KU LEUVEN

FACULTY OF BIOSCIENCE ENGINEERING

Beyond Convolution: Assessing Supervised Augmentation Techniques with Transformer versus Convolutional Neural Networks for Muscle Cell Segmentation

Voorbij Convolutie: Evaluatie van Gesuperviseerde Augmentatietechnieken met Transformer versus Convolutionele Neurale Netwerken voor Spiercel Segmentatie

Promotor: Prof. Jean-Marie Aerts
Begeleider: MSc. Panason Manorost
Departement Biosystemen
Afdeling Dier en Mens (A2H)

Masterproef voorgedragen
tot het behalen van het diploma van
Master of Science in de
bio-ingenieurswetenschappen:
Biosysteemtechniek, Major HHE

Lienert De Maeyer

2023 – 2024

Acknowledgment

I would like to thank Panason Manorost for his guidance and the opportunity to work on this project and I would like to thank Prof. Jean-Marie Aerts for his suggestions.

Abstract

In recent years, deep learning models have significantly advanced the analysis of microscopy images by shifting from manual feature extraction to machine learning methods that rely on data and computational power. This master thesis explores CNNs, like Attention U-Net and Mask R-CNN, and transformer-based networks, such as SegFormer, Swin-UNet, and TransResUnet, for cell segmentation in microscopy images. We will compare CNNs, known for spatial invariance and local connectivity, with transformers that capture long-range dependencies. Key evaluation metrics such as Intersection over Union (IoU), Sørensen-Dice coefficient, Average Precision (AP), and Precision-Recall (PR) curves are discussed, alongside a detailed visual analysis of binary output masks. Results show CNNs excel in detailed feature detection, while transformers offer superior generalization with larger datasets. The thesis considers practical aspects like output complexity and optimization impacts which show that transformer-based models have a significant potential to enhance traditional CNNs. This thesis aims to show that Transformer-based networks with smart attention mechanisms can effectively handle live-cell microscopy data, potentially outperforming traditional CNNs. Full code is available at ***KUL Onedrive Sharepoint****.*

Deep learning-modellen hebben in de afgelopen jaren de analyse van microscopiebeelden aanzienlijk verbeterd door over te schakelen van handmatig extraheren van kenmerken naar methoden in machine-learning, die steunen op uitgebreide datasets en computationeel vermogen. Deze masterproef exploreert de capaciteiten van Convolutionele Neurale Netwerken (CNN's), zoals Attention U-Net en Mask R-CNN, en Transformer netwerken, zoals Seg-Former, Swin-UNet en TransResUnet, voor het segmenteren van cellen in microscopiebeelden. We zullen CNN's, die bekend staan om hun ruimtelijke invariantie en lokale connectiviteit, vergelijken met Transformers, die afhankelijkheden over lange afstanden in het model mogelijk maken. Belangrijke evaluatieparameters zoals Intersection over Union (IoU), Sørensen-Dice coëfficiënt, Average Precision (AP) en Precision-Recall (PR) curves worden besproken, naast een gedetailleerde visuele analyse van binaire output masks. Uit de resultaten blijkt dat CNN's uitblinken in een gedetailleerde detectie van kenmerken, terwijl Transformators een superieure generalisatie bieden op grotere datasets. Deze masterproef heeft als doel om aan te tonen dat Transformer netwerken, met behulp van hun slimme attentiemechanismen, microscopie data van levende cellen op een effectieve manier kunnen analyseren, en mogelijks traditionele CNN's kunnen overtreffen. De volledige code is beschikbaar op ***KUL Onedrive Sharepoint****.*

Acronyms

AG Attention Gate
AP Average Precision
BCE Binary Cross-Entropy
BBOX Bounding Box
CLAHE Contrast Limited Adaptive Histogram Equalization
CNN Convolutional Neural Network
CV Computer Vision
DL Deep Learning
GeLu Gaussian Error Linear Unit
IoU Intersection over Union
IS Instance Segmentation
LN Layer Normalization
MLP Multilayer Perceptron
MSA Multi-Head Self Attention
NLP Natural Language Processing
PNG Portable Network Graphic
PR Precision-Recall
PS Panoptic Segmentation
ReLu Rectified Linear Unit
RGB Red, Green, Blue
RoI Region of Interest
SAM Segment Anything Model
SGD Stochastic Gradient Descent
VGG Visual Geometry Group
SS Semantic Segmentation
TIF Tagged Image File Format
ViT Vision Transformer

Contents

Chapter 1

Introduction

In recent years, deep learning models have significantly facilitated the analysis of microscopy images, by offering new tools for cell segmentation. Convolutional neural networks (CNNs), like Attention U-Net and Mask R-CNN, have been widely adopted in the field and demonstrate strong performance in segmenting diverse cell types. The emergence of transformer-based networks, like SegFormer, currently suggests a potential shift in the field of cell segmentation. Transformer-based networks are challenging the dominance of convolutional neural networks with their ability to capture long-range dependencies and complex patterns in image data. In this thesis, we aim to conduct a comparative analysis between transformer-based and convolutional architectures in cell segmentation tasks. Furthermore, we will explore semi-supervised learning as a means to enhance model training with limited annotated samples. Through this approach, we aim to evaluate whether semi-supervised techniques can effectively leverage unlabeled data and by this, potentially reduce the gap between model performance and the exhaustive requirements for labeled datasets in medical image analysis.

1.1 Basic Terminology in Digital Imaging

1.1.1 Bit Depth

Bit depth is a term that refers to the number of bits used to indicate the color of a single pixel in a digital image. In 8-bit images, for example, each pixel can represent 256 different intensity levels (2^8). 8-bit images are typically suitable for standard displays and photographic requirements. 16-bit images, however, offer a significantly higher range with 65,536 possible

intensity levels (2^{16}), which allows for much finer distinctions in intensity.

1.1.2 Image Channels

The term image channels refers to the components of color information in a digital image. In a standard RGB (Red, Green, Blue) image, there are three channels, each representing intensity values for red, green, and blue. This model allows for a wide range of colors through the combination of these primary colors at varying intensities. However, in grayscale images, there is only one channel that measures intensity from black to white, with no color information. This single-channel format is typically used in applications where color information is not necessary, such as in many forms of medical and scientific imaging, where emphasis is on light intensity or patterns rather than on color.

1.1.3 Resolution

Resolution refers to the number of pixels in each dimension of a digital image, for example, 1920x1080 pixels. Higher resolution images contain more pixels, which improves detail and clarity in the image. In scientific imaging, high resolution is recommended to capture subtle differences and small structures in the image.

1.2 Basic Terminology in Image Segmentation

1.3 Image Segmentation

Image segmentation is a process in computer vision that involves partitioning an image into multiple segments in order to simplify the representation of an image into something more meaningful and easier to analyze [1]. Image segmentation can be categorized into three main types: semantic segmentation, instance segmentation, and panoptic segmentation, as visualized in Figure D.3.

1.3.1 Semantic Segmentation

Semantic segmentation refers to the process in which each pixel is classified into one of the predetermined classes, which results in a set of binary masks $\{m_i\}_{i=1}^{G}$, where m_i is the binary mask for class i and G is the total number of classes. For an image $I \in \mathbb{R}^{H \times W \times 3}$, this

equates to generating a pixel-wise map $S = \{y_j\}_{j=1}^{H \times W}$ where y_j is the class label of pixel j. Each class has a unique mask m_i, such that for each pixel j, $m_{i,j} = 1$ if pixel j belongs to class i, otherwise $m_{i,j} = 0$. These masks are non-overlapping and cover every pixel in the image (as shown in Figure D.3b), delineating each class with a distinct color.

1.3.2 Instance Segmentation

Instance segmentation not only labels each pixel by class but also distinguishes between different instances of the same class. Its output is a collection of masks $\{m_i^k\}$, with m_i^k designating the k-th instance of class i. This means that each instance is given a unique contour in the segmentation. Instance masks for the same class can overlap, which reflects the distinct objects in the image. In Figure D.3c, different contours are used to differentiate between instances. The background typically remains unlabeled to highlight the objects themselves.

1.3.3 Panoptic Segmentation

Panoptic segmentation combines both the identification of classes and individual instances. Each pixel j is associated with a class label c_j and an instance identifier k_j, which results in a set of tuples $\{(\kappa_j, c_j)\}_{j=1}^{H \times W}$. Here, ”things” are countable objects given unique instance identifiers, while ”stuff” refers to amorphous regions like the sky or grass, which do not have separate instances and thus can be assigned a default identifier (commonly zero). Figure D.3d shows panoptic segmentation in which each color represents a class and shades of the same color represent different instances within that class, providing a rich, unified representation of the scene.

1.3.4 Binary Masks

Binary masks are 2D arrays that are used in cellular image segmentation to represent the presence or absence of objects in each pixel, typically assigning pixel values as 0 (background/black) and 255 (foreground/white), although some models may expect pixel values to be 0, 1, 2, etc., which makes the visualization more challenging. Some models, like Segformer, offer additional parameters to refine the training process. For example, the `ignore_index` parameter, which allows the model to disregard a specific pixel value during training. Another example is the `reduce_labels` parameter, which reduces the label of

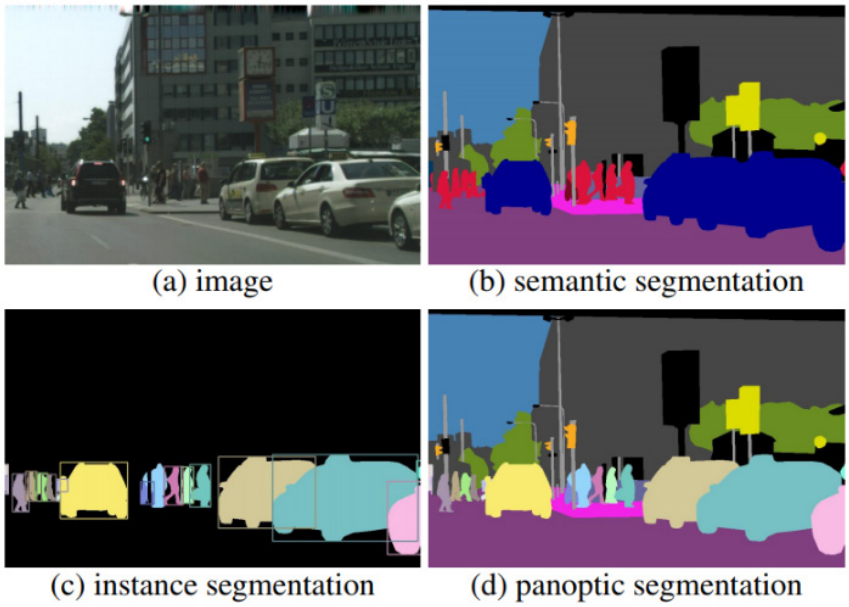


Figure 1.1: Comparison of semantic, instance, and panoptic segmentation [1].

each category by 1 and hereby enables the exclusion of certain classes from the training process, ensuring the model focuses on relevant cellular structures. For example, when 0 becomes 255, the `ignore_index` is often set to 255 by default, resulting in a label mapping change, where 1 becomes 0, 2 becomes 1, and so on, allowing for more effective training and segmentation of cellular images.

1.4 Motivation for Cell Segmentation in Biomedical Imaging

Cell segmentation is commonly used in medical research and diagnostics to analyze cellular structures and can be done using various imaging techniques. This process is important in order to obtain an accurate diagnosis of diseases and monitor the effectiveness of treatments, for example. Improvements in image processing have significantly enhanced the detection of medical conditions like cancer, which also influenced cancer treatment strategies [2]. Automated segmentation and tracking techniques are particularly useful for research on cellular behaviors, and can, for example, be crucial for understanding how cells respond to treatments and environmental changes [3]. Cell segmentation allows us to define cell boundaries in images, which, in turn, facilitates the analysis and modeling of subcellular patterns, the tracking of cell migration, and the exploration of morphological changes in response to perturbations [4; 5]. It is however crucial to have good accuracy in cell segmentation, as

errors can introduce biases that could compromise the reliability of subsequent analyses. Recent advancements in automated cell segmentation, especially through deep convolutional neural networks, have transformed this research field, enabling the high-throughput analysis of extensive image datasets. For example, in the DeepSea project, it was determined that despite their short G1 phase, embryonic stem cells exhibit cell size control during the G1 phase of the cell cycle [6].

1.4.1 Classical Approaches vs DL Approaches

In cell segmentation for microscopy analysis, automatically outlining objects like cells or their components is crucial for later analyzing cellular traits such as morphology, staining intensity, and spatial relationships. Both traditional computer vision (CV) and machine learning (ML) methods, along with newer deep learning (DL) techniques, have improved the accuracy of these segmentations. Deep learning methods are effective but often require large annotated datasets and can be difficult to interpret. On the other hand, classical approaches using traditional image processing and non-deep ML are generally more straightforward, needing less data and offering clearer interpretations.

1.4.2 Fine-Tuning Deep Learning Models

Implementing new deep learning models comes with a range of computational challenges, including complex software installations and managing dependencies often through command-line interfaces. These models are usually designed for specific types of data and require substantial fine-tuning for optimal performance on different datasets, a process that can divert time from core scientific research. The user experience with these tools can vary dramatically; some are equipped with extensive documentation and user-friendly libraries, while others offer the user minimal support.

Figure A.1 illustrates the detailed steps needed to adapt deep learning models for cell imaging, especially when dealing with new, unseen datasets that are out-of-distribution (OOD). This is initially shown in Figure A.1(a), where the new samples may significantly differ from the training data, resulting in poor segmentation accuracy. To address these differences, which can be as varied as cell staining differences or different imaging modalities, a style transfer technique is used to adapt the model to recognize and process images that look different

but share similar content with the training set. This adaptive process is further developed by annotating new training data that reflects the full range of the dataset's characteristics, as depicted in Figure A.1(b). This annotated data is then used to fine-tune the model, substantially improving its ability to accurately segment cells across various data types, as demonstrated in Figure A.1(c).

1.4.3 Software Platforms in Computer Vision

The computer vision (CV) field benefits from a variety of tools that are continuously enhanced through plugins and integrations, making them more versatile and user-friendly. Platforms like Fiji/ImageJ, CellProfiler, and Napari are popular due to their growing ecosystems which support a broad range of functionalities. For example, Fiji plugins like General Image Analysis of Nuclei-based Images and LABKIT are designed for specific tasks such as segmentation in 3D microscopy images and handling large, multiterabyte datasets, respectively. StarDist stands out for its detailed documentation and cross-platform availability, facilitating segmentation of star-convex shapes in graphical tools such as Fiji/ImageJ, Napari, and QuPath. Despite its popularity, StarDist's utility is limited when dealing with non-star-convex shapes, such as irregularly shaped cells or neurons, which are common in complex biological structures.

1.4.4 Generalist vs Specialist Model

The ultimate goal is to develop a generalist model that offers superior accuracy across various imaging modalities without relying on parameters. Such a model would ideally remove the need for manual annotations or fine-tuning and would be, on top of that, user-friendly and adaptable. While significant advancements are still needed, there have been notable developments in this area. For example, Cellpose 2.0, which is a model that requires fine-tuning, often shows a decrease in performance when it is applied to data outside its training set. In contrast, the top model from the MMCSC surpassed the performance of both the generalist models Cellpose and Omnipose, and a fine-tuned version of Cellpose 2.0, particularly when tested on unique biological images from an experimental source, showing its robust generalization capability. Additionally, Meta AI Research's Segment Anything Model (SAM) represents a foundation model for segmentation and is trained on an extensive dataset of 1.1 billion labels and 11 million images, including a few microscopy images. This led to the

development of Segment Anything for Microscopy, which is a specialized version of SAM that is fine-tuned for microscopy imagery.

1.5 Inter-Class and Intra-Class Imbalance in Cell Segmentation

In cell segmentation, effective model training must consider both inter-class and intra-class imbalances. Inter-class imbalances arise when there is a disproportionate number of images for different cell types in a dataset. For example, if a dataset contains a large number of neuron images but only a few glial cell images, the model may become adept at recognizing neurons but perform poorly on the less represented glial cells, resulting in a high rate of false negatives for glial cells.

Intra-class imbalance occurs within images of the same cell type that exhibit significant variability, such as differences due to cell stages or clustering. For example, in a dataset predominantly featuring clustered mitotic cells, the model may become adept at identifying these clusters while struggling to recognize isolated cells, often mistaking them for debris and leading to an increased false negative rate for these solitary cells. This type of imbalance within the same class can lead to a model that lacks uniform accuracy across different cell presentations. In the context of this thesis, which focuses on datasets containing a single cell type, the primary concern is the intra-class imbalance.

Chapter 2

CNN for Segmentation

2.1 Inductive biases in CNNs

The term inductive biases in machine learning refers to a set of assumptions about the data structure and nature, that guide the learning process and aid the generalization from training to unseen data. In image processing with deep learning, image-specific inductive biases accommodate the unique characteristics of image data and help the models in addressing certain challenges such as spatial hierarchies and variations in lighting, orientation, or scale. Convolutional Neural Networks (CNNs) are equipped with in-built inductive biases well-suited for image data, including:

2.1.1 Spatial Invariance

CNNs can recognize objects regardless of their position in the image. This is achieved through convolutional filters that the CNN models apply across the image.

2.1.1.1 Shift Invariance

Convolutional Neural Networks (CNNs) are characterized by shift invariance, meaning their performance remains unchanged by the location of objects within an input image. This concept is illustrated in Figure B.1, where a function $f : \mathbb{R}^d \to \mathbb{R}$ represents a 'cat detector'. Given an input image x, the function results in an output $f(x)$, which returns 1 if a cat is present. Even when x is shifted by a vector v using the operation S_v, resulting in $S_v x$, the function's output remains consistent and indicates a reliable detection regardless of object position.

Max pooling in Convolutional Neural Networks, as shown in Figure B.2, enhances the network's robustness to small local translations within the receptive field of the pooling regions. In max pooling, the maximum value is selected in each region, which effectively diminishes the effect of positional shifts and selectively down samples the feature maps from the convolutional layer. This introduces an 'approximate' invariance, allowing consistent feature detection as elements migrate within the image. Max pooling is non-linear in nature, which provides a special form of invariance that is especially beneficial for real-world applications where exact positional accuracy of objects is less critical than their detection.

2.1.1.2 Shift Equivariance

Shift equivariance in the context of image processing, in particular with CNNs, refers to the property where the output of an operation shifts in a manner that corresponds directly to a shift in the input. For a 'Cat segmentor' function $f : \mathbb{R}^d \to \mathbb{R}^d$, which assigns a value of 1 to pixels that are part of a cat and 0 otherwise, shift equivariance ensures that when the input image x is shifted by a vector v using a shift operator $S_v : \mathbb{R}^d \to \mathbb{R}^d$, the output $f(x)$ will shift in the same way, as shown in Figure B.1. Mathematically, if $f(x)$ is our original segmentation and $f(S_v(x))$ is the segmentation after the image has been shifted, for the function f to be shift-equivariant, it must satisfy equation 2.1. This means that the segmentation produced by f after the image is shifted is the same as shifting the original segmentation by the same vector v. It is crucial for tasks like semantic segmentation, where the location of each pixel relative to the object it represents must remain consistent after transformations like translation. This property ensures that the model's understanding of the object's structure and position in space is maintained even when the object's position in the input data changes.

Figure B.3 provides an example of shift equivariance in convolutional operations. When a convolutional layer with a filter is applied to an input image, the resulting feature map displays shift equivariance if it moves correspondingly with a shift in the input. Specifically, convolving the digit '3' with a filter generates a feature map, and shifting the '3' results in an analogous shift in the feature map, indicating that the convolution preserves the spatial relationship. The Fourier Transform provides a mathematical basis for understanding this phenomenon. The convolution $f(x) * g(x)$, where $g(x)$ is the filter, becomes a multiplication of their Fourier (see equation 2.2). A spatial shift in the image leads to a phase shift in the frequency domain

but leaves the magnitude of the Fourier Transform unchanged (see equation 2.3). Applying the inverse Fourier Transform yields the spatially shifted feature map (see equation 2.4). Consequently, this principle ensures that CNNs can reliably detect and analyze features in an image, irrespective of their spatial positioning.

$$f(S_v(x)) = S_v(f(x)) \tag{2.1}$$

$$\mathcal{F}\{f(x) * g(x)\} = \mathcal{F}\{f(x)\} \cdot \mathcal{F}\{g(x)\} \tag{2.2}$$

$$\mathcal{F}\{f(x - v) * g(x)\} = e^{-i\omega v} \cdot \mathcal{F}\{f(x)\} \cdot \mathcal{F}\{g(x)\} \tag{2.3}$$

$$\mathcal{F}^{-1}\{e^{-i\omega v} \cdot \mathcal{F}\{f(x)\} \cdot \mathcal{F}\{g(x)\}\} = f(x - v) * g(x) \tag{2.4}$$

2.1.2 Local Connectivity

CNNs analyze images by examining pixel groups that are close together, as these pixels usually belong to the same object. Small filters within the CNN move across the image and identify simple patterns like edges and areas of color. This approach makes use of the way images are naturally made up of smaller, simple patterns that form complex structures. CNNs have a built-in bias for object recognition, which makes them able to pick out objects from their surroundings without any training. This is shown in Figure B.4, where the yellow patches correspond to the image areas that the CNN identifies as objects, such as a cluster of horses or a boat, which illustrates that CNNs have an innate ability to localize multiple objects in diverse contexts. Cao and Wu [7] take a closer look at this quality of CNNs. In their study, this quality is used to improve the way CNNs are trained in self-supervised learning settings by teaching them to focus on the main object while altering the background. This technique not only helps with object detection but also works well across different training dataset sizes and under a variety of image modifications.

2.1.3 Spatial Hierarchy

Figure B.5 illustrates the inductive bias of spatial hierarchy in CNNs, demonstrating how they learn to understand images by constructing a hierarchy of features. At the lowest level, the CNN detects simple, foundational elements like edges and colors. At the mid-level, the CNN combines these elements into more complex shapes and patterns that begin to resemble parts of objects. Finally, at the highest level, the network integrates these components into abstract representations that correspond to complete objects or scenes. This enables the CNN to recognize and interpret various visual elements within the image.

2.2 Convolution-based Model Architectures

2.2.1 U-net

The U-Net model has undergone various adaptations to enhance its performance in diverse segmentation tasks, as illustrated in Figure B.6. U-Net-related publications have significantly increased between 2015 and 2022 (from about 100 to over 3000 publications, see Figure B.7), which illustrates its widespread application and widening relevance in medical image analysis. While segmentation remains its core use, the U-Net's auto-encoder structure has also been effectively used in other areas, such as image synthesis, denoising, reconstruction, and super-resolution. Azad et al. [8] categorized the various U-Net models into specific groups such as Backbone Design Enhancement and Skip Connection Enhancement.

U-Net Architecture

The U-Net architecture is consists of an encoder (contraction path) and a decoder (expansion path), see Figure B.10. The encoder begins with an input of 572x572 pixels and employs repeated pairs of 3x3 convolutions followed by ReLU activations and 2x2 max pooling with stride 2, successively doubling the feature channels from 64 to 512 while reducing spatial dimensions. This process ensures the extraction of increasingly abstract features at various resolutions. In contrast, the decoder path up-samples the feature maps via 2x2 convolutions, reducing channel depth while increasing spatial resolution, and merges them with matching feature maps from the encoder through concatenation. It hereby retains spatial details with high resolution which is crucial for precise segmentation. The process results in two 3x3 convolutions with ReLU activations and a final 1x1 convolution that classifies each pixel into

one of the desired categories. Altogether, U-Net comprises 23 convolutional layers within its architecture. A closer look at the convolutional layers shows how filter dimensions expand to accommodate the increasing channel depth. The details of this expansion through the convolutional operations are illustrated below.

In the encoder's first convolutional block (`self.conv1`), the input image undergoes processing by 64 filters of dimensions $3 \times 3 \times 1$, matching the single-channel input. This step produces 64 feature maps, which, after activation by a ReLU function, retain a channel depth of 64. The following convolution within the same block (`self.conv2`) then applies a further set of 64 filters, each with dimensions of $3 \times 3 \times 64$, to the previously obtained 64-channel feature maps. The output remains at 64 channels, setting the stage for the max pooling layer. In CNNs, the dimensions of the output feature map are influenced by the size of the input, the filter dimensions, the stride, and the padding. U-Net employs padding—often zero-padding—to maintain the spatial dimensions of the input throughout the convolutional layers. This approach allows the filters to cover the edge regions of the input, thereby maintaining the original input size in the output. Specifically, the formula for the output size including padding and stride is given by $\left\lfloor \frac{\text{Input size} - \text{Filter size} + 2 \times \text{Padding}}{\text{Stride}} \right\rfloor + 1$. For an input of 512×512 pixels, a filter size of 3×3, padding of 1, and a stride of 1, the output size is calculated in equation 2.5. Thus, with a stride of 1 and padding that compensates for the filter size, the U-Net architecture ensures that the spatial dimensions of the feature map remain constant at 512×512 after each convolutional layer.

$$\text{Output size} = \left\lfloor \frac{512 - 3 + 2 \times 1}{1} \right\rfloor + 1 = 512 \tag{2.5}$$

Motivation for Attention Gates

In a CNN, the receptive field of a neuron indicates the region of the input image it processes. Initial layers typically have smaller receptive fields, such as 3x3 pixels. As the network down samples the image through operations like max pooling or strided convolutions, the size of the feature map reduces, which makes each pixel in the down sampled image represent a larger area of the original image. For instance, after a 2x2 max pooling of a 256x256 image to 128x128, a 3x3 receptive field at a deeper layer will cover a 6x6 pixel area of the original image. This expanded receptive field allows neurons to capture more global and abstract features, which is beneficial for tasks like image classification that require a broader

understanding of the image. However, this can be disadvantageous for tasks that require high precision in localization, such as object detection or pixel-wise segmentation, where maintaining detail is crucial.

Hard, Soft and Self-Attention

Hard, soft, and self-attention mechanisms offer diverse strategies to focus on relevant features in the data. Hard attention, which includes techniques like iterative region proposal and cropping, is non-differentiable and often relies on reinforcement learning for training, which complicates the training process and limits the model's use in scenarios not suited for gradient-based optimization. In contrast, soft attention is probabilistic and differentiable, and integrates seamlessly with standard back-propagation methods. This makes it highly effective for tasks like sentence-to-sentence translation and image classification, where it assigns adaptive weights to different parts of the data. An example is channel-wise attention in image classification, which significantly enhances model performance and was a standout in the ILSVRC 2017 challenge. As an extension of these concepts, self-attention removes the need for external gating and allows a model to autonomously focus on different parts of the input. This effectively captures long-range dependencies and improves response contextuality, which is advantageous in class-specific pooling for image classification and leads to more accurate and robust models.

Attention Gates (AG)

An example of additive (spatial) soft attention is the Attention Gate proposed by Oktay et al. [9]. Attention Gates (AGs) are utilized to focus the model selectively on regions of interest within the input feature maps. They are defined as follows:

$$q_{att}^{l} = \psi^{T}\left(\sigma_{1}\left(W_{x}^{T} x_{i}^{l} + W_{g}^{T} g_{i} + b_{g}\right)\right) + b_{\psi} \tag{2.6}$$

$$\alpha_{i}^{l} = \sigma_{2}\left(q_{att}\left(x_{i}^{l}, g_{i}; \Theta_{att}\right)\right) \tag{2.7}$$

where $\sigma_2(x_{i,c})$ corresponds to the sigmoid activation function. The gating vector $g_i \in \mathbb{R}^{F_g}$ incorporates contextual information from a coarser scale, and α_i^l represents the attention coefficients at each pixel i and layer l. The output of AGs is the element-wise multiplication of input feature-maps and attention coefficients: $\hat{x}_{i,c}^{l} = x_{i,c}^{l} \cdot \alpha_{i,c}^{l}$. The AG mechanism is

characterized by a set of learnable parameters Θ_{att}, which include the linear transformations $W_x \in \mathbb{R}^{F_l \times F_{int}}$, $W_g \in \mathbb{R}^{F_g \times F_{int}}$, and bias terms $b_\psi \in \mathbb{R}$, $b_g \in \mathbb{R}^{F_{int}}$. These transformations are computed using channel-wise 1x1 convolutions for the input tensors. The intermediate attention features x' and g are linearly mapped to an F_{int} dimensional intermediate space. This approach enables the network to focus on salient features in the input data and hereby enhance model accuracy and interpretability. An example of how Attention Gates (AGs) operate within a CNN can be found in Appendix B.

The standard U-Net model was enhanced by incorporating an attention gate at every skip connection, leading to the creation of the Attention U-Net (Figure B.9). Experimental observations indicated that this modification boosts the model's sensitivity to foreground pixels, which eliminated the need for complex heuristics.

2.2.2 Mask-R-CNN

Mask R-CNN is an extension of Faster R-CNN, which is based on the original R-CNN architecture.

2.2.2.1 R-CNN

R-CNN, which is the first model in a subsequent series, divides the image into 2000 regions of varying sizes using *selective search*. Selective search involves proposing regions using initial sub-segmentation, and then grouping similar regions using a greedy non-maximum suppression algorithm. Each region is then reshaped (warped) into a square to facilitate processing by a convolutional neural network (ConvNet). The ConvNet generates features, which are then classified using a support vector machine (SVM) to determine the object class (e.g., cat, dog, etc.). Additionally, a bounding box (bbox) regressor is used to predict the object's location.

2.2.3 Fast R-CNN

Fast R-CNN improves upon R-CNN by processing the entire image through a feature extractor, rather than individual regions [10]. The feature extractor generates a feature map, which is then used to compute features for each region proposal. The region proposals are generated using selective search, and then resized and fed into a fully connected (FC) layer,

followed by a linear regressor for bbox prediction and a linear + softmax layer for classification. This model introduces the use of RoIPooling to extract fixed-size feature maps from these variable-sized region proposals, ensuring a consistent input size for the subsequent layers.

2.2.3.1 Faster R-CNN

Faster R-CNN operates in two stages to enhance object detection efficiency [11]. The first stage utilizes a *Region Proposal Network* (RPN) that directly generates candidate object bounding boxes and replaces the selective search method used in prior models. In the second stage, building on Fast R-CNN techniques, it extracts features from each candidate box using RoIPool, followed by tasks for bounding box prediction and classification.

2.2.3.2 Mask R-CNN

Mask R-CNN builds upon Faster R-CNN, incorporating a two-stage process where the first stage uses the same Region Proposal Network (RPN) to predict region proposals [12]. The second stage of Mask R-CNN employs RoIAlign, as opposed to Faster R-CNN which employed RoIPool. This modification allows for more accurate localization of features, which is important for the task of instance segmentation. Next to predicting the class and bounding box, Mask R-CNN also outputs a binary mask for each Region of Interest (RoI), providing a mask for each instance in addition to the bounding box and class label.

RoIPool

Region of Interest (RoI) Pooling in Fast R-CNN transforms the varying sizes of RoIs into a consistent format for the network's fully connected layers [10]. An example, as illustrated by Kemal Erdem [13], shows a $512 \times 512 \times 3$ image processed through VGG16 resulting in a $16 \times 16 \times 512$ feature map. An RoI of 145×200 pixels scales down by a factor of 32 to 4.53×6.25, and is quantized to 4×6 due to grid alignment, as depicted in Figure 2.1. This quantization results in data loss (dark blue) and addition (light green), shown in Figure 2.2. The RoI is then segmented into a 3×3 grid for pooling, where the division $6/3$ and $4/3$ leads to a quantized 2×1 grid, losing a row represented by the cyan area. Max pooling across these sections yields a $3 \times 3 \times 512$ RoI matrix.

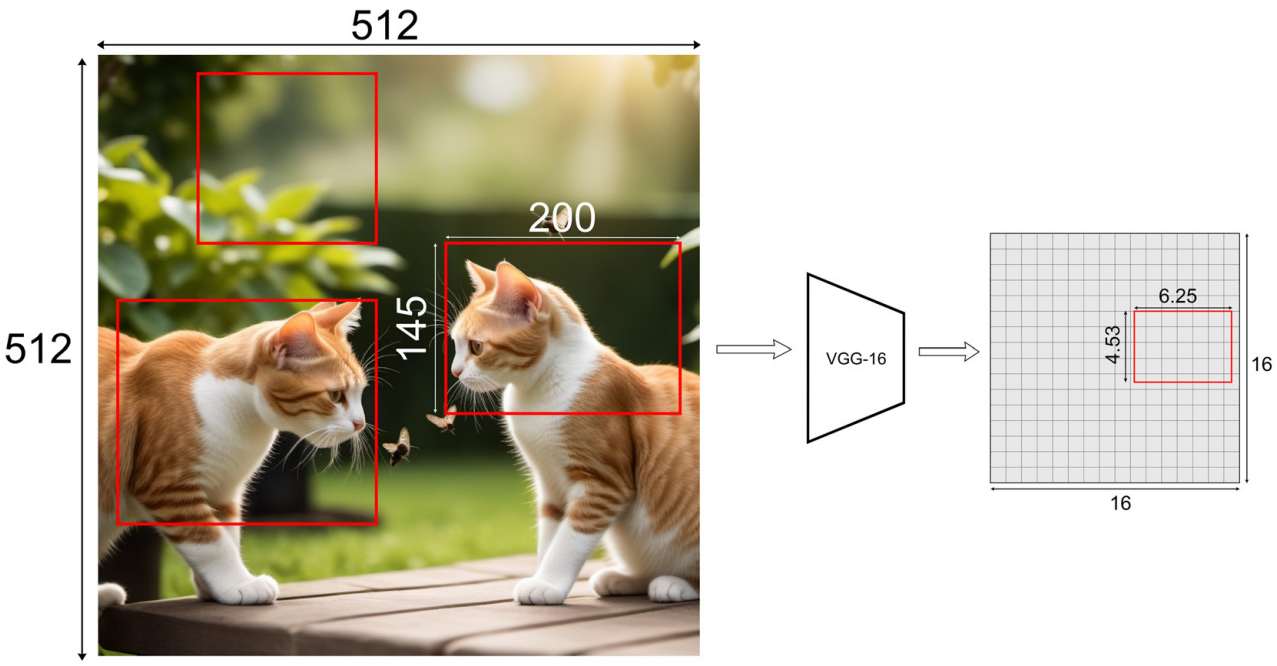


Figure 2.1: Illustration of Region of Interest (RoI) mapping in a Fast R-CNN architecture. An input image of size $512 \times 512 \times 3$ (width $\times$ height $\times$ RGB channels) is processed through VGG16, resulting in a $16 \times 16 \times 512$ feature map. This indicates a reduction by a factor of 32 in spatial dimensions and an expansion to 512 channels. An RoI of size 200×145 pixels scales down to 6.25×4.53 on the feature map, in line with the spatial reduction while retaining the channel depth for pooling. Image adapted from [13].

RoIAlign

Region of Interest (RoI) Align is an advanced method used in the Mask-RCNN model for both instance segmentation and object detection [12]. Unlike RoI Pooling, which may lead to data loss due to quantization, RoI Align maintains the exact spatial locations through bilinear interpolation. This process avoids the rough spatial quantization of RoI Pooling. Building upon our earlier examination of RoIPool, where we analyzed the sizes and configurations, in Kemal Erdem's examination of the RoIAlign process, the division of the mapped RoI into smaller boxes is similarly showcased [13]. In our case, the size of the pooling layer is 3×3. Given the mapped RoI dimensions of 6.25×4.53, each box would have the size of $\frac{6.25}{3} \times \frac{4.53}{3}$, yielding a box of approximately 2.08×1.51. Inside each box, four sampling points are calculated to gather data for the bilinear interpolation. The process involves computing the coordinates of the neighboring pixel values for each sample point within the boxes (see equation 2.8 for the calculation of the coordinates of the first sample point). Using these coordinates, bilinear interpolation is applied to obtain the pixel value at each sample point. Bilinear interpolation takes the closest integer pixel values around the point and performs a weighted average. The value for a sample point P is calculated in equation 2.9. In equation B.1, you can find

the filled-in values from our example. For the first sample point, the interpolated value is calculated to be approximately 1.4. This procedure is repeated for all sample points within each box, as demonstrated in Figure B.12. After interpolation, max pooling is applied over each pooling section, where the highest value from the four sample points per section is chosen, resulting in a final $3 \times 3 \times 512$ feature matrix for each RoI.

$$X = X_{\text{box}} + \left(\frac{\text{width}}{3}\right) \times 1 = 9.94 \qquad Y = Y_{\text{box}} + \left(\frac{\text{height}}{3}\right) \times 1 = 6.50 \tag{2.8}$$

$$P \approx \frac{y_2 - y}{y_2 - y_1}\left[\frac{x_2 - x}{x_2 - x_1}Q_{11} + \frac{x - x_1}{x_2 - x_1}Q_{21}\right] + \frac{y - y_1}{y_2 - y_1}\left[\frac{x_2 - x}{x_2 - x_1}Q_{12} + \frac{x - x_1}{x_2 - x_1}Q_{22}\right] \tag{2.9}$$

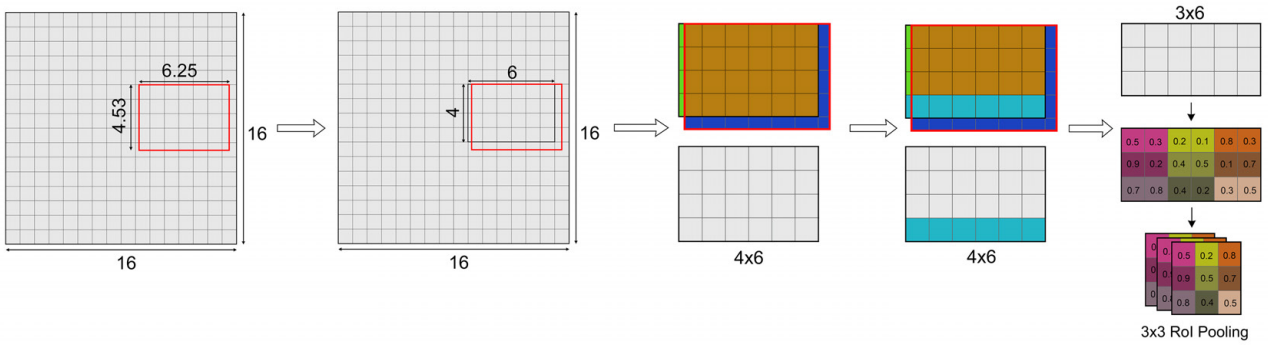


Figure 2.2: Overview of the Region of Interest (RoI) Pooling process featuring quantization effects and max pooling. An original RoI of 200×145 scales down to 6.25×4.53 on the 16×16 feature map, and is then quantized to 6×4, with data loss illustrated in dark blue and data addition in green. After resizing from 4×6 to 3×6, further data changes are depicted in aquamarine from the second quantization. The subsequent max pooling operation results in a $3 \times 3 \times 512$ matrix for each RoI, with 512 channels, by extracting the maximum value from each section of the feature map. This process transforms the feature space for the subsequent stages of classification and bounding box regression in object detection models. Image adapted from [13].

Chapter 3

Transformers for Segmentation

The Transformer architecture, introduced by Vaswani et al. in 2017 [14], has had a notable influence on sequence modeling and machine translation and has extended its impact across various NLP tasks. BERT [15] and RoBERTa [16], both introduced in 2019, leveraged deep bidirectional representations from unlabeled text and represented a shift towards models capable of understanding context. Similarly, GPT models [17] focused on unsupervised learning, with GPT-3 [18] notable for its extensive parameter count and versatility across tasks. The increasing references to Transformer models in the literature, as illustrated in Figure C.1, indicate their growing importance in NLP. The Vision Transformer (ViT) by Google Brain in 2020 [19] introduced transformer architectures to computer vision. ViT treats images as patch sequences for self-attention, which is a technique adapted from NLP. Subsequent models like DeiT and PVT evolved to refine ViT's approach and address its initial limitations. The integration of transformers into specialized areas is illustrated by TransUnet, developed by Chen et al. in 2021 [20], which combined transformer blocks with the U-Net architecture for medical image segmentation. The rise in ViT citations since 2020 indicates its growing influence in the computer vision community aswell (Figure C.1) .

3.1 Transformer architecture

The Transformer architecture is designed to handle sequential data without the need for recurrent networks. As illustrated in Figure C.2, the architecture is divided into two primary components: the encoder and the decoder.

3.1.1 Encoder and Decoder Structure

The encoder, which forms the left block of the architecture, is responsible for processing the input and learning rich representations. It consists of a stack of identical layers, each containing two sub-layers: a multi-head self-attention mechanism and a simple, position-wise fully connected feed-forward network. Positional encodings are added to the input embeddings to preserve information on sequence order, as these embeddings pass through each encoder layer. The decoder, on the right side and excluding the final linear and softmax layers, aims to decode the encoded representations and, in sequence-to-sequence tasks, combines them with additional input to predict the output. Similar to the encoder, the decoder is composed of a series of identical layers. However, it includes an additional sub-layer for cross-attention, where each layer attends to the output of the encoder stack.

3.1.2 Variants of Transformer Models

Three variants of Transformer models have emerged, each suited to different types of tasks: Encoder-only models, decoder-only models and encoder-decoder models. Encoder-Only Models, like BERT in NLP, focus on learning representations of the input data. They can be used for tasks such as classification where understanding the input is more important than generating new sequences. Decoder-Only Models, such as GPT-3, utilize only the decoder structure and are adapted for generative tasks where the emphasis is on producing new content based on learned data patterns. Encoder-Decoder Models, the full Transformer model, combines both encoder and decoder and excels in sequence-to-sequence tasks like machine translation, where the goal is to transform an input sequence into a new output sequence.

3.2 Vision Transformers

Directly applying self-attention to full-resolution images faces computational challenges, as the process scales poorly when the number of pixels is increased [19]. Previous efforts have been made to adapt Transformers for image processing—such as local self-attention by Parmar et al. (2018) [21], convolutional replacements proposed by Hu et al. (2019) [22] — which have encountered obstacles, especially in computational efficiency. Notably, the model by Cordonnier et al. [23], which applies self-attention to 2x2 pixel patches, closely

parallels the ViT approach but is limited to low-resolution images. In contrast, Dosovitskiy et al.'s Vision Transformer illustrates the efficacy of using transformers on medium-resolution images, significantly enhanced by large-scale pre-training, to exceed the performance of conventional CNNs without the need for complex modifications for compatibility with existing hardware accelerators.

3.2.1 Vision Transformer Architecture

The Vision Transformer (ViT) is an example of an encoder-only Transformer. It starts with embedded patches of an image, which are then processed through a stack of transformer blocks. These blocks consist of two primary components: the Multi-Head Attention Block, which facilitates interaction among embeddings to enhance the model's understanding of the image as a whole, and the Multi-Layer Perceptron (MLP), allowing individual analysis by each embedding based on the context provided by its neighbors. Both blocks use the majority of the compute resources. The incorporation of residual connections and layer normalization, each applied twice to ensure robustness in training, allows for optimization in the ViT architecture. Residual connections improve gradient flow across layers, which addresses the vanishing gradient problem common in DL models, whereas layer normalization stabilizes the learning process.

3.2.2 Generation of Patch Embedding $\mathbf{z}_1^i$

Vision Transformers process images by mimicking the tokenization step in natural language processing. An input image $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$ is divided into a set of non-overlapping fixed-size patches. This division into patches allows the model to observe the whole image, capturing the global context effectively. These patches are then transformed through a series of operations which include flattening, linear projection, and the addition of positional encodings. The outcome of these steps is a sequence of patch embeddings that are used as inputs for the transformer encoder.

3.2.2.1 Images to Patches

As shown in Figure 3.1, in Vision Transformers (ViTs), an input image $\mathbf{x} \in \mathbb{R}^{W \times H \times C}$ is segmented into a sequence of non-overlapping patches. Each patch is then flattened and linearly embedded. Additionally, positional encodings are added to the patch embeddings to

preserve positional information. The transformer encoder processes these embeddings, and utilizes self-attention mechanisms to contextualize each patch in relation to the entire image. This process enables the model to handle complex tasks like image classification by considering both local and global features. The total number of patches N is calculated as $N = \frac{H \times W}{P^2}$, where H is the height, W is the width of the input image, and C is the number of channels.

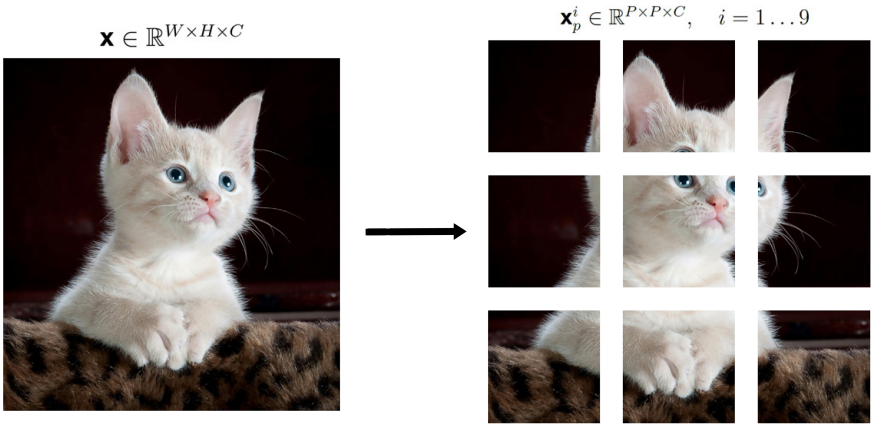


Figure 3.1: Image patching in Vision Transformer.

3.2.2.2 Linear Projection of Patches

After extracting patches from the original image, each patch $\mathbf{x}_p^i$, where $i = 1, \ldots, N$, undergoes a linear projection to a D-dimensional feature space (Figure 3.2). The mapping f takes each patch $\mathbf{x}_p^i$ to a feature vector $\mathbf{z}_1^i$ such that $f : \mathbf{x}_p^i \in \mathbb{R}^{P \times P \times C} \mapsto \mathbf{z}_1^i \in \mathbb{R}^{1 \times D}$. For instance, the first patch $\mathbf{x}_p^1$ is projected to $\mathbf{z}_1^1$, which consists of elements $z_1^{1,1}$ through $z_1^{1,D}$. This mapping, applied to all N patches, allows the transformer to process the image in a sequence, analogous to word processing in natural language applications. In practice this linear projection is done via MLP or convolution. The matrix Z_0 is constructed by stacking the class token $\mathbf{x}_{\text{class}}$ and the weighted patch embeddings. Each patch embedding $\mathbf{x}_p^i$ is premultiplied by the weight matrix W to obtain $\mathbf{z}_p^i \cdot W$, with W having dimensions $(P^2 \cdot C) \times D$. The positional embeddings matrix Z_{pos}, with dimensions $(N+1) \times D$, is then added to each row of Z_0. The resulting matrix is expressed as follows:

$$Z_0 = \begin{bmatrix} \mathbf{x}_{class} \\ \mathbf{x}_p^1 W \\ \mathbf{x}_p^2 W \\ \vdots \\ \mathbf{x}_p^N W \end{bmatrix} + \begin{bmatrix} Z_{pos}^0 \\ Z_{pos}^1 \\ Z_{pos}^2 \\ \vdots \\ Z_{pos}^N \end{bmatrix} = \begin{bmatrix} \mathbf{x}_{class} + Z_{pos}^0 \\ \mathbf{z}_1^1 + Z_{pos}^1 \\ \mathbf{z}_1^2 + Z_{pos}^2 \\ \vdots \\ \mathbf{z}_p^N + Z_{pos}^N \end{bmatrix},$$

where $\mathbf{x}_{class} \in \mathbb{R}^{1\times D}$, $\mathbf{x}_p^i \in \mathbb{R}^{1\times(P^2\cdot C)}$, $W \in \mathbb{R}^{(P^2\cdot C)\times D}$, $Z_{pos}^i \in \mathbb{R}^{1\times D}$, and $Z_0 \in \mathbb{R}^{(N+1)\times D}$.

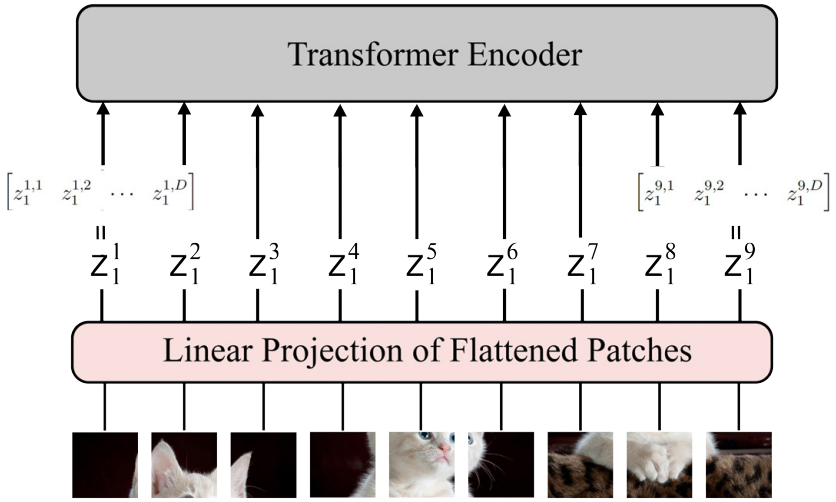


Figure 3.2: Linear Projection of Flattened Patches

3.2.2.2.1 MLP-based Linear Projection

The linear projection of each patch $\mathbf{x}_p^i$ for $i = 1, \ldots, N$ via the weight matrix W of the MLP layer, with no bias, is given by equation 3.1. Here, the weight matrix W of dimensions $\mathbb{R}^{P^2C\times D}$ is utilized to transform the reshaped patch vector $\mathbf{x}_p^i$ into the feature vector $\mathbf{z}_1^i$. This projection is performed for all patches, indexed by i, producing a set of feature vectors that represent the encoded information from the original image.

$$\begin{cases} \text{Reshape each patch:} & \mathbf{x}_p^i \in \mathbb{R}^{P\times P\times C} \to \mathbb{R}^{1\times P^2C}, \\ \text{Project onto feature space:} & \mathbf{z}_1^i = \mathbf{x}_p^i W, \end{cases} \tag{3.1}$$

3.2.2.2.2 Conv2-Based Linear Projection

An alternative to the MLP-based approach for linear projection of image patches is to employ a strided convolution operation. This method can offer efficiency benefits by exploiting the

inherent structure of the image data. The convolution operation for the i'th patch using a strided convolution is outlined as:

$$\begin{cases} \text{For the first element of the feature vector:} & \mathbf{z}_1^{i,1} = \mathbf{x}_p^i * W_1, \\ \text{For the last element of the feature vector:} & \mathbf{z}_1^{i,D} = \mathbf{x}_p^i * W_D, \\ \text{for} & i = 1, \ldots, N. \end{cases} \tag{3.2}$$

Here, $*$ denotes the convolution operation, W_1 through W_D represent the set of D convolutional filters, and $\mathbf{z}_1^{i,1}$ through $\mathbf{z}_1^{i,D}$ are the elements of the feature vector corresponding to the i'th patch. Each filter is applied with a stride of P, equal to the patch size, ensuring that the convolutional kernels process non-overlapping regions of the image, thereby projecting the patches linearly. The sequential processing of patches, from the first to the N'th, allows for the preservation of spatial hierarchies within the image and facilitates the parallelization of the projection step.

3.2.2.3 Learnable Positional embedding

The position embedding in ViTs serves to provide the model with information about where a patch is located within the original image, which is crucial since the flattening of patches into a sequence loses this spatial information. The positional encodings are typically learnable parameters that are optimized during training, unlike the fixed sinusoidal positional encodings used in some NLP models. They allow the model to better understand the arrangement and relationship between different parts of the image, which is important for tasks that depend on the spatial layout, such as object detection or scene understanding. Each patch embedding and the prepended class token are summed with their respective positional embeddings to encode positional information within the sequence, as depicted in Figure C.3.

3.2.2.4 Learnable Class embedding

Vision Transformers introduce a learnable class embedding, often referred to as the ”classification token” or simply ”class token.” This token is prepended to the sequence of patch embeddings, as illustrated in Figure C.3, and serves as a placeholder for aggregating information across the patches for the purpose of classification. Just like the patch embeddings, the class token has a corresponding position embedding associated with it. This position embedding is added to the class token to maintain positional information, although the class token itself does not correspond to any actual position in the input image. During the self-

attention process in the Transformer layers, the class token gathers global information from all patches, and by the end of the Transformer encoder, it serves as the aggregated representation of the entire image. After the self-attention and feed-forward processing within the Transformer encoder, the class token embedding is further refined for classification through an MLP (Multi-Layer Perceptron) head, as can be seen in Figure C.3. This MLP head consists of one or more dense layers and functions as the final classification layer, projecting the embedding into the output space of class labels. A softmax activation function is then applied to the MLP's output to generate a probability distribution across the possible classes.

3.2.2.5 Segmentation with Vision Transformers

Unlike classification, segmentation in Vision Transformers does not rely solely on the learnable class embedding. Instead, the focus shifts to utilizing the entire set of patch embeddings to understand the image at a granular level. The model processes embeddings by either up sampling them to the original image size or by applying additional convolutional layers. Following this, a softmax function is applied, which assigns class probabilities to each pixel or region.

3.2.2.6 Interpretation of Embeddings

The configuration of the embedding space in Vision Transformers corresponds to the visual similarity of patches: embeddings that are in close proximity represent patches with similar features. Additionally, specific directions in this space may signify semantic changes, such as transitions from day to night scenes or alterations in object sizes from small to large.

The goal of the self-attention process is to add contextual meaning to each embedding, whether it's a token or a patch in an image. For example, take the word "bank," which can refer to different concepts depending on the sentence. Initially, the embedding for "bank" is generic. However, as it passes through the transformer's layers, it absorbs context from adjacent words, refining its meaning to match the specific context of a river bank, a financial institution, or an airplane maneuver. Similarly, in Vision Transformers (ViTs), each image patch receives an initial embedding. Through the attention mechanism, these embeddings are updated by considering the broader context of the entire image. For example, a patch showing part of a building at sunset is initially just colors and shapes. But with context from the entire scene, it becomes part of a cityscape at a specific time of day.

The embedding matrix (W_E) is only one of several weight matrices. Other matrices include the Key (W_K), Query (W_Q), and Value (W_V) matrices (Table 3.1), which are used in the self-attention mechanism. These matrices help the model to focus on and understand different parts of the image by processing how each patch relates to the others. Following this, the Output Matrix (W_O) combines the attention-adjusted embeddings to form a cohesive understanding of the image. Other than these matrices, there are also weights associated with the positional encoding, FFN and the final Classification Layer.

Table 3.1: Overview of Weight Matrices in Vision Transformers (ViTs)

Matrix	**Purpose**
Embedding Matrix (W_E)	Embeds image patches into high-dimensional vectors.
Key Matrix (W_K)	Transforms embeddings into keys for attention.
Query Matrix (W_Q)	Transforms embeddings into queries for attention.
Value Matrix (W_V)	Transforms embeddings into values for attention.
Output Matrix (W_O)	Consolidates attention outputs into embeddings.
Positional Encoding Matrix (W_P)	Adds spatial information to embeddings.
Feed-Forward Network	Expands and compresses embeddings within layers.
Classification Layer	Maps aggregated features to class labels.

3.3 Context Size in Vision Transformers

In traditional text-based transformers, the network's context size, which is essentially the maximum number of tokens (words or subwords) it can process in one pass, limits its capacity to understand and generate language. This constraint directly impacts the model's ability to make predictions about the next word in a sequence, as it can only consider a fixed window of preceding tokens. For Vision Transformers (ViTs), the concept of context size translates differently due to the nature of visual data. Instead of words, ViTs deal with image patches, and the context size refers to the number of these patches the model processes simultaneously. This determines the portion of the image the model can ”see” and analyze at any given time. However, unlike in text transformers where the prediction of the next word is sequential and directly influenced by the preceding fixed number of tokens, ViTs generally do not predict the ”next patch” in a sequence. Instead, they aggregate the information from all patches within their context to make holistic predictions about the image—such as classifying it or identifying objects within it. The size of these patches directly impacts the

computational load: smaller patches mean more detail and higher computation and memory use, while larger patches reduce both, potentially overlooking some details.

3.3.1 Encoder in Vision Transformers

The encoder in Vision Transformers processes a sequence of patch embeddings to extract features from the input image. Each encoder layer, detailed in Figure C.3, performs a series of operations to refine these embeddings. Embedded patches enter the encoder and are first normalized. The normalized embeddings are then subjected to a multi-head self-attention (MSA) mechanism, which allows the model to weigh the importance of different patches relative to each other. The MSA output is combined with the original input through a residual connection, facilitating information flow and preventing the vanishing gradient problem. Equations 3.14 and 3.4 give an outline of these steps, where MSA denotes the multi-head self-attention function and LN denotes layer normalization. The subscript ℓ indicates the layer level, with $\ell = 1$ being the first encoder layer. The subsequent stage is the application of a multi-layer perceptron (MLP), which is a feed forward neural network with a single hidden layer employing the Gaussian Error Linear Unit (GELU) activation function. This network further processes the information, and the result is again added back to the initial input of this stage through another residual connection (see equations 3.5 and 3.6). Here, Z_ℓ^{out} is the final output at layer ℓ, which then serves as the input to the next layer or, in the case of the final layer, as the input to the classifier head. Through this iterative process applied over L layers, the feature representation is progressively enhanced. Each layer refines the output from the previous layer, ultimately producing an encoded representation of the image that is used for tasks like image classification or segmentation.

$$Z'_\ell = \text{MSA}(\text{LN}(Z_{\ell-1})) + Z_{\ell-1} \tag{3.3}$$

$$Z''_\ell = \text{LN}(Z'_\ell) \tag{3.4}$$

$$Z_\ell = \text{MLP}(Z''_\ell) + Z''_\ell \tag{3.5}$$

$$Z_\ell^{\text{out}} = \text{LN}(Z_\ell) \tag{3.6}$$

3.3.1.1 Attention Mechanism

3.3.1.1.1 Single-head Attention

This matrix Z is then used to project all the embeddings into the query-key space, which creates a distinct query and key vector for each embedding. The projection is accomplished by multiplying Z with the weight matrices W^Q, W^K, and W^V. These weight matrices are defined as $W^Q \in \mathbb{R}^{D\times d_k}$, $W^K \in \mathbb{R}^{D\times d_k}$, and $W^V \in \mathbb{R}^{D\times d_v}$, transforming the embeddings into lower-dimensional representations for queries, keys, and values, respectively. The resulting query, key, and value vectors for the entire sequence are represented by the matrices Q, K, and V, respectively, as defined by Equation (3.7).

$$\begin{cases} Q = ZW^Q \in \mathbb{R}^{N\times d_k}, \\ K = ZW^K \in \mathbb{R}^{N\times d_k}, \\ V = ZW^V \in \mathbb{R}^{N\times d_v} \end{cases} \tag{3.7}$$

where d_k is the dimensionality of the queries and keys, and d_v is the dimensionality of the values.

To capture contextual information within an image, the encoder utilizes a self-attention mechanism to evaluate the relevance of each patch relative to all others. This process involves calculating the dot products between all query and key vectors, as depicted in Equation 3.8 and visualized in Figure C.4. These calculations yield an attention matrix, shown in Figure 3.3, where the expression QK^T generates an $N \times N$ matrix. Each element of this matrix represents the dot product between a query vector and a key vector, effectively quantifying the similarity between all query-key pairs.

$$\text{Attention}(Q, K, V) = Z' = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V \in \mathbb{R}^{N\times d_v} \tag{3.8}$$

Building upon this concept, consider for example, the embedding $\mathbf{z}_1^5$ corresponding to a patch that captures the cat's face from the original image patches in Figure 3.1. This patch is projected to a query vector $\mathbf{q}_5$ and compared with another embedding $\mathbf{z}_1^8$, representing the cat's paws, now as a key vector $\mathbf{k}_8$ in Figure C.5. The similar directionality of $\mathbf{q}_5$ and $\mathbf{k}_8$ suggests that the query for the cat's face finds a match in the key for the cat's paws, implying that these patches are contextually relevant to one another.

This relationship is quantitatively represented by the large dot product between $\mathbf{q}_5$ and $\mathbf{k}_8$, indicating that the embedding of the cat's face patch is 'attending' to the paw's patch. By extending this process to all query-key pairs, the attention matrix effectively captures the most significant interactions between patches, as indicated by high dot product values.

To refine the representation for downstream processing, attention scores across the matrix are normalized into probability distributions via the softmax function applied to each query row in the attention matrix 3.3. This step is crucial due to the softmax function's sensitivity to significant variations among input values. Scaling the dot products by $\sqrt{d_k}$ ensures gradient stability during training and mitigates the potential saturation of the softmax function. As a result, softmax-normalized attention scores dictate the computation of a weighted sum of the value vectors within the matrix V, leading to the output matrix Z'. This matrix, as further illustrated, aggregates attention-weighted features for subsequent processing or classification tasks.

$$\begin{pmatrix} \mathbf{q}_1 \cdot \mathbf{k}_1 & \mathbf{q}_1 \cdot \mathbf{k}_2 & \cdots & \mathbf{q}_1 \cdot \mathbf{k}_8 & \mathbf{q}_1 \cdot \mathbf{k}_9 \\ \mathbf{q}_2 \cdot \mathbf{k}_1 & \mathbf{q}_2 \cdot \mathbf{k}_2 & \cdots & \mathbf{q}_2 \cdot \mathbf{k}_8 & \mathbf{q}_2 \cdot \mathbf{k}_9 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{q}_8 \cdot \mathbf{k}_1 & \mathbf{q}_8 \cdot \mathbf{k}_2 & \cdots & \mathbf{q}_8 \cdot \mathbf{k}_8 & \mathbf{q}_8 \cdot \mathbf{k}_9 \\ \mathbf{q}_9 \cdot \mathbf{k}_1 & \mathbf{q}_9 \cdot \mathbf{k}_2 & \cdots & \mathbf{q}_9 \cdot \mathbf{k}_8 & \mathbf{q}_9 \cdot \mathbf{k}_9 \end{pmatrix}$$

Figure 3.3: Dot product matrix with queries represented by rows and keys by columns

The goal is to compute a weighted sum along each row, with weights determined by the relevance of each patch. By applying the softmax across each row representing queries, we convert the raw scores into a probability distribution, where each value falls between 0 and 1. This distribution effectively indicates the relevance of each key (column) to the query (row), forming what is known as an attention pattern. For instance, if the patch of a cat's paws (key) is deemed relevant to the patch containing the cat's face (query), we seek to update the face patch's embedding to more accurately encode features specific to a cat's face.

Value vectors, obtained by projecting the embeddings onto a d_v-dimensional space using W_V, inform how each initial patch embedding should be adjusted based on its relevance to other patches. The operation of multiplying a patch embedding by W_V asks, "If this patch

is relevant to another, what specific information should be added to that other patch?" For each patch, the corresponding value vector is weighted by the attention scores, indicating the magnitude of its contribution to each other patch. For example, the updates for the fifth patch might be calculated as a weighted sum like $0.04\mathbf{v}_1 + 0.45\mathbf{v}_2 + \ldots + 0.58\mathbf{v}_8$, prioritizing patches with higher relevance.

The accumulation of these weighted contributions produces a delta change $\triangle\mathbf{E}_5$ for the original embedding $\mathbf{z}_1^5$, aiming for a refined vector that captures a richer context. This procedure, repeated for every row, generates a sequence of updates, which, when applied to the corresponding embeddings, results in a series of enhanced embeddings (Figure C.6). This collective update across all patches is what occurs within a single head of attention, yielding a set of embeddings Z' that are contextually enriched and more representative of the image's content.

3.3.1.1.2 Multi-head Attention

Multi-head attention extends the capability of single-head self-attention by allowing the model to process different parts of the input sequence in parallel (Figure C.4). Unlike single-head attention, which applies one set of weights to the input, multi-head attention operates with multiple sets of weights, referred to as heads. Each head is capable of capturing distinct features from the input data, thus enhancing the model's ability to represent a variety of patterns and dependencies. The multi-head attention process begins by transforming the input embedding matrix Z into sets of queries Q_j, keys K_j, and values V_j for each head j. This is expressed by the following equation 3.9 where j ranges from 1 to the total number of heads h and where W_j^Q, W_j^K, and W_j^V are the weight matrices corresponding to the j-th head for queries, keys, and values, respectively. For each head, the attention output Z'_j is computed using the scaled dot-product attention mechanism (see equation 3.10). After computing the attention for each head, the outputs Z'_j are concatenated and then linearly transformed to produce the final output Z' from the multi-head attention layer (see equation 3.11), where W^O is the projection matrix. The combination of information from multiple heads allows the model to capture a more nuanced representation of the input sequence, which may improve its performance on downstream tasks.

$$Q_j = ZW_j^Q, K_j = ZW_j^K, V_j = ZW_j^V \tag{3.9}$$

$$Z'_j = \text{Attention}(Q_j, K_j, V_j) \tag{3.10}$$

$$\text{Multi-Head}(Q, K, V) = Z' = \text{Concat}(Z'_1, Z'_2, \ldots, Z'_h)W^O \tag{3.11}$$

Parallelization and Computational Efficiency of Multi-head Attention

The efficiency of multihead attention is greatly augmented by its inherent parallelizability. Each head in the multihead attention mechanism can be executed in parallel, which aligns well with the capabilities of batch matrix multiplication—a procedure extensively optimized in modern deep learning frameworks such as PyTorch and TensorFlow. This parallel execution is further facilitated by the computational power of GPUs and TPUs, which are designed to efficiently handle such operations. To ensure the computational cost remains comparable to that of single-head attention, the dimensions of each head, d_k and d_v, are often set to a fraction of the original embedding dimensionality, specifically $d_k = d_v = \frac{D}{H}$. Consequently, the asymptotic complexity of multihead self-attention, excluding the projection steps, is $O(N^2 \cdot D)$, maintaining a quadratic relationship with the sequence length. This quadratic complexity poses challenges for scaling to longer sequences.

Multi-Layer Perceptron (MLP) in Encoder Block

The Multi-Layer Perceptron (MLP) is a key component in the architecture of transformer encoder blocks. It serves as the second most computationally intensive block within the structure. Once each embedding has shared information with one another through the attention mechanism, the MLP allows each embedding to process this information individually.
The standard MLP within the encoder block consists of a two-layered structure:

$$\text{MLP}(x) = W_2\sigma(W_1x + b_1) + b_2 \tag{3.12}$$

Here, $W_1 \in \mathbb{R}^{D\times 4D}$ and $W_2 \in \mathbb{R}^{4D\times D}$ are the weight matrices for the first and second linear layers, respectively. The b_1 and b_2 terms represent bias vectors. The non-linearity $\sigma(\cdot)$ is typically chosen to be a GELU (Gaussian Error Linear Unit) activation function, which provides a smoother output around the origin compared to other activation functions like ReLU. The use of an expansion factor of 4 in the MLP design has been empirically determined to perform well. This means that if we start with a D-dimensional embedding, we first project

it into a higher dimension, $4D$, using the matrix W_1, apply the GELU non-linearity, and then use the matrix W_2 to project it back down to the original dimensionality D. This expansion and subsequent compression allow the MLP to effectively perform complex transformations on the embeddings, which is crucial for the learning capability of the transformer model.

Understanding Residual Learning in Vision Transformers

Vision Transformers (ViT) utilize residual connections within their encoder architecture to facilitate the learning process and improve gradient flow through the network. The incorporation of residuals significantly simplifies the task that each layer must perform, shifting the focus from learning a complex input-output mapping to learning the perturbation needed to adjust the input. In a residual layer, the network does not attempt to learn the entire mapping from the input to the output directly. Instead, it learns a residual function f, representing the adjustments needed to bring the input closer to the target output. This is mathematically expressed in equation 3.13, where $f_1(X)$ is the output of the residual function for that layer, effectively acting as the perturbation. Figure 3.4 demonstrates that in a network with residual connections, we do not have to learn the entire mapping f_1 from scratch. Since X is already part of the input, only the residual $H_1 - X$ must be learned.

$$H_1 = f_1(X) + X \tag{3.13}$$

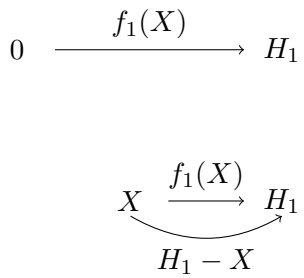


Figure 3.4: Illustration of learning with and without residual connections.

Multi-Head Self-Attention with Residuals

In the context of MSA within ViTs, the residual function R_{MSA} captures the transformation required by the MSA mechanism to refine its input (see equation 3.14. Consequently, the output of the MSA layer, with the residual connection can be seen in equation 3.15. This

ensures that the network only has to learn the residual function R_{MSA}, rather than the full transformation.

$$R_{\text{MSA}}(Z_{\ell-1}) = \text{MSA}(\text{LN}(Z_{\ell-1})) - Z_{\ell-1} \tag{3.14}$$

$$Z'_{\ell} = Z_{\ell-1} + R_{\text{MSA}}(Z_{\ell-1}) \tag{3.15}$$

Multi-Layer Perceptron with Residuals

The MLP part of the transformer similarly benefits from learning a residual function R_{MLP}, which is defined as the change needed to enhance the representation post-MSA, see equation 3.16. The final output after integrating the residual connection can be found in equation 3.17. The process of learning residuals rather than direct mappings is iterated over all L layers, allowing the ViT to incrementally refine the input embeddings. Each layer adds its learned adjustments, building up a sophisticated representation that is used for challenging tasks such as image classification or segmentation.

$$R_{\text{MLP}}(Z'_{\ell}) = \text{MLP}(\text{LN}(Z'_{\ell})) - Z'_{\ell} \tag{3.16}$$

$$Z''_{\ell} = Z'_{\ell} + R_{\text{MLP}}(Z'_{\ell}) \tag{3.17}$$

3.3.1.1.3 Intuition of Query in self-attention

Consider the sentence: ”The cold, refreshing lemonade, with its sweet yet tart flavor, is perfect for a hot day.” A query such as identifying the adjectives related to ”lemonade” can be represented by a query vector. This vector allows the model to determine the relevance of words in the sentence, specifically highlighting ”refreshing,” ”sweet,” and ”tart.” In Vision Transformers (ViTs), this concept is adapted for visual data. Instead of searching for textual features, a ViT queries visual features related to a specific image patch. If a patch shows part of an object, the query vector for this patch looks for other patches with related features or that contribute to understanding the object as a whole. This process effectively asks, ”What are the visual attributes or surrounding patches that share context with this one?” The attention mechanism then prioritizes patches that aid in fully understanding the object, similar to how adjectives add context to a noun in a sentence. Thus, in ViTs, each patch is analyzed not just

on its own but in relation to all other patches, ensuring a complete interpretation of the image. The configuration of the query vector in Vision Transformers varies with the implementation.

3.4 Taxonomy of ViTs

Image segmentation has greatly benefited from the emergence of Transformer architectures, which are generally categorized into patch-based and query-based approaches. Table **??** provides a detailed overview of a subset of popular models, all of which exploit the global receptive field characteristic of Transformers.

3.4.1 Patch-Based Transformers

Patch-based segmentation is a common approach in computer vision tasks, such as semantic and instance segmentation, where the image is divided into a grid of patches, and each patch is associated with a feature or query vector. The model then uses these feature or query vectors to attend to the corresponding patches and extract features from them. Patch-based segmentation has advantages, such as capturing fine-grained details and handling objects at multiple scales, but also limitations, such as computational expense and difficulty in capturing long-range dependencies.

3.4.2 Query-Based Transformers

Query-based object detection is an approach to object detection in which a set of learnable queries is mapped to objects in an image, with each query being associated with a particular type of object. During training, the model learns to associate each query with a type of object based on the annotations provided in the training data, and after training, it uses these learned queries to detect and classify objects in new images. Unlike patch-based approaches, where each patch of the image is associated with a query vector, in query-based object detection, each object in the image is associated with a query vector. This approach allows the model to detect multiple instances of the same type of object using the same query, but it also has limitations, as the model might not be able to distinguish between different instances of the same type of object based solely on the queries. Other factors, such as appearance or location, might be needed to distinguish between different instances.

Chapter 4

Objectives of the Study

The overarching aim of this master thesis is to investigate the efficacy of Transformer-based architectures in cell segmentation, which is a critical task for quantitative analysis in single-cell live microscopy. Cell-imaging is associated with inherent challenges such as low signal-to-noise ratios, the presence of non-cell particles, cell proximity, unpredictable movements, and the dynamic, non-rigid nature of cells undergoing mitosis. On top of this, the development of a universal and automatic cell segmentation technique still contains major challenges due to the extensive diversity observed in microscopy images. This diversity arises from variations in cell origins, microscopy types, staining techniques, and cell morphologies. The feasibility of automatic and precise cellular segmentation for specific microscopy image types and cell types has successfully been demonstrated in recent advances, such as fluorescence and mass spectrometry images [24; 25], differential interference contrast images of platelets [26], bacteria images [27], and yeast images [28; 29]. This master thesis aims to explore how Transformer networks can be adapted to address the above described challenges more effectively than conventional Convolutional Neural Networks (CNNs).

This thesis focuses on using Transformer architectures for cell segmentation, inspired by Rich Sutton's "The Bitter Lesson," [30] which strives for a shift from manually crafted features to models that inherently learn from data. Although CNNs have played a pivotal role due to their translation invariance, their limited receptive fields have proven to be a bottleneck when learning long-range dependencies [31]. Opposed to this, Vision Transformer (ViT) systems have a capacity for larger receptive fields, which gives them an advantage in medical diagnostics and results in their superior performance across various imaging tasks [32] [33]. In line with Sutton's observations on AI's evolution, which appear evident in areas like chess,

Go, speech recognition, and computer vision, this thesis states that computational power-centric methods will outperform methods that are embedded in domain-specific knowledge. By employing Transformer models that are proficient at handling the complex data of live-cell imaging, this master thesis aligns with the growing AI research trend that prioritizes scalable computational resources to address intricate problems.

By adding position embeddings to their architecture, a direct form of inductive bias is introduced in Vision Transformers (ViTs) in which information about the sequence order of image patches is embedded in the model. However, as Dosovitskiy et al. [19] discuss, these embeddings initially lack information on 2D spatial relationships, which means that initially the model does not have an understanding of the spatial interconnections among patches in two dimensions. This causes the need for the model to learn spatial relationships while training, which is different from CNNs in which spatial hierarchies and connectivity are inherent to the CNN architecture. Moreover, the self-attention mechanism in ViT enables each patch to interact with every other patch, regardless of their spatial proximity, which introduces an indirect inductive bias. This allows the model to allocate focus in a dynamic way to more distant parts of the image when this is necessary for the task. This is in contrast with CNNs, where deeper network structures or specific techniques such as dilated convolutions [34] would be required to address such long-range dependencies. Furthermore, Dosovitskiy et al. discuss the role of the MLP (Multi-Layer Perceptron) layers in transformers. They highlight that characteristics such as local and translational equivariance are primarily associated with these components. Their interpretation suggests that transformers have the capacity to learn and approximate these equivariances through its MLP layers, although their architecture does not inherently encode these properties.

A recent challenge, the ”2024 challenge”, aimed to benchmark universal algorithms capable of accurately segmenting cells from a wide range of microscopy images. Microscopy images were obtained from various imaging platforms and tissue types across four dimensions: cell origins, staining methods, microscope types, and cell morphologies. Participants were provided 1000 microscopy images with annotated masks and an additional 1725 unlabeled images [35]. The challenge showed that transformer-based deep learning models achieved superior performance. Transformers have exhibited robust performance and generalization capabilities across various computer vision tasks by integrating attention mechanisms for feature extraction. However, the potential of Transformers in biological image analysis re-

mains relatively unexplored [36]. Notably, Lee et al. [37] proposed using a SegFormer [38] as an encoder and a multiscale attention network [39] as a decoder. The authors concluded that whereas CNNs usually process local image patches, transformers use a self-attention mechanism that can capture global context and long-range dependencies in images. Secondly, they concluded that transformers have a larger model capacity than CNNs, enabling them to learn intricate patterns and model nuanced details. Thirdly, the authors concluded that transformers excel in transfer-learning settings, which allows transformer models to be pretrained on large datasets and subsequently fine-tuned for specific downstream tasks on new datasets with limited annotations.

Through this research, the thesis aims to advance the understanding of Transformer architectures' applicability and efficiency in cell segmentation tasks. This thesis aims to provide evidence on whether a Transformer-based network, tailored with smart attention mechanisms and minimal computational paradigms, can outperform traditional CNNs in handling the complexities of live-cell microscopy data. To this end, the research will focus on the development and evaluation of a Transformer-based network model for cell segmentation. The model will incorporate innovative strategies to instill inductive biases into the Transformer framework. This will include inductive biases such as hierarchical design, overlapping patch embedding, and a lightweight decoder similar to SegFormer or hybrid architectures like TransUNet, which combines CNNs with Transformer encoders for enhanced localization accuracy.

Chapter 5

Materials and Methods

5.1 Model Evaluation and Optimization

5.1.1 Validation Metrics

5.1.1.1 Notational Consistency for Evaluation Metrics

In Table 5.1, specific notational conventions are outlined that are used accross the evaluation metrics discussed in this thesis. A distinction is made between pixel-wise and object-wise metric calculations in semantic segmentation.

Table 5.1: Notation for pixel-wise and object-wise validation metrics

Notation	Description
tp	Pixel-wise true positives
fp	Pixel-wise false positives
fn	Pixel-wise false negatives
tn	Pixel-wise true negatives
TP	Object-wise true positives (post IoU thresholding)
FP	Object-wise false positives (post IoU thresholding)
FN	Object-wise false negatives (post IoU thresholding)
TN	Object-wise true negatives (not commonly used in segmentation)

5.1.1.2 Intersection over Union (IoU)

The Intersection over Union (IoU), also called the Jaccard similarity coefficient, is a metric used to evaluate the accuracy of a segmentation model. The IoU calculates the overlap between the predicted segmentation and the ground truth and is defined by equation 5.1, where $|\text{Intersection}(A, B)|$ refers to the number of pixels common to both the predicted segmentation and the ground truth (True Positives), and $|\text{Union}(A, B)|$ refers to the total number

of unique pixels present in both the predicted and actual segments. The value of IoU can range from 0, which indicates no overlap, to 1, which indicates full overlap (Figure D.1). Figure D.2 shows an example of the IoU. In the figure, a 5x5 grid of pixels is shown which represents a segmentation output, where white pixels mark the target area and black pixels the background. The IoU is calculated by dividing the count of True Positives (tp) by the sum of True Positives, False Positives (fp), and False Negatives (fn) (see the outcome for the example of Figure D.2 in equation D.1).

$$\text{IoU} = \frac{|\text{Intersection}(A, B)|}{|\text{Union}(A, B)|} \tag{5.1}$$

Computational Frameworks and Practical Implementation

In Table D.1, an overview is given of the subsequent steps to compute the Jaccard Index in binary classification. First, softmax predictions are converted to binary predictions using `argmax`. Second, these predictions and the true labels are flattened, which enables the calculation of their intersection and union. Third, the Jaccard Index is calculated by dividing the intersection by the union, which is adjusted by adding a small epsilon (ϵ) to prevent division by zero.

5.1.1.3 Sørensen–Dice Coefficient

The Sørensen–Dice coefficient (D), also known as the F1 score, evaluates the precision-recall balance in segmentation tasks by calculating the overlap between the predicted segmentation and the ground truth. For discrete data, the Dice coefficient is defined as twice the size of the intersection of the predicted and the ground truth segmentation masks, divided by the sum of their sizes (see equation 5.2). The Dice coefficient is computed separately for each class, and an average value is typically reported. Additionally, for boolean data within a given region of interest, the Dice coefficient is mathematically expressed in equation 5.3. Applying this formula to Figure D.2, the Dice coefficient can be computed (see equation D.2). Last, equation 5.4 shows how the Dice coefficient and the IoU are dependent on each other.

$$Dice\ Coefficient = \frac{2|Y \cap T|}{|Y| + |T|} \tag{5.2}$$

$$D = \frac{2 \times tp}{2 \times tp + fp + fn} \tag{5.3}$$

$$D = \frac{2 \times \text{IoU}}{1 + \text{IoU}} \tag{5.4}$$

5.1.1.4 Accuracy

The Accuracy score, also known as the Rand index, evaluates the number of correct predictions. Accuracy is calculated by the sum of correct positive and negative predictions divided by the total number of predictions, as can be seen in equation 5.5, where tp is true positives, tn is true negatives, fn is false negatives, and fp is false positives.

$$\text{Accuracy} = \frac{tp + tn}{tp + tn + fn + fp} \tag{5.5}$$

Although accuracy is a metric used by default to evaluate the total number of correct predictions, it is strongly discouraged to use this metric in Medical Image Segmentation (MIS). In MIS, a high class imbalance is typically present between regions of interest (ROIs) and the background and thus, a small percentage of pixels usually belong to the ROIs while the vast majority is background. As the accuracy metric incorporates the true negatives in its formula, it tends to produce misleadingly high scores in a scenario with high class imbalance. In this case, the accuracy metric does not accurately reflect the performance of the model in identifying the much smaller but more important ROIs.

Opposed to this, the Dice coefficient and the IoU both specifically penalize false positives and both metrics are therefore most commonly used for segmantic segmentation in highly class-imbalanced datasets such as those found in MIS. The IoU metric penalizes misclassification more heavily than the Dice score. This can be observed when comparing the two metrics in scenarios with moderate overlap and single misclassifications. As can be seen in Table D.2, the IoU metric has a lower value of 0.47 in the moderate overlap scenario, compared to the Dice score of 0.64, which shows that the IoU is more sensitive to misclassifications. The Dice score continues to have higher values even with significant false positives and false negatives, indicating that it is a more lenient performance measure [40]. In the scenario of a single misclassification, the Dice score remains relatively stable at 0.98, which shows that it is less sensitive to individual pixel errors. However, the IoU metric, which decreases slightly to 0.96, effectively captures the impact of a single misclassified pixel.

5.1.1.5 Mean Average Precision (mAP)

Object Detection

Object detection models are typically evaluated using the mean Average Precision (mAP), which is derived from precision and recall at different thresholds. Figure D.4 shows two instances of detection with different IoU scores. A higher IoU score (0.96) indicates a strong agreement between the predicted bounding box and the ground truth, whereas a lower IoU score (0.26) suggests a weaker agreement.

Precision and recall are used to evaluate the performance of object detection models per class and per image. It involves analyzing the model's predictions against the ground truths for a specific class in an image. In the context of the "dog" class for example, predictions are sorted by their confidence scores. A prediction is considered a True Positive (TP) if the Intersection over Union (IoU) with a ground truth is above a certain threshold; otherwise, it is a False Positive (FP). The precision for the class is computed by dividing the cumulative number of TPs by the sum of cumulative TPs and FPs (see Equation 5.6). The recall is calculated by dividing the cumulative number of TPs by the total number of ground truth instances of the class, as shown in Equation 5.7. Table D.4 gives details on the precision and recall values for successive predictions of the "dog" class within a given image, illustrating the calculation process.

$$\text{Precision} = \frac{Cumulative\ TP}{Cumulative\ TP + Cumulative\ FP} \tag{5.6}$$

$$\text{Recall} = \frac{Cumulative\ TP}{Total\ Ground\ Truths} \tag{5.7}$$

5.1.1.5.1 Semantic Segmentation

Semantic segmentation typically does not produce confidence scores for model predictions as offered by advanced instance segmentation models like Mask R-CNN and YOLOv8 Segment. For this reason, an adjusted average precision metric has been developed to better evaluate the performance in tasks that don't involve confidence scores. This new metric has been applied in various settings, including the Sartorius Cell Segmentation Challenge [41], assessments using the Cellpose model [42], and studies of single-cell segmentation in time-lapse microscopy , as seen with the implementation of the DeepSea model [6].

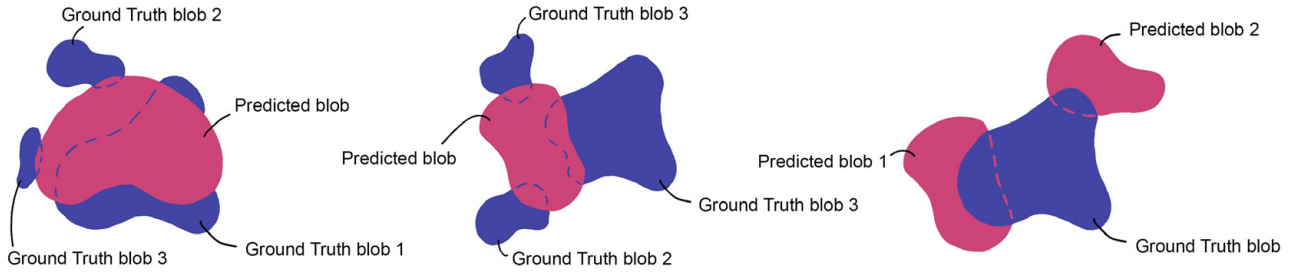


Figure 5.1: Overview of classification results. Left: True Positive (TP) – Predicted blob for which there is at least one Ground Truth blob with IoU $> t$. Middle: False Positive (FP) – Predicted blob for which there is no Ground Truth blob with IoU $> t$. Right: False Negative (FN) – Ground Truth blob for which there is no Predicted blob with IoU $> t$.

Average Precision (AP) at a threshold t is determined by the ratio of true positives (TP) to the sum of true positives, false positives (FP), and false negatives (FN), where a TP occurs when the predicted segmentation and a ground truth object have an IoU greater than t (see equation 5.8). To calculate AP, each blob within the ground truth and predicted masks must be extracted as an individual mask image, as shown in Figure 5.2. Subsequently, an IoU matrix is constructed to evaluate the overlap between each predicted and ground truth object and this evaluation is visualized through a heat map (Figure 5.3). For each ground truth blob, true positives (TP) and false positives (FP) are computed, and for each predicted blob, false negatives (FN) are calculated at varying thresholds t, as visualized in Figure 5.1. These values are used to calculate AP at different IoU thresholds. Table 5.2 gives an overview of these calculations and demonstrates how AP changes when adjusting the IoU threshold. This process is repeated for every image and then the AP values for each threshold are averaged to determine a final AP score for each threshold.

$$\text{Average Precision}(t) = \frac{TP(t)}{TP(t) + FP(t) + FN(t)} \tag{5.8}$$

IoU Threshold	TP	FP	FN	AP
0.5	36	3	1	0.923077
0.6	35	4	2	0.897436
0.7	32	7	5	0.820513
0.8	19	20	18	0.487179
0.9	4	35	33	0.102564

Table 5.2: Evaluation of TP, FP, and FN across different IoU thresholds for a single image, with the resulting AP calculated.

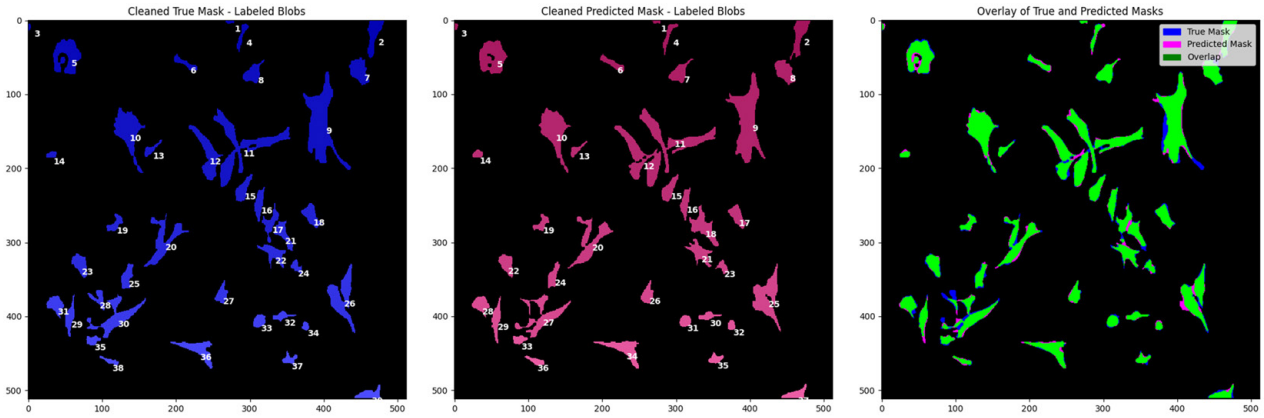


Figure 5.2: Visualization of cell labeling using `ndi.label`. Left: Cleaned True Mask showing labeled Ground Truth blobs. Center: Cleaned Predicted Mask with labeled predicted blobs. Right: Overlay of True and Predicted Masks, highlighting areas of overlap and identifying individual cells.

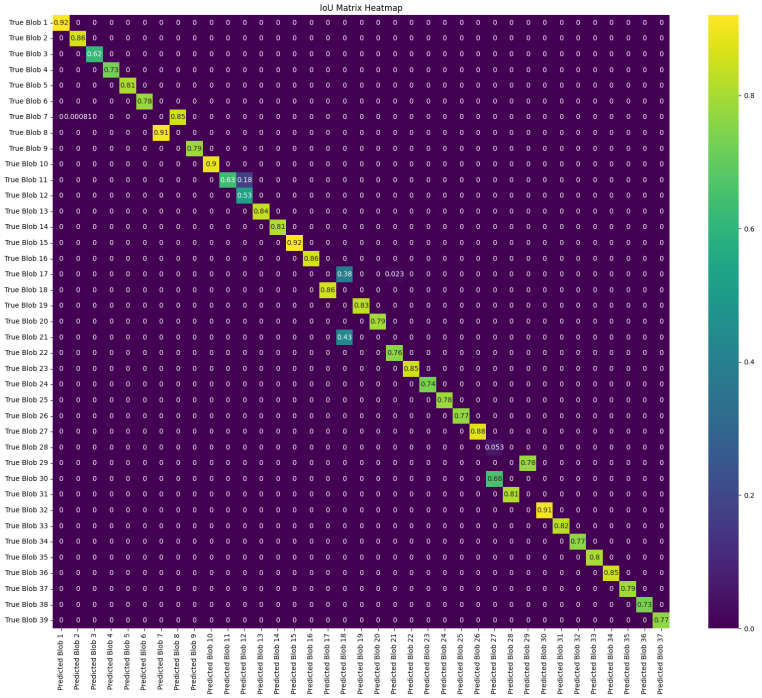


Figure 5.3: Visualization of cell labeling using `ndi.label`. Left: Cleaned True Mask showing labeled Ground Truth blobs. Center: Cleaned Predicted Mask with labeled predicted blobs. Right: Overlay of True and Predicted Masks, highlighting areas of overlap and identifying individual cells.

5.1.1.6 Pixel-Wise Precision-Recall Curve (PR-Curve)

The Precision-Recall Curve (PR-Curve) assesses model performance at the pixel level. For this evaluation, the output probabilities are varied across a wide range of thresholds. Subsequently, Precision and Recall are calculated for each threshold setting, as defined by equations 5.9 and 5.10.

$$\text{precision} = \frac{tp}{tp + fp} \tag{5.9}$$

$$\text{recall} = \frac{tp}{tp + fn} \tag{5.10}$$

These metrics, which relate directly to the confusion matrix (see Figure E.9), can be explained as follows: Recall (sensitivity) indicates the proportion of actual positive pixels (muscle cells) that are correctly identified by the segmentation model. High recall ensures that most muscle cell pixels are not missed. Precision measures the accuracy of the positive predictions. High precision means that a high percentage of pixels labeled as muscle cells by the model are true muscle cell pixels. This minimizes the misclassification of non-cell pixels as cells. In imbalanced datasets, the class of interest (e.g. muscle cells in medical images) often constitutes a smaller portion of the data compared to the background or other classes. This imbalance can lead to models that perform well on the majority class, while at the same time fails to identify the minority class. The PR curve helps to find a balance between precision and recall.

5.1.1.7 Relation to ROC Curve

The ROC Curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR) for different thresholds, incorporating True Negatives (TN). However, due to the high count of TNs in imbalanced datasets, as shown in Table 5.3, this can lead to an inflated AUC of 1.00 as depicted in Figure E.10.

5.1.2 Loss Functions for Image Segmentation

Selecting an appropriate loss function is essential in designing and training deep neural networks for image segmentation. According to surveys on loss functions specifically tailored for semantic segmentation, loss functions can be categorized into four primary types: 1)

Metric	Value
True Positives (TP)	433,612
False Positives (FP)	1,099
False Negatives (FN)	1,059,237
True Negatives (TN)	23,671,876
True Positive Rate (TPR)	0.29045938336697147
False Positive Rate (FPR)	0.000046424245368400045

Table 5.3: Classification results at a threshold of 0.99

Pixel-Level, 2) Region-based, 3) Boundary-based, and 4) Compounded loss functions [43], [44]. Each category is designed to address specific challenges and objectives within segmentation tasks. For an in-depth overview of these functions, we refer to Table D.3.

Pixel-level loss functions operate on the level of individual pixels. These functions assess the discrepancies between the predicted pixel values and their corresponding ground truth labels independently for each pixel. This approach is particularly effective in handling imbalanced class distributions by emphasizing pixels that are difficult to segment or by penalizing segmentation errors.

Region-based loss functions strive to minimize mismatches or, in other words, maximize overlaps between the predicted segmentation and the ground truth. These loss functions focus on overall class segmentation by trying to enhance the alignment between the predicted mask and the actual regions and thus prioritize accurate object segmentation over mere details on the level of pixels. Region-based loss functions are particularly useful in scenarios with inter-class imbalance and are robust to outliers and noisy labels. However, they may encounter optimization challenges, such as unstable gradients or issues with losses that are not entirely differentiable.

Boundary-based loss functions are a relatively newer category of loss functions that aim to minimize the distances or discrepancies between the predicted and actual boundaries of segmented regions. These functions are essential for achieving highly detailed boundary precision, which enables them to effectively separate overlapping objects. However, boundary-based loss functions can be challenging to train because of their non-convexity and may face issues such as exploding gradients or limitations in binary segmentation contexts.

Compounded loss functions integrate features from various types of loss functions, aiming to exploit their respective advantages while minimizing each of their limitations.

5.1.2.1 Pixel-Level Loss Functions

In the context of image segmentation, the cross-entropy loss function is widely adopted because of its effectiveness in pixel-wise classification. This function evaluates the accuracy of the output probability distribution for each pixel.

5.1.2.1.1 Pixel-wise Binary Cross-Entropy Loss

The binary cross-entropy loss function measures the performance of a classification model whose output is a probability value between 0 and 1. For each pixel i, the loss is calculated as can be seen in equation 5.11. Equation 5.11 sums up the negative logarithm of the predicted probability if the true label is 1, which corresponds to $-\log(y_{\text{pred}})$. In contrast, if the true label is 0, the equation considers $-\log(1 - y_{\text{pred}})$. Essentially, the loss is low when the predicted probability is close to the actual label and increases as the predicted probability diverges from the actual label.

$$BCE_{loss}(y_{\text{pred}}, y_{\text{true}}) = -\sum_{i}[y_{\text{true},i}\log(y_{\text{pred},i}) + (1 - y_{\text{true},i})\log(1 - y_{\text{pred},i})] \tag{5.11}$$

Figure D.6 visualizes the principles of the binary cross-entropy loss function. When the model correctly predicts a high probability, such as 0.83, for a true class label of 1, the loss is low $(-\log(0.83))$, and rewards the model's accuracy. In contrast, if the model incorrectly predicts a low probability for a true label of 1, it is penalized with a high loss $(-\log(0.17))$, which is the complement of the predicted probability. Similarly, a correct prediction with a low probability of 0.02 for a true label of 0 results in a low loss $(-\log(1 - 0.02))$, whereas an incorrect high probability prediction would lead to a higher loss, which effectively penalizes the model for its inaccuracy. This balance ensures that the model is tuned to increase precision by minimizing loss for correct predictions and incurring a cost for incorrect ones. The computation of the binary cross-entropy loss for an example pixel, as depicted in Figure D.6, is as follows: $CE = -[\log(1 - 0.02) + \log(1 - 0.01) + \log(0.83) - \log(1 - 0.08) - \log(1 - 0.06)] = 2.664$. The negative sign in the equation ensures that a lower (more negative) log probability contributes to a higher total loss, which encourages the model to adjust its predictions towards the true labels.

5.1.2.1.2 Focal Loss

The Focal Loss function has proven to be highly effective in addressing classification problems, especially those involving significant class imbalance. This loss function modifies the traditional cross-entropy loss to apply a higher weighting to examples that are difficult to classify, thereby focusing the training more on these cases. In equation 5.12, Focal Loss is mathematically defined, where p_t is the model's estimated probability for the class with the true label, α_t is a coefficient that weights the importance of the correct class in the loss function, and γ is the focusing parameter that adjusts the rate at which easy examples are down-weighted. Specifically, the term $(1 - p_t)^\gamma$ effectively reduces the loss contribution from easy examples (where p_t is high), which helps to minimize the impact of class imbalance by allowing the model to focus on more challenging samples that are misclassified.

$$F_{\text{loss}}(p_t) = -\alpha_t(1 - p_t)^\gamma \log(p_t) \tag{5.12}$$

5.1.2.2 Region-Based Loss Functions

5.1.2.2.1 Dice Loss

The Dice loss stems from the Dice Coefficient, a similarity metric between two data sets (see Dice Coefficient Equation 5.2). It can be considered as a relaxed, differentiable version of the Dice Coefficient, proposed by Milletari et al. [45]. The Dice loss, L_{dice}, for a single class is formulated in equation 5.13, where Y is the binary segmentation prediction mask, and T is the binary segmentation target mask for a single class. C represents the number of classes and N is the number of pixels in the images. The predictions are not taken as binary values but rather as probabilities between 0 and 1, allowing the function to be differentiable. The Dice loss is optimized using gradient descent methods and is particularly beneficial for imbalanced datasets, as it mitigates the model's propensity to ignore minority classes by focusing on the overlapping regions between the predicted and ground truth masks.

$$L_{dice} = 1 - \frac{1}{C}\sum_{c=0}^{C-1}\frac{2\sum_{n=1}^{N} y_n^c t_n^c}{\sum_{n=1}^{N} y_n^c + \sum_{n=1}^{N} t_n^c} \tag{5.13}$$

In binary segmentation tasks, the Dice coefficient typically focuses on the "positive" class, often representing the object of interest such as cells in medical imaging or a specific feature in an image. The coefficient quantifies the precision of overlap between the predicted positive class and the true positive class, hence evaluating the model's ability to correctly identify and

segment the object in question. The ”negative” class, or the background, is not factored into the Dice coefficient calculation as this metric aims to specifically assess the performance on the positive class. This is due to the inherent interest in the accuracy of the segmentation for the class that represents the phenomenon under study, rather than the background.

Consider the binary vectors for ground truth and predictions for a single class as follows:

$$\textbf{Ground Truth} = [1,1,1,0,0,0,1,1,1,0], \quad \textbf{Predictions} = [1,1,0,0,0,0,1,1,0,1].$$

The Dice coefficient D as we saw in equation 5.3, yields for our example, where $TP = 4$, $FP = 1$, and $FN = 2$, an outcome in equation 5.14. Accordingly, the Dice loss L_{dice} can be computed as $1 - D$, which results in equation 5.15. Thus, a Dice coefficient of 0.727 reflects a moderate overlap between predicted and actual positive instances, while the corresponding Dice loss of 0.273 quantifies the error in the segmentation model.

$$D = \frac{2 \times 4}{2 \times 4 + 1 + 2} = \frac{8}{11} \approx 0.727. \tag{5.14}$$

$$L_{dice} = 1 - \frac{8}{11} \approx 0.273. \tag{5.15}$$

5.1.2.3 Discussion on Loss Function Selection for Cell Segmentation Accuracy

Dice loss is valuable for cell segmentation as it measures the overlap between predicted and true cell regions, helping to balance precision and recall without being overly sensitive to class imbalances. Focal loss is advantageous for focusing the training on cells that are difficult to classify, ensuring that the model does not overlook difficult segments. Used together, Dice loss improves general segmentation accuracy, while Focal loss targets specific errors.

5.2 Dataset Description

5.2.1 Overview of Datasets

The Phase Contrast Time Lapse Microscopy Image Datasets comprise 49,919 images with a resolution of 1392x1040, stored as 16-bit TIF files, distributed across three datasets (see D.9). Each dataset is structured into 16 subfolders, ranging from exp1_F0001 Data to exp1_F0016

Data. Every subfolder contains an image sequence with the number of frames ranging between 1013 and 1062, varying according to the specific experimental conditions. The images are publicly available and can be accessed at `https://osf.io/ysaq2/`.

The entire dataset includes a total of 48 time-lapse sequences, recorded over approximately 3.5 days. The cells were cultured under four distinct media conditions: control, 50 ng/mL FGF2, 100 ng/mL BMP2, and a combination of 50 ng/mL FGF2 and 100 ng/mL BMP2. These conditions were chosen to study cellular behaviors and morphological changes due to the influence of these growth factors. Each treatment group is documented across 12 distinct phase-contrast time-lapse microscopy image sequences [46].

Time-lapse phase-contrast microscopy was conducted using a Zeiss Axiovert T135V microscope (Carl Zeiss Microimaging, Thornwood, NY) equipped with a 5X, 0.15 N.A. phase-contrast objective. This setup included a custom-stage incubator that could accommodate up to four 35mm Petri dishes and was operated using In Vitro software version 3.2 (Media Cybernetics Inc., Bethesda, MD). For each set of conditions, four fields of view representative of the cell density from four dishes were selected, resulting in 16 fields of view per experiment. This process was repeated three times to ensure comprehensive data collection across all conditions, capturing images every five minutes to document the cellular dynamics thoroughly.

In addition to the primary dataset from deepseas [47], a different expanded dataset was utilized to train a generalized, non-specific model. This supplementary dataset comprises 2075 images, each with corresponding masks. The dataset includes four distinct cell types, which are characterized under specific conditions and imaging techniques. The detailed descriptions of these cell types are illustrated in Figure 5.4.

5.2.1.0.1 Dataset Description of Binary Masks

The dataset used in this study includes binary .PNG masks sourced from the DeepSea data repository [47]. This collection includes 60 masks distributed across 8 subfolders, which results in a total of 480 masks. In order to ensure diversity across the dataset, every fifth image was selected for masking, which means four images were skipped between each selected mask. For each `exp1_F00XX` folder, images were chosen from 00001 to 00237,

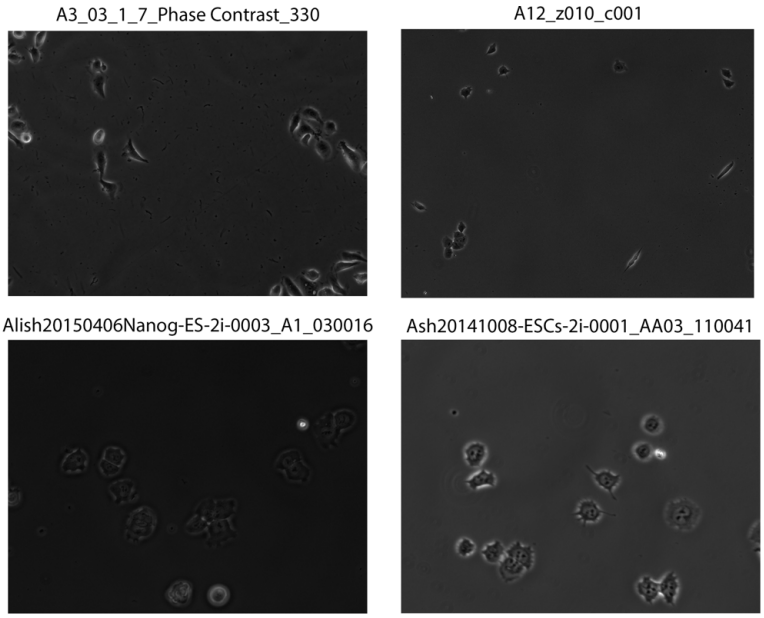


Figure 5.4: Combined image showing four different cell types: (Top left) A3_03_1_7_Phase Contrast_330, (Top right) A12_z010_c001, (Bottom left) Alish20150406Nanog-ES-2i-0003_A1_030016, (Bottom right) Ash20141008-ESCs-2i-0001_AA03_110041. Each quadrant represents a distinct cell type.

covering approximately one-fourth of an entire folder. This sampling strategy helps to cover a broad range of cell behaviors and morphologies.

5.2.1.0.2 Segmentation of Touching Cells and Filopodia

The segmentation of cellular images, as shown in Figure D.10, requires the accurate identification of individual cells, especially when they appear to be in contact. To achieve this, binary masks are designed to distinguish these cells as separate entities, assigning unique blobs to each, regardless of their proximity. It is also ensured that smaller cells are adequately represented in the masks. For instance, as demonstrated in the lower-middle example of Figure D.10, even the smallest cells that have just emerged due to mitosis are distinctly segmented. In addition, the segmentation process includes filopodia, which are cellular extensions distinct from the nucleus due to their more transparent and less dense nature, as well as their elongated shape. However, it should be noted that the segmentation of filopodia in this context is relatively rough. This is not a limitation of the DeepSea authors' approach, but rather a reflection of the significant manual effort required for more detailed segmentation. The primary focus remains on accurately identifying individual cells and adequately representing smaller cells in the binary masks.

5.2.2 Labeling Process and Tools

5.2.2.1 Label Creation with QuPath

In this study, QuPath, an open-source software for bioimage analysis, was employed to annotate cellular structures. The software's 'paint brush tool' facilitated the creation of complex non-overlapping shapes and enabled efficient classification of different structures such as tentacles and cell bodies. A total of 171 masks were labeled, with 5 in high-density areas and 166 in medium-density areas. Figure D.7 shows a screenshot of the QuPath interface during the annotation process, highlighting the tools panel and the application of the paint brush tool to both filopedia and cell bodies. Figure 5.5 provides a visual comparison of labeled cells in medium and high-density regions.

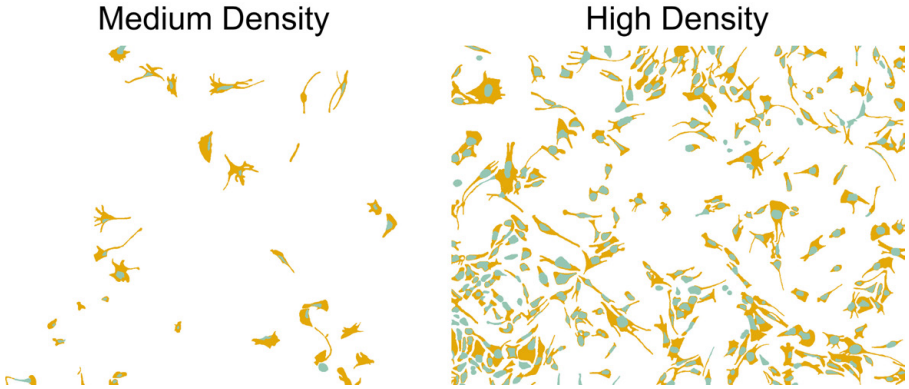


Figure 5.5: Comparison of filopodia and nucleus labeling in cell images at different densities. The images on the left show a medium density of filopodia (orange) and nuclei (greenish), while the images on the right exhibit a high density. This illustrates the robustness of the labeling technique across varying cell densities.

5.2.2.2 Handling labelling ambiguities

The process of labeling cellular structures presents several challenges, as depicted in Figure 5.6. The top left image illustrates the difficulty in distinguishing filopedia due to their low contrast against the background, which sometimes requires zooming out or reviewing sequence frames for clearer identification. Similarly, defining the cell body can be problematic; there isn't always a distinct separation from surrounding structures, as shown in the top left image. Lighting artifacts further complicate this issue, with instances like the white cell on the top of the top right image obscuring clear delineations. Overlapping cells, such as those in the bottom of the top right image, pose additional difficulties in distinguishing individual cell boundaries during phases of cell interaction. The bottom right image showcases a high-

density environment where the overlap between filopedia and cell bodies is frequent, complicating the accurate establishment of ground truth labels. In such densely packed scenes, the large and sometimes indistinct shapes of cell bodies blend into the filopedia, with filopedia occasionally extending across multiple cells.

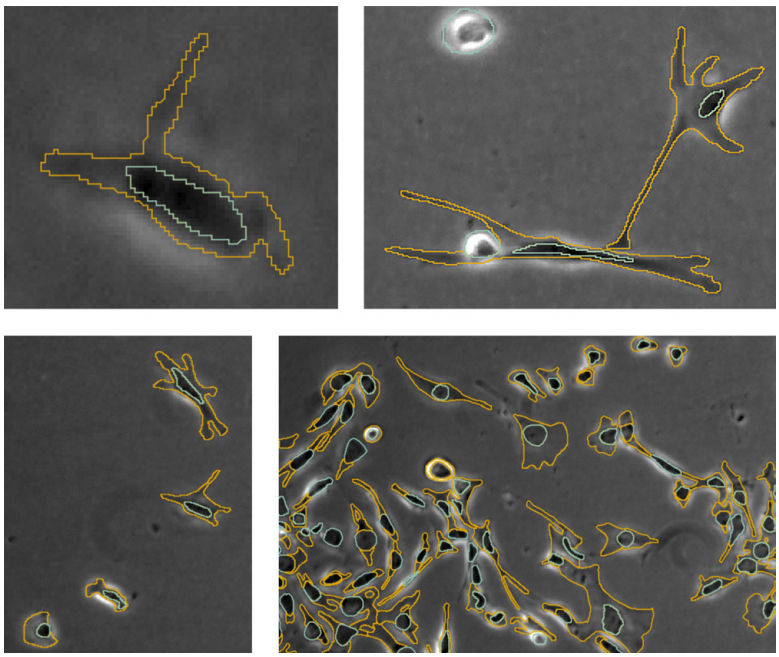


Figure 5.6: Detailed visualization of cell annotations using QuPath. Top left: Zoomed-in view of a single labeled cell highlighting the precision of boundary marking. Top right: Image of a cell body in close proximity to an elongated cell, with another long cell nearly touching. Bottom left: Four distinct cells clearly separated, demonstrating varied cell shapes and sizes. Bottom right: High-density labeling where numerous cells are in close contact, making it difficult to distinguish between cell bodies and filopedia.

5.2.3 Data Preprocessing

In the preprocessing of cell images, one transformation of particular interest is Contrast Limited Adaptive Histogram Equalization (CLAHE). According to Zargari et al. (2023) [6], CLAHE is effective in enhancing the visibility of cellular structures within images. The CLAHE technique adapts the image contrast locally and has been used extensively to improve the contrast of cell images, for example in situations where variability in lighting conditions may obscure critical details.

The following transformations are applied in our preprocessing pipeline for .tif images:

- **ToFloat**: Applied to normalize the image pixel values, scaling them to a range suitable for processing, with a maximum value set to 65535.0.

- **Resize**: Changes the image dimensions to 512×512 pixels, providing a consistent shape for input into neural networks.
- **CLAHE**: Applied through a custom function, it enhances local contrast in the images, making it easier to identify and segment individual cells.
- **Gaussian Noise**: Adds variability to the images with a variance limit set to 0.005, potentially increasing the robustness of the model to slight perturbations in new data.
- **Horizontal Flip**: Introduces horizontal flipping with a probability of 0.5, augmenting the dataset.
- **Random Brightness and Contrast Adjustments**: Applies brightness and contrast adjustments with limits set to 0.2, and a probability of 0.3, simulating different imaging conditions.
- **Gaussian Blur**: Applies Gaussian blur with a blur limit ranging from 3 to 7, and a probability of 0.3, further enhancing the variety of the augmented dataset.

Figure D.8 depicts a cell image with contrast enhanced by CLAHE, alongside the corresponding mask that shows the regions of interest (ROIs) in white.

5.2.3.1 Data Preprocessing and Loading

Images and corresponding masks are loaded from directories using the `ImageMaskDataset` class. This class lists relevant files (.tif for images and .png for masks) and loads each pair with the Python Imaging Library (PIL). Images and masks are converted to NumPy arrays and optional transformations are applied. If required, a channel dimension is added to the images and masks to ensure compatibility with deep learning models. Masks are thresholded to binary values. The processed images and masks are then converted to PyTorch tensors. The dataset is split into training and validation sets, with a 20% split for validation.

5.3 Hardware

The resources and services used in this work were provided by the VSC (Flemish Supercomputer Center), funded by the Research Foundation - Flanders (FWO) and the Flemish Government [48]. Additionally, resources from Kaggle were utilized [49].

5.3.1 Hardware Specifications

The following hardware was employed: h100, a100, and v100 from VSC HPC, as well as p100 and T4 from Kaggle. Notably, for the h100 hardware, an optimal configuration was achieved with `ntasks=16`, which was found to be an efficient parameter setting. Increasing this value required the use of additional GPUs. Furthermore, the following job settings were used: `mem-per-cpu=10000`.

5.4 Computational Tools and Frameworks

5.4.1 PyTorch

In this study, the neural network models were implemented using the PyTorch library (version 2.0.0), a widely used open-source machine learning framework that provides tools for deep learning [50].

5.4.1.1 Torchvision

The study utilized the torchvision library, a suite of tools, datasets, and pre-trained models designed for computer vision tasks. Torchvision is part of the PyTorch framework. This library offers well-documented and optimized models suitable for various vision-based tasks [51].

5.4.2 Tensorflow

This study utilized TensorFlow, an open-source machine learning framework developed by Google. It offers tools and libraries that facilitate the development and deployment of machine learning models across a wide range of applications [52].

5.4.2.1 Keras

This study utilized the Keras library, a high-level neural networks API that allows for quick development of deep learning models. Keras runs on top of TensorFlow and is known for its user-friendly and modular architecture [53].

5.4.3 HuggingFace Transformers

The Transformer vision model library from Hugging Face was used. It hosts a collection of readily usable and adaptable models. The library includes a diverse range of transformer-based architectures such as the original Vision Transformer (ViT) and its various adaptations including ViT Hybrid, ViT DeT, ViT Mae, Swin Transformer, Mask2Former, among others [54].

5.4.4 Additional Libraries and Tools

In this research, multiple Python libraries were utilized for specific tasks related to image processing and machine learning. **NumPy** and **Pandas** were employed for numerical operations and data handling respectively, providing essential tools for dataset manipulation. Image processing tasks were handled using **OpenCV** and **Scikit-Image**, which supported image transformations and analysis. For visualization, **Matplotlib** and **Seaborn** were used to generate graphs and plots to analyze and present data effectively.

The machine learning components of the project utilized **PyTorch** and **TensorFlow**, including high-level functionalities from **Torchvision** and **TensorFlow Keras**. These libraries provided the necessary frameworks for building and training deep learning models, specifically convolutional neural networks and transformers. **Albumentations** was used for advanced image augmentation techniques to enhance model training. Additionally, model evaluation and metrics computation were facilitated by **TorchMetrics** and **Sklearn**'s model selection tools, critical for optimizing and validating the segmentation models.

5.5 Implementation of Segmentation Models

5.5.1 Convolutional Approaches

5.5.1.1 Attention U-Net

5.5.1.1.1 Dataset and Loader Configuration

Original 16-bit .tif images were used, with transformations applied as detailed in Section 5.2.3.

5.5.1.1.2 Model Architecture

The Attention U-Net was implemented in PyTorch. The modified Attention U-Net model, detailed in 2.2.1, employs `nn.Conv2d` layers with a 3x3 kernel, stride of 1, and padding of 1, paired with `nn.BatchNorm2d` and `nn.ReLU`. Encoder blocks incorporate `nn.MaxPool2d` with a 2x2 kernel and stride of 2. Decoder blocks utilize `nn.Upsample` with a scale factor of 2 and `nn.Conv2d`. The network starts with a single input channel, expanding to 64 feature channels initially, with batch normalization applied across all convolutional blocks. The attention mechanism uses 1×1 convolutions, batch normalization, ReLU, and a sigmoid function to create the attention map ψ.

5.5.1.1.3 Loss Functions and Metrics

Experiments were conducted with focal loss and dice loss. Jaccard index for both training and validation set were saved each epoch.

5.5.1.1.4 Training Process and Configuration

Experiments were conducted using a batch size of 4 and 2 worker threads, with the Adamax optimizer set to a learning rate of 1e-3. The Attention UNet model was configured with 1 class and 1 input channel. During training, validation IoU and loss, as well as training IoU and loss, were monitored to assess performance. A regular U-net was trained for about 150 epochs on a P100 GPU, while an extended data version, consisting of 2075 examples, was trained on an H100 GPU in a high-performance computing (HPC) environment.

5.5.1.2 Mask R-CNN

5.5.1.2.1 Dataset and Loader Configuration

Each image undergoes the transformations detailed in Section 5.2.3. The original 16-bit .tiff images are processed, with bounding boxes computed from masks decoded from the annotations, formatted as tensors of type `torch.float32`. Labels are stored as `torch.int64` and masks as `torch.uint8`. The configuration for bounding box parameters is set to 'pascal_voc', with 'label_fields' specifying 'labels'. Additionally, the dataset includes a replay feature from Albumentations, which captures the transformations. In the DataLoader setup, a custom collate function, `custom_collate_fn`, is utilized to manage batch data during training. This function reorganizes the batch data by using `zip(*batch)`, which transposes the

batch, converting a list of tuples into a tuple of lists where each tuple contains tensors like images and labels.

5.5.1.2.2 RLE-Encoding

Annotations for cell instances within each image are encoded using Run-Length Encoding (RLE), as demonstrated in Table 5.4. RLE is a compact and computationally efficient method of storing data where sequences are stored as single data values and counts. In the context of our dataset, each number in the RLE string represents a part of an annotation for a single cell. The first number indicates the starting pixel index where a cell's mask begins, and the subsequent number indicates how many consecutive pixels are part of this mask. For example, '275601 1' indicates that starting at pixel index 275601, there is one pixel belonging to an annotation mask. This sequence continues, detailing each instance of a cell within the image. This method allows for storing detailed mask information in a single Excel file, providing a very compact representation for each image. Masks are decoded from this RLE format on-the-fly as images are requested by the DataLoader using a custom `rle_to_mask` function.

Table 5.4: RLE Encoding Examples in Dataset Annotations

id	annotation	width	height
exp1_F0010-00217	275601 1 276641 1 277681 2	1392	1040
exp1_F0010-00217	743615 3 744655 5 745695 5	1392	1040
exp1_F0010-00217	536798 5 537833 14 538870 19	1392	1040

5.5.1.2.3 Model Architecture

Mask R-CNN was implemented in PyTorch. A pre-trained Mask R-CNN model, `maskrcnn_resnet50_fpn` from torchvision, utilizing a ResNet-50 backbone and a Feature Pyramid Network (FPN), was adapted for this project. Initially trained on the COCO dataset, the model was customized for two classes by modifying the box and mask predictors within the ROI heads—specifically replacing the box predictor with a `FastRCNNPredictor` and the mask predictor with a `MaskRCNNPredictor` using a hidden layer size of 256. Pre-trained weights were manually loaded from local storage to ensure reproducibility and consistency across different computational environments. The model was then deployed on the appropriate computational device, set to training mode, and prepared for fine-tuning on the dataset specific to this study.

5.5.1.2.4 Loss Functions and Metrics

The Mask R-CNN model uses a combined loss function to train for object detection and instance segmentation. This function comprises: (1) *Classification Loss*—a softmax loss that classifies each Region of Interest (ROI) as an object or background, (2) *Bounding Box Regression Loss*—a smooth L1 loss that adjusts the positions of bounding boxes around detected objects, and (3) *Mask Loss*—a binary cross-entropy loss applied at the pixel level to create accurate segmentation masks for each instance. These loss components are summed to form the total loss, which the training process aims to minimize to improve both detection and segmentation accuracy.

5.5.1.2.5 Training Process and Configuration

The Mask R-CNN model was trained over 40 epochs on 1392x1040 images, using an SGD optimizer with parameters such as a learning rate of 0.001, momentum of 0.9, and weight decay of 0.0005. The model configuration included a mask threshold set at 0.5 for pixel retention and a maximum of 539 box detections per image with a minimum score of 0.59.

5.5.2 Transformer-Based Models

This subsection delves into models that leverage transformer architectures, highlighting how these models utilize attention mechanisms to improve segmentation accuracy.

5.5.2.1 SegFormer

5.5.2.1.1 Dataset and Loader Configuration

Images were reshaped and transformed as detailed in Section 5.2.3, with normalization adapted for 24-bit depth .png images (max value 65535 instead of 255) and resized to 512x512 pixels.

5.5.2.1.2 Model Architecture

The Segformer architecture was implemented in PyTorch. The Segformer architectures MiT-B0, MiT-B3, and MiT-B5 were used. These models were implemented using the Hugging Face library [55] [56]. Detailed configurations for each model variant, including layer depths, hidden sizes, and decoder configurations, are provided in Table 5.5.

Model Variant	Depths	Hidden Sizes	Decoder Hidden Size
MiT-B0	[2, 2, 2, 2]	[32, 64, 160, 256]	256
MiT-B3	[3, 4, 18, 3]	[64, 128, 320, 512]	768
MiT-B5	[3, 6, 40, 3]	[64, 128, 320, 512]	768

Table 5.5: Parameters and Performance of Selected Segformer Models

5.5.2.1.3 Loss Functions and Metrics

The SegFormer model employs Cross-Entropy Loss for multi-class segmentation tasks and Binary Cross-Entropy Loss for binary segmentation tasks [57].

5.5.2.1.4 Training Process and Configuration

During the training of the SegFormer model, logits are upsampled to match the ground truth label dimensions using bilinear interpolation (`F.interpolate(logits, size=labels.shape[-2:], mode='bilinear', align_corners=False)`). This ensures that the output size is consistent with the input label size for accurate loss calculation.

5.5.2.2 Swin-UNet

5.5.2.2.1 Dataset and Loader Configuration

The dataset loader for Swin-UNet reads and preprocesses 24 bit .png images and masks, resizing them to a standard dimension and normalizing pixel values to [0,1]. Masks are one-hot encoded for segmentation tasks. Augmentations include random cropping, resizing, and horizontal or vertical flips to enhance data variability.

5.5.2.2.2 Model Architecture

The Swin-UNet was implemented in Tensorflow. The Swin-UNet was configured in Keras TensorFlow and accepts RGB images of size 512×512 pixels. It starts with 32 filters and implements an encoder-decoder structure with a depth of 4, comprising two stacks each in both downward and upward paths. The model utilizes a patch size of 4×4, multi-head attention configurations of [4, 8, 8, 8] across the layers, and window sizes of [4, 2, 2, 2] for computing self-attention. Additionally, each Transformer block contains 156 MLP units, and the shift window mechanism is enabled to enhance feature processing across different image segments. The output layer uses a 'Softmax' activation function.

5.5.2.2.3 Loss Functions and Metrics

The Swin-UNet model was trained using the Focal loss, adapted from the categorical cross-entropy provided by Keras, and the Adam optimizer. The Jaccard index and Mean Intersection over Union (mIoU) served as the training metrics.

5.5.2.3 TransUNet

5.5.2.3.1 Dataset and Loader Configuration

24-bit .png images and masks, initially untransformed, were resized to 512x512 dimensions.

5.5.2.3.2 Model Architecture

The TransUNet was implemented in Tensorflow. TransResUNet employs a VGG16 backbone with residual blocks and modified skip connections, designed for binary segmentation. The network uses an input size of $512 \times 512 \times 1$, adjusting to three channels when necessary for compatibility with VGG16. It features three encoder blocks for downsampling, each linked to a corresponding decoder block through upsampling and convolution layers. Central to the model is a bridge between the deepest encoder and its decoder, optimizing feature transfer. The model outputs through a final convolution with sigmoid activation.

5.5.2.3.3 Loss Functions and Metrics

The TransResUNet model utilizes a custom Dice coefficient loss function for optimization. The model is compiled with the Adam optimizer and a learning rate of 1×10^{-5}. Performance metrics include the intersection over union (IoU), Dice coefficient, and binary accuracy. The IoU metric was modified with a smoothing factor of 100.

5.5.2.3.4 Training Process and Configuration

The TransResUNet models, both regular and extended, were trained for 100 epochs using the same configuration. The regular TransResUNet was trained on 384 examples, while the extended version was trained on 1967 samples from the dataset (see 5.2.1.0.1). Subsequently, the extended model was fine-tuned on the original 384 examples to enhance performance on the initial dataset (see 1.4.2).

5.5.3 Training Time and Platform Utilization:

The training times and hardware utilization for various models are detailed in Table 5.6. This table includes both convolutional-based models such as the Attention U-Net and Mask-RCNN, and transformer-based models like MiT-B0, MiT-B3, MiT-B5, TransResUnet, and Swin-Unet. These models were trained using different hardware configurations across platforms such as Kaggle and the VSC HPC.

Table 5.6: Training Times and Hardware Specifications

Training Times and Hardware Specifications				
Model Type	**Model**	**Minutes/Epoch**	**Hardware**	**Platform**
Convolutional-based	Attention U-Net	1:09	2 x T4	Kaggle
	Mask-RCNN	2:45	1 x P100	Kaggle
Transformer-based	MiT-B0	2:00	1 x H100	VSC HPC
	MiT-B3	2:49	1 x H100	VSC HPC
	MiT-B5	2:36	3 x H100	VSC HPC
	TransResUnet	1:02	1 x P100	Kaggle
	TransResUnet - Finetuned	5:39	1 x P100	Kaggle
	Swin-Unet	3:01	1 x H100	VSC HPC

5.5.4 Model Parameters and Inference time

Table 5.7 provides a summary of the parameter counts and inference times per image for all evaluated models.

Model Type	**Model Variant**	**Params (M)**	**Inference Time (s per image)**
Convolutional Approach	Attention U-Net	34.8	0.7625
	Mask-RCNN	43.9	1.2084
Transformer Approach	TransResUnet	5.9	3.7006
	Swin-UNet	2.2	0.324
	MiT-B0	3.7	0.0079
	MiT-B3	45.2	0.0063
	MiT-B5	82.0	0.0303

Table 5.7: Parameters and Performance of Selected Models Including Inference Time

Chapter 6

Results

6.1 Training Metrics

6.1.1 Jaccard index

A comparison of the Jaccard Index across models is shown in Figure 6.1. We observed that the Jaccard Index for MiT-B0 (shown in red) reaches a stabilization point at 0.81 in validation around epoch 120, after which the training Jaccard Index begins to diverge, starting around epoch 80. For MiT-B3 and MiT-B5 (shown in green and blue, respectively), the validation Jaccard Indexes are generally higher than MiT-B0, but have more significant divergence in the training phases.

Furthermore, the Attention U-Net stabilizes at a slightly higher Jaccard Index compared to MiT-B5, while the TransResUnet achieves a peak of approximately 0.84. Specific values observed include a Training Jaccard Index of 0.790 and a Validation Jaccard Index of 0.969 for MiT-B0 at epoch 80, with the Training Jaccard Index increasing to 0.828 by epoch 120.

Figure E.1 illustrates the training and validation Jaccard scores of an Attention U-Net model trained with an expanded dataset of 2075 images, as detailed in Section 5.2.1.

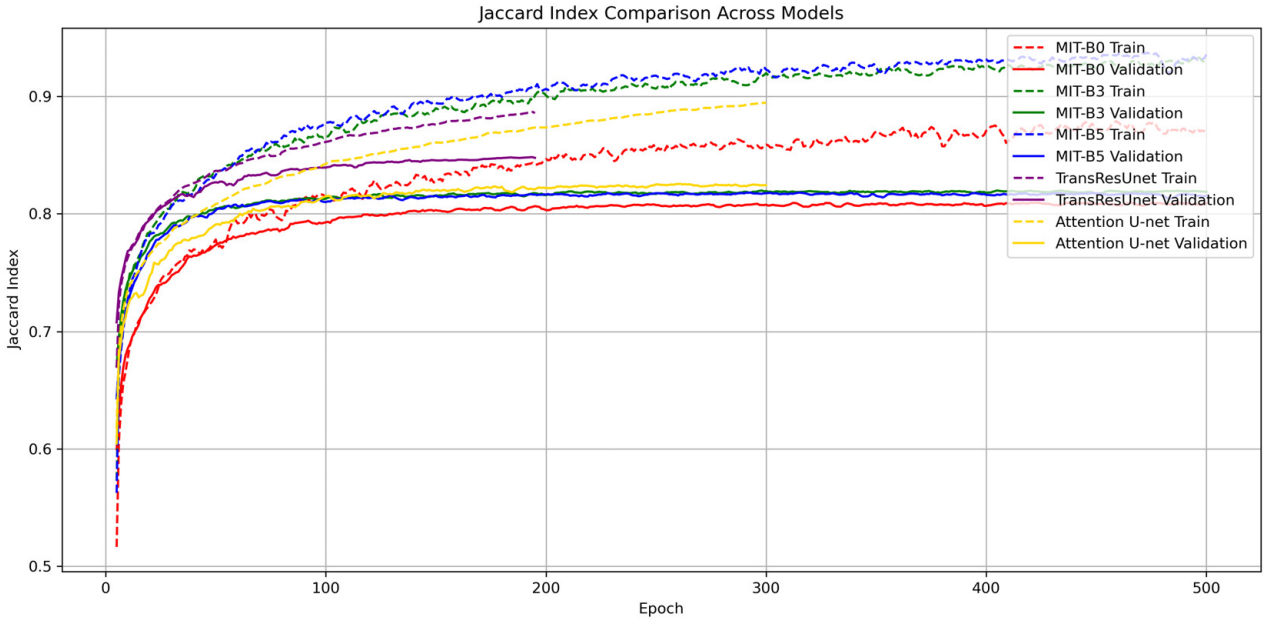


Figure 6.1: Comparative Jaccard index values across models during training and validation phases. This figure highlights differences in learning dynamics and generalization capabilities of the models.

6.2 Validation Metrics

6.2.1 Average Precision

Figure 6.2 shows the average precision (AP) of various deep learning models for object segmentation as a function of the decision threshold. Among the convolution-based models, the Att-U-Net starts with the highest AP of 0.8425 at a threshold of 0.5, showing a steady decline to 0.046 at a threshold of 0.9. This model consistently delivers the best performance across thresholds from 0.5 to approximately 0.74. The Mask R-CNN begins at an AP of 0.823 at a threshold of 0.5. While it ranks second in performance up to a threshold of 0.7, its decline in performance is notably steeper beyond this point, dropping to 0.2768 at 0.8 and going further down to 0.0155 at 0.9.

The transformer-based models show varying performance dynamics. The fine-tuned TransResUNet starts at an AP of 0.7244 at a threshold of 0.5, which surpasses the performance of Mask R-CNN at 0.7 with an AP of 0.623. Notably, TransResUNet exceeds the performance of Att-U-Net around a threshold of 0.74 and maintains superior precision beyond this point. The raw version of TransResUNet shows at start a lower AP of 0.656 at a threshold of 0.5, which declines to 0.0473 at 0.9. Among the SegFormer models, MiT-B5 and MiT-B3

start at APs of 0.573 and 0.563 respectively, with MiT-B3 closely following MiT-B5 and nearly converging in performance from a threshold of 0.7 onward. Swin-UNet has the lowest initial performance with an AP of 0.405 at 0.5, which reduces strongly to 0.009 at 0.9.

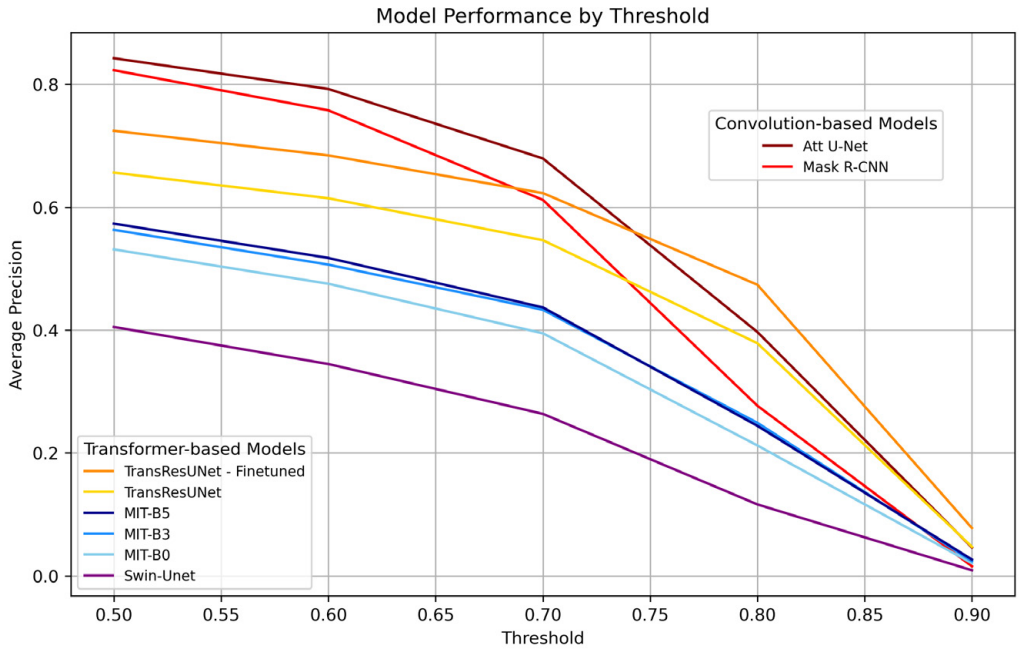


Figure 6.2: Performance comparison of convolutional-based and transformer-based models as a function of the decision threshold.

6.2.2 Comparison of AP DICE Loss and AP Focal Loss

Figure 6.3 shows the Average Precision (AP) versus Intersection over Union (IoU) Threshold for the Att U-Net model, comparing AP DICE Loss and AP Focal Loss. AP Focal Loss consistently demonstrates higher precision across all IoU thresholds. For example, at an IoU threshold of 0.5, AP Focal Loss is approximately 0.8425, compared to 0.8235 for AP DICE Loss. At an IoU threshold of 0.7, AP Focal Loss is 0.6794, while AP DICE Loss is 0.6435.

6.2.3 Precision-Recall Curves

As shown in Figure 6.4, the Attention U-Net shows almost perfect precision up to a recall value of 0.4, after which the precision gradually declines. This is similar to the performance of the Segformer MiT B-3, B-5 and B-0, although this last one has a steeper drop-off in precision. In contrast, the TransUNet has a more linear decline in precision and the Swin U-Net has a more irregular curve. The Mask R-CNN initially has a drop in precision, but then recovers around a recall value of 0.4. The Mask R-CNN does however not regain the same

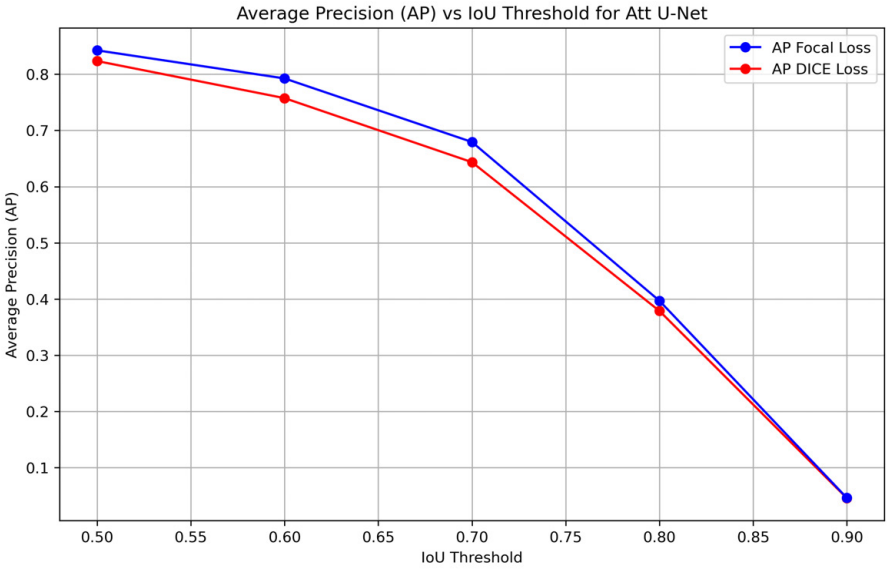


Figure 6.3: Average Precision (AP) vs IoU Threshold for Att U-Net

high precision as the other models. The Area Under Curve (AUC) scores for these models range from 0.93 for Mask R-CNN to 0.97 for both the Attention U-Net and Segformer MiT B-3.

Figure E.2 presents the overlaid precision-recall curves for each model, in which each model shows its distinct characteristics. In particular, their ability to maintain precision across varying levels of recall is shown. Notably, models like Att U-Net and Segformer MIT B-3 demonstrate superior performance and maintain higher precision over a broader range of recall, which is especially evident in the middle recall ranges (0.2 to 0.8).

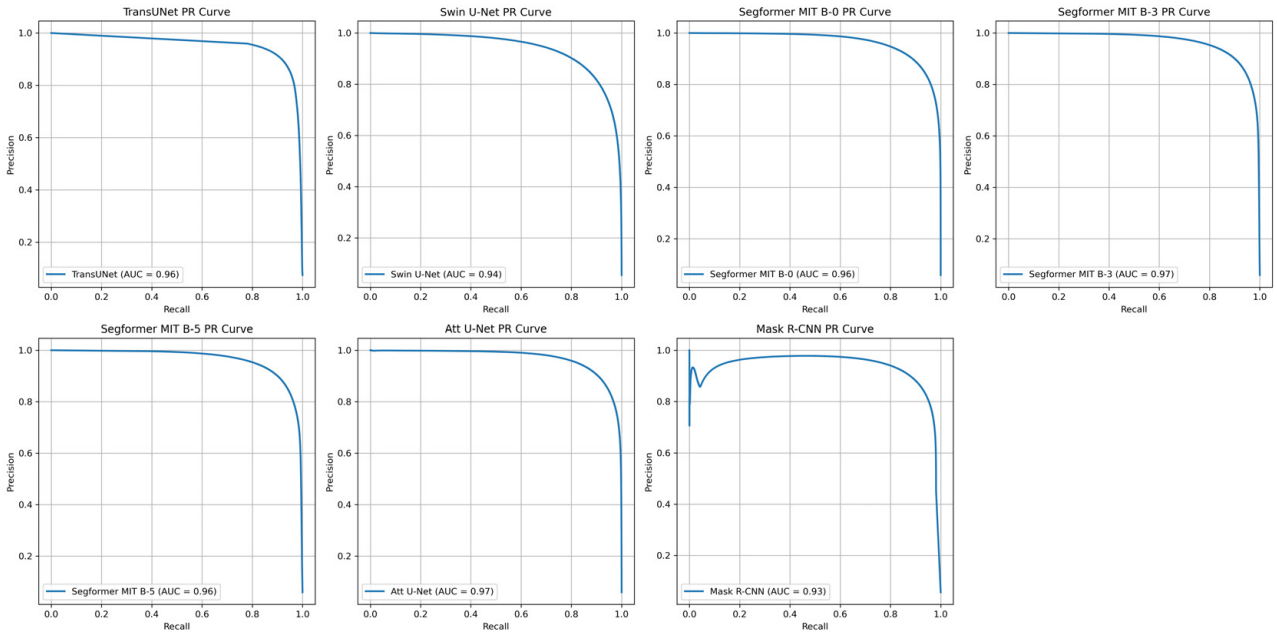


Figure 6.4: Precision-recall curves for different models.

6.3 Model Performance Visualization

6.3.1 Visual Analysis of Segmentation Model Outputs

In the outputs presented in Figure E.5, the Attention U-Net model is highly capable in identifying distinct shapes and very small objects with minimal noise. This is, for example, evident in the third image where cell clumps are well separated and clearly defined. In contrast, the Mask R-CNN has difficulties with accurately delineating borders when cells are clumped together, although it has the ability to identify cells. The segmentation by Mask R-CNN tends to produce broader and less precise borders, and hereby merges smaller details into larger blobs. This is especially noticeable in complex scenarios, such as in the fourth image where it incorrectly identifies a cell within a larger cell as a single unit.

Figure E.6 shows the performance of the transformer-based models. The Swin U-Net has difficulties with smooth border delineation, which leads to noisy output resulting in a generally disorganized segmentation. However, the Swin U-Net is capable of identifying smaller objects more effectively than the Mask R-CNN. The Segformer MiT B-0 improves slightly on edge refinement over the Swin U-Net, but still does not capture very fine details and does not accurately define borders between neighboring cells. The Segformer MiT B-3 shows enhancement in cell border delineation in dense areas. The TransResUNet generates unnecessary artifacts and overly rounded edges, not aligning with the sharper edges of the original ground truth; however, its fine-tuned version shows reduced noise and improved handling in less dense areas, although it continues to struggle with detailed segmentation in more densely areas.

6.3.2 Feature Visualization

Figure 6.5 shows that the Attention U-Net maintains clear edges between cells, whereas other models show a reduced edge definition. The TransResunet-F shows the least distinct separation. In Figure 6.6, the Attention U-Net outlines the inner cell edges effectively and SF MiT-B0 detects an indentation at the top (position 1). TransResUnet captures the filopodia at position 2, whereas Mask R-CNN shows a thicker delineation of filopodia pixels at position 3. Figure 6.7 shows spiky features labeled as 2, 3, and 4. Attention U-Net and Mask R-CNN produce more defined spiky masks. TransResunet and TransResunet-F detect feature 1 but

struggle to accurately represent the spiky features, which results in broader representations.

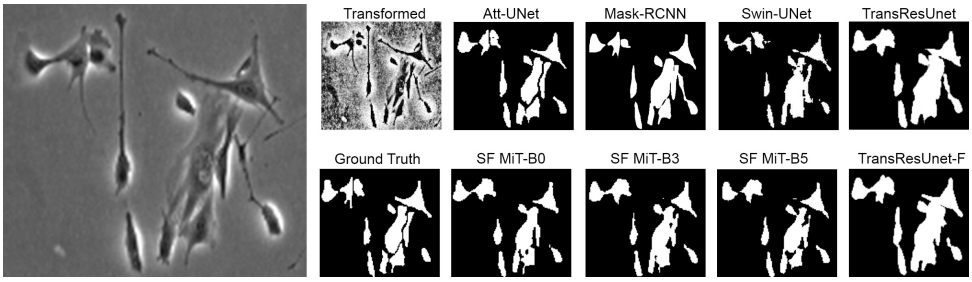


Figure 6.5: The leftmost image is the original input. Moving right, the first image is the transformed version, followed by binary masks produced by various models: Attention U-Net, Mask R-CNN, Swin-UNet, TransResUnet, SF MiT-B0, SF MiT-B3, SF MiT-B5, and TransResUnet-F. Notably, the Attention U-Net maintains clear and precise edges between cells, while subsequent models show a decline in edge precision, with TransResUnet-F exhibiting the least distinct separation between cells.

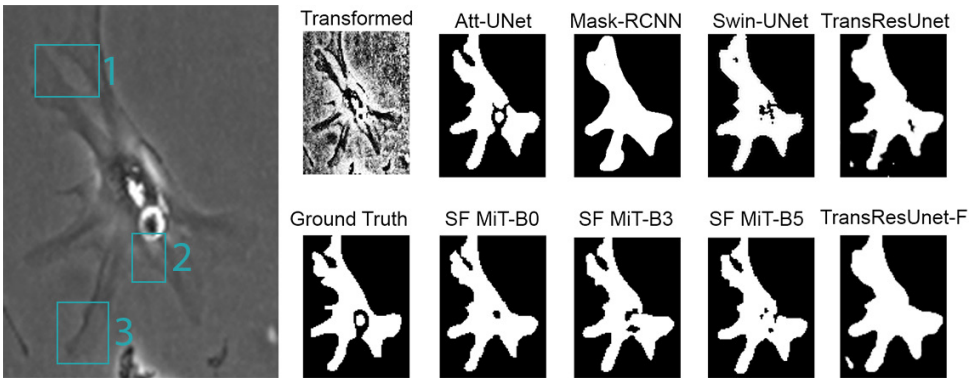


Figure 6.6: The leftmost image is the original input. To the right, the first image shows the transformed version, followed by binary masks from various models: Attention U-Net, Mask R-CNN, Swin-UNet, TransResUnet, SF MiT-B0, SF MiT-B3, SF MiT-B5, and TransResUnet-F. Attention U-Net delineates the edge within the inner cell. SF MiT-B0 captures the indentation at the top (position 1), and TransResUnet segments the filopodia at position 2. Mask R-CNN displays a thicker capture of filopodia pixels at position 3.

6.3.3 Out-of-training Distribution Inference

In Figures E.7 and E.8, five out-of-training distribution images were gathered from datasets 01 - 090303 and 02 - 090318 to assess generalizability, with inference performed using Attention U-Net and TransResUnet-F. A visual analysis was performed, since binary masks were unavailable. Figure 6.8 shows that TransResUnet-F excels at segmenting closely interconnected cells, while Attention U-Net shows many holes and lacks inter-connectivity. This demonstrates that TransResUnet-F has the capability to effectively model overlapping cells. In Figure 6.9, Attention U-Net, shows numerous holes, particularly at position 1, in what should be a closely packed cluster. This leads to a less accurate representation of cell

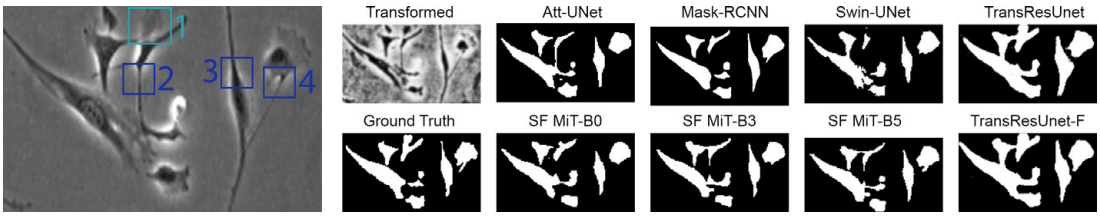


Figure 6.7: The leftmost image is the original input. To the right, the first image shows the transformed version, followed by binary masks from various models: Attention U-Net, Mask R-CNN, Swin-UNet, TransResUnet, SF MiT-B0, SF MiT-B3, SF MiT-B5, and TransResUnet-F. Feature 1 is captured by TransResUnet and TransResUnet-F. Features 2, 3, and 4 represent spiky (pointy) features. Attention U-Net and Mask R-CNN generate more pointy masks, while TransResUnet shows difficulties in accurately representing these features.

density. TransResUnet-F provides a more densely packed image, which accurately reflects the true cell density and captures the larger rounded cell body, which is missed by Attention U-Net. Finally, in Figure 6.10, Attention U-Net has difficulties with border detection and introduces several holes in the oval-shaped cluster, which is absent in the TransResUnet-F output. The light-blue oval in the original image shows rounded clustering, which is effectively detected by TransResUnet-F. This demonstrates the model's superior performance in maintaining the structure. Additionally, in the third row of Figure E.8, TransResUnet-F outputs extremely white binary masks, while Attention U-Net shows more densely packed images. However, TransResUnet-F shows significantly higher density in these areas. In general, Attention U-Net shows a smoother flow of cells due to the presence of cell-to-cell borders. This allows for long cells and flow. TransResUnet-F presents a more chaotic picture. In the first row of Figure E.7, there are more individual blobs and in the third row, TransResUnet-F shows greater inter-connectivity. In the fourth row, TransResUnet-F shows much higher density in the top-right area. Attention U-Net shows significantly less density, which does not accurately correspond with the original image.

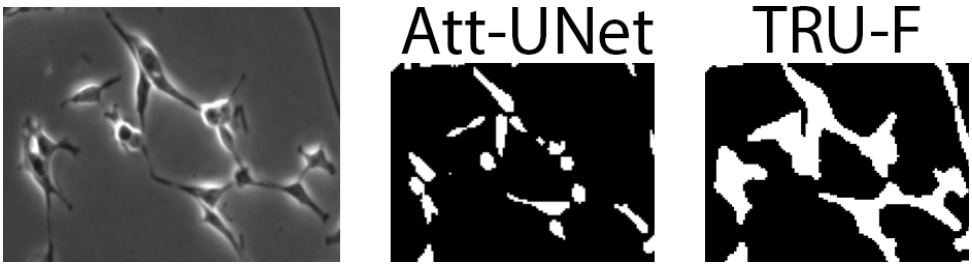


Figure 6.8: Comparison of cell interconnectivity: Original Image (left), Att-UNet (middle), and TRU-F (right).

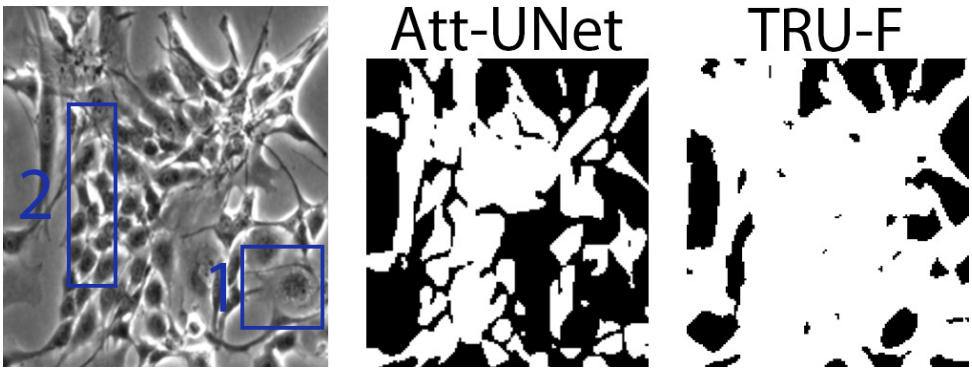


Figure 6.9: Comparison of cluster density: The original image (left) shows dense cell clusters with marked regions. In region 1, a feature is visible that is not visualized by Att-UNet (middle). Region 2 highlights a lot of open space in Att-UNet's output, whereas TransResUnet-F (right) shows these areas as completely packed.

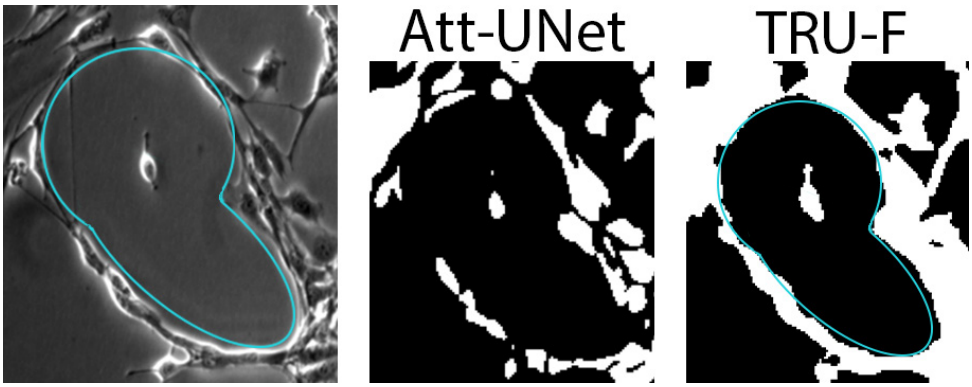


Figure 6.10: Comparison of clustering and border detection: The original image (left) features a 'sea-green oval' highlighting rounded clustering. This global border is effectively picked up by TransResUnet-F (right). In contrast, the Att-UNet (middle) shows significant open spaces and fails to capture this global structure.

6.4 Object Detection Using Mask R-CNN

Figure E.4 shows the results of applying non-max suppression (NMS) in the Mask R-CNN model, which effectively reduced multiple bounding boxes to fewer representations. However, this method has some limitations: the model sometimes fails to detect larger cells, such as the one in the right corner of the third image and another at coordinates (800, 300). Next to this, the model still predicts bounding boxes in areas where there are no cells, such as the top right corner of the third image — initially more numerous before NMS - which suggests false positives. In addition, the model tends to merge closely positioned cells into a single detection. This is evident in the second image where three small cells are identified as one. Despite these challenges, the model performs commendable in areas with less dense cell distributions, successfully identifying individual cells.

Chapter 7

Discussion

7.1 Model Performance Analysis

The Attention U-Net demonstrates performance very close to the ground truth, indicating its strong capability to mimic the data distribution (as discussed in Section 6.3.2). Section 5.2.1.0.1 noted that the images were gathered sequentially, leading to validation data having structures that are similar to the training data, although not identical. This sequential nature allows models like Attention U-Net and Mask R-CNN to closely replicate the training data, as evident by their high average precision, calculated against the ground truth mask.

7.2 Transformer Model Capabilities

In Figure 6.6, the Attention U-Net fails to segment the bottom filopodia at position 2, which is present in the original image. This indicates that it may have mimicked training data sequences that also lacked this segmentation, which is due to the inherent inductive biases in CNNs (Section 2.1.1.1) that allows these models to perform well on in-class data. Increasing the training data did not improve results for these models (as discussed in Section 6.1.1, which is opposed to the transformer-based models like TransResUNet, our highest-performing transformer model. Transformer models, which lack these inductive biases, benefit more from larger datasets. For example, pre-training on 1976 examples and then fine-tuning on the original data increased performance by 10.38%, which significantly enhanced the average precision through general training. This improvement highlights the potential of transformer-based models to refine their segmentation capabilities with more data and computational resources by blending general and specific datasets (see discussion in Section

1.4.4).

7.3 Detailed Feature Detection

Figure 6.7 shows that the transformer models are good at identifying cell structures. An example can be seen in the top filopodia (1), which was missed by other models. However, transformer models seem to struggle with precise segmentation, indicated in features 2, 3 and 4, where they seem to render fine structures too boldly. This is partly due to their capacity to capture long-range dependencies, which makes them focus on global context over local details (Section 2.1.1.2). Despite this, the overlap with the original images indicates that transformer models have good generalization capabilities, even if the masks are not perfectly accurate. It is interesting to observe how transformer models could further improve their segmentation when given access to more substantial data, more computational resources, and a mix of general and specific datasets. This approach shows the possibilities for enhancing the precision and effectiveness of these models in complex segmentation tasks.

7.4 Comparative Analysis of CNN and Transformer Models

When we compare Mask R-CNN with Attention U-Net, it achieves the second-best average precision, despite that it has about 10 million more parameters and 0.4375s longer inference time. Mask R-CNN produces the most crisp boundaries and can output pointy structures effectively. Given the small training set of 384 images, Mask R-CNN's performance is notable. A key advantage of Mask R-CNN over U-Net architectures is its ability to work with the original image size without cropping, unlike Swin U-Net, TransResUNet, and Attention U-Net, which require input images to be cropped to 512x512. Additionally, TransResUNet does not perform well with transformed images, unlike Attention U-Net, which handles them effectively. This suggests that TransResUNet might work better with reshaped rather than transformed images.

7.5 Practical Considerations

A practical disadvantage of Mask R-CNN is its output format, which includes a dictionary of labels, masks, and boxes for each instance, complicating the process as each instance

comes with its own box, label, and segmentation mask. The Non-Maximum Suppression (NMS) threshold has proven to be useful in reducing the number of output instances in a given space.

Swin-UNet, the worst performing model when considering average precision, is less affected by small objects and clutter because the average precision equation (Eq. 5.8) includes false positives in the denominator. Because of this, models that output many small objects inherently will score lower. Despite this, Swin-UNet's Precision-Recall curve shows a PR-AUC of 0.94, indicating that the precision remains high across varying thresholds.

7.6 Insights on Loss Functions and Optimization

Focal loss has proven to be an effective loss function, used in both Attention U-Net and Swin-UNet, with Adam or AdamW commonly serving as optimizers.

7.7 Factors Influencing Border Detection

Attention U-Net shows a superior capability in the detection of borders, which can be attributed to several factors. Firstly, the inherent inductive biases in CNNs, as discussed in Section 2.1.1.1, make these models particularly adapted to capture spatial hierarchies and local patterns, which is crucial for edge detection. The attention mechanism (Section 2.2.1) further enhances the model's ability to focus on important regions, which improves border delineation. Skip connections in U-Net architectures (Section 3.3.1.1.2) also help retain high-resolution features from the encoder to the decoder, which aids in precise border detection. On the other hand, transformer models like TransResUNet, which excel at capturing global context (Section 2.1.1.2), might lose some spatial resolution, which impacts their ability to define borders accurately. This distinction highlights the strengths of CNN-based models in tasks requiring detailed spatial information and edge clarity.

7.8 Inductive Biases, Spatial Resolution, and Global Pattern Recognition

The Attention U-Net benefits from the inherent inductive biases in CNNs, such as shift invariance and spatial hierarchy (Section 2.1.1.1). These inductive biases help to precisely detect inter-cell edges, making the model well-suited for tasks like the segmentation of inter-cell borders. This capability is evident in multiple figures, such as Figure 6.5 and the detailed high-level segmentation shown in the first row of Figure E.8. On the other hand, TransResUNet may lose some spatial resolution while effectively capturing global patterns. As discussed in Section 6.3.3, TransResUNet can capture global borders, as seen in Figure 6.10, and effectively model globally interconnected cells, as depicted in Figure 6.8. This is also apparent in the third row of Figure E.7 and the first row of Figure E.8.

7.9 Directions for Future Research

Incorporation of Pre-computed Weight Maps in Segmentation Models

The integration of pre-computed weight maps is a key feature that could potentially improve the performance of segmentation models, particularly when differentiating between adjacent and touching objects. The approach, inspired by the methodologies from the DeepSea model by Zargari et al. (2023) [6] and the U-Net architecture by Ronneberger et al. (2015) [58], leverages a weight map, $w(x)$, defined as:

$$w(x) = w_c(x) + w_0 \cdot \exp\left(-\frac{(d_1(x) + d_2(x))^2}{2\sigma^2}\right), \tag{7.1}$$

Here, $w_c(x)$ accounts for the balance of class frequencies, w_0 enhances the focus on the borders between touching cells, and σ determines the extent of this focus, with a value set to 25 to provide a broader area of emphasis compared to DeepSea's parameter of 5.
During model training, this weight map is incorporated into the loss function to prioritize the accurate segmentation of edges. The specific implementation in the loss calculation is highlighted by the line of code:

```
loss = criterion(masks_preds, true_masks)
       + 5 * criterion(wmap_preds, true_wmaps)
```

```
+ dice_loss(F.softmax(masks_preds, dim=1).float(),
            F.one_hot(true_masks, model.n_classes).permute(0, 3, 1, 2).float(),
            multiclass=True)
```

This formulation assigns a greater significance to edge predictions, indicated by the factor of 5 applied to the loss associated with weight map predictions, emphasizing the model's performance on segmenting cell boundaries. Although not implemented in this thesis, incorporating weight maps into the loss function could enhance transformer-based models, which typically find it more challenging to distinguish cell boundaries compared to Attention U-Net or Mask-RCNN.

Chapter 8

Conclusion

This master thesis aimed to demonstrate the feasibility of training a deep learning model for muscle cell segmentation using freely available compute resources. Reasonable performance was achieved with 480 images and masks. While transformer-based models posed challenges compared to CNNs due to output differences, Attention U-Net achieved the highest Average Precision. Transformer models, which have fewer inductive biases, showed improvements when fine-tuning on a pre-trained model, which was trained on a larger, non-specific dataset. A visual analysis indicated that TransResUNet performed well on dense cell clusters and effectively models inter-connectivity, density, and general borders through global attention and long-term dependency extraction. However, TransResUNet had difficulties with sharp edges and between-cell borders, which highlights the need to address intra-class imbalance, which can possibly be solved by incorporating pre-computed weight maps. Attention U-Net closely mimicked the training data analysis and underperformed on highly packed cells outside its training set.

A comparative analysis showed that Mask R-CNN achieved the second-best average precision and produces crisp boundaries and effective pointy structures. Its notable performance with a small training set, despite its high parameter count, highlights its robustness and ability to handle original image sizes without cropping, unlike Swin U-Net, TransResUNet, and Attention U-Net, which required cropping to 512x512. TransResUNet performed better with non-transformed images. Mask R-CNN produced many false positive bounding boxes, necessitating Non-Maximum Suppression (NMS) to reduce output instances. Visual analysis of segmentation outputs showed that Attention U-Net excelled in identifying distinct shapes and small objects with minimal noise, while Mask R-CNN struggled with border delineation

in clumped cells. Feature visualization showed that Attention U-Net maintained clear edges, while other models showed reduced edge definition, with TransResunet-F showing the least distinct separation.

This study demonstrated that despite lacking inductive biases, transformer-based models, though requiring more data, can accurately find and globally localize features such as filopodia and cell bodies, thus effectively handling live-cell microscopy data and potentially surpassing traditional CNNs. In evaluating the performance of the two types of models, the key question is whether the goal is strict semantic segmentation, where each pixel is assigned to a specific class, or accurate cell detection and delineation. If the focus is on semantic segmentation, TransResUNet outperforms Attention U-Net due to its superior ability to classify each pixel as part of a cell. However, if the objective is precise cell detection and delineation, then Attention U-Net demonstrates better performance.

Bibliography

[1] X. Li et al. Transformer-based visual segmentation: A survey, 2023.

[2] F. H. A. Shibly and L. K. R. Image processing for automatic cell nucleus segmentation using superpixel and clustering methods on histopathological images, 2023.

[3] N. Loewke et al. Automated cell segmentation for quantitative phase microscopy, 2018.

[4] M. V. Boland and R. F. Murphy. A neural network classifier capable of recognizing the patterns of all major subcellular structures in fluorescence microscope images of hela cells. *Bioinformatics*, 17(12):1213–1223, 2001.

[5] A. E. Carpenter et al. Cellprofiler: image analysis software for identifying and quantifying cell phenotypes. *Genome biology*, 7(10):R100, 2006.

[6] A. Zargari et al. DeepSea is an efficient deep-learning model for single-cell segmentation and tracking in time-lapse microscopy. Cell Reports Methods, 2023. Accessed: 2023-06-26.

[7] Y.-H. Cao and J. Wu. A random cnn sees objects: One inductive bias of cnn and its applications, 2021.

[8] R. Azad et al. Medical image segmentation review: The success of u-net, 2022.

[9] O. Oktay et al. Attention u-net: Learning where to look for the pancreas, 2018.

[10] R. Girshick. Fast r-cnn, 2015.

[11] S. Ren et al. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016.

[12] K. He et al. Mask r-cnn, 2018.

[13] K. Erdem. Understanding region of interest — (roi pooling). `https://towardsdatascience.com/understanding-region-of-interest-part-1-roi-pooling-e4f5dd65bb44`, 2020. Accessed: 2024-04-29.

[14] A. Vaswani et al. Attention is all you need, 2023.

[15] J. Devlin et al. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[16] Y. Liu et al. Roberta: A robustly optimized bert pretraining approach, 2019.

[17] A. Radford et al. Improving language understanding by generative pre-training. OpenAI Technical Report, 2018.

[18] A. Radford et al. Language models are unsupervised multitask learners. `https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf`, 2019. OpenAI Technical Report.

[19] A. Dosovitskiy et al. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

[20] J. Chen et al. Transunet: Transformers make strong encoders for medical image segmentation, 2021.

[21] N. Parmar et al. Image transformer, 2018.

[22] H. Hu et al. Local relation networks for image recognition, 2019.

[23] J.-B. Cordonnier et al. On the relationship between self-attention and convolutional layers, 2020.

[24] N. Greenwald et al. Whole-cell segmentation of tissue images with human-level performance using large-scale data annotation and deep learning, April 2022.

[25] M. Lee and et al. Cellseg: a robust, pre-trained nucleus segmentation and pixel quantification software for highly multiplexed fluorescence images. BMC Bioinformatics, 2022.

[26] C. Kempster and et al. Fully automated platelet differential interference contrast image analysis via deep learning. Sci. Rep., 2022.

[27] K. Cutler and et al. Omnipose: a high-precision morphology-independent solution for bacterial cell segmentation. Nature Methods, 2022.

[28] D. Bunk and et al. Yeastmate: neural network-assisted segmentation of mating and budding events in saccharomyces cerevisiae. Bioinformatics, 2022.

[29] N. Dietler and et al. A convolutional neural network segments yeast microscopy images with high accuracy. Nature Communications, 2020.

[30] R. S. Sutton. The bitter lesson. `http://www.incompleteideas.net/IncIdeas/BitterLesson.html`, 2019. Accessed: [Insert the date you accessed this resource].

[31] M. M. Taye. Theoretical understanding of convolutional neural network: Concepts, architectures, applications, future directions, 2023. Accessed: 2024-04-08.

[32] Y. Zhang et al. Deep learning and vision transformer for medical image analysis, 2023. Accessed: 2024-04-08.

[33] C. Wang and et al. CFATransUnet: Channel-wise cross fusion attention and transformer for 2D medical image segmentation, Jan 2024. Accessed: 2024-04-08.

[34] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions, 2016.

[35] J. Ma et al. The multimodality cell segmentation challenge: toward universal solutions, 2024.

[36] J. Ma and B. Wang. Towards foundation models of biological image segmentation. Nature Methods, 2023.

[37] G. Lee et al. Mediar: harmony of data-centric and model-centric for multi-modality microscopy, 2023.

[38] E. Xie and et al. Segformer: simple and efficient design for semantic segmentation with transformers, 2021.

[39] T. Fan et al. Ma-net: a multi-scale attention network for liver and tumor segmentation. IEEE Access, 2020.

[40] Willem. F1/dice-score vs iou. Stack Exchange answer, 2017. Available at: https://stats.stackexchange.com/questions/273537/f1-dice-score-vs-iou/276144#276144 (Accessed: date-of-access).

[41] Kaggle. Sartorius - cell instance segmentation competition, 2021. Available at https://www.kaggle.com/competitions/sartorius-cell-instance-segmentation.

[42] C. Stringer and M. Pachitariu. Cellpose metrics, 2023. Available at https://cellpose.readthedocs.io/en/latest/_modules/cellpose/metrics.html.

[43] S. Jadon. A survey of loss functions for semantic segmentation. In *2020 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*. IEEE, October 2020.

[44] R. Azad et al. Loss functions in the era of semantic segmentation: A survey and outlook, 2023.

[45] F. Milletari et al. V-net: Fully convolutional neural networks for volumetric medical image segmentation, 2016.

[46] D. F. E. Ker et al. Phase contrast time lapse microscopy datasets with human-generated ground truths and computer-aided cell tracking annotations. Scientific Data, 2018. Accessed: September 2023.

[47] Deep Sea Data Portal. Deep sea data repository. https://deepseas.org/datasets/, 2023. Accessed: September 2023.

[48] Vlaams Supercomputer Centrum. Vlaams supercomputer centrum (vsc) - high-performance computing. https://docs.vscentrum.be/index.html, 2024. Accessed: 2024-05-07.

[49] Kaggle. Kaggle: Your machine learning and data science community. https://www.kaggle.com/, 2024. Accessed: 2024-05-07.

[50] A. Paszke et al. Automatic differentiation in pytorch. 2017.

[51] torchvision. https://pytorch.org/vision/stable/index.html. Accessed: 2024-05-07.

[52] M. Abadi et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems, 2016.

[53] K. Team. Keras. https://keras.io, Year of Access.

[54] Segformer. https://huggingface.co/docs/transformers/main/en/model_doc/segformer. Accessed: 2024-05-07.

[55] X. et al. Segformer: Simple and efficient design for semantic segmentation with transformers. Hugging Face model hub, 2024.

[56] E. Xie et al. Segformer: Simple and efficient design for semantic segmentation with transformers. *CoRR*, abs/2105.15203, 2021.

[57] H. Face. Segformer model transformer. https://github.com/huggingface/transformers/blob/main/src/transformers/models/segformer/modeling_segformer.py, 2024.

[58] O. Ronneberger et al. U-net: Convolutional networks for biomedical image segmentation, 2015.

[59] N. Gogoberidze and B. A. Cimini. Defining the boundaries: challenges and advances in identifying cells in microscopy images. *Current Opinion in Biotechnology*, 85:103055, 2024.

[60] B. Kainz. Equivariance and invariance. https://www.doc.ic.ac.uk/~bkainz/teaching/DL/notes/equivariance.pdf, 2021.

[61] J. G. . T. Virdi. Convolutional neural networks explained. https://twopointseven.github.io/2017-10-29/cnn/, 2017. Image source for Convolutional Neural Networks explanation.

[62] S. O. contributors. Which is best for object localization among r-cnn, fast r-cnn, faster r-cnn, and yolo? https://stackoverflow.com/questions/44255411/which-is-best-for-object-localization-among-r-cnn-fast-r-cnn-faster-r-cnn-and, 2017. Accessed: YYYY-MM-DD.

[63] K. Erdem. Understanding region of interest — part 2: Roi align and roi warp. https://towardsdatascience.com/understanding-region-of-interest-part-2-roi-align-and-roi-warp-f795196fc193 2020. Accessed: 2024-04-29.

[64] Y. Liu et al. A survey of visual transformers, 2022.

[65] A. Hatamizadeh et al. Unetr: Transformers for 3d medical image segmentation, 2021.

[66] X. Li et al. Attransunet: An enhanced hybrid transformer architecture for ultrasound and histopathology image segmentation. *Computers in Biology and Medicine*, 152:106365, 2023.

[67] S. Li et al. Medical image segmentation using squeeze-and-expansion transformers, 2021.

[68] A. R. Khan and A. Khan. Maxvit-unet: Multi-axis attention for medical image segmentation, 2024.

[69] J. M. J. Valanarasu et al. Medical transformer: Gated axial-attention for medical image segmentation, 2021.

[70] J. Chen et al. Vit-v-net: Vision transformer for unsupervised volumetric medical image registration, 2021.

[71] Z. Gong et al. Convtransseg: A multi-resolution convolution-transformer network for medical image segmentation, 2022.

[72] H. Cao et al. Swin-unet: Unet-like pure transformer for medical image segmentation, 2021.

[73] A. Hatamizadeh et al. Swin unetr: Swin transformers for semantic segmentation of brain tumors in mri images, 2022.

[74] Y. Xie et al. Cotr: Efficiently bridging cnn and transformer for 3d medical image segmentation, 2021.

[75] A. Hatamizadeh et al. Unetformer: A unified vision transformer model and pre-training framework for 3d medical image segmentation, 2022.

[76] H.-Y. Zhou et al. nnformer: Interleaved transformer for volumetric segmentation, 2022.

[77] G. C. Ates et al. Dual cross-attention for medical image segmentation, 2023.

[78] B. Cheng et al. Per-pixel classification is not all you need for semantic segmentation, 2021.

[79] X. Huang et al. Missformer: An effective medical image segmentation transformer, 2021.

[80] W. Wang et al. Transbts: Multimodal brain tumor segmentation using transformer, 2021.

[81] R. Azad et al. Transdeeplab: Convolution-free transformer-based deeplab v3+ for medical image segmentation, 2022.

[82] A. He et al. H2former: An efficient hierarchical hybrid transformer for medical image segmentation. *IEEE Transactions on Medical Imaging*, 42(9):2763–2775, 2023.

[83] R. Strudel et al. Segmenter: Transformer for semantic segmentation, 2021.

[84] N. Carion et al. End-to-end object detection with transformers, 2020.

[85] B. Dong et al. Solq: Segmenting objects by learning queries, 2021.

[86] T. Prangemeier et al. Attention-based transformers for instance segmentation of cells in microstructures. In *2020 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE, December 2020.

[87] J. Hu et al. Istr: End-to-end instance segmentation with transformers, 2021.

[88] Y. Wang et al. End-to-end video instance segmentation with transformers, 2021.

[89] Y. Fang et al. Instances as queries, 2021.

[90] H. Wang et al. Max-deeplab: End-to-end panoptic segmentation with mask transformers, 2021.

[91] Unknown. How to remember all these classification concepts forever, June 2020.

Appendix A

Chapter 1: Introduction

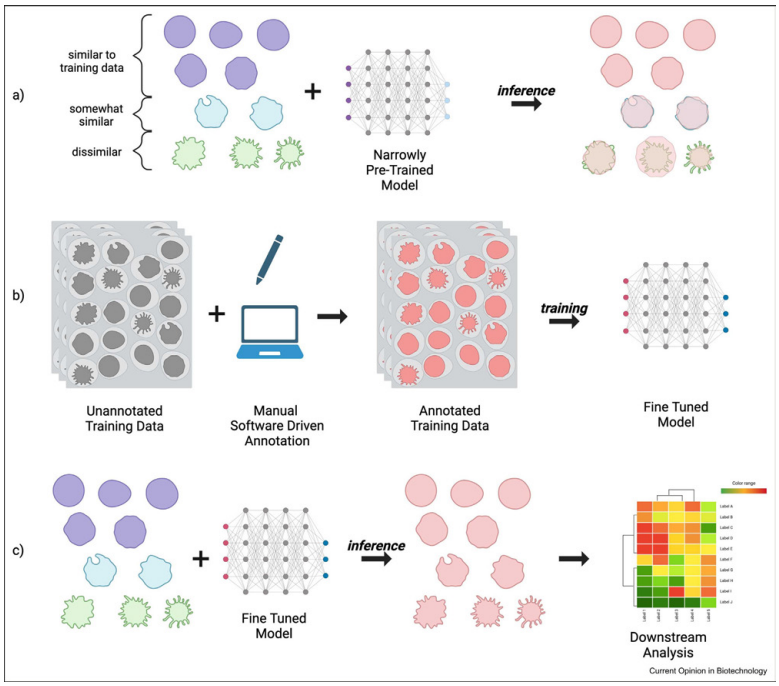


Figure A.1: Segmentation and fine-tuning process in deep learning models for cell imaging [59].

Appendix B

Chapter 2: CNN for Segmentation

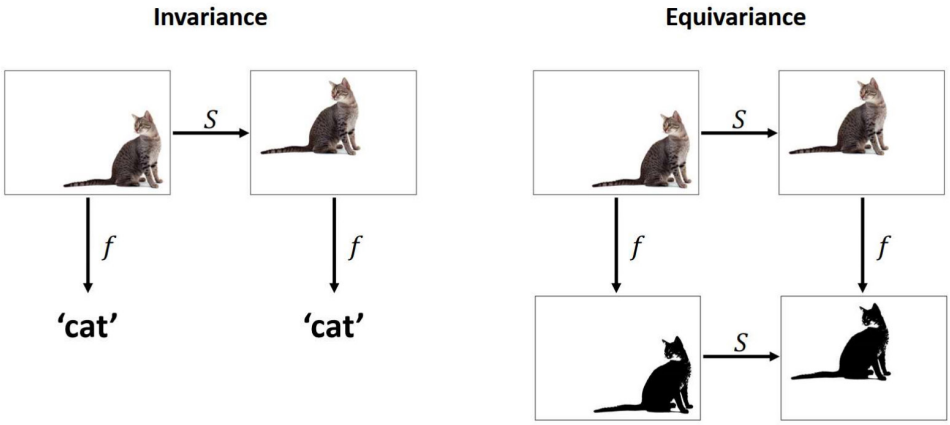


Figure B.1: Comparison of invariance and equivariance in deep learning models [60].

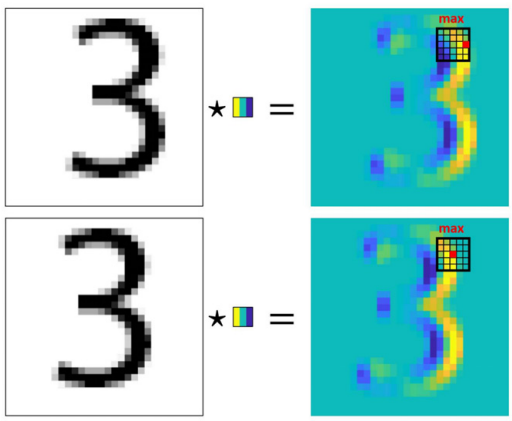


Figure B.2: Demonstration of shift invariance through pooling [60].

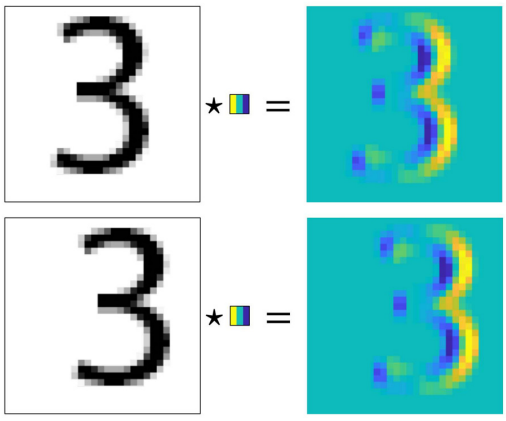


Figure B.3: Illustration of shift equivariance in convolutional operations [60].

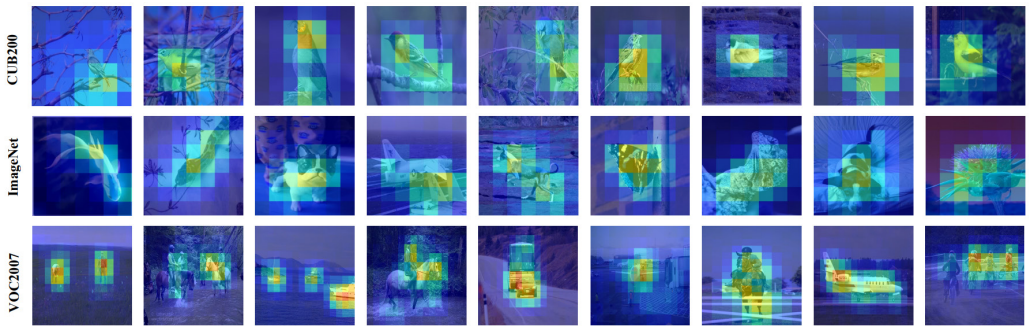


Figure B.4: A representation of heatmap localizations on a ResNet-50 network without prior training [7].

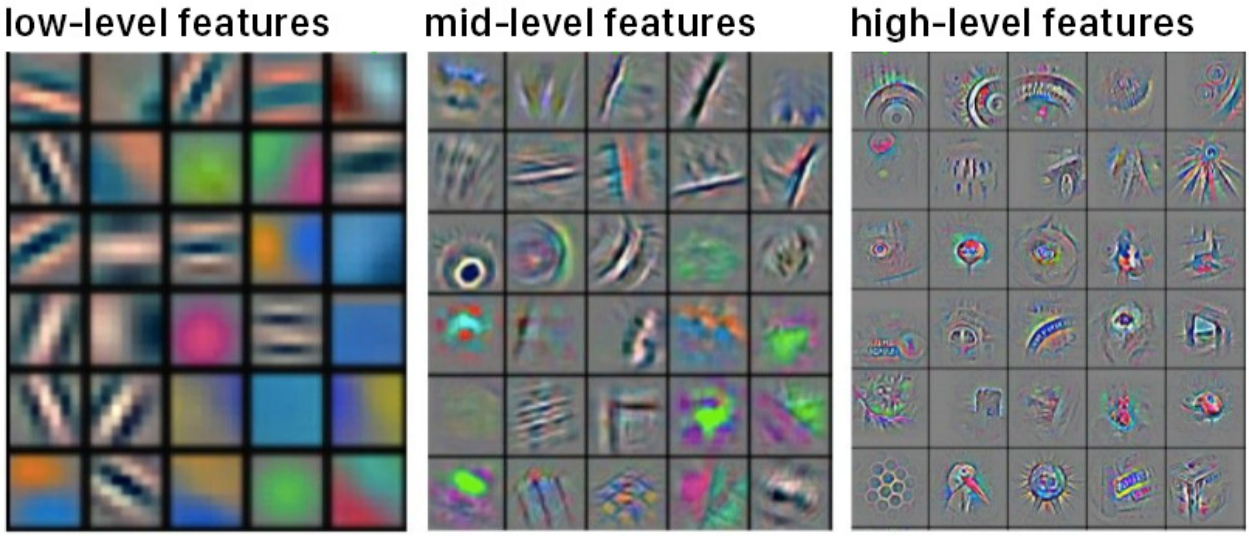


Figure B.5: Visualization of CNN features at different layers: low-level, mid-level, and high-level features [61].

Example of Attention Gates (AG)

An example of how Attention Gates (AGs) operate within a CNN can be illustrated with an example involving different feature map dimensions. Let us consider an input feature map, x^l, originating from the fourth encoder block with dimensions $64 \times 64 \times F_l$ where $F_l = 512$. In parallel, we have a gating signal, g, from the deepest network layer (often referred to as the bottleneck), with dimensions $32 \times 32 \times F_g$ and $F_g = 1024$. The gating signal is processed directly by the attention mechanism without upsampling. The attention mechanism operates as follows:

1. The weight matrix W_x, equipped with F_{int} filters, performs 1×1 convolutions with a stride of 2 on the feature map x^l. This operation downsizes the spatial dimensions of the feature map with F_l channels, resulting in an intermediate feature map x'^l with dimensions $32 \times 32 \times F_{int}$, where $F_{int} = 256$.

Figure B.6: U-Net Taxonomy illustrating the main groups: Transformers, Probabilistic Design, Rich Representation Enhancement, Backbone Design Enhancement, Skip Connection Enhancement, and Bottleneck Enhancement [8].

2. The gating signal g undergoes 1×1 convolutions using a weight matrix W_g, also comprising F_{int} filters. This operation maintains the spatial dimensions and reduces the channel depth, producing an intermediate feature map g' with dimensions $32 \times 32 \times F_{int}$.

3. A bias term b_g is applied to each channel of the feature maps g' and x'^l, providing additional learning parameters for the network.

4. The feature maps g' and x'^l are then added element-wise, resulting in a single feature map of dimensions $32 \times 32 \times F_{int}$.

5. This combined feature map goes through a ReLU activation function σ_1, introducing non-linearity that allows the network to learn complex patterns.

6. The activated feature map is then processed by ψ, involving another set of 1×1 convolutions that compress the F_{int} channels into a single-channel attention map with dimensions $32 \times 32 \times 1$. A bias term b_ψ is added after this convolution.

7. A sigmoid activation function σ_2 is applied to the single-channel attention map to generate the attention coefficients α, which are used to scale the original feature map x^l via

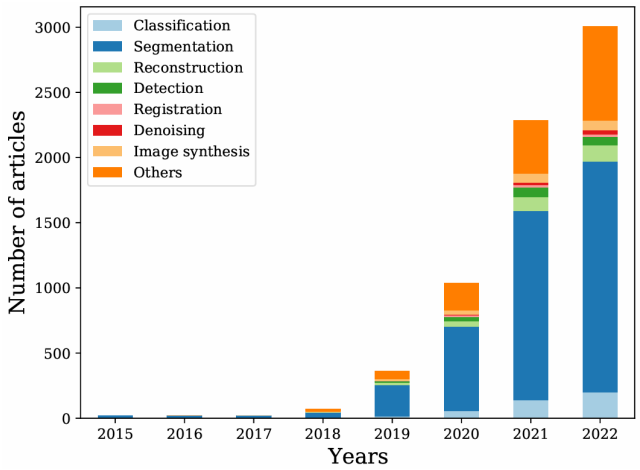


Figure B.7: Bar chart indicating a surge in U-Net-related publications from roughly 100 to over 3000 from 2018 to 2022, signaling its escalating significance in medical image analysis [8].

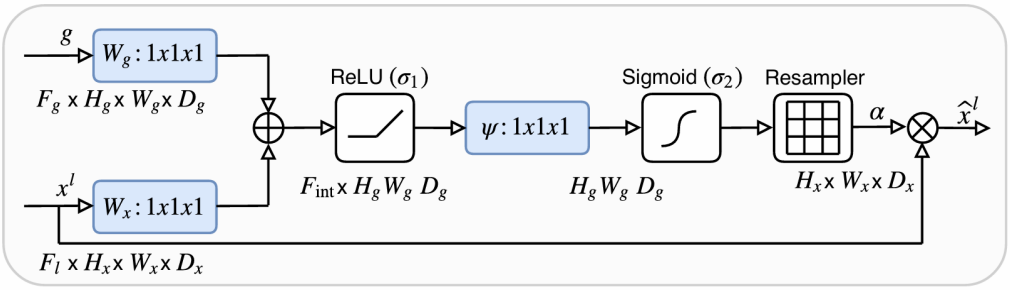


Figure B.8: Overview of the additive attention gate (AG) mechanism within the network. The input feature map (x') undergoes transformation and combination with the gating signal (g), resulting in attention coefficients (α). These coefficients are computed in the AG, where the spatial focus is directed by evaluating activations and contextual inputs. Attention is fine-tuned using trilinear interpolation for grid resampling.

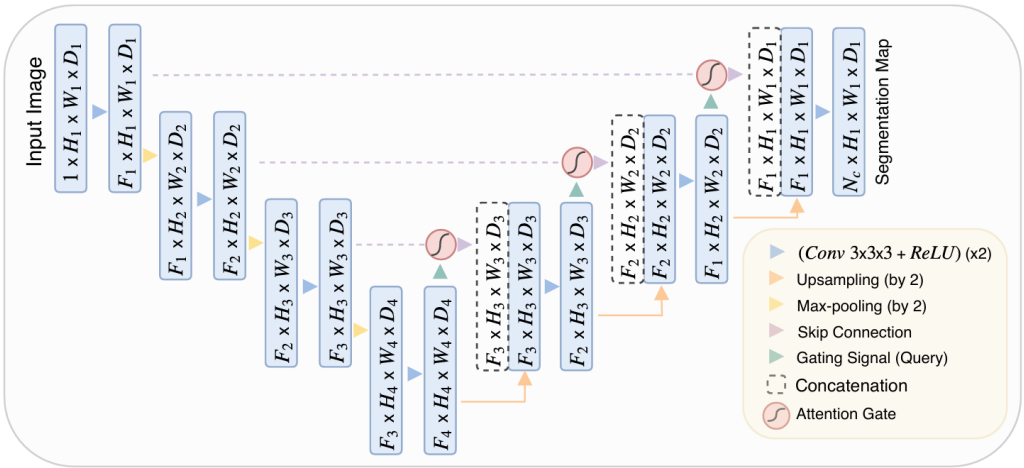


Figure B.9: Illustration of the Attention U-Net architecture for image segmentation. The network progressively downsamples and filters the input image by a factor of 2 at each encoding layer. The number of output classes is denoted by N_c. Attention mechanisms are incorporated to refine feature propagation, with a detailed view provided in a separate figure. Attention is modulated using contextual information from coarser scales of the network.

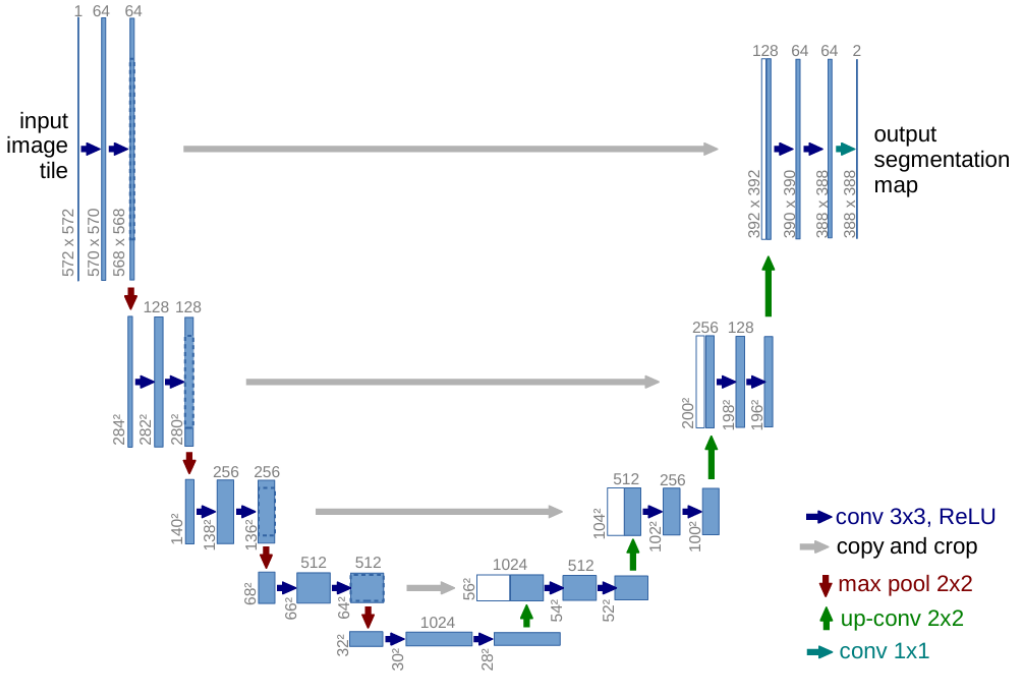


Figure B.10: The U-Net architecture for semantic segmentation with an example resolution pathway for a 32x32 pixels tile at the lowest level. Each multi-channel feature map is represented by a box, with the channel count indicated on top and spatial dimensions at the lower edge. White boxes signify feature map duplication and the arrows indicate various operations such as convolution, max pooling, and up-convolution.

element-wise multiplication, focusing on the regions most relevant for the segmentation task.

The output of the AGs remains at the dimensions $32 \times 32 \times 512$. This modulated output is then upsampled to $64 \times 64 \times 512$ and concatenated with the upsampled features that were previously enlarged from $32 \times 32 \times 1024$ to $64 \times 64 \times 512$. The resulting feature map after concatenation is $64 \times 64 \times 1024$, which combines the features processed by the attention mechanism with high-resolution details from the upsampled path to enhance segmentation accuracy.

$$P \approx \frac{7.5-6.5}{7.5-6.5}\left[\frac{10.5-9.94}{10.5-9.5}\cdot 0.1+\frac{9.94-9.5}{10.5-9.5}\cdot 0.2\right]+\frac{6.5-6.5}{7.5-6.5}\left[\frac{10.5-9.94}{10.5-9.5}\cdot 0.1+\frac{9.94-9.5}{10.5-9.5}\cdot 0.7\right] \tag{B.1}$$

$$\approx 0.14$$

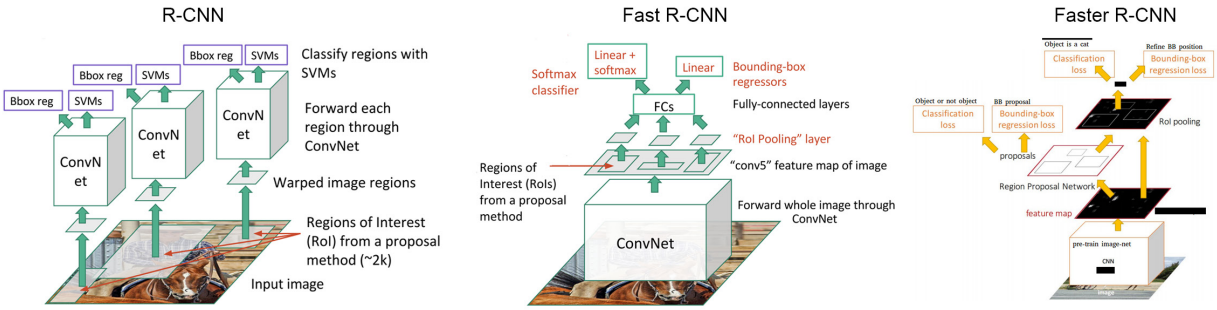


Figure B.11: Evolution of object detection frameworks from left to right: R-CNN utilizes selective search to propose regions and classifies them with SVMs. Fast R-CNN accelerates the process using a shared ConvNet for feature extraction and RoI pooling. Faster R-CNN introduces a Region Proposal Network, eliminating selective search and further speeding up detection [62].

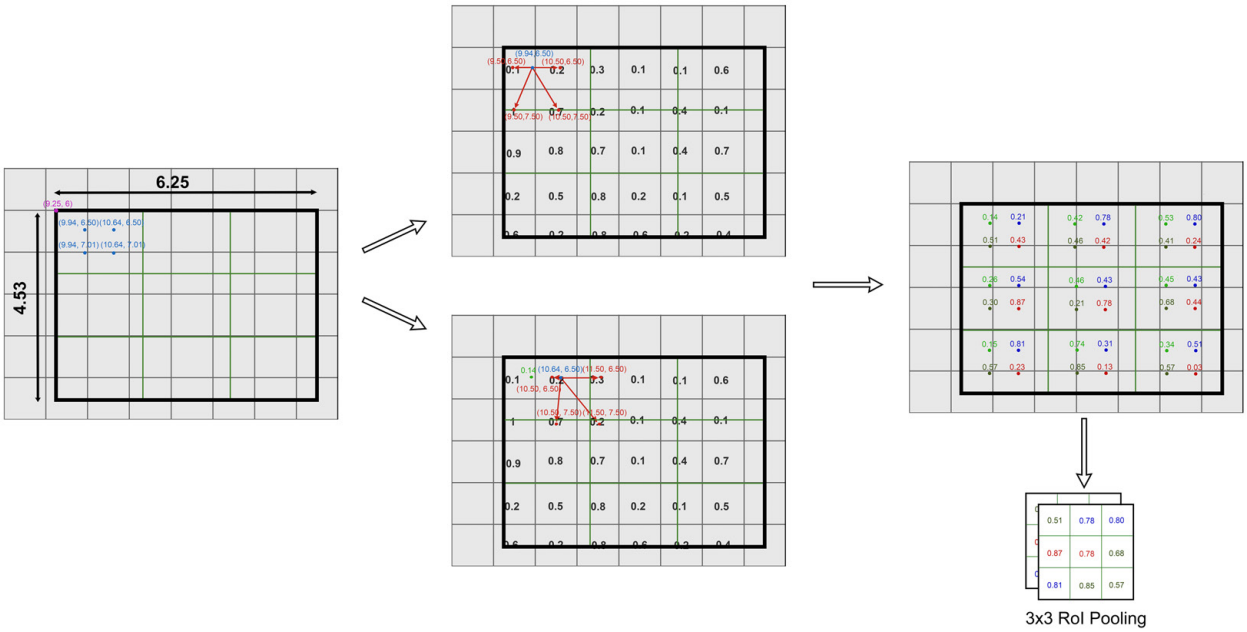


Figure B.12: Illustration of the RoI Align process utilized in the Mask-RCNN model for instance segmentation as well as object detection. The graphic demonstrates the division of a proposed RoI into equal-sized sections, calculation of sampling points, application of bilinear interpolation, and the subsequent max pooling. This technique effectively mitigates the data loss and addition typical of quantization in traditional RoI Pooling, highlighted by the absence of dark blue (data loss) and green (data addition) areas within the interpolated sections. The end result is a $3 \times 3 \times 512$ feature matrix per RoI. Image adapted from [63].

Appendix C

Chapter 3: Transformers for Segmentation

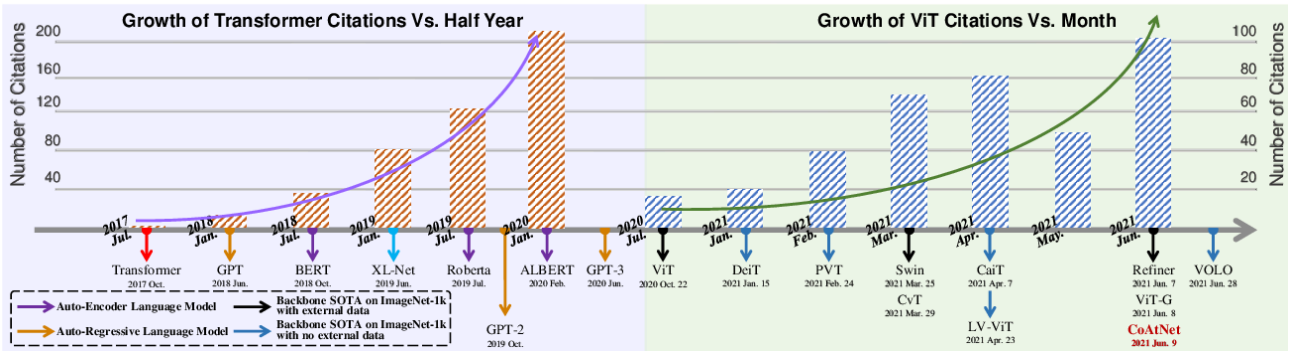


Figure C.1: The growth of Transformer citations over time, indicating their expanding influence across various domains, including computer vision [64].

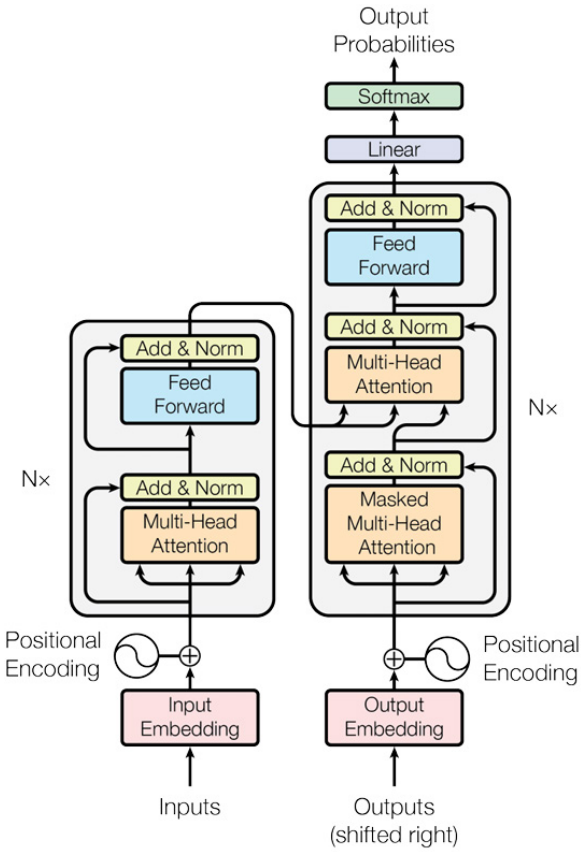


Figure C.2: Transformer encoder-decoder architecture [14].

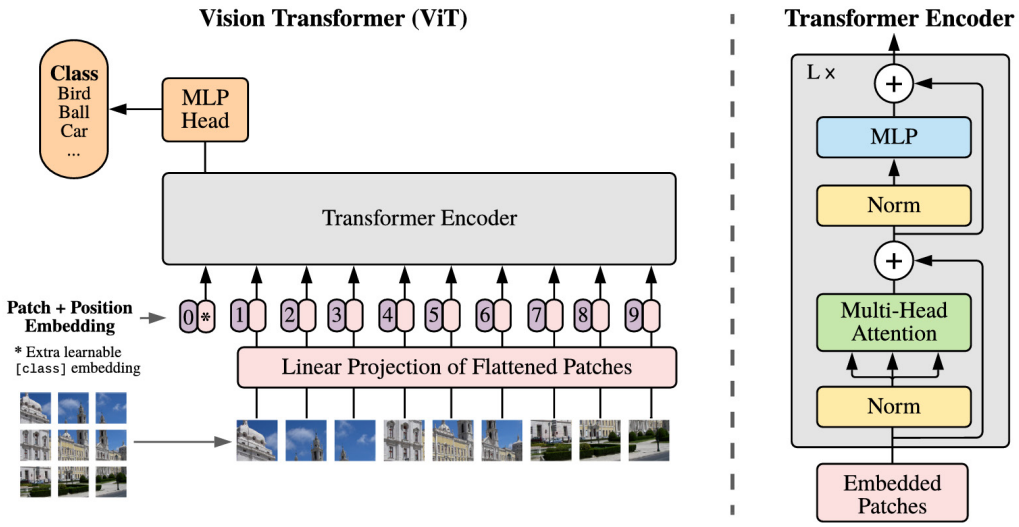


Figure C.3: Vit encoder-only architecture [19].

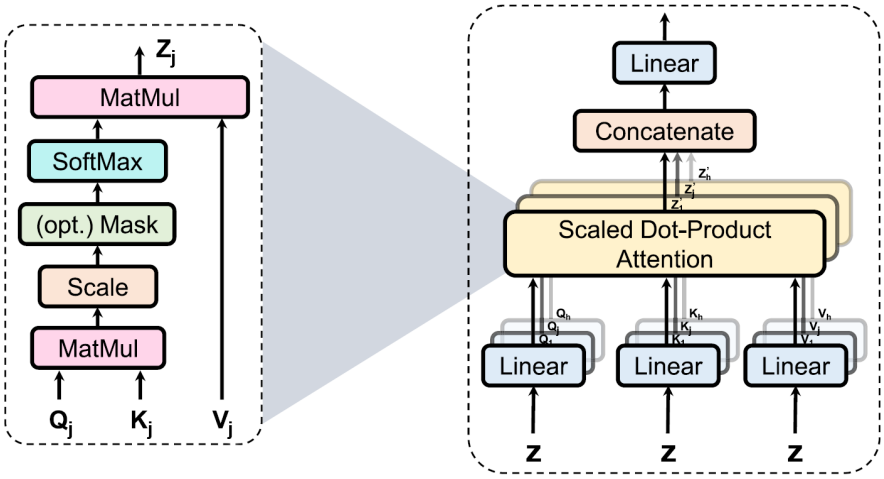


Figure C.4: This figure illustrates the scaled dot-product multi-head attention mechanism employed in a Vision Transformer. Linear transformations are applied to the input matrix Z, generating query Q_j, key K_j, and value V_j matrices for each attention head j. The attention scores are computed through the scaled dot-product of Q_j with K_j, and subsequently normalized using softmax to obtain weighting coefficients. These coefficients weight the value matrices V_j, yielding the interim attention outputs Z'_j. The outputs Z'_j from each head j are then concatenated as per Equation 3.11, culminating in the final output Z', which integrates the attentive information across all heads. (Adapted from Liu et al. [64]).

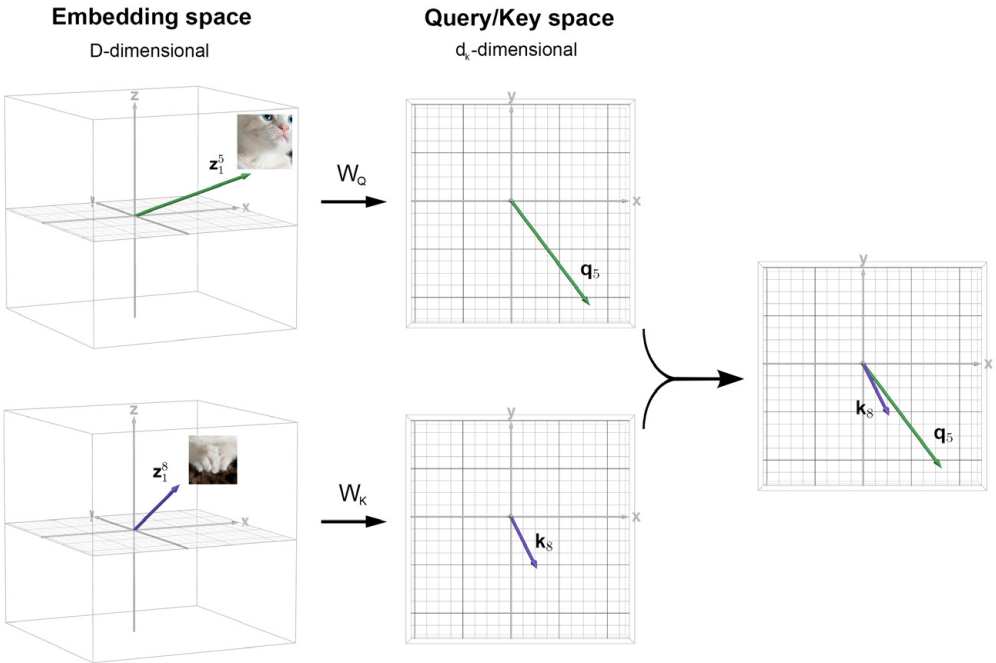


Figure C.5: Projection of patch embeddings into the query/key space in a Vision Transformer. The embedding $\mathbf{z}_1^5$, representing a cat's face, is projected into a d_k-dimensional query vector $\mathbf{q}_5$ using the weight matrix W_Q. Conversely, the embedding $\mathbf{z}_1^8$, depicting the cat's paws, is projected into a key vector $\mathbf{k}_8$ using W_K.

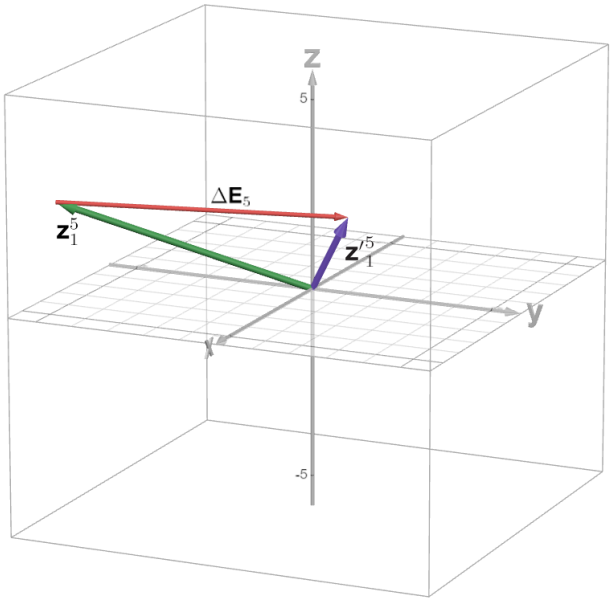


Figure C.6: Illustration of the embedding update process in the attention mechanism. The original patch embedding $\mathbf{z}_1^5$, representing the cat's face, is updated by a weighted sum of all value vectors ($\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_8$) to form $\Delta \mathbf{E}_5$. These weights are derived from the attention scores, ensuring that more relevant patches have a larger influence on the update. The new vector $\mathbf{z}_1'^5$ results from applying this delta change to the original embedding, thereby refining its feature representation. This enhanced embedding encapsulates a richer contextual understanding.

Table C.1: Categorization of Architectures with Transformer blocks (TF blocks for Segmentation)

Vision Transformer Segmentation Approaches				
Approach	TF block(s) in Encoder	TF block(s) in-between Encoder-Decoder	TF block(s) in Decoder	TF block(s) in Both
Patch-Based	UNETR [65]	ATTransUNet [66]	SegTran* [67]	MaxViT-UNet* [68]
	MedT* [69]	ViT-V-Net [70]	ConvTransSeg [71]	Swin-UNet [72]
	Swin-UNETR* [73]	CoTr [74]	UNetFormer* [75]	nnFormer [76]
	TransUNet* [20]	DCA [77]	MaskFormer [78]	MISSFormer [79]
	TransBTS* [80]			TransDeepLab [81]
	H2Former* [82]			Segmenter [83]
	Segformer [82]			
Query-Based	DETR [84]		SOLQ [85]	Cell-DETR* [86]
	ISTR* [87]			VisTR* [88]
				QuerInst [89]
				MaX-DeepLab [90]

Appendix D

Chapter 5: Material and Methods

$$\text{IoU} = \frac{tp}{tp + fp + fn} = \frac{8}{8 + 2 + 5} = 0.533 \tag{D.1}$$

$$\text{IoU} = \frac{Area\ of\ overlap}{Area\ of\ union} = \frac{}{}$$

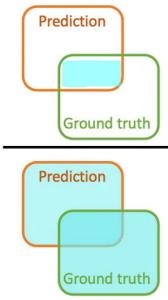


Figure D.1: Graphical representation of the IoU calculation, indicating the overlap and union areas between predicted segmentation and ground truth.

Table D.1: Jaccard Index Calculation Steps with Two Classes

Step	Tensor Representation
Softmax Predictions	`[[0.1, 0.9], [0.8, 0.2]]`
Binary Predictions (tf.argmax)	`[[0, 1], [1, 0]]`
Flattened Binary Predictions	`[0, 1, 1, 0]`
Binary True Labels (tf.argmax)	`[[0, 1], [0, 1]]`
Flattened Binary True Labels	`[0, 1, 0, 1]`
Intersection	$\sum$(Flattened Predictions $\times$ Flattened True Labels) $= 1$
Union	$\sum$(Flattened Predictions) + $\sum$(Flattened True Labels) $-$ Intersection $= 3$
Jaccard Index	$\frac{\text{Intersection}}{\text{Union}+\epsilon} = \frac{1}{3+\epsilon}$

$$D = \frac{2 \times 8}{2 \times 8 + 2 + 5} = \frac{16}{23} \approx 0.695 \tag{D.2}$$

Scenario	Dice Score	IoU
Moderate Overlap	0.64	0.47
Single Misclassification	0.98	0.96

Table D.2: Comparison of Dice Score and IoU for different segmentation scenarios.

Pixel Level	**Region Level**	**Boundary Level**	**Combination**
Cross-Entropy Loss	Dice Loss	Boundary Loss	Combo Loss
TopK Loss	Log-Cosh Dice Loss	Hausdorff Distance Loss	Exponential Logarithmic Loss
Focal Loss	Generalised Wasserstein Dice Loss	Boundary-aware Loss	Unified Focal Loss
Distance map derived cross-entropy loss	IoU (Jaccard) Loss	Active Boundary Loss	
	Lovász-Softmax loss	InverseForm Loss	
	Tversky Loss	Conditional Boundary Loss	
	Focal Tversky Loss	Boundary Difference Over Union Loss	
	Sensitivity Specificity Loss	Region-wise Loss	
	Region Mutual Loss		
	Robust T-Loss		

Table D.3: Categorized list of loss functions used in image segmentation [43][44].

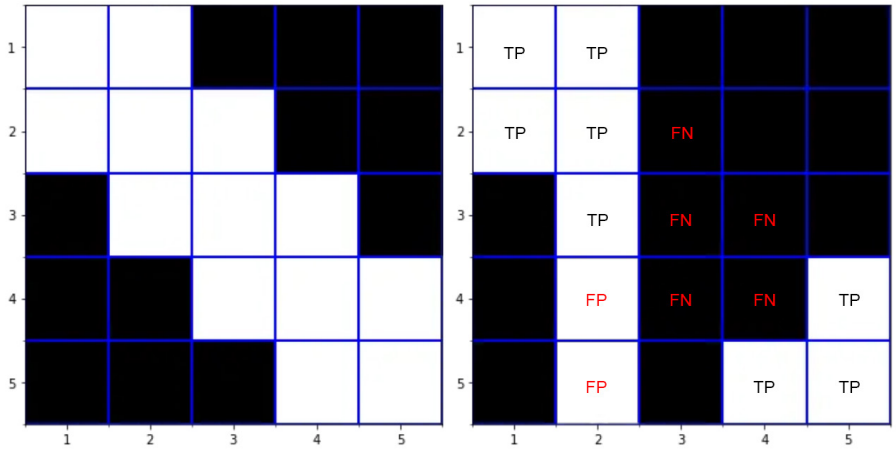


Figure D.2: A 5x5 pixel grid showing the segmentation model's predictions with the region of interest in white. The grid is annotated to highlight True Positives (tp), False Positives (fp), and False Negatives (fn), essential for the calculation of the IoU.

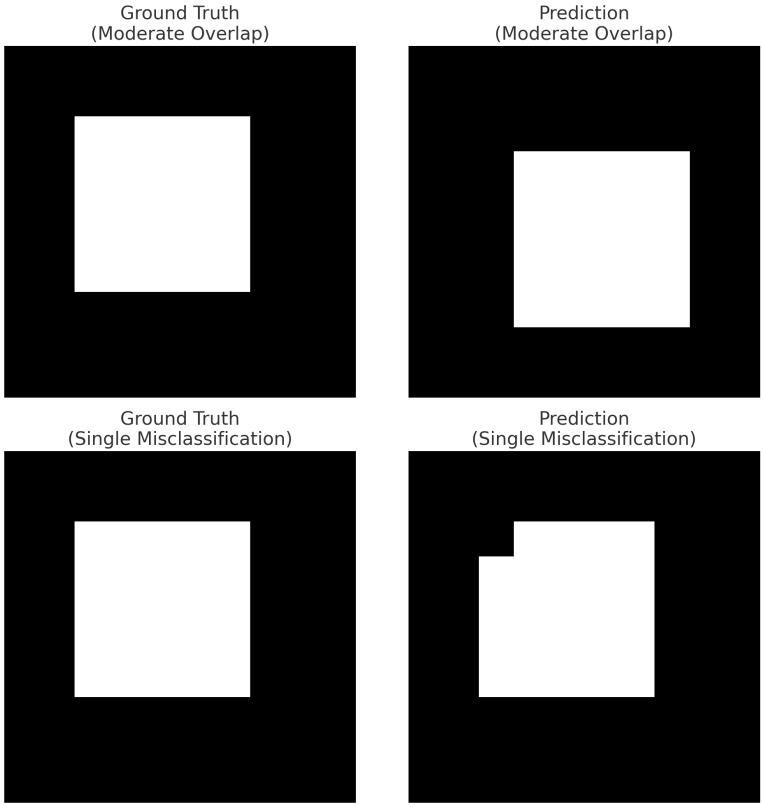


Figure D.3: The left column depicts ground truth segmentations, and the right column shows predicted segmentations. Top row: Moderate overlap case. Bottom row: Single misclassification case.

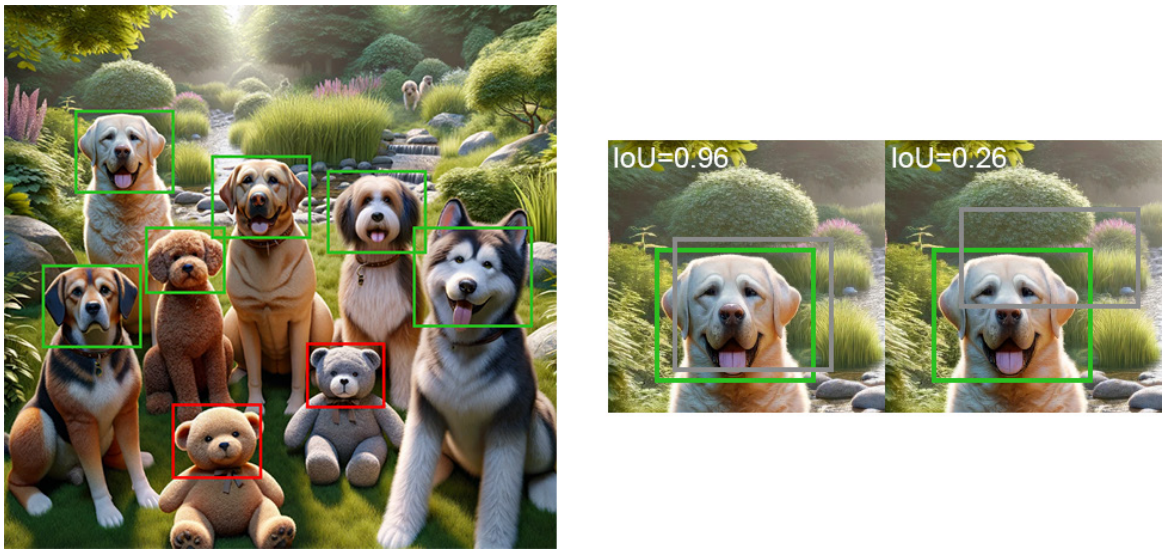


Figure D.4: Left: Object detection results with True Positives (TP) indicated by green bounding boxes and False Positives (FP) indicated by red bounding boxes, where the algorithm successfully identifies several dogs as TP and mistakenly classifies teddy bears as FP. Right: Comparison of object detection results showing two bounding boxes with different IoU scores; the box on the left indicates a high IoU of 0.96, suggesting a strong match with the ground truth, while the box on the right has a lower IoU of 0.26, indicating a less accurate detection.

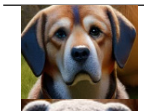
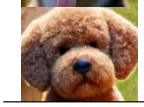
Preds.	Conf.	Matches	Cumulative TP	Cumulative FP	Precision	Recall
	0.92	TP	1	0	1.00	0.13
	0.91	FP	1	1	0.50	0.13
	0.88	TP	2	1	0.66	0.30
	0.86	FP	2	2	0.50	0.30
	0.77	TP	3	2	0.60	0.38
	0.63	TP	4	2	0.66	0.50
	0.58	TP	5	2	0.71	0.63

Table D.4: Precision and Recall for object detection of dogs and teddy bears.

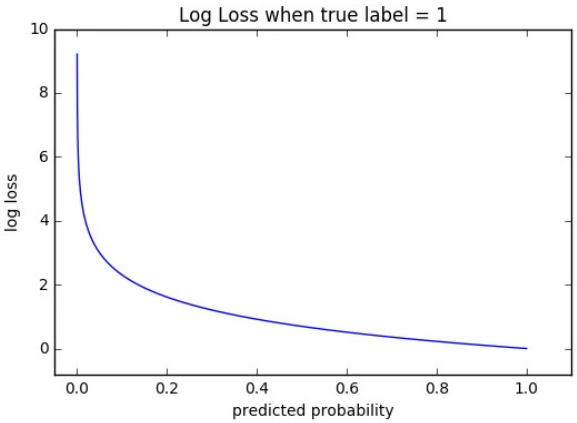


Figure D.5: Log Loss when the true label is 1.

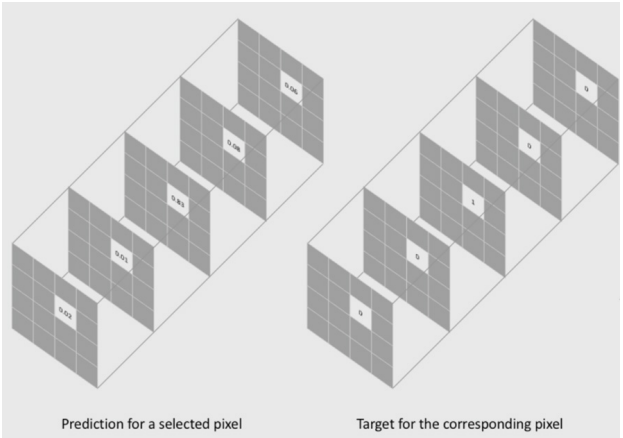


Figure D.6: Prediction and target for a selected pixel.

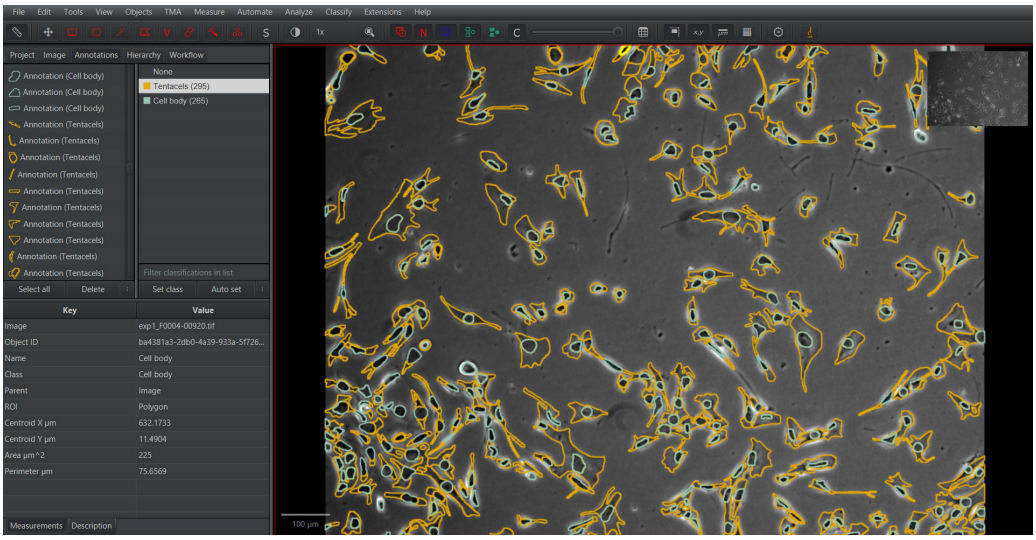


Figure D.7: Annotated Microscopy Image using QuPath. This image displays two primary classes of annotations: ”Tentacles” (filopodia) and ”Cell body”. On the left panel, different images are listed along with tools at the top including the ellipse tool, line annotation tool, and closed polygon annotation tool. In the bottom left corner, the statistics for each polygon are displayed, showing centroid location coordinates, area, image name, and object ID, providing detailed insights into the cellular structures studied.

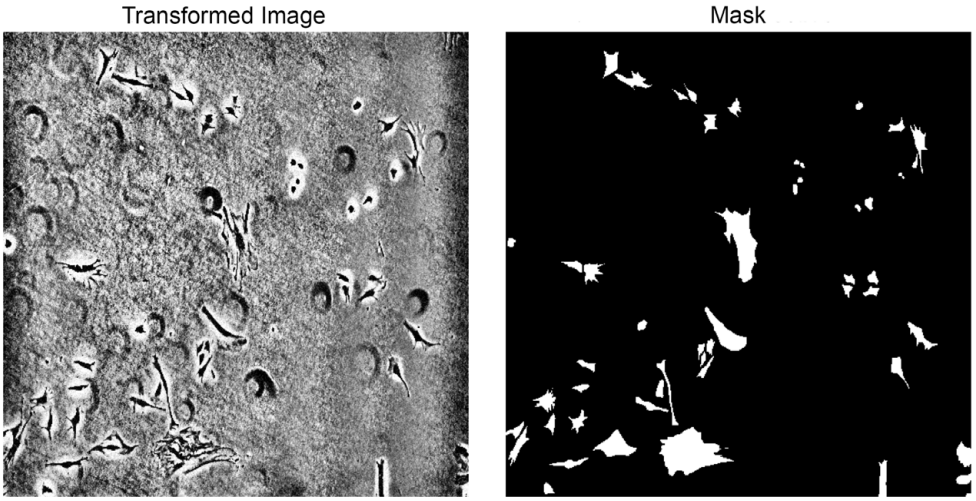


Figure D.8: An example of a 512×512 cell image post-processing with CLAHE (Contrast Limited Adaptive Histogram Equalization), showing enhanced contrast on the left. On the right is the corresponding binary mask highlighting regions of interest.

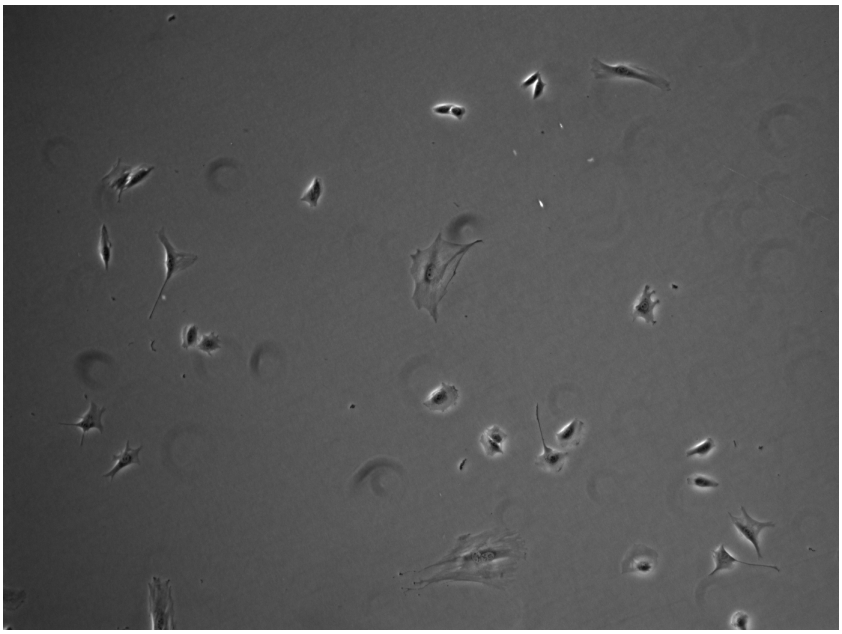


Figure D.9: Phase Contrast Time Lapse Microscopy image of mouse C2C12 muscle progenitor cells.

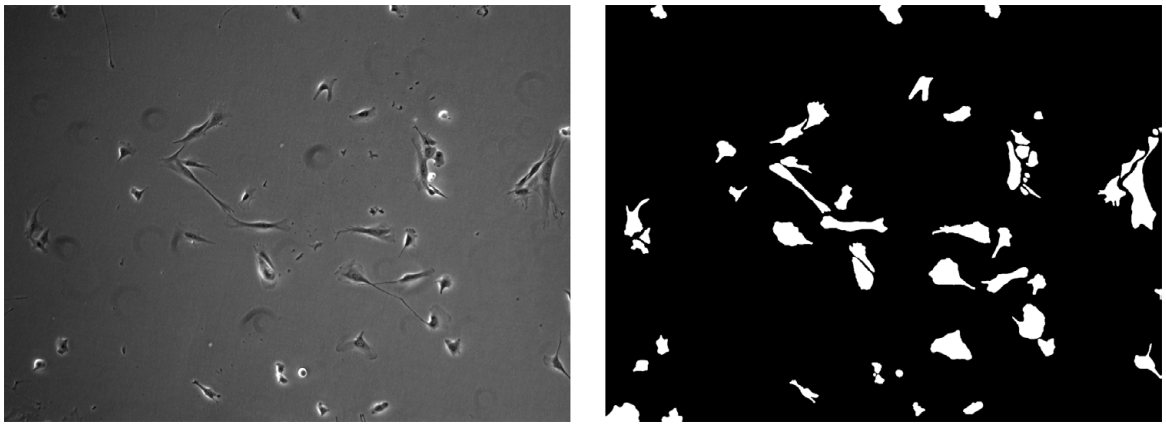


Figure D.10: Phase Contrast Time Lapse Microscopy image of mouse C2C12 muscle progenitor cells (left) and the corresponding binary mask (right).

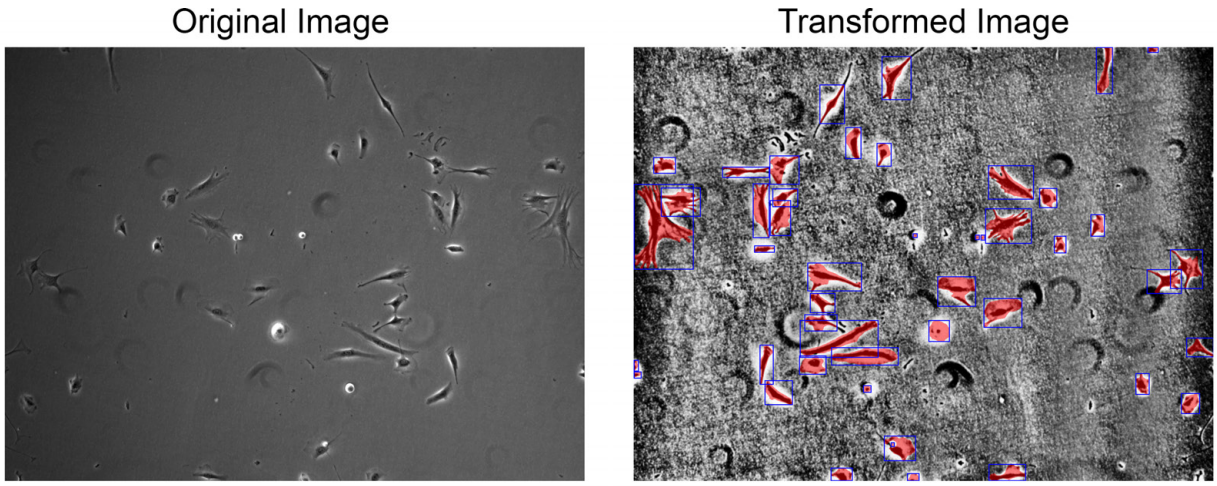


Figure D.11: Left: Original Image. Right: Transformed Image with bounding boxes and mask overlays.

Appendix E

Chapter 6: Results

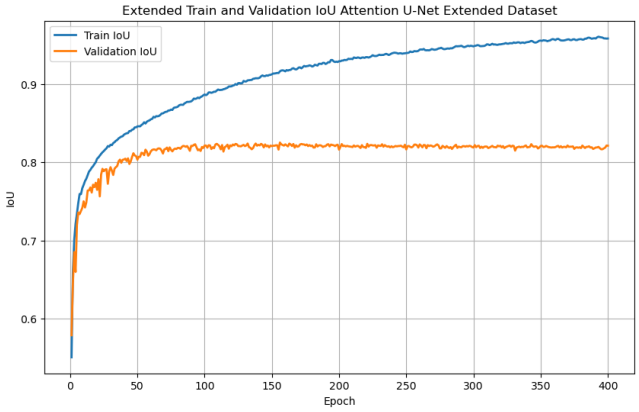


Figure E.1: Graph depicting the training and validation Jaccard scores of the Attention U-Net trained with an extended dataset comprising 2075 images.

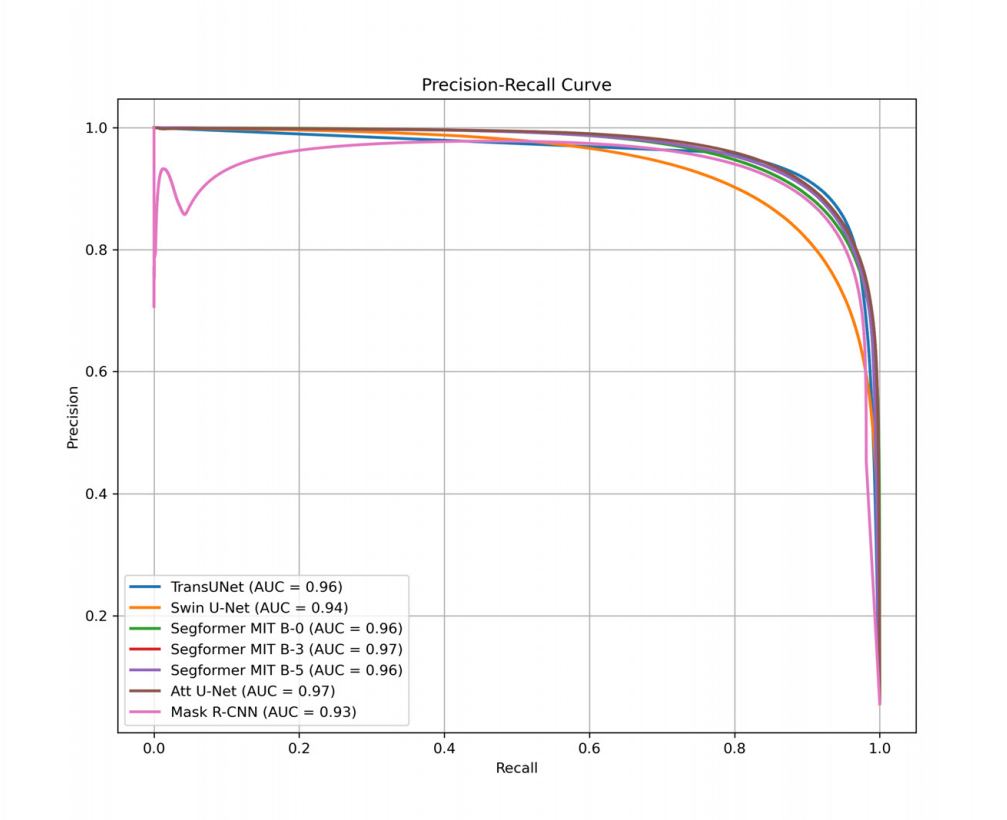


Figure E.2: Precision-recall curves for different models plotted on 1 graph

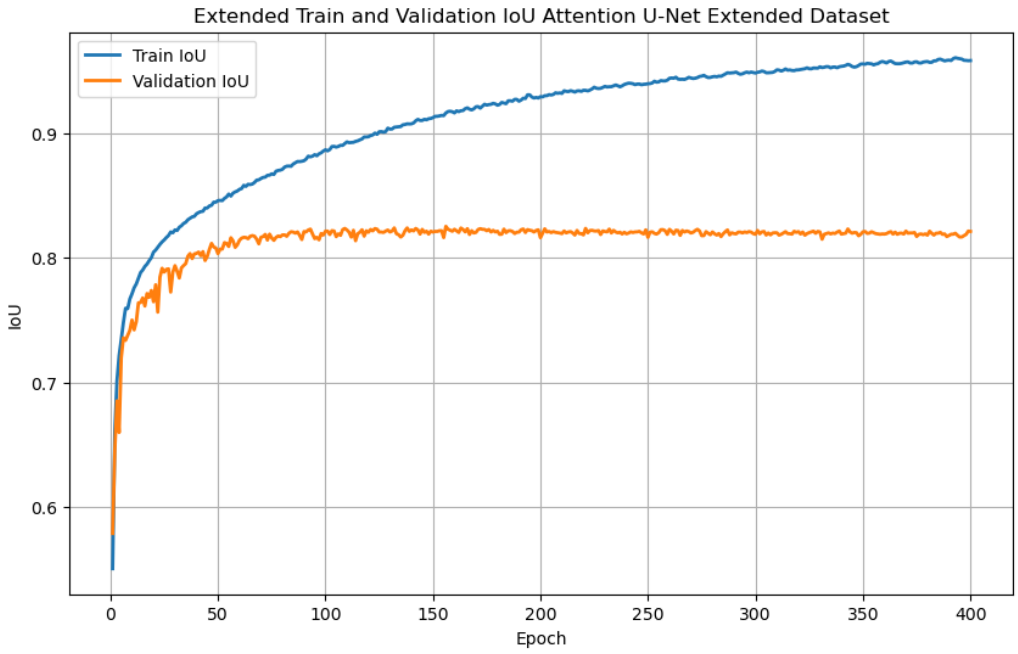


Figure E.3: Extended Train and Validation IoU for Attention U-Net on Extended Dataset

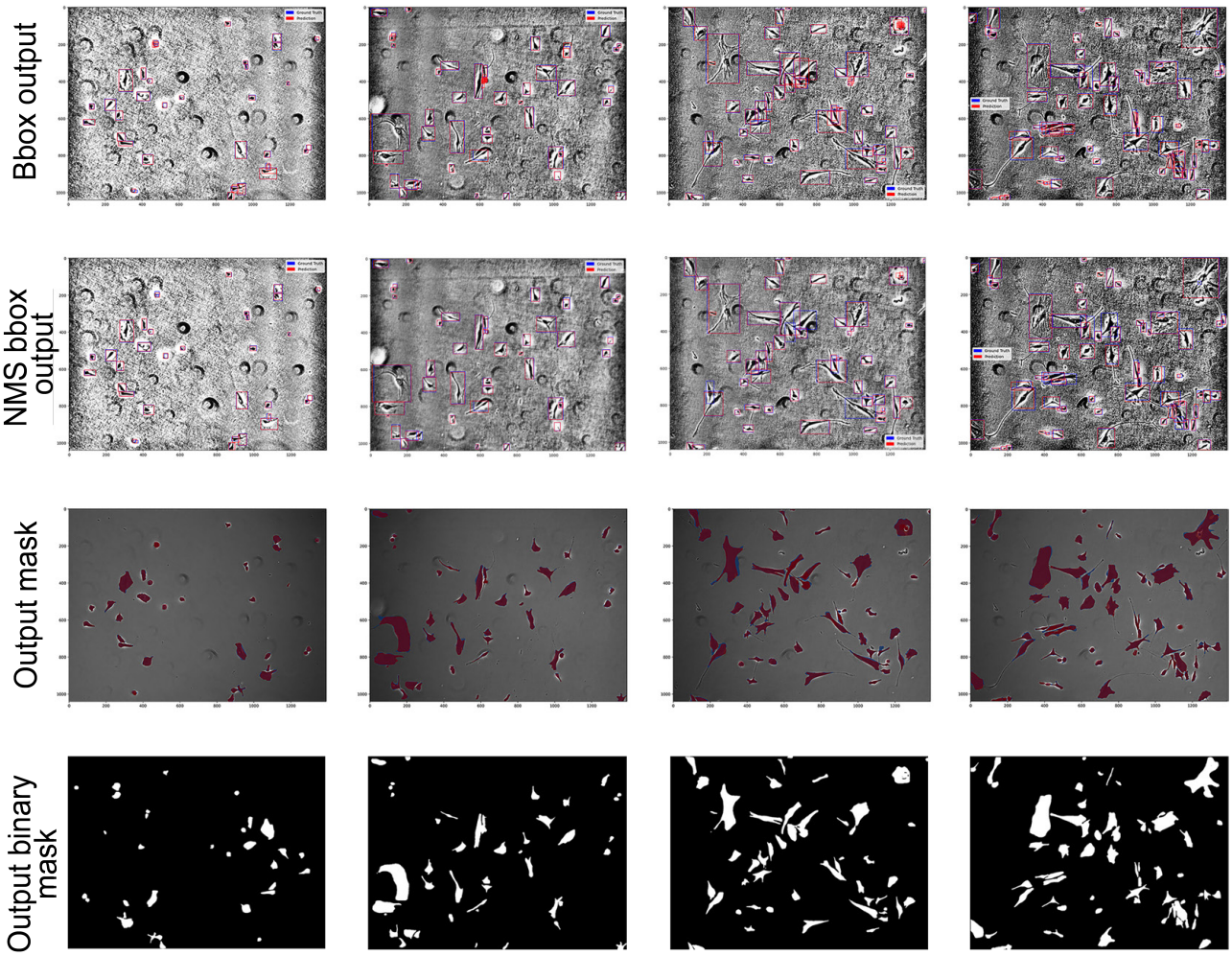


Figure E.4: Analysis of Mask R-CNN outputs. First row: Bounding box (red for predicted output and blue for ground truth) output before applying Non-Maximum Suppression (NMS). Second row: Bounding box output on transformed images with NMS applied. Third row: Predicted masks in red and output masks in blue. Fourth row: Binary mask output.

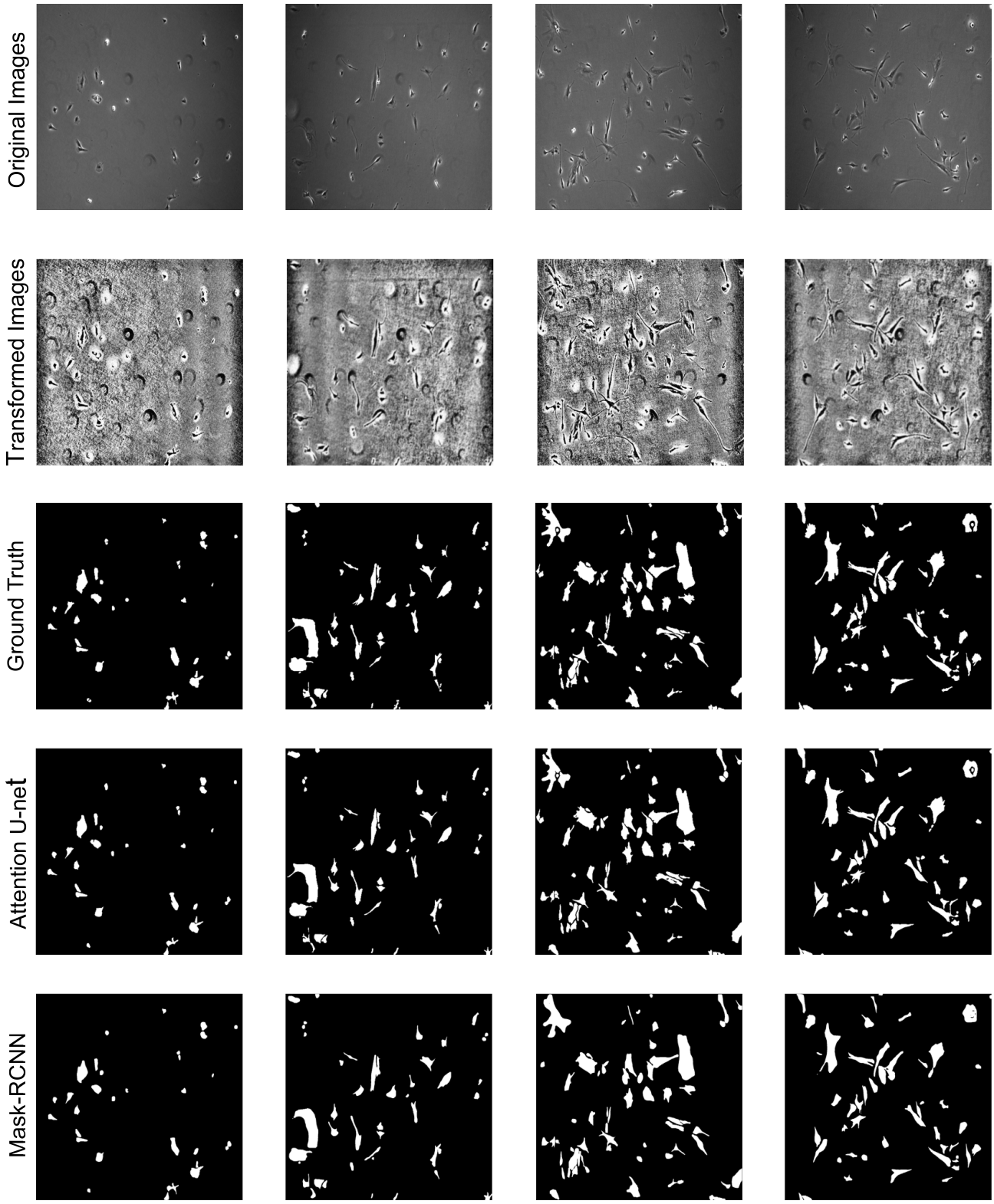


Figure E.5: Detailed overview of the output of Convolution-based model: Attention U-Net and Mask-RCNN with original images and transformed images as reference

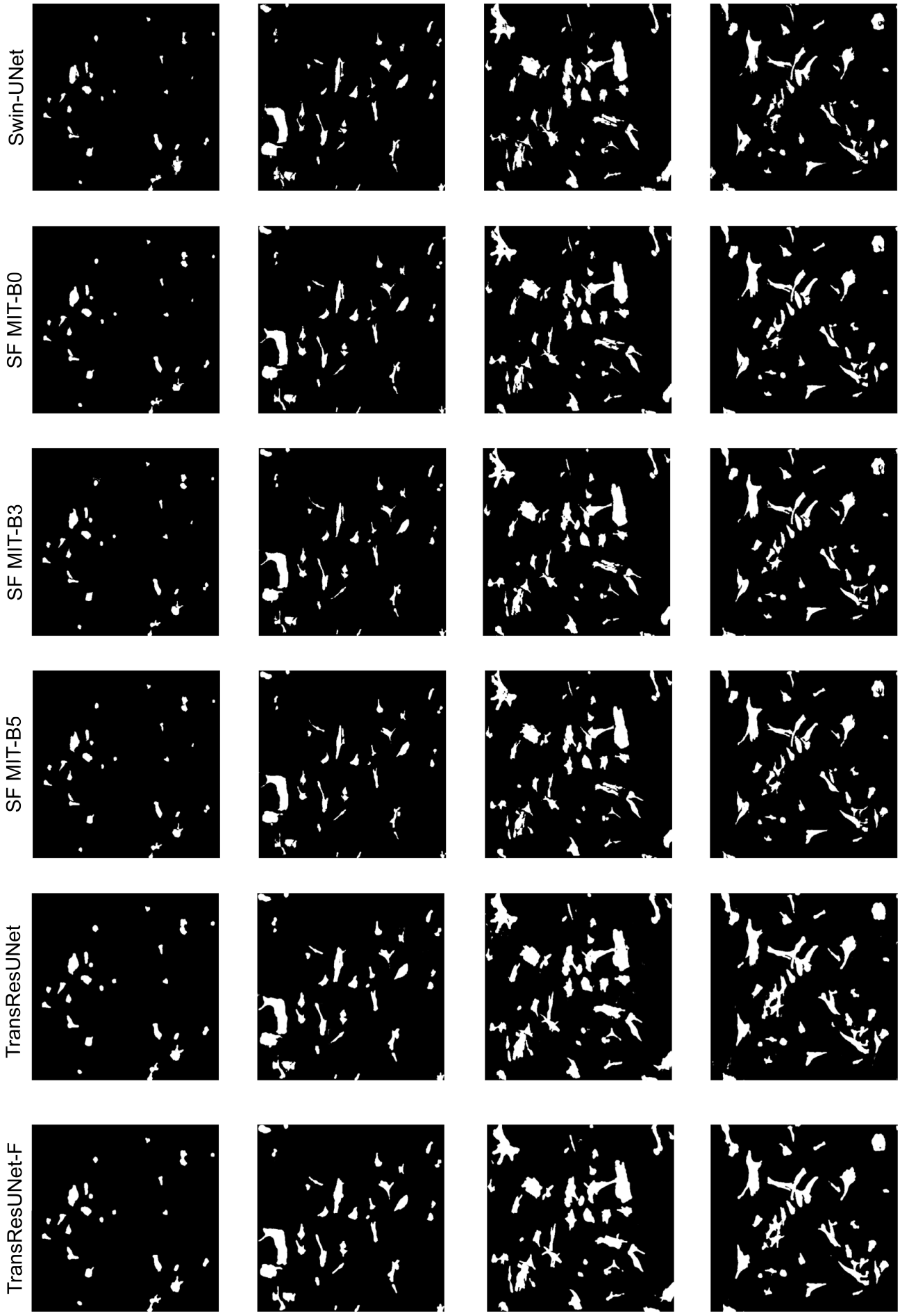


Figure E.6: Continuation of Figure E.5 showing the output of transformer-based models: Swin-UNet, Segformer MiT-B0, Segformer MiT-B3, Segformer MiT-B5, TransResUnet, TransResUnet-Finetuned

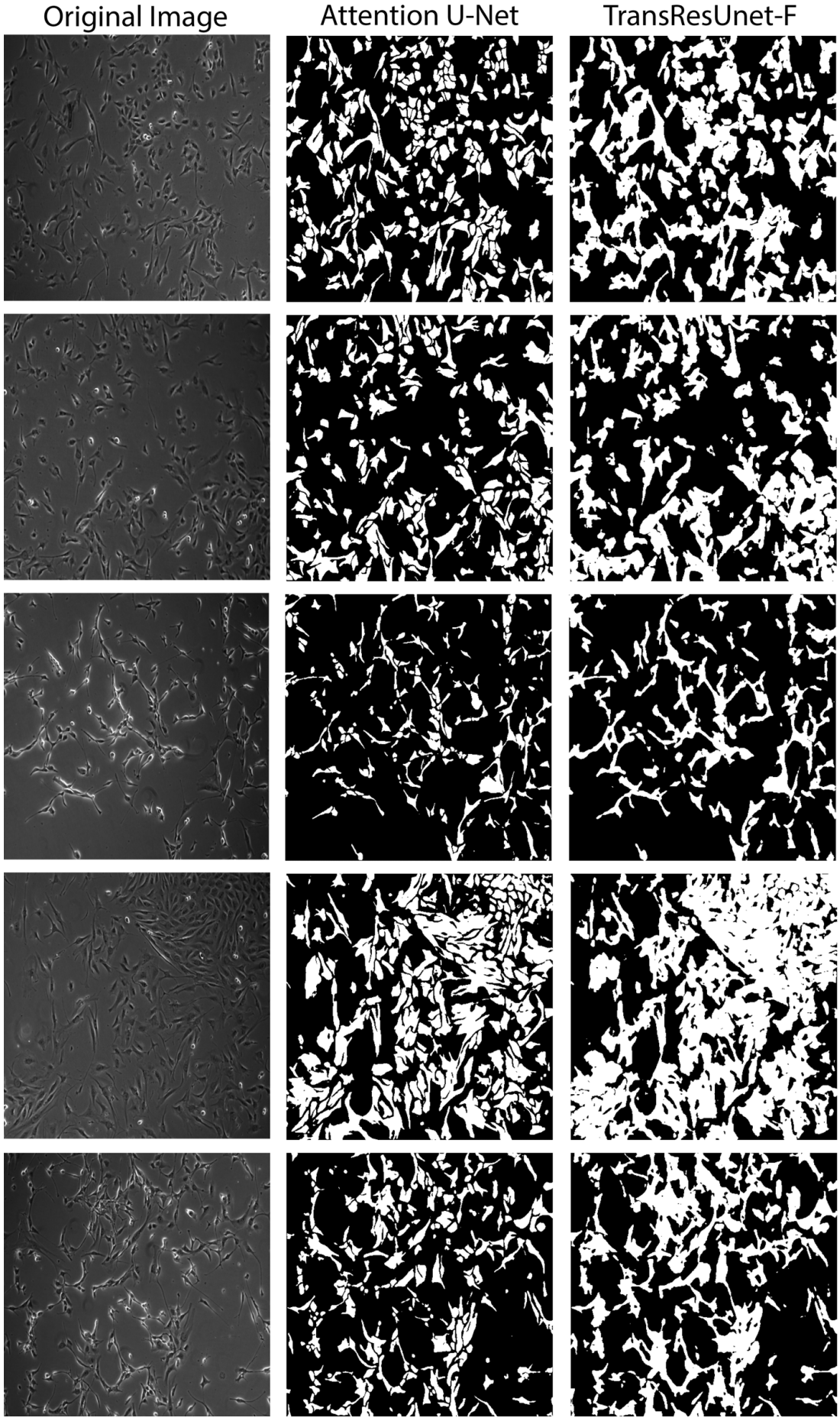


Figure E.7: Comparison of binary output masks from Attention U-Net and TransResUNet-F with out-of-training distribution data for different densities in Dataset 01 - 090303. The left column shows the original images, the middle column shows the output of Attention U-Net, and the right column shows the output of TransResUNet-F.

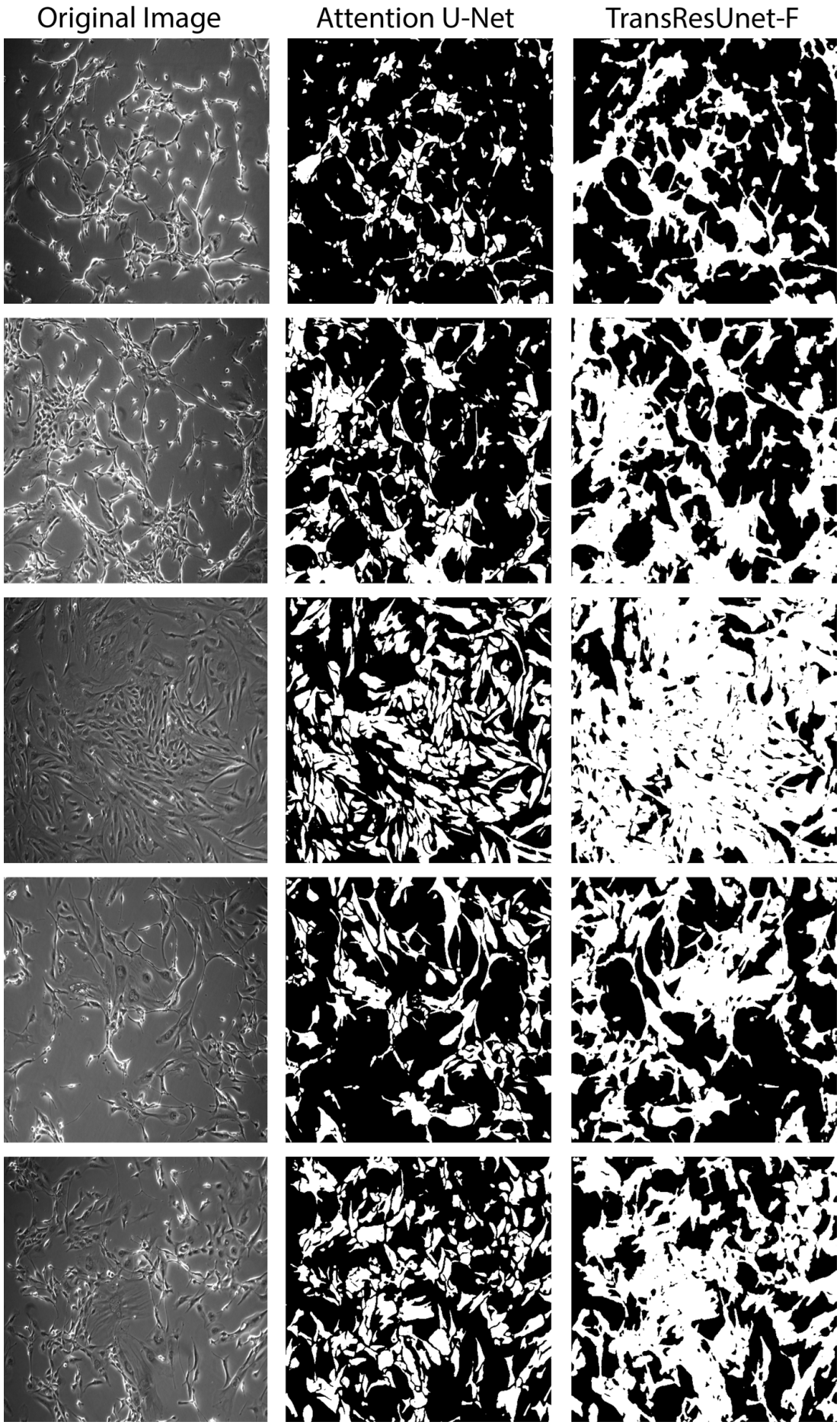


Figure E.8: Comparison of binary output masks from Attention U-Net and TransResUNet-F with out-of-training distribution data for different densities in Dataset 02 - 090318. The left column shows the original images, the middle column shows the output of Attention U-Net, and the right column shows the output of TransResUNet-F.

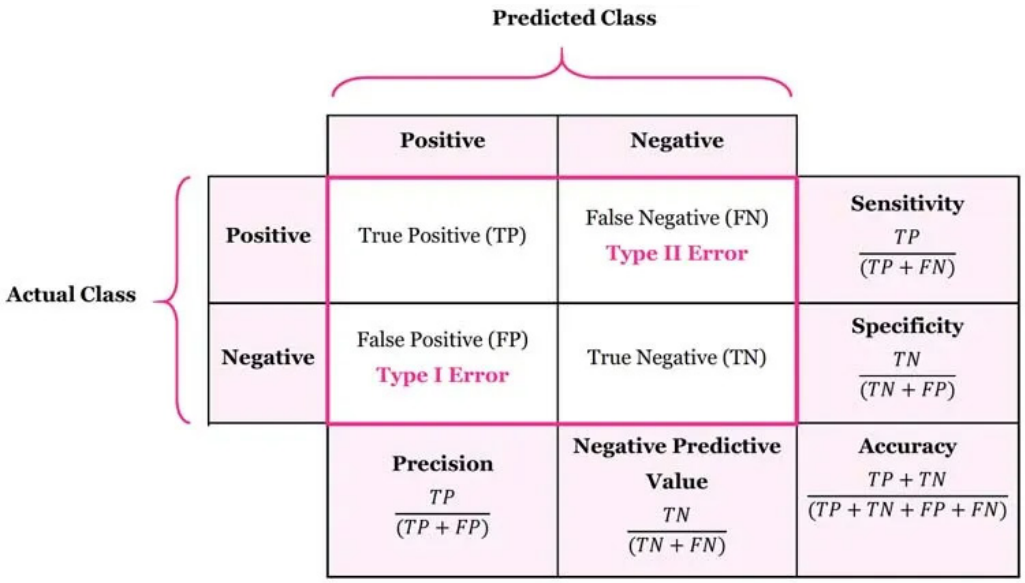


Figure E.9: Confusion Matrix and Performance Metrics [91]

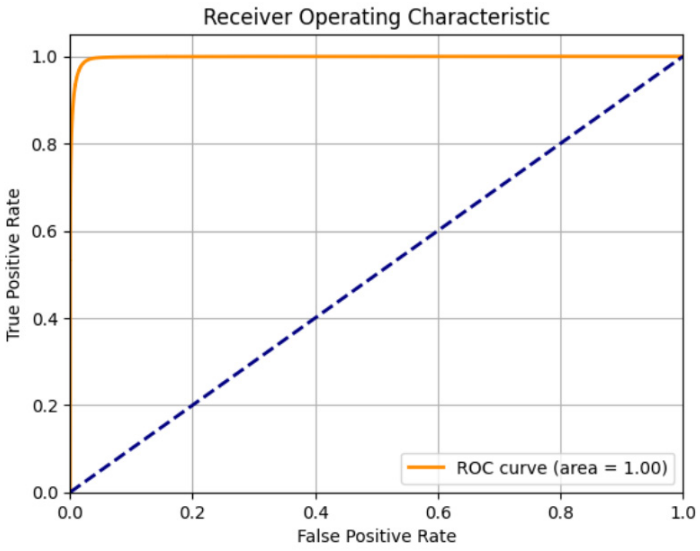


Figure E.10: ROC curve with an area of 1.00

Gebruik van Generatieve Artificiële Intelligentie (GenAI) - In te vullen formulier

Naam student: De Maeyer Lienert
Studentennummer: R0680156

Geef met "X" aan of het gaat om een cursusopdracht, een BIG-project of een masterproef:

X Dit formulier heeft betrekking op mijn masterproef.
Titel masterproef: **Voorbij Convolutie: Evaluatie van Gesuperviseerde Augmentatietechnieken met Transformer versus Convolutionele Neurale Netwerken voor Spiercel Segmentatie**
Promotor: Jean-Marie Aerts

O Dit formulier heeft betrekking op een BIG-project.
Titel BIG-project: ...
Promotor: ...

O Dit formulier heeft betrekking op een cursusopdracht.
Cursusnaam: ...
Cursusnummer: ...

Geef met "X" aan:

o Ik heb geen GenAI tool gebruikt.
X Ik heb wel een Gen AI tool gebruikt. Geef in dat geval aan welke (bv. ChatGPT/GPT-4/...): ChatGPT

Geef met "X" aan (mogelijk meerdere keren) op welke manier je de Gen AI tool hebt gebruikt:

- **X** **Als taalassistent voor het herwerken of verbeteren van zelf geschreven teksten zonder nieuwe inhoud toe te voegen.** (Dit gebruik is gelijk aan spelling- en grammaticacheckers en hoef je niet expliciet te vermelden.)
- **X** **Als zoekmachine om eerste informatie te krijgen over een onderwerp of voor een eerste aanzet om op zoek te gaan naar literatuur.** (Deze toepassing is gelijkaardig aan het gebruik van een gewone zoekmachine bij de uitwerking van een persoonlijk werkstuk. Als student ben je verantwoordelijk voor het controleren en verifiëren van de afwezigheid of correctheid van referenties. Ga dus vervolgens zelf op zoek naar wetenschappelijke referenties en voer zelf een analyse van de brondocumenten. Interpreteer, analyseer en verwerk de verkregen informatie; kopieer ze niet zomaar. Indien je vervolgens eigenhandig een tekst opstelt, hoef je het gebruik niet te vermelden.)
- o **Om tekstblokken te genereren.** (Wanneer je tekstblokken van GenAI output letterlijk overneemt, dan vermeld je de GenAI bronnen en citeer je, m.a.w. je meldt duidelijk dat het item via GenAI is aangemaakt d.m.v. citatie/referentie.)
- o **Om grafieken of figuren te genereren.** (Wanneer je grafieken/figuren van GenAI output letterlijk overneemt, dan vermeld je de GenAI bronnen en citeer je, m.a.w. je meldt duidelijk dat het item via GenAI is aangemaakt d.m.v. citatie/referentie.)
- o **Om code te laten genereren als deelaspect binnen een grotere opdracht.** (Let op, dit mag enkel als de docent hier EXPLICIET toestemming voor heeft gegeven.)
- o **Anders** (Neem contact op met de docent van de cursus of de promotor van de thesis of het BIG-project. Leg uit hoe je voldoet aan artikel 84 van het examenreglement. Leg uit wat het nut of de toegevoegde waarde van het gebruik van GenAI.)

Bijkomende richtlijnen en opmerkingen

De faculteit volgt het beleid van KU Leuven om GenAI verantwoord in te zetten. Daarbij is transparantie over het gebruik van GenAI door de student essentieel, waarvoor dit formulier hulp biedt. Onverantwoord en niet-transparant gebruik van GenAI kan beschouwd worden als een onregelmatigheid en gesanctioneerd worden. Studenten die overwegen om GenAI te gebruiken, moeten zich verder informeren via de universitaire website met bijkomende richtlijnen (Hoe GenAI correct te citeren en naar te refereren? Wat is (niet) toegelaten? Tips en aandachtspunten voor een verantwoord gebruik):

https://www.kuleuven.be/onderwijs/student/onderwijstools/artificiele-intelligentie