Institute of Imaging and Computer Vision | RWTH AACHEN UNIVERSITY

Titel der Arbeit in mehreren Zeilen weil lang und so (Tolle Vorlage, wohoo!)

Master Thesis

presented by

Stud Ent

Supervisor:
Bet Reuer

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

Erklärung nach §18 Abs. 1 ÜPO

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

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

Stud Ent
Aachen, 42.13.2037

Contents

List of Figures

List of Tables

List of Listings

Nomenclature

1 Introduction

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

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

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

2 Related Work

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

2.1 Convolutional Neural Networks (CNNs)

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

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

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

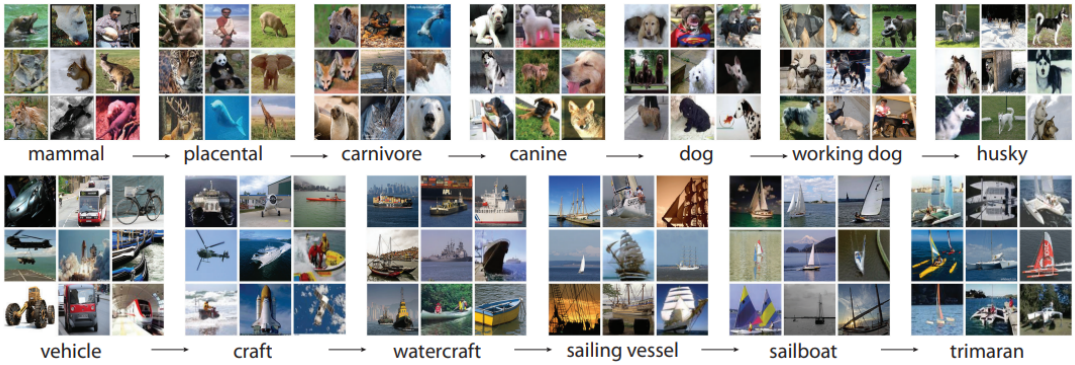

Figure 2.1: Example Images from the ImageNet dataset.

2.1.1 ImageNet

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

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

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

2.1.2 U-Net Architecture

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

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

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

Formally, the segmentation prediction can be seen as a pixel-wise classification problem, where the final layer of the U-Net outputs a probability map indicating the likelihood of each pixel belonging to a particular class (e.g., foreground or background in binary cell segmentation). The combination of hierarchical feature extraction and precise localization makes the U-Net particularly effective for segmenting structures of varying size and shape, as commonly encountered in biomedical images.

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

2.2 Image Segmentation

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

2.3 Semantic and Instance Segmentation

The vanilla U-Net assigns pixels to distinct, predifined classes. Usually, at the last layer of the decoder path, the original image resolution is restored (neglecting the missing pixels from unpadded convolutions). At this point, one would map the channel feature map depth (e.g. 64 for vanilla U-Net) via a 1x1 convolution to the amount of classes each pixel can be assigned to. Thereby one ends up having a probability distribution of predefined classes over all pixels. Logits are mapped to

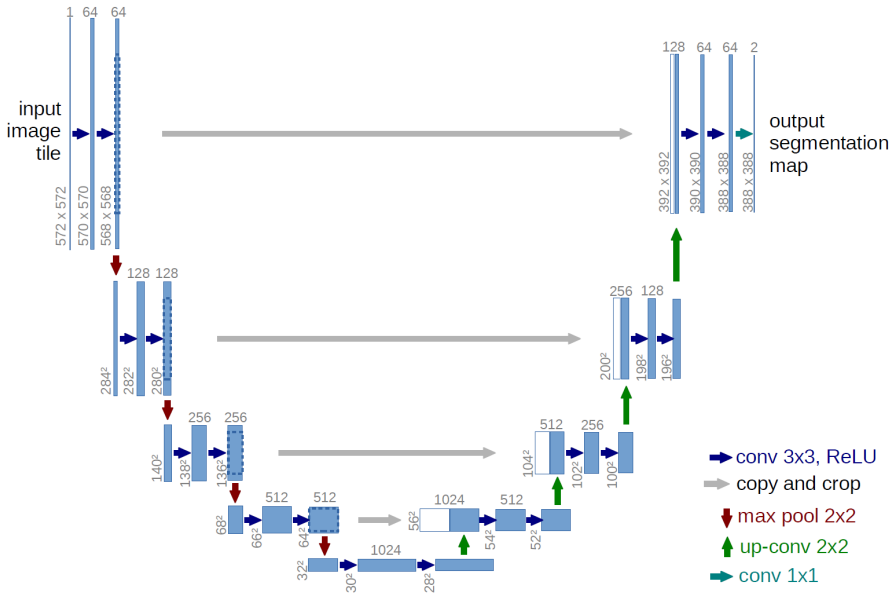

Figure 2.2: Schematic illustration of the U-Net architecture for image segmentation. The network consists of a contracting path (left side) to capture context via successive convolutional (3×3, ReLU) and max pooling (2×2) operations, and an expansive path (right side) that enables precise localization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

probabilities with Soft-Max and with Non-Maximum-Supression (NMS), each pixel is assigned to a class, so in the end the resolution corresponds to $\mathbf{H} \times \mathbf{W}$ with each pixel integer value corresponding to a certain class. These classes can represent all kinds of types, such as binarary values like foreground/background (cell or no cell), or for modern use cases, surrounding objects on a street. This type of segmentation is refered to as *semantic segmentation*. Oftentimes some other sophisticated post processing methods like binary closing or removal objects with size below a certain threshold are applied.

For many applications, it is desirable to use a framework that not only separates image regions into semantic classes, but also assigns each object instance within the same class a unique identifier. This task is known as instance segmentation. A common strategy is to first perform semantic segmentation and then separate individual instances in a post-processing step (a bottom-up approach). However, such methods are often prone to errors, for example when adjacent objects of the same class touch or overlap, leading to under-segmentation (merging of multiple cells) or over-segmentation (splitting a single cell into several fragments). Alternatively, instance segmentation can be formulated as detecting object proposals via bounding boxes, which are then refined into masks, as done in methods like Mask R-CNN. While effective in many domains, this top-down approach becomes inaccurate in dense cellular images where bounding boxes frequently overlap, making

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

2.3.1 Loss Functions and Metrics

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

While loss functions guide the optimization process during training, *evaluation metrics* serve to assess the quality of the model's predictions, typically on a separate validation or test set. Metrics quantify different aspects of performance that are relevant to the problem at hand and are especially important in applications such as cell segmentation, where a model's utility is not solely determined by its training loss.

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

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

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

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

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

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

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

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

threshold t, the Average Precision is computed as:

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

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

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

2.4 Foundation Models

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

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

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

2.5 Transformer

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

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

2.5.1 The Attention Mechanism

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

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

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

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

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

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

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

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

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

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

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

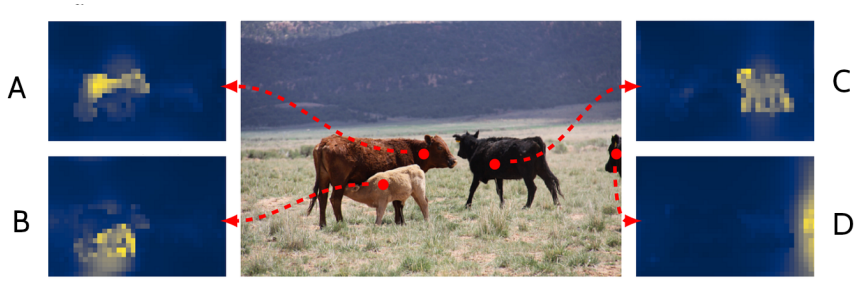

Figure 2.3: Encoder self-attention for a set of reference points. The encoder is able to separate individual instances (A-D).

2.5.2 Transformer

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

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

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

2.5.3 Vision Transformer

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

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

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

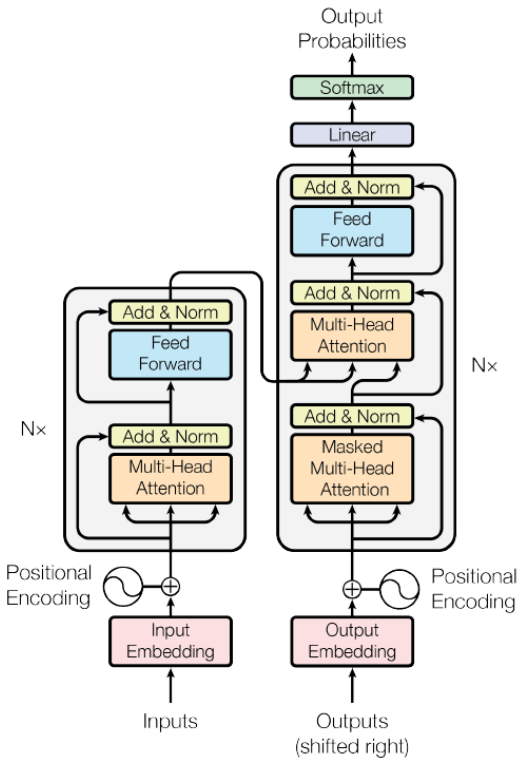

Figure 2.4: Transformer architecture. For vision tasks, usually the left part (encoder) is taken.

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

2.5.4 SegFormer

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

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

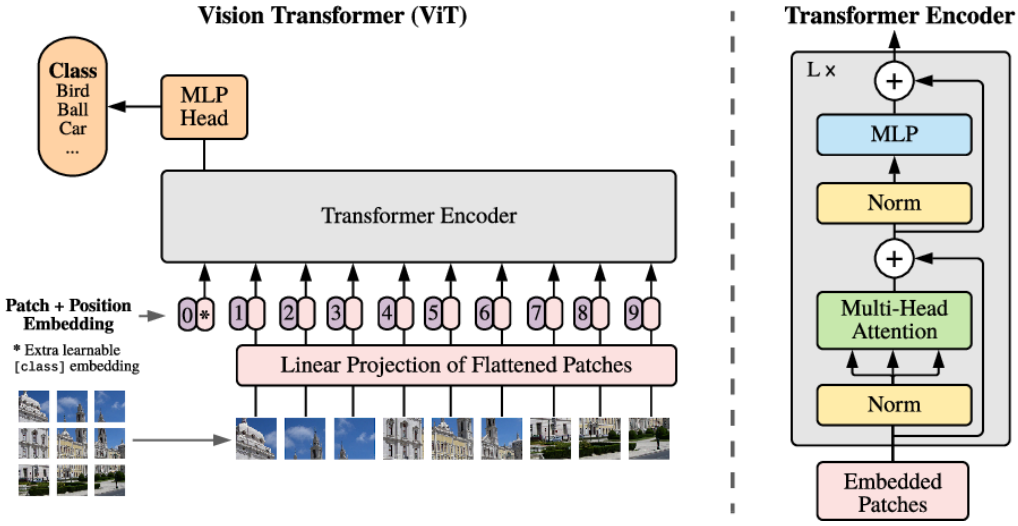

Figure 2.5: Vision transformer as presented in

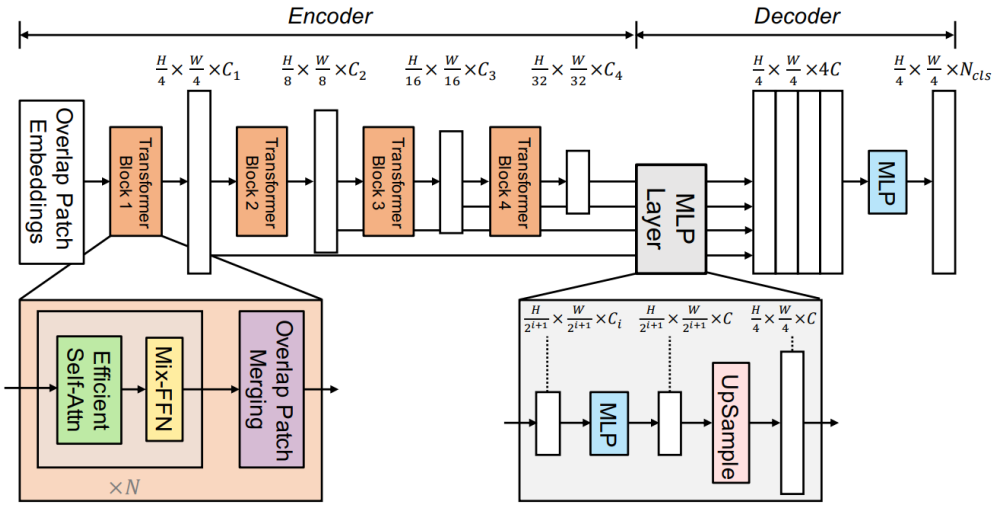

Figure 2.6: SegFormer architecture as presented in

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

state-of-the-art performance across several benchmarks while remaining computationally lightweight. A diagram of the network architecture is depicted in figure 2.6.

2.5.5 MA-Net: Multi-Scale Attention Network

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

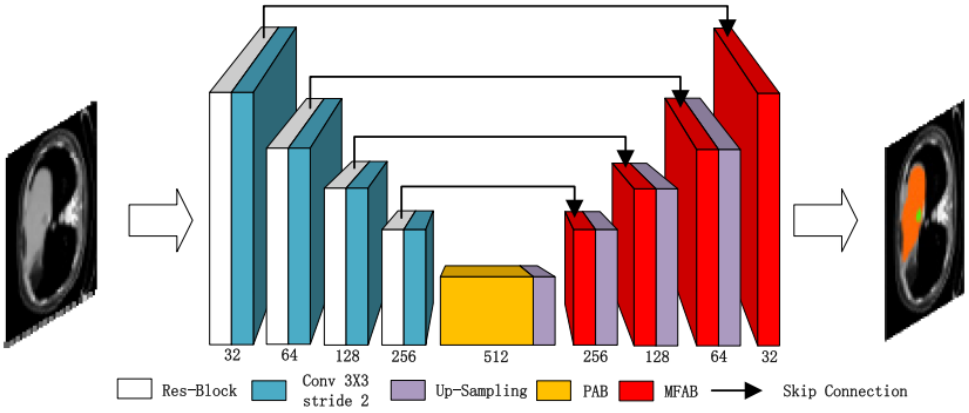

Figure 2.7: Architecture of the MA-Net.

The encoder extracts hierarchical features through standard convolutional layers, while the attention-enhanced decoder refines these features using two sufisticated attention modules before producing the final segmentation output. Skip connections between encoder and decoder stages are preserved, allowing the network to retain important spatial information during upsampling. The full high level architecture is displayed in 2.7.

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

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

A low-level view of the PAB block is given in 2.8. The local feature map $I \in \mathbb{R}^{H\times W\times 256}$ (in this net, the lowest resolution feature of the encoder) is first transformed by a 3×3 convolution to produce $I_0 \in \mathbb{R}^{H\times W\times 512}$. Three 1×1 convolution layers generate feature maps $A \in \mathbb{R}^{H\times W\times 64}$, $B \in \mathbb{R}^{H\times W\times 64}$, and $C \in \mathbb{R}^{H\times W\times 512}$.

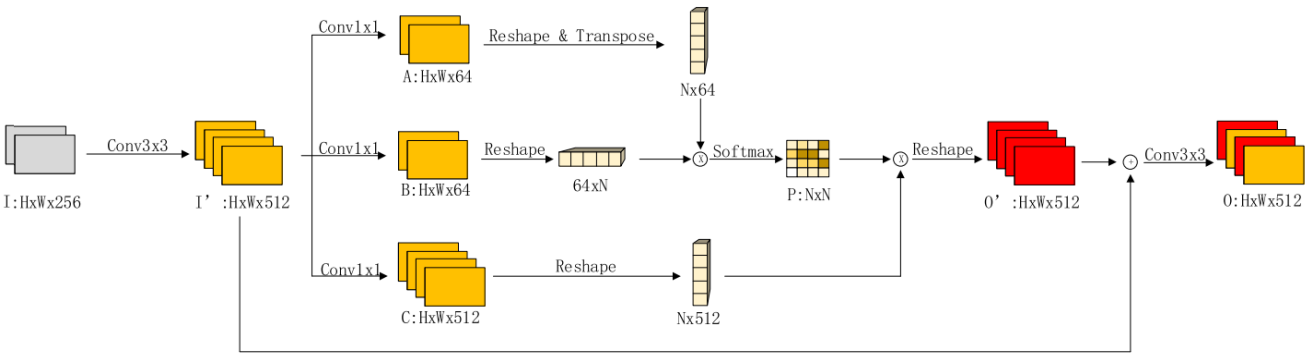

Figure 2.8: Low-level view of the PAB block.

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

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

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

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

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

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

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

The MFAB facilitates semantic feature fusion across multiple scales by combining feature maps of varying resolutions. This enables the decoder to integrate both fine-grained details and high-level semantic cues, which the authors claim to obtain more precise boundary delineation and region consistency. A low-level view of the MFAB block is given in 2.9.

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

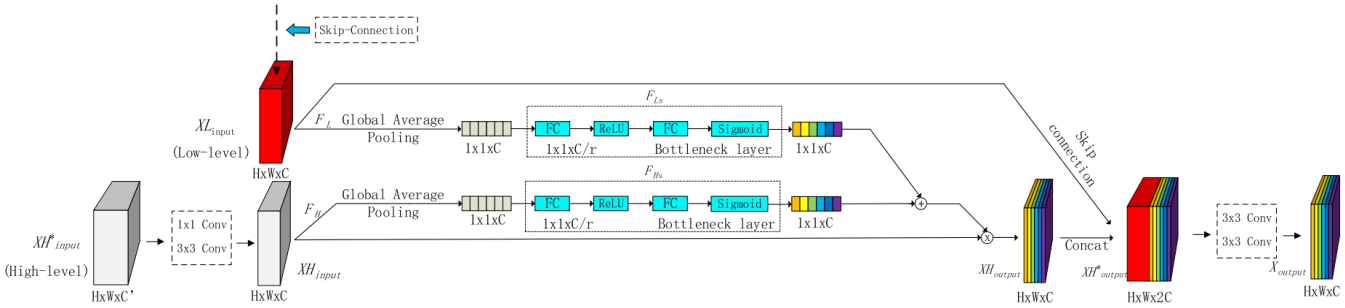

Figure 2.9: Multi-scale Fusion Attention Block (MFAB) structure.

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

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

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

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

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

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

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

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

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

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

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

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

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

2.6 Masked Autoencoders

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

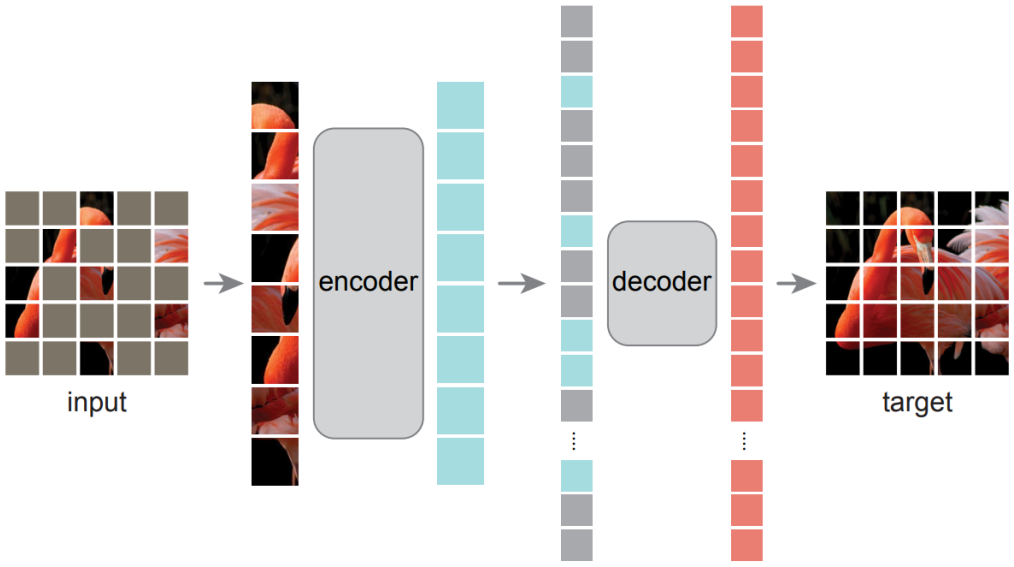

Figure 2.10: MAE architecture as presented in [1].

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

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

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

2.7 Segment Anything

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

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

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

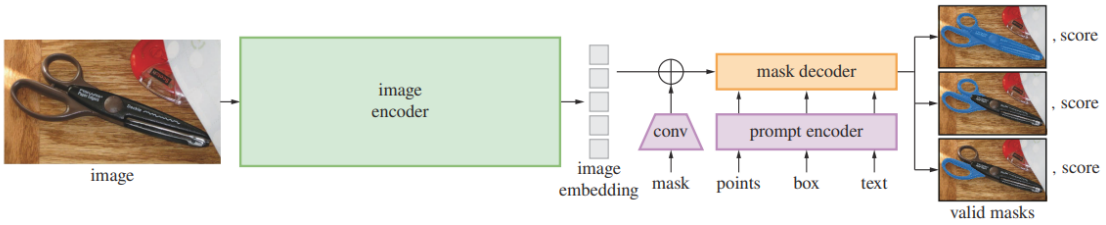

Figure 2.11: MAE architecture as presented in [1].

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

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

the IoU prediction branch is supervised with mean squared error.

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

2.7.1 Segment Anything 2

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

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

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

2.7.2 Hiera

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

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

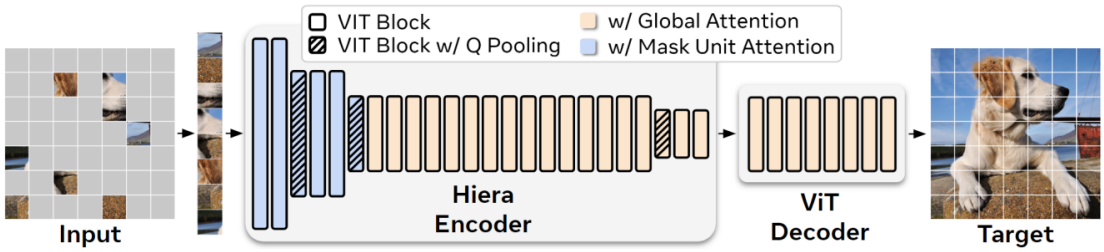

Figure 2.12: MAE architecture as presented in [1].

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

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

3 Methods

Bibliography

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

A Erstes Anhang-Kapitel

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

```
class LucasKanadeAAMFitter(AAMFitter):
    def __init__(self, params=None):
        # [...]
        self.do_stuff(params)

    def .do_stuff(self, params=None):
        # [...]
        print('oiski poiski')
```

Listing A.1: Code Blubb Blubb.