Institute of Imaging and Computer Vision | RWTH AACHEN UNIVERSITY

Development of a Scalable and Robust Deep Learning-Based Method for 3D Cell Segmentation in Microscopy Data

Master Thesis

presented by

Daniel Geiger, B.Sc.

Supervisor:
Zhu Chen, M.Sc

Institute of Imaging & Computer Vision
Prof. Dr.-Ing. Johannes Stegmaier
RWTH Aachen University

Erklärung nach §18 Abs. 1 ÜPO

Hiermit versichere ich, dass ich die vorgelegte Master Thesis selbständig angefertigt habe. Es sind keine anderen als die angegebenen Quellen und Hilfsmittel benutzt worden. Zitate wurden kenntlich gemacht.

I hereby confirm that I have written this Master Thesis independently using no sources or aids other than those indicated. I have appropriately declared all citations.

Daniel Geiger, B.Sc.
Aachen, 42.13.2037

Contents

List of Figures

List of Tables

List of Listings

Nomenclature

1 Introduction

Dies ist eine Vorlage zur Masterarbeit am LfB. Oder Bachelorarbeit. Jenachdem, wofür gerade du sie missbrauchen möchtest.

Du siehst hier auch die übliche Struktur: Einleitung und dann State of the Art. Danach sinnvollerweise ein Kapitel zu Methoden, eins zu Ergebnissen, eins zur Auswertung und dann noch Zusammenfassung / Ausblick.

Ich versuche in dieser Vorlage mal ein bisschen Beispielcode zu geben. In den Tex-Files (insbesondere in der thesis.tex) finden sich aber noch viel mehr Anmerkungen.

2 Related Work

In this chapter, we introduce the main ideas and methods that will form the foundation for our approach and evaluation in Chapter 3. First, we will capture the fundamental concepts of Deep Learning in an image segmentation context. Regarding this, we will introduce Multilayer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs) alongside the basic concepts of optimization theory. Afterwards, we will go over the way the so called Transformer Networks use attention mechanisms, to create a new way of feature representation of sequenced input data. This Transformer Network architecture is then extended to handle image data. Some extensions are then introduced, that are adopted in @todo

2.1 Convolutional Neural Networks (CNNs)

Convolutional neural networks (CNNs) are a class of deep learning models designed to learn features directly from data, making them especially effective for inputs with spatial structure [1]. Rather than relying on handcrafted descriptors, CNNs acquire representations through convolutional layers, where small filters slide across the input to detect local patterns. Early layers typically respond to simple structures such as edges or gradients, while deeper layers combine these responses into progressively more abstract features, including textures, shapes, or semantic concepts.

To expand their representational power, convolutional operations are followed by non-linear activation functions, most commonly the rectified linear unit (ReLU). These non-linearities prevent the network from collapsing into a purely linear model and allow it to approximate complex functions. Pooling layers are often included to reduce the spatial resolution of feature maps by summarizing local neighborhoods. This downsampling lowers computational cost, introduces translational invariance, and emphasizes the most salient information. Together, convolutions, non-linearities, and pooling operations form a hierarchy that transforms raw data into increasingly abstract and task-relevant features.

Additional design choices such as batch normalization, dropout, and skip connections further improve stability, regularization, and information flow. Nonetheless, the core strength of CNNs lies in their hierarchical feature acquisition, which enables robust generalization and has established them as a fundamental architecture in modern machine learning across diverse domains.

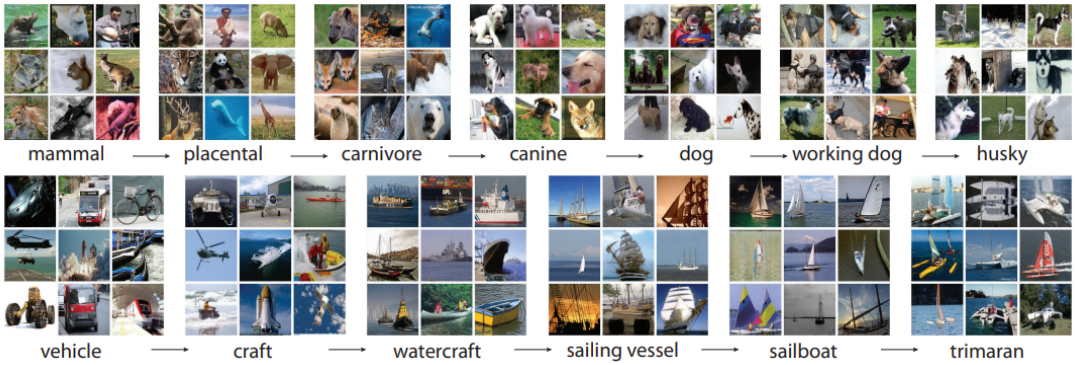

Figure 2.1: Example Images from the ImageNet dataset.

2.1.1 ImageNet

The development of modern deep learning for image analysis is closely linked to the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). Introduced in 2009, ImageNet is a large-scale, annotated image database containing over 14 million images across thousands of object categories. The ILSVRC provided a benchmark for evaluating classification and detection methods on a massive and diverse dataset, driving rapid advancements in computer vision.

A major breakthrough came in 2012 with AlexNet, a deep convolutional neural network (CNN) that significantly outperformed previous approaches on the ImageNet classification task. This success demonstrated the power of deep architectures trained on large datasets and marked the beginning of widespread adoption of CNNs for a variety of vision tasks. Following AlexNet, CNNs evolved through multiple directions to improve performance. Increasing network depth and width allowed models to capture more complex patterns and richer feature hierarchies. Enhanced connectivity patterns, such as skip connections and residual blocks, improved gradient flow and facilitated training of very deep networks. In addition, more sophisticated forms of convolution, including dilated and grouped convolutions, allowed for multi-scale feature extraction and more efficient parameter usage.

These architectural advances enabled CNNs to become the backbone for a wide range of vision applications, from image classification and object detection to segmentation and beyond. The combination of hierarchical feature extraction, nonlinear activations, and spatial aggregation operations continues to make CNNs highly effective and generalizable across diverse domains, including medical and biological imaging, where large annotated datasets are often limited. While the developement of CNNs has benefitted a lot from the ImageNet dataset, also novel architectures such as Transformer based models, which be later introduced, make use of the diverse ImageNet dataset to be pretrained for many downstream tasks.

2.1.2 U-Net Architecture

A widely adopted convolutional neural network architecture for biomedical image segmentation is the *U-Net*, introduced by Ronneberger et al. in 2015. The U-Net was specifically designed for tasks where precise localization is essential, such as cell segmentation in microscopy images. Its name derives from its characteristic U-shaped architecture, consisting of a contracting path (encoder) and an expansive path (decoder).

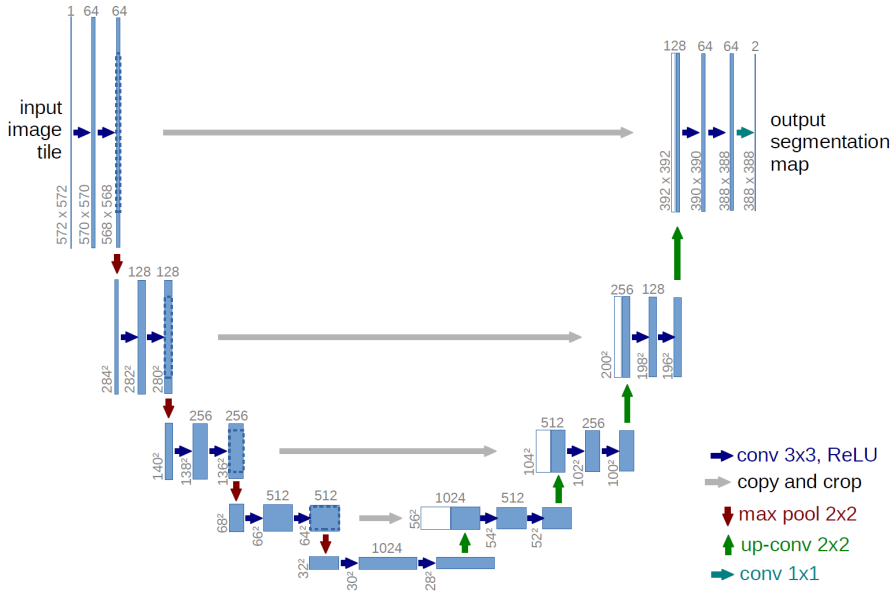

Figure 2.2: Schematic illustration of the U-Net architecture for image segmentation. The network consists of a contracting path (left side) to capture context via successive convolutional (3×3, ReLU) and max pooling (2×2) operations, and an expansive path (right side) that enables precise localization through up-convolutions (2×2) and concatenation with high-resolution features from the contracting path. The final segmentation map is generated using a 1×1 convolution. Feature map sizes and channel numbers are indicated for each layer.

The **contracting path** progressively reduces the spatial dimensions of the input while increasing the number of feature channels. This is achieved through repeated applications of convolutional layers followed by non-linear activations and downsampling operations such as max pooling. The purpose of this path is to capture the contextual and semantic information of the input image at various scales.

The **expansive path**, on the other hand, restores the spatial resolution through a sequence of upsampling operations (e.g., transposed convolutions) and concatenations with corresponding feature maps from the contracting path via *skip connections*. These skip connections provide fine-grained spatial details from earlier layers, improving the precision of the segmentation boundaries.

Formally, the segmentation prediction can be seen as a pixel-wise classification problem, where the final layer of the U-Net outputs a probability map indicat-

ing the likelihood of each pixel belonging to a particular class (e.g., foreground or background in binary cell segmentation). The combination of hierarchical feature extraction and precise localization makes the U-Net particularly effective for segmenting structures of varying size and shape, as commonly encountered in biomedical images.

U-Net and its derivatives have since become the backbone of many state-of-the-art cell segmentation pipelines in both fluorescence and label-free imaging modalities.

2.2 Image Segmentation

In this section, we will introduce the main types of segmentation in image data and will further explain the techniques that help us achieve an accurate segmentation result. We will learn how modified learning objectives will lead to better results and

2.2.1 Semantic and Instance Segmentation

The vanilla U-Net assigns pixels to distinct, predifined classes. Usually, at the last layer of the decoder path, the original image resolution is restored (neglecting the missing pixels from unpadded convolutions). At this point, one would map the channel feature map depth (e.g. 64 for vanilla U-Net) via a 1x1 convolution to the amount of classes each pixel can be assigned to. Thereby one ends up having a probability distribution of predefined classes over all pixels. Logits are mapped to probabilities with Soft-Max and with Non-Maximum-Supression (NMS), each pixel is assigned to a class, so in the end the resolution corresponds to $\mathbf{H} \times \mathbf{W}$ with each pixel integer value corresponding to a certain class. These classes can represent all kinds of types, such as binarary values like foreground/background (cell or no cell), or for modern use cases, surrounding objects on a street. This type of segmentation is refered to as *semantic segmentation*. Oftentimes some other sophisticated post processing methods like binary closing or removal objects with size below a certain threshold are applied.

Figure 2.3: Semantic Segmentation in the Traffic Environment using the DeepLabv3+ Model

For many applications, it is desirable to use a framework that not only separates image regions into semantic classes, but also assigns each object instance within

the same class a unique identifier. This task is known as instance segmentation.

A common strategy is to first perform semantic segmentation and then separate individual instances in a post-processing step (a bottom-up approach). However, such methods are often prone to errors, for example when adjacent objects of the same class touch or overlap, leading to under-segmentation (merging of multiple cells) or over-segmentation (splitting a single cell into several fragments).

Alternatively, instance segmentation can be formulated as detecting object proposals via bounding boxes, which are then refined into masks, as done in methods like Mask R-CNN. While effective in many domains, this top-down approach becomes inaccurate in dense cellular images where bounding boxes frequently overlap, making NMS unreliable.

For cellular instance segmentation, it is therefore common to edit the learning objective itself so that the model directly predicts instance-specific features (e.g., object centers, flow fields, or boundary cues). These features can then be leveraged during post-processing to produce accurate boundaries and clearly separated instances. StarDist for example, directly predicts a set of ray distances to the object boundary for every pixel. Fusing this feature with a per pixel cell probability map, the set of proposed polygon shaped objects can be accurately post processed with NMS to predict even spatially dense distributed cells.

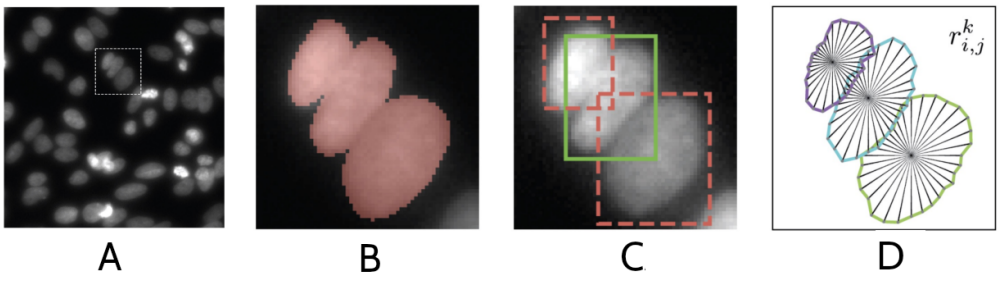

Figure 2.4: Different instance segmentation methods, A: original image, B: Bottom-up approach, C: Top-down approach, D: Direct instance segmentation, where the model predicts instance-specific features that are used to delineate individual objects.

2.2.2 Cellpose

While StarDist aims to predict distance maps to the object boundary, Cellpose introduced a novel approach by predicting spatial gradients that point toward the center of each cell. Classical approaches such as the watershed algorithm rely on intensity basins in the image to define object regions, but in many microscopy images, cells exhibit multiple intensity basins due to inhomogeneous labeling or nuclear exclusion of fluorescent markers. To overcome this limitation, Cellpose constructs an intermediate representation of each cell as a single smooth topological basin.

These topological maps are generated from manually annotated masks using a simulated diffusion process, which produces horizontal and vertical gradients point-

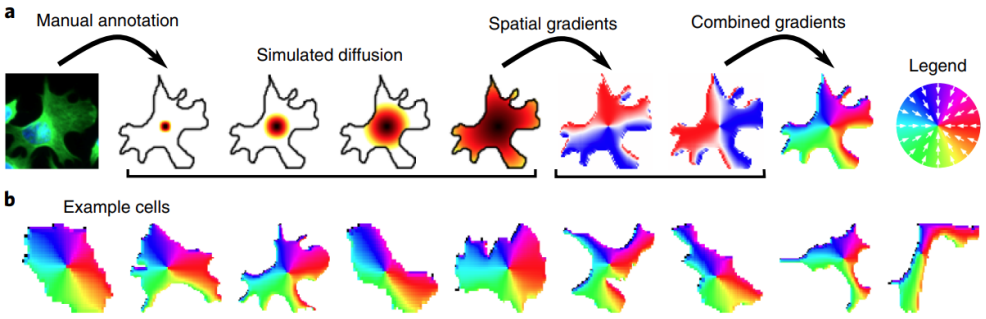

Figure 2.5: t-SNE embeddings of image styles from cellpose pretraining data.

ing toward the center of each cell. The network is then trained to predict 3 channels: 2 channels for xy gradients together with a channel per pixel cell probability. At post processing, the predicted gradients define a vector field through which pixels are iteratively routed toward their corresponding cell centers. By grouping pixels that converge to the same point, the network recovers individual cell instances with accurate shapes, which are further refined using the predicted binary mask to remove pixels outside of cells.

Similarly to StarDist, the first launch of cellpose transforms manually annotated masks to the learning objective shape and uses a U-Net which maps the features to the learning objective channels with conv2d layers. Global average pooling of the lowest-resolution features is used to extract an image 'style' vector, which is incorporated throughout the upsampling stages to account for differences in imaging modalities and staining patterns. Insterestingly, plotting the style vectors with t-SNE shows clear distinctions in cell modalities, as k-means clustering groups similar cell types together.

In practice, it turns out that this feature representation performs best across many benchmarks and is considered state-of-the-art for cell segmentation. Although for specialized datasets, StarDist performs very similarly well, Cellpose's learning objective generalizes better across diverse cell types and imaging modalities. The model itself is considered a foundation model for cell segmentation, due to its very heavy pretraining. Therefore, it performs very well on unseen data, even without any finetuning.

2.2.3 Loss Functions and Metrics

The loss function, denoted by $\mathcal{L}$, plays a central role in training neural networks by providing a quantitative measure of the error between the predicted output and the true target values. During training, the network's parameters are updated to minimize this loss, improving its predictive performance on the given task.

While loss functions guide the optimization process during training, *evaluation metrics* serve to assess the quality of the model's predictions, typically on a separate validation or test set. Metrics quantify different aspects of performance that are relevant to the problem at hand and are especially important in applications such

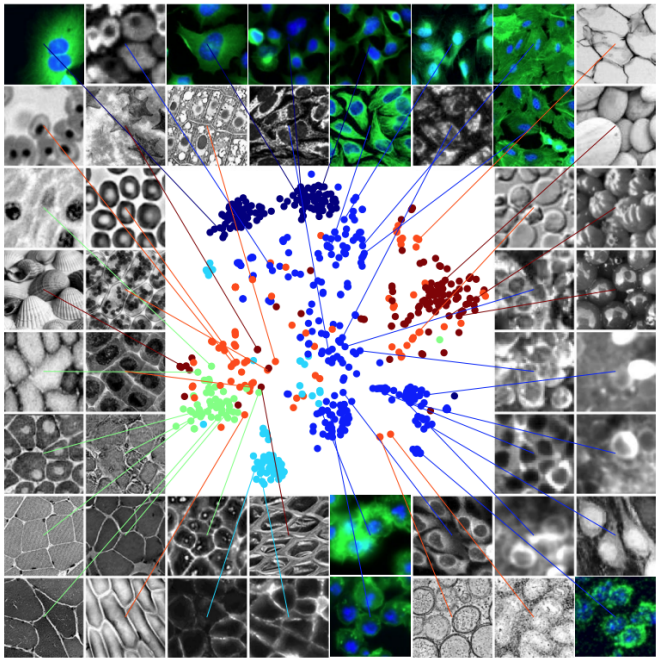

Figure 2.6: t-SNE embedding of image styles from cellpose pretraining data.

as cell segmentation, where a model's utility is not solely determined by its training loss.

In cell segmentation tasks, common evaluation metrics include the *Intersection over Union (IoU)*, *Dice coefficient*, and *Average Precision (AP)*, all of which compare the predicted segmentation masks to the ground truth annotations.

The **Intersection over Union (IoU)**, also known as the Jaccard index, measures the overlap between the predicted mask P and the ground truth mask G relative to their union:

$$\text{IoU}(P, G) = \frac{|P \cap G|}{|P \cup G|}$$

A higher IoU indicates a better agreement between prediction and ground truth, with a value of 1 representing perfect overlap.

The **Dice coefficient**, closely related to IoU, is another overlap-based metric commonly used in segmentation tasks. It is defined as:

$$\text{Dice}(P, G) = \frac{2|P \cap G|}{|P| + |G|}$$

Like IoU, the Dice coefficient ranges from 0 to 1, where a value of 1 indicates perfect correspondence between the predicted and ground truth masks. Compared to IoU, the Dice score tends to be slightly more forgiving in cases of small object segmentation.

Another widely used metric in instance segmentation is the **Average Precision (AP)**. It summarizes the precision-recall curve into a single value by integrating over different confidence thresholds. Given precision $\text{Prec}(t)$ and recall $\text{Rec}(t)$ at

threshold t, the Average Precision is computed as:

$$\mathrm{AP} = \int_0^1 \mathrm{Prec}(\mathrm{Rec})\, d\mathrm{Rec}$$

In practice, this integral is often approximated using discrete recall points. In cell segmentation benchmarks, AP is typically reported at different IoU thresholds (e.g., AP@0.5, AP@0.75) to capture performance at varying levels of prediction-stringency.

Together, the choice of loss function for optimization and the metrics for evaluation provide complementary perspectives: the former drives model improvement during training, while the latter ensures the resulting model performs reliably and meaningfully on the specific segmentation task.

2.3 Foundation Models

In recent years, the concept of foundation models has gained significant attention in deep learning. These models are large-scale, pre-trained networks trained on vast and diverse datasets, so that they can capture broad and general-purpose knowledge about language, images, or other data modalities. Unlike traditional models that are built and trained for a specific task, foundation models are designed to be adaptable, allowing them to be fine-tuned for a wide range of downstream applications with little additional data or training.

A key characteristic of foundation models is their ability to generalize across domains. For example, a language foundation model like GPT or an image foundation model like CLIP can perform well on tasks it was not explicitly trained for, such as summarizing text, answering questions, or classifying medical images, after minimal adaptation. Their ability to perform well on unseen modalities (often referred to as zero-shot inference) makes them particularly valuable in biomedical imaging, where annotated data is often limited because its (labor-) expensive to obtain.

However, foundation models also bring new challenges, as their large size demands significant computational resources for training and deployment. CellposeSAM for example, has an order of magnitude more parameters compared to its CNN based predecessor Cellpose-1 and can practically not be run without a GPU, as they print as a warning in their own code. Additionally, vision foundation models are almost exclusivly 2D, making 3D inference challenging. Despite these drawbacks, foundation models mark a major step towards more flexible and scalable models that offer great opportunities to fit in various downstream tasks.

2.4 Transformer

Transformer architectures have recently gained significant attention in the field of medical image analysis, demonstrating strong performance in tasks such as image classification, detection, and segmentation. Their ability to capture long-range dependencies through self-attention mechanisms makes them particularly well-suited

for learning segmentation features, where spatial relationships extend beyond local neighborhoods. In this section, we first introduce the attention mechanism as the core building block underlying transformer-based models. We then present the transformer architecture itself, followed by its adaptation to image data through the Vision Transformer (ViT). Finally, we discuss further developments and extensions tailored for image segmentation tasks, such as MA-Net and SegFormer.

2.4.1 The Attention Mechanism

The *attention mechanism* is a key concept in modern deep learning models, designed to dynamically focus on the most relevant parts of the input data when making predictions. Unlike traditional convolutional operations, which process local neighborhoods with fixed receptive fields, attention allows the model to consider relationships between all elements in the input, regardless of their spatial or sequential distance.

Formally, given an input sequence of n feature vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$, these are linearly projected into three different representations:

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V, \tag{2.1}$$

where $Q, K, V \in \mathbb{R}^{n \times d_k}$, and $W^Q, W^K, W^V \in \mathbb{R}^{d \times d_k}$ are learnable projection matrices.

The attention scores are obtained by computing scaled dot-products between queries and keys:

$$A = \frac{QK^\top}{\sqrt{d_k}}, \quad A \in \mathbb{R}^{n \times n}, \tag{2.2}$$

where each element A_{ij} measures the similarity between the i-th query and the j-th key. Scaling with with the embedding dimension d_k is important for numerical stability. For instance, training with torch autocast enabled, can cause float16 values to overflow during the matrix multiplication between the queries and keys. To obtain normalized attention weights, a row-wise softmax is applied:

$$\alpha_{ij} = \frac{\exp(A_{ij})}{\sum_{j'=1}^{n} \exp(A_{ij'})}. \tag{2.3}$$

The final output of the attention mechanism is then a weighted sum of the value vectors:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V. \tag{2.4}$$

In the *self-attention* setting, Q, K, V are all derived from the same input X, enabling the model to capture long-range dependencies within a single data sample. This formulation underpins the Transformer architecture and its extensions for natural language processing, computer vision, and beyond.

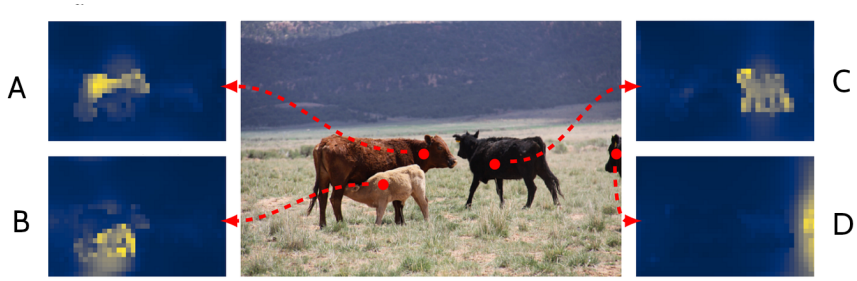

Figure 2.7: Encoder self-attention for a set of reference points. The encoder is able to separate individual instances (A-D).

2.4.2 Transformer

Building on the attention mechanism, the transformer architecture was originally introduced for sequence modeling tasks in natural language processing, but has since proven highly versatile across domains. Its core innovation lies in replacing recurrent and convolutional operations with a stack of self-attention and feedforward layers, enabling efficient modeling of long-range dependencies.

A standard transformer consists of an encoder–decoder structure. For vision-related tasks such as image analysis and segmentation, typically only the encoder part is used, since the decoder was designed for sequence-to-sequence tasks like language translation. Each encoder layer is composed of two main components: a multi-head self-attention mechanism, which allows the model to jointly attend to information from different representation subspaces, and a position-wise feed-forward network, which applies non-linear transformations independently to each position in the input.

To preserve information about the order or spatial position of elements, positional encodings are added to the input embeddings before they enter the encoder. Stacking multiple encoder layers enables the model to capture increasingly abstract and global relationships within the data. This ability to integrate long-range dependencies is particularly valuable in vision applications, including medical image analysis and cell segmentation, where spatial context plays a crucial role.

2.4.3 Vision Transformer

The Vision Transformer (ViT) extends this transformer architecture from sequential data to image analysis by treating images as sequences of patches. Instead of processing entire images using convolutional layers, ViT divides an image into fixed-size, non-overlapping patches, flattens them, and projects them into a lower-dimensional embedding space. These patch embeddings are then combined with positional encodings and fed into a standard transformer encoder.

ViTs have demonstrated strong performance in various computer vision tasks by leveraging the self-attention mechanism's ability to model global dependencies between image regions, without the locality constraints of convolutional operations.

Although the vanilla ViT has shown impressive results on ImageNet, there are

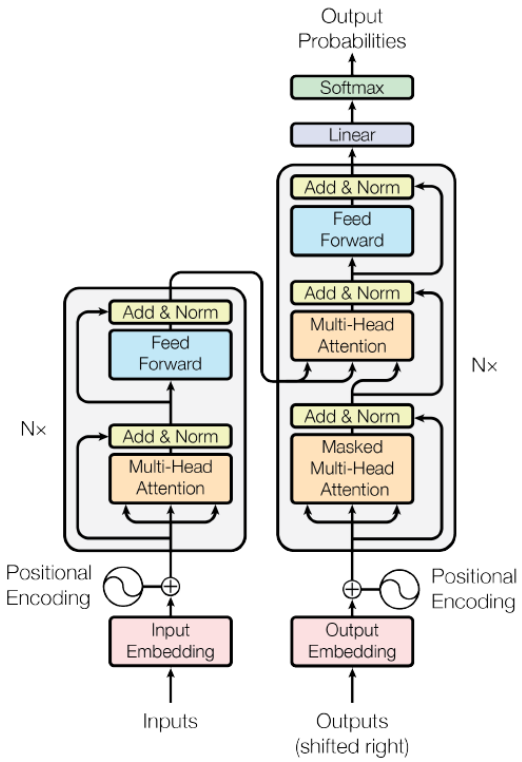

Figure 2.8: Transformer architecture. For vision tasks, usually the left part (encoder) is taken.

some theoretical concerns about the way vanilla ViT encode its features. While in CNN encoders, high- and low-res features are obtained in a corresponding layer depth, no such built-in multi-scale feature extraction is existent in vanilla ViTs. In addition, the non-overlapping, fixed-size tokenization of ViTs leads to high computational costs, as the complexity is $\mathcal{O}(N^2 \cdot D)$, scaling quadratically with the image size N times the embedding dimension D.

2.4.4 SegFormer

To tackle the above mentioned issues of the vanilla ViT, xy et al (cite) proposed the SegFormer architecture. It is composed of a hierarchical Transformer-based encoder and a lightweight decoder, designed specifically for efficient and accurate semantic segmentation. The encoder consists of multiple Mix Transformer (MiT) blocks, each incorporating three main components: efficient self-attention, Mix Feed-Forward Networks (Mix-FFN), and Overlapping Patch Mappings.

To reduce the computational complexity of the standard self-attention mechanism used in ViTs, they introduce an efficient self-attention variant: While the conventional self-attention has a complexity of $\mathcal{O}(pN^2)$, this is reduced to $\mathcal{O}(pN^2/R)$ by modifying the key tensor K as $K = \text{Linear}(\text{reshape}(K'))$, where $K' \in \mathbb{R}^{N/R \times C \cdot R}$, effectively grouping tokens into spatial regions. This significantly improves scalability on large input images. In the Mix-FFN, they replace positional encodings

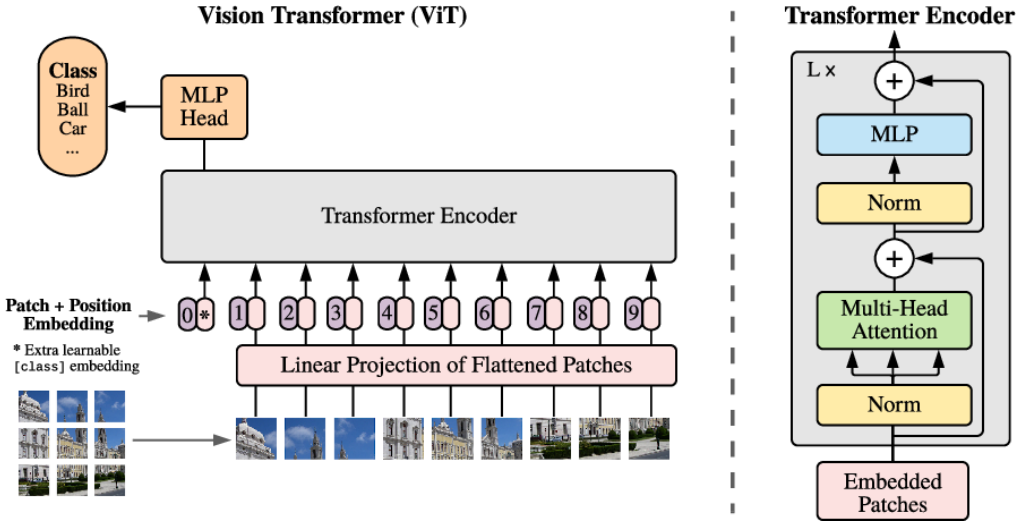

Figure 2.9: Vision transformer as presented in

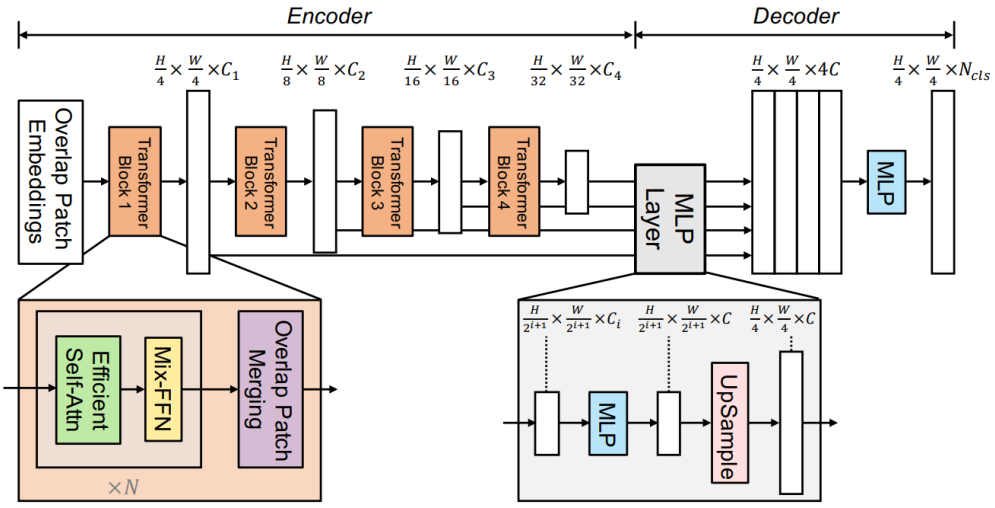

Figure 2.10: SegFormer architecture as presented in

with a 3×3 depthwise convolution, which not only captures local spatial dependencies but also introduces an implicit positional bias, enabling the model to better encode structure. Overlapping Patch Mapping is used in each stage to partition the image into patches with shared pixels, preserving local continuity and reducing edge artifacts between patches. This patch merging is the main component of the downsampling part of the encoder. It thereby outputs multi-scale features from different stages, which are then processed by a lightweight decoder. The decoder projects all feature maps to a unified embedding space, upsamples them to a common resolution, and fuses them via concatenation. A linear prediction layer then generates per-pixel class logits. This minimalist decoder design ensures fast inference while maintaining segmentation accuracy. By combining hierarchical feature extraction, efficient attention, and a streamlined decoder, SegFormer achieves

state-of-the-art performance across several benchmarks while remaining computationally lightweight. A diagram of the network architecture is depicted in figure 2.10.

2.4.5 MA-Net: Multi-Scale Attention Network

The Multi-scale Attention Network (MA-Net), proposed by Fan et al., is a segmentation architecture designed to improve accuracy in medical image analysis, particularly for liver and tumor segmentation. It builds upon the encoder-decoder structure of U-Net, but enhances it with attention-based modules that enable the network to better capture long-range dependencies and multi-scale contextual information. MA-Net has been shown to outperform state-of-the-art models on the MICCAI 2017 LiTS Challenge.

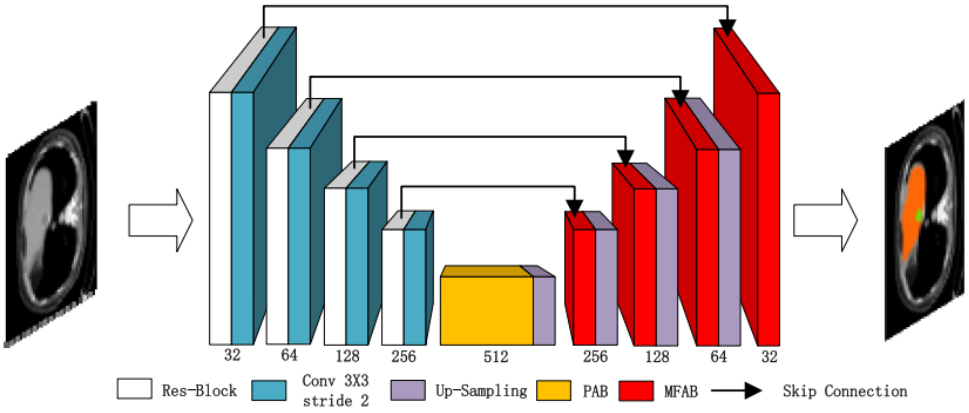

Figure 2.11: Architecture of the MA-Net.

The encoder extracts hierarchical features through standard convolutional layers, while the attention-enhanced decoder refines these features using two sufisticated attention modules before producing the final segmentation output. Skip connections between encoder and decoder stages are preserved, allowing the network to retain important spatial information during upsampling. The full high level architecture is displayed in 2.11.

The key innovation in MA-Net lies in its decoder, which introduces two specialized attention modules: the *Position-wise Attention Block* (PAB) and the *Multi-scale Fusion Attention Block* (MFAB).

The PAB is designed to model feature inter-dependencies in a global spatial context by computing position-aware attention maps, which helps the network focus on relevant regions within the entire image.

A low-level view of the PAB block is given in 2.12. The local feature map $I \in \mathbb{R}^{H\times W\times 256}$ (in this net, the lowest resolution feature of the encoder) is first transformed by a 3×3 convolution to produce $I_0 \in \mathbb{R}^{H\times W\times 512}$. Three 1×1 convolution layers generate feature maps $A \in \mathbb{R}^{H\times W\times 64}$, $B \in \mathbb{R}^{H\times W\times 64}$, and $C \in \mathbb{R}^{H\times W\times 512}$.

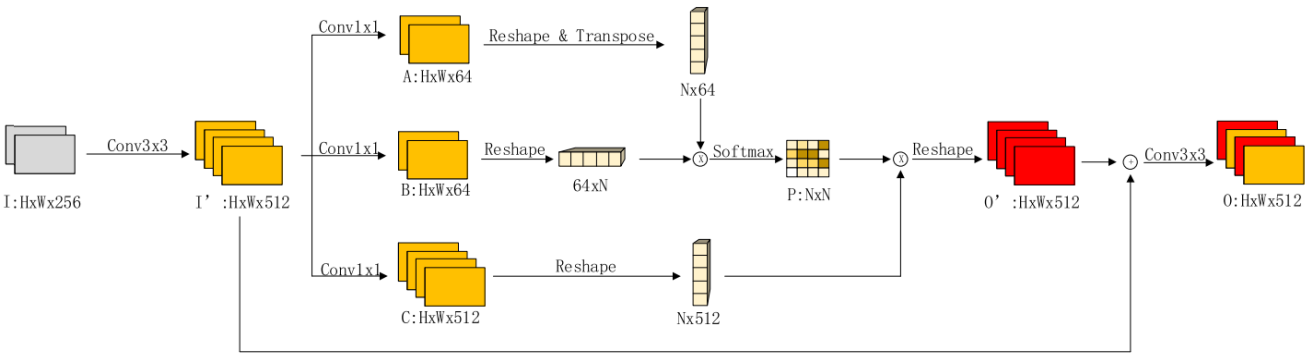

Figure 2.12: Low-level view of the PAB block.

This basically mimics the learnable projection matrices (as defined in eq. 2.1) in transformers, turning input embeddings into pseudo QKV representations. Reshaping A and B into $A \in \mathbb{R}^{N\times 64}$ and $B \in \mathbb{R}^{64\times N}$, where $N = H \cdot W$ is the number of pixels, a matrix multiplication followed by softmax yields the spatial attention map $P \in \mathbb{R}^{N\times N}$:

$$p_{ji} = \frac{\exp(A_i B_j)}{\sum_{i=1}^{N} \exp(A_i B_j)}.$$

This map encodes the influence of the i-th position on the j-th position. The feature map C (pseudo Key matrix) is reshaped to $C \in \mathbb{R}^{N\times 512}$, and multiplied with P to obtain a context-aware representation:

$$O'_j = \sum_{i=1}^{N} P_{ji} C_i.$$

Finally, the result is reshaped back to $O'_0 \in \mathbb{R}^{H\times W\times 512}$, added element-wise to I_0, and passed through a 3×3 convolution to produce the final output $O \in \mathbb{R}^{H\times W\times 512}$:

$$O_j = \alpha O'_j + I_{0,j},$$

where α is a learnable scalar initially set to 0. Each output position is thus a weighted sum of features across all positions combined with the original feature, providing a global contextual view that improves segmentation consistency. Note, that the matrix multiplication of A and B is not normed as in the origin self-attention mechanism (see eq. 2.2). While this might reduce computational complexity, it makes the forward pass prone to overflow issues due to the lack of numerical stability from the norm operation.

The MFAB facilitates semantic feature fusion across multiple scales by combining feature maps of varying resolutions. This enables the decoder to integrate both fine-grained details and high-level semantic cues, which the authors claim to obtain more precise boundary delineation and region consistency. A low-level view of the MFAB block is given in 2.13.

The high-level feature map $X_H^* \in \mathbb{R}^{H\times W\times C_0}$ is first transformed by a 3×3 convolution followed by a 1×1 convolution to obtain $X_H \in \mathbb{R}^{H\times W\times C}$, aligning it

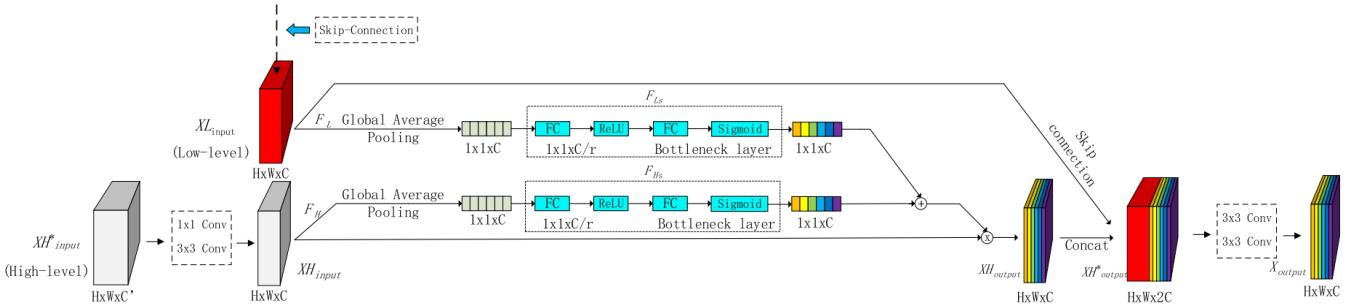

Figure 2.13: Multi-scale Fusion Attention Block (MFAB) structure.

with the low-level skip feature $X_L \in \mathbb{R}^{H \times W \times C}$. Global average pooling is applied to both high- and low-level features to produce channel descriptors $s_c^{(H)}$ and $s_c^{(L)}$:

$$s_c^{(H)} = \frac{1}{HW} \sum_{i=1}^{H} \sum_{j=1}^{W} X_H^{(c)}(i,j), \quad s_c^{(L)} = \frac{1}{HW} \sum_{i=1}^{H} \sum_{j=1}^{W} X_L^{(c)}(i,j).$$

These descriptors are passed through a bottleneck network with two fully connected layers (implemented as 1×1 convolutions in practice) with reduction ratio r and activations ReLU (δ) and sigmoid (σ) to obtain channel attention weights:

$$z^{(H)} = \sigma\big(P_1\, \delta(P_2\, s^{(H)})\big), \quad z^{(L)} = \sigma\big(P_1\, \delta(P_2\, s^{(L)})\big).$$

The weights from high- and low-level features are combined by addition:

$$z = z^{(H)} + z^{(L)},$$

and the high-level feature map is rescaled channel-wise:

$$\tilde{X}_H^{(c)} = z_c \cdot X_H^{(c)}.$$

The rescaled high-level features are concatenated with the low-level skip features:

$$X_{\text{concat}} = [\tilde{X}_H, X_L],$$

and refined through two successive 3×3 convolutions to produce the final output:

$$X_{\text{out}} = \text{conv2d}_{3\times3}(\text{conv2d}_{3\times3}(X_{\text{concat}})).$$

Considering they basically reimplemented the concepts of Sequeeze and Exitement (SE) layers along with some concatinations and additions from two resolutions, the term attention may be a little misleading here. Nevertheless this way of scaling feature maps with fused SE results has been proven to perform really well.

2.5 Masked Autoencoders

Masked Autoencoders (MAE) are a self-supervised learning framework designed to pretrain large vision models efficiently. The key idea is to randomly mask a high

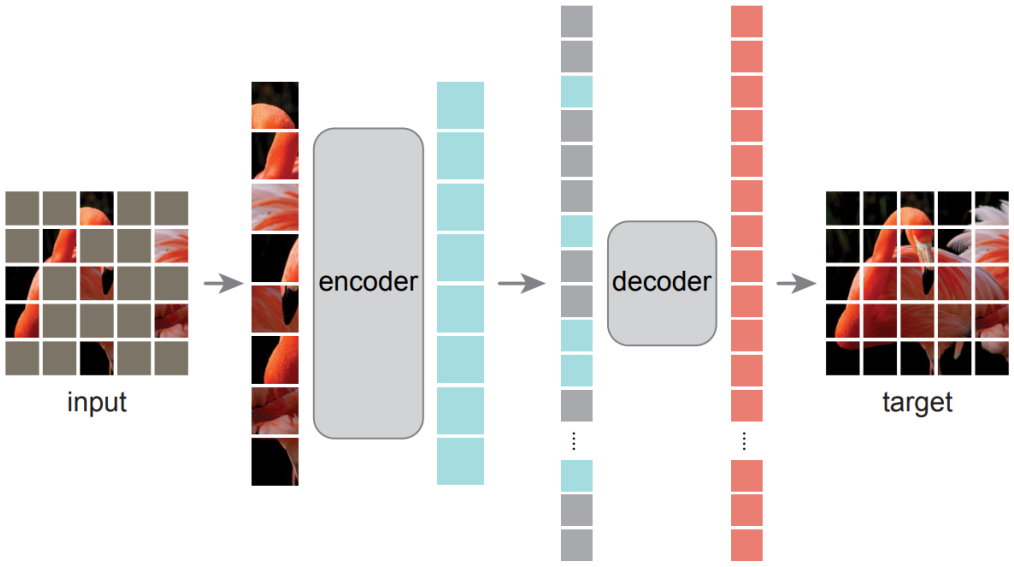

Figure 2.14: MAE architecture as presented in [1].

proportion of image patches and to train the network to reconstruct the missing pixels from the remaining visible content. This forces the model to learn meaningful representations of visual structures, as it must capture global context in order to accurately fill in the missing regions.

The architecture of an MAE is based on an asymmetric encoder–decoder design. The encoder processes only the subset of visible patches without introducing explicit mask tokens. This makes the encoder highly efficient, as it avoids computation over large portions of the input image. The decoder, by contrast, is lightweight and is responsible for reconstructing the original image from the latent representation of the visible patches in combination with the mask tokens that indicate the missing regions. In this way, most of the modeling capacity is allocated to the encoder, while the decoder only serves as a reconstruction head during pretraining.

This design enables masked autoencoders to train large models efficiently and effectively. Compared to traditional supervised pretraining approaches, MAEs achieve significant gains in training speed, with reported improvements of up to a threefold acceleration, while also improving downstream accuracy. After pretraining, the decoder is discarded, and the encoder is applied to complete, unmasked images. The pretrained encoder then serves as a powerful feature extractor for a variety of recognition and segmentation tasks, often providing strong performance with limited additional task-specific training.

2.6 Segment Anything

The Segment Anything Model (SAM) represents a new class of foundation models for image segmentation. Unlike conventional approaches that are trained for a

specific downstream task, SAM is designed as a promptable segmentation model: given an image and a prompt, it returns a valid segmentation mask. Prompts specify what to segment and can be provided in different forms, including spatial inputs such as points, bounding boxes or scribbles, or textual descriptions such as "a cat with black ears." A central requirement is that the model always outputs a reasonable mask, even in cases of ambiguity, by producing at least one plausible segmentation result.

The training objective of SAM is a promptable segmentation task, which allows the model to generalize across a wide range of domains. Downstream applications can be reformulated as prompt engineering, which makes it possible to reuse the same foundation model for diverse segmentation problems without additional training. The design of SAM therefore emphasizes three objectives: the ability to support heterogeneous prompts, the ability to deliver results quickly enough for interactive use, and explicit handling of ambiguous prompts.

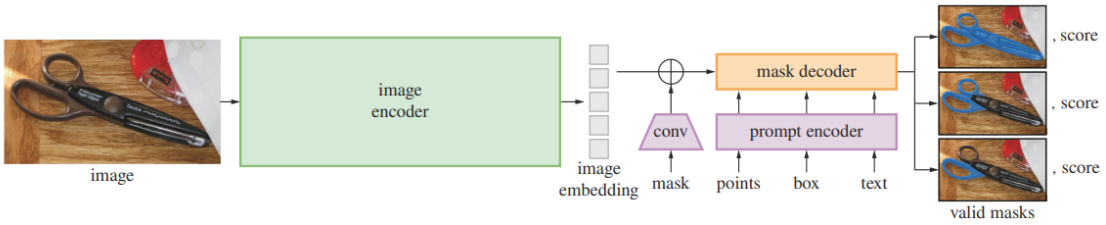

Figure 2.15: MAE architecture as presented in [1].

The architecture of SAM is composed of three decoupled components: an image encoder, a prompt encoder, and a mask decoder. The image encoder is actually just a vanilla ViT with small adaptations, such as the choice of global attention computation on only a few layers and different embedding dimensions for specific size versions (small, medium, large, huge), each pretrained in a masked autoencoder style. The image encoder processes an input image once to compute a dense embedding, which can then be reused for multiple prompts. This amortization of computational cost is crucial for interactive applications. The prompt encoder embeds different forms of prompts, whether spatial or textual, into a common representation. The mask decoder then fuses image and prompt embeddings to predict segmentation masks. It employs a modified Transformer decoder block with both self-attention over prompts and bidirectional cross-attention between prompt and image embeddings. After two decoding blocks, the image embedding is upsampled and an MLP maps the output tokens to a dynamic linear classifier, which produces per-pixel mask probabilities.

To handle ambiguity, the decoder generates multiple candidate masks for a single prompt. During training, only the mask with the lowest loss, corresponding to the best overlap with the ground truth, is used for backpropagation. An additional branch predicts the intersection-over-union score for each mask, which enables the ranking of the different outputs at inference time. The loss functions used in training combine focal loss and Dice loss with a strong weighting toward focal loss, while

the IoU prediction branch is supervised with mean squared error.

A key factor in SAM's ability to generalize is the unprecedented scale of its training data. To create such a dataset, the authors developed a data engine, a model-in-the-loop annotation strategy with three progressive stages. In the assisted-manual stage, SAM supported human annotators in producing masks, similar to classical interactive segmentation. In the semi-automatic stage, the model was able to generate masks for a subset of objects automatically, while annotators concentrated on refining difficult cases. Finally, in the fully automatic stage, SAM was prompted with a grid of points, producing on average around 100 high-quality masks per image. This pipeline resulted in the SA-1B dataset, which contains more than one billion masks from eleven million licensed and privacy-preserving images. Compared to existing segmentation datasets, this is larger by two orders of magnitude and provides the foundation for SAM's strong generalization abilities.

2.6.1 Segment Anything 2

Following the release of the original Segment Anything Model, a second version (SAM 2) was introduced to further improve versatility and applicability. While SAM 1 demonstrated the feasibility of promptable segmentation at scale, SAM 2 extends these capabilities with a stronger focus on efficiency, scalability, and temporal reasoning.

SAM 2 preserves the core idea of promptable segmentation with a decoupled architecture of image encoder, prompt encoder, and mask decoder, but introduces several refinements. The image encoder is optimized for faster inference, making the model more practical for large-scale or interactive applications. Moreover, SAM 2 improves ambiguity handling by refining the multi-mask output strategy, leading to more consistent predictions in complex scenes.

A key extension of SAM 2 lies in its support for *video and spatio-temporal segmentation*. By incorporating mechanisms to reuse information across frames, SAM 2 enables object tracking and segmentation over time, which broadens its applicability beyond static image analysis. At the same time, the training data has been further scaled, with an expanded dataset that improves robustness across domains and object categories.

2.6.2 Hiera

Hiera is a hierarchical vision transformer architecture designed to combine the strengths of transformer-based models with the efficiency of convolutional networks. Unlike standard Vision Transformers, which operate on a flat sequence of image patches, Hiera introduces a hierarchical structure that progressively reduces spatial resolution while expanding the feature dimension. This design mirrors the multiscale feature extraction used in convolutional neural networks and is particularly well-suited for dense prediction tasks such as segmentation.

The model is constructed as a stack of transformer blocks organized into stages. Each stage processes tokens at a given spatial resolution before applying patch

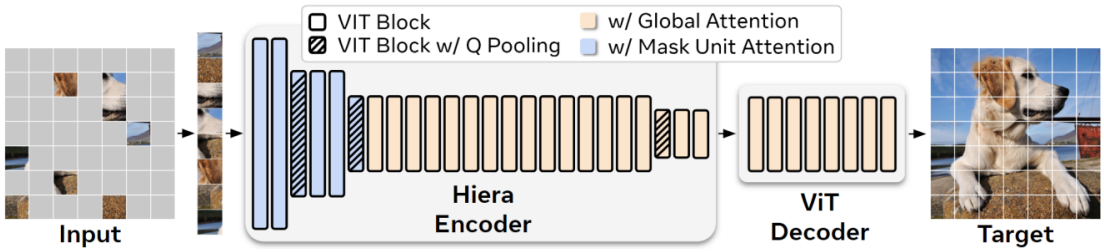

Figure 2.16: MAE architecture as presented in [1].

merging to reduce resolution and increase the channel dimension. This hierarchical representation enables the network to capture both fine-grained local patterns and coarse global context. In addition, Hiera incorporates efficient attention mechanisms that scale sub-quadratically with image size, making it computationally more efficient than standard ViT backbones while retaining strong modeling capacity.

A key advantage of Hiera is its flexibility as a general-purpose backbone. It can be pretrained on large-scale image classification tasks and subsequently fine-tuned for a wide range of downstream applications, including semantic segmentation, instance segmentation, and object detection. Empirical studies have shown that Hiera achieves competitive accuracy compared to state-of-the-art transformer backbones, while offering improved efficiency and scalability across different model sizes.

2.7 Mediar

Mediar is yet another encoder decoder style architecture, designed specifically for cell segmentation. It was developed in the context of the 2022 NeurIPS Cell Segmentation Challenge, where it achieved top performance across a variety of imaging modalities. Their approach is based on the Cellpose learning objective, but instead of using a U-Net style CNN backbone, they adopt the encoder part of SegFormer and combine it with the decoder blocks of MA-Net. This still ensures a hierarchical feature integration from the SegFormer multi-resolution features, to the MFAB blocks of the MA-Net via skip connections. They use Mish in the encoder and decoder for better generalization. Although prior work use a single segmentation head, meaning they map the features to a single output tensor with multiple channels for each output (cell probability, x- and y- flows), the authors used two seperate heads to map features to the cell probability and flows, respectively. They argue that semantic prediction for the objects and mse regression for the spatial gradient field interfers with each other, while using the same feature space [**?**]. Therefore, they use two seperate conv2d layers with batch norm to map the features to the respective outputs.

They pretrained MEDIAR on a large set of public datasets, namely Omnipose, Cellpose, Livecell and DataScienceBowl-2018, which makes a combined total of 7,242 labeled images. For the Challenge Dataset, they came up with a sophisticated way of training two seperate models (phases) with each different subsets of challenge data in the pretraining set, to perform ensembling prediction at test

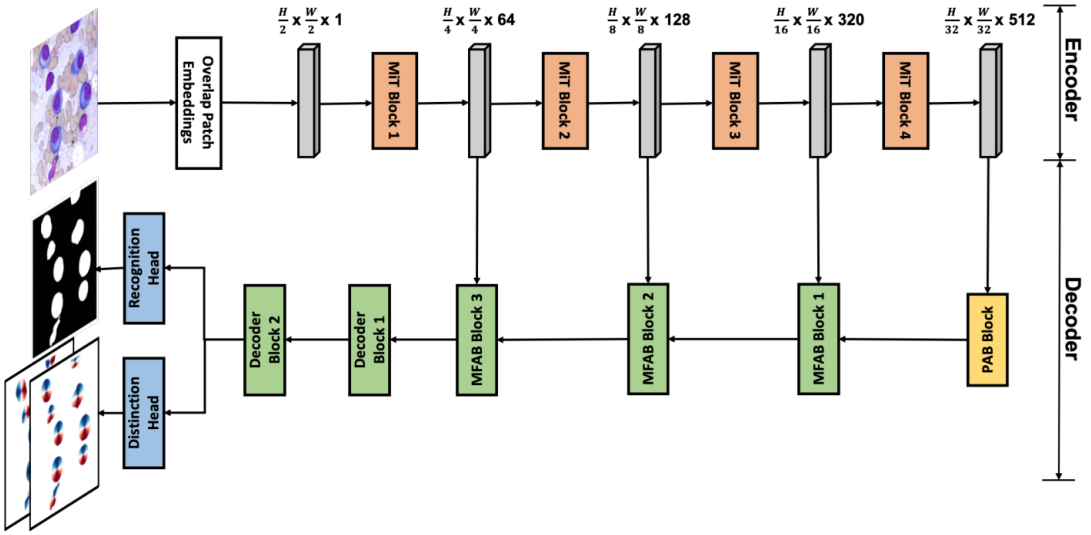

Figure 2.17: Encoder embeddings of the train set clustered with k-mean and intuition of amlplified sampling [1].

time. In the first phase, the net is initialized with ImageNet-1k pretrained encoder weights, and further trained on the public datasets for 80 epochs to establish strong general-purpose cell representations. In the second phase, this model was further trained for 60 epochs on the combined set of public datasets and the challenge training data, allowing the network to bridge between general cell morphologies and the specific modalities of the target domain. Fine-tuning was then conducted separately on the models from both phases. The phase 1 model was fine-tuned for 200 epochs on the challenge training set, specializing it to modalities present only in the target data. The phase 2 model was fine-tuned for 25 epochs, leveraging its broader knowledge of both public and challenge domains. Interestingly, the two models exhibited complementary behavior: the phase 1 fine-tuned model excelled at adapting to unseen modalities in the challenge, whereas the phase 2 fine-tuned model maintained stronger generalization across shared modalities. To further enhance robustness, they introduced a relabeling step at phase 2 fine-tuning, correcting images where the pretrained and phase 1 fine-tuned models produced misaligned predictions, thereby mitigating the impact of noisy annotations. Similarly to the style embeddings from Cellpose, they examine modality dependent embeddings in the pretraining dataset. To do that, they cluster the encoder embeddings from the phase-1 pretrained model using k-means with 40 clusters. They observed an uneven distribution across samples and concluded it might negatively impact model performance on underrepresented modalities. To mitigate this, they balance the sampling ratio in training by over-sampling underrepresented clusters, ensuring that the model receives a more uniform exposure to all modalities. Figure 2.18 illustrates the discovered latent modalities and the corresponding sample amplification strategy.

This transformer-based backbone performed really well in the challenge, but

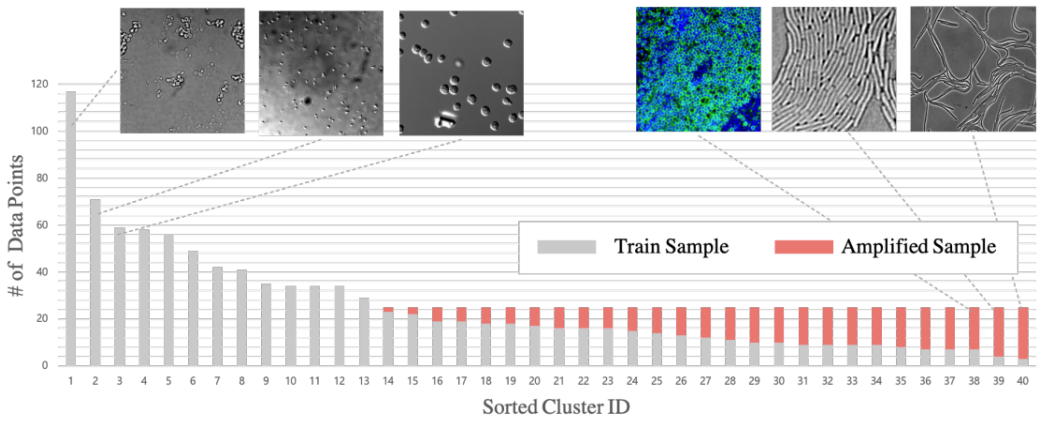

Figure 2.18: Encoder embeddings of the train set clustered with k-mean and intuition of amlplified sampling [1].

the authors of Cellpose were sceptical about the benchmarkings, claiming that conditions for pretraining of the Cellpose model were impaired. This sparked a debate about the feasibility of transformer-based models being superior to CNN backbones in the cell segmentation domain. From a theoretical perspective, this is not very simple. While the inductive bias of CNNs makes them limited to a certain receptive field but also very data efficient, transformers are able to capture long-range dependencies and global context, which may or may not be beneficial, depending on the specific task and dataset.

2.8 CellposeSAM

CellposeSAM is a very recent extention of the original Cellpose model. Since the initial launch of Cellpose1, the field of computer vision has seen significant advancements, particularly with the introduction of foundation models like the Segment Anything Model (SAM). These models have demonstrated remarkable capabilities in generalizing across diverse image domains. Motivated by these developments, the authors of CellposeSAM decided to integrate the strengths of SAM into the Cellpose framework to enhance its segmentation performance and versatility. Interestingly, they did not use the original SAM decoder, but rather combined the image encoder of SAM1 with the Cellpose learning objective. So from a high-level perspective, CellposeSAM is just Cellpose1 with a ViT, instead of the CNN backbone. For the ViT, there are only minor adjustments such as reduced patch size from 16x16 to 8x8 and input spatial dimensions from 1024x1024 to 256x256. For the local attention layers in the original SAM1, they changed all layers to global attention instead of only global in layers 6, 12, 18 and 24. In contrast to Mediar and Cellpose1, they do not use hierarchical features from different stages of the encoder, but only the output of the last encoder layer. So decoder of the network is composed of just one conv2d layer, which converts the ViT encoder neck to the cellpose learning objective. An additional, deterministic fixed kernel transposeConv2d layer

is used to transform the image from token to pixel space. The resulting model with an order of magnitude more parameters (305M vs 44M) compared to the original Cellpose1, which makes it impractical to run without a GPU.

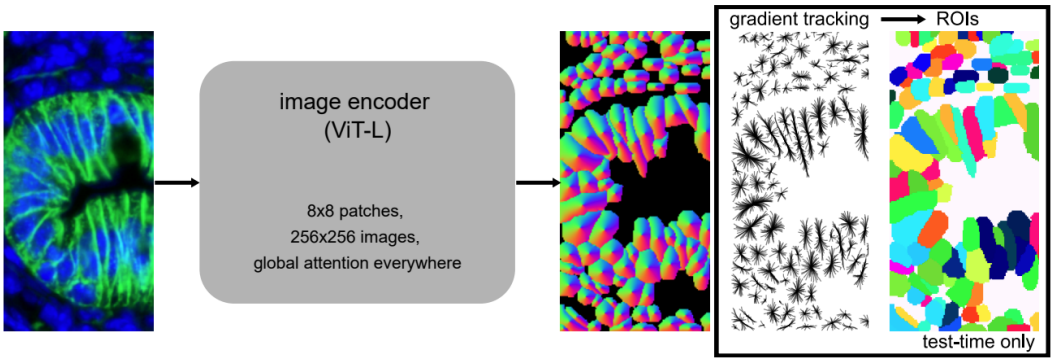

Figure 2.19: Encoder embeddings of the train set clustered with k-mean and intuition of amlplified sampling [1].

A major strength of CellposeSAM is the broad range of pretraining data. Weights are initialized with the SAM weights pretrained on the SA-1B dataset **??**, and further trained on an updated dataset of cells and nuclei containing 22,826 train images with a combined 3,341,254 training regions of interest (ROIs). This leads to a really good performance on various types of cell- and imaging modalities, as they outperform or match other models on all datasets they tested on, setting a new state-of-the-art on cellular segmentation benchmarks.

3 Methods

3.1 Datasets

In this section, we introduce some important datasets used in this thesis for model training and performance evaluation. First, we present every datasets involved in pretraining of our models. For finetuning and inference, the most interesting dataset in the context of the research of this project is the 3D+t dataset of zebrafish embryo provided by the biologists at @todo. There are multiple challenges when working with these datasets. The most important challenge is the lack of annotated data. Even for fully annotated datasets, inter-annotator agreement is often an upper boundary for evaluation, as even for the specialist a clear annotation is sometimes not obvious. This can be due to a certain level of noise in the images or boundaries that are just inherently hard to define due to the cell type or imaging modality. This is especially the case for the non-synthetic data, provided by the Celltracking Challenge (CTC) [**?**]. Next to limited annotations, 3D data can quickly blow up computational resources due to their large size. For this matter, some shifted window approach has to be considered, if the image size exceeds computational limits. Anisotropy can also can also lead to distortions in feature aquisition and performance impairments at inference, if not taken care of in each processing step.

3.1.1 Pretraining Datasets

For pretraining our models, we want to use as many 2D cell microscopy data as we can get. The modalites range from H&E stained histology images from various organs, over fluorescence, phase-contrast and bright-field microscopy images. Cell types range from human organ cells, to bacteria, worms, yeast and animal embryo nuclei.

- BCCD: The blood cell segmentation dataset consists of blood smear images taken with a light microscope (https://www.kaggle.com/datasets/jeetblahiri/bccd-dataset-with-mask). There are 1,169 training and 159 validation images in the dataset, all of which were used for training.
- Cellpose: We used the dataset from the original Cellpose paper. It is composed from various sources. We manually selected 764 images for training.
- CoNIC: The CoNIC dataset consists of 4,981 H&E images with labeled nuclei and nuclei classification (https://www.kaggle.com/datasets/aadimator/conic-challenge-dataset?select=data). We sorted out the images without labels and ended up with 4,841 images which we used for training.

- CPM15+17: CPM 15 and 17 consist of H&E images with labeled nuclei. CPM 15 + 17 are from brain cancer patients, with 15 and 32 training images per dataset, respectively.
- DataScienceBowl: The 2018 DataScienceBowl dataset contains 841 images with 22 cell types and five visually similar groups. We used 670 images for training.
- DeepBacs: We used the following segmentation datasets from DeepBacs: S. aureus bright-field and fluorescence with 66 training patches, E. coli bright-field with 34 training images, and B. subtilis fluorescence with 90; in total 190 training images.
- IHC_TMA: The IHC TMA dataset consists of TMA sections from non-small cell lung cancer patients with labeled nuclei (https://doi.org/10.5281/zenodo.7647846). There are 266 images, which were used for training.
- LiveCell: The LiveCell dataset consists of 4,704 images of 8 different cell lines collected using phase-contrast microscopy (https://sartorius-research.github.io/LIVECell/). We manually selected 3,699 images that have complete annotations.
- LynSec: LynSec consists of 699 IHC and H&E images from lymphoma patients with labeled nuclei (https://zenodo.org/records/8065174). We used all images for training.
- NeurIPS: The NeurIPS 2022 challenge training dataset consists of 1,000 images with labeled cells from bright-field, fluorescent, phase-contrast, and differential interference contrast imaging modalities (https://neurips22-cellseg.grand-challenge.org/neurips22-cellseg/). We manually selected 914 images that were annotated to a satisfactory degree.
- NuInsSeg: The NuInsSeg dataset consists of H&E images from 31 different human and mouse organs with labeled nuclei (https://www.kaggle.com/datasets/ipateam/nuinsseg). We used all 664 available images for training.
- Omnipose: This dataset contains a mixture of 14 bacterial species. We used 611 bacterial cell microscopy images and discarded 118 worm images.
- PanNuke: The PanNuke dataset consists of 7,898 H&E images from 19 tissue types from cancer patients with labeled nuclei and nuclei classification (https://warwick.ac.uk/fac/cross_fac/tia/data/pannuke). We manually selected 4,950 images that had enough annotations.
- TissueNet: The TissueNet dataset consists of 3 folds of 2,580, 3,118, and 1,324 images, collected using fluorescent microscopy on 6 tissue types with labeled cells and nuclei (https://datasets.deepcell.org/). We used all of them for training, although some appeared to have very heavy augmentations.

- TNBC: This dataset consists of 50 images from triple negative breast cancer patients, all of which were used for training (https://drive.google.com/drive/folders/1l55cv3DuY-f7-JotDN7N5nbNnjbLWchK).

- YeaZ: The YeaZ dataset consists of bright-field and phase contrast images of yeast cells. We used 401 2D images from the phase contrast dataset for training, and 306 images from the bright-field dataset for training.

3.1.2 Evaluation Datasets

To test the performance of our pretrained models, we collected datasets with fully annotated stacks and others that only have partially annotated images. Unlike the pretraining data, the evaluation datasets have to be 3D. This dramatically increases annotation costs, so finding public, fully annotated datasets here is hard. Again, DDPMs could substitute this labor intense task of annotating full 3D stacks. However, since we dont rely on true 3D-training data, the few datasets available for testing are sufficient.

BlastoSPIM

BlastoSPIM contains two large-scale, high-resolution collections of annotated light-sheet images of mouse embryos. The first dataset, *BlastoSPIM 1.0*, primarily includes embryos from the 8-nuclei stage to the 64-nuclei stage, while the second dataset, *BlastoSPIM 2.0*, focuses on later stages, from the 64-nuclei stage to embryos containing more than 100 nuclei. Together, these datasets provide a total of 653 annotated 3D images, with 573 images in BlastoSPIM 1.0 and 80 images in BlastoSPIM 2.0.

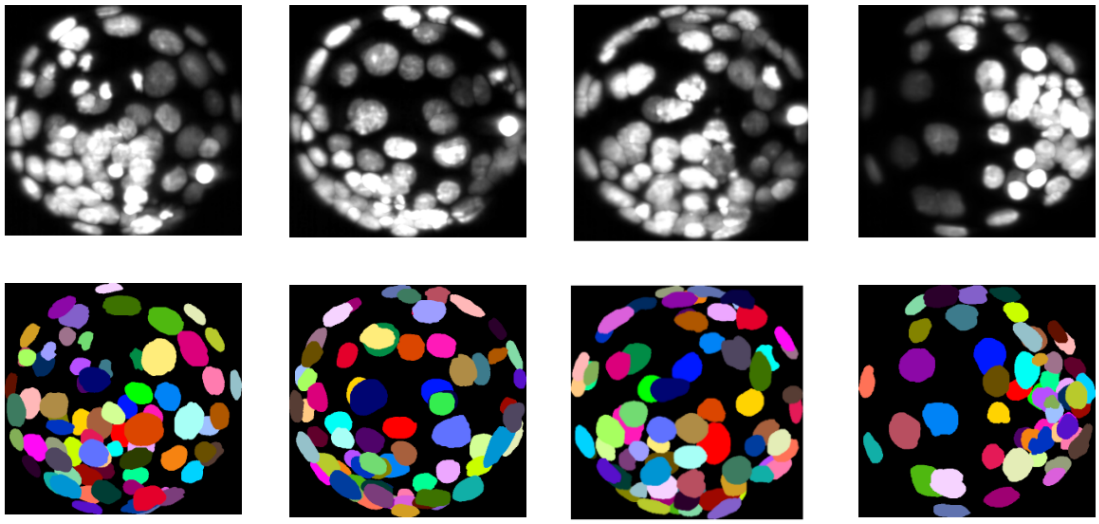

Figure 3.1: Encoder embeddings of the train set clustered with k-mean and intuition of amlplified sampling [1].

Each image in the BlastoSPIM datasets has an xy-resolution of 0.208 µm and a z-resolution of 2.0 µm, capturing fine cellular structures across multiple focal planes. In total, 18,336 nuclei have been manually segmented, with contours drawn in every z-slice where each nucleus appears. This meticulous annotation provides precise 3D instance labels, making BlastoSPIM one of the most comprehensive resources for studying early embryogenesis and for training and evaluating segmentation models in 3D microscopy.

CTC Datasets

The CTC dataset provides both 2D and 3D time-lapse microscopy sequences. It includes recordings of fluorescently counterstained nuclei and cells, either migrating on top of or embedded within a substrate. In addition, the dataset contains 2D brightfield, phase contrast, and differential interference contrast (DIC) microscopy videos that capture the dynamics of cells moving on flat substrates.

Interesting, challenging datasets with very dense instances are the Fluo-N3DL datasets, which are light-sheet 3D+t images of developing insect embryo. Each image is really large and requires careful memory handling in inference. Due to the large size and density in cell instances, annotations are very limited. For every image, only a few z-slices have annotations, which usually only cover a tiny fraction of the cells present.

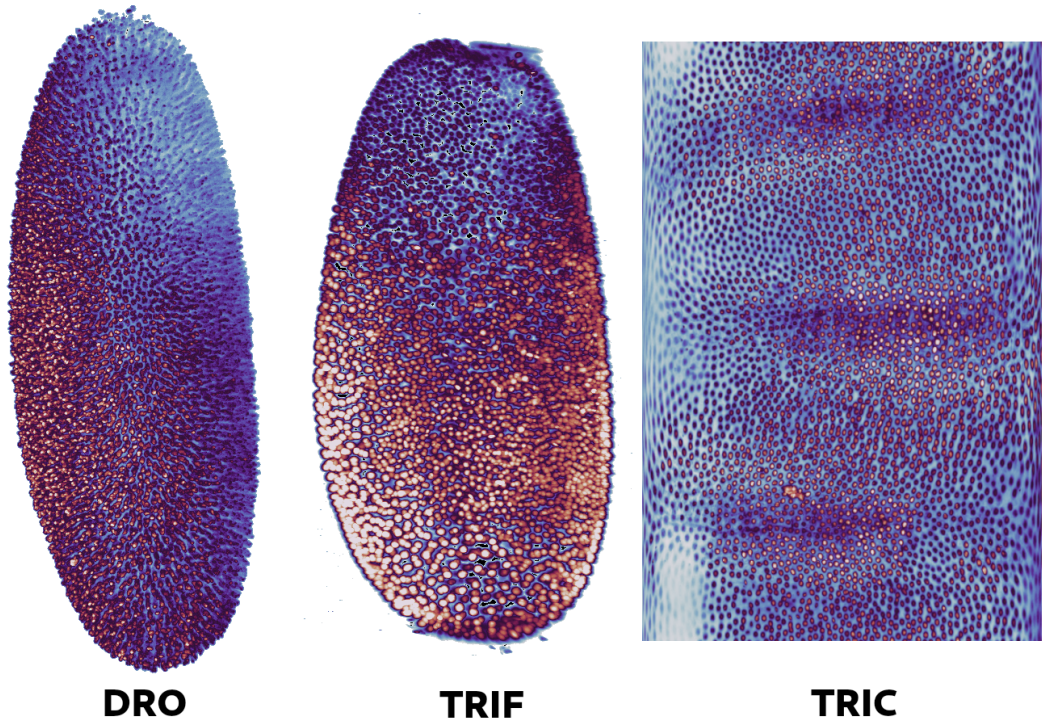

Figure 3.2: Fluo-N3DL-DRO, TRIF and TRIC

We will also use some synthetic images from the Fluo-N3DH-SIM dataset, which simulates nuclei of HL60 cells stained with Hoechst. Due to the high level of

noise in the data, clear boundaries are not really present, limiting the performance evaluation capabilities.

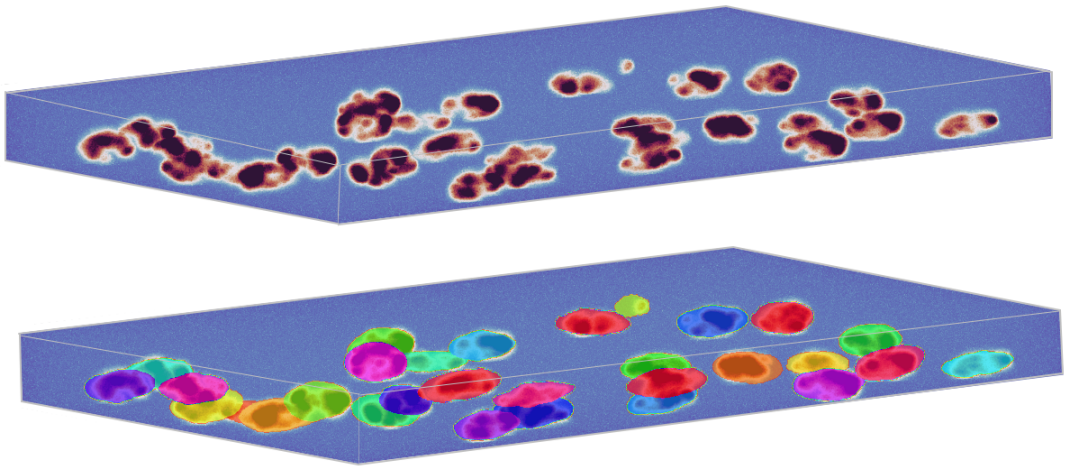

Figure 3.3: Fluo-N3DL-DRO, TRIF and TRIC

Zebrafish Dataset

This dataset captures zebrafish embryo's lateral line in 3D+t. In zebrafish embryos, the lateral line system is composed of small mechanosensory organs called *neuromasts*. Neuromasts originate from cranial placodes and are deposited along the trunk and tail during early development (around 24–48 hours post fertilization). This deposition is carried out by the posterior lateral line *primordium*, a migrating cluster of progenitor cells that travels from the head toward the tail and leaves behind groups of cells that mature into neuromasts. Each neuromast consists of centrally located sensory hair cells surrounded by supporting and mantle cells, which provide structural and regenerative functions. By 3–5 days post fertilization, zebrafish larvae possess a functional lateral line system with multiple neuromasts arranged in stereotyped rows. For our purposes, we focus on segmenting the migrating cluster of progenitor cells as well as the leftover neuromasts. Images are acquired with a Zeiss spinning disk and Nikon laser scanning microscope, resulting in two channels for each membrane and nuclei. Although the mebrane channel can act as a useful bias in addition to the nuclei channel in tracking, the level of noise and unclear boundaries makes the membrane channel unfeasible for segmentation. Therefore, we will primarily focus on the nuclei channel for segmentation tasks.

As depicted in Figure 3.4, the primordium cells can be recognized quite easily with the bare eye. This is not only due to favourable intensity contrasting, but also because of the unqiue shapes compared to the background cells. Nevertheless, if there is no bias towards the different types of cells, generalizable networks will not be able to differ forground from background cells and thus produce a lot of false positives.Although there are a lot images at different timestamps and images for

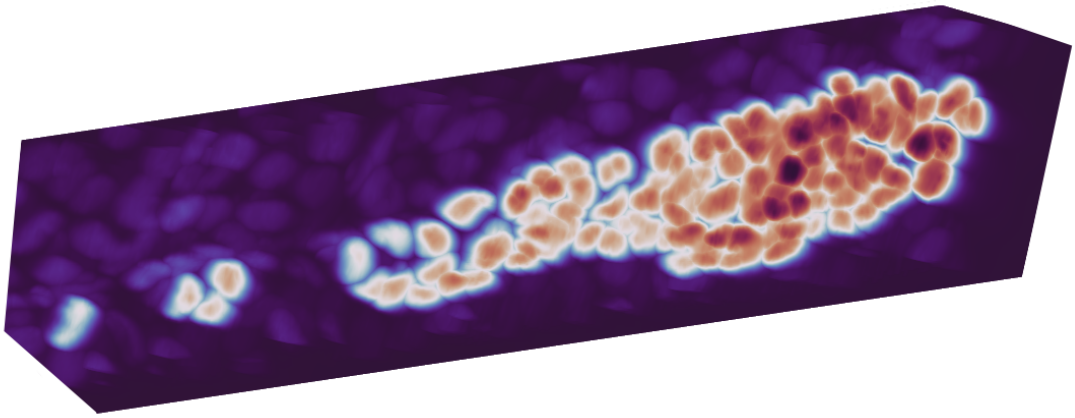

Figure 3.4: Fluo-N3DL-DRO, TRIF and TRIC

wildtype variations, there is just one single 3D image labeled. Although most of our models train on 2D slice inference anyway, testing and training on the same image at different slices is rather infeasible due to strong spatial correlations between test and train data.

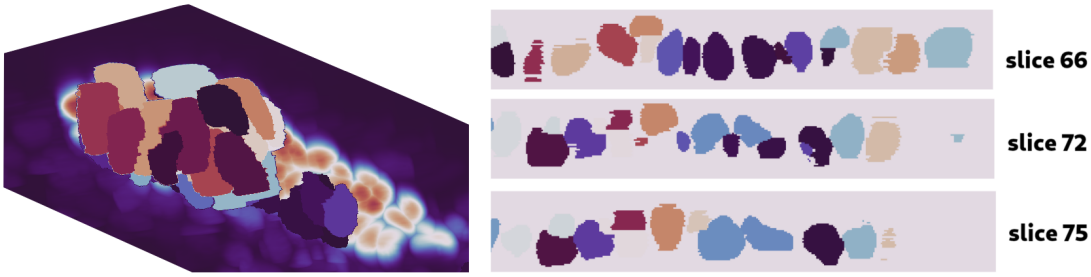

Figure 3.5: Fluo-N3DL-DRO, TRIF and TRIC

Another challenge with this dataset is not only that there is just one annotated image, but that the segmentation quality also impaired by the way the annotation was created. As it can be easily spotted in 3.5, there are inconsistencies in cell annotations towards different z slices. Most likely, the annotator marked cells in just axial direction, leaving out annotations on some slices, while on some z slices further they are marked again. Also, the boundary sizes are inconsistent over z. This is particularly crucial, as our models predict slices in axial, sagital and coronal directions. So when slicing through x for example, the inconsistencies will create distortions in the flow feature space, which is aggregated from all inference directions.

3.2 Mediar3D

In this section, we introduce a way of extending Mediar to 3D. First, we present a way of using predictions along different image reslices, to compute a 3D flow

field, which can be post processed into 3D predictions. We will briefly introduce challenges that go along with this method and discuss the feasibility of extending Mediar to true 3D. This would correspond to an architecture that learns actual 3D features during training.

3.2.1 3D Flowfield Prediction

For the extension to 3D flow predictions, we adapt the flow accumulation of Cellpose. For 2D inference, a two-dimensional vector field is predicted. A simple way to perform inference on 3D data, would be to loop over the depth z of the image, to obtain the vertical and horizontal flows dx and dy related to z. Now when permuting the spatial dimensions to slice over x instead of z, essentially treating x as the new depth dimension, one obtains the dy and dz flows. Finally, repeating this over the y dimension grants dx and dz. While this sounds very intuitive, it is important to keep track of the correct way of inversely permuting the flows back to the original coordinate system. This is achieved by accumulating them into a resulting 3D flow-field vector at the correct indices. To threshold invalid flows, predicted cell probabilities are accumulated along flows as well.

Algorithm 1 3D Flow Accumulation with Iterative Permutations

Initialize 3D flow-probability volume $\mathbf{F} \in \mathbb{R}^{4\times H\times W\times D}$ with zeros
Define permutations π_p and inverse permutations π_p^{-1} for $p \in \{x, y, z\}$
for each spatial dimension $p \in \{x, y, z\}$ **do**
    Permute input volume I to $I^p = \pi_p(I)$
    for each slice s along depth of I^p **do**
        Compute 2D flow $\mathbf{f}_s = (dx_s, dy_s)$ and cell probability $P_s = \sigma(I_s)$ via 2D inference
    end for
    Stack slices to form 3D block of flows $\mathbf{F}^p \in \mathbb{R}^{4\times H\times W\times D}$
    Inversely permute flows back: $\mathbf{F}^p_{\text{orig}} = \pi_p^{-1}(\mathbf{F}^p)$
    Accumulate: $\mathbf{F} \leftarrow \mathbf{F} + \mathbf{F}^p_{\text{orig}}$
end for
return $\mathbf{F}$ (final 3D flow field and cell probability map)

3.2.2 Post-Processing of 3D Flows

We now take a look at computing instance masks. Given the network forward outputs, we optain spatial gradients $\mathbf{dP} \in \mathbb{R}^{3\times L_z\times L_y\times L_x}$ and cell probabilities $\mathbf{P} \in [0, 1]^{L_z\times L_y\times L_x}$ to process the resulting cell instances. Candidate voxels are first identified by thresholding the probability map, defining the foreground region

$$\Omega = \{(z, y, x) \mid \mathbf{P}(z, y, x) > \tau_p\}, \tag{3.1}$$

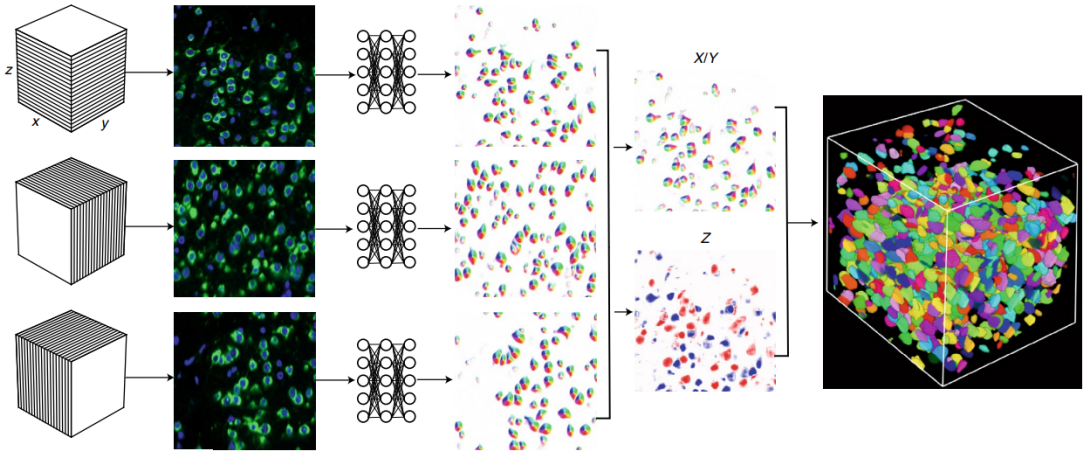

Figure 3.6: Encoder embeddings of the train set clustered with k-mean and intuition of amlplified sampling [1].

where τ_p is the cell probability threshold. For each voxel $(z, y, x) \in \Omega$, the position is iteratively advected along the predicted flow vectors according to

$$\mathbf{p}_{t+1} = \text{clip}(\mathbf{p}_t + \mathbf{dP}(\mathbf{p}_t),\ -1, 1)\,, \tag{3.2}$$

where $\mathbf{p}_t \in \mathbb{R}^3$ denotes the normalized voxel position at iteration t. The integration is performed for a fixed number of steps T (typically $T = 200$) using Euler dynamics, which drives voxels belonging to the same object towards common attractors in the flow field. The final positions $\mathbf{p}_T$ of all advected voxels are accumulated into a discrete histogram volume

$$H(\mathbf{u}) = \sum_{i=1}^{|\Omega|} \delta(\mathbf{p}_T^{(i)} - \mathbf{u}), \tag{3.3}$$

where $\delta(\mathbf{p}_T^{(i)} - \mathbf{u})$ acts as a one-hot indicator function that evaluates to 1 if the advected voxel $\mathbf{p}_T^{(i)}$ coincides with the discrete location $\mathbf{u}$ and to 0 otherwise. Local maxima of H are detected by n-dimensional max pooling with kernel size 5, and peaks with histogram counts above a fixed threshold are retained as seeds, which represent candidate cell centers. Around each seed, a local neighborhood of size $11 \times 11 \times 11$ voxels is extracted and iteratively expanded using morphological max pooling under the constraint $H(\mathbf{u}) > 2$, thereby forming seed masks corresponding to attraction basins of the detected maxima. The converged voxel positions $\mathbf{p}_T$ are then mapped to these seed regions, and each voxel in Ω is assigned a label $m \in \{1, 2, \ldots, N\}$, where N is #detected seeds. This produces an initial labeled segmentation. Finally, masks that are larger than a fraction τ_s of the total image volume are discarded, very small masks are removed and filled, and masks that exhibit insufficient alignment with the flow vectors are pruned using a flow-consistency criterion. Remaining masks are renumbered to enforce consecutive integer labels, resulting in the final instance segmentation.

3.3 Segmentation Network Design Choices

In this section, we introduce different approaches to improve the Mediar architecture by using different encoder and decoder building blocks. First, we will sketch the fundamental structure of our segmentation networks on a high-level view. Based on this concept, we will be build new models by modifying indivial components. Inspired by the release of CellposeSAM, we will present a model that uses the SAM2 pretrained Hiera image encoder instead of the SegFormer MiT blocks for the Mediar Network. Furthermore, we will consider the use of plain convolutional decoder blocks instead of the MA-Net decoder. We will also attempt to reason the rationale behind these design choices and, in doing so, set our expectations for the resulting performance of the network modifications.

3.3.1 High-Level Network Design

In this subsection, we deduce some general concepts to our network design and point out their design choices on a high level. As previous work such as CellposeSAM, Mediar and other segmentation networks suggest the encoder-decoder style learning, we adapt this as our underlying concept for our network architecture.

For the encoder part, the image is transformed into a latent feature space by alternating channel size while downsampling spatial dimensions as the network gets deeper. From a high-level perspective, this can be regarded as a module $E(\mathbf{I})$, that inputs an image $\mathbf{I} \in \mathbb{R}^{3\times H\times W}$ and outputs a tuple of multiple feature spaces:

$$E(\mathbf{I}) = \mathbf{F}_i \in \mathbb{R}^{C_i\times H/2^i\times W/2^i} \mid i = 1,\dots,N, \tag{3.4}$$

where the tuple $\mathbf{F}_i$ may contain an arbitrary number of feature resolutions i, each with its own channel dimension C_i and spatial size $H \times W$. Note that while it is not necessary to reduce spatial size to integer powers of 2, this is commonly the case due to the nature of downsampling mechanisms such as strided convolutions or pooling operations.

The decoder fuses all information gathered by the encoder into an overall network prediction by mapping the features to the learning objective of the network. This includes upsampling the spatial dimensions back to pixel space and reducing the channel dimensions to the desired output shape. From a high-level perspective, the decoder can be written as a mapping

$$D(\mathbf{F}_i) = \hat{\mathbf{Y}} \in \mathbb{R}^{C\text{out}\times H\times W}, \tag{3.5}$$

where C_{out} denotes the number of output channels, depending on the learning objective. In semantic segmentation, this corresponds to the number of semantic classes, with a softmax distribution assigning each pixel to a class. For cellular instance segmentation, the decoder instead maps the features to outputs such as flow fields or probability maps, which are subsequently post-processed into instance masks, as done in StarDist or Cellpose.

Beyond this high-level network layout, which already involves many design decisions, each building block obviously contains numerous other parameters and layer

configurations that must be carefully chosen. But focussing on the above mentioned high-level design criterions is important to keep track of the bigger picture when it comes to a large change in network layout, which will be presented in the following subsections. Understanding the changes made on the above described high-level layout, will make a seemingly completely new architecture, very tangible and the low-level design choices a lot more intuitive.

For instance, the authors of Cellpose launched a completely revised version of their network a few weeks after the start of my thesis. What seems to be a major rework of their repository, turns out to be just a few lines of code, as they barely touch the low-level components of the modules they take from established methods. So from a high-level view, they switched the CNN based encoder part of their network with the image encoder of SAM1, and used just a single 1x1 convolutional layer (along non-learnable layer to convert patch- to pixel space) to fuse their single resolutional feature map to the cellpose learning objective.

3.3.2 SAM2 Encoder Registry

First, we want to setup experiments using the Hiera encoder of the SAM2 model instead of the SegFormer encoder blocks for the Mediar model. The SegFormer backbone blocks are composed of multiple Mix Transformer(MiT) blocks with consecutive overlapping patch merging, efficient self-attention and Mix-FFN, which provide multi-scale features due to the patch merging process at multiple stages. From a high-level perspective, the MiT B5 encoder blocks produce 4 feature maps with different channel sizes and spatial resolutions. The Hiera image encoder is composed of standard ViT blocks, except global attention is replaced by Mask Unit Attention at shallow layers with high spatial resolution. A total of 3 pooling layers enables the encoder to output features at four different resolutions. This excatly matches the SegFormer encoder outputs on a high-level, except the feature maps produced by the Hiera encoder all have 256 channels, whereas the feature maps of the MiT B5 module all have increasing channel sizes for deeper feature maps. Note that also the default config used to build the SAM2 registry defines a parameter that makes the image encoder discard the lowest resolution feature after forwarding the neck, resulting in only 3 feature resolutions instead of 4. Hardcoding this value or editing the default registry config fixes this problem because the powerful pretrained weights from the SAM2 registry are not effected by the amount of feature outputs of the image encoder. This is because the features are computed in the forward pass regardless of the parameter that discards them afterwards by parameterized indexing in the neck forward function.

3.3.3 Merging SAM2 Image Encoder with Mediar

The actual integration of the SAM2 model into the Mediar network architecture is not too difficult if conducted carefully with debugging tools. Challenging aspects include dealing with cryptic, undocumented backend code of MA-Net and MixTransformer and more importantly, editing the MA-Net blocks with regard to

numerical stability and compability with the SAM2 registry. Pseudo QKV operations in the PAB Block of the MA-Net decoder are now normed with respect to the embedding dimension as suggested in **??**, to prevent float16 overflow in network training. Also, MA-Net assumes the first feature to be the encoder input image and thus discards the shallowest feature per default. This is now adjusted to match the backbone outputs from Hiera. Skip connections have been removed for the Decoder Block 1, as this block was connected to a dummy feature from the SegFormer output.

3.3.4 Lightweight Decoder

Many segmentation networks [?, ?] have shown great performance without relying on heavy-weight decoder modules such as it is the case for Mediar and its MA-Net decoder blocks. We therefore want to conduct experiments using leight-weight decoder blocks and the SAM2 image encoder.

The decoder blocks take a feature map as input and a skip connection from the encoder if there it is not the center block. The input is first upsampled by a factor of two using nearest-neighbor interpolation. When a skip connection is provided, the upsampled features are concatenated with the skip features along the channel dimension, increasing the input dimensionality of the subsequent convolution. The combined tensor is then processed by two sequential conv2dReLU units, each consisting of a 3×3 convolution with padding of 1, followed by a ReLU activation, and a batch normalization layer. The first convolution reduces the concatenated feature channels to the target output dimensionality, and the second further refines the representation while maintaining the same number of channels. This design focusses on spatial resolution recovery and integration of encoder features at corresponding scales, without using the heavy-weight pseudo attention modules of the MA-Net.

3.4 Training Challenges and Solutions

In this section we will analyse the training procedure of our networks. First, the general training pipeline is outlined, describing the main components. After setting the baseline for cellpose learning objective training, we will look into challenges that come along training with only partly annotated data.

3.4.1 Training Pipeline

Now we look into the general pipeline for our network training. In general, we want to train our models in such a way, that it is capable of performing well on zero-shot predictions. To achieve this, we follow the approach of previous work about generalizable model training. This includes *pretraining* the data on a large, diverse dataset. This model can be further *finetuned* on a specific downstream modality. The original Mediar training config suggests a multi-phase training of two seperate

models that predict in an ensembling manner (this multi-phase training is described in **??**). Since this multi-phase training is tailored for the best performance on a large challenge dataset, this method is unsuitable for small target datasets, especially if the target datasets have limited annotations. Within this thesis, we therefore aim to produce a single phase, generalizable model, which Mediar refers to as their *from_phase1.pth* model.

Data preperation and sampling

Since pretraining and finetuning share mostly the same pipeline, adjustments should be configured in a seperate train config json file. The main difference between pretraining and finetuning configs lies the data preperation and sampling. Pretraining mainly focusses on using a very large, diverse dataset. This can also be improved, by the discovery of latent modalites, as the original Mediar paper suggests 2.18. By clustering the encoder embeddings from the phase-1 pretrained model using k-means with 40 clusters, they amplify the sampling ratio in favour of underrepresented modalities. We adapt this method as the majority of pretraining images are worm images from the Livecell-dataset. Other than the amount of data, it is also important to handle the dimensionality of the target data for the finetuning. Since we want to finetune our model to perfrom well on specific 3D data, we train our model to predict the 2D flows and cell probablities on the 2D slices of the volume in each orientation. This requires to prepare slices of the 3D finetuning data for every direction.

Loading the training data into the training pipeline requires the mapping of train images and ground truth masks to json file. This is done via a modified version of the mediar mapping script, which enables the user to map multiple datasets at once, without naming constraints. The json files are loaded and split into train and validation data with a custom dataloader, that is configured via user inputs defined in a seperate train config json file. Using the monai package, which is a precompiled library for biomedical image processing, certain train and validation transformations are defined in the dataloader to augment the batch data once its loaded in a training iteration.

Forward pass

After loading the batch data in each training epoch, the network forward pass predicts 2D flows and cell probability logits, which are passed into the metric function to compute the loss. For the ground truth data, the flow representation is computed runtime. While this sounds very inefficient, precomputing the flow ground truths comes along with a few challenges. The precomputed flows have to be integrated into the dataloading pipeline. This includes applying the same spatially deforming augmentation transformations to the flows as the training data. Since the flow generation of ground truth data is not invariant to all spatial transformations, the precomputed flows dont match the flows computed at runtime after the transformation. The unit test for this matter has shown, that in most cases the reconstructed masks from the precomputed and runtime computed flows differed

significantly, while the time costs saved was in the magnitude of a few milliseconds per epoch, partly because the dataloading and augmentation of the precomputed flow has to be substracted from the time it takes to generate the flows runtime. Although these few milliseconds scale up pretty quickly to a 1-2 hours for 80 epochs at 5500 iterations per epoch, the performance loss can not be justified. Eschweiler et al [?] suggested to simplify the learning objective to a variant of the tanh to speed up the expensive generation of flows at neglectable performance loss. While this might be a promising alternative, it has not been implemented and tested yet.

With runtime computed ground truth flows, the loss can be calculated after the forward pass with a binary cross-entropy term for the cell probabilities and a mean squared error term for the flow components:

$$\mathcal{L} \;=\; \|\hat{y}_0 - 5F_x\|^2 \;+\; \|\hat{y}_1 - 5F_y\|^2 \;+\; \mathcal{L}_{\mathrm{BCE}}\big(\sigma(\hat{y}_2), P\big), \tag{3.6}$$

Here, $\hat{y}_0$ and $\hat{y}_1$ denote the predicted horizontal and vertical flow channels, while F_x and F_y represent the corresponding ground-truth flow fields. The term $\hat{y}_2$ refers to the raw (logit) prediction of the cell probability map, and P denotes the ground-truth binary mask of cell locations. The function $\sigma(\cdot)$ corresponds to the sigmoid activation, which maps logits to probabilities, and $\mathcal{L}_{\mathrm{BCE}}(\cdot,\cdot)$ is the binary cross-entropy loss between the predicted and ground-truth probabilities. The ground-truth flows are scaled by a factor of 5 to match relative contribution of the L2 loss and cross-entropy loss. Although it may seem like a semantical error to scale the ground truth flows before substraction, it works well in practice. Unfortunetely, there was no explaination given in the paper on why they implemented it like this. A raised github issue addressing this issue remained unanswered. Usually, it is common to scale loss contributions after the whole loss component is calculated. Either way, experiments with a different way of scaling the loss contribution have shown no difference in performance.

Backward pass

In practice, we implemented the individual loss components are computed seperately for logging purpose, which can be handy to tune the weighting of the loss components. Interestingly, the Mediar code logs the summed loss components as dice loss, although the dice loss is neither computed in valid epochs, nor in the training iterations. We log the loss to Weights and Biases, which is an interactive cli and webtool that can be integrated into the training pipeline to plot losses and performance while training. It can be used for free with an academic license. The per-pixel MSE flow and BCE cell probablity loss is aggregated into a scalar value with the build-in mean reduction method from torch and backpropagated with the AdamW optimizer.

Since all networks share the same learning objective, the training pipeline can be mostly reused. Apart from hyperparameters that depend on the network's feature aquisition and amount of pretraining, the metric function and dataloading including the module for ground truth flow generation stays the same. So the pipeline is used

for all networks, but changes in the way the loss is computed, depending on is the training data is fully annotated. How these challenges are faced, will be described on the following subsection.

3.4.2 Training with Limited Annotations

Since 3D data labeling is an expensive task, many challenge or use-case datasets are only labeled partially, as described in **??**. Since oftentimes this is the only resource to significantly improve the zero-shot performance for these target datasets, we will look into ways to finetune our networks with limited annotations.

Using Unsupervised Learning Methods

Previous work has explored various strategies to incorporate large sets of unlabeled images into training. For example, the Mediar framework investigated several semi-supervised approaches, including consistency regularization with strong data augmentations, reconstruction losses with an auxiliary head (both restricted to the cell regions and over the full image), and pseudo-labeling from pretrained or fine-tuned models. However, these attempts did not yield improvements over training with labeled data alone, leading the authors to ultimately rely only on the annotated subset. This highlights the difficulty of leveraging unlabeled data effectively in training and motivates alternative approaches.

Masked Loss Function

Finetuning the models with limited annotations, requires careful evaluation of network forward passes. Naively applying BCE and MSE loss to the predictions will punish false positives a lot, in case only a few instances are labeled. Not only will the contour accuracy decrease, but the arbitrarely labeled instances will make it impossible for the network to make distinctions for true and false positives as epochs progress. To decrease gradient distortions of unlabeled areas, the image shown to the network in forward pass can be cropped to a ROI, corresponding to a bounding box of a few nearby instance labels, with some spare buffer spaces to provide some image context. While this heavily reduces the effects of the majority of the unlabeled image parts, this approach is still prone to gradient distortions in cases of dense cell regions, where a buffered bounding box can not be established without including unlabeled instances.

To prevent false network weight gradient flow due to pseudo false positives completely, one could crop the image to individual cell instances without buffer space to prevent gradient distortions by unlabeled neighboring cells. While this is not always feasible, especially in really dense cell distributions, this also prevents the network to learn spatial context features of the cell instances. As the instance cropped images has to be padded severely to fit the input dimensions of the network, the network would mostly see zero values around the cell. This could lead to even worse feature learning, especially because the receptive field of transformer based encoders can be global.

Another way of dealing with the gradient contribution of unlabeled regions would be to define a supervision mask, that sets the loss to zero for regions, where no annotations are present. As there would be no false positive feedback loss contribution for the case that the loss is only calculated for positive regions, it is necessary to somehow extend the region of loss contribution beyond labeled regions. In order to not only get false negative feedback, the supervision mask for the loss could be extended with binary operations such as dilation. The kernel size for dilation and the amount of iterations for consecutive dilation, have to be carefully tuned, as potential pseudo false positive can be introduced again. This can be handled again by introducing some sort of distance weighting of the loss mask around the labeled region. The loss contribution could for example decay linearly by the cartesian distance, or even in a guassian manner. Note that the tuning of the parameters such as dilation iterations, kernel size and distance weighting are heavily depended on the specific dataset.

3.5 3D Feature Network

In this section, we investigate a few strategies to encourage the network to learn 3D features. First, we look into a method of adding 3D flow gradients to the loss. Then, we shortly discuss the feasibility using 3D building blocks for the encoder and decoder. Lastly, we build a new architecture that encodes 2D slice features and uses them to learn 3D features in the decoder path, using 3D convolutions.

3.5.1 3D Gradient Flow Training

We first look into a method of adapting the same 3D flow aggregation technique for training, as it used for inference. The idea of 3D flows is, to slice a volume into 2D slices for every cartesian direction (x, y, z), forwarding them into the network to predict 2D flows for every direction, to then be able to aggregate them into 3D flows. This can be integrated into finetuning, given that 3D instance annotations are provided. The 3D flow loss requires multiple consecutive slices to be forwarded within one batch, to then calculate 3D flows and getting three MSE flow loss components. The algorithm is summerized in Algorithm 2. In case of unlabeled regions, the same loss masking could be applied as described in the previous section.

Algorithm 2 Training with masked 3D flow loss

1: **for** each optimizer step **do**
2: Load one 3D image I and its mask M
3: Slice I into 2D images along YX, ZY, and ZX directions to form a batch B
4: Perform a forward pass of B through the network to obtain predictions $\hat{B}$
5: Using knowledge of YX, ZY, and ZX indices in $\hat{B}$, compute 3D flows $\hat{F}$
6: Compute masked MSE loss between $\hat{F}$ and ground-truth flows F
7: Update network parameters via backpropagation
8: **end for**

While this approach sounds quite logical at first, there are a few issues. First and most importantly, as the network still forwards 2D slices, there are no actual 3D features learned like it would be the case for the 3D Cellpose network proposed by Eschweiler et al. In our setup, the incorporation of 3D flows in the loss metric function primarily acts as a regularizer. While it does encourage consistency of predicted flows across slices in all three spatial directions, it doesnt make use of the 3D annotations to learn actual 3D spatial features. Another drawback of this method is, that loading even subvolumes in a sliding window manner quickly becomes very memory expensive. In practice, comsumer grade GPUs can not handle more than a batch size of 9 with all network weights at the same time for training. Due to these limitations, we decided to not persue this approach any longer.

3.5.2 Feasibility of Extending Mediar to True 3D

Since the proposed network of Eschweiler et al. has shown great performance in learning true 3D features, by using 3D convolutional layers, it seems reasonable to extend the encoder and decoder components of Mediar to true 3D. While this would not be technically impossible, as previous work has successfully implemented 3D attention layers for image segmentation [**?**], global attention in 3D would severely influence computational complexity. More importantly, training these blocks from scratch would require a lot of annotated 3D training data, due to the lack of inductive bias in the architecture. We would ultimately lose the strong generalizable power from the heavily pretrained 2D encoders. Since pretraining data for 3D cell-segmentation is in the magnitudes of a few hundrets, this approach seems unlikely to succeed. As Carsen Stringer, the original author of cellpose suggests, CNNs would definetely be more suitable for training a true 3D cellpose variant from scratch because of the above mentioned reasons.

3.5.3 Hybrid Approach for Learning True 3D Features

In this subsection, we present a different approach for learning true 3D features. We adapt the design of xy et al [**?**], who keep the strong pretaining power of a foundational encoder and train a seperate 3D convolutional decoder to learn 3D features based on the forwarded slices of the 2D encoder.

Specifically, they used the SAM1 image encoder registry as the encoder and a lightweight decoder. The input image $I \in \mathbb{R}^{H \times W \times D}$ is sliced along the depth dimension, converted to a 3-channel image and forwarded through the encoder. The resulting slice embeddings $F = [f_1, f_2, \ldots, f_D] \in \mathbb{R}^{\frac{H}{16} \times \frac{W}{16} \times D \times 256}$ are stacked to form a 3D feature volume F, which is then processed by the 3D decoder to capture depth-wise relationships and map the features to the Cellpose learning objective. The forward pass is summarized in Algorithm 3.

Algorithm 3 Hybrid 3D Feature Network Forward Pass

Require: Volumetric input $I \in \mathbb{R}^{H \times W \times D}$
Ensure: Cellpose learning objectives $\hat{Y} \in \mathbb{R}^{H \times W \times D}$
Split I into D slices along the depth axis: $I = \{I_i\}_{i=1}^{D}$, $I_i \in \mathbb{R}^{3 \times H \times W}$
for $i = 1$ to D **do**
    Compute slice embeddings: $f_i = \text{Enc}(I_i)$ $\{f_i \in \mathbb{R}^{H/16 \times W/16 \times 256}\}$
end for
Stack slice embeddings to form 3D feature volume:
$F = [f_1, f_2, \ldots, f_D] \in \mathbb{R}^{H/16 \times W/16 \times D \times 256}$
Pass the stack embeddings to the 3D decoder layers:
for *block* in *DecoderBlocks* **do**
    $\hat{Y} = \text{block}(F)$
end for
return $\hat{Y}$

To further tailor this method for our purpose, we take not only the pretrained SAM1 image encoder registry, but also use the weights from the pretrained CellposeSAM encoder, that should already be quite potent for providing cell segmentation slice embeddings. We therefore dont need to train the encoder seperately and freeze the weights completely, while training the 3D decoder from scratch.

Although this approach performed well on medical image segmentation, there are still some potential drawbacks of this hybrid method. As the 3D decoder is trained from scratch, a lot of 3D annotated data will be needed which is not always available for specific datasets. Also, strong performance across multiple modalities, particularly on modalities not present during training, is not expected for the amount of pretraining data available. Potential cures for that matter could be diffusion models that synthesize large amounts of fully annotated training data. This has not yet been tested.

4 Experiments

In this chapter we want to test our proposed architectures on several datasets, presented in **??**. First, we will pretrain our models on a large 2D dataset, using different sampling and augementation strategies. After that, we target specific datasets to finetune our models on. With these weights, we compare inference performance on various datasets. Finally, we will conduct some experiments with the 3D feature network.

4.1 Model Pretraining

For pretraining our Mediar backbone based models, we collected and formatted 27,422 of 2D microscopy images from various imaging modalities and cell types. To mitigate bias from overrepresented modalities, a custom dataloader generates training batches according to predefined sampling probabilities for each dataset. We manually define hardcoded sampling probabilities based on dataset sizes, cell type variability and imaging modalities. The sampling probabilities are displayed in table 4.1

Table 4.1: Sampling probabilities for each dataset.

Dataset	**#Images**	**Variability**	**Probability**
LiveCell	3,699	mid	0.05
Cellpose	764	very high	0.19
DataScienceBowl	670	mid	0.08
BCCD	1,328	mid	0.04
CoNIC	4,841	low	0.02
Deepbacs	190	low	0.04
IHC_TMA	266	low	0.03
TNBC	50	low	0.05
CPM15	15	mid	0.06
CPM17	32	mid	0.07
LynSec	699	low	0.04
NeurIPS	914	mid-high	0.06
NuInsSeg	664	low	0.03
Omnipose	611	mid	0.09
PanNuke	4,950	low	0.03
TissueNet	7,022	mid-high	0.06
Yeast_BF	306	very low	0.04
Yeast_PhC	401	very low	0.03

The qualitative evaluation of the dataset's variability is based on the correlation analysis of Cellpose, between the style vectors (global average pooling of CNN

Cellpose feature maps at bottleneck) of dataset subsets. Visual inspection is done for those datasets that have not been in the correlation study of cellpose. Note that a more sophistcated approach would be to persue a similar approach to the method used in the original Mediar paper, where they analyse the clustered t-SNE representation of the encoder embeddings. However, this approach would also require to manually pick the amount of k clusters for the k-means algorithm, if no elbow heuristics are present for a within-cluster sum of square plot. We chose a similiar weighting of the sample probabilities to the CellposeSAM paper, but scaled down the weighting of the cellpose dataset, as they weighted it up to 59%, which would be almost 60 times higher than the weights for most other datasets they used. Not that the term sampling probability might be a little misleading, as the probability of an image of a certain dataset being batched scales also with the dataset size.

Table 4.2: Development environments and requirements.

Environment	**Specification**
System	Rocky Linux 9.6
CPU	2x Intel Xeon 8468 Sapphire CPU@2.1 GHz, 48 cores each
RAM	512GB
GPU	4x NVIDIA H100 (96 GB HBM2e)
CUDA version	12.1
Programming language	Python 3.9.21
Deep learning framework	Pytorch (v2.1.2, with torchvision v0.16.2)
Code dependencies	MONAI (v1.3.0), Segmentation Models (v0.3.3)

For the Mediar network with the MiT Segformer encoder module, we use the pretrained ImageNet-1k weights from the SegFormer paper. The networks with the Hiera encoder, are intilized with the SAM2 image encoder registry, pretrained on the SA-V dataset. We discard the multi-phase training approach from the original Mediar training pipeline, because it was tailored for the unsupervised NeurIPS challenge dataset. In short what they do in their multi-phase training is training two seperate models that have different levels of bias towards the target modalities. This ensembling only has meaningful impact if the target datasets has a lot of different modalities such as it was the case in the NeurIPS challenge dataset. So for training a model that generalizes well over unseen modalities and that is able to be finetuned for one target modality, training just one model considered the best choice. To achieve that single genralizable model, we follow the same path as CellposeSAM by pretraing the model on basically every public 2D dataset there is. The resulting dataset described in **??** is almost four times larger than the pretraining data from Mediar. For the networks with the Hiera encoder, we use the SAM2 image encoder registry, pretrained on the SA-V dataset. All models, we train for 80 epochs with a batch size of 64, distributed over four H100 GPUs with model weight wrapping Multi-GPU training. The optimizer is set to AdamW with an initial learning rate of 5e-5, decaying using cosine scheduler with intervals of 100 without restarts. The environmental and learning setups are listed in tables **??** and **??**, respectively.

Table 4.3: Pretraining Setups

Setup	**Mediar**	**MediarSAM**	**MediarSAM light**
Encoder	MixTransformer []	Hiera []	Hiera []
Decoder	MA-Net []	MA-Net []	Conv + Res Blocks
Initialization (Encoder)	Imagenet-1K []	SA-V	SA-V
Initialization (Decoder)	He normal init	He normal init	He normal init
Loss function	MSE, BCE	MSE, BSE	MSE, BSE
Mixed precision []	disabled	disabled	enabled
Optimizer	AdamW []	AdamW []	AdamW []
Initial learning rate (lr)	5e-5	5e-5	5e-5
Lr decay schedule	Cosine [] (100 interval)	Cosine [] (100 interval)	Cosine [] (100 interval)
Epochs	80	80	80
Training time	87h	94h	76h
Model parameters	121.30 M	223.12 M	219.79 M

For the lightweight decoder we use torch autocast, amplifying the training by autocasting images to float16. Due to numerical stability restrictions, this can not be applied to the MA-Net decoder based architectures, slowing down training. Due to accumulating memory in the default Mediar backbone, a custom image load function is used to enable pretraining on nodes with limited RAM. Batched images are preprocessed and augmented runtime in the dataloader, which slows down training to a certain extend, but is far superior in providing diverse training images to prevent overfitting. Because of runtime augmentation, we do not precompute ground truth flows due to the spatial and intensity transformation invariance of the flow generation transform. The preprocessing dataloading pipeline is summerized in table 4.4.

Note, that ground truth have to be spatially augmented the same way as train images. Therefore, image and label keys are handled with the same spatial augmentation parameters per iteration.

4.2 Model Finetuning

In this section, we take the pretrained models and finetune them on specific target datasets, as presented in **??**. In general, the aim of finetuning is to optimize the model weights themself and the image processing pipeline around the model to maximize accuracy for inference on specific datasets. When working in a very target oriented environment, it is often common to not only train the model weights on the specific datasets, but also edit the pipeline around the model. For example, if there is knowledge about physical constraints concerning spatial cell properties or reflection coefficients, this dataset specific knowledge can be directly incorporated into the pre- and postprocessing. For instance, known reflection coefficients can be refered to a certain range in pixel itensities after image aquisition. A subsequent histogram operation based on known intensities can be a powerful tool, to get rid of background cells. Spatial cell properties can be directly formulated as constraints in feature space. If there is a known min/max diameter, area or any numerical

Table 4.4: Overview of runtime batch loading and their implementation details.

Strategy	Details
(P) CustomLoadImage (1.0)	Loads images and labels without accumulating memory.
(P) CustomNormalizeImaged (1.0)	Memory-safe percentile norm. Rescales intensities to [0, 255] using robust percentiles ([0.0, 99.5]).
(P) EnsureChannelFirstd (1.0)	Ensures channel-first layout (C, H, W) for all inputs.
(P) RemoveRepeatedChanneld (1.0)	Collapses repeated channels in label map (`repeats=3`), useful for RGB masks.
(P) ScaleIntensityd (1.0)	Scales intensity values to [0, 1].
(S) RandZoomd (0.5)	Random zoom: factor [0.25, 1.5]. Area (image) / Nearest (label) interpolation. Output is not resized.
(S) SpatialPadd (1.0)	Zero-pads to a minimum spatial size of 512×512.
(S) RandSpatialCropd (1.0)	Random crop of fixed ROI size 512×512.
(S) RandAxisFlipd (0.5)	Randomly flips along spatial axes.
(S) RandRotate90d (0.5)	Random rotation by multiples of 90° over axes (0,1).
(I) IntensityDiversification (1.0)	Randomly rescales intensity for $\approx$ 40% of selected cells (factors [0, 0.7]).
(I) RandGaussianNoised (0.25)	Adds Gaussian noise ($\mu = 0$, $\sigma = 0.1$).
(I) RandAdjustContrastd (0.25)	Gamma correction: γ randomly sampled in [1, 2].
(I) RandGaussianSmoothd (0.25)	Gaussian smoothing: σ along x-axis randomly sampled in [1, 2].
(I) RandHistogramShiftd (0.25)	Randomly shifts histogram using 3 control points.
(I) RandGaussianSharpend (0.25)	Random sharpening applied.
(O) EnsureTyped (1.0)	Converts input data to PyTorch tensor format.

*(P): Pre-processing (S): Spatial Augmentation (I): Intensity Augmentation (O): Others
*The probability of applying the strategy is given in parentheses

boundary for aspect ratios, extents, solidities or essentricities, one could already eliminate most false positives. This can change the inductive bias to a large scale, making the model lose it's ability to generalize well over different modalities.

4.2.1 Finetuning on Fully Annotated Data

First, we want to finetune the models on fully annotated data. We will try to

The loss function does not have to be edited in such a way, that only annotated regions contribute to losses. So on a high level, the pretraining pipeline can be reused.

4.2.2 Finetuning on Partially Annotated Data

4.3 Model Inference

4.3.1 Testing on synthetic Data

4.3.2 Testing on BlasoSPIM

4.3.3 Testing on CTC Data

4.3.4 Testing on Zebrafish Data

4.4 3D Feature Network

4.4.1 Pretraining

4.4.2 Finetuning

Bibliography

[1] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel, "Handwritten digit recognition with a back-propagation network," in *Advances in Neural Information Processing Systems* (D. Touretzky, ed.), vol. 2, Morgan-Kaufmann, 1989.

A Erstes Anhang-Kapitel

Tja, hier schreibst du halt deinen Anhang rein. Bei der Gelegenheit: Falls du mal Python-Code schreiben musst, dafür gibt es lustige Funktionen, die ich in der thesis.tex definiert habe. Zum Beispiel für `in line code ()` und auch für ein ganzes Listing:

Algorithm 4 3D Post-processing and Mask Reconstruction

Input: Flow field $\mathbf{dP} \in \mathbb{R}^{3\times L_z\times L_y\times L_x}$, probability map $\mathbf{P} \in [0,1]^{L_z\times L_y\times L_x}$, thresholds τ_p, τ_s, iterations T
Output: Segmentation volume M
Identify foreground voxels: $\Omega = \{(z,y,x) \mid \mathbf{P}(z,y,x) > \tau_p\}$
for each voxel $(z,y,x) \in \Omega$ **do**
    Initialize $\mathbf{p}_0 \leftarrow$ normalized position of (z,y,x)
    for $t = 0$ to $T-1$ **do**
        Update position: $\mathbf{p}_{t+1} \leftarrow \text{clip}(\mathbf{p}_t + \mathbf{dP}(\mathbf{p}_t), -1, 1)$
    end for
    Store final position $\mathbf{p}_T$
end for
Initialize histogram $H(\mathbf{u}) = 0$
for each voxel i with final position $\mathbf{p}_T^{(i)}$ **do**
    Increment histogram: $H(\mathbf{p}_T^{(i)}) \leftarrow H(\mathbf{p}_T^{(i)}) + 1$
end for
Detect local maxima in H using nD max pooling (kernel size = 5)
Retain peaks with counts above threshold as seeds
for each seed s **do**
    Initialize seed mask in $11 \times 11 \times 11$ neighborhood
    Expand by morphological max pooling under constraint $H(\mathbf{u}) > 2$
end for
for each voxel $i \in \Omega$ **do**
    Assign voxel to label m of corresponding seed region
end for
for each mask **do**
    if mask size $> \tau_s \cdot$ image volume **then**
        Discard mask
    else if mask size $<$ min_size **then**
        Remove and fill mask
    else if flow-consistency is low **then**
        Prune mask
    end if
end for
Renumber masks to enforce consecutive labels $\{1, \ldots, N\}$
return M