28,28,28)`. Finally, this attention-modified output is concatenated along the channel dimension with the unmodified `x2`, reconstructing the full tensor with shape `(1,336,28,28,28)` for further processing (see also figure 2.5). This detailed tracing highlights not only the shape transformations but also the source of the heavy memory usage, the large attention matrix, which becomes especially problematic for high-resolution 3D volumes.

However, if only one single `Conv3D` layer is used in the head of the `SHViT3D` model, resulting in a reduced downsampling, or if the input volume is large, then the input to the `SHSA3D` block increases in spatial resolution. In this case, the input shape becomes `(1,336,56,56,56)`, and the resulting attention matrix grows to a size of $(56^3, 56^3) = (175616, 175616)$, which contains over 30 billion elements. This represents a 64 increase in memory consumption compared to the previous case, highlighting the exponential scaling behavior of the attention mechanism in 3D space.

This observation clearly illustrates why the `SHViT3D` model, particularly the `SHSA3D` block, is highly memory-intensive. Careful architectural decisions are necessary to balance segmentation quality and memory feasibility. In practice, using two `Conv3D` stages in the head provides a good compromise, sufficient downsampling to reduce memory usage, while still preserving enough spatial detail for accurate segmentation.

Several strategies have been proposed in the literature to reduce memory consumption while preserving model performance. The following techniques are commonly used:

- Precision Reduction: Use lower precision data types, like `float8` or `float16` instead of `float32`, to reduce the memory consumption with minimal accuracy loss [Bra+25]. A further reduction in quantization down to 4- or 2-bit was also studied in [She+20]. The implementation of mixed-precision computation has proven to be an effective

strategy for reducing memory consumption during model training and inference. In this thesis, mixed-precision training was adapted to optimize the memory efficiency of the model within the available hardware constraints. The code uses the `Lightning` library from `Fabric` to facilitate and manage mixed-precision training efficiently[6].

- Perform Attention on Smaller Blocks: Divide the input sequences into smaller blocks and perform attention on these blocks separately. This approach, known as »block sparse attention«, »local attention«, or »sliding-window«, reduces matrix size. For example, Longformer replaces full attention with a sliding-window local attention plus a few global tokens. This makes memory scale linearly with sequence length [Bel+20]. This idea was implemented using the code snippet from SimViT [Li+22a] and described further below.

- Use of Sparse Matrices: If the matrices contain many zeros or negligible values, using sparse matrix representations could save memory. Literature research shows that Child et al. [Chi+19] introduced Sparse Transformers that attend with fixed strided and block-sparse patterns. Sparse matrices typically maintain accuracy comparable to dense attention and in some cases even improve training speed, at the cost of introducing heuristic patterns or additional overhead from sorting or clustering [Dee20]. Due to its high complexity it was not implemented in the code.

2.3.2 Sliding-Windows Mechnism

Due to the fact that spatially distant regions in an image generally do not exhibit strong relationships, it is reasonable to process images using a sliding window approach with overlapping regions. This method enables the efficient capture of local spatial structures by focusing only on a patch and its immediate neighbors, while ignoring distant points (or tokens) [Fu+25]. Within each window, as shown in figure 2.7, a Single-Head Central Self-Attention (SHCSA) mechanism is applied, where only the central patch serves as the query, and the surrounding patches are used as keys and values. While the original method is based on Multi-Head Central Self-Attention (MHSA) as proposed by Li et al. in the SimViT paper [Li+22a], this thesis adapts the concept by integrating only the attention mechanism from MHSA into SHSA from SHViT. This results in an enhanced version, referred to as SHCSA, which significantly reduces computational cost while maintaining a high level of performance and accuracy.

[6] https://lightning.ai/docs/fabric/stable/fundamentals/precision.html

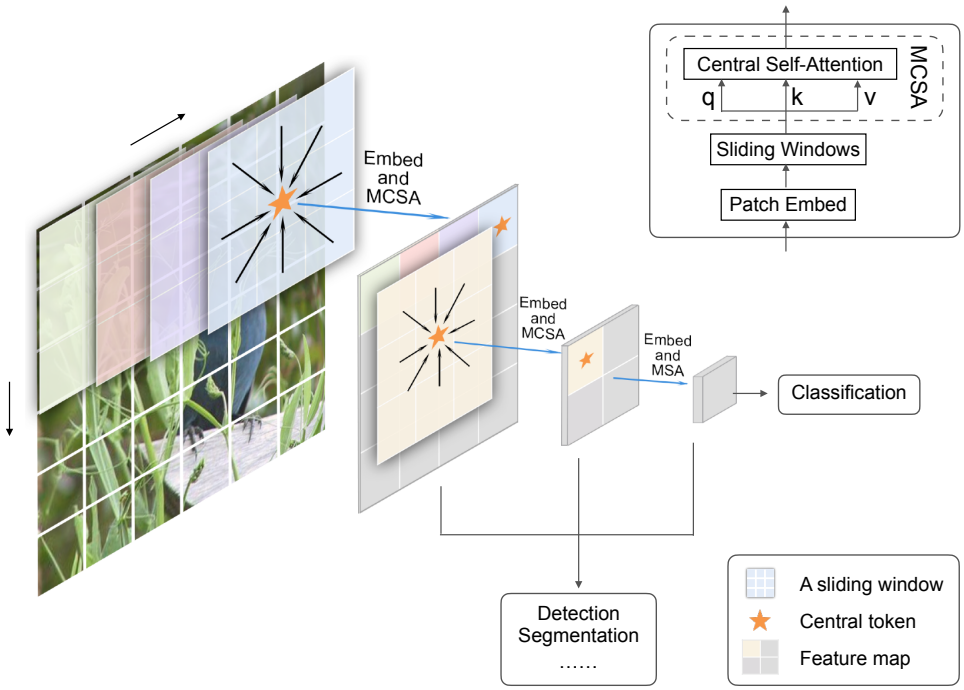

Figure 2.7: Visualisation of the sliding window mechanism with overlapping regions applied to the feature map of the image. Within each window, only the central patch (highlighted) acts as the query, while the neighboring patches serve as keys and values. Image taken from [Fu+25].

2.4 Metrics Used for Model Evaluation

In the context of neural network training, metrics and loss functions play an important roles in guiding the optimization process and evaluating model performance.

Evaluating the performance of segmentation models involves several metrics, including [Aza+23]:

Intersection of Union (IoU) or **Jaccard index**, which is the ratio of the intersection area to the union area of the predicted (P) and ground truth segments (G):

$$\text{IoU} = \frac{|P \cap G|}{|P \cup G|} = \frac{|P \cap G|}{|P| + |G| - |P \cap G|} \tag{2.2}$$

The Jaccard Distance Loss is the defined as:

$$\text{Jaccard-Distance-Loss} = 1 - \text{IoU} \tag{2.3}$$

If there is no intersection between the two classes P and G, then $|P \cap G| = 0$ and IoU $= 0$ and the Jaccard-Distance-Loss $= 1$. In an experiment with multiple segmentations IoU is averaged over all single IoUs:

$$\text{mIoU} = \frac{1}{N} \sum_{i=1}^{N} \text{IoU}_i \tag{2.4}$$

An often used coefficient is the **Dice Coefficient** (or **F1-score**). It is defined by the following formula (where P are the predicted and G are the ground truth segments):

$$\text{Dice} = \frac{2\,|P \cap G|}{|P| + |G|} \tag{2.5}$$

$$= \frac{2 \cdot \text{IoU}}{1 + \text{IoU}} \tag{2.6}$$

and the Dice-Loss is then defined as

$$\text{Dice-Loss} = 1 - \text{Dice} \tag{2.7}$$

Dice/F1 and IoU are monotonically related, so a better overlap raises both scores. However, Dice/F1 tends to be higher for the same overlap (e.g. IoU $= 0.5 \rightarrow$ Dice/F1 $\approx$ 0.67). This means that Dice/F1 places greater emphasis on the intersection, whereas IoU penalizes false positives and false negatives more heavily, similar to how L2 loss emphasizes large deviations compared to L1. In non-medical computer vision, IoU or mIoU is predominantly used as the evaluation metric, particularly in object detection, while Dice-based loss functions are primarily employed during training for segmentation tasks, especially when class imbalance or small object sizes are present [Aza+23].

Additionally, both metrics can be misleading when averaged across many samples, especially when ground truth regions are very small. In such cases, even minor errors can disproportionately reduce the overall score, giving excessive weight to trivial misclassifications.

Another loss function, which can be used for image segmentation, is the **cross-entropy** loss. It also measures the difference between predictions and ground truth but treats each pixel independently, regardless of the class it belongs to. Therefore, it is very sensitive to class imbalance unless it is weighted. For example, in imbalanced datasets with a lots of background in the image/volume, the cross-entropy loss is very much influenced by the majority class. As a result, the model may learn to prioritize accuracy on the frequent

classes while ignoring others. In contrast, Dice/F1 and IoU losses measure the overlap between predicted and true regions, and therefore offering better performance on imbalanced datasets [Aza+23].

For the training the segmentation model of 3D scans in this thesis, the Dice/F1-score is used as the evaluation metric due to its widespread adoption and consistent definition in the existing literature. This metric allows comparison with previous research and ensures that the performance assessment aligns with established standards in the field. From this point onward, only the term »F1-score« will be used, as it is equivalent to the Dice-coefficient.

Chapter 3

Data Preparation

Before starting to develop and train a machine learning model, it is important to examine the dataset at hand. In this first step, the dataset must be checked in detail to learn how it is structured, what types of data it contains and whether the data is consistent and reliable. This in-depth exploration is essential, as the quality and structure of the data have a direct impact on the performance of the model and the validity of its predictions.

3.1 Detailed Analysis of the Dataset

By analyzing the dataset, inconsistencies, missing values or anomalies that could potentially distort the results can be identified. In addition, this process helps to understand the relationships between different variables, ensure that the data is representative of the problem area, and can provide accurate and meaningful insights.

3.1.1 Shoe Dataset Acquisition Using Computed Tomography

The shoe dataset used in this thesis is the same as used in a previous study, the results of which were presented in a conference paper [Lei23]. 40 high-resolution 3D scans of shoes in boxes are created using a Werth Tomoscope operating at 200 kV. The scans produced high-resolution volumetric data with a voxel size of 340 µm and volumes approximately 1000^3 voxels in size. Each scan captured not only the shoe structure but also the surrounding packaging materials (e.g., cardboard boxes and stuffing paper).

To generate training data for the segmentation, the CT volumes were manually labeled using 3D Slicer [Fed+12]. The process involved annotating each boxed shoe pair in three

stages, corresponding to the segmentation steps. Seed points are placed in regions representing the shoe, packaging material, and background. Theses seed points are gradually expanded, followed by manual corrections near boundaries where shoe and packaging overlapped. The shoe was further segmented into outsole, insole, and upper material based on differences in CT density and structure. The outsole, being dense and thick, was easily distinguishable, and the insole appeared as a softer layer above it, while the the upper was thinner and more variable in material. The labeling process of each shoe pair took up to three hours, depending on their complexity. The authors note that the labeled dataset contains some minor misclassification errors, mostly due to boundary ambiguities and labeling complexity. These are assumed to be random and are addressed during training [Lei23].

Each 3D shoe scan is saved in `.rek`-format[1], with file sizes ranging from 500 MB to almost 5 GB, and is intended to represent the input data for the segmentation model to be created. The corresponding ground-truth dataset with annotated segmentation labels are compressed and saved in a proprietary file format `.seg.nrrd`[2] designed for efficient storage and retrieval and the file sizes ranges from 13 MB to nearly 160 MB.

3.1.2 Visualisation of 3D-Dataset

The first step in analyzing the dataset involves visualizing the 3D dataset to gain a clearer understanding of its structure and contents. Hence, a Python program was written to process and visualize the data.

The libraries `rek2py` and `slicerio`[3] are used to extract the data of the scan dataset with `.rek` file format and the annotated segmentation data of the ground-truth `.seg.nrrd` files. All 3D-scan files in the folder were read in individually with the script, their data extracted into header and volume with the `rek2py` function, the volume data then was split into slices, and saved. The sizes of the volume data ranges from `(958, 410, 685)` up to `(1056, 1071, 1068)`, which also correlates quite well with the file sizes.

During analysis of the individual images, it became evident that image quality varied significantly. This inconsistency was mainly caused by brightness scaling across the entire

[1] Before the data can be extracted from the file, the Python package `PythonTools-3.7.0-py2.py3-none-any.whl` must be installed manually.

[2] NRRD stands for »Nearly Raw Raster Data« https://pynrrd.readthedocs.io/en/stable/background/about.html

[3] https://pypi.org/project/slicerio/

volume, propably due to different CT scan settings. Such variability posed challenges for consistent preprocessing and model input normalization. To address this, the concept of histogram stretching was applied: the minimum and maximum percentiles were set to 1% and 99%, respectively, based on trial and error, in order to enhance the contrast and brightness of the image. This approach helped reduce the influence of outliers and improve visual consistency across the dataset. Based on these percentile thresholds, the corresponding minimum and maximum intensity values for each shoe were calculated using the NumPy function `np.percentile()`. These values were then needed to scale and clip the entire volume to the range `[0,1]`. An exception was made for the shoe `PetrolioSch-40-float_down2_2_2`, where the lower percentile threshold had to be set to 4%.

The example plot of two views of a 3D shoe scan (figure 3.1) shows the full volume on the left, while the image on the right displays a cross-sectional cut through the scan.

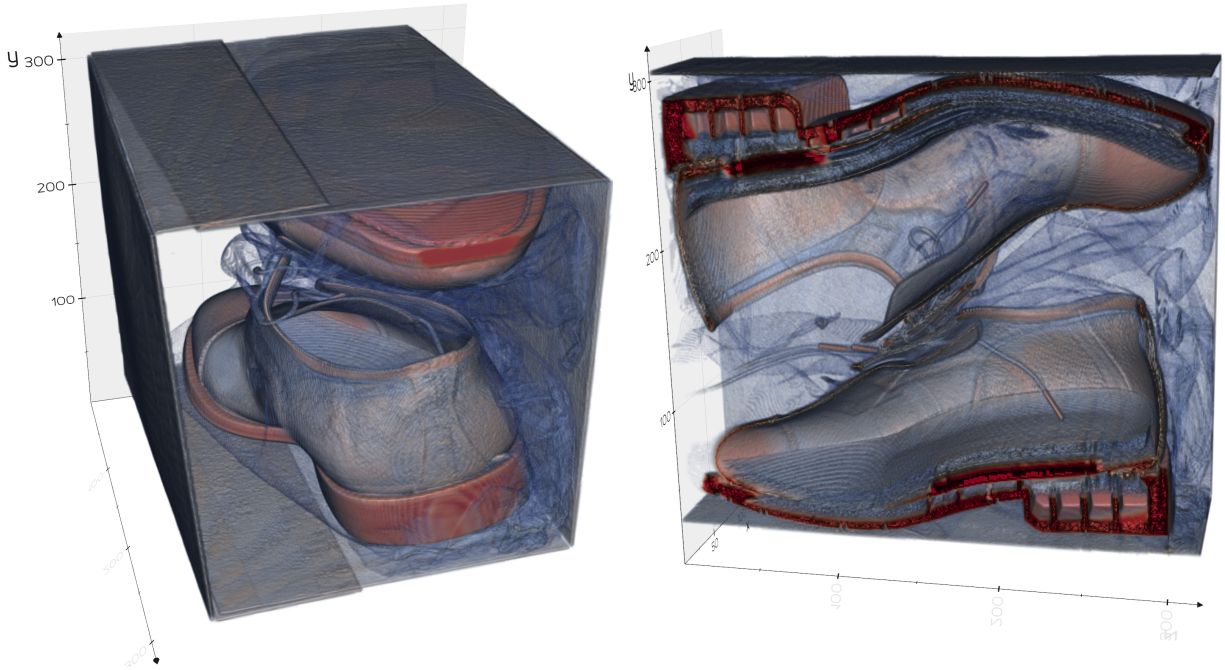

Figure 3.1: 3D-scan of `Bruschi_down2_2_2.rek` with the original resolution of `(876,650,475)` downsampled to `(320,320,320)`.

Another display variant is a series of slices through the scan. Figure 3.2 shows 24 cross-sections of the same shoe. In these images, the pair of shoes inside the box is visible and the soles, upper materials, box, and packing materials are clearly distinguishable.

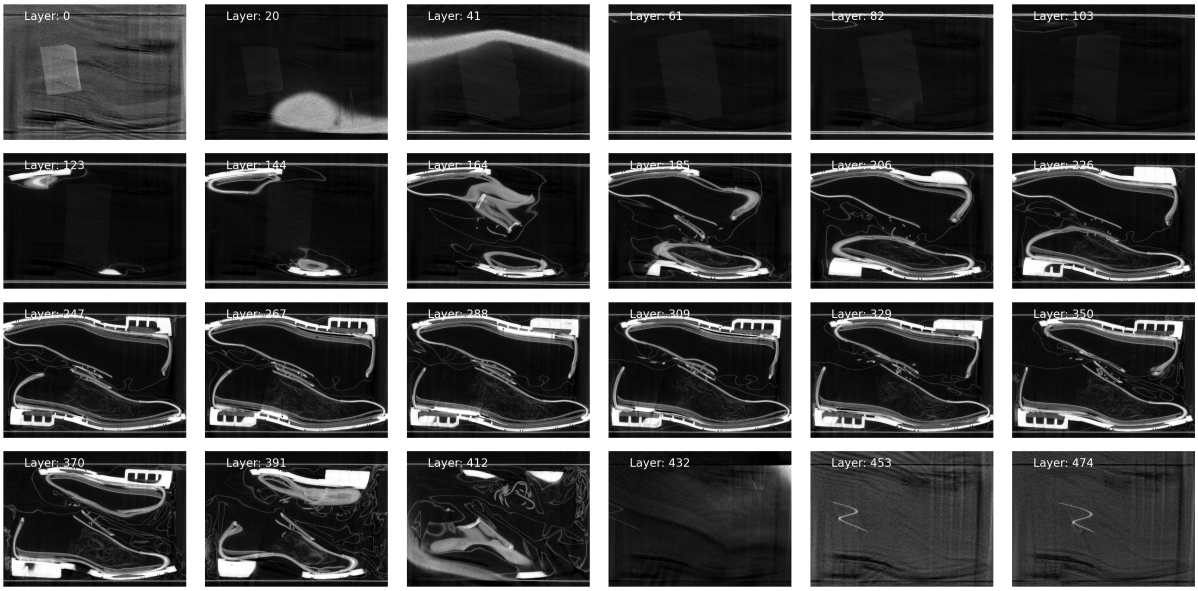

Figure 3.2: File `Bruschi_down2_2_2.rek` with the resolution `(876, 650, 475)`. Shown are 24 different layers representing the 3rd dimension.

When retrieving the associated segmentation plots for the file `Bruschi_down2_2_2`, the 12 segment names are `Background`, `Schuh_1`, `Schuh_2`, `Karton`, `Innenvolumen_1`, `Innenvolumen_2`, `Obermaterial`, `Innensohle`, `Au(ß)ensohle`[4], `F(ü)llmaterial`, `Zunge`, `Dummyschuh`. However, for some shoes, only 11 labels may be present. In such cases, the label `Dummyschuh` is typically missing but can be reconstructed by merging the labels of `Schuh_1` and `Schuh_2`. Alternatively, some files use the label `Schuh` or `Segment_1` instead of `Dummyschuh`. These discrepancies likely result from differences in annotation practices during the labeling process.

Due to inconsistencies in the ordering of labels across annotation files, labels were not assigned based on index positions. Instead, each segment was identified and assigned according to its label name to ensure correctness regardless of order. This approach ensured that label mappings remained robust even when annotation files differed in structure or completeness.

Figure 3.3 shows the 12 different segmentations for layer 335 of shoe `Bruschi_down2_2_2` individually. A closer look at the images reveals that some segmentations occur several times. For example, `Schuh_1` and `Schuh_2` or `Obermaterial`, `Innensohle`, `Außensohle` and `Zunge` are included in `Dummyschuh`.

[4]When extracting the labels, the German umlauts and the 'ß' character are not displayed.

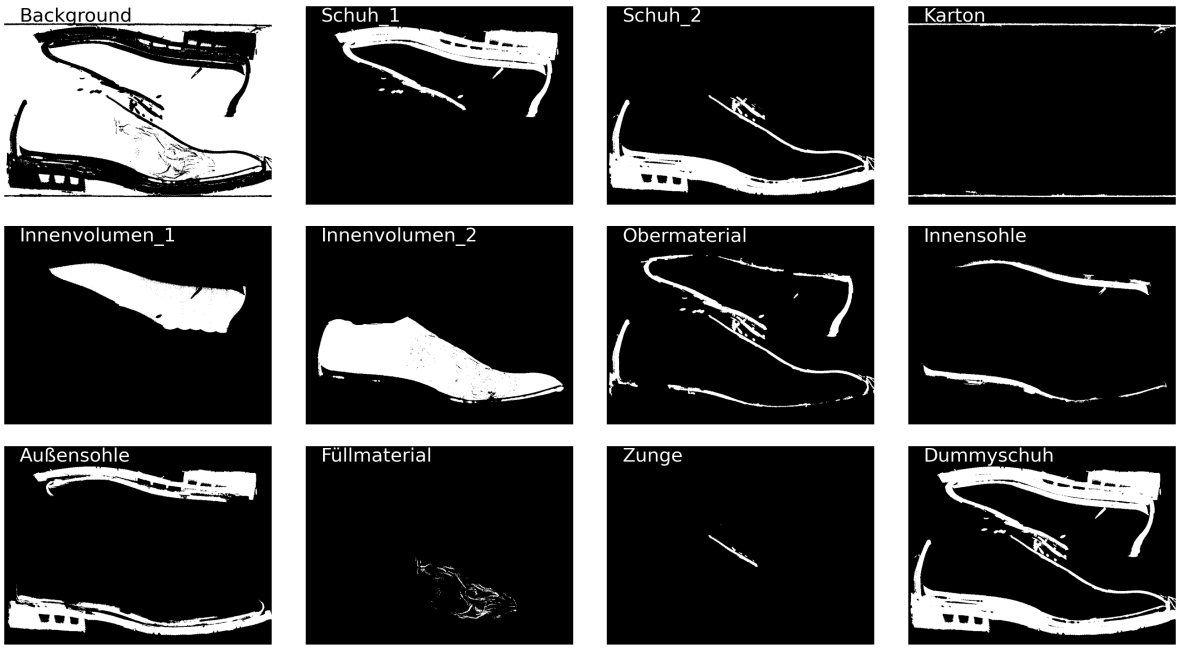

Figure 3.3: Visualisation of all 12 segments for annotation file `Bruschi_down2_2_2.seg.nrrd`, Layer: 335

Only the six segments `Karton`, `Außensohle`, `Innensohle`, `Obermaterial`, `Zunge`, and `Füllmaterial` are used as ground-truth labels for the segmentation for the training of the model. The original `Background` image provided in the dataset could not be used because it was incorrect. It should highlight only regions that do not belong to any of the shoe or packaging components. However, the original version contained inconsistencies, for example, some parts of the shoes were marked as background. Therefore, a new background mask was created by inverting the sum of the six segments mentioned above. The background, as shown in figure 3.3, is now correct.

For the segmentation of the 2D images, a representative layer was selected by hand for each shoe somewhere in the middle of the 3D data set. The orientation for seven pair of shoe scans was also adjusted so that they can be seen in the sectional plane from the side. The shoe images were saved under the file name of the `rek` file and the seven label or segment images under the name of the annotation file. The image/segment files are assigned via key-value pairs in a dictionary.

The volumes were scaled to the dimensions depth (D) × height (H) × width (W), whereby depth, height, and width were set to be equal. The segment file was then created

by reducing the volume by a factor of 2^N, where N is the number of `Conv2d` or `Conv3d` layers in the head, which can take values between 1 and 4. The depth, height, and width were also identical.

3.2 Data Augmentation used for Training

Data augmentation is essential when training deep learning models on small datasets. Given the limited availability of only 40 volumetric shoe scans, the network faces a significant risk of overfitting and reduced generalization [Hao+21]. Augmention of the training set with transformed copies of each volume artificially expands the dataset and better approximates the true data distribution. Segmentation tasks in particular usually rely heavily on both the quantity and the quality of ground truth data, and annotation of 3D volumes is very costly [Sut+24]. To avoid this, augmentation techniques that apply geometric transforms to the 3D volumes are commonly deployed. More generally, surveys of segmentation on small datasets emphasize that »data augmentation techniques are of great importance« because collecting sufficient volumetric annotations is »notoriously difficult« [Alo+23].

Data augmentation should be matched to the nature of the dataset, ideally reflecting its intrinsic variations. In the case of the shoe dataset, only horizontal flipping and 90° rotations were applied, as these transformations align with the realistic orientations found in the scanned volumes. Shoes are generally placed in a consistent and orderly manner within the boxes, rather than in random or chaotic positions. Additionally, variations in brightness were observed, caused by fluctuations in exposure time that differ between individual CT scans.

To increase diversity of the shoe-volume dataset, several standard volumetric augmentation methods are applied. In 3D space, simple affine transformations can generate new views of each shoe without altering its shape. In addition to geometric transformations, adjustments to brightness and contrast are also employed to simulate variations in scan conditions and improve model robustness. Commonly used operations include [Alo+23; Hao+21; Sut+24]:

- Axis-aligned flips (mirroring): Reflecting the volume across one of its principal planes (e.g. sagittal, coronal or axial) produces a mirror image of the shoe. This is equivalent to swapping one coordinate axis (for example flipping left-right across the sagittal plane). Flips preserve all voxel neighborhoods and create a valid alternative orientation.

In practice, mirroring along each axis can roughly double or triple the dataset size (depending on symmetry).

- 90° Orthogonal rotations: Rotating the entire 3D volume by 90°, 180°, or 270° about any of the three coordinate axes creates new views. For instance, turning the shoe scan by 90° around the vertical (z) axis yields a different front-back orientation, while a 90° rotation about the left-right (x) axis tilts the shoe forward. By applying rotations in 90° increments it is ensured that voxel values remain aligned on the grid (no interpolation artifacts) and the shoe's shape is preserved.

- Combined transforms: These basic flips and rotations can be composed to create additional variations. For example, a shoe volume might first be flipped left–right and then rotated 180° about the vertical axis, resulting in a novel viewpoint. Each combination still preserves the shoe geometry while presenting it differently.

- Brightness and contrast adjustment: To enhance robustness against variations in CT imaging conditions, voxel intensities are augmented through global brightness and contrast adjustments. Brightness augmentation involves adding a small offset to all voxel values, simulating differences in exposure levels. In contrast, contrast augmentation rescales voxel intensities relative to the mean, thereby enhancing or diminishing the distinction between dense and less dense regions. These transformations preserve the structural integrity and spatial relationships within the volume, while enabling the network to generalize more effectively to scans acquired under different imaging settings or with varying material compositions. For the augmentation, brightness values are randomly selected from the set `[-0.2, -0.1, 0.0, 0.1, 0.2]`, and contrast factors from `[0.8, 0.9, 1.0, 1.1, 1.2, 1.3]`. Given the 3D volume »vol«, the following formulas are applied:

$$\text{vol} = (\text{vol} - 0.5) \cdot \text{contrast} + 0.5 \tag{3.1}$$

$$\text{vol} = \text{vol} + \text{brightness} \tag{3.2}$$

After applying these transformations, voxel intensities are clipped to the range `[0,1]` to maintain valid intensity values.

Each of these augmentations produces a new training example. When applied to all 40 original scans, the effective training set grows by a factor equal to the number of distinct transforms used. Importantly, because these are rigid-body transformations, the

ground-truth segmentation masks can be identically transformed (flipped or rotated) to match the augmented volumes. This guarantees label consistency because the shoe and its segmentations rotate together.

This practice of expanding volumetric datasets is well supported in the literature and has been shown to improve segmentation accuracy in analogous medical-imaging problems [Alo+23; Hao+21; Sut+24]. By applying these augmentations, the model can better generalize from only 40 original scans to the full variability of real-world shoe orientations and CT imaging conditions, such as differences in brightness and contrast.

3.3 Implementation Details

Before training, all input data undergo a structured preprocessing pipeline to ensure consistency and compatibility with the segmentation model. The preprocessing procedure consists of two main stages: processing the volumetric image data and preparing the corresponding annotation files.

In the first stage, each shoe scan, which is stored as a 3D volume, is loaded from disk. To establish a uniform viewpoint across the dataset, some of the volumes are rotated such that all shoes are viewed consistently from the side. Following this orientation correction, all volumes are resized to a common cubic shape, meaning that the height, width, and depth of each volume are scaled to the same fixed value. This same resizing guarantees that all inputs have identical dimensions, which is particularly important when working with 3D convolutional neural networks, as it simplifies the design of the model and ensures spatial uniformity. After resizing, brightness normalization is applied as described in section 3.1.2, where pixel intensity values are scaled and clipped based on fixed percentile thresholds. The preprocessed volumes are then saved in NumPy format for downstream processing with only one »color« channel.

In the second stage, the corresponding annotation files are handled. Each annotation file contains a full voxel-wise segmentation map in which multiple segments or labels are defined (see also section 3.1.2). For the purposes of this work, only seven (including background) specific labels are retained. These selected segments represent the most structurally and semantically relevant parts of the shoe. The segmentation volumes are then rotated using the same transformation applied to the associated shoe volume to preserve alignment. Next, the label volumes are resized according to the spatial resolution required by the

segmentation head of the network, taking into account the number of convolutional layers and their respective downsampling effects.

Finally, the segmentation labels are stored in a single 3D NumPy array per shoe, where voxel values range from 0 (background) to 6, with each integer value uniquely representing one of the six selected shoe segments. This compact and standardized format ensures efficient loading during training and allows for direct compatibility with loss functions that expect integer-coded segmentation masks.

This study implemented the standard SegFormer with Mix-Vision Transformer (MVT) and SHViT backbones using PyTorch. TensorFlow was initially considered for the implementation. However, a memory leak was observed during its use, with the available GPU memory gradually decreasing over time. This issue has been documented in TensorFlow's GitHub issues and has persisted without resolution[5]. Notably, it is present since TensorFlow v2.11 but was not seen in v2.9. To mitigate this issue, the TensorFlow code of SegFormer was ported to PyTorch, which offers more transparent memory management and enhanced control over resource allocation. SHViT was already coded in PyTorch, thus requiring only minor adjustments to harmonize the interfaces for MVT and SHViT.

Furthermore, the code initially developed for 2D applications was restructured to also process 3D volumetric scans. A notable enhancement was the integration of the sliding-window attention mechanism, elaborated upon in the preceding section (see section 2.3). The original source code, taken from the Simple Vision Transformer (SimViT) GitHub repository[6], underwent slight modifications. Notably, the Python scripts are designed to function without installing additional libraries, such as `timm`[7], which are often used.

To enable experiments with SHViT focusing on variables such as kernel size and the use or omission of central self-attention, additional modifications were made to the initialization routines of the respective classes. Details of the entire code are available in the appendix B.1.7 on pages 61ff.

The `__init__` functions of the two SegFormer models, `SegFormer3D` (with MVT) and `SegFormer3D_SHViT`, were designed to accept the same input parameters, even if some were not used universally. This approach allows the models to be easily interchanged without requiring modifications to the model invocation in the main program.

[5] https://github.com/tensorflow/tensorflow/issues/61791
[6] https://github.com/ucasligang/SimViT/blob/main/classification/simvit.py
[7] https://pypi.org/project/timm/

The main program used for training is a Jupyter notebook file. In this file, the model is loaded, and the data loaders are initialized for training, validation, and test datasets with augmentation. Additionally, the functions for the F1-score metric and -loss are defined. The program includes callbacks for warm-up, early-stopping, logging, and saving model parameters if improvements in the validation metrics occur.

The training function begins with a warm-up phase reserved for the initial 20 epochs, during which the learning rate gradually increases to its target value. Following the warm-up, the actual training process starts, utilizing gradient accumulation with `BATCH_SIZE=1` and `ACCUMULATION_STEPS=5`. After the data loader is applied to the training data, validation is conducted using the validation dataset to evaluate F1-score and -loss.

Although data augmentation techniques are used to artificially expand the dataset, the underlying diversity remains limited to the original 40 shoe-samples. Cross-validation ensures that the models's ability to generalize is tested across different subsets of the original base shoes because normal train-test-splits can result in insufficient data for either training or validation purposes. Cross-validation, with `kFold=5` in this thesis, addresses this limitation by utilizing every sample for both training and validation across different folds, thereby maximizing the informational value extracted from available data. It is important that the same original image does not appear in both training and validation sets within the same fold [Koh95; Arl+09]. Known as »data leakage prevention«, this principle maintains the independence between training and test data, and it is fundamental to get unbiased performance estimates[8].

At the end of each epoch, the F1-score and -loss values for both training and validation are displayed. If no improvement in validation F1 is observed within a specified number of epochs, the learning rate is reduced and the internal counter is reset, and training continues. Should the metric still fail to improve within another set of epochs, early stopping is triggered and training is terminated.

Finally, curves for F1-score and -losses during training and validation are plotted. Subsequently, the best model, determined by the lowest validation loss, is loaded and used with the test dataset. All input images or volumes are loaded, segmentation predictions are generated, and the results are visualized by comparing the original input, the predicted segmentation, and the ground truth.

[8] see also section 2.4 of [Kap+22]

Chapter 4

Experimental Setup, and Evaluation Results

In this chapter, the experiments and results of the training and evaluation of the 3D shoe scan segmentation models are presented. Several experiments were conducted to analyze the performance and efficiency of different network configurations. The baseline model is the standard SegFormer architecture, which uses a MVT as backbone. This model is compared against two alternative configurations: SegFormer with SHViT as backbone, and a modified version of SHViT: Single-Head Vision Transformer with Central Self-Attention (SHViT with CSA) that incorporates a central self-attention mechanism designed to improve spatial focus but also reduces memory consumption.

The experiments are structured into five main parts:

(1) Backbone comparison: SegFormer with MVT, SHViT, and SHViT with CSA backbones are compared under various parameter settings. Model performance is evaluated using the F1-score as the primary accuracy metric, and GPU memory consumption is reported to assess computational efficiency.

(2) Volume test: The maximum input volume size that can be processed by the SHViT-based model under hardware constraints is determined, providing practical insights into their scalability.

(3) Convolution layer depth: The number of convolutional layers in the SHViT head is varied, and its impact on segmentation performance (in terms of F1 but also voxel resolution) is analyzed.

(4) Kernel size analysis: The effect of varying kernel sizes in both the central self-attention mechanism and the attention block of the SHViT model is investigated to understand their influence on model accuracy.

(5) Direct comparison with literature: The segmentation performance of the proposed SHViT-based model is compared against existing approaches with CNNs approaches in the literature [Lei23]. The evaluation focuses on benchmark metrics, particularly the F1-score for each segmented component on the same dataset, offering both qualitative and quantitative insights into how SHViT performs relative to CNN-based methods.

Together, these experiments aim to provide a comprehensive evaluation of transformer-based architectures for 3D volumetric segmentation of shoe scans, highlighting trade-offs between model complexity, accuracy, and computational requirements.

For each configuration in the experiment, six independent training runs were performed to account for variability due to random initialization and data augmentation. From these six simulations, the best five (based on validation performance) were selected. The final reported F1-score for each configuration is the average of these five values. Error bars in the plots in the following sections represent the standard deviation across these best five runs, providing a measure of the model's stability and robustness.

4.1 Backbone Comparison: SegFormers with MVT, SHViT, and SHViT with Central Self-Attention

The first experiment focuses on evaluating the impact of different backbone architectures on the segmentation performance of 3D shoe scans. As a baseline, the standard SegFormer model with a MVT backbone is used. This backbone has shown strong performance in 2D semantic segmentation tasks [Xie+21] and serves as a reference point for comparison.

To explore more efficient and potentially better-suited alternatives for volumetric data, the MVT backbone is replaced with SHViT, which is designed to better capture spatial dependencies in 3D data while maintaining computational efficiency. Furthermore, a modified version of SHViT is introduced, in which the conventional self-attention mechanism is replaced with a central self-attention block. This modification aims to enhance local spatial sensitivity and improve segmentation quality, particularly in regions with fine-grained structural details but also helps to reduce memory consumption.

All models in this experiment use a fixed input volume size of `(224,224,224)` voxels, and a segmentation output size of `(56,56,56)`, determined by the number of downsampling steps and convolutional layers. Specifically, a two-layer convolutional encoder head is used across all configurations to map the feature representations to the final segmentation mask. Larger volumes such as `(256,256,256)` exceed the available GPU memory of 24 GB when using the SHViT backbone with the B5/S4 configuration and a fair comparison would not be possible.

Each of these backbone configurations is tested using various hyperparameter settings to assess their robustness and performance under different conditions. The models are evaluated based on their segmentation accuracy, measured using the F1, their GPU memory consumption , and number of trainable parameters. This comparison provides insights into the trade-offs between model complexity, resource requirements, and segmentation quality when applying transformer-based architectures to 3D imaging data.

Figure 4.1 shows a plot of F1-score versus GPU memory consumption for the different backbone architectures and parameter configurations. Three distinct accuracy levels can be observed in the plot.

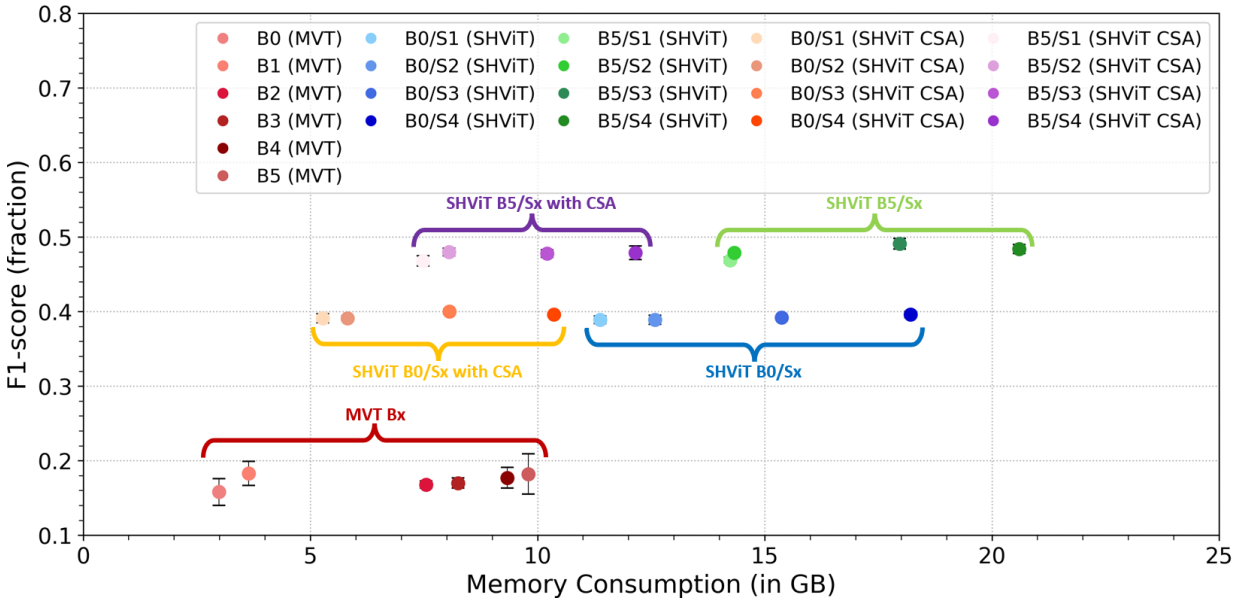

Figure 4.1: F1-score versus GPU memory consumption for different parameter sets for MVT, SHViT, and SHViT with central self-attention (CSA)

The lowest group, with F1 values between 0.1 and 0.2, corresponds to models using the MVT backbone, indicating limited segmentation performance on the 3D shoe scan task.

The second group, with F1-score around 0.4, results from experiments using the SHViT backbone with the B0/Sx[1] parameter settings, demonstrating improved accuracy compared to MVT while maintaining relatively low memory usage. The highest accuracy group, with F1 values slightly below 0.5, is achieved using SHViT with the B5/Sx configuration, suggesting that deeper and more expressive backbone variants lead to significantly better segmentation performance, but this comes with increased memory consumption.

While the standard SHViT models already outperform MVT in terms of accuracy, they tend to consume more GPU memory. However, introducing the central self-attention mechanism significantly reduces memory usage between 43% to 54% (for B0/Sx-configurations) and between 41% to 47% (for B5/Sx) without compromising segmentation performance. This highlights the central self-attention variant as a more efficient alternative, offering a favorable trade-off between computational cost and predictive accuracy.

An interesting observation from the experiment is the differing sensitivity of the two backbone types to their respective hyperparameter settings. For the standard SegFormer model with the MVT backbone, the F1 accuracy improves noticeably when progressing from the B0 to the B5 configuration. This indicates that increasing model capacity through deeper and wider transformer layers positively impacts segmentation performance. In contrast, the SHViT-based models show relatively stable F1 performance across the different Sx configurations (S1 to S4), regardless of whether B0 or B5 decoding dimensions are used. This suggests that the SHViT backbone is less sensitive to internal architectural scaling, and that even its smaller configurations are able to extract meaningful features from the 3D volumes effectively.

It was initially expected that the MVT-based SegFormer model would achieve significantly better results on the 3D shoe scan data, given its strong performance on 2D segmentation benchmarks. In 2D experiments, the MVT backbone consistently produced high IoU values [Xie+21]. However, in this study, the MVT model performed notably worse on the volumetric data, achieving only small IoU scores with 0.2 and below[2]. This performance gap suggests that while the MVT architecture is well-suited for 2D spatial relationships, it does not generalize effectively to 3D volumetric data without further architectural adaptations.

Another noteworthy observation is the significant difference in the number of trainable parameters between the MVT-based and SHViT-based models as shown in figure 4.2. For

[1] see table 2.1 on page 18.
[2] For this comparison the IoU was also calculated.

the MVT backbone, the number of parameters spans a wide range, from approximately 5 million in the B0 configuration up to around 120 million for B5. In contrast, all SHViT-based models, including those augmented with central self-attention, remain significantly more compact, with a maximum of approximately 20 million trainable parameters. This considerable reduction in model size highlights the efficiency of the SHViT architecture, which achieves comparable or superior segmentation performance while maintaining a much lower parameter count, making it particularly attractive for resource-constrained environments.

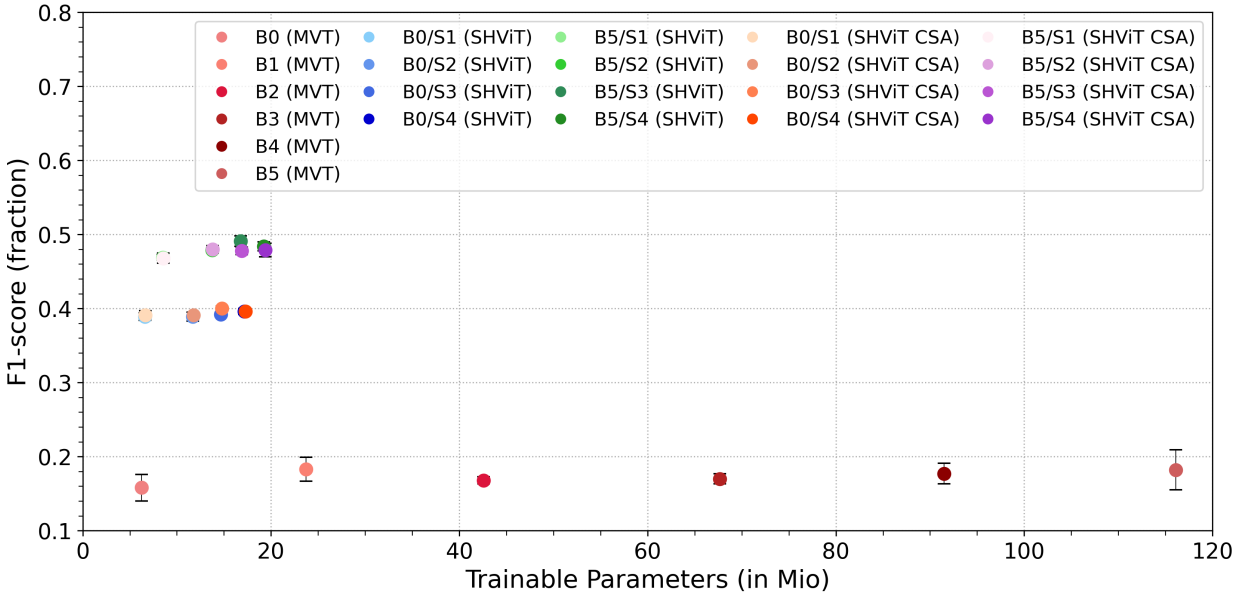

Figure 4.2: F1-score versus number of trainable parameters for different parameter sets for MVT, SHViT, and SHViT with central self-attention (CSA)

Some results of the trained segmentation model on selected shoe volumes are shown in figure 4.3 for SHViT-model with central self-attention and input volume size of `(224,224,224)`. For each case, a central slice of the 3D volume is displayed, showing from left to right: the input image, the predicted segmentation, and the corresponding ground truth. These examples illustrate the model's performance in identifying and segmenting the relevant components within the shoe scans.

For the subsequent experiments, the SHViT model with central self-attention and the B5/S2[3] configuration was selected, as it provides a favorable balance between segmentation accuracy and memory efficiency in the initial comparison.

[3]see table 2.1 on page 18.

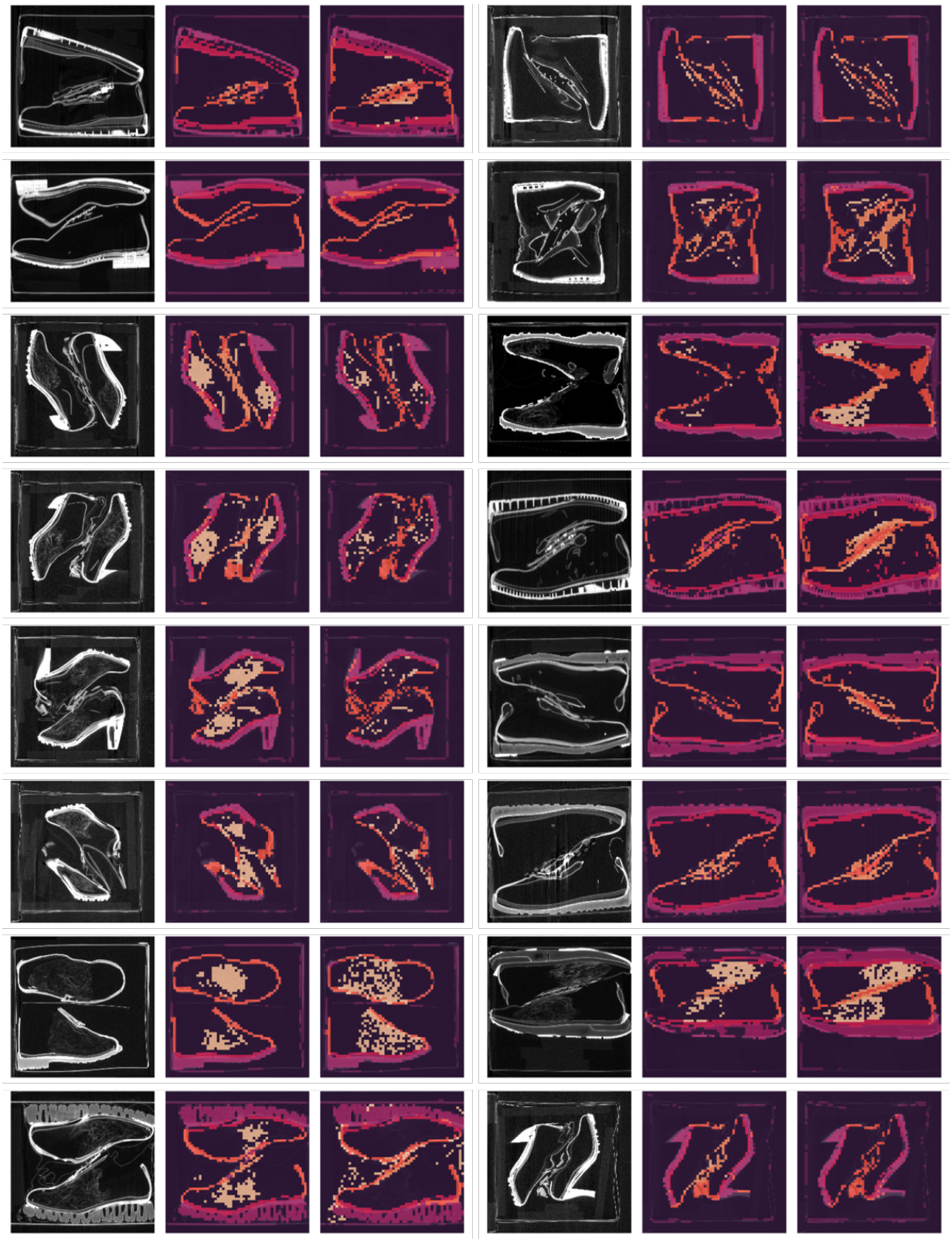

Figure 4.3: Results of some shoes trained with 3D-SHViT model with central self-attention. From left to right: the input image, the predicted segmentation, and the corresponding ground truth segmentation image for each of the two columns.

4.2 Evaluation of Maximum Volume Size for SHViT on a 24 GB GPU

In the second experiment, the objective is to determine the maximum input volume size that can be processed by SHViT-based models under given hardware limitations. Understanding the scalability of the architecture is crucial for real-world applications, where high-resolution volumetric data is often desirable but constrained by GPU memory capacity.

By systematically increasing the input dimensions while monitoring memory usage, this analysis identifies the practical upper bounds for volume size on a 24 GB GPU. The results provide valuable insights into the efficiency of the SHViT architecture and its suitability for processing large-scale 3D inputs, especially in comparison to more memory-intensive transformer backbones.

As demonstrated in the previous section, the SHViT model with central self-attention outperforms the other configurations in terms of segmentation accuracy and memory efficiency. Therefore, this model configuration is selected for the subsequent experiment to further explore its performance with different volume sizes.

Figure 4.4 shows the relationship between F1-score and GPU memory consumption for different input volume sizes from the SHViT B5/S2-model. It can be observed that for very small volumes (e.g., `(96,96,96)`, the F1-score is with 0.375 relatively low. As the volume size increases, the F1-score improves until it reaches a plateau at a volume size of `(256,256,256)`. After this point, the F1-score remains flat, with only a very slight increase observed starting from a volume size of `(320,320,320)`. The volume size of `(352,352,352)` was not tested due to memory limitations, as it would have increased GPU-memory usage to approximately 54 GB.

The lower F1-score at a volume size of `(96,96,96)` may be attributed to both the lower resolution of the input volume and the relatively small size of the segment volume, which has a resolution of `(24,24,24)`. This small segment volume limits the ability to capture fine details. As the volume input size increases, the segment volume increases as well, leading to better performance. The table 4.1 illustrates the volume sizes at each stage of the SHViT model with 2 convolution layers in SHViT head, which already reduces the input size by a factor of 4.

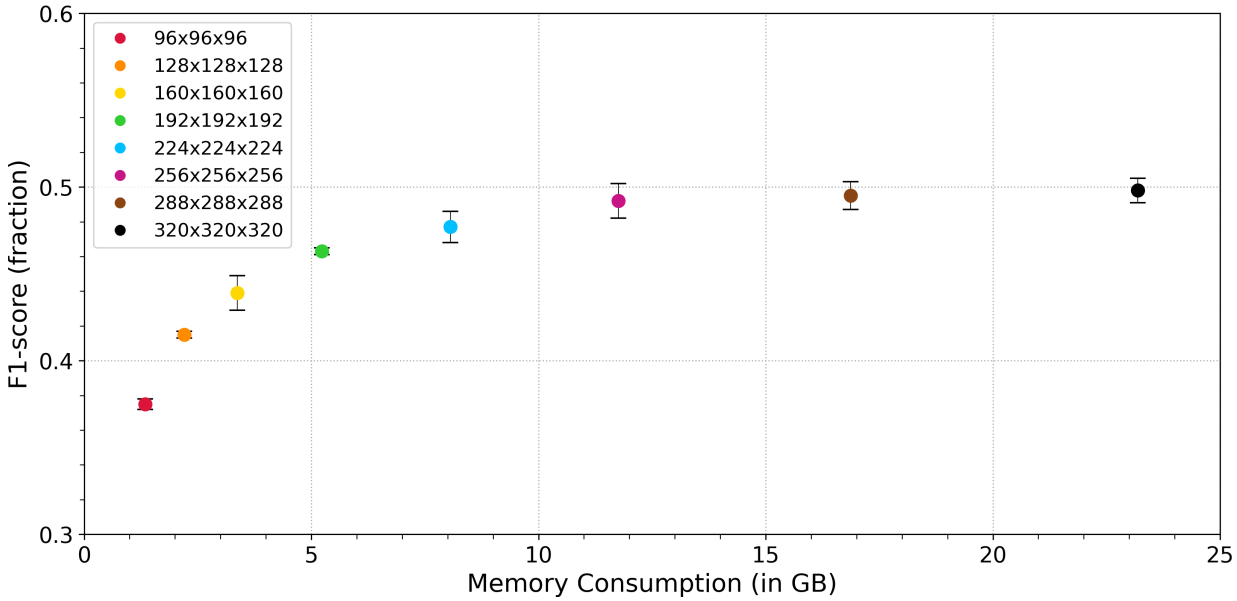

Figure 4.4: F1-score versus GPU memory consumption for different volume sizes. Base model is SHViT B5/S2 with central self-attention.

volume input size	size before 1st stage	size before 2nd stage	size before 3rd stage	size after 3rd stage	segment volume size	max. memory consumption
96	24	24	12	6	24	1.33 GB
128	32	32	16	8	32	2.19 GB
160	40	40	20	10	40	3.08 GB
192	48	48	24	12	48	5.35 GB
224	**56**	**56**	**28**	**14**	**56**	**7.94 GB**
256	64	64	32	16	64	11.81 GB
288	72	72	36	18	72	17.55 GB
320	80	80	40	20	80	23.19 GB
352	88	88	44	22	88	>50 GB

Table 4.1: Overview of input volume sizes and their corresponding resolutions after each stage of the SHViT B5/S2-model. The input volume is progressively reduced by convolutional layers (each with a stride of 2), and the resulting resolutions before/after each stage are shown. Between each stage is also a downsampling layer which reduces the volume by factor of 2 (see figure 2.6 for details). The final segmentation volume is constructed from the outputs in SegFormer decoder of the three stages of SHViT.

It is important to note that the segment output is derived from the last three stages of the SHViT model, meaning the input volume size of the first stage dictates the size of the segment output. Furthermore, not all volume sizes are feasible due to the structure of the model. After each stage, a convolution layer reduces the volume size by a factor of 2. Therefore, the output size at each stage should always be even (except for the last stage where it could also be uneven), as the upsampling in the MLP layer and subsequent concatenation with previous layers require matching dimensions.

The predicted segmentation images for three different input volume sizes are shown in figure 4.5.

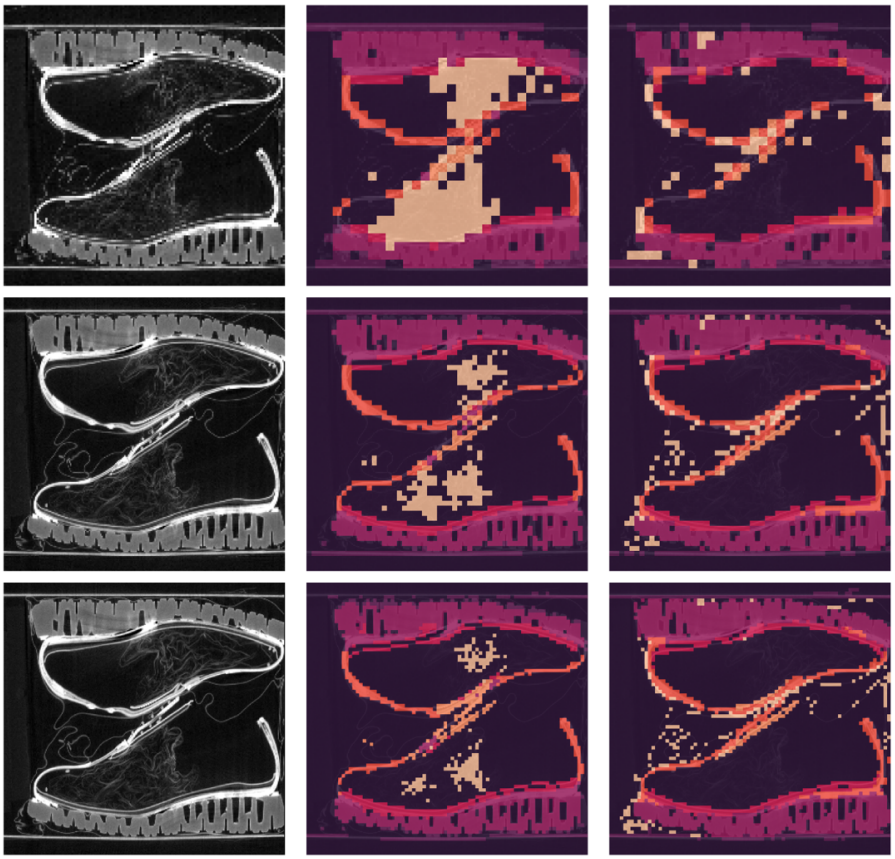

Figure 4.5: Comparison of the results with different volume sizes for `Shoepassion_40_down2_2_2`. From left to right: the input image, the predicted segmentation, and the corresponding ground truth segmentation image. From top to bottom: input sizes increases from `(128,128,128)` to `(224,224,224)` and `(320,320,320)`, with corresponding segmentation sizes from `(32,32,32)` to `(56,56,56)` and `(80,80,80)`. Notably, for this pair of shoes the predicted segmentation includes the filling material (visible inside the shoe) which is absent in the ground truth image, pointing to a potential inconsistency or omission in the labeling of this particular instance.

The visualizations clearly demonstrate how the resolution of the segmentation output varies with the input size. In particular, the coarseness of the segmentation at lower resolutions becomes evident, highlighting the limitations of small input volumes in accurately capturing fine structural details.

4.3 Impact of Convolutional Depth on F1-score

In this section, the influence of the number of convolutional layers in the SHViT segmentation head on both segmentation accuracy and GPU memory consumption is analyzed. The convolutional layers in the head are responsible for processing the output features from the backbone and generating the final segmentation volume. By adjusting the depth of these layers, the trade-off between model complexity, segmentation quality (measured by F1-score), and computational efficiency is explored.

Each convolutional layer in the SHViT head reduces the spatial dimensions of the input by a factor of 2 due to the use of a stride of 2. Consequently, the output shape after N convolutional layers can be calculated using the following formula:

$$\text{Output Shape} = \left(\frac{D}{2^N}, \frac{H}{2^N}, \frac{W}{2^N}\right) \tag{4.1}$$

where D, H, and W represent the depth, height, and width of the input volume, respectively, and N is the number of convolutional layers. This relationship is essential for selecting compatible input sizes and ensuring consistent output dimensions across experiments.

For this experiment, four different configurations of the SHViT head were evaluated, with the number of convolutional layers ranging from 1 to 3. To ensure comparability across configurations, the input volume size was chosen such that the resulting segmentation volume size remained constant, here: `(56,56,56)` for all cases. This approach avoids effects related to differences in output resolution, allowing a focused analysis of the impact of convolutional depth on segmentation performance and memory consumption. The following table 4.2 summarizes these configurations. Due to GPU memory constraints, the setup with four convolutional layers could not be executed and was therefore excluded from the experiment.

In addition to the high GPU memory consumption associated with using four convolutional layers and a large input volume, the resulting file size per shoe amounts to approximately 2.7 GB. This significantly increases the computational burden, not only in

terms of memory usage but also in terms of training time, as the large volume files must be read from the SSD for each epoch. This I/O overhead further limits the practicality of such configurations in resource-constrained environments.

number of conv layers N	volume input size	size after N conv layers	segment volume size	max. memory consumption
1	112	56	56	7.62 GB
2	**224**	**56**	**56**	**7.94 GB**
3	448	56	56	13.15 GB
4	896	56	56	>50 GB

Table 4.2: Overview of input volume sizes and their corresponding segmentation resolutions of the SHViT B5/S2-model for different number of convolutional layers.

The following figure 4.6 illustrates the relationship between F1-score and GPU memory consumption for three different SHViT head configurations. As expected, the configuration with only one convolutional layer yields the lowest accuracy, likely due to the reduced input resolution.

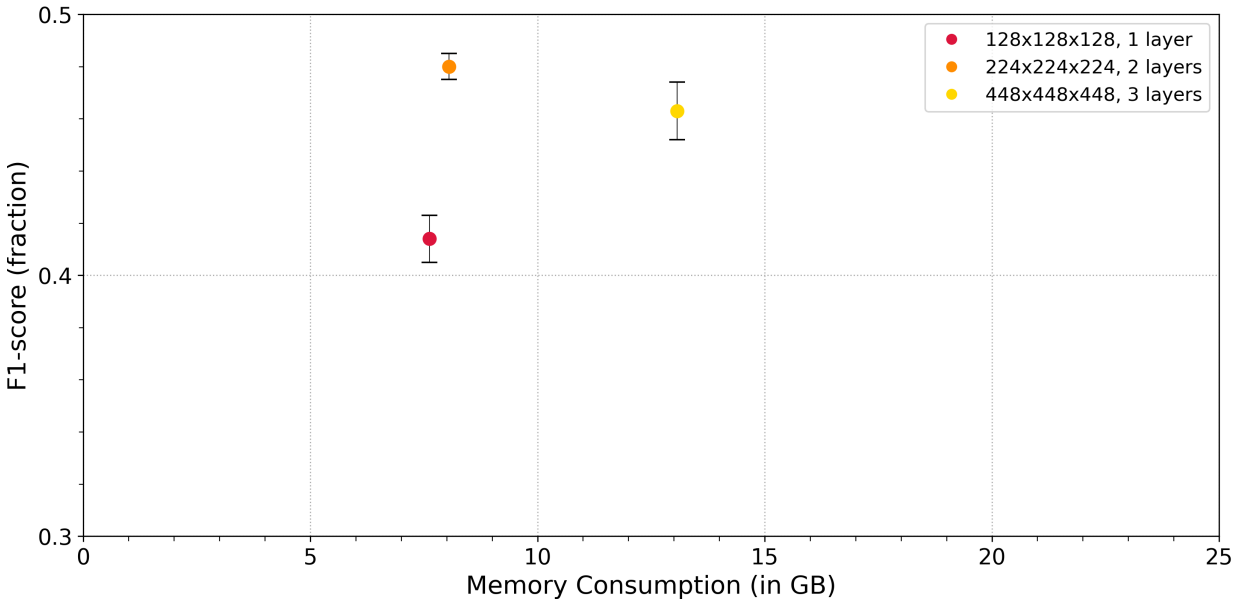

Figure 4.6: F1-score versus GPU memory consumption for different number of convolutional layers and volume sizes keeping segmentation volume same. The experiment with 4 convolutional layers was not executed due to very high memory demand.

Interestingly, the configuration with two convolutional layers outperforms the one with three layers, which was not anticipated. However, the results for the three-layer configuration exhibit a noticeably larger error bar compared to the others, indicating higher variance across runs. This suggests that the observed trend may not be reliable. Additional training runs could help to reduce the variance and provide more conclusive evidence regarding the optimal number of convolutional layers.

4.4 Effects of Kernel Size on F1-score and Memory

This experiment was conducted to examine the impact of the kernel size used in the central self-attention block on segmentation performance. The kernel size determines the spatial extent over which attention is applied and thus influences the model's ability to capture contextual information. In this experiment, different kernel sizes were evaluated to assess their effect on F1-score.

Kernel sizes of 1, 3, 5, and 7 were evaluated for the standard SHViT-SegFormer B5/S2-model with central self-attention and with 2 convolution layers and an input size of `(224,224,224)`, and the corresponding results illustrating the relationship between F1-score and GPU memory usage are shown in figure 4.7.

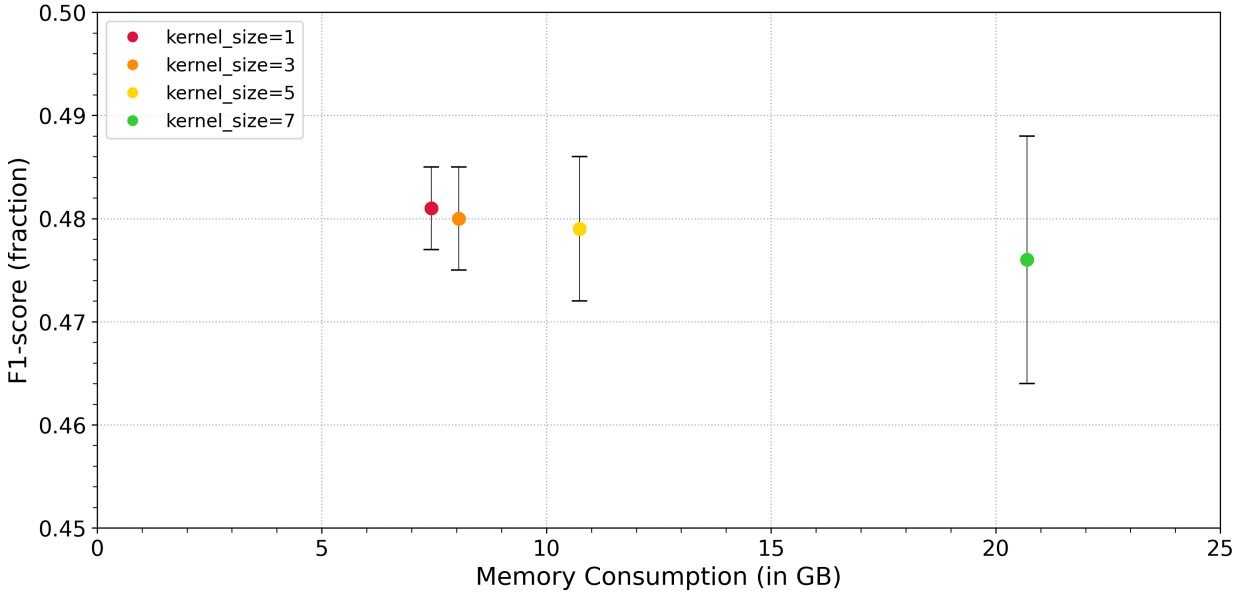

Figure 4.7: F1-score versus GPU memory consumption for different kernel sizes.

It can be observed that the F1-score value gradually decreases with increasing kernel size. The error bars become larger compared to the standard case with a kernel size of 1 and 3. While the training time per epoch is approximately 13 seconds for a kernel size of 1, it rises to 51 seconds for a kernel size of 5, and reaches as high as 415 seconds/epoch for a kernel size of 7.

A clear trend can be observed: the F1-score gradually decreases from `kernel_size=1` to `kernel_size=7`. The error bar for `kernel_size=7` is noticeably larger than for the other sizes. However, since the error bars overlap, it is difficult to confidently determine a clear winner in this experiment. Considering both the performance trend and computational cost, `kernel_size=1` or `3` represents the best compromise when treating this parameter as a tunable hyperparameter.

In conclusion, these results showed that smaller attention kernel sizes led to higher F1-score than larger ones, a result, that was not expected at first. However, literature suggests that smaller local attention windows are better at capturing fine structural details, which is especially important in 3D (medical) segmentation tasks. Studies such as [Hua+24; Wu+24; Guo+24] support this by showing that small-window attention helps preserve local texture and boundary precision, often leading to improved segmentation accuracy. While these results are well established for convolutional kernels, they suggest a similar benefit may extend to attention-window kernels as well.

As explained in section 2.3, the SHCSA block processes volumes of size `(28,28,28)`, which are already relatively small. In this context, larger kernel sizes are less effective at capturing fine details.

4.5 Segmentation performance: 3D SHViT Segformer versus Residual Squeeze and Excitation (SE) UNet

To evaluate the segmentation performance and validate the 3D SHViT Segformer model, four distinct experiments were carried out. The experimental framework was designed to perform side-by-side comparison between the Residual Squeeze and Excitation (SE) UNet [Lei23] and SHViT Segformer architecture across different segmentation tasks. The Performance evaluation is based on F1-score calculated for individual semantic segments.

This comparative analysis will provide insights into the relative strengths and limitations of convolutional versus transformer-based approaches for semantic segmentation tasks. To

ensure a fair and direct comparison, both models use identical input specifications with volumetric data of $256 \times 256 \times 256$ voxels. The SHViT model is configured with hyperparameter settings of B5/S2 with central self-attention, and a `kernel_size` of 3, representing a balanced configuration between model complexity and computational efficiency.

4.5.1 Experiments 1-3: Comparative Analysis with Published Work

The first three experiments were designed to replicate and compare 3D SHViT results with those reported in the existing literature [Lei23]. These experiments serve as a benchmark to validate the implementation and ensure reproducibility of previously published findings. Each experiment follows the same methodological framework as the reference paper, allowing for direct performance comparison under identical conditions. The following tables summarize the obtained results.

	Network	
F1-score	Residual SE UNet	SHViT Segformer
Total	0.821	0.717 ± 0.009
Background	0.991	0.958 ± 0.002
Shoe	0.882	0.748 ± 0.009
Packaging Material	0.589	0.445 ± 0.007

Table 4.3: Experiment 1: Comparison of segmentation performance (F1-scores) between the Residual SE UNet and SHViT Segformer models across three different classes. The UNet model consistently outperforms SHViT Segformer, particularly for the »Shoe« and »Packaging Material« categories. Data for Residual SE UNet are taken from Table 2 [Lei23].

The results indicate that in all three comparative cases, the SHViT model demonstrates suboptimal performance in detecting and segmenting the defined segmentation classes, revealing challenges in the transformer-based approach for this specific application domain. Since the total F1-score is the averaged score of all segmentation classes, a poor result in one (or more) classes has a significant impact on the final score.

Experiments 2 and 3 continue the comparative analysis using the same model architectures but with modified segmentation class definitions. In Experiment 2, the task is

simplified to a binary segmentation problem with only two classes, while Experiment 3 introduces a more complex setting with four distinct segmentation classes.

	Network	
F1-score	Residual SE UNet	SHViT Segformer
Total	0.902	0.841 ± 0.012
Background	0.994	0.980 ± 0.002
Inside Volume	0.810	0.701 ± 0.023

Table 4.4: Experiment 2: Comparison of segmentation performance (F1-scores) between the Residual SE UNet and SHViT Segformer models across two different classes. The UNet model achieves higher overall performance due to the better prediction of the »Inside Volume«. Data for Residual SE UNet are taken from Table 3 [Lei23].

	Network	
F1-score	Residual SE UNet	SHViT Segformer
Total	0.873	0.702 ± 0.003
Background	0.997	0.984 ± 0.000
Inner Sole	0.795	0.459 ± 0.012
Outer Sole	0.887	0.777 ± 0.006
Upper Material	0.813	0.586 ± 0.008

Table 4.5: Experiment 3: Comparison of segmentation performance (F1-scores) between the Residual SE UNet and SHViT Segformer models across four different classes. The F1-score of UNet model for the segments »Inner Sole« and »Upper Material« is higher then for SHViT Segformer model. Data for Residual SE UNet are taken from Table 4 [Lei23].

Also here the results of the experiments demonstrate that the performance of SHViT is not as good as UNet-CNN. A subsequent literature review provides further insight into this performance gap.

Convolutional networks like the Residual SE UNet generally outperform transformer-based models such as SHViT Segformer in semantic segmentation tasks, especially on small datasets. One of the main reasons is that CNNs having a strong inductive bias [Dos+21]. They are designed to recognize local patterns like edges and textures, which helps them

generalize well even from limited data. In contrast, Vision Transformers lack this bias and need much more training data to learn spatial relationships effectively [Ron+15; Ste+22].

Moreover, CNNs like UNet use skip connections that preserve high-resolution features, which is especially important for accurately segmenting small or fine-grained objects (e.g., shoe tongues or packaging material) [Zho+18b]. Transformers, when trained from scratch, tend to produce coarser outputs and struggle to capture such details. This implies that without large datasets or pretrained weights, ViTs struggle to capture fine structure, which supports the claim of coarser outputs and weaker detail segmentation. As a result, the UNet model performs more robustly in data-scarce, detail-sensitive segmentation scenarios [Ste+22].

4.5.2 Experiment 4: Standard 3D SHViT Segformer Model Evaluation

The fourth experiment uses a standard 3D SHViT model configuration to establish baseline performance metrics for the transformer-based segmentation approach. This experiment provides a reference point for evaluating the effectiveness of the hierarchical vision transformer architecture in the specific application domain.

	Network
F1-score	SHViT Segformer
Total	0.485 ± 0.009
Background	0.950 ± 0.009
Box	0.442 ± 0.024
Inner Sole	0.462 ± 0.010
Outer Sole	0.742 ± 0.009
Upper Material	0.522 ± 0.006
Tongue/Flap	0.156 ± 0.028
Filling Material	0.120 ± 0.013

Table 4.6: Experiment 4: Segmentation performance (F1-scores) of the SHViT Segformer across seven classes. Results for the Residual SE UNet are not available for this experimental setup. The model performs well on large structures like »Background« and »Outer Sole«, but struggles with fine-grained regions such as »Filling Material« and »Tongue / Flap«.

Small objects or specific components such as the shoe tongue or filling material exhibit poor segmentation performance, resulting in significantly low F1-scores for these classes. This poor performance in detecting and segmenting fine-detailed or smaller features has a substantial impact on the overall F1-score of the model, as these classes contribute disproportionately to the averaged total score.

It is important to note that the current results were obtained using only 40 pairs of shoes, split for training in a 5-fold cross-validation setup with data augmentation. Given the limited size of the dataset, these results should be considered as a baseline, providing a foundation for further investigations or improvements through larger datasets, refined annotations, or more advanced model architectures.

4.6 Recommended Model Configuration

Based on the investigations conducted and discussed in the preceding sections of this chapter, the following conclusions have been drawn and corresponding recommendations are provided: The model achieving the highest F1-accuracy, along with the most favorable accuracy-to-memory trade-off, is the SHViT-SegFormer with central self-attention configured with the B5/S2 parametrization[4] and two convolutional heads in the SHViT encoder head. The kernel size of the central self-attention layer is 3 due to its F1-score with low deviation. It achieves for an input volume of `(224,224,224)` an F1 of 0.480 ± 0.005, with a memory consumption of 8.05 GB and a training time of around 13 seconds per epoch. With this configuration input volumes up to `(320,320,320)` are possible.

[4] see also table 2.1 in section 2.2.7 on page 16

Chapter 5

Summary, Conclusions, and Future Work

In this thesis, existing two-dimensional (2D) Vision Transformer models, SegFormer with Mix-Vision Transformer (MVT) and Single-Head Vision Transformer (SHViT), were adapted to handle three-dimensional (3D) volumetric data. These models were specifically utilized to identify seven segments: Background, `Karton`, `Außensohle`, `Innensohle`, `Obermaterial`, `Zunge`, and `Füllmaterial` from the 3D shoe scans. The F1-score was employed as the evaluation metric. To address the problem of memory-intensive attention mechanisms, with a complexity of $O(n^2)$, central self-attention (sliding window) was implemented to enhance performance.

The modifications applied to SegFormer with MVT and SHViT have effectively facilitated the processing of 3D volumetric data for complex segmentation tasks, specifically targeting shoe scans. By incorporating central self-attention through a sliding window approach, the typical high memory demands of standard attention mechanisms were substantially reduced. This innovation not only extend the applicability of Vision Transformers but also establish a foundation for more efficient and scalable solutions within 3D data environments. This research highlights the potential for implementing advanced model architectures across new domains, offering valuable perspectives on integrating 2D model principles into 3D datasets. Future endeavors may focus deeper into optimizing attention mechanisms and expanding experimentation to include various types of 3D segmented data.

A series of experiments were conducted to compare the SegFormer model integrated with different backbones: MVT, SHViT, and SHViT with CSA evaluating both F1-score and GPU

memory consumption, along with varied hyperparameter configurations. The experiments revealed the superior performance of SHViT models over MVT, showing comparable F1-score between SHViT and the central self-attention equipped model. Nevertheless, the central self-attention model exhibited approximately 45% lower memory usage compared to the standard SHViT models.

In terms of hyperparameter configurations, Single-Head Vision Transformer with Central Self-Attention (SHViT with CSA) with two convolutional layers in the PatchEmbed block and B5/S2 configuration emerged as the most effective, demonstrating the optimal F1/GPU memory ratio. This configuration allowed for an increase in maximum volume dimensions up to `(320,320,320)`, while constraint memory usage remained within the 24 GB limit of the graphics card. Additionally, when adjusting kernel sizes, a kernel size of 1 displayed slightly superior potential relative to the standard variant with a kernel size of 3. The advantage of kernel size 1 lies in its precision in detail focus, whereas kernel size 3 and larger resulted in excessive convolution across the relatively small volume of `(28,28,28)` in the central self-attention block, decreasing efficiency.

These advancements not only broaden the functional scope of Vision Transformers but also enhance the efficiency and scalability of solutions within 3D data environments. The insights gained from this research contribute to the ongoing development of attention mechanisms and model architectures, paving the way for further exploration into diverse 3D data segmentation applications.

Future research could explore the implementation of additional Vision Transformer models as backbones and compare their performance with SHViT. Addressing the high memory demand of the attention block remains an area for investigation. New advanced techniques might offer further potential for reducing memory consumption. However, going beyond the currently maximal usable volume of `(320,320,320)` may prove challenging due to the constraints imposed by the 24 GB limit of the graphics card. For SHViT models with two convolutional layers in the PatchEmbed block within the SHViT head, the next larger volume would be `(352,352,352)`, but for sure hard to reach. Therefore, innovative strategies are needed to enable efficient scaling without surpassing hardware limitations.

Appendix A

Setting up Windows Subsystem for Linux (WSL) on Windows 11

To facilitate the execution of Linux-based software and tools in a Windows environment, Windows Subsystem for Linux (WSL) was set up on a Windows 11 machine. The following steps outline the process of setting up WSL, installing Miniconda, and creating a virtual environment to run PyTorch 2.60 with GPU support.

After downloading and installing Ubuntu 24.04 using Microsoft Store the following steps have to be carried out. Before step 4 is executed Miniconda had to be downloaded[1]:

```
1 sudo apt update -y
2 sudo apt upgrade -y
3 cd /mnt/c/Users/USERNAME/Downloads/
4 bash Miniconda3-latest-Linux-x86_64.sh
5 cd ~
6 ~/miniconda3/bin/conda init bash
7 ~/miniconda3/bin/conda init zsh
8 exit
```

The installation of PyTorch is very easy and does not require additional installation of CUDA drivers or other dependencies, unlike TensorFlow. PyTorch comes with built-in CUDA support[2], meaning it can be simply installed using `pip`, and it will automatically handle the necessary CUDA libraries.

[1] https://www.anaconda.com/download/success
[2] https://pytorch.org/get-started/locally/

```
conda create -n torch260 python=3.12
conda activate torch260
pip3 install torch torchvision torchaudio --index-url
https://download.pytorch.org/whl/cu126
conda install -c conda-forge opencv matplotlib tqdm seaborn pandas
plotly lightning
pip install torchsummary torchviz pydicom slicerio unfoldNd
sudo apt-get install graphviz
cd /mnt/c/Users/USERNAME/Documents/Python/shoes_segformer/Software/
pip install PythonTools-3.7.0-py2.py3-none-any.whl
conda install conda-forge::libsqlite --force-reinstall
conda install conda-forge::sqlite --force-reinstall
```

After setting up PyTorch, essential libraries like `matplotlib`, `tqdm`, `seaborn`, etc. needed for running the SegFormer scripts have to be installed. Additionally, `pydicom` and `slicerio` were included for handling the annotation files for the shoe files. To extract data from the original shoe volume data from the files in `.rek` dataformat, the `PythonTools` software was required.

While running the scripts in VSCode, the kernel occasionally crashed. This issue was resolved by reinstalling the `sqlite` and `libsqlite` packages at the end[3].

[3] https://stackoverflow.com/a/79484466/27900239

Appendix B

3D-Segmentation Models in Pytorch

The following code is a modification of the original implementation [IMv24], with adjustments and enhancements made to handle 3D image data effectively. In the following sections, the modified code for all modules needed to perform image segmentation on 3D images is shown.

The code for SHViT in section B.1.7 is based on the original implementation from [ysj24] and has been slightly modified to function as a standalone version without requiring additional package installations.

Note that in this thesis also code for 2D image segmentation is used but only the code for 3D segmentation is shown in the sections below. The 2D version can be implemented by removing all 3D-specific parts and replacing them with their 2D counterparts (i.e. `Conv3D` by `Conv2D`, or remove code with `D=D` and the lines that use the shape of depth).

The full source code used in this thesis is available on GitHub: https://github.com/CGD16/SHViT_Segformer/tree/main

B.1 Sourcecode for 3D-Segformer models

The main models are located in the file `segformer_3d.py`. Here the individual modules for Mix-Vision Transformer, SHViT and SegformerHead are combined to form the final Segformer models. The source code for the remaining modules is organized into separate program files, with links to the corresponding GitHub repository are provided below.

B.1.1 segformer_3d.py

The code implements the 3D segmentation models `SegFormer3D` and `SegFormer3D_SHViT`.

Key Components:

- `SegFormer3D`: The main segmentation model that integrates `MixVisionTransformer3D`, `SegFormerHead3D`, and `ResizeLayer3D`, and supports different configurations specified in `MODEL_CONFIGS`.
- `SegFormer3D_SHViT`: The main segmentation model integrates `SHViT3D`, `SegFormerHead3D`, and `ResizeLayer3D`, and supports different configurations specified in `MODEL_CONFIGS` and `SHVIT_CONFIG`.

The SegFormer model implementation, along with its variants used in this thesis, is available in the project's GitHub repository: https://github.com/CGD16/SHViT_Segformer/blob/main/models_torch/segformer_3d.py.

B.1.2 attention_3d.py

The 3D Attention Layer is a neural network module implemented using PyTorch. This layer is designed to handle 3-dimensional input data and apply a self-attention mechanism, which allows the network to focus on different parts of the input for each output. Code is based on [IMv24], it was modified for Pytorch implementation and adapted to support 3D segmentation.

Key features of the `Attention3D` Layer:

- Multi-head Attention: This allows the model to jointly attend to information from different representation subspaces at different positions, enhancing its ability to capture various features of the input data.
- Spatial Reduction: Optionally, the input data can be spatially reduced, which can help in managing large input dimensions and improving computational efficiency.
- Query, Key, and Value Projections: The layer uses linear projections to create queries, keys, and values from the input data, which are essential components of the attention mechanism.
- Dropout Mechanism: Dropout is applied both to the attention weights and the final projections to prevent overfitting and to improve the model's generalization abilities.

- Layer Normalization: When spatial reduction is applied, layer normalization helps in stabilizing and speeding up the training process.

The code for 3D Attention Layer is available in the project's GitHub repository: https://github.com/CGD16/SHViT_Segformer/blob/main/models_torch/attention_3d.py.

B.1.3 head_3d.py

The code provides implementations of various neural network modules to handle 3-dimensional data and includes a Multi-Layer Perceptron (MLP) layer, a Convolutional module, and a segmentation head based on the SegFormer architecture. Code is based on [IMv24], it was modified for Pytorch implementation and adapted to support 3D segmentation.

Key Components:

- `MLP3D` Layer: This module performs a simple linear projection, effectively transforming the input tensor's dimensions.
- `ConvModule3D`: A 3D Convolutional module that applies convolution, batch normalization, and ReLU activation to the input tensor.
- `SegFormerHead3D`: A segmentation head designed for processing 3D tensors. It uses MLP layers to transform features and a convolutional layer to fuse them, followed by dropout and final prediction layers.

Implementation details of the modules for processing 3D data, including a MLP layer, a convolutional block, and a SegFormer-based segmentation head, can be found in the project's GitHub repository: https://github.com/CGD16/SHViT_Segformer/blob/main/models_torch/models_torch/head_3d.py.

B.1.4 modules_3d.py

The code describes a 3D Vision Transformer model with various neural network modules, including a depth-wise convolution layer, a multi-layer perceptron (MLP) with depth-wise convolution, a transformer block with multi-head self-attention, overlapping patch embedding, and the main transformer model is available at the project's GitHub repository: https://github.com/CGD16/SHViT_Segformer/blob/main/models_torch/models_torch/modules_

3d.py. Code is based on [IMv24], it was modified for Pytorch implementation and adapted to support 3D segmentation.

Key Components:

- `DWConv3D`: A depth-wise convolution layer designed to process 3D data, applying convolution across each spatial dimension.
- `Mlp3D`: A multi-layer perceptron that includes linear projections, depth-wise convolution, activation, and dropout mechanisms.
- `Block3D`: A transformer block that combines multi-head self-attention with an MLP, including layer normalization and dropout.
- `OverlapPatchEmbed3D`: A patch embedding layer that creates overlapping patches from the input 3D data, transforming spatial dimensions into sequence representations.
- `MixVisionTransformer3D`: The main transformer model that combines patch embedding and multiple transformer blocks to process 3D data, enabling the extraction of hierarchical features from the input.

B.1.5 `utils_3d.py`

The code defines two utility layers aimed at enhancing the performance and usability of 3D neural network models in PyTorch. These layers include a layer for resizing 3D input tensors and another for applying the DropPath regularization technique to improve model generalization. Code is based on [IMv24], it was modified for Pytorch implementation and adapted to support 3D segmentation.

Key Components:

- `ResizeLayer3D`: This layer resizes 3D input tensors to a target depth, height, and width using trilinear interpolation.
- `DropPath3D`: This layer applies the stochastic depth (DropPath) regularization technique, which randomly drops paths during training to prevent overfitting.

The code is accessible via the GitHub repository: https://github.com/CGD16/SHViT_Segformer/blob/main/models_torch/utils_3d.py.

B.1.6 `center_attention_3d.py`

In the code a local (kernel-based) central self-attention over a 3D volume is implemented. Instead of attending globally to every voxel, it:

- Slides a cubic window (k kernel) over the volume. Each query only attends its local k^3 neighborhood (rather than the full volume).
- For each center position, computes Query (Q) from that center, and Key (K)/Value (V) from its local neighborhood.
- Performs scaled dot-product attention within each cube.
- Outputs an updated feature for each center voxel.

The code is very efficient because it is using `unfold` and batched Q/KV to reduce memory compared to a full self-attention function. Code is based on [Li+22a], it was modified for Pytorch implementation and adapted to support 3D segmentation and is available here: https://github.com/CGD16/SHViT_Segformer/blob/main/models_torch/center_attention_3d.py

B.1.7 `shvit_3d.py`

The code represents a modified version of a PyTorch module designed for visual processing tasks, such as image classification or detection, utilizing the Vision Transformer architecture. Code is based on [Yun+24], it was modified and adapted to support 3D segmentation.

Key Components:

- `GroupNorm` Class: Implements Group Normalization for input tensors. Normalizes the input tensor across spatial dimensions (height and width).
- `Conv3d_BN` Class: Combines a 3D Convolutional layer followed by Batch Normalization. Includes an optional activation function and the ability to fuse convolution and batch normalization layers.
- `make_divisible` Function: Ensures that a value is divisible by a specified divisor, which is useful for network architectures that require certain quantities (like the number of channels) to be divisible by a specific number.

- `SqueezeExcite3D` Class: Implements the Squeeze-and-Excitation (SE) module, which recalibrates input features based on their inter-channel dependencies.
- `PatchMerging3D` Class: Implements a patch merging mechanism using convolutional layers to reduce the spatial dimensions of the input tensor while increasing the number of channels.
- `Residual3D` Class: Implements a residual connection for layers, optionally including dropout.
- `FFN3D` Class: Defines a Feed-Forward Network (FFN) with two `Conv3d_BN` layers and a `ReLU` activation function between them.
- `SHSA3D` Class: Defines a Single-Head Self-Attention (SHSA) mechanism for processing partial input dimensions.
- `SHCSA3D` Class: Defines the modified SHSA mechanism with central self-attention for processing partial input dimensions.
- `BasicBlock3D` Class: Represents a basic block for the SHViT architecture, combining convolutional layers, self-attention, and feed-forward networks through residuals.
- `SHViT3D` Class: Incorporates the main SHViT model with hierarchical stages. Contains patch embedding mechanism and multiple stages, each consisting of blocks that process the input tensor through convolutional, self-attention, and feed-forward layers. The forward method processes the input tensor through these stages and returns the output tensors for different stages.

This code implements a Vision Transformer model with hierarchical stages, incorporating custom features like Group Normalization, Squeeze-and-Excitation, and Residual connections: https://github.com/CGD16/SHViT_Segformer/blob/main/models_torch/shvit_3d.py. The goal is to enhance the Vision Transformer's performance by integrating these additional components for improved feature processing and representation.

List of Abbreviations

2D two-dimensional

3D three-dimensional

AViT Adapting Vision Transformer

CSA Center Self-Attention

CNN Convolutional Neural Networks

CPU Central Processing Unit

CT Computed Tomography

EfficientViT Efficient Vision Transformer

FFN Feed-Forward Network

FPN Feature Pyramid Network

ID Identification

IoU Intersection of Union

GPU Graphics Processing Unit

K Key

LSTM Long Short-Term Memory

MHSA Multi-Head Self-Attention

MHSA Multi-Head Central Self-Attention

mIoU mean Intersection of Union

MLP Multi-Layer Perceptron

MVT Mix-Vision Transformer

NLP Natural Language Processing

PVT Pyramid Vision Transformer

Q Query

ReLU Rectified Linear Unit

RNN Recurrent Neural Network

SE Squeeze-and-Excitation

SHCSA Single-Head Central Self-Attention

SHSA Single-Head Self-Attention

SHViT Single-Head Vision Transformer

SHViT with CSA Single-Head Vision Transformer with Central Self-Attention

SimViT Simple Vision Transformer

SSD Solid-State-Drive

TeViT Temporally Efficient Vision Transformer

UNETR UNEt TRansformers

V Value

VIS Video Instance Segmentation

ViT Vision Transformer

List of Figures

List of Tables

Bibliography

[Alo+23] K. Alomar, H. I. Aysel, and X. Cai. Data Augmentation in Classification and Segmentation: A Survey and New Strategies. *Journal of Imaging*, 9(2):46, 2023. URL: https://doi.org/10.3390/jimaging9020046 (cited on pp. 30, 32).

[Arl+09] S. Arlot and A. Celisse. A Survey of Cross Validation Procedures for Model Selection, July 2009. DOI: 10.1214/09-SS054. URL: https://www.researchgate.net/publication/45864474_A_Survey_of_Cross_Validation_Procedures_for_Model_Selection (cited on p. 34).

[Aza+23] R. Azad, M. Heidary, K. Yilmaz, M. Hüttemann, S. Karimijafarbigloo, Y. Wu, A. Schmeink, and D. Merhof. Loss Functions in the Era of Semantic Segmentation: A Survey and Outlook, 2023. URL: https://arxiv.org/pdf/2312.05391 (cited on pp. 22–24).

[Bel+20] I. Beltagy, M. E. Peters, and A. Cohan. Longformer: The Long-Document Transformer, April 2020. URL: https://arxiv.org/abs/2004.05150 (cited on p. 21).

[Bra+25] R. Bravin, M. Pavan, H. Shalby, F. Pittorino, and M. Roveri. EmbBERT-Q: Breaking Memory Barriers in Embedded NLP, February 2025. URL: https://arxiv.org/pdf/2502.10001 (cited on p. 20).

[Chi+19] R. Child, S. Gray, A. Radford, and I. Sutskever. Generating Long Sequences with Sparse Transformers, 2019. URL: https://arxiv.org/pdf/1904.10509 (cited on p. 21).

[Das+25] B. K. Das, A. Singh, G. Zhao, H. Liu, T. J. Re, D. Comaniciu, E. Gibson, and A. Maier. VIViT: Variable-Input Vision Transformer Framework for 3D MR Image Segmentation, 2025. URL: https://arxiv.org/abs/2505.08693 (cited on p. 2).

[Dee20] DeepSpeed. DeepSpeed Sparse Attention, September 2020. URL: https://www.deepspeed.ai/2020/09/08/sparse-attention.html (cited on p. 21).

[Dos+21] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *9th International Conference on Learning Representations (ICLR)*. OpenReview.net, 2021. URL: https://openreview.net/forum?id=YicbFdNTTy (cited on pp. 1, 49).

[Du+23] S. Du, N. Bayasi, G. Hamarneh, and R. Garbi. AViT: Adapting Vision Transformers for Small Skin Lesion Segmentation Datasets. In M. E. Celebi, M. S. Salekin, H. Kim, S. Albarqouni, C. Barata, A. Halpern, P. Tschandl, M. Combalia, Y. Liu, G. Zamzmi, J. Levy, H. Rangwala, A. Reinke, D. Wynn, B. Landman, W.-K. Jeong, Y. Shen, Z. Deng, S. Bakas, X. Li, C. Qin, N. Rieke, H. Roth, and D. Xu, editors, *Medical Image Computing and Computer Assisted Intervention – MICCAI 2023 Workshops*, pages 25–36, Cham. Springer Nature Switzerland, 2023. ISBN: 978-3-031-47401-9. URL: https://link.springer.com/chapter/10.1007/978-3-031-47401-9_3 (cited on pp. 8–10, 15).

[Fed+12] A. Fedorov, R. Beichel, J. Kalpathy-Cramer, J. Finet, J.-C. Fillion-Robin, S. Pujol, C. Bauer, D. Jennings, F. Fennessy, M. Sonka, J. Buatti, S. Aylward, J. V. Miller, S. Pieper, and R. Kikinis. 3D Slicer as an Image Computing Platform for the Quantitative Imaging Network. *Magnetic Resonance Imaging*, 30(9):1323–1341, November 2012. DOI: 10.1016/j.mri.2012.05.001. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC3466397/pdf/nihms383480.pdf (cited on p. 25).

[Fu+25] Z. Fu, W. Song, Y. Wang, X. Wu, Y. Zheng, Y. Zhang, D. Xu, X. Wei, T. Xu, and X. Zhao. Sliding Window Attention Training for Efficient Large Language Models, February 2025. URL: https://arxiv.org/abs/2502.18845 (cited on pp. 21, 22).

[Gan+25] D. Gan, M. Chang, and J. Chen. 3D-EffiViTCaps: 3D Efficient Vision Transformer with Capsule for Medical Image Segmentation. In A. Antonacopoulos, S. Chaudhuri, R. Chellappa, C.-L. Liu, S. Bhattacharya, and U. Pal, editors, *Pattern Recognition*, pages 141–156, Cham. Springer Nature Switzerland, 2025.

ISBN: 978-3-031-78186-5. URL: https://arxiv.org/abs/2403.16350 (cited on p. 2).

[Guo+24] B. Guo, N. Cao, R. Zhang, and P. Yang. SCENet: Small Kernel Convolution with Effective Receptive Field Network for Brain Tumor Segmentation. *Applied Sciences*, 14(23), 2024. ISSN: 2076-3417. URL: https://www.mdpi.com/2076-3417/14/23/11365 (cited on p. 47).

[Hao+21] R. Hao, K. Namdar, L. Liu, M. A. Haider, and F. Khalvati. A Comprehensive Study of Data Augmentation Strategies for Prostate Cancer Detection in Diffusion-Weighted MRI Using Convolutional Neural Network. *Journal of Digital Imaging*, 34:862–876, 2021. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC8455796/pdf/10278_2021_Article_478.pdf (cited on pp. 30, 32).

[Hat+22] A. Hatamizadeh, Y. Tang, V. Nath, D. Yang, A. Myronenko, B. A. Landman, H. R. Roth, and D. Xu. UNETR: Transformers for 3D Medical Image Segmentation. In *Proc. IEEE Winter Conf. on Applications of Computer Vision (WACV)*, pages 1748–1758. IEEE, 2022. URL: https://arxiv.org/pdf/2103.10504 (cited on p. 1).

[Hua+24] X. Huang, Y. Zhu, M. Shao, M. Xia, X. Shen, P. Wang, and X. Wang. Dual-branch Transformer for semi-supervised medical image segmentation. *Journal of Applied Clinical Medical Physics*, 25, August 2024. URL: https://www.researchgate.net/publication/383057595_Dual-branch_Transformer_for_semi-supervised_medical_image_segmentation (cited on p. 47).

[IMv24] IMvision12. Segformer-tf, 2024. URL: https://github.com/IMvision12/SegFormer-tf.git. commit #c59819 (cited on pp. 17, 18, 57–60).

[Jol+24] L. Jollans, M. Bustamante, L. Henriksson, A. Persson, and T. Ebbers. Vision Transformers increase efficiency of 3D cardiac CT multi-label segmentation, 2024. URL: https://arxiv.org/abs/2310.09099 (cited on p. 2).

[Kap+22] S. Kapoor and A. Narayanan. Leakage and the Reproducibility Crisis in ML-based Science, July 2022. URL: https://arxiv.org/abs/2207.07048 (cited on p. 34).

[Ke+21] L. Ke, M. Danelljan, X. Li, Y.-W. Tai, C.-K. Tang, and F. Yu. Mask Transfiner for High-Quality Instance Segmentation, 2021. URL: https://arxiv.org/abs/2111.13673 (cited on p. 6).

[Kir+19] A. Kirillov, R. Girshick, K. He, and P. Dollár. Panoptic Feature Pyramid Networks, 2019. URL: https://arxiv.org/abs/1901.02446 (cited on pp. 6, 7).

[Koh95] R. Kohavi. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection, January 1995. URL: https://www.ijcai.org/Proceedings/95-2/Papers/016.pdf (cited on p. 34).

[Lei23] Leipert, Martin and Herl, Gabriel and Stebani, Jannik and Zabler, Simon and Maier, Andreas. Three-Step Volumetric Segmentation for Automated Shoe Fitting. *e-Journal of Nondestructive Testing*, 28(3), March 2023. ISSN: 1435-4934. URL: http://dx.doi.org/10.58286/27736 (cited on pp. 25, 26, 36, 47–49).

[Li+22a] G. Li, D. Xu, X. Cheng, L. Si, and C. Zheng. SimViT: Exploring a Simple Vision Transformer with Sliding Windows. In *2022 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6, 2022. URL: https://arxiv.org/pdf/2112.13085 (cited on pp. 21, 61).

[Li+22b] Y. Li, G. Yuan, Y. Wen, J. Hu, G. Evangelidis, S. Tulyakov, Y. Wang, and J. n. EfficientFormer: Vision Transformers at MobileNet Speed, 2022. URL: https://arxiv.org/abs/2206.01191 (cited on p. 13).

[Liu+22] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie. A ConvNet for the 2020s, 2022. URL: https://arxiv.org/abs/2201.03545 (cited on p. 13).

[Liu+23] X. Liu, H. Peng, N. Zheng, Y. Yang, H. Hu, and Y. Yuan. EfficientViT: Memory Efficient Vision Transformer With Cascaded Group Attention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14420–14430, June 2023. URL: https://openaccess.thecvf.com/content/CVPR2023/papers/Liu_EfficientViT_Memory_Efficient_Vision_Transformer_With_Cascaded_Group_Attention_CVPR_2023_paper.pdf (cited on pp. 8, 11, 12, 15).

[Per+24] S. Perera, P. Navard, and A. Yilmaz. SegFormer3D: An Efficient Transformer for 3D Medical Image Segmentation, 2024. URL: https://arxiv.org/abs/2404.10156 (cited on pp. 1, 8, 16).

[Ron+15] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation, 2015. URL: https://arxiv.org/abs/1505.04597 (cited on p. 50).

[She+20] S. Shen, Z. Dong, J. Ye, L. Ma, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer. Q-BERT: Hessian Based Ultra Low Precision Quantization of BERT. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):8815–8821, 2020. URL: https://doi.org/10.1609/aaai.v34i05.6409 (cited on p. 20).

[Ste+22] A. Steiner, A. Kolesnikov, X. Zhai, R. Wightman, J. Uszkoreit, and L. Beyer. How to train your ViT? Data, Augmentation, and Regularization in Vision Transformers, 2022. URL: https://arxiv.org/abs/2106.10270 (cited on p. 50).

[Sut+24] K. Sutassananon, W. Kusakunniran, M. Orgun, and T. Siriapisith. 3D augmentation for volumetric whole heart segmentation. *Scientific Reports*, 14(1):21459, 2024. URL: https://doi.org/10.1038/s41598-024-72469-x (cited on pp. 30, 32).

[Tan+22] Y. Tang, D. Yang, W. Li, H. R. Roth, B. Landman, D. Xu, V. Nath, and A. Hatamizadeh. Self-Supervised Pre-Training of Swin Transformers for 3D Medical Image Analysis. In *Proc. IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 20730–20740, 2022. URL: https://openaccess.thecvf.com/content/CVPR2022/papers/Tang_Self-Supervised_Pre-Training_of_Swin_Transformers_for_3D_Medical_Image_Analysis_CVPR_2022_paper.pdf (cited on p. 1).

[Vas+23a] P. K. A. Vasu, J. Gabriel, J. Zhu, O. Tuzel, and A. Ranjan. FastViT: A Fast Hybrid Vision Transformer using Structural Reparameterization, 2023. URL: https://arxiv.org/abs/2303.14189 (cited on p. 13).

[Vas+23b] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention Is All You Need. https://arxiv.org/abs/1706.03762, 2023. arXiv: 1706.03762 [cs.CL] (cited on p. 18).

[Wu+24] X. Wu, X. Chen, Z. Gao, S. Qu, and Y. Qiu. Few-Shot Medical Image Segmentation with Large Kernel Attention, 2024. URL: https://arxiv.org/abs/2407.19148 (cited on p. 47).

[Xie+21] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez, and P. Luo. SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 12077–12090. Curran Associates, Inc., 2021. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/64f1f27bf1b4ec22924fd0acb550c235-Paper.pdf (cited on pp. 8, 9, 15–17, 36, 38).

[Yan+22] S. Yang, X. Wang, Y. Li, Y. Fang, J. Fang, W. Liu, X. Zhao, and Y. Shan. Temporally Efficient Vision Transformer for Video Instance Segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2885–2895, June 2022. URL: https://openaccess.thecvf.com/content/CVPR2022/papers/Yang_Temporally_Efficient_Vision_Transformer_for_Video_Instance_Segmentation_CVPR_2022_paper.pdf (cited on pp. 8, 12, 13, 15).

[ysj24] ysj9909/SHViT. SHViT: Single-Head Vision Transformer with Memory Efficient Macro Design. *GitHub repository*, 2024. URL: https://github.com/ysj9909/SHViT/blob/main/model/shvit.py. commit #6a729c (cited on p. 57).

[Yun+24] S. Yun and Y. Ro. SHViT: Single-Head Vision Transformer with Memory Efficient Macro Design. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5756–5767, June 2024. URL: https://openaccess.thecvf.com/content/CVPR2024/papers/Yun_SHViT_Single-Head_Vision_Transformer_with_Memory_Efficient_Macro_Design_CVPR_2024_paper.pdf (cited on pp. 8, 13–18, 61).

[Zho+18a] B. Zhou, H. Zhao, X. Puig, T. Xiao, S. Fidler, A. Barriuso, and A. Torralba. Semantic Understanding of Scenes through the ADE20K Dataset, 2018. URL: https://arxiv.org/abs/1608.05442 (cited on p. 6).

[Zho+18b] Z. Zhou, M. M. Rahman Siddiquee, N. Tajbakhsh, and J. Liang. UNet++: A Nested U-Net Architecture for Medical Image Segmentation. In July 2018. URL: https://www.researchgate.net/publication/324574642_UNet_A_Nested_U-Net_Architecture_for_Medical_Image_Segmentation (cited on p. 50).

[Zho+24] T. Zhou, W. Xia, F. Zhang, B. Chang, W. Wang, Y. Yuan, E. Konukoglu, and D. Cremers. Image Segmentation in Foundation Model Era: A Survey, 2024. URL: https://arxiv.org/abs/2408.12957 (cited on p. 5).