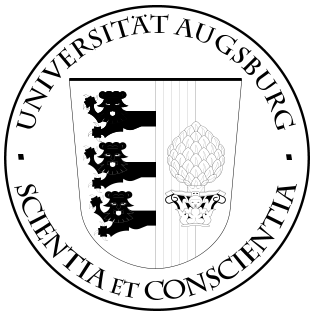
Institut für Informatik
Universität Augsburg



Bachelorarbeit

Decoder Architectures for Retaining Full Resolution in Semantic Segmentation

Tim Lukas Dirr

Gutachter:	Prof. Dr. Rainer Lienhart
Zweitgutachter:	Prof. Dr. Elisabeth André
Betreuer:	Robin Schön
Datum:	28. September 2022

verfasst am September 26, 2022
Lehrstuhl für Maschinelles Lernen und Maschinelles Sehen
Prof. Dr. Rainer Lienhart
Institut für Informatik
Universität Augsburg
D–86135 Augsburg, Germany
http://www.informatik.uni-augsburg.de

Abstract

The field of computer vision is deeply intertwined with neural networks, as they are the most prominent approach for solving tasks such as semantic segmentation. Semantic segmentation is a pixel-wise classification of images, wherein each pixel is classified into the class it represents on the surface of the image. Due to the inherent complexity of this task many networks output segmentation maps of lower resolution than the input image to reduce the computational burden. These lower size masks then typically get resized to full image resolution to compare them to the ground truth map. This practice can lead to a loss in performance, especially in applications where small details around the edges of objects are of importance.

This work focuses on designing two decoder frameworks, able to output segmentation maps with the same resolution as the input images. To accomplish this, we use SegFormer as our baseline and replace its All-MLP decoder. Specifically, our decoder frameworks make use of attention based transformer blocks from the SegFormer encoder and combine them with components inspired by U-Net. The U-Att-Large framework uses the four-level features passed by the standard SegFormer MiT-B0 backbone while U-Att-Full is combined with a slightly modified backbone - MiT-B0-Full - and receives six-level features. In addition to that, a third decoder framework is introduced. This U-Att-Small framework also uses MiT-B0 as its encoder and produces segmentation maps with 1/4 of the original images width and height, identical to the SegFormer All-MLP decoder.

We separately compare the performance of different models on MiT-B0 and MiT-B0-Full after fine-tuned on the Cityscapes dataset. On both backbones, we are able to improve the models accuracy significantly when compared to the standard SegFormer decoder. On MiT-B0 our best U-Att-Large model achieves a mIoU of 66.43% compared to the 62.96% mIoU using SegFormer's All-MLP decoder. Furthermore U-Att-Small is able to score a mIoU of 66.20% while also reducing the computational complexity in comparison to the All-MLP decoder. On MiT-B0-Full we improve the All-MLP performance of 62.09% mIoU to 65.20% mIoU with U-Att-Full.

Contents

1. Introduction

Computer Vision describes the capability of computers to gather important contextual information from images and videos. Computers are not only able to 'see', but can also 'understand', what they are seeing. This broad field has various applications, with one of them being the ability of computers to classify images. Such classification tasks range from simple image classification, where each image is categorized into exactly one class, to much more complex tasks, such as semantic segmentation. Semantic segmentation is a pixel-wise classification of images, meaning each pixel in an image is assigned a class. Use cases of semantic segmentation lie in a broad spectrum ranging from land mapping through satellite images to autonomous driving. Due to the complexity and computational cost of classifying millions of pixels in each image, many semantic segmentation networks output a segmentation map smaller than the original image. To determine the performance of the network, these lower-resolution output maps are then resized to match the size of the original image. This can lead to a loss in accuracy which might be acceptable in some use cases, where the increased speed due to fewer parameters and therefore lower computational cost is beneficial. However in cases where the accuracy of a network is the main factor, this reduced performance can be detrimental.

In this thesis components from U-Net [1] and SegFormer [2] are combined into three similar decoder frameworks. The proposed frameworks make use of convolution, self-attention, and feed-forward networks merged in a U-Net inspired architecture for combining multi-level features. The performance of these frameworks with various settings is compared to the native SegFormer decoder. This works goal is to evaluate the accuracy and computational complexity of different upsampling strategies as well as to determine the benefits of retaining full image resolution throughout the network by outputting segmentation maps with the same size as the input image. In addition, we also experiment with extending the SegFormer approach of using transformer [3] mechanisms, from the encoder to the decoder.

Overview Chapter 2 will focus on explaining relevant background information. This mainly includes the different network architectures serving as building blocks for our decoder architectures. In the following chapter 3, the tested decoder frameworks will be introduced and explained. Additionally, some changes to the backbone of the tested models will be shown. The conducted experiments and their results will be presented in chapter 4, followed by the conclusion to this work in chapter 5.

2. Background

2.1. Semantic segmentation

Semantic segmentation is a pixel-wise classification of images, wherein each pixel in an image is assigned to a class/category. The pixels are categorized into the class they represent on the surface of the image. The introduction of fully convolutional networks (FCN) [4] marks an important work in semantic segmentation, with FCNs able to classify images in an end-to-end manner. Until the introduction of vision transformer (ViT) [5], most of the researches focus had laid on improving these networks, enhancing the receptive field, or improving the fusion of coarse and fine features. ViT showed that the popular transformer architecture, previously used for natural language processing purposes and yielding state-of-the-art results, can successfully be used for Machine Vision tasks.

2.2. Encoder-Decoder Networks

Typically used for semantic segmentation are encoder-decoder architectures, which are very loosely defined and can take various forms. In general, the encoder - also known as the backbone of the network - is used to extract high-dimensional feature tensors at various resolutions smaller than the input image. In those features, important information, contained in the input image, is captured. The decoder aggregates these features and can combine features with different depths and resolutions, extracted at different stages of the encoder to create an output segmentation map. This segmentation map contains the predicted class of each pixel.

2.3. Terminology

To simplify the reading of this work, we define some terms that will be used throughout. Convolution in formulas will have the form

$$\mathrm{Conv}(K, S, P)(X),$$

with K representing the kernel size $K \times K$, S representing stride (S, S) and P representing padding added to all four sides of the input.

The Rectified Linear Unit (ReLU) activation function is referenced as

$$\text{ReLU}(x)$$

and the concatenation of n features $F_1, ..., F_n$ as

$$\text{Concat}(F_1, ..., F_n).$$

2.4. Upconvolution

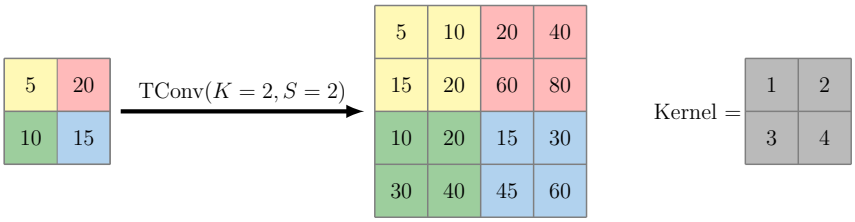


Figure 2.1.: Transposed convolution

What is referenced in this thesis as up-convolution is typically known as transposed convolution or fractionally-strided convolution. Relevant to this work are transposed convolutions with kernel size 2×2 and stride 2. Given input $I \in \mathbb{R}^{H \times W}$ and kernel $K \in \mathbb{R}^{2 \times 2}$, a transposed convolution with stride $S = 2$ results in output matrix $O \in \mathbb{R}^{2H \times 2W}$. If $M_{x,y}$ is the value of matrix M at position (x, y) this transposed convolution is formulated as

$$O = \text{TConv}(K = 2, S = 2)(I) \tag{2.1}$$

$$O_{2x-c,2y-r} = I_{x,y} * K_{2-c,2-r}, \tag{2.2}$$

with $x \in [1, ..., W], y \in [1, ..., H]$ and $c, r \in [0, 1]$. For simplicity we further define

$$\text{UpConv}(x) = \text{TConv}(K = 2, S = 2)(x). \tag{2.3}$$

Figure 2.1 shows an example for up-convolution.

2.5. U-Net

The U-Net [1] model is based on FCN [4] and mainly improves its decoder. U-Net can be divided into a nearly symmetrical encoder and decoder as shown in Figure 2.2. The authors also call U-Net's encoder the contracting and its decoder the expanding path of the network.

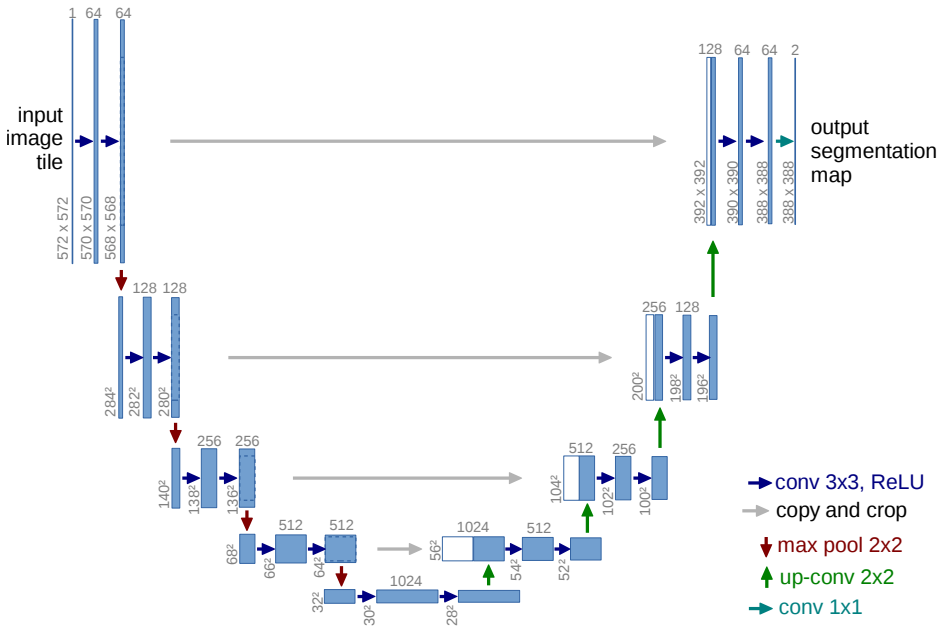


Figure 2.2.: UNet architecture taken from [1]

Encoder U-Net's encoder has a structure similar to CNNs and can be divided into four stages with $i \in [1, 2, 3, 4]$. An image X serves as input to the first layer. Every stage applies a U-Net-Block on its input. A U-Net-Block is made up of two 3×3 unpadded convolutions each followed by a ReLU activation.

$$\text{U-Net-Block}(x) = \text{ReLU}(\text{Conv}(3, 1, 0)(\text{ReLU}(\text{Conv}(3, 1, 0)(x)))) \tag{2.4}$$

The hereby calculated features F_i of stage i are down-sampled with a 2×2 max pooling operation ($\text{MaxPool}_{2\times2}(\cdot)$) and are the input of the next stage. The encoder is formulated as

$$F_1 = \text{U-Net-Block}(X) \tag{2.5}$$

$$F_i = \text{U-Net-Block}(\check{F}_{i-1}) \qquad i \in [2, 3, 4] \tag{2.6}$$

$$\check{F}_i = \text{MaxPool}_{2\times2}(F_i) \tag{2.7}$$

Skip connections The U-Net architecture connects the multi-level features F_i produced by the four encoder stages with the corresponding decoder stage through so-called skip connections. To accomplish those connections, features F_i need to be cropped (Crop$(\cdot)$) to equal the resolution used in the matching decoder stage. This is a consequence of the unpadded convolutions that are used, as they result in a lower resolution in decoder stage i compared to encoder stage i.

Decoder U-Nets decoder is built up nearly symmetrically to the encoder but uses five stages with $i \in [1, 2, 3, 4, 5]$. As the encoder, each decoder stage applies a U-Net-Block on its input.
Input to the decoder stage 5 is $\check{F}_4$, the downsampled features produced by the fourth and last encoder stage. Input to stages $i \in [1, 2, 3, 4]$ is the concatenation of cropped encoder features F_i with the upsampled output of the previous decoder stage D_{i+1}. Upsampling is achieved through up-convolution as explained earlier. Output segmentation map M is calculated by applying a 1×1 convolution on the last decoder layer's output D_1. Expressed in formulas this reads as:

$$D_5 = \text{U-Net-Block}(\check{F}_4) \tag{2.8}$$
$$M = \text{Conv}(1, 1, 0)(D_1) \tag{2.9}$$
$$D_i = \text{U-Net-Block}(\text{Concat}(\text{Crop}(F_i), \text{UpConv}(D_{i+1}))) \qquad i \in [1, ..., 4] \tag{2.10}$$

Channels The previous explanations focused on kernel size, stride, and padding within the used convolutions. Of great importance is also the number of channels before and after each convolution and the thereby resulting dimension of the features produced in each stage. Each U-Net-Block implements two convolutions. The input passed to the first convolution has $\vec{C}_i$ channels and its output has C_i channels. The second convolutions number of in- and output channels is C_i as well. Table 2.1 shows $\vec{C}_i$ and C_i for stages $[1, 2, 3, 4]$ of the encoder and stages $[5, 4, 3, 2, 1]$ of the decoder.

Encoder

i	$\vec{C}_i$	C_i
1	1	64
2	64	128
3	128	256
4	256	512

Decoder

i	$\vec{C}_i$	C_i
1	128	64
2	256	128
3	512	256
4	1024	512
5	512	1024

Table 2.1.: Number of Channels in the U-Net encoder and decoder

Moreover, the up-convolutions in the decoder half the number of channels from its input. Given an input of shape $H \times W \times C$, the output of the up-convolution will be of shape $2H \times 2W \times \frac{C}{2}$. The segmentation map M produced by the network has the same number of channels as there are classes in the specific segmentation task.

2.6. Transformer

The transformer architecture was first introduced in 2017 [3] for natural language processing (NLP). A typical NLP task is the translation of an input sequence x i.e. a sentence from one language to another, resulting in output sequence z. Transformer surpassed the previous state-of-the-art models in various translation tests. Before its introduction, Recurrent Neural Networks (RNN) and CNNs - divided into an encoder and a decoder - were the most prominent type of networks used for NLP. Some of these networks also used an attention mechanism to connect the encoder and decoder [3]. The transformer model does not implement any recurrence or convolutions but is solely based on attention. Attention mechanisms in general, allow neural networks to focus on important parts of the input while disregarding other less important parts.

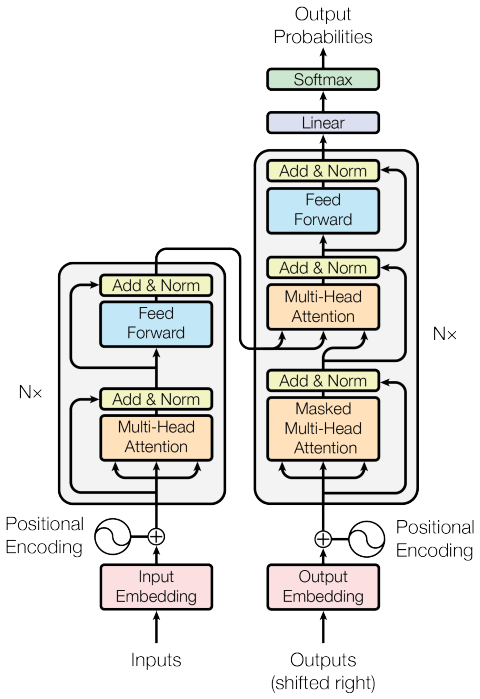


Figure 2.3.: Transformer architecture taken from [3]

Figure 2.3 shows the transformer architecture, splitted into an encoder (left) and decoder (right) stack. In the following the individual components of the network and how they are connected will be explained.

Embedding Transformer uses a learnable embedding to transform the input and output tokens to vectors of size d_{model}. Both embeddings share a set of weights, also shared with the linear transformation sitting at the end of the decoder part of the network. This linear layer combined with a SoftMax layer outputs the predicted next token probabilities [3].

Positional encoding To retain information about the order of the input sequence within the model, transformer uses positional encoding. Vectors with the same dimension of the input embeddings (d_{model}) are calculated and added to those embeddings. The values of those vectors are determined through sine and cosine functions. For a token at position *pos* in the input sequence, its positional encoding $PE(pos)$ is defined as

$$PE(pos) = \begin{bmatrix} PE(pos)_1 \\ PE(pos)_2 \\ \vdots \\ PE(pos)_{d_{\text{model}}} \end{bmatrix}, \tag{2.11}$$

with

$$PE(pos)_{2i} = sin(pos/10000^{2i/d_{\text{model}}}) \tag{2.12}$$

$$PE(pos)_{2i+1} = cos(pos/10000^{2i/d_{\text{model}}}). \tag{2.13}$$

Attention As already stated, transformer is based on attention. This attention mechanism can be thought of as similar to human attention, as it allows the network to focus on the important parts of the input and disregard less important parts. "An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors." [3] The query and the keys have dimension d_k, the values have dimension d_v. Each value has a corresponding key, whose compatibility with the query serves to compute the weight of the value. The output is the weighted sum of values. To calculate the attention for multiple queries at once, it is implemented as follows: M queries are combined in a matrix $Q \in \mathbb{R}^{M \times d_{\text{k}}}$ and the set of N key-value pairs are combined into matrices $K \in \mathbb{R}^{N \times d_{\text{k}}}$ and $V \in \mathbb{R}^{N \times d_{\text{v}}}$. A SoftMax is applied on the multiplication of Q with the transpose of K and then multiplied with V.

$$\text{Attention}(Q, K, V) = \text{SoftMax}(QK^T)V \tag{2.14}$$

Transformer scales this attention by dividing through $\sqrt{d_k}$, calling the modified version scaled dot-product attention.

$$\text{Attention}(Q, K, V) = \text{SoftMax}(\frac{QK^T}{\sqrt{d_k}})V \tag{2.15}$$

Multi-Head Attention The concrete implementation of attention in transformer follows the multi-head attention (MHA) mechanism introduced by the authors. The input to the MHA are keys K, values V and queries Q of size d_{model}. Instead of calculating a single attention function on the inputs, $K, V$and Q are linearly projected h times to dimensions d_k, d_k and d_v respectively. The attention is then calculated on each of the

projected versions - for each head - producing h outputs of dimension d_v, which are then concatenated and linearly projected. This results in the final output of the MHA. Formulated this reads as:

$$\text{MultiHead}_h(Q, K, V) = \text{Concat}(head_1, head_2, ..., head_h)W^O \tag{2.16}$$

$$\text{where } head_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{2.17}$$

The linear projections are represented as weight matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$ [3]. Figure 2.4 shows the Multi-Head attention.

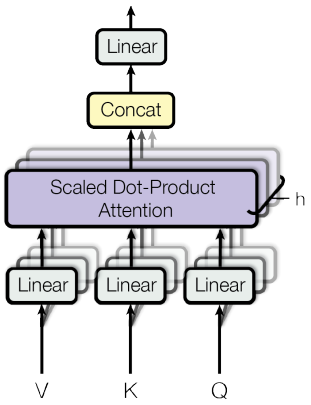


Figure 2.4.: Multi-head attention taken from [3]

Self-Attention While on the topic of attention, one more important concept has to be explained. The transformer model uses self-attention (SA) in both the encoder and decoder. SA is calculated through standard attention with $Q = K = V$. This means keys, values, and queries are the same. For an input x, self-attention is defined as

$$\text{SelfAttention}(x) = \text{Attention}(x, x, x). \tag{2.18}$$

Additionally, multi-head self-attention (MSA) is defined as

$$\text{MSA}_h(x) = \text{MultiHead}_h(x, x, x). \tag{2.19}$$

Feed-Forward Network The last component of the encoder/ decoder stacks in transformer is a feed-forward network (FFN) with one hidden layer formulated as

$$\text{FFN}(x) = \text{ReLU}(xW_1 + b_1)W_2 + b_2. \tag{2.20}$$

The FFN is applied over each position in the input sequence separately, but sharing the same set of weights in one layer. The first linear transformation is expansive, with $W_1 \in \mathbb{R}^{d_{\text{model}} \times 4 \cdot d_{\text{model}}}$ and $b_1 \in \mathbb{R}^{4 \cdot d_{\text{model}}}$, resulting in a vector with four times the input dimension. The second linear transformation is contracting, with $W_2 \in \mathbb{R}^{4 \cdot d_{\text{model}} \times d_{\text{model}}}$ and $b_2 \in \mathbb{R}^{d_{\text{model}}}$ resulting in an output vector of the same dimension as the input.

Encoder Combining the previously explained components, we take a look at the encoder of transformer. The encoder is made up of N identical blocks stacked on top of each other. The input to the first encoder block is the embedded and positional encoded input sequence x_{emb} as seen in Figure 2.3. The output of each encoder block serves as input to the next block. An encoder block is made up of two sub-layers, each surrounded by a residual connection and followed by a layer normalization. The first sub-layer is a MSA, and the second is a position-wise FFN (both as previously explained). The output y_{enc} of the encoder is the output of the last encoder block F_N. With $i \in [1, ..., N]$ the transformer encoder can be formulated as:

$$F_{1,1} = \text{LayerNorm}(x_{\text{emb}} + \text{MSA}_8(x_{\text{emb}})) \tag{2.21}$$
$$F_{i,1} = \text{LayerNorm}(F_{i-1,2} + \text{MSA}_8(F_{i-1,2})) \tag{2.22}$$
$$F_{i,2} = \text{LayerNorm}(F_{i,1} + \text{FFN}(F_{i,1})) \tag{2.23}$$
$$y_{\text{enc}} = F_{N,2} \tag{2.24}$$

Decoder Equivalent to the encoder, the transformer decoder is made up of N identical blocks stacked on top of each other. The decoder's output is the probability of the next token y_{pred} and is calculated by applying a SoftMax on the linear transformation of the last decoder blocks $i = N$ output. Decoder blocks are built up similarly to the encoder blocks but implement one additional sub-layer. Analog to the encoder, all sub-layers are surrounded by a residual connection and followed by a layer normalization.
The initial input of the decoder x_{prev} is the embedded and positional encoded sequence of a `[start]` token -signaling the start of the sequence - followed by the already generated output tokens from previous steps (none, if no tokens have yet been generated). Each block's output D_l serves as input to the following block.
The first sub-layer is a MSA layer, followed by another multi-head attention layer, where keys K and values V are the encoder output y_{enc} and queries Q are the output of the previous sub-layer. Sub-layer three is a position-wise FFN. With $i \in [1, ..., N]$ the decoder can be formulated as:

$$D_{1,1} = \text{LayerNorm}(x_{\text{prev}} + \text{MSA}_8(x_{\text{prev}})) \tag{2.25}$$
$$D_{i,1} = \text{LayerNorm}(D_{i-1,2} + \text{MSA}_8(D_{i-1,2})) \tag{2.26}$$
$$D_{i,2} = \text{LayerNorm}(D_{i,1} + \text{MultiHead}_8(D_{i,1}, y_{\text{enc}}, y_{\text{enc}})) \tag{2.27}$$
$$D_{i,3} = \text{LayerNorm}(D_{i,2} + \text{FFN}(D_{i,2})) \tag{2.28}$$
$$y_{\text{pred}} = \text{SoftMax}(D_{N,3}W^{\text{emb}}) \tag{2.29}$$

2.7. Vision transformer

The vision transformer (ViT) [5] architecture is directly based on transformer. Instead of being applied to NLP tasks and using embedded words as tokens, ViT embeds patches of images to feed them to a nearly standard transformer encoder to achieve image classification. In the following we will take a closer look at how ViT works.

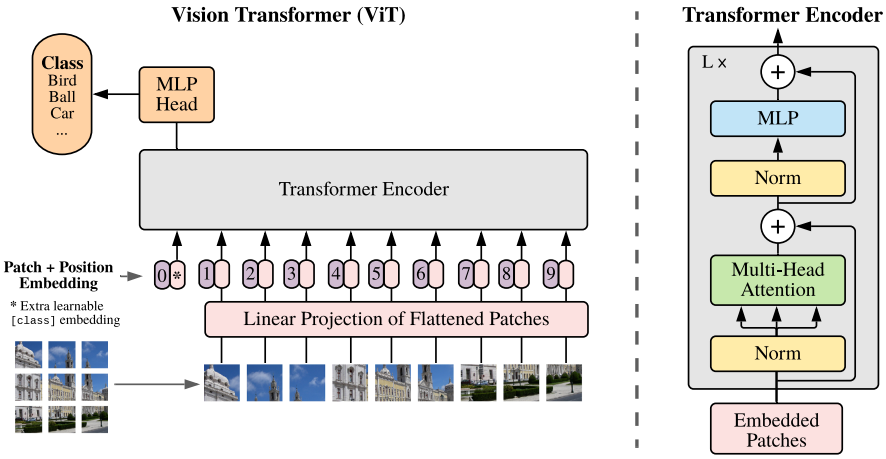


Figure 2.5.: Vision transformer architecture taken from [5]

Patch Embedding As the goal is to classify images, ViT receives an input image $\mathrm{x} \in \mathbb{R}^{H \times W \times C}$ with height H, width W and channels C (for standard RGB images $C = 3$). The input x is split into N patches of resolution $P \times P$ and flattened into 2 dimensions resulting in $\mathrm{x}_p \in \mathbb{R}^{N \times (P^2 \cdot C)}$. $N = \frac{HW}{P^2}$ is the number of patches. Using a linear transformation with weights $E \in \mathbb{R}^{(P^2 \cdot C) \times D}$ for embedding, x_p gets mapped to size $\mathbb{R}^{N \times D}$. D is the token size, which is constant throughout the model ($D = 768$ for the base ViT).

Class token In addition to the embedded input patches, a `[class]` token is added as the first item of the sequence. This token is a learnable parameter and also has dimension D. At the end of the encoder, the state of this token is used for classification. The embedded input sequence is defined as:

$$z'_0 = [x_{\text{class}}, x_p^1 E, x_p^2 E, ..., x_p^N E] \tag{2.30}$$

Positional Embedding Following the idea of transformer, ViT wants to retrain positional information about the patches in the original image. To accomplish this, a 1D positional embedding $E_{\text{pos}} \in \mathbb{R}^{(N+1) \times D}$ is used, which is added to z'_0 and learned during

training. This defines the sequence z_0 as:

$$z_0 = z'_0 + E_{\text{pos}} \tag{2.31}$$

Transformer encoder The sequence z_0, produced by the positional embedding, is then passed to the main component of ViT - a transformer encoder. This encoder is very similar to the one proposed in [3] (as explained in 2.6). It also comprises multiple encoder blocks stacked on top of each other ($L\times$). The encoder blocks use the same components as the original transformer encoder, only in a rearranged order. The first sub-layer is a MSA with k heads ($k = 12$ for base ViT) and the second sub-layer is a FFN as in the original encoder. The only difference in implementation is that ViT uses Gaussian Error Linear Units (GELU) for activation in the FFN, instead of ReLU. This lets us define the FFN as

$$\text{FFN}(x) = \text{GELU}(xW_1 + b_1)W_2 + b_2, \tag{2.32}$$

with $W_1 \in \mathbb{R}^{D\times 4\cdot D}$,$b_1 \in \mathbb{R}^{4\cdot D}$, $W_2 \in \mathbb{R}^{4\cdot D\times D}$ and $b_2 \in \mathbb{R}^D$.
ViT places the layer normalization before each sub-layer and residual connections after each sub-layer (as shown in Figure 2.5). With $i \in [1, ..., L]$ the encoder can be defined as:

$$z'_i = \text{MSA}_k(\text{LayerNorm}(z_{i-1})) + z_{i-1} \tag{2.33}$$
$$z_i = \text{FFN}(\text{LayerNorm}(z'_i)) + z'_i \tag{2.34}$$
$$y_{\text{enc}} = LayerNorm(z^0_L) \tag{2.35}$$

As mentioned, the state of the `[class]` token at the end of encoder (z^0_L) is passed to the classification head to determine the class of the input image.

Classification Head The output produced by the encoder is handled in two ways. In case of pre-training, the classification head is made up of a multilayer perceptron (MLP) with one hidden layer. During fine-tuning, a single linear transformation is used.

Training method ViT's training plays a crucial role in its performance. The authors show that pre-training datasets with limited training images and low regularization result in an unexceptional performance after fine-tuning. They claim that this can be attributed to the lack of inductive bias transformers have compared to similar CNNs and their inferior capability to generalize, if trained on insufficient data. The models best accuracies are achieved after being pre-trained on the JFT-300M dataset [6] with 300 million images and then fine-tuned for the specific task at hand, performing near or better than the state of the art at the time of its introduction.

2.8. SegFormer

Like transformer and ViT, SegFormer [2] is a SA-based network. It is developed for the task of semantic segmentation and combines a multi-level feature-producing encoder with a lightweight MLP decoder to produce segmentation maps for an input image. Its design is heavily inspired by ViT and pyramid vision transformer (PVT) [7]. The model backbone/encoder is pre-trained on the Imagenet-1K dataset [8]. Figure 2.6 depicts the SegFormer architecture.

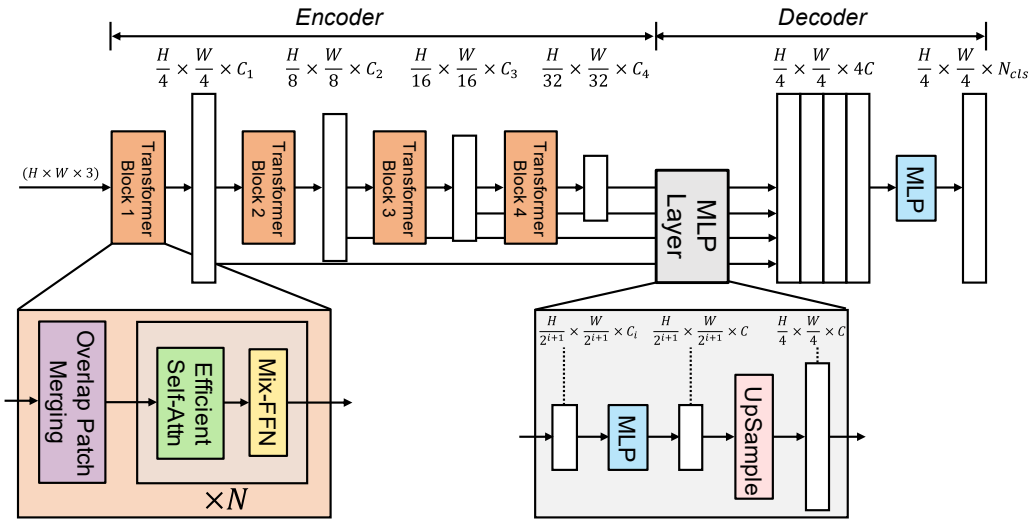


Figure 2.6.: SegFormer architecture adapted from [2]

2.8.1. Encoder

The SegFormer encoder is made up of four stages, each producing features at different scales of the input image. The authors define six different encoder settings from MiT-B0 to MiT-B5, increasing parameters through added layers and increased channel dimensions of the produced encoder features. This leads to a boost in accuracy but reduces speed and extends training time. As it is the most relevant for this work, the encoder will be described using the MiT-B0 settings, however the general method stays the same for all other five settings. Each stage implements a transformer block, combining an overlapped patch merging layer with a stack of efficient SA and FFN sub-layers.

Hierarchical feature representation As mentioned, the SegFormer encoder produces features at multiple scales of the input image. In particular, the features F_i produced by the four encoder stages using the MiT-B0 settings, have dimensions $\frac{H}{2^{i+1}} \times \frac{W}{2^{i+1}} \times C_i$ with $i \in [1, 2, 3, 4]$ and C_i as $[32, 64, 160, 256]$ from C_1 to C_4.

Overlapped Patch Merging Overlapped patch merging (OPM) is the first layer of each transformer block. The idea of OPM is similar to the one behind the patch embedding ViT uses, however it is not only applied on the input image, but also on the features passed by the previous stage to create features with reduced size and increased depth/ channel size. OPM is implemented through a simple convolution, as it maps overlapping patches of size $K \times K \times C_{i-1}$ to a vector of size $1 \times 1 \times C_i$. For the first encoder stage, a convolution with patch size $K = 7$, stride $S = 4$ and padding $P = 3$ is used. The three following stages implement OPM through a convolution with $K = 3$, $S = 2$ and $P = 1$. This allows SegFormer to produce multi-level features as explained in the previous paragraph. The convolution is followed by a layer normalization to complete the OPM.

$$\text{OPM}(K, S, P)(x) = \text{LayerNorm}(\text{Conv}(K, S, P)(x))$$

The OPM layer is followed by N sub-layer stacks that are made up of an efficient self-attention (ESA) and a mix-FFN. In the case of MiT-B0, each stage implements 2 stacks.

Efficient Self-Attention ESA applies the same scaled dot-product attention explained in 2.6, but to reduce the computational cost, the input features are reduced in size. Given input of size $H \times W \times C$ and flattening it to sequence $N \times C$ - with $N = HW$ as the length of the sequence - Q, K and V would have dimension N. The computational complexity of self-attention on this input is $O(N^2)$ and therefore very demanding for larger image sizes. To reduce this, SegFormer implements a reduction method proposed in [7]. The flattened input $I \in \mathbb{R}^{N \times C}$ gets reshaped to $\vec{I} \in \mathbb{R}^{N/R^2 \times C \cdot R^2}$ on which a linear projection is applied resulting in $I' \in \mathbb{R}^{N/R^2 \times C}$. I' is used as keys and values in the calculated attention. The ESA achieved through this reduction only has a computational complexity of $O(\frac{N^2}{R^2})$. R_i is set to $[8, 4, 2, 1]$ from stages 1 to 4.

$$\vec{I} = \text{Reshape}(\frac{N}{R^2} \cdot C, C \cdot R^2)(I) \tag{2.36}$$

$$I' = \text{Linear}(C \cdot R^2, C)(\vec{I}) \tag{2.37}$$

$$\text{ESA}(I) = \text{MultiHead}_H(I, I', I') \tag{2.38}$$

ESA is calculated multi-headed, with MiT-B0 using $[1, 2, 5, 8]$ heads from stages 1 to 4.

Mix-FFN The Mix-FFN introduced in SegFormer combines a convolutional layer - with kernel size $K = 3$, stride $S = 1$ and padding $P = 1$ - and two linear transformations. The authors argue that the convolutional layer makes positional encoding unnecessary,

as location information is leaked into the net by the convolution. Mix-FFN is formulated as:

$$\text{Mix-FFN}(x) = \text{Linear}(E \cdot C, C)(\text{GELU}(\text{Conv}(3,1,1)(\text{Linear}(C, E \cdot C)(x)))) + x \tag{2.39}$$

As the input sequence x to the Mix-FFN is passed from the preceding ESA layer, x is of shape $N \times C$ with $N = H \cdot W$. In order to apply the convolution, x gets reshaped to $H \times W \times C$. The convolution is also performed depth- wise. In this case, the convolution does not change the dimension of the input, therefor we can think of input and output as C features of size $H \times W$. Through the depth-wise convolution, each $H \times W$ output feature is derived from a corresponding $H \times W$ input feature using a unique kernel. After the convolution, the features are again flattened to shape $N \times C$. Similar to the FFN of ViT, the first linear transformation in the Mix-FFN expands the size of the features to $N \times E \cdot C$, and the second linear transformation contracts the features back to $N \times C$. MiT-B0 uses $E = 4$ for all four stages.

With F_0 as input image of size $\mathbb{R}^{H \times W \times 3}$ and $i \in [1,2,3,4]$ the MiT-B0 encoder can be formulated as:

$$\begin{aligned}
\text{Stack}(x) &= \text{LayerNorm}(\text{Mix-FFN}(\text{LayerNorm}(\text{ESA}(x)))) \\
\vec{F_i} &= \text{OPM}(F_{i-1}) \\
F_i &= \text{LayerNorm}(\text{Stack}(\text{Stack}(\vec{F_i})))
\end{aligned}$$

2.8.2. Decoder

The SegFormer All-MLP decoder is simple and lightweight as it only implements a few layers to produce a segmentation map M of shape $\frac{H}{4} \times \frac{W}{4} \times N_{\text{cls}}$. N_{cls} represents the number of categories in the segmentation task. Multi-level features F_i - produced by the four stages of the encoder ($i \in [1,2,3,4]$) - are passed to the decoder as input. The dimensions of F_i are $\frac{H}{2^{i+1}} \times \frac{H}{2^{i+1}} \times C_i$. A linear transformation is applied on F_i to unify the features channel dimension, resulting in F_i' with dimensions $\frac{H}{2^{i+1}} \times \frac{H}{2^{i+1}} \times C$. Each feature F_i' gets bilinearly up-sampled to size $\frac{H}{4} \times \frac{W}{4} \times C$. The features are then concatenated with each other to F of size $\frac{H}{4} \times \frac{W}{4} \times 4 \cdot C$. F is passed through two linear transformations, the first reducing channels size from $4 \cdot C$ to C, and the second from C to N_{cls}. Expressed as a formula the All-MLP decoder reads as

$$\begin{aligned}
F_i' &= \text{Linear}(C_i, C)(F_i) && \forall i \\
\vec{F_i} &= \text{Upsample}(\frac{H}{4} \times \frac{W}{4})(F_i') && \forall i \\
F &= \text{Concat}(\vec{F_1}, \vec{F_2}, \vec{F_3}, \vec{F_4}) \\
M &= \text{Linear}(C, N_{\text{cls}})(\text{Linear}(4C, C)(F)).
\end{aligned}$$

3. Method

In this work different decoder architectures are compared to the native decoder of SegFormer. The goal is to design a decoder able to produce output segmentation maps over the full input image resolution. Furthermore, we experiment with decoders producing segmentation maps of size $\frac{H}{4} \times \frac{W}{4}$ like SegFormer's All-MLP decoder and test, if we can improve the performance of the network even if we are not retaining full image resolution. As backbone to these tested models the MiT-B0 encoder of SegFormer was chosen as it is lightweight and does not require huge training times. SegFormers All-MLP decoder serves as a baseline for evaluating the performance of the introduced decoder architectures. All segmentation maps smaller than the input image get bi-linearly up-sampled to full input image resolution $H \times W$. This enables us to compare them to the ground truth map.

This chapter introduces three similarly designed decoder frameworks. We also slightly modify the MiT-B0 backbone to fit one of these frameworks and adjust the All-MLP decoder accordingly.

3.1. Modification of the SegFormer Backbone (MiT-B0)

To produce segmentation maps of full image resolution, a portion of the experiments use a modified version of the MiT-B0 backbone, which will be referenced as MiT-B0-Full in the following. The goal of this modification is to stay as close as possible to MiT-B0 but to provide the decoder with features over the full image resolution. Therefor, two additional stages in form of patch merging blocks are added in front of the original MiT-B0 encoder. Patch merging blocks are made up of an overlapping patch merging (OPM) layer followed by a GELU activation function and a layer normalization.

$$\text{PatchMergeBlock}(x) = \text{LayerNorm}(\text{GELU}(\text{OPM}(x)))$$

Since OPM is implemented through simple convolution, the effective modification is the addition of two convolutional layers to MiT-B0. Kernel K, stride S and padding P are chosen as $[K = 3, S = 1, P = 1]$ for the first stage and $[K = 3, S = 2, P = 1]$ for the second stage. Both stages produce features with the number of channels $C_i = 32$, as the first stage of the original MiT-B0 encoder also has an output with 32 channels. Through this change, the MiT-B0-Full encoder produces additional features of shapes $H \times W \times 32$ and $\frac{H}{2} \times \frac{W}{2} \times 32$.

One additional modification had to be made, as the first transformer block - which has moved from stage 1 to stage 3 - now gets passed features of shape $\frac{H}{2} \times \frac{W}{2} \times 32$ instead of an input image with shape $H \times W \times 3$. Therefore the OPM layer in transformer block 1 needs to use $K = 3, S = 2, P = 1$ to produce features of shape $\frac{H}{4} \times \frac{W}{4} \times 32$. The remaining transformer blocks stay exactly as before. Figure 3.1 shows the MiT-B0-Full encoder.

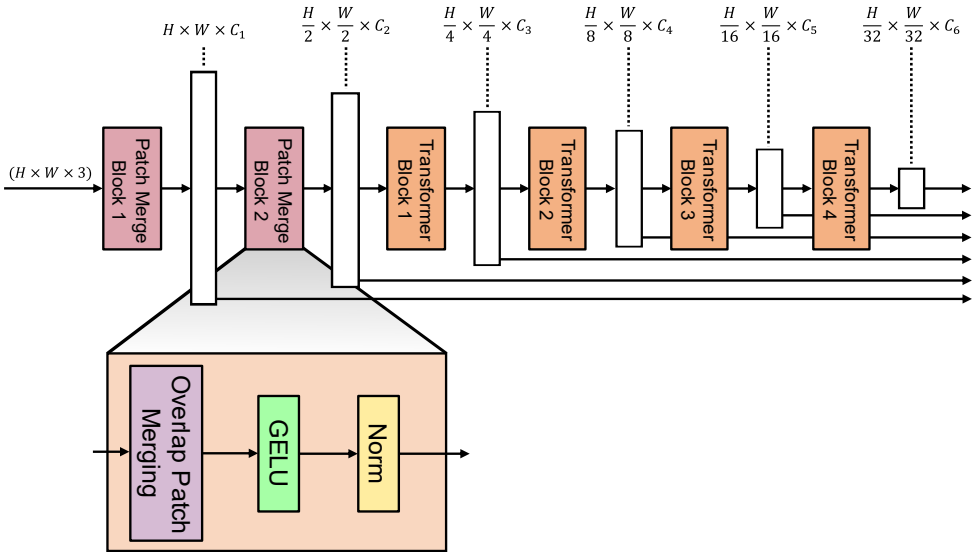


Figure 3.1.: MiT-B0-Full architecture modified from [2]

With F_0 as input image of size $\mathbb{R}^{H \times W \times 3}$ and $i \in [1, 2, 3, 4, 5, 6]$ the MiT-B0-Full encoder can be expressed in formulas as:

$$\begin{aligned} \text{Stack}(x) &= \text{LayerNorm}(\text{Mix-FFN}(\text{LayerNorm}(\text{ESA}(x)))) \\ F_i &= \text{PatchMergeBlock}(F_{i-1}) && i \in [1, 2] \\ F_i &= \text{LayerNorm}(\text{Stack}(\text{Stack}(\text{OPM}(F_{i-1})))) && i \in [3, 4, 5, 6] \end{aligned}$$

3.2. Modification of the SegFormer All-MLP Decoder

For additional comparison with SegFormer, the All-MLP Decoder is slightly modified to use the full six-level features produced by MiT-B0-Full. This modified version will be referenced as All-MLP-Full decoder and produces segmentation maps M of shape $H \times W \times N_{\text{cls}}$ by using the two additional features of shape $H \times W \times 32$ and $\frac{H}{2} \times \frac{W}{2} \times 32$ and bilinearly upsampling to $H \times W$. Its formulation is nearly identical to the original All-MLP Decoder:

$$
\begin{aligned}
F_i' &= \text{Linear}(C_i, C)(F_i) && \forall i \\
\vec{F_i} &= \text{Upsample}(H \times W)(F_i') && \forall i \\
F &= \text{Concat}(\vec{F_1}, \vec{F_2}, \vec{F_3}, \vec{F_4}, \vec{F_5}, \vec{F_6}) \\
M &= \text{Linear}(C, N_{\text{cls}})(\text{Linear}(6C, C)(F))
\end{aligned}
$$

3.3. U-Att-Decoder

The three decoder framework that will be tested in the work are very similar and use the same components. We name those frameworks

- U-Att-Full
- U-Att-Small
- U-Att-Large

and the specific settings used within them, enable the testing of various different decoders. The proposed U-Att decoder apply a progressive upsampling strategy inspired by U-Net and SETR-PUP [9], combining it with the transformer blocks introduced in SegFormer.

U-Att-Full The U-Att-Full framework uses the MiT-B0-Full backbone and gets passed multilevel features F_i of shape $\frac{H}{2^{i-1}} \times \frac{H}{2^{i-1}} \times C_i$ from the six encoder stages with $i \in [1, 2, 3, 4, 5, 6]$. Matching this, U-Att-Full also implements six stages, aggregating the encoder features stage by stage. Input $\vec{F_i}$ to stage i is very similar to the U-Net decoder. $\vec{F_i}$ is the concatenation of features F_i - produced by the corresponding encoder stage - with the upsampled output of the previous decoder stage D_{i+1} (apart from the first stage, as there is no previous stage). The upsampling is achieved through an up-convolution. Each stage applies a decoder block on $\vec{F_i}$. Depending on the later listed settings, this block is either a U-Net or a transformer block. Figure 3.2 shows a decoder block.

$$\text{DecBlock}_i(x) = \begin{cases} \text{U-Net-Block}(x) \\ \text{TransformerBlock}(x) \end{cases} \tag{3.1}$$

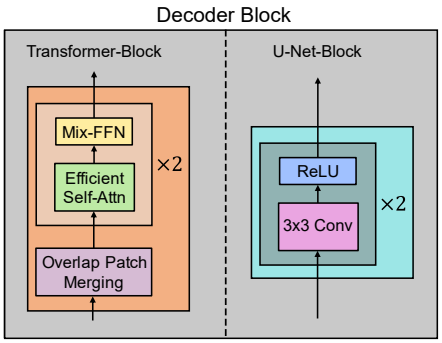


Figure 3.2.: Decoder block

A U-Net block is made up of two convolutions with $K = 3, S = 1$ and $P = 1$ surrounded by two ReLU activations.

$$\text{U-Net-Block}(x) = \text{ReLU}(\text{Conv}(\text{ReLU}(\text{Conv}(x)))) \tag{3.2}$$

A transformer block on the other hand is implemented nearly identical to the transformer blocks in MiT-B0. OPM with $K = 3, S = 1, P = 1$ is applied on the input followed by two stacks of normed efficient self-attention (ESA) and Mix-FFN. Depending on the resolution of the input features, the ESA reduction ratio R_i, the MHA heads H_i, and the Mix-FFN expansion ratio E_i are set. These settings are duplicated from MiT-B0s' transformer blocks with the same input resolutions (see Table 4.2).

$$\text{Stack}(x) = \text{LayerNorm}(\text{Mix-FFN}(\text{LayerNorm}(\text{ESA}(x)))) \tag{3.3}$$

$$\text{TransformerBlock}(x) = \text{LayerNorm}(\text{Stack}(\text{Stack}(\text{OPM}(3,1,1)(x)))) \tag{3.4}$$

Lastly, the output D_1 of the last decoder layer is linearly transformed from channel size C_1 to N_{cls}, identical to SegFormer. Combined, the U-Att-Full framework can be formulated as

$$\vec{F}_6 = F_6 \tag{3.5}$$

$$\vec{F}_i = \text{Concat}(F_i, \text{UpConv}(D_{i+1})) \qquad i \in [1,2,3,4,5] \tag{3.6}$$

$$D_i = \text{DecBlock}_i(\vec{F}_i) \qquad \forall i \tag{3.7}$$

$$M = \text{Linear}(C_1, N_{\text{cls}})(D_1) \tag{3.8}$$

Given output D_{i+1} from the previous decoder stage with shape $\frac{H}{2^{i+2}} \times \frac{H}{2^{i+2}} \times C_{i+1}$, up-convolution is applied transforming it to the same shape as encoder features $F_i : \frac{H}{2^{i+1}} \times \frac{H}{2^{i+1}} \times C_i$. For U-Att-Full C_i is set as $[32, 32, 32, 64, 160, 256]$ from C_1 to C_6. Figure 3.3 shows the U-Att-Full framework.

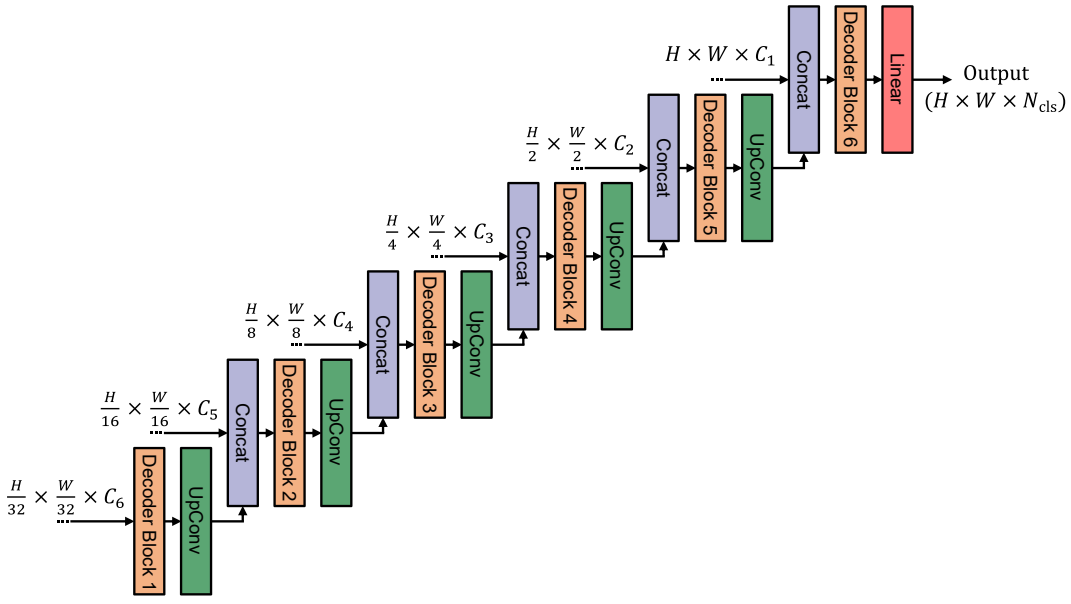


Figure 3.3.: U-Att-Full framework

U-Att-Small The U-Att-Small framework is nearly identical to U-Att-Full with the difference, that only four instead of six stages are used. This is the result of U-Att-Small being combined with the original MiT-B0 backbone, which itself passes multi-level features F_i of shape $\frac{H}{2^{i+1}} \times \frac{H}{2^{i+1}} \times C_i$ from its four stages with $i \in [4, 3, 2, 1]$. The output of U-Att-Small is segmentation map M of shape $\frac{H}{4} \times \frac{W}{4} \times N_{\text{cls}}$.

$$
\begin{aligned}
\vec{F}_4 &= F_4 \\
\vec{F}_i &= \text{Concat}(F_i, \text{UpConv}(D_{i+1})) && i \in [1, 2, 3] \\
D_i &= \text{DecBlock}(\vec{F}_i) && \forall i \\
M &= \text{Linear}(C_1, N_{\text{cls}})(D_1)
\end{aligned}
$$

Figure 3.4 shows the U-Att-Small framework.

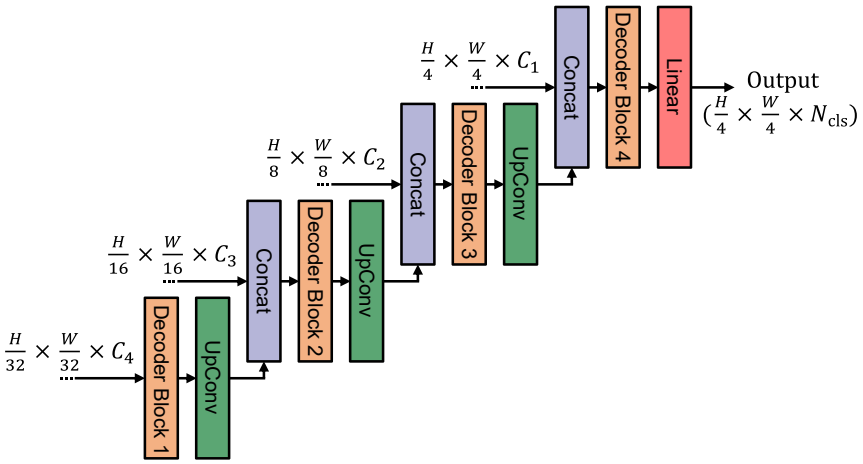


Figure 3.4.: U-Att-Small framework

U-Att-Large U-Att-Large can be described as a mix of U-Att-Small and U-Att-Full. The decoder produces segmentation maps retaining full input image resolution by implementing six stages like U-Att-Full. However it only gets passed four-level features as it is used with the MiT-B0 encoder. There are only four stages in the encoder and six stages in the decoder, leading to the two additional decoder stages only applying its layers on the upsampled output of the previous stage and not the concatenation with

encoder features. The U-Att-Large Framework expressed as a formula is

$$\vec{F}_6 = F_4 \tag{3.9}$$
$$\vec{F}_i = \text{Concat}(F_{i-2}, \text{UpConv}(D_{i+1})) \qquad i \in [3,4,5] \tag{3.10}$$
$$\vec{F}_i = \text{UpConv}(D_{i+1}) \qquad i \in [1,2] \tag{3.11}$$
$$D_i = \text{DecBlock}(\vec{F}_i) \qquad \forall i \tag{3.12}$$
$$M = \text{Linear}(C_1, N_{\text{cls}})(D_1) \tag{3.13}$$

with $\vec{F}_i$ of dimension $\frac{H}{2^{i-1}} \times \frac{W}{2^{i-1}} \times C_i$ and C_i as $[32, 32, 32, 64, 160, 256]$ from C_1 to C_6. Figure 3.5 shows the U-Att-Large framework.

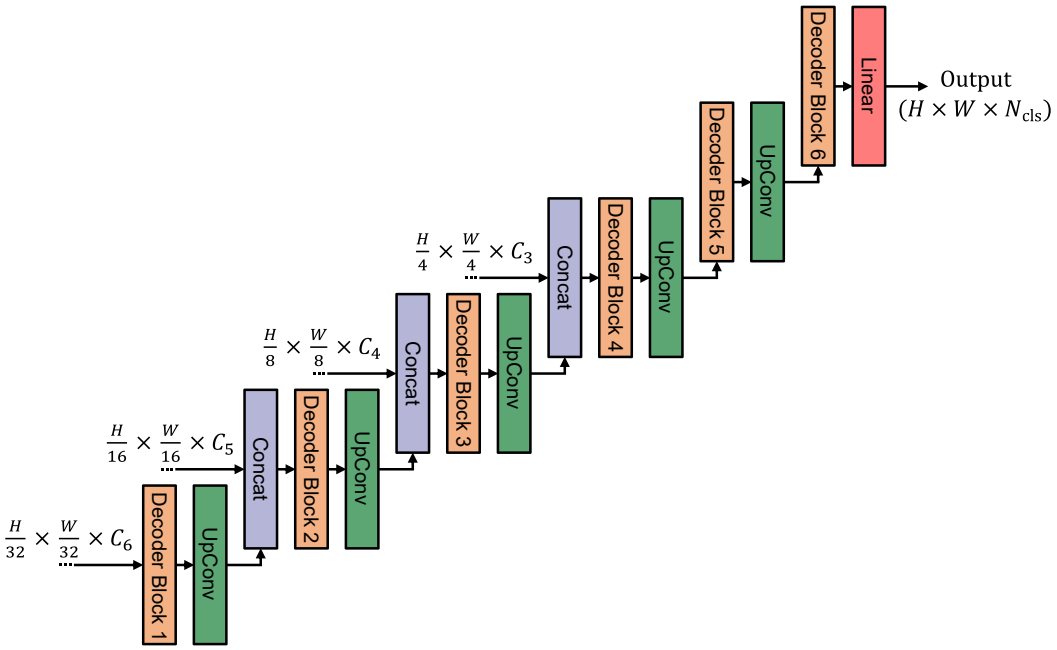


Figure 3.5.: U-Att-Large framework

4. Experiments and Results

4.1. Pre-training

Both MiT-B0 and MiT-B0-Full Backbones were pre-trained on Imagenet-1K [8] and later fine-tuned with different randomly initialized decoders on Cityscapes.

Imagenet-1K dataset The dataset referenced as Imagenet-1K is the popular dataset used in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012 [8]. It contains 1.28 million training images and 50 thousand validation images of 1000 different classes. The images in this dataset have various resolutions.

MiT-B0 For the MiT-B0 backbone, the Imagenet-1k pre-trained model by the authors of SegFormer [2] was used. The exact settings they applied during pre-training are not publicly known. However, it can be assumed that this version was trained far longer than MiT-B0-Full, considering their respective Top 1 accuracy as shown in Table 4.1.

MiT-B0-Full During the training of MiT-B0-Full, the Imagenet-1k images were randomly cropped and resized to resolution ($224 \times 224 \times 3$) as well as randomly rotated before being passed to the network. The backbone was trained for 60 epochs on the whole dataset using the AdamW optimizer [10] with $\beta_1 = 0.9, \beta_2 = 0.999$, a batch size of 32 and a weight decay of 0.05. The learning rate is set to $3.125 \cdot 10^{-5}$ and scheduled using cosine annealing with linear warm-up over 20 epochs. Because of resource limitations MiT-B0-Full could not be trained to the extend of MiT-B0. Though given enough training time, MiT-B0-Full should be able to achieve comparable Top 1 accuracy to MiT-B0 as their architecture is very similar.

Methode	Top 1
MiT-B0-Full	54.8
MiT-B0	70.5

Table 4.1.: Top-1 Accuracy on ImageNet-1K

4.2. Fine-tuning

Cityscapes The dataset all models are fine-tuned on is Cityscapes [11]. Cityscapes provides 3475 fully annotated images of which 2975 are used for training and the other 500 for evaluation. All images have resolution 1024×2048 and are annotated pixel-wise using 30 different classes. 19 of those classes are actually used to train the models as the other ones are too rare. During training, the images are randomly resized, cropped to resolution 192×192, and then randomly flipped before being passed to the network.

Class
road
sidewalk
building
wall
fence
pole
traffic light
traffic sign
vegetation
terrain
sky
person
rider
car
truck
bus
train
motorcycle
bicycle

Training All models are trained for 300.000 iterations on the Cityscapes dataset. Each iteration's batch size is set to eighth. Identical to pre-training, the models are trained using the AdamW optimizer with $\beta_1 = 0.9, \beta_2 = 0.999$. Learning rate α is set to $6 \cdot 10^{-5}$ and scheduled with polynomial decay and 500 iterations of linear warm-up, finishing training at $\alpha = 0$. Additionally, the decoder uses an increased learning rate of $10 \cdot \alpha$. Loss is calculated through the cross entropy loss function.

4.3. Evaluation

Data As stated above, Cityscapes provides 500 fully annotated validation images used for the evaluation of the tested models. The images get resized to resolution 256×512 and evaluated using a sliding window. Patches of size 192×192 are taken and forward-passed through the model. The output segmentation maps of these patches are combined into one segmentation map of full image size and then compared to the ground truth segmentation map. The exact implementation is done with the help of the mmsegmentation package.

Result presentation As mentioned in 3.3, the settings used for the proposed U-Att decoder frameworks determine which type of decoder block is applied in each stage. A decoder block is either a U-Net or a Transformer block. We present the settings as a combination of **U**s for U-Net-Blocks and **T**s for Transformer Blocks. Read from left to right, the first letter references the block applied to the lowest resolution input features while the last letter references the block applied to the highest resolution input features. For example, U-Att-Small-TTUU describes the setting:

$$\begin{aligned} \text{DecBlock}_i(x) &= \text{U-Net-Block}(x) && i \in [1, 2] \\ \text{DecBlock}_i(x) &= \text{TransformerBlock}(x) && i \in [3, 4] \end{aligned}$$

The settings used in the Transformer blocks at each input feature resolution are shown in Table 4.2.

Input Feature Size	Setting	U-Att-		
		Small	Large	Full
$H \times W$	Input channels	-	$C_1 = 32$	
	Transformer Block	-	$R_1 = 32$	
		-	$H_1 = 1$	
		-	$E_1 = 4$	
$\frac{H}{2} \times \frac{W}{2}$	Input channels	-	$C_2 = 32$	
	Transformer Block	-	$R_2 = 16$	
		-	$H_2 = 1$	
		-	$E_2 = 4$	
$\frac{H}{4} \times \frac{W}{4}$	Input channels	$C_1 = 32$	$C_3 = 32$	
	Transformer Block	$R_1 = 8$	$R_3 = 8$	
		$H_1 = 1$	$H_3 = 1$	
		$E_1 = 4$	$E_3 = 4$	
$\frac{H}{8} \times \frac{W}{8}$	Input channels	$C_2 = 64$	$C_4 = 64$	
	Transformer Block	$R_2 = 4$	$R_4 = 4$	
		$H_2 = 2$	$H_4 = 2$	
		$E_2 = 4$	$E_4 = 4$	
$\frac{H}{16} \times \frac{W}{16}$	Input channels	$C_3 = 160$	$C_5 = 160$	
	Transformer Block	$R_3 = 2$	$R_5 = 2$	
		$H_3 = 5$	$H_5 = 5$	
		$E_3 = 4$	$E_5 = 4$	
$\frac{H}{32} \times \frac{W}{32}$	Input channels	$C_4 = 256$	$C_6 = 256$	
	Transformer Block	$R_4 = 1$	$R_6 = 1$	
		$H_4 = 8$	$H_6 = 8$	
		$E_4 = 4$	$E_6 = 4$	

Table 4.2.: Settings of Transformer Blocks used in U-Att Decoder at each input stage

4.4. Evaluation Metrics

IoU and dice score The most widely used evaluation metrics for semantic segmentation are the intersection over union (IoU) and dice score/percentage. Reported is the mean IoU (mIoU) and mean dice (mDice) score over all classes, as well as the class-wise IoU score. All scores are written as percentages.
Given an input image with pixels I, the IoU and Dice score of a class are calculated with the models' predicted pixels P_{pred} and the ground truth pixels P_{gt} for that class. We define sets true positives (TP), true negatives (TN), false positives (FP), and false

negatives (FP) as

$$\mathrm{TP} = \{x | x \in I \wedge x \in P_{\mathrm{pred}} \wedge x \in P_{\mathrm{gt}}\} \tag{4.1}$$
$$\mathrm{TN} = \{x | x \in I \wedge x \notin P_{\mathrm{pred}} \wedge x \notin P_{\mathrm{gt}}\} \tag{4.2}$$
$$\mathrm{FP} = \{x | x \in I \wedge x \in P_{\mathrm{pred}} \wedge x \notin P_{\mathrm{gt}}\} \tag{4.3}$$
$$\mathrm{FN} = \{x | x \in I \wedge x \notin P_{\mathrm{pred}} \wedge x \in P_{\mathrm{gt}}\} \tag{4.4}$$

Using these sets the IoU score of a class is calculated as

$$\mathrm{IoU} = \frac{|\mathrm{TP}|}{|\mathrm{TP}| + |\mathrm{FP}| + |\mathrm{FN}|} \tag{4.5}$$

and the dice score of a class as

$$\mathrm{Dice} = \frac{2 \cdot |\mathrm{TP}|}{2 \cdot |\mathrm{TP}| + |\mathrm{FP}| + |\mathrm{FN}|} \tag{4.6}$$

FLOPs Another point of comparison is floating point operations (FLOPs) needed for the forward pass of one input image. Reported are the FLOPs of each model for an input image of resolution 256×512 as GFLOPs (10^9 FLOPs). FLOPs give a valuable indication on the speed of the networks, as more operations typically result in longer computation times.

4.5. Comparison using the MiT-B0 encoder

We compare the proposed U-Att-Small and U-Att-Large decoder frameworks to the All-MLP decoder used in SegFormer. All decoders use the pre-trained MiT-B0 encoder as its backbone and are fine-tuned on the Cityscapes dataset as explained in 4.2. We use early stopping and use the models with the highest mIoU during training. Table 4.3 presents the mIoU, the mDice and the FLOPs of each model. Table 4.4 shows the class-wise IoU of different U-Att-Small settings in comparison to the All-MLP decoder, where as table 4.5 shows the class-wise IoU of different U-Att-Large settings in comparison to the All-MLP decoder.

Encoder	Decoder		Results		
			GFLOPs ↓	mIoU ↑	mDice ↑
	Framework	Settings			
MiT-B0	U-Att-Small	UUUU	**2.17**	61.99	74.70
		TUUU	2.23	62.20	74.86
		TTUU	2.40	64.10	76.60
		TTTU	2.56	65.36	77.66
		TTTT	2.80	66.20	78.37
		U**TTT**	2.73	65.70	78.01
		UU**TT**	2.56	65.36	77.70
		UUU**T**	2.41	65.09	77.47
	U-Att-Large	UUUUUU	5.46	63.86	76.38
		TUUUUU	5.52	63.49	76.06
		TTUUUU	5.69	64.38	76.81
		TTTUUU	5.85	65.10	77.42
		TTTTUU	6.09	66.42	78.50
		TTTTTU	7.04	66.18	78.34
		TTTTTT	10.84	**66.43**	**78.56**
	All-MLP		3.47	62.96	75.64

Table 4.3.: Comparison of different U-Att-Small and U-Att-Large settings with All-MLP decoder on MiT-B0 backbone

It can be observed, that the models using the U-Att-Small framework need less FLOPs than the baseline SegFormer model with the All-MLP decoder. U-Att-Large on the other hand has higher FLOPs than the baseline. The model with the highest accuracy score uses the U-Att-Large Framework, with transformer blocks over all six stages (TTTTTT). It achieves 66.43% mIoU and 78.56% mDice, outperforming the All-MLP model with 62.96% mIoU and 75.64% mDice significantly. This comes at an increase in computational cost, as it takes 10.84 GFLOPs in comparison to the baseline 3.47

GFLOPs. The best performing U-Att-Small setting also uses transformer blocks over all of its four stages (TTTT), achieving 66.20% mIoU and 78.37% mDice with 2.8 GFLOPs. This is an exceptional performance improvement in comparison to the baseline All-MLP decoder, because not only are mIoU and mDice higher, but the FLOPs are lower. We improved the computational cost as well as the accuracy of the network. The same is true for U-Att-Small Settings [TTUU, TTTU, UTTT, UUTT, UUUT].

Besides one outlier that can be attributed to the randomness of training neural networks, replacing a U-Net-Block with a transformer block resulted in a higher accuracy of the model with increased computational cost. Furthermore, it can be observed that a transformer block in the higher resolution stages tends to impact performance more than in the lower resolution stages, provided that stages input is the concatenation of encoder and decoder features.

When comparing U-Att-Large to U-Att-Small, it is noticable how the two added stages increase accuracy. This was assumed, as U-Att-Large applies a learned upsampling method to the output of U-Att-Small. It is to be expected that these additional trained parameters outperform the static bilinear upsampling applied to the $\frac{H}{4} \times \frac{W}{4}$ segmentation maps produced by U-Att-Small.
If the use of transformer blocks in those stages is beneficial can not be determined. The U-Att-Large TTTTUU and TTTTTT setting yield nearly identical accuracies while almost doubling the FLOPs through the replacement of the two U-Net blocks with two transformer blocks.

Class	U-Att-Small Decoder Settings								All-MLP
	UUUU	**T**UUU	**TT**UU	**TTT**U	**TTTT**	U**TTT**	UU**TT**	UUU**T**	
sidewalk	76.30	76.39	76.69	77.34	**77.68**	77.22	76.94	77.36	76.48
building	88.07	87.99	88.25	88.60	88.59	**88.64**	88.42	88.38	88.34
wall	48.51	50.36	55.31	**55.75**	54.88	51.62	53.78	53.96	51.49
fence	41.17	42.68	**44.75**	44.01	43.75	44.17	43.32	42.65	44.28
pole	41.23	41.26	40.54	41.76	44.26	**44.57**	43.16	43.31	38.69
traffic light	42.75	43.85	42.67	45.01	48.66	**49.34**	47.67	46.86	45.34
traffic sign	55.68	56.37	56.15	57.27	**59.71**	59.47	59.04	58.95	57.34
vegetation	88.98	88.98	88.86	89.18	**89.49**	89.47	89.24	89.15	89.08
terrain	59.21	58.94	59.96	60.19	**61.67**	59.98	61.37	59.29	60.00
sky	91.99	91.92	92.12	92.28	**92.36**	92.16	92.31	92.29	92.22
person	65.25	65.72	65.93	67.16	68.17	**68.23**	67.15	67.19	65.79
rider	39.70	41.19	43.10	43.77	46.46	**46.90**	43.92	43.56	41.55
car	89.32	89.41	89.51	90.20	**90.54**	90.03	90.36	90.22	88.85
truck	50.78	54.36	58.08	62.21	**64.88**	57.01	58.45	60.05	45.94
bus	65.25	64.10	68.93	71.34	**72.18**	70.65	69.80	69.89	65.93
train	35.82	30.56	50.64	52.48	50.37	54.95	**54.97**	50.87	41.67
motorcycle	39.44	39.15	38.78	43.41	43.36	43.00	41.80	43.18	**44.14**
bicycle	61.48	61.75	60.71	62.92	**63.88**	63.80	63.32	62.65	62.21

Table 4.4.: Class wise IoU-score: Comparison of different U-Att-Small settings with All-MLP decoder on MiT-B0 backbone

Class	U-Att-Large Decoder Settings							All-MLP
	UUUUUU	**T**UUUUU	**TT**UUUU	**TTT**UUU	**TTTT**UU	**TTTTT**U	**TTTTTT**	
road	96.85	96.87	**97.06**	96.86	97.00	96.96	97.02	96.86
sidewalk	76.41	76.79	77.38	76.48	77.36	77.16	**77.87**	76.48
building	88.36	88.38	88.49	88.36	88.71	**89.06**	88.91	88.34
wall	48.84	50.86	52.41	50.95	52.46	54.03	**57.19**	51.49
fence	42.82	43.42	43.28	43.40	45.08	**47.25**	45.79	44.28
pole	44.71	44.30	44.89	44.99	46.90	47.63	**48.35**	38.69
traffic light	43.48	45.18	45.50	44.74	49.65	49.77	**51.77**	45.34
traffic sign	57.69	57.69	57.92	59.03	60.76	61.13	**61.62**	57.34
vegetation	89.32	89.28	89.36	89.28	89.72	89.77	**89.91**	89.08
terrain	58.39	59.17	60.56	60.74	**61.58**	61.27	61.51	60.00
sky	92.44	92.34	92.66	92.36	92.59	92.82	**92.93**	92.22
person	66.65	66.78	66.68	67.17	68.31	68.47	**69.38**	65.79
rider	42.40	43.35	43.04	42.99	44.85	44.00	**45.84**	41.55
car	90.04	89.58	90.04	90.17	90.33	90.25	**90.57**	88.85
truck	59.39	52.80	60.15	61.70	**65.23**	59.76	61.37	45.94
bus	68.84	68.56	68.84	71.84	**75.06**	72.62	70.92	65.93
train	47.12	42.96	45.38	**57.51**	54.25	53.83	46.80	41.67
motorcycle	37.13	35.70	37.97	37.72	38.51	38.11	40.32	**44.14**
bicycle	62.47	62.26	61.67	60.56	63.67	63.58	**64.15**	62.21

Table 4.5.: Class wise IoU-score: Comparison of different U-Att-Large settings with All-MLP decoder on MiT-B0 backbone

4.6. Comparison using the MiT-B0-Full encoder

The U-Att-Full decoder framework is compared with the All-MLP-Full decoder. Both use MiT-B0-Full as its backbone. For additional comparison the standard All-MLP decoder was also trained on MiT-B0-Full, only using encoder features F_3 to F_6 as inputs. This model is very similar to the standard SegFormer model that served as baseline in the previous section. Identical to the MiT-B0 comparison, all models are fine-tuned on Cityscapes. We use early stopping and use the models with the highest mIoU during training. Table 4.6 presents the mIoU score, the mDice score and the FLOPs of each model. Tables 4.7 and 4.8 show the class-wise IoU score of the tested models.

Encoder	Decoder		Results		
			GFLOPs ↓	mIoU ↑	mDice ↑
	Framework	Settings			
MiT-B0-Full	U-Att-Full	UUUUUU	7.45	62.68	75.36
		TUUUUU	7.52	61.82	74.52
		TTUUUU	7.69	63.83	76.33
		TTTUUU	7.84	63.13	75.73
		TTTTUU	8.08	63.01	75.56
		TTTTTU	9.03	63.83	76.31
		TTTTTT	12.84	64.45	76.90
		U**TTTTT**	12.77	64.19	76.63
		UU**TTTT**	12.60	**65.20**	**77.62**
		UUU**TTT**	12.45	64.39	76.89
		UUUU**TT**	12.21	64.48	76.97
		UUUUU**T**	11.26	64.87	77.27
	All-MLP *		**3.95**	62.09	74.86
	All-MLP-Full		55.36	63.35	76.01

Table 4.6.: Comparison of different U-Att-Full settings with All-MLP decoder on MiT-B0-Full backbone

Depending on the setting, the U-Att-Full framework models need approximately two to three times the amount of FLOPs as the All-MLP* model, but significantly less than the All-MLP-Full model. The model with the accuracy uses the U-Att-Full Framework, with transformer blocks in the last four stages (UUTTTT). It achieves 65.20% mIoU and 77.62% mDice, outperforming the All-MLP* model - with 62.09% mIoU and 74.86% mDice - as well as the All-MLP-Full model - with 63.35% mIoU and 76.01% mDice - by a significant margin.
Disregarding one outlier, the All-MLP* decoder has a lower accuracy than all tested U-

*Only MiT-B0-Full Features F_3 to F_6 are used as input

Att-Full settings. The All-MLP-Full decoder performs worse than all U-Att-Full decoder settings using at least one transformer block in its last two stages.

Similar to the previous experiment, it can be observed that transformer blocks in the higher resolution stages impact performance more than in the lower resolution stages. The UUUUUT setting achieves a mIoU of 64.87% - the second best score in the comparison - while the TUUUUU setting has the lowest mIoU in the comparison with 61.82%, even though both use one transformer block.

The accuracy of the models with MiT-B0-Full is generally lower than the models using the MiT-B0 backbone. This is to be expected as the pre-training of MiT-B0 was significantly better. If given equally pre-trained backbones, it can be assumed that U-Att-Full would perform at a similar or slightly better accuracy compared to U-Att-Large and U-Att-Small simply due to the bigger network size. We can use the All-MLP decoder as a point of reference which achieved 62.09% mIoU with MiT-B0-Full and 62.96% mIoU with MiT-B0, indicating how the pre-training impacted the performance of the models. Comparing the TTTTTT setting on U-Att-Full and U-Att-Large we can also see a clear gap in accuracy with 64.45% mIoU to 66.43% mIoU.

Class	U-Att-Full Decoder Settings							All-MLP *	All-MLP-Full
	UUUUUU	**T**UUUUU	**TT**UUUU	**TTT**UUU	**TTTT**UU	**TTTTT**U	**TTTTTT**		
road	96.71	96.66	96.86	96.88	96.89	96.82	**96.99**	96.55	96.74
sidewalk	75.36	75.25	76.06	76.70	76.56	76.20	**77.33**	74.68	75.75
building	88.00	88.06	88.08	88.15	88.46	88.65	**88.76**	87.78	88.21
wall	47.61	44.70	48.95	**49.78**	49.29	46.48	47.75	49.56	48.89
fence	42.21	42.25	42.51	42.64	42.70	42.51	42.33	42.50	**44.67**
pole	44.76	45.22	45.36	45.66	47.02	47.00	**48.60**	39.46	42.95
traffic light	43.82	44.03	45.37	45.16	46.74	48.33	**48.67**	44.71	46.12
traffic sign	58.13	57.61	57.69	57.52	58.53	60.66	**61.22**	55.63	57.55
vegetation	89.34	89.41	89.47	89.37	89.56	89.63	**89.67**	88.98	89.44
terrain	57.91	57.41	56.83	58.56	58.42	**59.91**	55.02	57.17	57.56
sky	92.66	92.55	92.82	92.66	92.66	92.78	92.88	92.21	**93.12**
person	66.03	66.59	66.06	66.27	67.59	67.13	**67.79**	64.63	65.97
rider	41.91	**41.95**	40.00	37.19	41.95	41.00	40.79	38.68	40.47
car	89.54	89.65	89.82	89.76	89.70	89.56	**89.95**	89.07	89.17
truck	48.77	53.96	61.55	57.93	54.70	60.73	**61.74**	55.67	57.04
bus	65.76	62.85	**69.11**	65.73	64.68	66.05	68.20	65.45	65.78
train	49.41	31.32	**50.64**	39.97	32.34	44.17	46.46	39.35	44.87
motorcycle	30.90	33.77	33.88	37.80	36.44	32.95	**38.00**	36.46	37.18
bicycle	62.16	61.29	61.67	61.71	**62.92**	62.13	62.53	61.14	62.21

Table 4.7.: Class wise IoU-score: Comparison of different U-Att-Full settings with All-MLP-Full decoder on MiT-B0-Full backbone

Class	U-Att-Full Decoder Settings						All-MLP-Full
	UUUUU**T**	UUUU**TT**	UUU**TTT**	UU**TTTT**	U**TTTTT**	**TTTTTT**	
road	96.93	96.87	96.74	96.89	96.97	**96.99**	96.74
sidewalk	76.83	76.30	75.81	76.53	77.03	**77.33**	75.75
building	88.38	88.41	88.28	88.64	**88.86**	88.76	88.21
wall	45.93	47.41	**50.83**	48.63	50.80	47.75	48.89
fence	42.09	41.44	42.50	43.11	42.14	42.33	**44.67**
pole	46.79	48.34	48.40	**49.38**	47.96	48.60	42.95
traffic light	46.26	48.12	48.99	**50.85**	49.57	48.67	46.12
traffic sign	60.23	61.27	60.14	**61.66**	61.40	61.22	57.55
vegetation	89.58	89.76	89.63	**89.91**	89.82	89.67	89.44
terrain	58.13	58.97	**59.89**	58.62	59.04	55.02	57.56
sky	92.91	92.86	92.21	92.80	92.76	92.88	**93.12**
person	67.60	67.95	67.77	**68.65**	67.84	67.79	65.97
rider	**44.16**	41.43	40.84	43.91	39.83	40.79	40.47
car	**90.44**	90.26	89.97	89.89	90.00	89.95	89.17
truck	**62.04**	58.04	50.19	55.46	53.11	61.74	57.04
bus	**71.31**	66.03	64.70	66.83	68.81	68.20	65.78
train	47.70	48.45	**60.13**	52.76	47.90	46.46	44.87
motorcycle	**42.38**	39.81	34.27	40.44	33.38	38.00	37.18
bicycle	62.81	63.41	62.13	**63.83**	62.44	62.53	62.21

Table 4.8.: Class wise IoU-score: Comparison of different U-Att-Full settings with All-MLP-Full decoder on MiT-B0-Full backbone

Figure 4.1.: Sample output of some of the tested models

4.7. FLOPs comparison

In order to determine how the models' computational cost is impacted by different input image resolutions, the best performing U-Att decoder settings are compared to the All-MLP and All-MLP-Full decoder. Table 4.9 depicts the models' FLOPs needed for the forward pass of input images with increasing resolution.
The results show, that the All-MLP and All-MLP-Full decoder are less affected by increased input resolution than the U-Att decoder.

Encoder	Decoder	Input Resolution			
		[192, 192]	[256, 256]	[512, 512]	[768, 768]
MiT-B0	U-Att-Small-TTTT	**0.69**	**1.28**	**6.57**	20.31
	U-Att-Large-TTTTTT	2.41	4.63	28.04	98.80
	All-MLP	0.93	1.67	7.42	**19.47**
MiT-B0-Full	U-Att-Full-UUTTTT	2.91	5.52	31.47	106.03
	All-MLP-Full	15.52	27.62	111.21	252.98

Table 4.9.: GFLOPs of tested models forward pass with different image sizes

5. Conclusion

In this thesis, three very similar U-Att decoder frameworks for semantic segmentation were introduced and evaluated. The decoders use a step-by-step upsampling technique inspired by U-Net and SETR-PUP, in order to combine multi-level features - produced by the encoder - into an output segmentation map. Each U-Att decoder stage either applies standard convolution or efficient self-attention in combination with a feed-forward network on the input. Combining the different decoder architectures with the pre-trained SegFormer MiT-B0 backbone or a slightly modified version of it, all models are fine-tuned on the Cityscapes dataset using exactly the same settings.

Through the experiments, we can conclude, that there are huge benefits to learned upsampling techniques compared to static bilinear upsampling. Those benefits are not limited to accuracy increases at the expense of higher computational cost, as we showed multiple U-Att-Small decoder settings outperforming the baseline SegFormer decoder not only in mIoU and mDice but also with lower FLOPs. This accuracy can be further improved by adding additional stages to output segmentation maps of full image resolution.

The experiments also showed that self-attention-based transformer blocks are not only a valuable addition in the encoder, but can also hugely improve accuracy when used in the decoder. Especially if applied on higher resolution input features, transformer blocks seem to add a significant accuracy boost while increasing computational costs.

List of Figures

List of Tables

Bibliography

[1] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *CoRR*, vol. abs/1505.04597, 2015. arXiv: 1505.04597. [Online]. Available: http://arxiv.org/abs/1505.04597.

[2] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez, and P. Luo, "Segformer: Simple and efficient design for semantic segmentation with transformers," *CoRR*, vol. abs/2105.15203, 2021. arXiv: 2105.15203. [Online]. Available: https://arxiv.org/abs/2105.15203.

[3] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017. arXiv: 1706.03762. [Online]. Available: http://arxiv.org/abs/1706.03762.

[4] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," *CoRR*, vol. abs/1411.4038, 2014. arXiv: 1411.4038. [Online]. Available: http://arxiv.org/abs/1411.4038.

[5] A. Dosovitskiy, L. Beyer, A. Kolesnikov, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *CoRR*, vol. abs/2010.11929, 2020. arXiv: 2010.11929. [Online]. Available: https://arxiv.org/abs/2010.11929.

[6] C. Sun, A. Shrivastava, S. Singh, and A. Gupta, "Revisiting unreasonable effectiveness of data in deep learning era," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 843–852. DOI: 10.1109/ICCV.2017.97.

[7] W. Wang, E. Xie, X. Li, *et al.*, "Pyramid vision transformer: A versatile backbone for dense prediction without convolutions," *CoRR*, vol. abs/2102.12122, 2021. arXiv: 2102.12122. [Online]. Available: https://arxiv.org/abs/2102.12122.

[8] O. Russakovsky, J. Deng, H. Su, *et al.*, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015. DOI: 10.1007/s11263-015-0816-y.

[9] S. Zheng, J. Lu, H. Zhao, *et al.*, "Rethinking semantic segmentation from a sequence-to-sequence perspective with transformers," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 6877–6886. DOI: 10.1109/CVPR46437.2021.00681.

[10] I. Loshchilov and F. Hutter, "Fixing weight decay regularization in adam," *CoRR*, vol. abs/1711.05101, 2017. arXiv: 1711.05101. [Online]. Available: http://arxiv.org/abs/1711.05101.

[11] M. Cordts, M. Omran, S. Ramos, *et al.*, "The cityscapes dataset for semantic urban scene understanding," *CoRR*, vol. abs/1604.01685, 2016. arXiv: 1604.01685. [Online]. Available: http://arxiv.org/abs/1604.01685.

A. Example outputs

Figure A.1.: Example output of baseline and U-Att-Small models

Figure A.2.: Example output of U-Att-Large models

Figure A.3.: Example output of U-Att-Full models