CHALMERS
UNIVERSITY OF TECHNOLOGY

Lane Detection with Attention Mechanisms

Master's thesis in Engineering Mathematics and Computational Science

OLLE MÅNSSON
THERESE GARDELL

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021

Master's thesis 2021

Lane Detection with Attention Mechanisms

OLLE MÅNSSON
THERESE GARDELL

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2021

Lane Detection with Attention Mechanisms
OLLE MÅNSSON
THERESE GARDELL

Supervisor: Lennart Svensson, Department of Electrical Engineering
Advisors: Erik Werner & Adam Tonderski, Zenseact
Examiner: Lennart Svensson, Department of Electrical Engineering

Master's Thesis 2021
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2021

Lane Detection with Attention Mechanisms
OLLE MÅNSSON
THERESE GARDELL
Department of Electrical Engineering
Chalmers University of Technology

Abstract

For autonomous vehicles, perception of the surroundings is key. In order to avoid collisions and to keep vehicles on the road, the identification of lane markers is of the utmost importance. Lane detection, the task of detecting lanes on a road from a camera, is traditionally performed with convolutional neural networks. These networks excel when you want to capture local relationships in an image, which is often key in computer vision tasks. However, lanes can stretch across the input image or be occluded by vehicles, debris or other alien objects on the road. This raises the question of whether global perception or long-range dependencies are also needed to effectively detect lanes under challenging conditions similar to those described above. Convolutional neural networks, by design, have a limited receptive field or a limited capacity to capture long-range dependencies.

In this thesis, we explore how attention mechanisms, from the field of natural language processing, can be used to augment convolutional neural networks and improve their ability to capture long-range dependencies. We combine a residual neural network encoder with a U-net style decoder and a self-attention mechanism in the form of a Transformer encoder. The model architecture is evaluated over a range of encoder-decoder configurations and the addition of an attention mechanism improves semantic segmentation performance across the board. The results are validated on both a synthetic dataset engineered to require long-range dependencies as well as on a lane detection dataset captured in real traffic.

Keywords: Attention, Deep Learning, Computer Vision, Semantic Segmentation, Lane Detection, Autonomous Driving

Acknowledgements

First of all we wish to express our gratitude to our academic supervisor and examiner Lennart Svensson. Lennart has showed a great interest throughout the project and provided countless exciting ideas and a fantastic support. We would also like to thank our advisors at Zenseact, Erik Werner and Adam Tonderski. Our weekly meetings and around-the-clock correspondences have truly made this project possible. Their relentless dedication and technical expertise have been extremely helpful in critical and tricky situations. Finally, we would like to thank Zenseact AB and all employees for providing crucial support and resources throughout the project.

Olle Månsson & Therese Gardell, Gothenburg, June 2021

Contents

List of Figures

List of Tables

1

Introduction

Autonomous vehicles have the potential to increase safety on the road by substantially reducing the number of accidents caused by human error. In addition, there will likely be environmental and economical benefits as well since autonomous vehicles are expected to improve on areas such as fuel efficiency and traffic flow.

At the heart of the autonomous driving problem, we find the need to be able to perceive the environment. Without knowledge of our surroundings, it is impossible to make safe and sound decisions. We can gain this knowledge through a number of different sensors mounted on the vehicle, such as radar, lidar and cameras. In this thesis, we focus on the latter. Computer vision allows us to interpret images from these cameras and gain a higher-level understanding of what they capture. The most common perception techniques include object detection and image segmentation. In object detection you try to assign each object with a corresponding bounding box and a class label. In image segmentation you classify each and every pixel.

Image segmentation can in turn be divided into instance segmentation and semantic segmentation. While instance segmentation assigns separate labels for each instance of a class, semantic segmentation does not and for example assigns all cars with the label *car*. In this thesis the task is limited to lane detection and since we do not wish to differentiate between different instances of for example a dashed lane we will be using semantic segmentation.

Like most computer vision tasks, semantic segmentation has been dominated by convolutional neural networks (CNNs). This class of deep neural networks has fewer parameters to learn compared to fully connected neural networks. This has allowed for greater resolution of the input image as well as greater depth and scale of the network architecture. CNNs particularly excel when you want to capture local relationships of an image, since they have limited receptive fields. This can however lead to difficulties when you need to exploit non-local relationships.

The attention mechanism [1] and the Transformer architecture [2], from the field of natural language processing (NLP), have in the past couple of years become a popular area of research within the field of computer vision [3]. The attention mechanism, contrary to CNNs, was introduced specifically to capture non-local relationships by enabling all parts of the input to attend to each other directly. In lane detection,

which is the focus of this thesis, the targets can stretch across wide image frames and be partly occluded by vehicles or other objects. These cases can be difficult for a CNN alone to fully grasp. By incorporating attention mechanisms to the network it can gain a more global perception of the input image.

1.1 Problem formulation

The thesis aims to investigate the effects of attention mechanisms on semantic segmentation of lane markers. The task, applicable in the scene of autonomous driving, has the objective to label each pixel of a road image as either lane marker or background. This is usually done with convolutional neural networks but recent advances [4], [5] in the field of computer vision suggest that attention mechanisms can be a powerful addition. However, since attention-augmented computer vision is a rather new field it is currently not widely used in the automotive industry.

By presenting an attention-augmented model and comparing it to a baseline free from attention the thesis attempts to answer the following research questions:

- How can existing semantic segmentation models for lane detection based on convolutional neural networks be augmented with attention mechanisms to improve their quantitative performance?
- How can these attention mechanisms be incorporated without dramatically increasing the training and inference time?
- What is the qualitative performance of the attention-augmented model compared to the baseline?

In this thesis, the quantitative performance is measured by calculating the Intersection over Union (IoU) score of the predicted segmentation map, further described in Subsection 2.3.1. The inference time is measured by calculating the time it takes for the model to predict a segmentation map for one image. Qualitative performance on the other hand is harder to measure since it involves subjective measurements like the perceived quality of the segmentation map, but can nevertheless be an important tool for evaluating the full capabilities of a model.

1.2 Scope

Some attention-based models for computer vision tasks have achieved very impressive results by leveraging massive computational resources and datasets. For example, the Vision Transformer [6] for image classification took 2500 TPUv3-core days to train on a proprietary dataset with 300 million training samples. If you were to train this model on the public cloud, where each TPUv3-core costs approximately $1 per hour as of June 2021, it would cost somewhere around the neighbourhood of $60000. This makes pursuing a similar model cost-prohibitive for this project and we limit our scope to models with more modest computational requirements.

In the autonomous driving industry, real-time performance is a must. To achieve this, models are fine-tuned to optimize their inference speed and memory footprint. However, this is a time-consuming process. For the purpose of fully exploring the research questions of this project, the model will not be required to run in real-time, instead the focus will be aimed at the qualitative performance. With that said, we measure and evaluate the inference time for different model configurations to at least get an upper bound for the impact of the attention mechanism.

The models are trained on two separate datasets. The first, based upon the Pathfinder challenge [7], is synthetically generated to provide a problem that requires non-local reasoning. The second is provided by Zenseact and contains footage collected by a car mounted with a front camera. Both datasets provide ground truth segmentation masks.

As described in Section 1.1 the segmentation task is limited to two classes, the lane markers and the background. The lane marker class contains both solid and dashed lane markers.

1.3 Contribution

The main contribution of this thesis is an attention-augmented CNN-based model for performing semantic segmentation of lane markers. The results show that by adding a Transformer encoder to the bottleneck of a U-Net inspired CNN-based model the qualitative performance can be improved significantly while maintaining a feasible inference time. The model is evaluated over a range of encoder model sizes, decoder model sizes and transformer model sizes to enable a thorough analysis of the trade-off between quantitative performance and computational complexity.

One of the main factors that separates this thesis from previous work is that the model is built specifically for computer vision tasks for autonomous vehicles. The focus is on performing semantic segmentation of lane markers. This enables us to fine-tune the model architecture towards this task. With that being said, the model is constructed so that it can be generalized to other tasks as well, such as semantic segmentation of cars and pedestrians and even object detection.

1.4 Related works

In **Attention Augmented Convolutional Networks** [8], the authors also identify that CNNs have been the architecture of choice in many computer vision applications, but that they only operate on a local neighborhood. They propose to combine convolutions with self-attention mechanisms by concatenating convolutional feature maps with feature maps produced by the attention mechanisms. Their experiments show that this architecture improves accuracy on a variety of datasets for image classification as well as object detection.

While they do use a self-attention mechanism, they do not use the transformer

architecture that has been so successful in many other deep learning applications. They also focus their performance comparison on a trade-off between accuracy and the number of trainable parameters, which does not necessarily translate well to real-time performance.

An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale [6] introduces a vision model based on the original transformer only, with no convolutional networks at all, called Vision Transformers (ViT). Instead of treating every pixel like a token in the input sequence to the transformer, they instead divide the input image into a grid of square patches, with the width and height of 16 pixels, which acts as the input sequence instead. After extensive pretraining on a proprietary dataset and finetuning on downstream image classification datasets, they show that their model can outperform comparable CNNs in image classification.

Vision Transformers are without a doubt successful in the computer vision domain, but their extensive pretraining can be a hurdle for other researchers who wish to replicate and/or extend their work. As previously mentioned in the scope of this thesis, ViTs are trained on a proprietary dataset with over 300 million images and the total training time required for their best model takes over 2500 TPUv3-core days to train [6]. Another issue is the fact that the original ViT architecture only works for image classification, while in this thesis we focus on dense predictions like semantic segmentation tasks.

In **End-to-End Object Detection with Transformers** [4], the authors propose DETR - a object detection model based on both the encoder and the decoder side of the transformer. Similarly to the model proposed in this thesis, DETR has a CNN based encoder backbone with a transformer encoder acting on the final feature map produced by said backbone. They then employ a transformer decoder to directly predict bounding boxes for the object detection task. DETR outperformed existing object detection models while running at comparable inference speeds compared to existing two-stage models.

However, even the smallest and fastest DETR model runs at less then 30 frames per second, which could be too slow for some automotive applications. Inference speed is also only reported for the object detection setting, so while they show that DETR can be extended to predict segmentation masks as well, this is done by adding yet another stage to the model which most likely lowers the inference speed even further.

2

Theory

This chapter provides an introduction to the methods used in the thesis as well as the tasks and objectives used to evaluate the methods. First we present key concepts of deep learning leading up to the convolutional neural network and the concept of learning. Then we present the attention mechanism and the Transformer that is under investigation in this thesis. Finally we present the task of semantic segmentation.

2.1 Deep learning

Deep learning is a subfield of Artificial Intelligence (AI). The term Artificial Intelligence was coined by computer scientists in the 1950s. This field of research was founded on the assumption that human or so called natural intelligence "can be so precisely described that a machine can be made to simulate it" [9].

The process of simulating natural intelligence can take many different forms. In the earlier days of artificial intelligence, research was mainly focused on systems where intelligence was represented by a large set of handcrafted rules. This approach worked well for applications like chess playing computer programs but for most other tasks, it is not feasible to create such a ruleset by hand. But what if computer programs could learn to perform these tasks on their own, without carefully handcrafted rules? This question is explored in the machine learning field and its subfield deep learning.

2.1.1 Machine learning

Machine learning (ML) goes beyond following a set of human engineered rules and is centered around the concept of machines learning these rules on their own. Machine learning systems are trained by being exposed to examples, often referred to as training data, relevant to a task. By leveraging statistical structures in training data, the system can then learn to construct a set of (often abstract) rules which then can be used to predict new examples. In other words, a machine learning system can learn tasks without any explicit instructions from the programmer to do so.

If the training data is labeled, as in the training data is provided as a list of pairs of input examples and desired output examples, then we refer to the machine learning

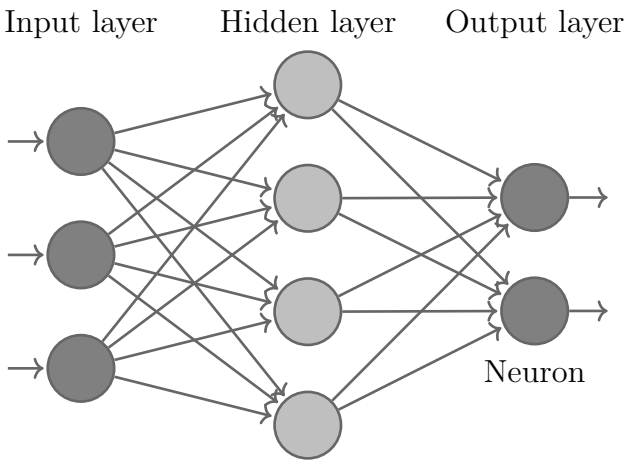

Figure 2.1: Artificial neural network.

system as supervised learning. Supervised learning is the type of machine learning that will be leveraged in this thesis. The two most fundamental components of a supervised learning system is the model used to predict an output given an input and the learning method used to train the model. The concept of learning will be introduced in later sections. For now, we will focus on models and algorithms of machine learning.

2.1.2 Artificial neural networks

Artificial neural networks (ANNs) is a class of machine learning algorithms whose building blocks are vaguely inspired by biological neural nets or animal brains. Just like our own cerebrum, the ANN includes components like neurons and synapses. These neurons are connected in a network of connections or synapses, hence the name neural network. With that said, artificial neural networks are just inspired by animal brains and they are not models of a real brain. In particular, the way ANNs are usually trained is not biologically plausible [10].

The feed-forward neural network (FNN) is one of the simplest types of artificial neural networks. Here, the neurons are connected from input to output without any loops. In other words, the graph representing the connections of the neurons do not form a cycle. Other types of networks allow information to propagate both forward and backward, but those types of networks are not considered in this thesis.

In most artificial neural networks, including FNNs, the neurons are divided into layers. These layers often include an input layer, an output layer and a number of hidden layers. Each layer is then composed of a number of neurons. If all neurons in one layer are connected to all neurons in the previous layer, we say that the layer is fully connected. An example of a fully connected feed-forward neural network with one hidden layer can be seen in Figure 2.1.

Each neuron in the artificial neural network has a set of inputs or connections from other neurons in the network. Each input is associated with a certain *weight*. The

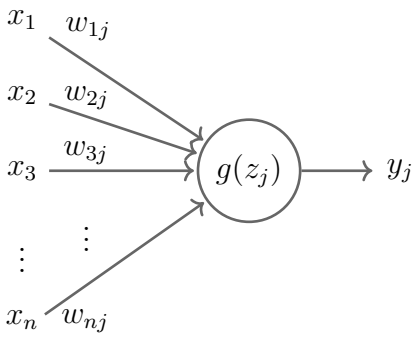

Figure 2.2: Single neuron.

weighted sum of the inputs, together with an optional *bias* term, is known as the neuron's *activation*. The weights and bias of the neuron are trainable parameters to the model. Each weight represents the relative importance of the input or the connection in questions. The bias term can be trained to shift the activation into a more suitable range. Finally, the neurons output is the result of an *activation function* which is applied to the activation. For a fully connected layer, the output y_j of neuron j in layer l can be described with

$$y_j = g\Big(\sum_{i=1}^{n} x_i w_{ij} + b_j\Big) \tag{2.1}$$

where x_i is the output from neuron i in the previous layer $l-1$ with n neurons and g is the activation function. Here, the weight on the connection between neuron j and neuron i in the previous layer is denoted as w_{ij} and the bias for neuron j as b_j. An illustration of a single neuron can be found in Figure 2.2.

The activation function, or $g(z_j)$ in (2.1) where z_j denotes the weighted sum of the inputs shifted by the bias, is a key component of an artificial neural network. If the activation function would be chosen as a linear function, the ANN would only be able to represent linear or so called affine transformations of the input space [11]. While some tasks can be solved effectively with such transformations, it is often too restrictive. Therefore, it is common practice to apply a non-linear activation function, such as

$$\sigma(z) = \frac{1}{1+e^{-z}} \tag{2.2}$$

which is also known as the sigmoid activation function. This activation function has the property of being continuously differentiable, which can be beneficial when the weights of the network are updated, see Subsection 2.1.5. In short, the gradient of the activation function is used and thus the magnitude of the derivative of the activation function can be important. For the sigmoid, the derivative is defined for all $z \in \mathbb{R}$ and $\frac{\partial \sigma}{\partial z} \in (0,\, 1)$.

While being continuously differentiable is often a desirable property for an activation function, the fact that the sigmoid's gradient is limited to (0, 1) can lead to issues. In short, for neural networks with multiple hidden layers, the gradient is multiplied

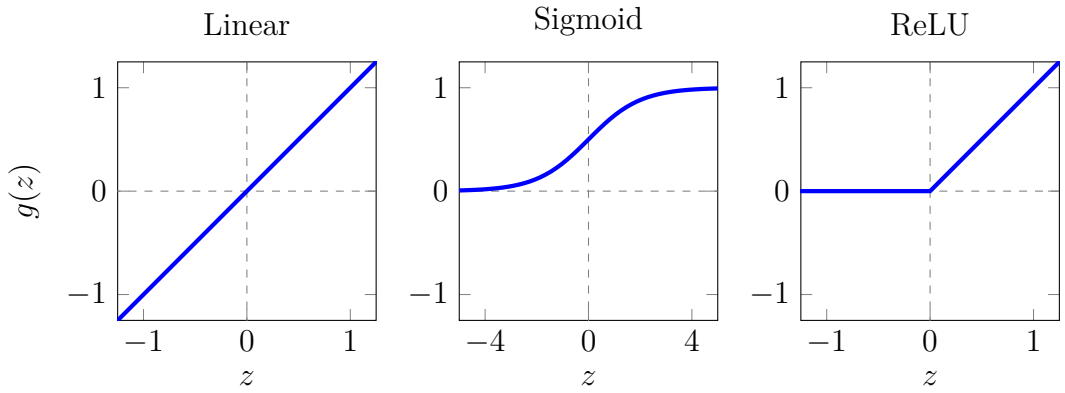

Figure 2.3: Linear, sigmoid and ReLU activation functions.

several times for each hidden layer, so the result tends to zero as the number of hidden layers increase. This phenomenon is known as the vanishing gradient problem [12]. Another problem with the sigmoid activation function is the number of operations required to calculate the activation.

The Rectified Linear Unit (ReLU) activation function, defined as the positive part of its argument

$$g_{\text{ReLU}}(z) = \max(0, z), \tag{2.3}$$

does not suffer from the vanishing gradient problem to the same degree, since the derivative of ReLU is either 0 or 1. It is also considerably cheaper to compute, compared to the sigmoid activation function. Therefore, it has been a popular choice for artificial neural networks with many hidden layers, especially in the computer vision field. The linear, sigmoid and ReLU activation functions are depicted in Figure 2.3.

2.1.3 Deep neural networks

Deep neural networks is a subfield to both deep learning and artificial neural networks. The "deep" in deep learning and deep neural networks refers to the representation depth. For artificial neural networks, the depth is then the number of layers in the network, including input and output layers. For simple ANNs, like feed-forward neural networks, a deep neural network has a number of hidden fully-connected layers. Such a network is also commonly referred to as a multilayer perceptron or MLP.

Modern deep neural networks are composed of tens or even hundreds of hidden layers. Together with non-linear activation functions, these networks can learn complex representations of the input data. Deep feed-forward neural networks have been successful at solving many difficult tasks in a variety of domains [13]. However, for certain fields, such as computer vision, a standard multilayer perceptron operating directly on the input image can be inefficient and even infeasible.

An image is composed of a number of pixels in a number of channels. For standard

color image, each pixel has 3 channels; red, green and blue. In a multilayer perceptron, where every pixel in every channel is connected to every other pixel in every channel, the number of connections and the number of parameters grow quickly. Simple MLPs also ignores the spatial structure of the input data, which means that pixels that are far apart in the image are treated in the same way as pixels that are close together. For computer vision tasks, the most important representations can usually be learned from spatially local relationships and thus it is inefficient to have fully connected layers. In the past decade, most researches and practitioners of computer vision have therefore been focused on another type of deep neural network called convolutional neural network.

2.1.4 Convolutional neural network

Like multilayer perceptrons, a convolutional neural network (CNN) is composed of an input layer, a number of hidden layers and an output layer. These layers can include fully connected layers, but most layers in a CNN are so called convolutional layers. Other types of layers that are commonly used in CNNs are pooling layers, which introduce some translational invariance to the network. These layers and the receptive field are presented below.

Convolutional layer

A convolutional layer performs a series of spatial convolutions over the given input data. Each convolution is performed by a kernel, which can be seen as a small sliding window with a certain size that slides across the input data. While convolutional layers can be applied to one-dimensional data such as time series, we will from now on limit ourselves to 2D convolutional layers which are commonly used in computer vision.

The three-dimensional input to the convolutional layer is often referred to as the feature map, with two spatial axes, H and W, as well as a depth axis - also known as the channels axis [14]. The depth axis is not to be confused with the depth of the network. The shape of the feature map is usually denoted as $H \times W \times C$. Note that H and W can, although not necessarily, represent the height and width of the image. They can also represent the height and width of an intermediate representation of the input image, after passing through a number of hidden layers. For example, when applying a convolutional layer to the input image in color, the channel axis C is set to 3, representing the red, green and blue channel. As we slide a single kernel over a three-dimensional feature map, we produce a new two-dimensional feature map.

Given a certain kernel size and the feature map, this layer type also has some additional parameters such as the stride, the type of padding and the number of output channels. The stride and the type of padding will determine the number of valid locations for the kernel in the feature map. The stride determines the spatial shift from each kernel to the next kernel. It is possible to use a different stride in the horizontal and vertical direction, but we limit ourselves to symmetrical stride in this thesis. The stride is often set to 1, meaning that the distance between the centers of

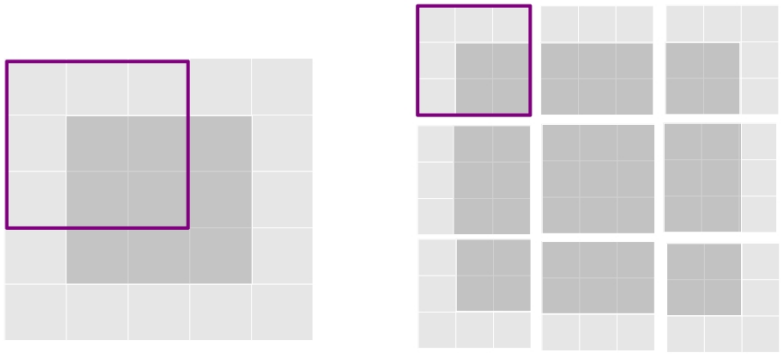

Figure 2.4: Illustration of the process of passing an unpadded feature map through a convolutional layer with kernel size 3 and stride 1. Without padding the convolutional layer reduces the spatial dimensionality of the input from 5×5 to 3×3.

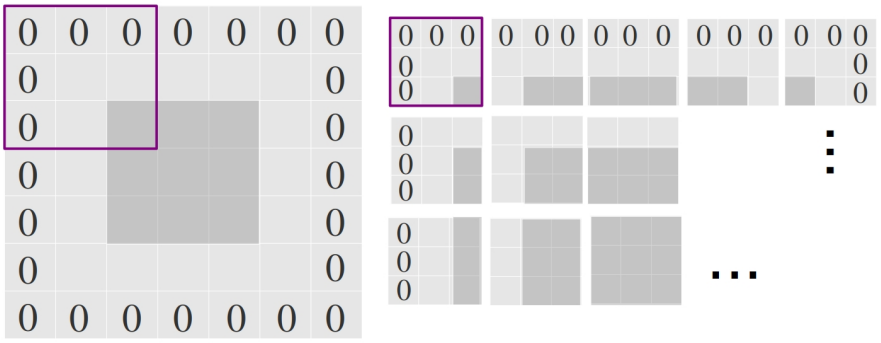

Figure 2.5: Illustration of the process of passing a zero-padded feature map through a convolutional layer with kernel size 3 and stride 1. With zero-padding the convolutional layer maintains the spatial dimensionality of the input.

two valid locations of the kernel is 1. If the stride is greater than 1, the convolutional layer is usually referred to as a strided convolution. Strided convolutions can be used to downsample the spatial dimensionality of the feature map. This is covered when we introduce pooling layers further down in the section.

Even with the stride set to 1, the number of valid locations for the kernel can be less than the number of spatial positions in the feature map, see Figure 2.4. To ensure that the resulting feature map after a convolutional layer has the same spatial dimensionality, padding can be used. With a technique called zero-padding, an appropriate amount of zeros is appended to the borders of the feature map to ensure that a kernel can be centered around every spatial position in the feature map. See Figure 2.5 for a visualisation of zero-padding.

The final parameter covered in this section is the number of output channels. This

number is decided by, and equal to, the number of filters/kernels in the convolutional layer. In a convolutional layer, it is commonplace to not only let a single kernel slide over the feature map but rather to have a set of kernels. The resulting feature map produced by each kernel is then stacked along the channels axis. This means that for a convolutional layer, with stride 1 and zero-padding, we have

$$H \times W \times C_{\text{in}} \rightarrow H \times W \times C_{\text{out}} \tag{2.4}$$

where C_{in} is the number of input channels and C_{out} is the number of output channels.

With the basic parameters covered, it can be important to get an understanding of the motivation behind using convolutional layers in a neural network. First off, convolutional layers naturally exploit spatial locality since the kernel has a limited size. By stacking many convolutional layers in a deep convolutional neural network together with non-linear activations, we get increasingly global and more abstract representations of the input space. This approach has empirically been proven to be fairly successful for many computer vision tasks [15], [16].

What enables the stacking of layers is how parameter efficient convolutional neural networks are compared to feed-forward neural networks. Kernel weight sharing, as in each kernel has a fixed set of weights for each position in the feature map, allows for a dramatic reduction of the number of trainable parameters. Given a 256×256 input image with 3 channels, the number of parameters for a single neuron in a fully connected layer is $H \cdot W \cdot C = 196\ 608$. Meanwhile, the number of parameters for a single kernel of size 3×3 is $3 \cdot 3 \cdot C = 27$, regardless of the spatial size of the feature map.

Pooling layer

Convolutional neural networks often include a number of pooling layers. The pooling layer reduces the spatial dimensionality of the feature map, mainly to reduce the amount of computation needed in the network. While the number of parameters for a convolutional layer does not depend on the spatial dimensionality, the amount of computations needed for that layer does depend on the amount of valid positions on the feature map, which in turn depends on the spatial dimensionality. Therefore, it is common to see pooling operations periodically interleaved in the CNNs to progressively reduce the dimensionality. This can also help combat a phenomenon known as overfitting, see Subsection 2.1.5. Additionally, pooling layers can also introduce some translational invariance which can make the network more robust [17].

Pooling is often implemented similarly to a strided convolution with stride 2 and kernel size 2×2. Instead of multiplying each window with weights, an operator is applied spatially. A commonly used operator is max pooling, where each position in the resulting feature map is the result of a max operation of each window. A max pooling layer will therefore effectively downsample both the width and the height of a feature map with a factor of two.

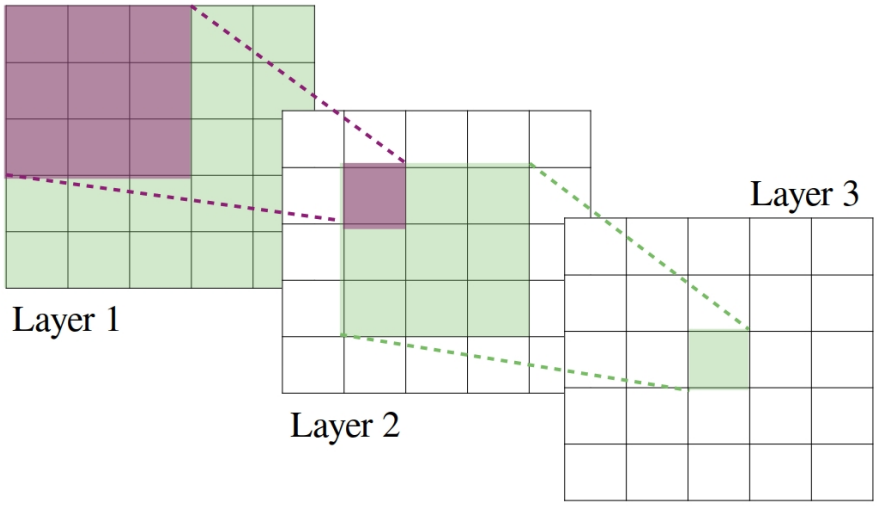

Figure 2.6: The receptive field for a CNN with three convolutional layers, with kernel size 3 and stride 1.

Another alternative to max pooling is to use a strided convolutional layer. With stride 2, we also achieve the desired reduction in spatial dimensionality. With a strided convolutional layer, with stride 2 and zero-padding, we have

$$H \times W \times C_{\text{in}} \rightarrow \frac{H}{2} \times \frac{W}{2} \times C_{\text{out}} \tag{2.5}$$

where C_{in} is the number of input channels and C_{out} is the number of output channels, resulting in a 75% reduction in the number of activations.

Receptive field

The receptive field of a convolutional neural network is defined as the size of the region in the input image that produces a feature in the final feature map. In other words, which input signals may affect certain output signals. Note that fully connected neural networks have a global receptive field. This is not necessarily the case for convolutional neural networks. The receptive field for a CNN with three convolutional layers, with kernel size 3 and stride 1, is depicted in Figure 2.6. The receptive field in the figure shows a theoretical upper limit. For each convolutional layer in the CNN, the input/output signal becomes more indirect.

2.1.5 Learning

The previous sections introduced deep neural networks including convolutional neural networks, but glanced over how these networks are trained or how they learn better values for their weights and biases. In this section, the concept of learning will be introduced, along with loss functions and optimization.

In machine learning, learning can be described as the adaptation of a model to better handle a task by considering training data. For deep learning, the model in question is a deep neural network. Adaptation of the model can be described as

the adjustment of parameters such as the weights and biases to further improve the performance of the model. Adaptation or adjustment is done to maximize a certain objective, or minimize a loss function.

Learning can be divided into roughly three subfields where the main differentiator is how the machine learning model is exposed to the training data. In *reinforcement learning*, the model interacts with an environment, creating a feedback loop between the model, the learning system and the experiences collected from interaction with the environment [14]. In *unsupervised learning*, the model is exposed to unlabeled data where the goal can be to learn the true distribution of the dataset or to cluster similar datapoints together based on their features [14]. Finally, we have *supervised learning*, which is the subfield we will focus on in this thesis.

In supervised learning, the model is exposed to data where each datapoint is associated with a label or target. From here on we will denote the datapoint with x and the target with y. The labels of the data can then be viewed as being provided by a teacher or a supervisor, hence the term supervised learning. Supervised learning includes tasks such as regression and classification as well as more intricate variants of these two. In classification, the goal is to learn the correct mapping from input data to known targets or annotations, given a set of labeled examples. To measure how well a certain model can perform these tasks, the predicted mapping and the target is fed into a loss function. The loss function not only provides an indication of the performance of the model - it is also a key component in the training of the model since it influences how the weights are updated. This will be covered when we introduce the concept of optimization.

Loss functions

In a deep learning context, the loss function, which is also sometimes referred to as a cost function or an error function, is used to measure or quantify model errors. In most deep learning tasks, the loss function is not part of the task definition, so the loss function has to be chosen carefully to represent the error. In many cases, the loss function is also chosen based on desirable properties such as convexity.

For regression, the mean squared error (MSE) loss function, also known as the L2-loss, is a popular choice. It is defined as

$$L_{\text{MSE}}(y,\, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.6}$$

where y_i and $\hat{y}_i$ is the target and the prediction for sample i out of n total training samples. While MSE loss can be used for classification as well, other loss functions are often used, since they can have a more reasonable probabilistic interpretation and can lead to faster training as well [18].

A common choice for classification is the binary cross entropy (BCE) loss defined as

$$L_{\text{BCE}}(y,\, \hat{y}) = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right] \tag{2.7}$$

where $\hat{y}_i \in (0, 1]$ is the prediction and $y_i \in \{0, 1\}$ the binary target. When used in the classification task semantic segmentation the n in (2.7) is the number of pixels in the image. For binary and multi-class segmentation the targets are usually given in the format of one-hot encoding, meaning that each class has a separate bitmap where the pixel-value is 1 if the pixel belongs to the class and 0 otherwise. This enables the loss to be used for settings with more than two classes. When the number of classes is more than two it is usually referred to as categorical cross entropy.

For certain imbalanced segmentation problems the binary cross entropy loss is not sufficient. For example in the lane detection problem. Lane markers that go off into the distance naturally appear smaller in the image and thus their pixel-wise loss does not impact the total loss as much as lane markers nearby. To compensate for this you can calculate a weighted loss that gives a higher-value loss to pixels further up in the image. This does not generally work for objects far away but lane markers are often centered in the image and thus generally shrink the further up you get. In the thesis this is referred to as distance-weighted binary cross entropy loss and defined as

$$L_{\text{DW-BCE}}(y, \hat{y}) = -\frac{1}{n}\sum_{i=1}^{n} w_i \left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right] \tag{2.8}$$

where $w_i \in \mathcal{W}^{H\times W}$ is the weight for pixel i. Here $\mathcal{W}^{H\times W}$ is the weight matrix where $H \times W$ is the size of the image. The bottom row elements of $\mathcal{W}$ are set to 1 and the above increase row-wise by a set value.

Optimization

With a loss function in place, which represents some measurement of the error of the model, the process of learning or training the model is then equivalent to minimizing this loss function. The parameters or weights used in the model are updated according to an optimization algorithm. In deep learning, most optimization algorithms are based on the *gradient descent* method.

For a model $f(x, \theta)$, where x is the input data and θ is the set of parameters of the model, we define the gradient of the loss function as

$$\nabla_\theta \bar{L} = \frac{1}{n}\sum_{i=1}^{n} \frac{\partial L(y_i,\, f(x_i;\, \theta))}{\partial \theta} \tag{2.9}$$

where $\bar{L}$ is the average of L over all datapoints. With gradient descent, we then update our existing parameters according to

$$\theta \leftarrow \theta - \eta \nabla_\theta \bar{L} \tag{2.10}$$

where η is the *learning rate*. Here, we are considering the full dataset x along with targets y. When computing the loss and the gradient over the full dataset, the algorithm is often referred to as batch gradient descent. However, using batch gradient descent can be problematic for several reasons. First of all, it can be computationally prohibitive to calculate the gradient for larger datasets. Batch

gradient descent has also shown poor performance in practice, often converging to a local minima in the loss landscape.

Instead, stochastic gradient descent (SGD) has proven to be very successful for deep learning. In stochastic gradient descent, a small subset of the training dataset is used to calculate the gradient for each parameter update. The size of the subset is fixed, regardless of the size of the dataset. It is therefore possible to train a model on a dataset with billions of training examples, using updates computed only on tens or hundreds of examples. These small subsets are usually called batches or minibatches, not to confuse with batch gradient descent. With minibatch size m, the update becomes

$$\theta \leftarrow \theta - \eta g, \quad g = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial L(y_i,\, f(x_i;\, \theta))}{\partial \theta}. \tag{2.11}$$

The smaller size of the minibatches can also prevent convergence to local minima. Gradients calculated on smaller subsets are more noisy, which can offer a regularizing effect [19]. However, the noise also adds instability to the training process, resulting in the need for a lower learning rate to maintain stability. Using a small minibatch size, such as 1, together with a reduced learning rate can result in a significantly high runtime since the optimization algorithm needs to take more steps [19]. It will also take a larger number of steps to observe the whole dataset. Therefore, the choice of minibatch size can control the balance between runtime, hardware constraints and the need for regularization.

Optimization methods are an active area of research and many methods have been proposed as alternatives to SGD. For some machine learning tasks and with some deep learning models, SGD is notorious for being hard to tune. More specifically, it can be hard to find a suitable learning rate [20]. Variants of SGD such as *Adam* [21], employ adaptive per-parameter learning rates and have been proven to be robust and successful for deep learning models that utilizes so called attention mechanisms [20], which we will cover in Section 2.2.

2.2 Attention

In natural language processing (NLP), which like computer vision can be described as a subfield of deep learning, the goal is to process large amounts of text. Tasks include language modeling, language parsing and translation. In the past decade, so called sequence to sequence models (seq2seq) have been widely used in sequence prediction tasks, such as language modelling and machine translation. In seq2seq models, the recurrent neural network (RNN) is a key component. Compared to feed-forward neural networks, covered in Subsection 2.1.2, recurrent neural networks have feedback connections. There are several variants and extensions of RNNs used in practice, with long short-term memory (LSTM) being one of the most popular.

In this thesis, we will not cover RNNs and LSTMs in depth, since they are not used in our models. However, they can assist in the introduction of attention mechanisms,

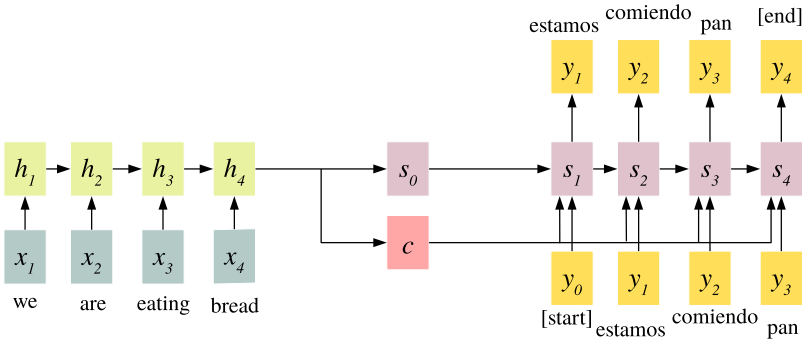

Figure 2.7: The architecture of a recurrent neural network used for translating a sentence from English to Spanish.

since attention mechanisms were introduced as an augmentation to NLP models, more specifically LSTMs.

In a seq2seq model for text translation, the input to the model is a sequence of word embeddings or tokens, $(x_1, x_2, \ldots, x_N)$, where N is the sequence length. The input is processed sequentially in what is usually referred to as the encoder side of the model. For each token in the encoder, the RNN is processing not only the specific token itself, but also the output from the processing of the previous token. This output can be seen as a representation of all previous tokens, in what is usually referred to as the hidden state or the context vector. For input tokens $(x_1, x_2, \ldots, x_N)$, the hidden states can be denoted as $(h_1, h_2, \ldots, h_N)$.

On the decoder side the output of the model, $(y_1, y_2, \ldots, y_M)$, is generated one token at a time and the length of the output sequence M can, but does not have to, be equal to N. The generated token for one position in the sequence is used as the input for the next position. Again, for each token in the output, the RNN is processing not only the specific token in question, but the hidden states from the previous position is also taken into account, here denoted as $(s_1, s_2, \ldots s_M)$. Additionally, the final context vector from the input sequence, h_N, is also taken into account, see Figure 2.7 for reference. The hidden states from previous tokens as well as the context vector from the input sentence are what enables the RNN to capture so called long-range dependencies.

However, the model's ability to capture long-range dependencies is greatly reduced with respect to the length of the input sequence. Since every input token has to be represented with a single context vector, the path length of information can be long for a input sequence with a great number of tokens. In addition, another limitation of standard seq2seq models are the every token in the decoder has access to the same context vector representation of the input sequence. One could imagine that different tokens in the target sentence, in a translation task, could be more or less related to certain tokens in the input sentence.

Attention mechanisms have been proposed as a solution to both of these limitations.

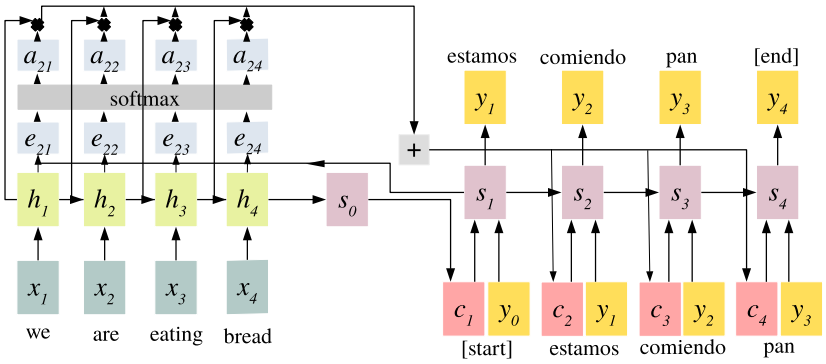

Figure 2.8: The architecture of an attention-augmented recurrent neural network used for translating a sentence from English to Spanish.

Rather than building a shared context vector for all the tokens in the decoder, attention allows for direct connections to each hidden state in the encoder, with customizable weights for each output token, see Figure 2.8.

Before we introduce a more formal definition of the attention mechanism, let us first consider a normal seq2seq2 model. Let

$$x = (x_1, x_2, \ldots, x_N) \quad \text{and} \quad y = (y_1, y_2 \ldots, y_M) \tag{2.12}$$

represent the input and the target sequence respectively. The context vector c is set to the last hidden state of the encoder, h_N. For position j in the target sequence, y_j becomes

$$y_j = f_j(y_{j-1},\, c) = f_j(y_{j-1},\, h_N) \tag{2.13}$$

where f_j represents the processing done by the RNN for that specific position.

Now, let us instead consider the same setup but with an attention mechanism. We then have a unique context vector c_j for each position in the target sequence

$$c_j = \sum_{i=1}^{N} a_{j,i} h_i \tag{2.14}$$

where $a_{j,i}$ represents the attention weight for position i in the input sequence. The attention weight determines how much of each hidden state from the input sequence that is to be considered for position j in the target sequence. There is a number of different methods for constructing suitable attention weights, but for this thesis we will focus on the scaled dot-product attention mechanism [2]. In scaled dot-product attention, we use a so called alignment score function $e_{j,i}$ defined as

$$e_{j,i} = \frac{s_j \cdot h_i}{\sqrt{d}} \tag{2.15}$$

where d is the embedding dimension of the input sequence hidden state. The attention weight for position j in the decoder sequence and for the position i is then softmax over all alignment scores, defined as

$$a_{j,i} = \frac{\exp(e_{j,i})}{\sum_{k=1}^{N} \exp(e_{j,k})}. \tag{2.16}$$

With the attention weights defined, we now have

$$y_j = f_j(y_{j-1}, c_j) = f_j(y_{j-1}, \sum_{i=1}^{N} a_{j,i} h_i) \tag{2.17}$$

which compared to (2.13) has more direct and customizable connections to every hidden state from the decoder.

2.2.1 Transformers

Although attention-augmented RNNs outperformed other seq2seq models at the time when they were developed [1], they still suffered from some of the drawbacks with recurrent neural network. One of the most prominent limitations with RNNs is the fact that the input and output sequences have to be processed sequentially. This can lead to slow training and slow inference times. The Transformer architecture on the other hand, does not suffer from these kind of limitations.

Transformers were introduced as an alternative model for seq2seq models, complete with an encoder and a decoder, but unlike most other NLP models it uses no recurrent neural networks and only attention. Therefore, this architecture lends itself very well to parallelization and it has been the model of choice for NLP problems in the past years [22]. An overview of the Transformer model can be found in Figure 2.9.

The key component in the Transformer is the multi-head self-attention mechanism. With self-attention, also known as intra-attention, the input sequence is allowed to attend to itself. The input sequence is projected into queries, keys and values, and the attention output is given by

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^\top}{\sqrt{n}})V \tag{2.18}$$

where n is a scaling factor and Q, K and V are defined as the linear projections

$$Q = W_Q x, \quad K = W_K x, \quad V = W_V x \tag{2.19}$$

where x is the input sequence and W_Q, W_K and W_V weight matrices to be learned. The multi-head self-attention mechanism is also visualized in Figure 2.10.

The multi-head attention mechanism used in the Transformer is based on scaled dot-product attention. However, rather than computing the attention weights once, multi-head attention calculates the scaled dot-product attention multiple times in parallel. The output from these attention mechanisms are then concatenated. According to the original authors of the Transformer architecture, the multi-head attention allows the Transformer to jointly attend to information from different representation subspaces at different positions [2].

In this thesis, we will mainly focus on the encoder side of the Transformer, also known as the Transformer encoder. The encoder consists of several stacked encoder layers. Each layer has a multi-head self-attention mechanism, a position-wise fully connected

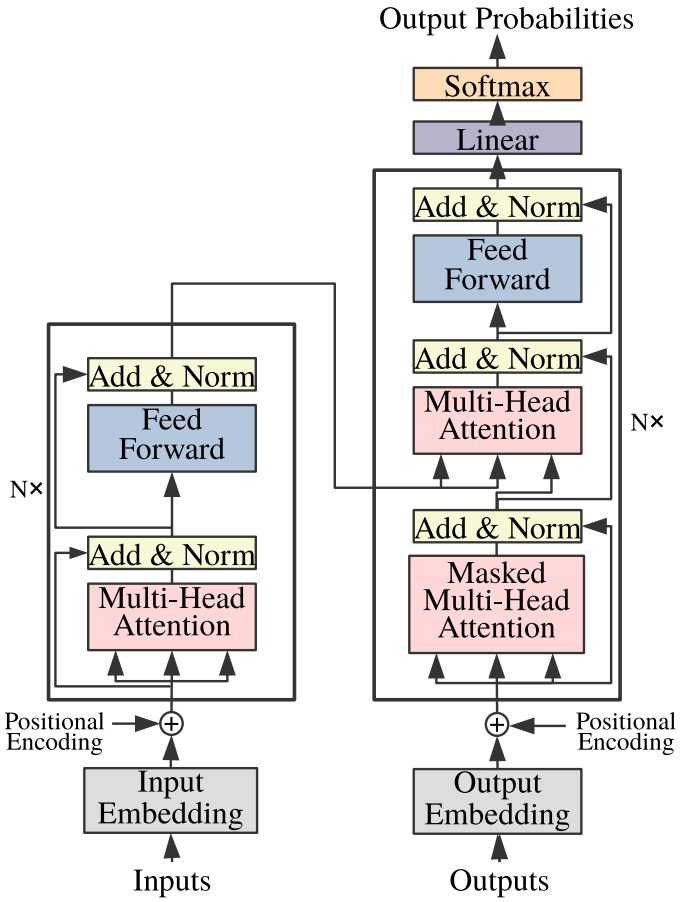

Figure 2.9: An overview of the Transformer model introduced in [2].

feed-forward neural network, a normalization layer and residual connections, see Figure 2.10. The number of layers and the number of heads in the multi-head attention mechanism are hyperparameters that can be tuned for the task in question.

Since the Transformer was developed for NLP, it needs to operate on sequences of tokens. However, the Transformer architecture naturally operates on a set of tokens. To preserve positional information in the input sequence, a positional encoding is often added to the queries, keys and values. In the original Transformer, the authors experimented with both learned positional encodings and a fixed sinusoidal pattern. They found that the two variants produced nearly identical results [2].

2.2.2 Linformer

The attention mechanism can be computationally expensive and add significant slowdown in inference time. Therefore, so called efficient Transformers is an active field of research, with dozens of articles published only in the last year. The proposed efficient Transformer models can be divided into different categories, including factorized attention patterns, learnable attention patterns, sparse attention patterns, kernelization and low-rank approximations of the attention matrix [23].

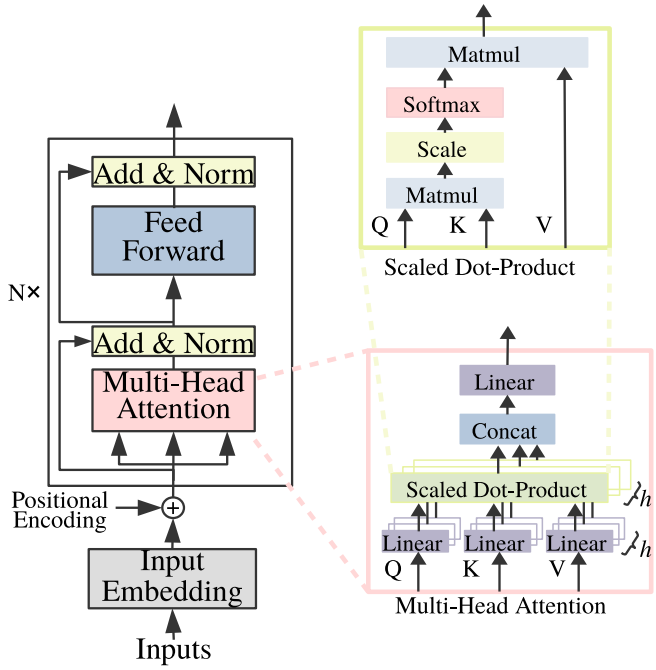

Figure 2.10: The Transformer encoder as introduced in [2]. To the right you find illustrations of the multi-head attention module from the Transformer encoder and the scaled dot-product from the multi-head attention module.

Linformer is an efficient Transformer based on low-rank approximation [24], which uses a self-attention mechanism with linear space and time complexity. The Linformer achieves this by decomposing the scaled dot-product attention matrix with linear projections, which forms a low-rank factorisation of the original attention matrix. To perform this decomposition, you need to assume a fixed sequence length, denoted k, which can be a problem for NLP applications. However, that is not necessarily a problem in other domains like computer vision. The Linformer has been used successfully in production applications and it has also proven to be successful in synthetic benchmarks [25].

2.3 Semantic segmentation

In computer vision there are different ways for the computer to gain perception of an image. We have the simplest technique, image classification, where the image is assigned a label corresponding to the dominant object of the image. In order of detail this is followed by object detection where multiple objects in the image are assigned labels and corresponding bounding boxes. Finally, we have image segmentation which goes a step further and performs pixel-wise classification. Image segmentation can in turn be divided into semantic and instance segmentation. In this thesis we will be using semantic segmentation.

Semantic segmentation is a classification task with the objective of assigning each pixel of an image with a class label generating a so called segmentation mask. This

Figure 2.11: Example image of semantic segmentation, from the Cityscapes Dataset [26].

is illustrated by assigning a corresponding color as shown in Figure 2.11. The class label can, for example, be *Car*, *Pedestrian* or *Road*. With this technique you treat every instance, belonging to the same class, as one, such that all cars are assigned with the label *Car*. This can be compared to instance segmentation where you assign a label to each instance of a class separately, for example, *Car-1*, *Car-2* and so on. For our problem we, for example, want all lane markers assigned to the same class.

The complexity and detail of the segmentation problem is decided by a set of predefined class labels and the corresponding ground truth segmentation maps used to train the classifier. If an object is not included in the set of class labels it is usually assigned the label background. The chosen class labels of our problem are defined in Section 3.5 where the datasets are presented.

2.3.1 Metrics

When an image has been segmented, you need a way to evaluate the performance. One approach is to calculate the overall pixel accuracy which is defined as

$$\text{Accuracy} = \frac{\text{Correctly classified pixels}}{\text{All pixels}}.$$

This metric can however be misleading when your image is imbalanced and, for instance, consists mostly of background. As you will come to see, this is the case for the problem of this thesis. A more appropriate approach is using Intersection over Union (IoU). For each class of the image you calculate an IoU-score which is defined as the intersection of the ground truth segmentation and the predicted segmentation, divided by their union, as illustrated in Figure 2.12.

To get a total score of a segmented image you calculate the mean IoU defined as

$$\text{mIoU} = \frac{1}{n}\sum_{i}^{n} \text{IoU}_i, \tag{2.20}$$

where IoU_i is the IoU-score of class i and n is the number of classes. Depending on the segmentation problem the background class can be left out of the calculations. Both IoU and mIoU return a score in the range $[0, 1]$, where a score of one represents a perfect overlap.

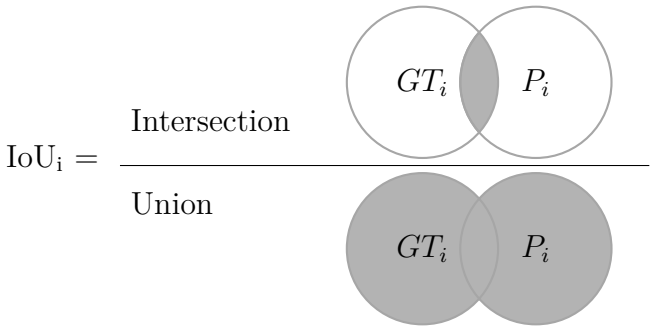

Figure 2.12: Intersection over union, where GT_i represents the area of the ground truth segmentation and P_i represents the area of the predicted segmentation for class i.

For this thesis we only have two classes, regardless of dataset, where the second class is the background class. For the problem at hand it is sufficient to calculate the IoU for the first class. We will refer to this as class specific IoU.

3

Methods

In this chapter the methods, models and datasets used for evaluating the effects of the attention mechanisms are described. We describe the U-Net [27] and the ResNet [28] that have provided inspiration for important building blocks of the resulting models. This is followed by our U-net baseline, which is a network free from attention mechanisms. The attention-augmented model presented next is constructed from the baseline and referred to as Transformer U-net. Both models have the objective of performing semantic segmentation on two separate datasets, that are presented next. Finally, the experiments are presented which includes the setup for training and evaluation.

3.1 U-Net

There are various published network architectures that output a segmented image. In this thesis we will use a network that is inspired by the U-Net architecture. A simplified illustration of the original architecture can be seen in Figure 3.1. The U-Net was introduced in [27] with the purpose of segmenting biomedical images such as cell membranes and neural structures.

The model performs supervised learning in an encoder-decoder manner. On a high level, the model is a convolutional network that first down-samples the input image to a low-resolution feature map (encoder), using convolutions and max pooling. Next it up-samples the feature map to a high resolution feature map (decoder), using transposed convolutions. To calculate the error of the prediction a pixel-wise softmax in combination with cross entropy loss is applied to the final feature map. The number of channels in the final feature map matches the number of classes of the segmentation problem. The skip connections are the key elements of the U-Net. By sharing information between the encoder and the decoder, the model can salvage spatial information and resolution that would otherwise have been lost in the process.

What does not show in the simplified illustration in Figure 3.1 is that the encoder uses unpadded convolutions and cropping before passing the feature map down and across. This results in lower resolution feature maps in the decoder (right) compared to the encoder (left). For comparison, in our model we use zero-padding to maintain spatial resolution between convolutional layers and the spatial resolution is symmetric

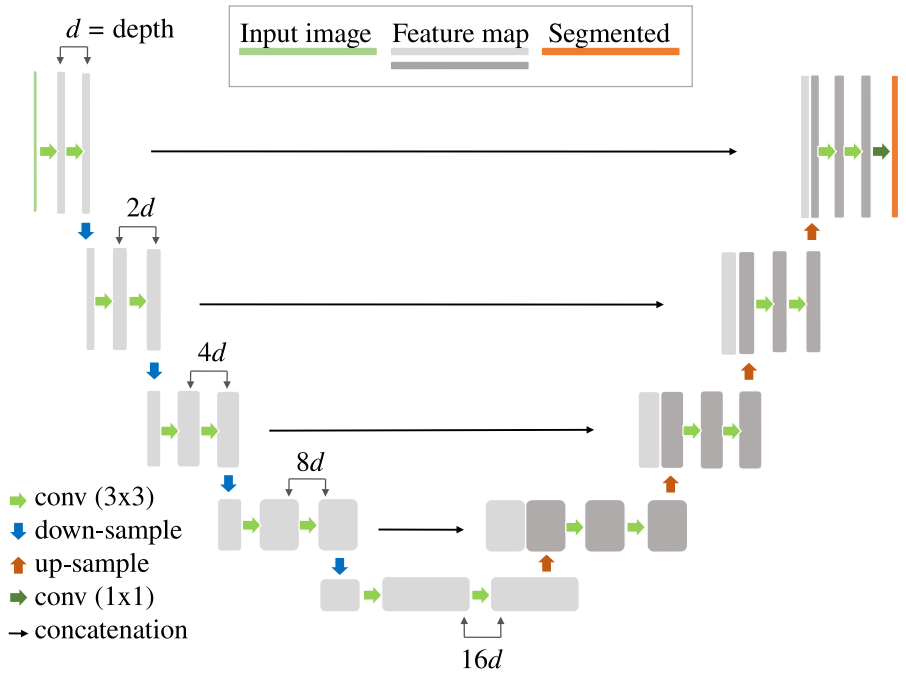

Figure 3.1: The U-Net architecture. To the left you find the down-sampling path also known as the encoder and to the right the up-sampling path also known as the decoder. In between, the arrows represent the skip connections that provide the decoder with feature maps from the encoder. The depth represents the number of channels.

across the skip connections.

3.2 ResNet

In the previous section we presented the original U-Net that has inspired our model. One big difference between the original U-Net and our implementation is that we use a residual neural network as our contracting path.

The residual neural network (ResNet) was introduced in [28] as a response to the degradation problem that arise when training "very deep" neural networks: When increasing the number of layers in a deep neural network one might expect to increase its performance due to the additional number of trainable parameters. However, this is not necessarily the case, when the number of layers gets to a certain point the accuracy of the network gets saturated and then decreases significantly [28].

To solve this problem [28] introduces deep residual learning that utilizes identity mapping by skip connections adopted to every few following convolutional layers in the deep network. This network is referred to as a ResNet. The ResNet is built up by so called residual blocks such as the one in Figure 3.2 that are stacked one after the other. The residual block in Figure 3.2 shows an example with two convolutional layers. The feature map x is passed through the convolutional layers, $F(x)$, and

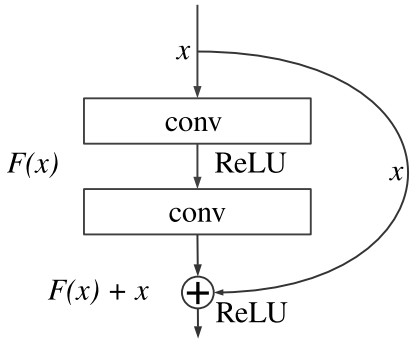

Figure 3.2: The architecture of a residual block from a ResNet [28].

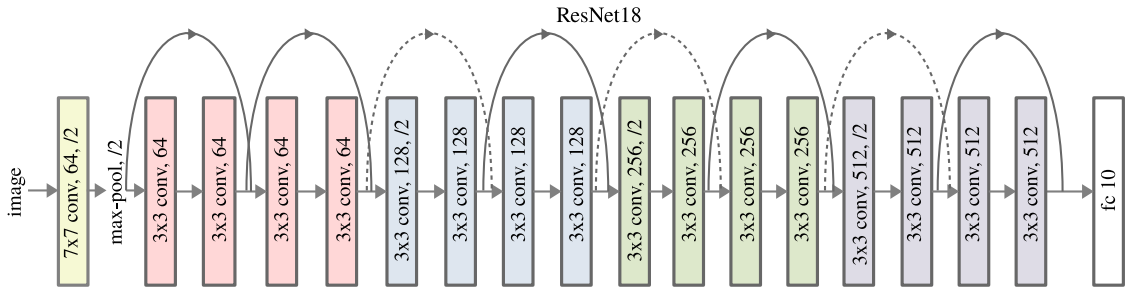

Figure 3.3: The architecture of ResNet18 as introduced in [28].

on the other side the initial feature map is element-wisely added to the output by a skip connection, $F(x) + x$. This is referred to as identity mapping above. From here on we will refer to these skip connections as short cuts so they are not confused with the skip connections of the U-net. The resulting feature map is then passed through a rectified linear unit (ReLU) activation function before it is passed to the next block. The identity mapping is parameter free and hence does not add to the computational complexity. However, if the convolutional layers resize the feature map then the short cuts needs to resize the initial feature map accordingly which adds to the number of trainable parameters by utilizing a 1×1 convolutional layer with a fitting stride.

The different model configurations presented in [28] are referred to as ResNetX where X represents the number of convolutional layers in the network. The smallest model they present is ResNet18 which is used to a certain degree in this thesis. ResNet18 is seen in Figure 3.3 where eight residual blocks are stacked. The short cuts that coincide with the resized feature maps are represented by dashed lines. The first layer is a 7×7 convolutional layer followed by a 3×3 max-pooling operation. Then you find the residual blocks, with 3×3 convolutional layers. The final layer of the ResNet utilizes global averaging pooling and a 1000-way fully connected layer with a softmax. The final layer is not used in our implementation.

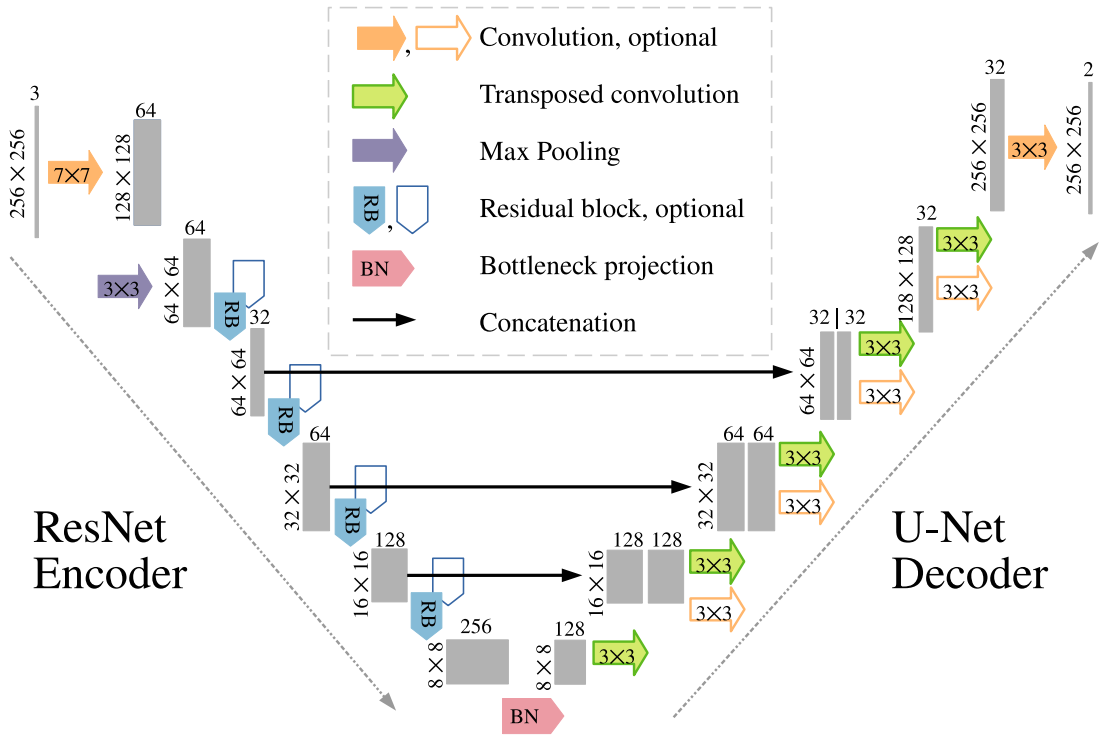

Figure 3.4: Network architecture of the baseline model. The gray boxes represent multi-channel feature maps. To the left of each box you find the spatial resolution given that the input has resolution $H \times W = 256 \times 256$. On top of the boxes you find the number of channels. The convolutional and pooling arrows are marked with the corresponding kernel-size. The hollow arrows represent optional layers that are used in certain model configurations, see Subsection 3.3.2.

3.3 U-net Baseline

In this section the network architecture and different model configurations of the baseline model are presented. The baseline is used as a reference when evaluating the performance of the attention-augmented model presented in Section 3.4.

3.3.1 Network architecture

The network architecture is presented in Figure 3.4. The overall structure is inspired by the U-Net architecture described in Section 3.1. Similar to U-Net, the baseline model has skip connections that concatenate feature maps from the encoder to the decoder. In contrast to U-Net, the encoder of this model is a residual neural network (ResNet) described in Section 3.2. The model comes in five different configurations of varying magnitude that will be presented in Subsection 3.3.2. The layers affected by the configurations are denoted *optional* in Figure 3.4.

The model takes as input an image with three channels that first goes through the down-sampling path of the encoder. The encoder is responsible for picking up various features of the image. This is done with convolutional layers in the ResNet that

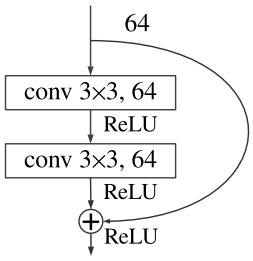

Figure 3.5: Residual block with 3×3 convolutions and 64 filters.

sequentially expand the feature dimension and lower the spatial dimension. The image is now referred to as a feature map. The feature map is passed through the bottleneck projection to the decoder. The decoder upsamples the feature map near symmetrically to the encoder. For the three lowest stages the feature maps are concatenated with the feature maps from the encoder. Then the feature map is further upsampled in the spatial dimension while maintaining the channel dimension. Finally the feature map is passed through a convolutional layer that projects the features to a two-channel feature map matching the number of classes. The values of the elements are in logits. To obtain a segmented image the map is passed through an argmax-function not shown in the figure.

The convolutional layers throughout the network use zero-padding to maintain the spatial dimensionality of the feature maps. Alterations in the spatial dimension are performed by strided convolutions if nothing else is mentioned. Below the architecture of the parts of the model are described in detail.

Encoder. As mentioned, the encoder is a ResNet. In accordance with the original ResNet [28] the first part of the encoder, consists of a 7×7 convolutional layer of stride 2 followed by batch normalization, a rectified linear unit (ReLU) and a 3×3 max-pooling layer of stride 2. The remaining part of the encoder is built up by four stages of one or two (depending on the model configuration) residual block(s) like the one in Figure 3.5, where the channel dimension varies. The residual blocks are in turn built up by two 3×3 convolutional layers, each followed by batch normalization and a ReLU. As shown in Figure 3.4 the last three stages provide a spatial reduction which is achieved by letting the first one of the convolutional layers of each stage, as well as the corresponding short cut, have a stride of value 2. The last three stages also provide the decoder with a copy of the feature map through a skip connection.

Bottleneck. Between the encoder and the decoder you find a bottleneck projection that uses a 1×1 convolutional layer to project the feature map to half the number of channels. The spatial dimensionality is kept intact.

Decoder. The decoder up-samples the output from the bottleneck with transposed 3×3 convolutional layers of stride 2. Each convolutional layer is followed by batch normalization and a ReLU. The first three stages of the decoder receive a feature map from the encoder which is concatenated to the up-sampled feature map. Depending on the model configuration an additional 3×3 convolutional layer can be added

Table 3.1: Configuration of the residual blocks in the encoder. Within the braces you find the kernel-size and the number of filters of the convolutional layers of each residual block. To the right of the braces you find the number of residual blocks of each stage. Stage 2-4 performs down-sampling with a stride of 2. Note that x-large is inspired by ResNet18 as presented in [28].

stage	output size	tiny	small	medium	large	x-large (ResNet18)
0	$H/2 \times W/2$	7×7, 64, stride 2				
1	$H/4 \times W/4$	3×3 max pool, stride 2				
		$\left.\begin{matrix}3\times3,32\\3\times3,32\end{matrix}\right\}\times 1$	$\left.\begin{matrix}3\times3,32\\3\times3,32\end{matrix}\right\}\times 1$	$\left.\begin{matrix}3\times3,32\\3\times3,32\end{matrix}\right\}\times 1$	$\left.\begin{matrix}3\times3,32\\3\times3,32\end{matrix}\right\}\times 1$	$\left.\begin{matrix}3\times3,32\\3\times3,32\end{matrix}\right\}\times 2$
2	$H/8 \times W/8$	$\left.\begin{matrix}3\times3,64\\3\times3,64\end{matrix}\right\}\times 1$	$\left.\begin{matrix}3\times3,64\\3\times3,64\end{matrix}\right\}\times 1$	$\left.\begin{matrix}3\times3,64\\3\times3,64\end{matrix}\right\}\times 1$	$\left.\begin{matrix}3\times3,64\\3\times3,64\end{matrix}\right\}\times 2$	$\left.\begin{matrix}3\times3,64\\3\times3,64\end{matrix}\right\}\times 2$
3	$H/16 \times W/16$	$\left.\begin{matrix}3\times3,128\\3\times3,128\end{matrix}\right\}\times 1$	$\left.\begin{matrix}3\times3,128\\3\times3,128\end{matrix}\right\}\times 1$	$\left.\begin{matrix}3\times3,128\\3\times3,128\end{matrix}\right\}\times 2$	$\left.\begin{matrix}3\times3,128\\3\times3,128\end{matrix}\right\}\times 2$	$\left.\begin{matrix}3\times3,128\\3\times3,128\end{matrix}\right\}\times 2$
4	$H/32 \times W/32$	$\left.\begin{matrix}3\times3,256\\3\times3,256\end{matrix}\right\}\times 1$	$\left.\begin{matrix}3\times3,256\\3\times3,256\end{matrix}\right\}\times 2$	$\left.\begin{matrix}3\times3,256\\3\times3,256\end{matrix}\right\}\times 2$	$\left.\begin{matrix}3\times3,256\\3\times3,256\end{matrix}\right\}\times 2$	$\left.\begin{matrix}3\times3,256\\3\times3,256\end{matrix}\right\}\times 2$

Table 3.2: Configuration of the convolutional layers of the encoder. Transposed convolutional layers are written in lightface, $3 \times 3,\ 64$, and regular convolutional layers are written in boldface, $\mathbf{3 \times 3},\ \mathbf{64}$, where 3×3 refers to the kernel-size and 64 to the number of filters.

stage	output size	tiny	small	medium	large	x-large
0	$H/16 \times W/16$	$3 \times 3, 128$				
1	$H/8 \times W/8$	$3 \times 3, 64$	$3 \times 3, 64$	$3 \times 3, 64$	$3 \times 3, 64$	$3 \times 3, 64$ $\mathbf{3 \times 3, 64}$
2	$H/4 \times W/4$	$3 \times 3, 32$	$3 \times 3, 32$	$3 \times 3, 32$	$3 \times 3, 32$ $\mathbf{3 \times 3, 32}$	$3 \times 3, 32$ $\mathbf{3 \times 3, 32}$
3	$H/2 \times W/2$	$3 \times 3, 32$	$3 \times 3, 32$	$3 \times 3, 32$ $\mathbf{3 \times 3, 32}$	$3 \times 3, 32$ $\mathbf{3 \times 3, 32}$	$3 \times 3, 32$ $\mathbf{3 \times 3, 32}$
4	$H \times W$	$3 \times 3, 32$	$3 \times 3, 32$ $\mathbf{3 \times 3, 32}$	$3 \times 3, 32$ $\mathbf{3 \times 3, 32}$	$3 \times 3, 32$ $\mathbf{3 \times 3, 32}$	$3 \times 3, 32$ $\mathbf{3 \times 3, 32}$

to certain stages. These layers are denoted optional in Figure 3.4. The final stage applies a 3×3 convolutional layer of stride 1 that projects the feature map such that the number of channels matches the number of classes of the segmentation task. In this thesis the number of classes is always two.

3.3.2 Model configurations

As mentioned in Subsection 3.3.1 there are five different model configurations of varying magnitude. For the encoder this magnitude reflects the number of residual blocks and for the decoder the number of additional convolutional layers. In Figure 3.4 these additional blocks and layers are denoted optional and illustrated with a hollow arrow. The encoder configurations are presented in Table 3.1 where the residual blocks for stages 1-4 are in braces. The x-large configuration is inspired by ResNet18, that is shown in Figure 3.3, and the smaller models are scaled down versions. The decoder configurations are presented in Table 3.2 where the transposed convolutional layers are written in lightface and the optional convolutional layers are written in boldface.

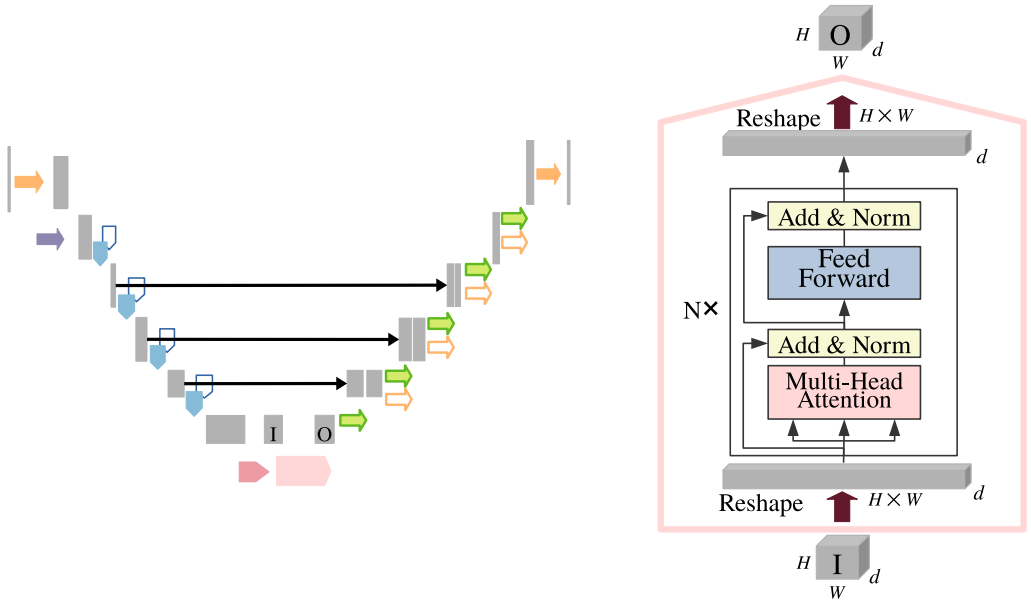

Figure 3.6: Network architecture of the attention-augmented model. The feature map denoted 'I' in the network (left) is reshaped and passed through the Transformer encoder (right). In the figure to the right you see one layer of the Transformer encoder, the remaining of the N layers are identical and stacked directly after the first. After the final layer the feature map is reshaped again before being passed to the decoder. Here H and W represent the height and width of the down-sampled feature map and d is the number of channels.

3.4 Transformer U-net

The attention-augmented model is constructed by adding a Transformer encoder to the bottleneck of the baseline, more specifically between bottleneck projection and the first up-sampling step of the decoder. The motivation behind placing it in the bottleneck is twofold. Firstly and mainly we want the decoder to have direct access to the long-range interactions of the features and use these as a base for up-sampling. Secondly we want to keep the time and memory complexity computationally feasible. This given that the complexity of the Transformer encoder scales with $\mathcal{O}(n^2)$ with regards to the input length n.

The network architecture is presented in Figure 3.6. Apart from the bottleneck, the baseline is kept intact and hence the network architecture and model configurations from Subsection 3.3.2 respectively Subsection 3.3.1 are applicable here as well. The Transformer encoder, as presented in Subsection 2.2.1, takes as input the feature map flattened in the spatial dimension and outputs a feature map in the original shape. We experimented with the number of layers, in the transformer encoder, and the number of heads, in the multi-head self-attention module, and settled with two configurations presented in Table 3.3. One small and one large to investigate how the magnitude of the transformer encoder effects the results.

Table 3.3: Configurations of the number of heads and layers of the Transformer encoder.

Transformer encoder	small	large
Heads	2	8
Layers	2	6

Table 3.4: Properties of datasets. $H \times W$ is the spatial resolution of the samples after cropping.

	Synthetic dataset	Lane dataset
Number of samples	$\sim 1\,\mathrm{M}$	$\sim 0.1\,\mathrm{M}$
$H \times W$	256×256	352×1280
Number of channels	3	3
Classes	Positive, Background	Lane marker, Background

3.5 Datasets

In this thesis we use two separate datasets to train and evaluate the models on. The first one is referred to as the *synthetic dataset* and the second one as the *lane dataset*. Their respective properties are shown in Table 3.4. In the sections below these datasets, as well as the motivation behind the synthetic dataset, are described in detail.

3.5.1 Synthetic dataset

The synthetic dataset is based upon the Pathfinder challenge introduced in [7]. Example samples of our modified version is shown in Figure 3.7 where you find the raw input in the first row and the corresponding ground truth segmentation mask below. This dataset only contains two classes, the background class and the positive class.

The segmentation problem that this dataset presents can be easiest described with a question: *Are the dots connected by a path?* If the answer is yes then the dots and the corresponding path should be treated as the positive class, if not they should be treated as the background class, along with the actual black background. As can be seen in Figure 3.7 there are samples with no positive class objects and only positive class objects as well. By looking at the sample to the far right you also find that some cases can be difficult to classify by human visual inspection. One might guess that there are two positive class object instead of one.

The high level motivation behind creating the synthetic dataset lies in the need for a short feedback cycle when developing a new model. Since attention mechanisms, and deep learning models in general, require a lot if computational resources we concluded that being able to decide the resolution and content of the data as well as the size of the dataset can be highly beneficial. It is also an extremely cheap way to obtain near unlimited amounts of annotated data.

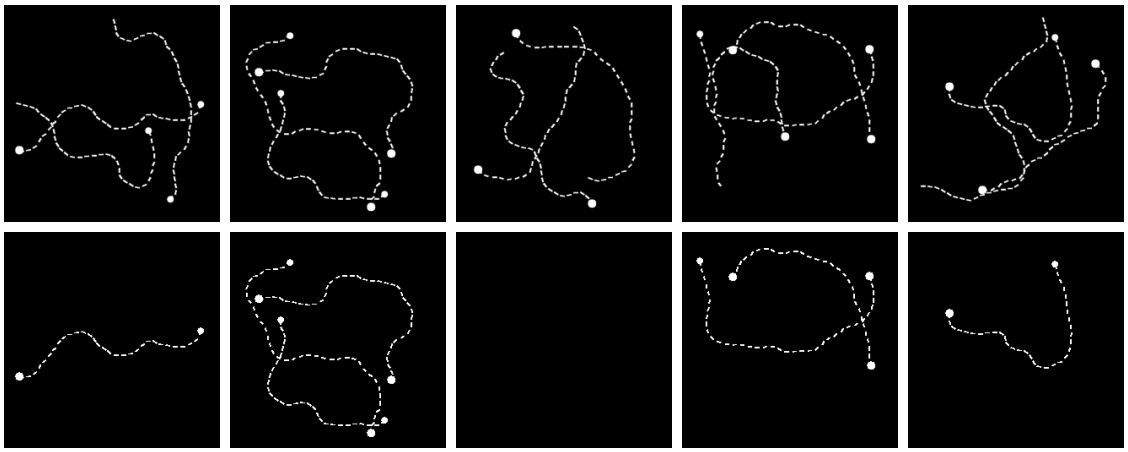

Figure 3.7: Example samples of the synthetic dataset. On the first row you find the raw input images and below you find the corresponding ground truth segmentation masks. If two dots are connected by a path they should be considered as objects of the positive class.

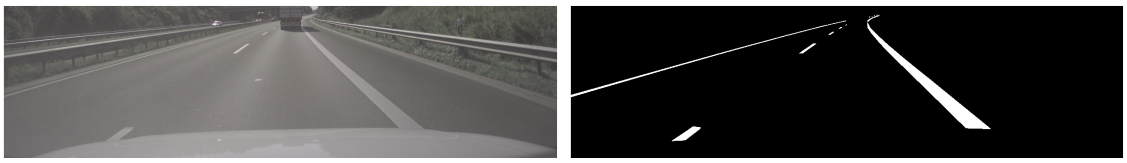

Figure 3.8: Example sample of the Lane dataset. To the left you find the input image and to the right the ground truth segmentation mask.

The motivation behind the content of the data lies in the need for a problem that requires non-local reasoning. Since the paths can be stretched across the entire image it can be difficult for a CNN, with limited receptive fields, to correctly declare a path as positive or not.

3.5.2 Lane dataset

The Lane dataset is provided by Zenseact and consists of road data in various traffic environments. The data is collected by a car mounted with a front camera. Similar to the synthetic dataset we only have two classes which in this case are lane marker and background. The lane marker class consists of both solid and dashed lane markers. The sample shown in Figure 3.8 is an example of an input image and its corresponding ground truth segmentation mask.

3.6 Experiments

The model architecture is evaluated over a range of encoder-decoder configurations. For the U-net baseline these are referred to as *tiny-tiny* if we use a tiny encoder and a tiny decoder. For the Transformer U-net the size of the transformer encoder is added to the end such as *tiny-tiny-small* if we use a small transformer encoder. The models are in turn evaluated separately on the two datasets presented in Section 3.5.

While the model architecture coincides between the two datasets there are some differences in how the models are trained and evaluated that are described below.

3.6.1 Training

The models are trained and evaluated on the input images and their corresponding ground truth segmentation masks. We use the same training set up for the U-net baseline as for the Transformer U-net. There are however some differences between the set up for the synthetic data and the lane data that will be highlighted.

We use batchsize 64 for the synthetic data and batchsize 8 for the lane data. The reason behind this lies in the resolution of the input image and memory limitations. The batchsize decides how many images are passed through the network in parallel during the training phase and larger images naturally consume more memory. We train the synthetic dataset for 25 epochs and the lane dataset for 40 epochs. These numbers were chosen empirically on the grounds that the learning curve had flattened and we did not experience problems with overfitting. For optimization we use Adam [21] with $\beta_1 = 0.9$ and $\beta_2 = 0.999$, where β_1 and β_2 denote the exponential decay rate for the first and second moment estimates. The corresponding learning rate is set to $5 \cdot 10^{-5}$. For the lane data we add a scheduled warmup of the learning rate that starts from $1 \cdot 10^{-5}$ and gradually increases to $5 \cdot 10^{-5}$ over the first three epochs.

To measure the error of the prediction two different loss functions are used, one for each dataset. For the synthetic data we use binary cross entropy loss as defined in (2.7) and for the lane data distance-weighted binary cross entropy loss as defined in (2.8), both described in Subsection 2.1.5. The distance weighted loss is applied to enhance the weight of the loss for lane markers higher up in the image since these lane markers occupy less pixels.

3.6.2 Evaluation

To evaluate the quantitative performance of the models we use the metric Intersection over Union (IoU) described in Subsection 2.3.1. More specifically we use class specific IoU that only takes into account the positive class and the lane marker class for the respective datasets. The IoU-scores presented in Chapter 4 show the maximum IoU obtained during validation of all epochs in one run. More specifically the mean value over all batches in the epoch. We also display the IoU evaluated on the training data corresponding with the maximum validation IoU.

To provide an understanding of the magnitude of the models we measure the number of trainable parameters and floating points operations (FLOPs) of each configuration. In the results these values are plotted against the maximum IoU-score. The number of trainable parameters is defined as the number of elements in the network that are affected by backpropagation. For a 2D convolutional layer this is calculated as

$$\text{Trainable parameters} = (K \cdot C_{\text{in}} + 1) \cdot C_{\text{out}} \tag{3.1}$$

where K is the two-dimensional size of the kernel, C_{in} the number of input channels

and C_{out} the number of output channels or kernels. The $+1$ takes into account the bias term for each kernel. For a 3×3 convolutional layer with 32 kernels, following an input layer with 1 channel this becomes

$$\text{Trainable parameters} = (3 \cdot 3 \cdot 1 + 1) \cdot 32 = 320. \tag{3.2}$$

The number of FLOPs is defined by, as the name suggest, the number of floating point operations, in other words multiplication and addition. Unlike the number of trainable parameters of a convolutional layer the number of FLOPs also takes into account the size of the output feature map. Again, for a 2D convolutional layer this is calculated to

$$\text{FLOPs} = 2 \cdot C_{\text{in}} \cdot K \cdot C_{\text{out}} \cdot H \cdot W \tag{3.3}$$

where H and W are the height and width of the output feature map, respectively. Following the same example as in (3.2) with $H = W = 64$ we get

$$\text{FLOPs} = 2 \cdot 1 \cdot 3 \cdot 3 \cdot 32 \cdot 64 \cdot 64 = 2\ 359\ 296. \tag{3.4}$$

For experiments performed on the lane dataset we also provide the inference time. This is the time it takes to pass one image through the network and obtain a segmented image. To eliminate overhead from data loading, randomized noise is used as inputs instead of real data. The average inference time over 1000 forward passes is measured. This is then repeated 5 times. The lowest average forward pass time is then selected from these runs to minimize the influence from external factors.

4

Results

In this chapter the experiments described in Chapter 3 are performed on both datasets to evaluate the performance of our models. Metrics such as Intersection over Union for both the training data and the validation data, the number of trainable parameters and the number of floating point operations per forward pass are collected for each model configuration on each dataset. The results are presented both in tables and in figures. In addition, qualitative results are presented for both datasets and inference times are evaluated for the lane dataset. Finally, an ablation study is made to analyze the impact of positional encodings, efficient attention mechanisms and different model architectures.

4.1 Synthetic dataset

In this section, both the baseline models and the attention-augmented models are evaluated on the synthetic dataset. As seen in Table 4.1, the attention-augmented model reaches an IoU for the positive class of **0.953** while the baseline model reaches **0.945**. Adding a Transformer to the bottleneck of the U-net model seems to increase performance by at least 0.8 percentage points, irrespective of the model configuration. In some cases, for smaller models in particular, the increase in performance is much larger.

The baseline model performs well for larger configurations. However, for smaller configurations, the baseline model falls short. For the smallest configuration, tiny-tiny, the baseline model reaches an IoU of **0.658**. Meanwhile, the attention-augmented model tiny-tiny-small, with a comparable number of trainable parameters and floating point operations reaches **0.931**, resulting in an increase by 27.3 percentage points. This is illustrated in Figure 4.1 and Figure 4.2, where the latter shows a more limited range of IoU values compared to the first.

Table 4.1: Performance for different model configurations on the synthetic dataset. Model configurations are denoted as encoder size-decoder size. IoU are presented for both the training dataset and the validation dataset. The comparison also includes the number of trainable parameters as well as the number of floating point operations (FLOPs). Note how the attention-augmented model outperforms the baseline model for all model configurations, with a relativly small increase in the number of trainable parameters and FLOPs.

Baseline U-net

Model	IoU - train.	IoU - valid.	#param.	FLOPs
tiny-tiny	0.661	0.658	1.64M	1.61G
small-small	0.906	0.889	3.11M	1.88G
medium-medium	0.952	0.931	3.48M	2.16G
large-large	0.958	0.942	3.57M	2.39G
xlarge-xlarge	0.961	**0.945**	3.60M	2.85G

Transformer U-net - small

Model	IoU - train.	IoU - valid.	#param.	FLOPs
tiny-tiny	0.937	0.931	1.91M	1.65G
small-small	0.955	0.939	3.38M	1.92G
medium-medium	0.956	0.943	3.74M	2.19G
large-large	0.963	0.947	3.83M	2.43G
xlarge-xlarge	0.965	**0.953**	3.86M	2.89G

Transformer U-net - large

Model	IoU - train.	IoU - valid.	#param.	FLOPs
tiny-tiny	0.950	0.944	2.44M	1.73G
small-small	0.956	0.945	3.91M	2.00G
medium-medium	0.961	0.948	4.27M	2.27G
large-large	0.963	**0.949**	4.36M	2.51G
xlarge-xlarge	0.963	0.948	4.39M	2.97G

4.1.1 Qualitative Results

Included in Figure 4.3 are some qualitative results on the synthetic dataset. The left column represents the input images, the middle column represents the target images and the right column represents the predictions made by the models. The top three rows of predictions have been produced by the smallest baseline model, tiny-tiny, on three different input images. The bottom three rows have been produced by the smallest attention-augmented model, tiny-tiny-small, on the same set of three input images.

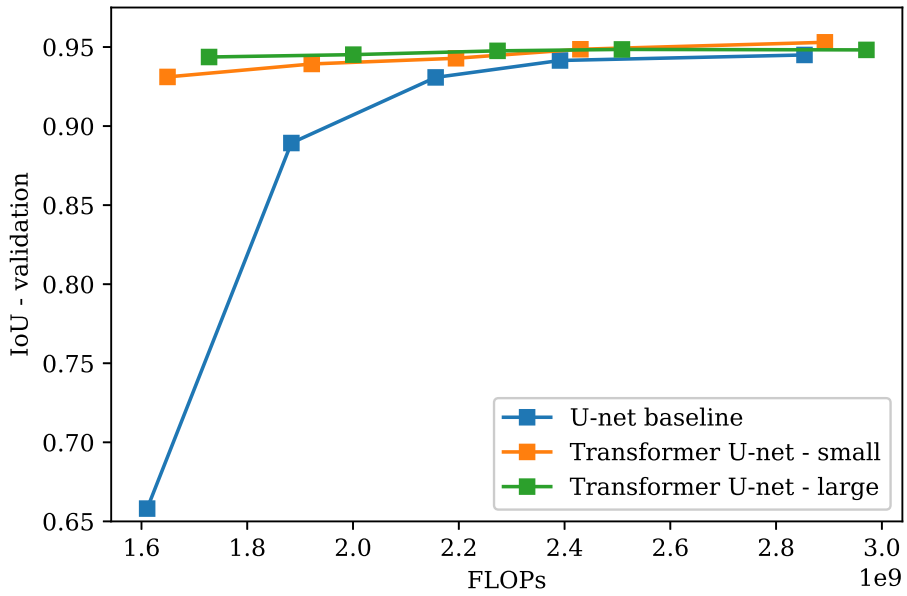

Figure 4.1: Performance for different model configurations on the synthetic dataset where IoU is compared to the FLOPs. Each dot represents a different model configuration. Each color represents model configurations with the same Transformer size – or no Transformer at all for the baseline models.

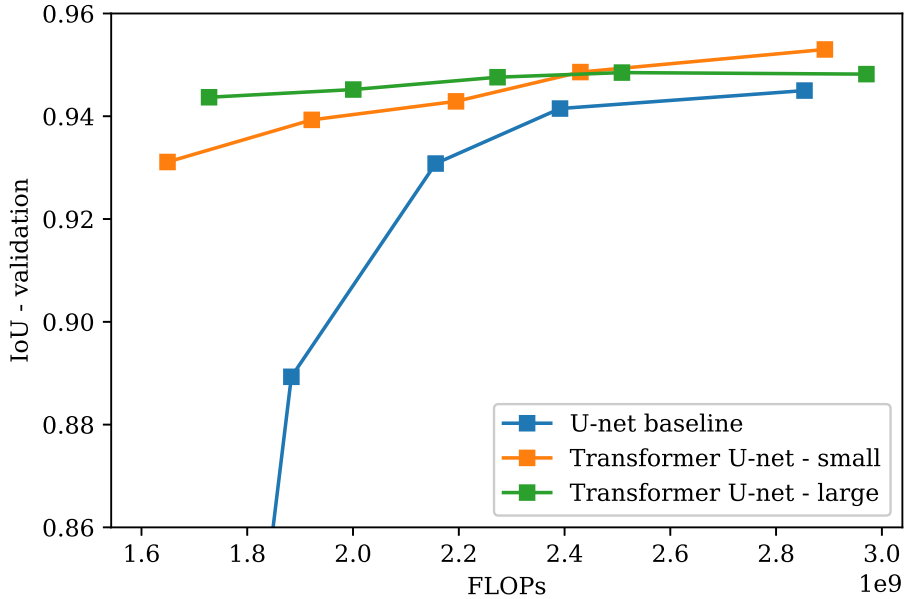

Figure 4.2: Same comparison as in Figure 4.1 but with a more limited range of IoU values.

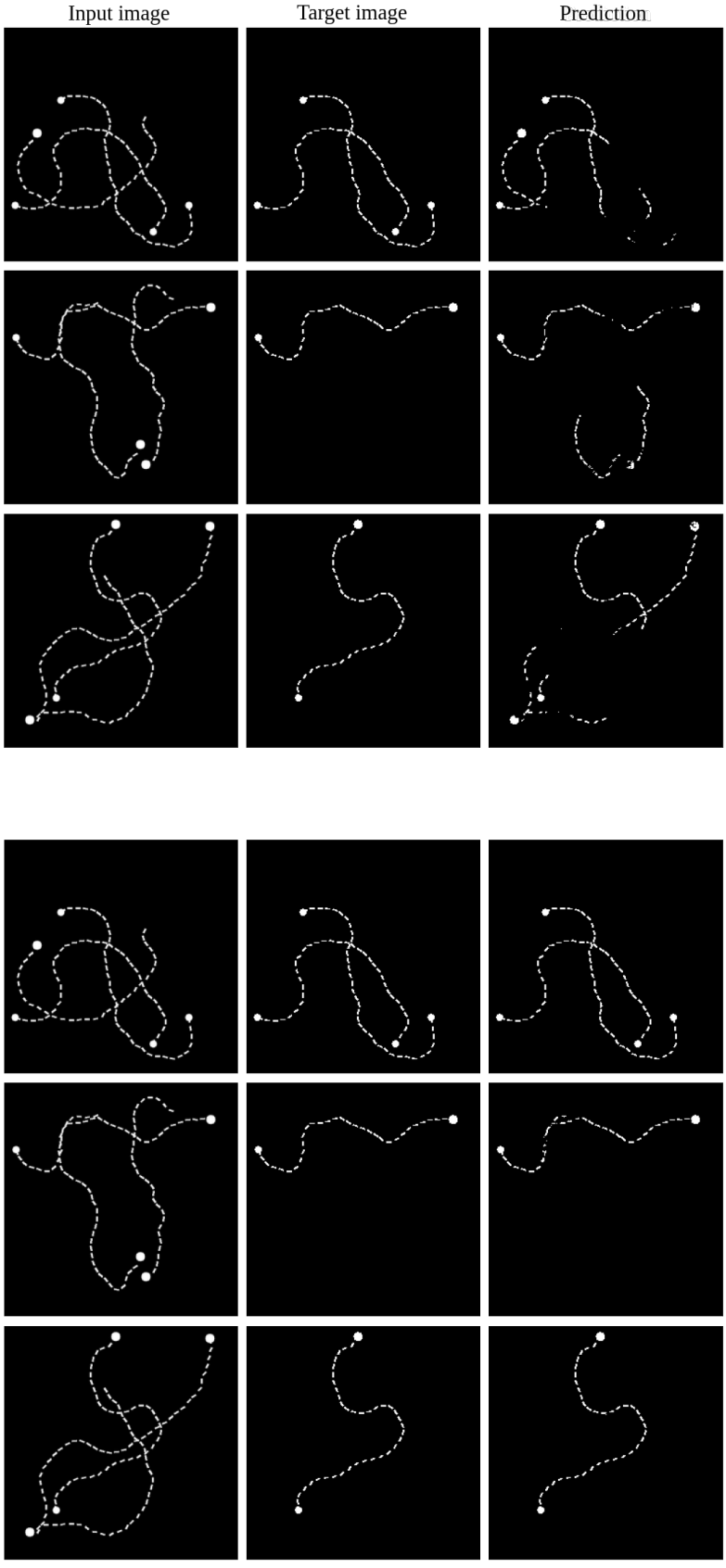

Figure 4.3: Baseline U-net (top 3 rows) and Transformer U-net (bottom 3 rows) predictions for the tiny-tiny-small model configuration on the synthetic dataset.

4.2 Lane dataset

In this section, both the baseline models and the attention-augmented models are evaluated on the lane dataset. The baseline models reach an IoU of **0.826** while the attention-augmented models reach an IoU of **0.835**. Again, we find that adding a Transformer to the bottleneck of the baseline model improves performance for all model configurations by at least 0.6 percentage points. However, for this dataset, we do not see such a dramatic difference for smaller model sizes.

Overall, we find that the performance increase associated with adding a Transformer mechanism to the baseline model is in the neighbourhood of 1-1.5 percentage points for all model configurations, see Table 4.2. The trade-off between performance and FLOPs is illustrated in Figure 4.4.

Table 4.2: Transformer U-net performance for different model configurations on the lane dataset. Model configurations are again denoted as encoder size-decoder size. IoU is presented for both the training dataset and the validation dataset. The comparison also includes the number of trainable parameters as well as the FLOPs.

Baseline U-net

Model	IoU - train.	IoU - valid.	#param.	FLOPs
tiny-tiny	0.852	0.812	1.64M	11.1G
small-small	0.877	0.817	3.11M	12.9G
medium-medium	0.881	0.822	3.48M	14.8G
large-large	0.881	0.826	3.57M	16.4G
xlarge-xlarge	0.881	**0.826**	3.60M	19.6G

Transformer U-net - small

Model	IoU - train.	IoU - valid.	#param.	FLOPs
tiny-tiny	0.863	0.824	1.91M	11.5G
small-small	0.875	0.828	3.38M	13.4G
medium-medium	0.876	0.830	3.74M	15.2G
large-large	0.877	0.832	3.83M	16.7G
xlarge-xlarge	0.877	**0.834**	3.86M	20.0G

Transformer U-net - large

Model	IoU - train.	IoU - valid.	#param.	FLOPs
tiny-tiny	0.862	0.827	2.44M	12.4G
small-small	0.872	0.831	3.91M	14.3G
medium-medium	0.874	0.833	4.27M	16.2G
large-large	0.875	0.833	4.36M	17.8G
xlarge-xlarge	0.874	**0.835**	4.39M	21.0G

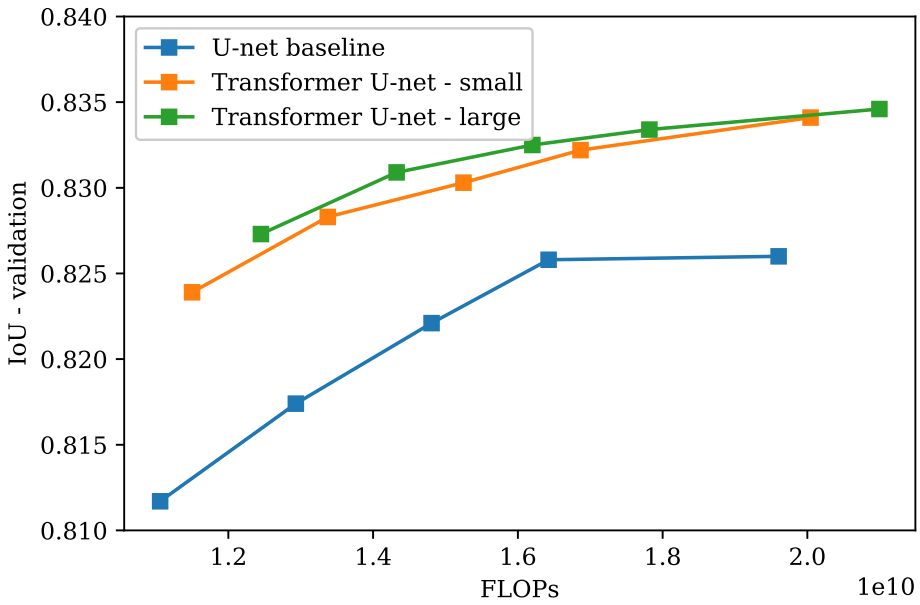

Figure 4.4: Performance for different model configurations on the lane dataset where IoU is compared to the FLOPs. Each dot represents a different model configuration. Each color represents model configurations with the same Transformer size – or no Transformer at all for the baseline models.

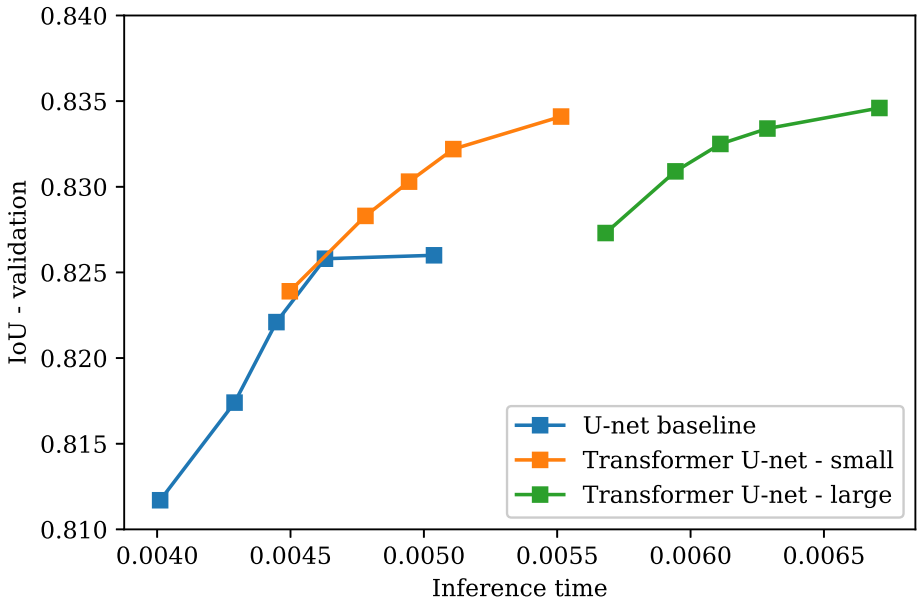

Figure 4.5: Same model configurations as in Figure 4.1 but comparing IoU to inference time instead.

Table 4.3: Transformer U-net performance for different model configurations on the lane dataset.

Baseline U-net

Model	IoU - train.	IoU - valid.	Inference time
tiny-tiny	0.852	0.812	4.01 ms
small-small	0.877	0.817	4.29 ms
medium-medium	0.881	0.822	4.45 ms
large-large	0.881	0.826	4.63 ms
xlarge-xlarge	0.881	**0.826**	5.04 ms

Transformer U-net - small

Model	IoU - train.	IoU - valid.	Inference time
tiny-tiny	0.863	0.824	4.50 ms
small-small	0.875	0.828	4.78 ms
medium-medium	0.876	0.830	4.94 ms
large-large	0.877	0.832	5.11 ms
xlarge-xlarge	0.877	**0.834**	5.51 ms

Transformer U-net - large

Model	IoU - train.	IoU - valid.	Inference time
tiny-tiny	0.862	0.827	5.67 ms
small-small	0.872	0.831	5.94 ms
medium-medium	0.874	0.833	6.11 ms
large-large	0.875	0.833	6.29 ms
xlarge-xlarge	0.874	**0.835**	6.71 ms

4.2.1 Inference time

The attention-augmented models improved performance across the board for all model configurations on both datasets with a relatively small difference in the number of trainable parameters and the number of floating point operations. With that said, the number of trainable parameters and the number of floating point operations do not paint the whole picture. For real world applications of lane detection, the inference time is another important metric, since real-time requirements are common. In Figure 4.5, a comparison of the IoU over the inference time of the models is visualized.

The addition of an attention mechanism in the form of a Transformer encoder does seem to affect the inference time substantially, see Table 4.3.

4.2.2 Qualitative Results

Included in Figure 4.6, Figure 4.7 and Figure 4.8 are some qualitative results on the lane dataset. In all these figures, the top picture represents the input image, while the second from the top represents the target image. The second from the bottom shows a prediction made by the smallest baseline model and the bottom picture shows the corresponding prediction made by the smallest attention-augmented model. In some of the figures, interesting areas where the two models differ are highlighted with red.

Figure 4.6: Top picture – input image. Second from the top - target image. Second from the bottom – prediction made by the smallest baseline model. Bottom picture – prediction made by the smallest attention-augmented model. Note how the rightmost lane is missed by the baseline model but picked up by the attention-augmented model.

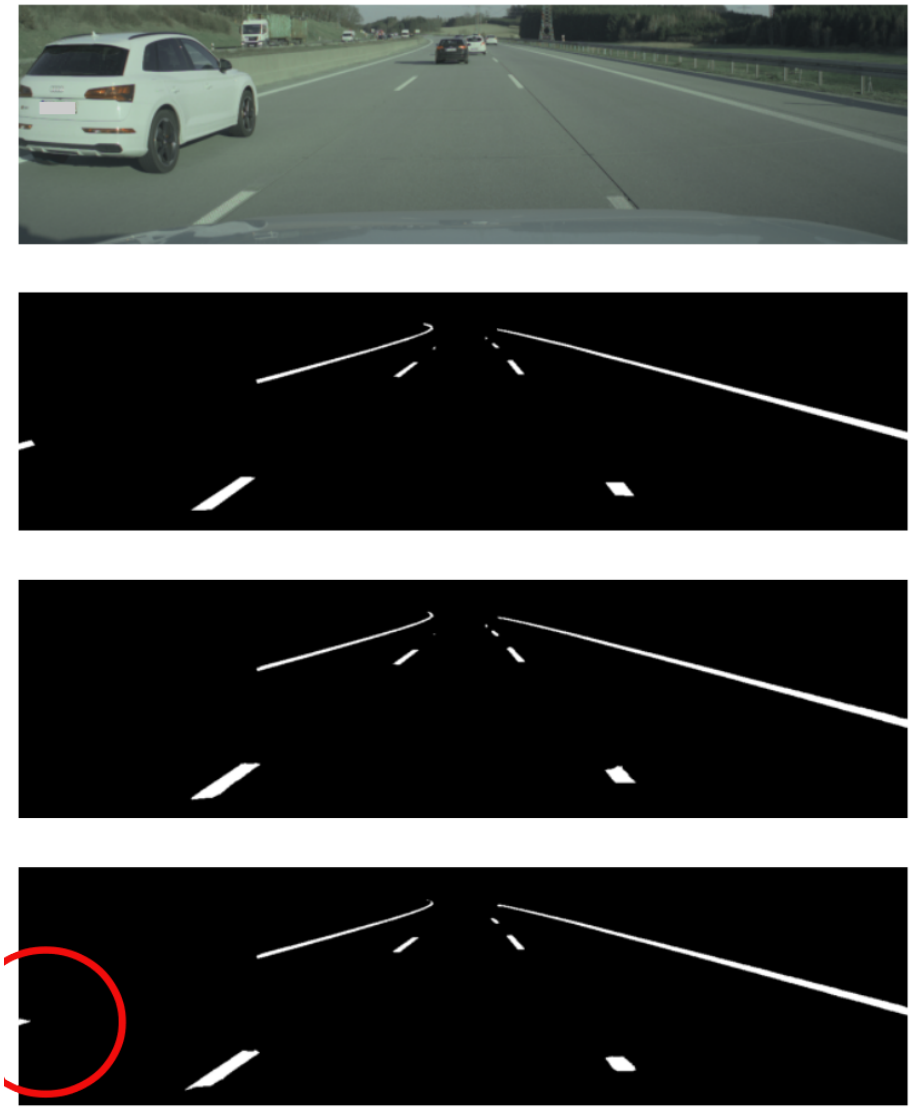

Figure 4.7: Top picture – input image. Second from the top - target image. Second from the bottom – prediction made by the smallest baseline model. Bottom picture – prediction made by the smallest attention-augmented model. Note how the leftmost lane is missed by the baseline model but picked up by the attention-augmented model, despite the lane being occluded and the visible area of the lane being very small.

Figure 4.8: Top picture – input image. Second from the top - target image. Second from the bottom – prediction made by the smallest baseline model. Bottom picture – prediction made by the smallest attention-augmented model. Note that in this case the attention-augmented model misses a lane that is picked up by the baseline.

4.2.3 Attention Maps

With the attention-augmented models, it is also possible to visualize what the Transformer encoder attends to in so-called attention maps. These attention maps are created by averaging the attention output from a certain position in the feature map. Included in Figure 4.9 and Figure 4.10 are attention maps from two input images.

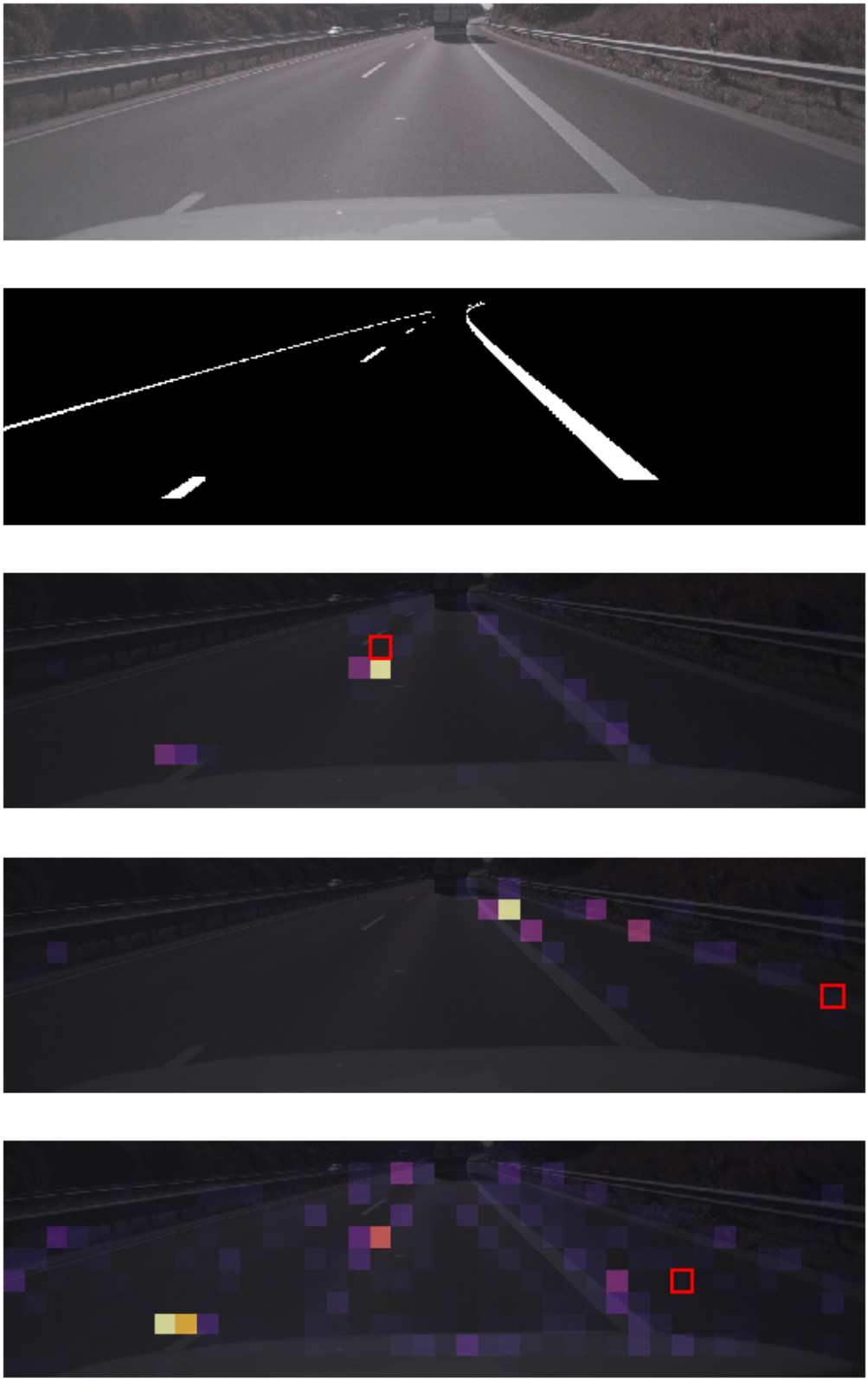

Figure 4.9: Top picture – input image. Second from the top – target image. Bottom three pictures are attention map visualizations. The red square in each attention map represents the position in the input image from where the attention map is based on.

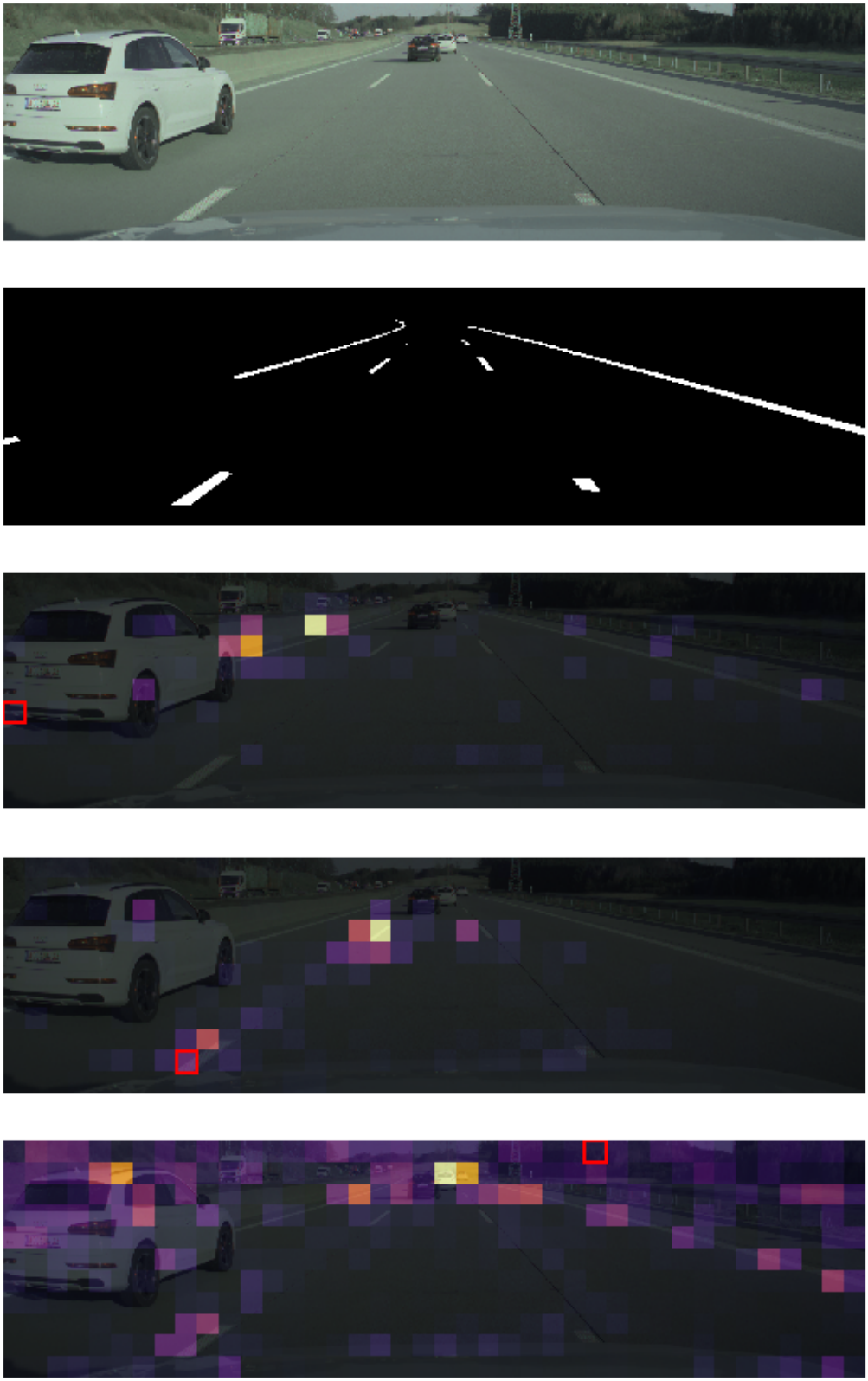

Figure 4.10: Top picture – input image. Second from the top – target image. Bottom three pictures are attention map visualizations. The red square in each attention map represents the position in the input image from where the attention map is based on.

4.3 Ablations

In this section we compare different versions or modifications of the proposed attention-augmented model to determine the impact of said modifications, including positional encodings, efficient attention mechanisms and variations of the model architecture.

4.3.1 Positional encodings

In the original article introducing the Transformer architecture, positional encodings were used, as mentioned in Section 2.2. We investigated the impact of explicit positional encodings for our model, in the form of fixed sinusoidal positional encodings. For the impact on the small Transformer encoder see Figure 4.11 and for the impact on both Transformer encoder sizes see Table 4.4. The results show that positional encodings do not serve an important role in the proposed model and the quantitative performance is not affected significantly. For this reason, we chose to omit positional encodings for the rest of our experiments and in our final model.

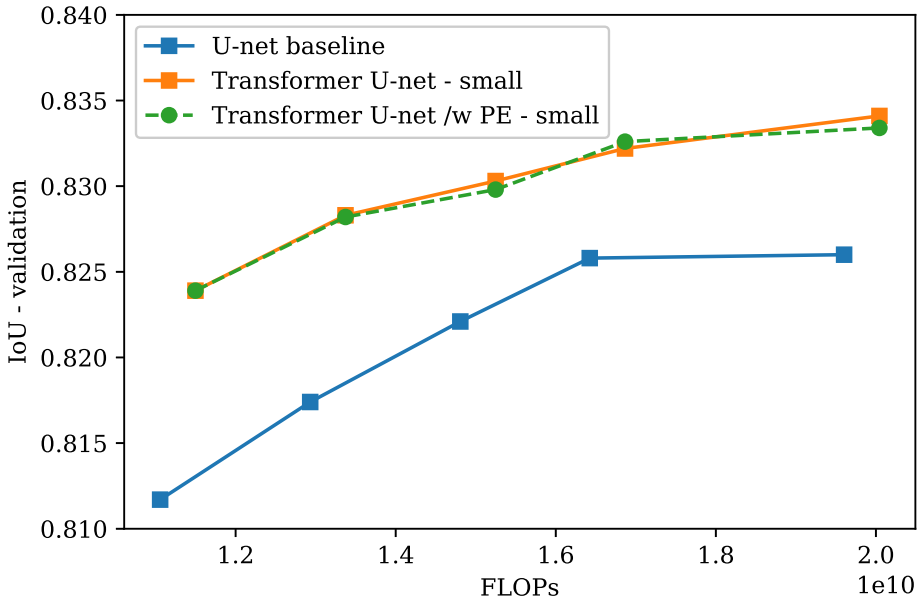

Figure 4.11: Performance for different model configurations on the lane dataset where IoU is compared to the FLOPs. Each dot represents a different model configuration. Green and orange here represent two sets of model configuration of an attention-augmented model with and without explicit positional encoding. Notice how the effect of explicit positional encodings is hardly noticeable.

Table 4.4: Transformer U-net performance for different model configurations on the lane dataset without and with positional encoding.

Transformer U-net - small, IoU

Model	train.	valid.	train. /w PE	valid. /w PE
tiny-tiny	0.863	0.824	0.864	0.824
small-small	0.875	0.828	0.876	0.828
medium-medium	0.876	0.830	0.878	0.830
large-large	0.877	0.832	0.878	0.833
xlarge-xlarge	0.877	**0.834**	0.874	0.833

Transformer U-net - large, IoU

Model	train.	valid.	train. /w PE	valid. /w PE
tiny-tiny	0.862	0.827	0.864	0.830
small-small	0.872	0.831	0.873	0.830
medium-medium	0.874	0.833	0.873	0.835
large-large	0.875	0.833	0.875	0.833
xlarge-xlarge	0.874	**0.835**	0.877	0.834

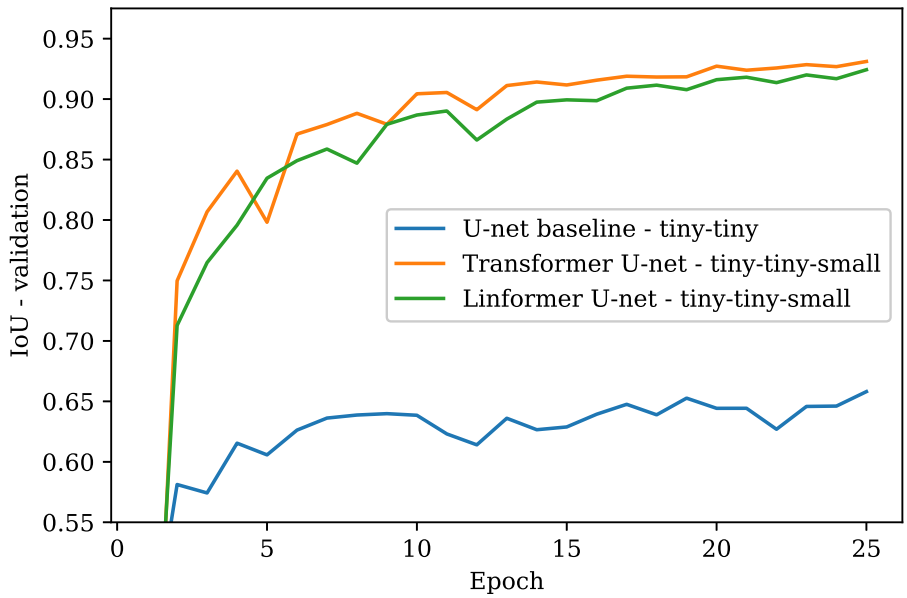

Figure 4.12: Performance of the Transformer U-net and Linformer U-net on the synthetic dataset. The encoder and decoder are of size tiny and the Transformer/Linformer encoder is of size small. Here, the Linformer projects the length of the input sequence from 64 to 32. Note that the x-axis shows number of epochs.

4.3.2 Linear attention

As noted in Section 2.2, the attention-augmented models can suffer from time and space complexity. Since we noted that our inference time was increased by adding a Transformer encoder to the bottleneck of our baseline model, we wanted to experiment with efficient Transformers. We first implemented a Linformer where we set the projected length of the input sequence to $k = 32$, which is half the original sequence length. To ensure that this reduction in sequence length did not decrease the performance of our original model too much, we first evaluated the Linformer on the synthetic dataset.

While the Linformer model initially performed almost 5 percentage points worse than the original Transformer for the first five epochs, it narrowed this gap over the course of the full training, as seen in Figure 4.12. Towards the end, the difference between the Transformer and the Linformer based models was relatively low. Therefore, we decided to evaluate the Linformer on the lane dataset as well, where inference time really matters.

For the lane dataset, the same pattern emerged. Here, the sequence length, k, was projected down to 256 instead of 440 as in the original model. After letting the Linformer train for the full training schedule, it reached an IoU of 0.823 for the smallest model and 0.834 for the biggest model. When comparing this to the original Transformer, the difference in performance is only 0.1 percentage points for both model configurations. However, the inference time of the models was not affected greatly. While the inference time for the larger model was decreased with about 0.1ms, the inference time for the smaller model was even increased slightly, by approximately 0.02 ms. See Table 4.5 for the full details and Figure 4.13 for a visualization.

Table 4.5: Performance of the Transformer U-net and the Linformer U-net for the smallest and largest model configurations. Model configurations are denoted as encoder size-decoder size-Transformer/Linformer encoder size.

Transformer U-net

Model	IoU - valid.	Inference time
tiny-tiny-small	0.824	4.50 ms
xlarge-xlarge-large	0.835	6.71 ms

Linformer U-net

Model	IoU - valid.	Inference time
tiny-tiny-small	0.823	4.52 ms
xlarge-xlarge-large	0.834	6.58 ms

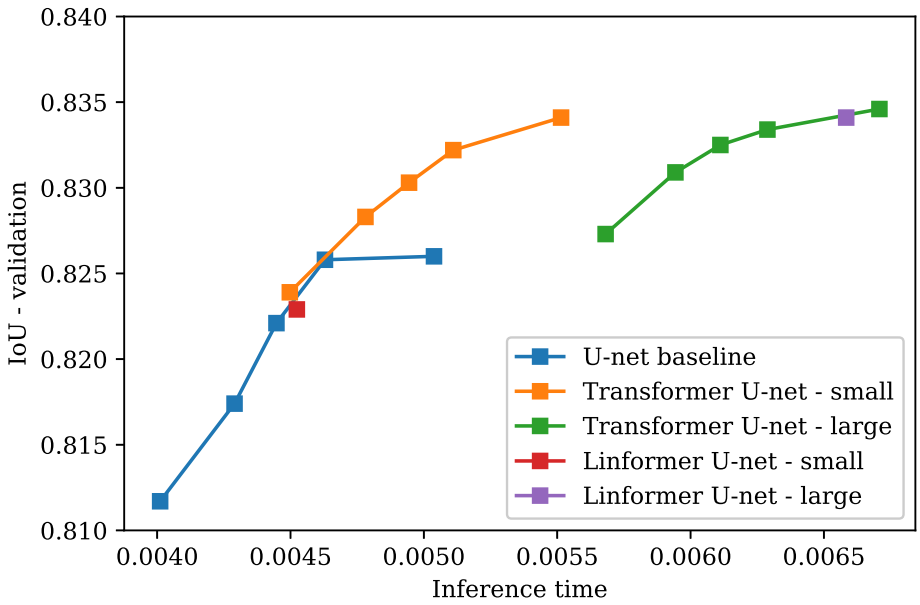

Figure 4.13: Performance of the Transformer U-net and Linformer U-net on the synthetic dataset. Each dot represents a different model configuration. Here, the Linformer projects the length of the input sequence from 440 down to 256. Notice how the Linformer based models do not offer a lot of improvement when it comes to the trade-off between performance and inference time.

4.3.3 Transformer Decoder

We also experimented with applying the Transformer decoder in our model as an alternative to the concatenation normally done in U-net decoders. By letting an attention mechanism attend to the up-sampled feature map from the bottleneck, it could in theory learn to extract more interesting features from the skip connection.

Note that when using a Transformer decoder in the first up-sampling stage in the U-net decoder, the spatial resolution is doubled. This means that the number of positions in the feature map or the sequence length is quadrupled. Therefore, the memory usage prevented us from using a standard Transformer. Luckily, by leveraging the Linformer architecture covered in the previous subsection with $k = 256$, the memory usage was reduced to the point where it was feasible to train again.

In practice, we found that using the Transformer Decoder resulted in very minor improvements over a portion of the standard attention-augmented models, see Figure 4.14 and Table 4.6. In Table 4.6 it is seen that for the smallest models the validation IoU is slightly higher and for the largest model it is slightly lower. With that said, the concept seems promising and variations of this model show great potential.

Table 4.6: Performance of the Transformer U-net and the Transformer decoder U-net for the smallest and largest model configurations. Model configurations are denoted as encoder size-decoder size-Transformer encoder size.

Transformer U-net

Model	IoU - train.	IoU - valid.
tiny-tiny-small	0.863	0.824
xlarge-xlarge-large	0.874	0.835

Transformer decoder U-net

Model	IoU - train.	IoU - valid.
tiny-tiny-small	0.878	0.826
xlarge-xlarge-large	0.884	0.834

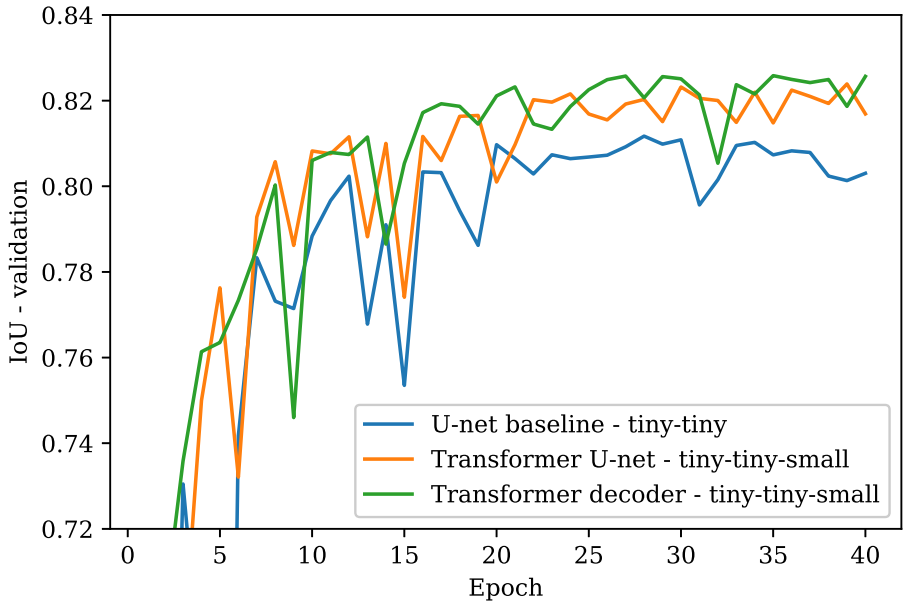

Figure 4.14: Performance of the Transformer U-net and Transformer decoder U-net on the lane dataset. The encoder and decoder are of size tiny and the Transformer encoder is of size small. Here, the Linformer in the Transformer decoder projects the length of the input sequence from 1760 to 256. Note that the x-axis shows the number of epochs.

5

Discussion

In this chapter, we discuss the results from Chapter 4 on both the datasets. We also discuss the methodology used to arrive at these results as well as some of the ablations made. In addition, we discuss some areas where we think there are room for improvement and possible future work.

5.1 Performance on the synthetic dataset

As seen in Section 4.1, the attention-augmented model improves IoU performance across the board. However, for smaller models, the increase in performance is much greater. This indicates that the synthetic dataset, with the task of segmenting only connected paths, is challenging for a smaller convolutional neural network with a limited receptive field. To properly segment a connected path, one needs to take both endings of the path into account. Therefore, long-range dependencies seem to be helpful, or even necessary in some cases, and provide a possible explanation to why the attention-augmented methods perform so well.

In the qualitative results for the synthetic dataset, see Subsection 4.1.1, it is seen how the attention-augmented model segments most of the connected paths successfully. However, the baseline model fails spectacularly, even on some paths that appear to be fairly simple. It should be noted that the input image can be much more complex and in some cases it is difficult even for a human annotator to correctly segment the paths. Failure to segment these simpler examples indicates, as previously mentioned, that the task requires long-range dependencies and that the limited receptive field of the baseline model is an obstacle.

Based on the performance improvements, both in terms of IoU and from inspection of the qualitative results, the addition of an attention mechanism to the baseline model seems to enhance the models capacity to capture long-range dependencies. For a task that is engineered around requiring some form of global reasoning, our proposed model outperforms the baseline model for all tested model configurations. In addition, the attention-augmented model allows for a much smaller model while still demonstrating effectiveness in solving the segmentation task, which is useful when operating under a computational budget.

5.2 Performance on the lane dataset

For the lane dataset, the attention-augmented model once again improves performance across the board for all tested model configurations, see Section 4.2. However, compared to the synthetic dataset, we do not see such a dramatic difference for smaller model sizes.

This indicates that the lane detection problem requires long-range dependencies, although they are not as important for this dataset compared to the synthetic dataset. After inspecting a number of frames from the dataset, this is true for many lanes as well. In most cases, for semantic segmentation of lane markers, one can segment one end of a lane without taking the other end of the lane into account. Segmenting just a part of a longer lane and an occluded lane is still possible, while for the synthetic dataset you want to segment either the whole path or nothing of the path. While capturing long-range dependencies can help improve the model, it is not strictly necessary. For other, more advanced, lane detection tasks such as lane instance segmentation, long-range dependencies might have a larger impact. This was however not investigated further due to time restrictions.

From the qualitative results, we see that in some cases, the attention-augmented model performs better on input images where the lanes are occluded, see Figure 4.6 and Figure 4.7. However, this is not the case for all occluded lanes. We can also determine that the attention-augmented model is not strictly better for all lane detections. In some cases, the baseline model manages to capture a lane while the attention-augmented model misses it completely.

For lane detection in the context of autonomous vehicles, real time performance is a must. Therefore, the inference time is another important metric to factor in. As noted in Subsection 4.2.1, the addition of a Transformer encoder to the baseline model resulted in a measurable increase in inference time. While the small configuration of the Transformer still seems competitive, the large configuration can be a tougher sell. However, it is possible that the transformer encoder implementation is sub-optimal.

First of all, although our transformer uses standard operations in a deep learning framework, the operations may not be very optimized in the CUDA software layer which interfaces with the graphics processing unit. Meanwhile, the CUDA implementation of operations used by convolutional neural networks have been fine-tuned over many years.

Additionally, the order of dimensions in the input tensor for the transformer can be further optimized. For example, in convolutional neural networks, it is common to use the “channels first” format for performance reasons, where the channels come before the height and the width of the input image. In our transformer implementation, we use a channels or tokens last format. It is possible that another format might yield a better performance.

5.3 Analysis of ablations

As suggested by the results in Section 4.3, positional encodings do not seem to be needed for attention-augmented models to perform well. This can seem a bit surprising, since the the Transformer architecture is by default treating the input as a set of tokens, not a sequence of tokens. The Transformer itself does not have any sense for the position of each token, but there is still a need to incorporate the order of the tokens for NLP tasks as language translation. Positional encodings are added to the input sequence to give each token some information about its position in the sequence.

Positional encodings can be either fixed, for example based on a sinusoidal pattern, or learned. We hypothesize that the convolutional backbone acts as a form of learned positional encoding. Therefore, the need for explicit positional encodings is diminished. In recent related works [29], a single convolutional layer is successfully used as a learned positional embedding which seems to support our theory.

We also encountered another surprising result in the inference speed of the Linformer based models. While the performance of these models was excellent, even when reducing the length of the input sequence to half, the inference time was not affected that much. We even tried to reduce the sequence length by a factor of ten and the speedup was still underwhelming. Our conclusion from these experiments was that the implementations of our attention mechanisms, both in the Transformer and the Linformer, were not limited by the sequence length itself but by some other unknown factor.

On the other hand, the memory usage was reduced when leveraging the Linformer as it was the only way to implement the Transformer Decoder without running out of GPU memory. Unfortunately we did not have the time to accurately measure the reduction in memory usage.

5.4 Analysis of methodology

Our results show the impact of augmenting a convolutional neural network with attention mechanisms. We see that adding a Transformer encoder to the bottleneck of a CNN based model improves performance across the board for all tested configurations. For problems that require global reasoning, such as the synthetic dataset, we show that the attention mechanism is even more effective for smaller model configurations.

With that said, although the total number of runs for which our model architecture is tested is high, since we test many model configurations and a set of ablations, we only measure our metrics for each individual configuration from one single run. This makes it harder to eliminate the effect of noise in our results, and it would be beneficial to collect the results from several runs for each configuration. Then you have the possibility to report the average metric over these runs as well as the

standard deviation. The reason why we chose to limit ourselves to one run per configuration was due to time and resource constraints. Each run took several days to train. With over 30 different model configurations to test, not including any ablations, there was simply not enough time nor resources to collect more data for each configuration.

Another area where we were limited was the hyperparameter optimization. While we focused on exploring many different configurations of the encoder and decoder backbone, other hyperparameters of our models did not get the same treatment. We used the default values from the original articles that introduced the components for hyperparameters like the transformer encoder output dropout rate and self-attention dropout rate. We did however perform a more extensive parameter search for the learning rate, where we experimented with both higher and lower values before we arrived at the value used in our thesis.

5.5 Future work

For future work, we see a number of exciting areas where we think our work can be extended. First of all, we are interested in applying the attention-augmented models to other domains of computer vision. We ran some preliminary experiments on an object detection network and found that performance for larger objects was improved.

We also think that the Transformer decoder shows promise and we are interested in pushing this concept further. Not only is it possible to try to use the decoder side of the Transformer to replace even more concatenations in the U-net decoder, there is also the option to replace convolutions entirely for a convolution free decoder. Combined with related works where convolution free encoders or backbones have been proposed, a computer vision model based only on Transformers could be created, without resorting to diving the image into patches.

Future work further includes addressing some of the limitations of our work. A more comprehensive study or profiling of the inference time is of interest, to see where the attention-augmented model falls short and to better utilize the improved performance of the Transformer U-net model. This can be done in tandem with a deep dive into different efficient attention mechanisms to determine which of the different techniques is best suited for the field of computer vision. Most studies and surveys of efficient attention mechanisms are done with natural language processing in mind. Other works have suggested that for vision tasks, one might not need some of the more limiting, in terms of time and space complexity, components of the traditional self-attention mechanisms such as the softmax. Other alternatives might offer dramatic speedup which in turn make attention-augmented models even more attractive.

6

Conclusion

This thesis set out to investigate the effects of attention mechanisms on semantic segmentation by introducing an attention-augmented model and an attention-free baseline. Specifically the aim was to explore how the task of lane detection can benefit from long-range dependencies. The models were tested on two separate datasets, a synthetic dataset and a lane dataset. The first, engineered to portray a situation in need for long-range dependencies. The second, used to test how the models adapt to the task of lane detection.

The results of the experiments clearly showed that the attention-augmented model outperformed the baseline across the board. The positive effect of the Transformer encoder was most dramatic for smaller model sizes tested on the synthetic dataset. For the lane dataset we also saw a consistently positive effect, however, not as dramatic for the smaller sizes. These findings indicate that the synthetic dataset greatly benefits from the global perception provided by the Transformer encoder, at least for the smaller configurations. Taken together with the corresponding results of the lane dataset this suggests that the problem of lane detection might not be dependent of a global perception to the same extent as the synthetic problem. However, qualitative results indicate that certain common failure cases of lane detection can be salvaged with help of the attention mechanism.

With that said, lane detection is an important task where real-time performance is a must. Although inference speed was measured in the thesis, optimization of it was not included in the scope. A natural progression of this work is therefore to investigate how the model can be improved in terms of computational efficiency. In addition, the ablation study invites future work to explore efficient attention mechanisms further such as the Linformer which in combination with the Transformer decoder shows promise to push the model even further towards an attention-dominated direction.

Bibliography

[1] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, u. Kaiser, and I. Polosukhin, "Attention is all you need," *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17, Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6000–6010.

[3] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah, "Transformers in vision: A survey," *CoRR*, vol. abs/2101.01169, 2021. arXiv: 2101.01169.

[4] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part I*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds., ser. Lecture Notes in Computer Science, vol. 12346, Springer, 2020, pp. 213–229.

[5] H. Wang, Y. Zhu, B. Green, H. Adam, A. L. Yuille, and L.-C. Chen, "Axial-deeplab: Stand-alone axial-attention for panoptic segmentation," *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part IV*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds., ser. Lecture Notes in Computer Science, vol. 12349, Springer, 2020, pp. 108–126.

[6] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," *CoRR*, vol. abs/2010.11929, 2020. arXiv: 2010.11929.

[7] D. Linsley, J. Kim, V. Veerabadran, and T. Serre, "Learning long-range spatial dependencies with horizontal gated-recurrent units," *CoRR*, vol. abs/1805.08315, 2018. arXiv: 1805.08315.

[8] I. Bello, B. Zoph, Q. Le, A. Vaswani, and J. Shlens, "Attention augmented convolutional networks," *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 3285–3294.

[9] J. McCarthy, M. L. Minsky, N. Rochester, and C. E. Shannon, "A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955," *AI Magazine*, vol. 27, no. 4, p. 12, 2006.

[10] Y. Bengio, D.-H. Lee, J. Bornschein, and Z. Lin, "Towards biologically plausible deep learning," *CoRR*, vol. abs/1502.04156, 2015. arXiv: 1502.04156.

[11] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," *CoRR*, vol. abs/1811.03378, 2018. arXiv: 1811.03378.

[12] H. H. Tan and K. H. Lim, "Vanishing gradient mitigation with deep learning neural network optimization," *2019 7th International Conference on Smart Computing Communications (ICSCC)*, 2019, pp. 1–4.

[13] F. Emmert-Streib, Z. Yang, H. Feng, S. Tripathi, and M. Dehmer, "An introductory review of deep learning for prediction models with big data," *Frontiers in Artificial Intelligence*, vol. 3, p. 4, 2020, ISSN: 2624-8212.

[14] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[15] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, 1998, pp. 2278–2324.

[16] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," *CoRR*, vol. abs/1411.4038, 2014. arXiv: 1411.4038.

[17] C. Mouton, J. C. Myburgh, and M. H. Davel, "Stride and translation invariance in cnns," *Artificial Intelligence Research*, A. Gerber, Ed., Cham: Springer International Publishing, 2020, pp. 267–281.

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

[19] D. Wilson and T. R. Martinez, "The general inefficiency of batch training for gradient descent learning," *Neural Networks*, vol. 16, no. 10, pp. 1429–1451, 2003, ISSN: 0893-6080.

[20] J. Zhang, S. P. Karimireddy, A. Veit, S. Kim, S. J. Reddi, S. Kumar, and S. Sra, "Why are adaptive methods good for attention models?," 2020. arXiv: 1912.03194.

[21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.

[22] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, and J. Brew, "Huggingface's transformers: State-of-the-art natural language processing," *CoRR*, vol. abs/1910.03771, 2019. arXiv: 1910.03771.

[23] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler, "Efficient transformers: A survey," *CoRR*, vol. abs/2009.06732, 2020. arXiv: 2009.06732.

[24] S. Wang, B. Z. Li, M. Khabsa, H. Fang, and H. Ma, "Linformer: Self-attention with linear complexity," *CoRR*, vol. abs/2006.04768, 2020. arXiv: 2006.04768.

[25] Y. Tay, M. Dehghani, S. Abnar, Y. Shen, D. Bahri, P. Pham, J. Rao, L. Yang, S. Ruder, and D. Metzler, "Long range arena: A benchmark for efficient transformers," 2020. arXiv: 2011.04006.

[26] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[27] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2015 - 18th International Conference Munich, Germany, October 5 - 9, 2015, Proceedings, Part III*, N. Navab, J. Hornegger, W. M. W. III, and A. F.

Frangi, Eds., ser. Lecture Notes in Computer Science, vol. 9351, Springer, 2015, pp. 234–241.

[28] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, IEEE Computer Society, 2016, pp. 770–778.

[29] X. Chu, B. Zhang, Z. Tian, X. Wei, and H. Xia, "Do we really need explicit position encodings for vision transformers?" *CoRR*, vol. abs/2102.10882, 2021. arXiv: 2102.10882.