NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Herman Ryen Martinsen

Autonomous Driving: Vision Transformers for Dense Prediction Tasks

Master's thesis in Computer Science
Supervisor: Frank Lindseth
Co-supervisor: Gabriel Kiss
June 2022

Herman Ryen Martinsen

Autonomous Driving: Vision Transformers for Dense Prediction Tasks

Master’s thesis in Computer Science
Supervisor: Frank Lindseth
Co-supervisor: Gabriel Kiss
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Abstract

Vision Transformers have become extremely popular in the deep learning community in recent years. It all started back in October 2020 with the release of the very first Vision Transformer. This deep learning architecture was inspired by the Transformer model from the natural language processing (NLP) field, but applied the model to visual data instead. By replacing the commonly used convolutional layers with layers of self-attention, the Vision Transformer was able to achieve impressive results. Since then, many new Vision Transformers have been proposed that outperforms previous state-of-the-art models in a wide range of vision tasks.

This thesis investigates how Vision Transformers can be used to process and understand visual data in an autonomous driving setting. More specifically, it explores how segmentation and depth estimation can be done using only a single image as input. First, state-of-the-art Vision Transformers for semantic segmentation and monocular depth estimation are presented, focusing on their contribution to the field. Then, two of these models are selected and combined into a multitask model that is able to perform both tasks. Finally, the proposed multitask model is evaluated through multiple experiments on four different street image datasets. The experiments show that the multitask approach significantly reduces the total inference time, while maintaining a high accuracy for both tasks. Additionally, the experiments show that changing the size of the Transformer-based backbone can be used as a trade-off between inference speed and accuracy.

Collecting and labelling real-world data for dense prediction tasks is a tedious and expensive task. Synthetic data, on the other hand, can easily be generated in large quantities from simulated environments. Motivated by this, one of the experiments investigates how the use of additional synthetic data affects model performance. The results indicate that pre-training on a synthetic dataset effectively increases the accuracy of the model when there is little real-world data available.

Sammendrag

Vision Transformers har blitt ekstremt populære innen dyp læring de seneste årene. Det hele startet i oktober 2020 da den aller første Vision Transformer-modellen ble lansert. Modellen hentet inspirasjon fra Transformer-modellen innen naturlig språkbehandling (NLP), men benyttet den på visuelle data isteden. Ved å bytte ut de tradisjonelle konvolusjonelle lagene med lag med "self-attention", oppnådde Vision Transformer-modellen imponerende resultater. Siden den gang har det blitt lansert mange nye Vision Transformers som utkonkurrerer tidligere toppmodeller innen en rekke visuelle oppgaver.

Denne oppgaven undersøker hvordan Vision Transformers kan brukes til å prosessere og forstå visuelle data knyttet til selvkjørende biler. Mer spesifikt utforskes det hvordan segmentering og dybdeestimering kan gjøres ved å kun benytte enkeltbilder som inndata. Først presenteres toppmodeller av typen Vision Transformers for segmentering og dybdeestimering, med fokus på modellenes bidrag til feltet. Deretter velges det ut to modeller som kombineres til en multitask-modell for begge oppgavene. Til slutt evalueres den foreslåtte multitask-modellen gjennom en rekke eksperimenter på fire ulike datasett bestående av gatebilder. Eksperimentene viser at bruk av en multitask-modell senker den totale inferenstiden betraktelig, samtidig som nøyaktigheten holdes høy. I tillegg viser eksperimentene at ulike "backbone"-størrelser kan benyttes for å regulere mellom høy inferenshastighet og høy nøyaktighet.

Innsamling og annotering av reelle data til prediksjonsoppgaver på piksel-nivå er en møysommelig og kostbar oppgave. Syntetiske data kan derimot enkelt genereres i store mengder ved hjelp av simulerte miljøer. Motivert av dette undersøker et av eksperimentene hvordan bruk av ekstra syntetiske data påvirker modellens nøyaktighet. Resultatene indikerer at trening på syntetiske datasett øker modellens nøyaktighet når det er lite reelle data tilgjengelig.

Preface

This master thesis is a product of my work at the Department of Computer Science at NTNU in Trondheim. The thesis explores the potential of Vision Transformers, a new type of deep learning architecture, in the field of autonomous driving. It has been really exiting to work in such a recent field, with new publications being released all the time. Hopefully my work can be of interest to others in the AI and deep learning community.

I would like to thank my supervisors Frank Lindseth and Gabriel Kiss for their help during the writing of this thesis. Frank and Gabriel have guided me throughout this semester, and provided me with data and computational resources for the experiments of this thesis. I would also like to thank my family and friends for their support and motivating words during this semester.

Herman Ryen Martinsen
Trondheim, June 2020

Contents

Figures

Tables

Acronyms

AI Artificial Intelligence. vii, 5, 7

CNN Convolutional Neural Network. 14, 18, 20, 25

FPS Frames Per Second. 44, 45, 52, 59

GPU Graphics Processing Unit. 41, 59

IoU Intersection over Union. 8, 9

mIoU mean Intersection over Union. 8, 44, 46, 57

MLP Multilayer Perceptron. 15, 18, 23, 29, 30

NAPLab NTNU Autonomous Perception Laboratory. 15, 16, 26, 27, 29, 40, 57

NLP Natural Language Processing. iii, v, 1, 10, 13

NTNU Norwegian University of Science and Technology. vii, 16, 41

RAM Random Access Memory. 41

ReLU Rectified Linear Unit. 7

RMSE Root Mean Squared Error. 10

RNN Recurrent Neural Network. 10

SGD Stochastic Gradient Descent. 7

ViT Vision Transformer. 14, 18, 20, 29, 30

Chapter 1

Introduction

1.1 Motivation

In recent years the research and development of autonomous vehicles has gained a lot of interest. Big companies such as Tesla, Waymo and GM are investing large amounts of time and resources into developing new technology and improving the field. The introduction of autonomous vehicles will provide a lot of advantages, such as reduced number of road accidents, higher traffic efficiency, easier accessibility for disabled and old people, and lower emissions of greenhouse gasses. At the same time it is crucial that the vehicles are reliable and work as expected, or else human lives could possibly be at stake.

For an autonomous vehicle to operate it has to sense its environment. This is usually done by equipping the vehicle with a wide range of different sensors, such as cameras, radar and LiDAR. These sensors are used to identify objects around the vehicle and determine the distance to the objects with high accuracy. While cameras are relatively cheap and compact, a LiDAR sensor is usually both large and expensive. It would therefore be beneficial if cameras could replace LiDARs entirely.

Over the last few years there has been a rapid evolution in the machine learning field. The availability of large amounts of data and powerful computing resources has made deep learning a viable option to solving real-world problems. The field of computer vision focuses on applying deep learning to visual data such as images and videos. This field is especially important for autonomous driving, as it allows the vehicle to extract useful information about its surroundings using cameras. Techniques that are commonly used for autonomous driving are object detection, segmentation and depth estimation.

In 2017 Vaswani *et al.* [1] introduced a novel deep learning architecture named the Transformer. The architecture was designed to solve natural language processing (NLP) tasks such as machine translation, and achieved impressive results. Inspired by the massive success of the Transformer, Dosovitskiy *et al.* [2] successfully applied the architecture to vision tasks in 2020. The novel architecture, named the Vision Transformer, achieved new state-of-the-art results and took the deep learning community by storm. Since then, many new Vision Transformer architectures have been proposed.

This thesis will investigate how Vision Transformers can be used for dense prediction tasks in autonomous driving. It will focus on the tasks of monocular depth estimation and semantic segmentation, and design a multitask model that is able to perform both tasks simultaneously. To study the effectiveness of the model, it will be trained and evaluated on multiple autonomous driving datasets.

1.2 Goal and Research Questions

The main goal of this thesis is to design a multitask Vision Transformer for dense prediction tasks and investigate how the architecture performs in an autonomous driving setting. The architecture should be able to perform both semantic segmentation and monocular depth estimation simultaneously. It should also be able to run in real-time so the architecture has potential to be deployed in an autonomous vehicle. In order to achieve this goal the following research questions are proposed:

RQ 1: How does a multitask Vision Transformer trained for both segmentation and depth estimation simultaneously perform compared to models trained for individual tasks?

RQ 2: Can synthetic datasets be used to increase model performance when there is little real-world data available?

RQ 3: How accurate are the absolute depth predictions from the model?

1.3 Research Method

The chosen research strategy for this thesis is experiment. This research strategy involves studying the correspondence between a cause and effect. The experiments of this thesis will investigate how factors such as training approach and choice of backbone affects performance. Observation and documents will be used as data generation methods. Already existing datasets and models will mostly be used, which can be regarded as documents. Observation will be used to evaluate the performance of the models. For this evaluation, a combination of quantitative and qualitative analysis will be used. Quantitative analysis involves computing the accuracy or error of the model on a validation dataset using common evaluation metrics. Qualitative analysis involves visual inspection of individual predictions to look for abnormalities and other interesting findings.

1.4 Contributions

This thesis has two main contributions. First, it provides a literature review on Vision Transformers for the dense prediction tasks of monocular depth estimation and semantic segmentation. The literature review presents state-of-the-art architectures for the respective tasks and describes their contributions to the field. Next, a multitask model for both monocular depth estimation and semantic segmentation is proposed. Two existing Vision Transformer architectures are combined into a single model to perform both tasks simultaneously. The effectiveness of the model is evaluated through multiple experiments.

1.5 Thesis Structure

The thesis is divided into the following chapters:

Chapter 1 - Introduction: Presents the motivation for the study, the goal and research questions, the research method of choice and the contributions of the thesis.

Chapter 2 - Background and Related Work: Introduces theory related to deep learning, computer vision and Vision Transformers, and presents relevant datasets and state-of-the-art models.

Chapter 3 - Methodology: Presents the chosen models and datasets, necessary data preparations and training details.

Chapter 4 - Experiments and Results: Contains qualitative and quantitative results from the experiments conducted.

Chapter 5 - Discussion: Discusses the implications of the results and relates the findings to the research questions.

Chapter 6 - Conclusion and Future Work: Concludes the thesis, summarizes the work done and key findings, and presents possible directions for further work.

Chapter 2

Background and Related Work

This chapter presents the background material and related work of this thesis. First, the fundamentals of deep learning and computer vision are explained. Then, the Transformer and Vision Transformer architectures are described in detail. Finally, relevant datasets and state-of-the-art Vision Transformers for semantic segmentation and monocular depth estimation are presented.

2.1 Deep Learning

Deep learning is a subfield of artificial intelligence (AI) and machine learning that uses artificial neural networks to solve problems. These networks mimic the way the human brain works by connecting artificial neurons into massive networks. When trained on large amounts of data, the networks are able to achieve impressive results in a wide range of different tasks. This section covers the basics of artificial neural networks.

2.1.1 Artificial Neuron

Artificial neurons are the basic building blocks of a neural network. They are inspired by the biological neurons found in the human brain. An artificial neuron is designed as a mathematical function that takes multiple values x_i as input. Each input x_i is multiplied by a weight w_i to regulate the importance of the input. The sum of the weighted inputs are calculated, and a bias b is added to obtain a value z. This can be expressed through the following equation:

$$z = \sum_{i=1}^{n} w_i x_i + b \tag{2.1}$$

The value z is sent through an activation function f to introduce a non-linearity in the network. The final output value $a = f(z)$ is then sent to the next neuron of the network.

2.1.2 Neural Network

To make use of artificial neurons they have to be combined into a structure called a neural network. The neural network consists of multiple layers of artificial neurons, with connections between each layer. The first layer of the network is the input layer. The input layer is responsible for taking the input data and passing it on to the rest of the network. Next are the hidden layers, which are responsible for processing of the input data. It is common to have multiple hidden layers in a neural network. Finally, there is the output layer, which takes the processed data from the last hidden layer and produces the final results. An example of a simple artificial neural network can be seen in Figure 2.1.

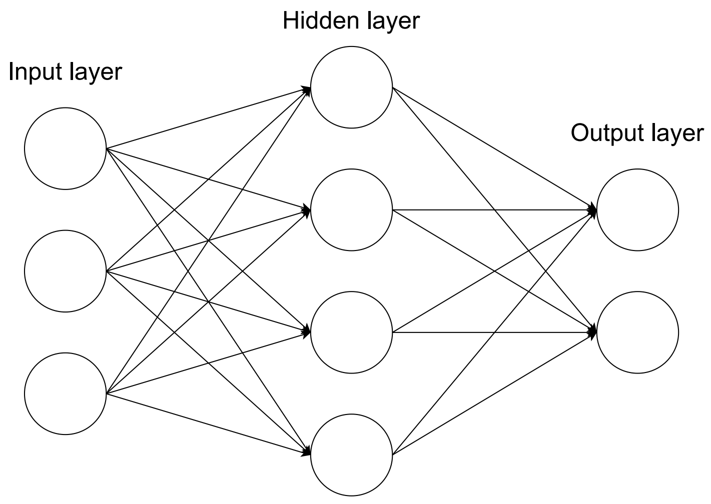


Figure 2.1: A simple artificial neural network

2.1.3 Activation Function

An activation function is a function used in neural networks to process the output of artificial neurons. Activation functions are usually applied in both the hidden layers and the output layer of the network. In the hidden layers the activation function introduces non-linearity to the network, and ensures that the output values are within a certain range. In the output layer the activation function converts the results to the desired format, e.g. a probability distribution.

There are a wide range of different activation functions to choose from. One of these is the *sigmoid function*. This is a logistic function that outputs values between 0 and 1. It is thus commonly used to convert an output value to a probability score. The function is known to cause something known as the vanishing gradient problem, and is thus normally not applied in the hidden layers of the network. The sigmoid function is defined as:

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

The *rectified linear unit (ReLU)* is the most commonly used activation function. It is a fairly simple function that returns 0 for negative input values, and the value itself for positive input values. As ReLU does not cause the vanishing gradient problem, it is well suited for the hidden layers in the network. The ReLU function is defined as:

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

Another widely used activation function is the *softmax* function. The function takes K values as input and converts them to a probability distribution where the values add up to 1. The softmax function is commonly used in the output layer of the network for classification tasks. It is defined as:

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{2.4}$$

2.1.4 Forward Pass and Backpropagation

When processing data, the neural network sends the input data through all of its layers in the forward direction. This is known as a *forward pass*. In each layer, the artificial neurons process the values from the previous layer using Equation (2.1), and apply an activation function. Then, the calculated output value is sent to the next layer. This process is repeated for all of the layers, until the data reaches the output layer of the network. Here, the final results are calculated.

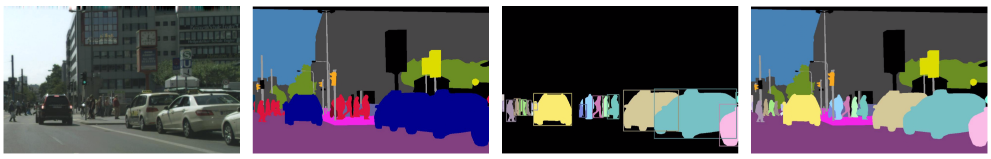
In order for the model to learn, it has to go through a process called *backpropagation*. First, the error of the predicted results is calculated using a *loss function*. Then, the weights of the network are adjusted to minimize the error. This is done by calculating the gradient of the loss function with respect to each weight. The calculations are done step by step in the backward direction, starting from the output layer and working towards the input layer, until all weights are updated. An *optimizer* algorithm is responsible for the updating of the weights. There are a wide range of different optimizers to choose from. Two of the most commonly used optimizers are stochastic gradient descent (SGD) and Adam [3].

2.2 Computer Vision: Tasks and Metrics

Computer vision is a subfield of artificial intelligence (AI) that focuses on visual data such as images and videos. The goal of computer vision is to make computers able to understand visual data in the same way as a humans do. This involves extracting useful information from the data, such as the position and category of objects, and using this information to take actions. Traditionally, conventional algorithms were used for computer vision. Today, the field is largely dominated by deep learning methods. Over the years, a large number of different vision tasks have been proposed. This thesis mainly focuses on the tasks of segmentation and depth estimation. This section gives a brief introduction to the tasks, and presents common evaluation metrics.

2.2.1 Segmentation

Segmentation is a dense prediction task where the image pixels are labeled into different categories. There are three different segmentation tasks: semantic, instance and panoptic segmentation. For semantic segmentation each image pixel is assigned a category corresponding to what kind of object/structure the pixel is a part of, e.g. car, sky or building. The segmentation does not separate between different instances of the same category. For instance segmentation each instance of a category is assigned a separate label. Usually only the categories of interest are labelled, not the whole image. Panoptic segmentation is a combination of semantic and instance segmentation. A visual example of the different segmentation tasks can be seen in Figure 2.2. In this thesis the semantic segmentation task is explored.



(a) Original image **(b)** Semantic **(c)** Instance **(d)** Panoptic

Figure 2.2: Examples of the different segmentation tasks. Images are taken from Kirillov *et al.* [4].

Evaluation Metrics

Multiple different metrics can be used to evaluate segmentation methods. One of the simplest is the *pixel accuracy*. It is found by calculating the percentage of image pixels that are labelled correctly. Unfortunately, the pixel accuracy has a tendency to provide misleading results, especially if the class distribution of the image is imbalanced. A more robust evaluating metric is the *Intersection over Union (IoU)*, also referred to as the Jaccard index. Given a ground truth segmenatation mask y and a predicted segmentation mask $\hat{y}$, the IoU is calculated using the following equation:

$$IoU = \frac{y \cap \hat{y}}{y \cup \hat{y}} = \frac{TP}{TP + FP + FN} \tag{2.5}$$

Here TP, FP and FN refers to the number of true positives, false positives and false negatives, respectively. The *mean Intersection over Union (mIoU)* is commonly used when the dataset contains multiple classes. It is found by calculating the sum of the class-wise IoU scores and dividing by the total number of classes N:

$$mIoU = \frac{1}{N}\sum_{i=1}^{N} IoU_i \tag{2.6}$$

2.2.2 Depth Estimation

Depth estimation is a dense prediction task that uses images to predict depth in a scene. The goal is to obtain the distance from the camera to each pixel in the image. There are mainly two methods used for this task: stereo depth estimation and monocular depth estimation. Stereo is the traditional approach, and uses two cameras with a known distance between each other to estimate depth. This is done by finding matching pixels in the two images and calculating the difference between them. The obtained value, called the disparity, can then be used to calculate depth. More recently, the rise of deep learning has made it possible to estimate depth using a single image only. This approach, commonly referred to as monocular depth estimation, uses deep neural networks to predict the depth of each pixel. Monocular depth estimation can be done using supervised learning with images and corresponding ground truth depth maps, or with self-supervised learning and no ground truth at all. This thesis explores the supervised approach for monocular depth estimation.

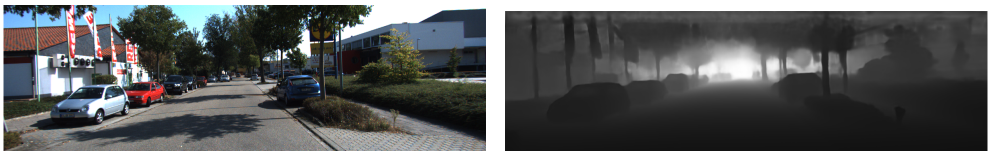

(a) Original image **(b)** Depth prediction

Figure 2.3: Example of depth estimation

Evaluation Metrics

Most recent work related to monocular depth estimation uses the metrics presented by Eigen *et al.* [5] to evaluate model performance. This section describes these commonly used evaluation metrics, and shows how they are calculated given a ground truth depth map y containing N valid pixels, and a predicted depth map $\hat{y}$.

The absolute error $|y_i - \hat{y}_i|$ gives the difference between the predicted value and the ground truth value. However, it does not take into account the magnitude of the ground truth value. This is problematic when dealing with values of different magnitudes, since a given difference should be more significant for small values than large values. The absolute relative error incorporates the magnitude by dividing the absolute error by the actual value. Both *absolute relative error* and *squared relative error* are commonly used as evaluation metrics:

$$AbsRel = \frac{1}{N}\sum_{i=1}^{N}\frac{|y_i - \hat{y}_i|}{y_i} \tag{2.7}$$

$$SqRel = \frac{1}{N}\sum_{i=1}^{N}\frac{||y_i - \hat{y}_i||^2}{y_i} \tag{2.8}$$

Another commonly used evaluation metric is the *root mean squared error (RMSE)*, which has two different versions:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}||y_i - \hat{y}_i||^2} \tag{2.9}$$

$$RMSElog = \sqrt{\frac{1}{N}\sum_{i=1}^{N}||\log y_i - \log \hat{y}_i||^2} \tag{2.10}$$

Finally, there is the *threshold accuracy*. This is the percentage of the image pixels that have a max ratio between prediction and ground truth value below a given threshold th. For the threshold th, the values 1.25, 1.25^2 and 1.25^3 are commonly used. The threshold accuracy is calculated using the following equation:

$$\%\text{ of } y_i \text{ s.t. } \max\left(\frac{y_i}{\hat{y}_i}, \frac{\hat{y}_i}{y_i}\right) = \delta < th \tag{2.11}$$

2.3 Transformers

The field of natural language processing (NLP) was originally dominated by recurrent neural networks (RNNs). These architectures performed reasonably well, but also had some essential flaws. First, RNNs were unable to capture long range dependencies in sentences, which in turn affected the performance of the models. Second, RNNs processed sentences one word at a time, which led to slow training times. This all changed when the Transformer architecture was introduced by Vaswani *et al.* [1] in 2017. The Transformer is based entirely on attention mechanisms, which helps the model capture long range dependencies and increases accuracy. In addition, the model can process multiple words in parallel to significantly speed up the training process. The Transformer achieved impressive results on machine translation tasks, and has since inspired the development of other successful language models such as BERT [6] and GPT [7]. The architecture has also been successfully applied to other fields, such as computer vision [2]. This section describes the Transformer architecture in detail and explains how the attention mechanism works.

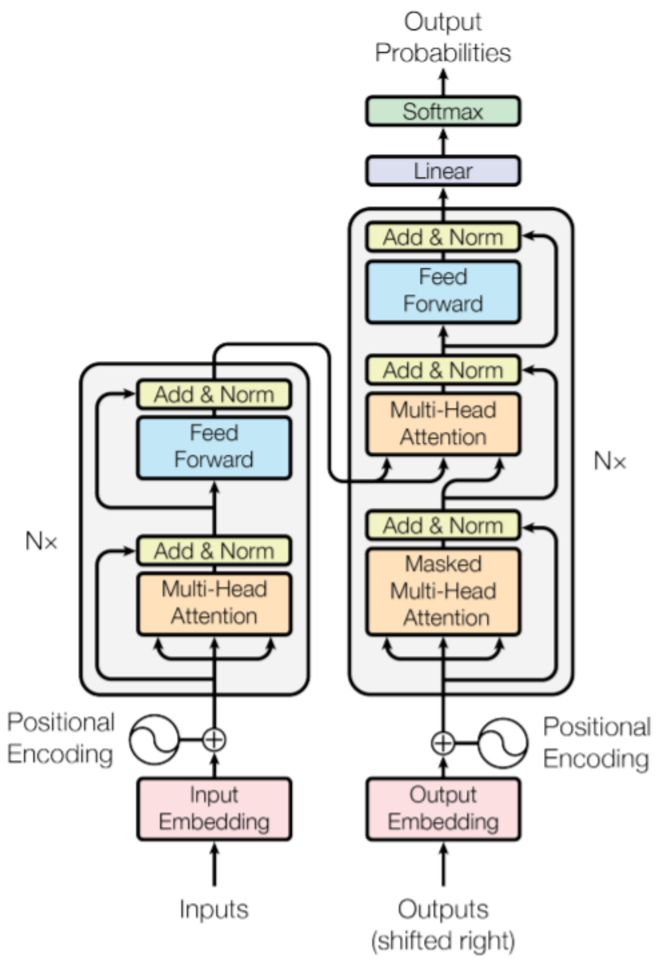


Figure 2.4: The Transformer model architecture. Image taken from Vaswani *et al.* [1].

2.3.1 Architecture

The Transformer architecture uses an encoder-decoder design. First, an input sequence is fed into the encoder part of the network. The encoder maps the input sequence to an intermediate representation z that contains contextual information about the sequence. Then, z is passed on to the decoder part of the network. The decoder uses z to generate the output sequence y in an autoregressive manner. This means that y is generated one word at the time, using all the previously generated words at each step. An overview of the entire Transformer architecture can be seen in Figure 2.4.

Pre-processing

Computers are not able to understand words and sentences. Thus, the input sequence has to be converted into a numerical representation. This representation has to preserve the contextual and positional information of the sequence. This is done by using an embedding space and positional encoding. The embedding space maps the words of the sequence into vector representations. The vector representations preserve contextual information, meaning that similar words have similar vectors. Next, positional encoding is added to the vectors to preserve positional information. This step is crucial to produce accurate results, as words can have different meanings depending on where they appear in the sequence.

Encoder

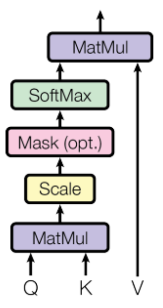
The encoder consists of N identical layers, each with two crucial components: a multi-head attention operation and a simple feed-forward network. First, the input sequence is processed by the multi-head attention operation. Here self-attention is used to gain information about the relationship between the words of the sequence. More details on the attention mechanism are given in Section 2.3.2. Next, the calculated attention vector is sent through a feed-forward network. The feed-forward network reshapes the vector so it can be processed by the subsequent layer. After applying N encoder layers the final output is passed on to the decoder.

Decoder

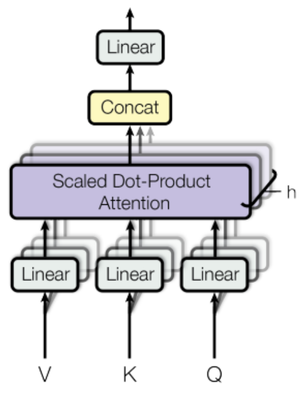
The decoder consists of N identical layers, and has a fairly similar structure to the encoder. An additional masked multi-head attention operation is added to the beginning of each decoder layer. This is a modified version of the multi-head attention that masks out some of the attention scores of the target sequence. This ensures that the model only uses attention scores of previous words in the target sequence for prediction. Next is a multi-head attention operation that processes attention scores from both the encoder and decoder. Here the model gains information about the relationship between words in the input and target sequence. Finally, a feed-forward network is applied, just like in the encoder.

Linear Layer and Softmax

The final output of the decoder after N layers is processed by a linear layer and a softmax function to make the final prediction. The dimensions of the output gets expanded to a size n by the linear layer, where n is the total number of words in the vocabulary. The softmax function produces a probability distribution that is used to predict the next word in the sequence.



(a) Scaled Dot-Product Attention



(b) Multi-Head Attention

Figure 2.5: Scaled Dot-Product Attention and Multi-Head Attention. Image taken from Vaswani *et al.* [1].

2.3.2 Attention

The Transformer was the first deep learning architecture based entirely on attention mechanisms. This clever use of the self-attention mechanism made the Transformer a state-of-the-art architecture in natural language processing (NLP). This section describes how the self-attention mechanism works, and how the calculation of self-attention is done in parallel using multi-head attention.

Scaled Dot-Product Attention

The self-attention mechanism used in the Transformer architecture is referred to as scaled dot-product attention. It is calculated using a set of queries Q, keys K and values V, which are different vector representations of the input sequence. The queries and keys have a common dimension of d_k, while the values have a dimension of d_v. The scaled dot-product attention is calculated using the following equation:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.12}$$

A visual representation of how the scaled dot-product attention is calculated can be seen in Figure 2.5a. Here the optional masking step applied in the decoder is also included.

Multi-Head Attention

The Transformer uses multi-head attention to calculate attention scores. This means that the self-attention calculation is done using h different heads. First, the queries, keys and values are projected to each of these heads. Then, the scaled dot-product attention is calculated in parallel using the h heads. Finally, the h calculated attention vectors are concatenated and projected into a final attention score. This whole process is illustrated in Figure 2.5b. The big advantage of this approach is that the heads can attend to various parts of the input sequence individually.

2.4 Vision Transformer (ViT)

The Vision Transformer (ViT) was introduced by Dosovitskiy *et al.* [2] in October 2020. It is one of the earliest attempts at applying Transformers to computer vision tasks. Some previous works attempted to use the attention mechanism in conjunction with CNNs. The Vision Transformer, on the other hand, uses a more or less pure Transformer architecture for the task of image classification. As the Transformer usually works with sequences, the input image has to be split into a series of patches that are fed into the model. The Vision Transformer achieved state-of-the-art results and revolutionized the field of computer vision. This section describes how the Vision Transformer architecture works.

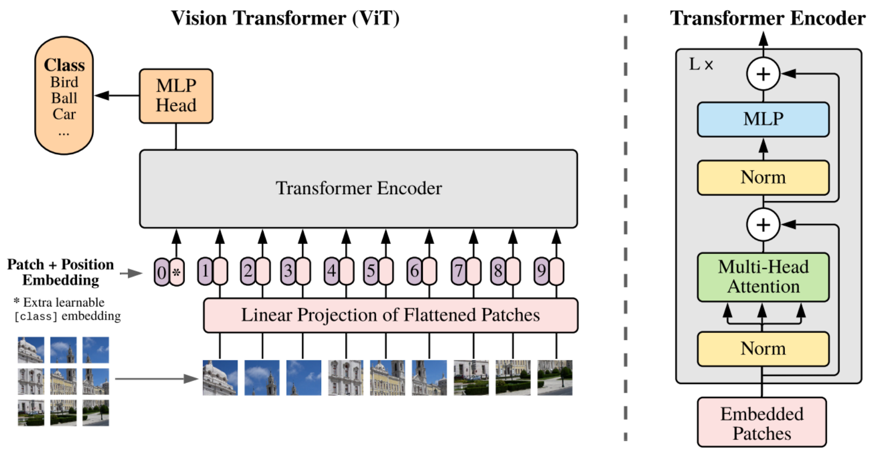


Figure 2.6: The Vision Transformer architecture. Image taken from Dosovitskiy *et al.* [2].

2.4.1 Architecture

An overview of the Vision Transformer architecture can be seen in Figure 2.6. The architecture uses the original Transformer encoder proposed by Vaswani *et al.* [1] with some small modifications for the image classification task. As the Transformer originally processes sequences of words, the input image has to be converted to a form that is digestible by the Transformer encoder. This is done by splitting the image into a sequence of patches, each with a size of 16×16 pixels. The patches are then are flattened and linearly projected to produce patch embeddings. Positional embeddings are added to the patch embeddings in order to preserve the positional information of the image. Then, the final patch embeddings are fed into the Transformer encoder. The encoder processes the patch embeddings in more or less the same way as the original architecture.

A learnable class token is used for the image classification task. The class token is fed into the Transformer encoder together with the patch embeddings. At the other end of the encoder, the output for the class token serves as a intermediate image representation. The image representation is processed by a simple MLP classification head to acquire the final class prediction for the image.

2.5 Datasets

This section presents potential datasets for the experiments of this thesis. Both real-world and synthetic datasets are included. All the datasets, except NAPLab, contain ground truth for both semantic segmentation and depth estimation.

2.5.1 Real-World Datasets

Cityscapes

Cityscapes [8] is a large-scale dataset and benchmark for semantic urban scene understanding. The dataset covers the three major segmentation tasks of semantic segmentation, instance segmentation and panoptic segmentation. It consists of 5 000 images with high quality pixel-level annotations, and an additional 20 000 images with coarse annotations. In addition the dataset contains disparity maps computed from stereo images, which can be used to train and evaluate depth estimation models. The data were collected in 50 different cities in Germany and neighbouring countries by a moving vehicle.

KITTI

The KITTI vision benchmark suite [9] and KITTI dataset [10] was released back in 2012, but is still widely used to train and evaluate models at autonomous vehicle-related tasks. The dataset covers several vision tasks including stereo, optical flow, visual odometry, object detection, monocular depth estimation, segmentation and tracking. KITTI contains 94 500 images with corresponding depth maps obtained from LiDAR data, which can be used for the task of monocular depth estimation. The dataset also has 400 images with annotations for semantic segmentation. The segmentation dataset is mainly suited for evaluation and fine-tuning due to the small number of annotated images. The data were collected in the mid-size city of Karlsruhe in Germany, in rural areas and on highways.

KITTI-360

The KITTI-360 dataset [11] was released in 2021 and is the successor of the original KITTI dataset. The dataset consists of LiDAR point clouds, front-facing stereo images and 180-degree fisheye images from each side of the vehicle, which results in a full 360-degree view around the vehicle. KITTI-360 focuses on semantic scene understanding and provides dense semantic and instance annotations for both LiDAR point clouds and 2D images. The labelling is done by annotating 3D point clouds with bounding primitives, which is then transferred to the 2D image domain. This makes it possible to annotate a large amount of images in a short period of time. Thanks to this efficient labelling method the dataset has a total of 61 280 images with semantic segmentation annotations and corresponding LiDAR point clouds. The data were collected in multiple suburbs of Karlsruhe, the same city that was used for the KITTI dataset.

Audi Autonomous Driving Dataset (A2D2)

The Audi Autonomous Driving Dataset (A2D2) [12] is a large dataset developed for autonomous driving tasks. The dataset consists of images and 3D point clouds collected with multiple cameras and LiDAR sensors, giving a full 360 degree view around the vehicle. In addition, the dataset contains annotations for the tasks of semantic segmentation, point cloud segmentation and 3D object detection. For semantic segmentation the dataset has a total of 41 277 annotated images covering 38 different semantic classes. The images also have corresponding LiDAR point clouds that can be projected to 2D depth maps. The data were collected on highways, country roads and cities in the south of Germany under varying weather conditions.

NAPLab

The NTNU Autonomous Perception Laboratory (NAPLab) [13] is a research group at NTNU in Trondheim. The group focuses on research and development of deep learning architectures for autonomous vehicles in a Nordic environment. During the writing of this thesis, NAPLab provided a small dataset collected in the streets of Trondheim. The dataset contains 10 images with semantic segmentation annotations. Unfortunately, no depth ground truth is provided.

2.5.2 Synthetic Datasets

Virtual KITTI

Virtual KITTI [14] was released in 2016 and is a virtual recreation of the real-world videos from the KITTI dataset. The dataset was created using the Unity game engine, and provides automatically generated annotations for object detection, semantic segmentation, instance segmentation, depth, optical flow and tracking. It contains 5 different driving sequences under different imaging and weather conditions, resulting in a total of 21 000 annotated images. In 2020 Virtual KITTI 2 [15] was released, which is an updated version of the dataset with increased photorealism.

SYNTHIA

The SYNTHetic collection of Imagery and Annotations (SYNTHIA) [16] is a synthetic dataset released in 2016 that focuses on semantic segmentation of urban street scenes. The dataset is captured in a virtual city during different seasons and under varying weather and lighting conditions. Images from multiple viewpoints are available, giving a effective 360-degree field of view around the virtual vehicle. The dataset consists of more than 213 400 photo-realistic images with automatically generated annotations for both segmentation and depth. The segmentation labels contain a total of 13 semantic classes.

Apollo Synthetic Dataset

The Apollo Synthetic Dataset [17] is a photo-realistic virtual dataset for autonomous driving released in 2019. The dataset consists of 7 different virtual environments that was created using the Unity game engine. The environments resembles real-world settings such as highway, urban, residential, downtown and indoor parking garage. The dataset contains 273 000 images with automatically generated annotations for multiple vision tasks, including semantic segmentation, instance segmentation, depth estimation, object detection and lane mark detection. Different lighting conditions, weather conditions and road surface qualities are simulated to bring more variety to the dataset.

All-In-One Drive

All-In-One Drive (AIODrive) [18] is a synthetic dataset created with the goal of uniting all autonomous driving tasks into one single dataset. The dataset was generated using the CARLA simulator [19] and contains 500 000 images annotated for all the common vision tasks, including semantic segmentation and depth estimation. Multiple sensors with 360-degree field of view are available, such as RGB, stereo, depth, LiDAR, radar and GPS. The dataset also provides varying weather and lighting conditions, together with rare traffic scenarios like fast driving, violation of traffic rules and car crashes.

2.6 Related Work: Vision Transformers for Dense Prediction Tasks

This section presents related works using Vision Transformers for semantic segmentation and monocular depth estimation. When finding models for this section, three different properties were considered. First, the model should be able to achieve a high accuracy on an autonomous driving dataset. Next, the model should be lightweight and able to perform inference close to real-time. Finally, the model should provide a significant contribution to the field.

2.6.1 Semantic Segmentation

Segmentation Transformer (SETR)

One of the earliest attempts at applying Transformers to the task of semantic segmentation is the Segmentation Transformer (SETR), which was introduced by Zheng *et al.* [20] in December 2020. Prior to their work, the common approach for semantic segmentation was to utilize a CNN backbone. One major limitation with this approach is that CNNs have a limited receptive field, which makes it difficult to capture long-range dependencies in the image. To cope with this issue, a novel architecture that integrates a Transformer encoder was proposed. The Transformer encoder has a global receptive field at every layer, and is thus better at modelling the global context of the image. The encoder is combined with a lightweight convolutional decoder that produces the final predictions. The model achieves a mIoU of 82.15 on the Cityscapes dataset.

SegFormer

In May 2021 Xie *et al.* [21] introduced SegFormer, a simple and efficient encoder-decoder design for semantic segmentation with Transformers. The model uses a Transformer encoder inspired by ViT [2] with multiple modifications to make it better suited for semantic segmentation. The encoder has a hierarchical structure and outputs multi-scale feature maps to the decoder. It does not use any positional encoding, something that increases performance when the inference resolution differs from the training resolution, since no interpolation is needed. In addition, the encoder uses the efficient self-attention calculation method introduced in PVT [22], which gives faster processing of high resolution images. The model uses a lightweight decoder consisting of only MLP layers that further contributes to the efficiency of the model. There are a total of 6 different model sizes, where the smallest one is fast and suitable for real-time applications, while the largest model achieves high accuracy on both the Cityscapes and ADE20K dataset. The model also performs well on corrupted Cityscapes data, which indicates that the model is robust and and could potentially be suited for safety-critical tasks such as autonomous driving. The largest model achieves a mIoU of 84.0 on the Cityscapes dataset.

MaskFormer

Traditionally the tasks of semantic segmentation and instance segmentation are handled separately. Semantic segmentation is treated as a per-pixel classification task, while instance segmentation utilizes mask classification. However, Cheng *et al.* argues that the tasks can be unified and solved using a single model. In July 2021 they introduced the MaskFormer [23] architecture, which utilizes mask classification to generate semantic segmentation predictions. The architecture incorporates a Transformer decoder that predicts binary segmentation masks with corresponding class labels. MaskFormer is tested with multiple different backbones, but the Swin Transformer [24] gives the best results. Unfortunately, the paper only reports results on the Cityscapes dataset using a ResNet backbone. With this backbone, the model achieves a mIoU of 81.4.

Mask2Former

In December 2021 Cheng *et al.* [25] introduced a new architecture named Mask2-Former, which build upon the original MaskFormer architecture. Several improvements are made in order to increase performance and simplify training. A masked attention technique is implemented in the Transformer decoder, which limits attention calculations to local features around the predicted segments. This gives faster convergence and better performance. In addition, multi-scale high-resolution features are used in order to better detect small objects. Finally, multiple small changes are made to further improve performance and save training memory. Mask2Former achieves a mIoU of 84.5 on the Cityscapes dataset.

SeMask

Vision Transformers for semantic segmentation often pre-train the encoder on a large image classification dataset such as ImageNet [26], and then fine-tune the model on a smaller semantic segmentation dataset. Due to the small size of the segmentation dataset and the differences between the classification and segmentation task, the encoder struggles to capture the semantic context of the image. To solve this issue Jain *et al.* [27] introduced SeMask in December 2021. The architecture implements a semantic layer that follows the Transformer layer at every stage of the encoder. The semantic layer consists of multiple SeMask blocks that applies a semantic attention operation to the feature maps. During training, prior maps from the semantic layers are processed by a lightweight semantic decoder to give additional supervision. The SeMask block can be integrated with any hierarichal Vision Transformer, but in the paper the Swin Transformer [24] and SegFormer [21] is utilized. The best performing model is able to achieve an impressive mIoU of 84.98 on the Cityscapes dataset.

Lawin Transformer

In January 2022 Yan *et al.* [28] introduced the Lawin Transformer, an efficient Transformer-based architecture for semantic segmentation. They argue that current Vision Transformers for semantic segmentation are not able to produce contextual information at multiple scales, something that harms both the performance and efficency of the models. To cope with this issue, a novel decoder called large window attention spatial pyramid pooling (LawinASPP) is introduced. The decoder is inspired by atrous spatial pyramid pooling (ASPP) [29], but uses a new technique called large window attention instead of atrous convolutions. The idea behind large window attention is to let the patches query a larger area of the feature map. By using different ratios between the patch size and the queried area the decoder is able to produce multi-scale contextual information. The LawinASPP decoder can be combined with any hierarchical Vision Transformer encoder, and both SegFormer and Swin Transformer are used as encoders in the paper. The Lawin Transformer with a SegFormer encoder is able to increase the accuracy while lowering the computational cost compared to the original SegFormer architecture. When utilizing a Swin Transformer encoder, the Lawin Transformer is able to achieve a mIoU of 84.4 on the Cityscapes dataset.

2.6.2 Monocular Depth Estimation

AdaBins

In November 2020 Bhat *et al.* [30] introduced the AdaBins architecture for monocular depth estimation. The architecture uses a simple CNN encoder-decoder design together with a novel Transformer-based building block named AdaBins in order to achieve impressive results. The depth estimation problem is regarded as a classification task, where the depth range is split into adaptive bins with varying size for each input image. The bin widths are estimated using a smaller version of the original ViT [2] architecture, named mini-ViT. In order to avoid sharp depth discontinuities in the predicted depth maps, a linear combination of the bin centers is used for the final prediction. AdaBins achieves an absolute relative error of 0.058 on the KITTI dataset.

Dense Prediction Transformer (DPT)

The Dense Prediction Transformer (DPT) was proposed by Ranftl *et al.* [31] in March 2021, and is a novel architecture for the tasks of monocular depth estimation and semantic segmentation. The architecture adapts a traditional encoder-decoder design, but replaces the commonly used CNN backbone with the ViT [2] architecture. The ViT backbone provides a global receptive field through all of its layers, and preserves the initial feature map resolution. These properties makes it possible to capture finer details in the images. The ViT backbone is combined with a convolutional decoder that fuses feature maps from different stages of the encoder in order to produce the final prediction. The architecture is pre-trained on a massive monocular depth estimation dataset of 1.4 million images, and then fine-tuned on the KITTI dataset to achieve an absolute relative error of 0.062. When trained for semantic segmentation on the ADE20K dataset, the model achieves a mIoU of 49.02.

GLPDepth

In January 2022 Kim *et al.* [32] introduced a novel architecture and training strategy for the task of monocular depth estimation. They argue that understanding both the global and local context of an image is essential to generate accurate depth predictions. In order to achieve this a Transformer-based global-local path network named GLPDepth is proposed. The architecture uses the SegFormer encoder introduced by Xie *et al.* [21] to capture global dependencies, and a lightweight decoder with skip connections to integrate local information. A new depth-specific augmentation technique is proposed to further improve the performance of the model. The augmentation method is inspired by CutDepth [33], but is modified to better preserve vertical information in the image. GLPDepth achieves an absolute relative error of 0.057 on the KITTI dataset.

DepthFormer

The DepthFormer architecture for monocular depth estimation was proposed by Li *et al.* [34] in March 2022. Just like Kim *et al.* [32], they emphasize the importance of being able to capture both the global and local information of the image during the encoder stage. This is accomplished by using two separate encoder branches. The first branch utilizes the Swin Transformer [24] to model long-range correlation in the image, while the second branch uses convolutions to extract local information. A novel Hierarchical Aggregation and Heterogeneous Interaction (HAHI) module is introduced to combine the information from both branches and enhance the feature maps. For the KITTI dataset the model achieves an absolute relative error of 0.052, which is state-of-the-art level.

BinsFormer

In April 2022 Li *et al.* [35] introduced a novel architecture for the task of monocular depth estimation named BinsFormer. The architecture is inspired by AdaBins [30], but makes multiple modifications to further improve model performance. A Transformer decoder is implemented to enhance the adaptive bins generation. Similarly to the approach of Carion *et al.* [36] for object detection, the bin generation is viewed as a direct set prediction problem. The Transformer decoder incorporates a multi-scale design where the feature maps are processed in a coarse-to-fine manner. In addition, an auxiliary scene classification task is used during training to improve the model performance. The architecture is compatible with any backbone, but using the Swin Transformer [24] gives the best results. BinsFormer achieves an absoulte relative error of 0.052 on the KITTI dataset, and outperforms all existing methods on the remaining evaluation metrics, making it the current state-of-the-art method for monocular depth estimation.

Chapter 3

Methodology

This chapter discusses the choice of models and datasets for the experiments of this thesis. The chosen models are described thoroughly, and an detailed explanation of the necessary dataset preparations is provided. The chapter also presents the computational hardware and training strategy used in the experiments.

3.1 Choice of Models

The main goal of this thesis is to design a multitask Vision Transformer for segmentation and depth estimation by combining two existing models. Thus, two of the models presented in Section 2.6 have to be chosen, one for each task. This section goes through each of the models and gives an explanation of why it was chosen/not chosen. During the selection process, both the accuracy and the efficiency of the models were considered.

3.1.1 Semantic Segmentation Model

Segmentation Transformer (SETR)

The Segmentation Transformer (SETR) was one of the first successful attempts at using Transformers for semantic segmentation. The architecture achieved impressive results at the time of publication, and has since been an important source of inspiration for other Transformer-based segmentation models. The field of computer vision evolves rapidly, and many architectures have been proposed that outperforms SETR in terms of both accuracy and efficiency. SETR will therefore not be used in the experiments.

SegFormer

The SegFormer architecture uses an efficient self-attention calculation and a lightweight all-MLP decoder. These design choices makes the model efficient and suitable for real-time applications. SegFormer also provides 6 different backbones sizes, which gives a lot of flexibility regarding speed-accuracy trade-off. Finally, the model shows robustness on corrupted data, which indicates that the model could be suited for safety-critical tasks like autonomous driving. All of these properties makes SegFormer a really strong candidate, and it is thus chosen as the segmentation model of this thesis.

MaskFormer and Mask2Former

The fact that MaskFormer and Mask2Former are able to perform multiple different segmentation tasks makes them interesting models for this thesis. In addition, the semantic segmentation task is solved in a novel way by using mask classification. Mask2Former is considered the most promising candidate as it makes multiple improvements over the original MaskFormer architecture. However, the Mask2Former decoder is quite heavy compared to some of the other architectures, and it would thus be hard to accomplish real-time segmentation with the model. Consequently, the model was not chosen for the experiments.

SeMask

SeMask is able to increase the performance of semantic segmentation models by incorporating a small, task-specific layer that captures semantic context into the Transformer encoder. The new layer is lightweight and efficient, so the overall computational complexity is only slightly increased. Since the encoder modifications are made specifically for the task of semantic segmentation, it is uncertain how they affect other dense prediction tasks such as monocular depth estimation. Considering that the main goal of this thesis is to design a multitask model with a shared encoder, the contributions of SeMask seem too task-specific. Consequently, SeMask will not be used for the experiments.

Lawin Transformer

The Lawin Transformer introduces a new and efficient decoder design for the task of semantic segmentation. The decoder can be incorporated with the SegFormer architecture in order to increase accuracy while lowering the computational cost. The original plan was thus to use the Lawin Transformer for the experiments. However, at the time of writing this thesis, only a small part of the code was publicly available. An attempt was made to integrate the code with the official SegFormer code, but the achieved results did not match the ones reported in the paper. It was therefore decided to not explore the Lawin Transformer further in this thesis.

3.1.2 Depth Estimation Model

AdaBins

AdaBins proposes to view depth estimation as a classification task where the depth range is divided into adaptive bins. This idea is new and interesting, and the model is able to achieve impressive results. The model uses a CNN backbone, and only a small part of the architecture is Transformer-based. As the goal of this thesis is to explore the use of Vision Transformers for dense prediction tasks, it is desirable to pick a model that utilizes Transformers to a greater extent. Consequently, AdaBins will not be used for the experiments of this thesis.

Dense Prediction Transformer (DPT)

The Dense Prediction Transformer (DPT) is an interesting model as it is able to perform both the task of monocular depth estimation and semantic segmentation. It was also one of the earliest attempts at using Transformers for monocular depth estimation. During the writing of this thesis, the training code for DPT was not released. In addition, Ranftl *et al.* [31] describes direct fine-tuning of the model as difficult, since predictions are arbitrarily scaled and shifted with potentially large magnitudes. It would thus be challenging to train DPT for the experiments, so the model was discarded.

GLPDepth

GLPDepth uses a lightweight decoder that fuses the global and local context of an image in order to generate accurate depth predictions. The lightweight decoder makes the architecture well suited for real-time depth estimation. Additionally, the architecture uses the same encoder as SegFormer, which is the chosen segmentation method. This makes it convenient to integrate GLPDepth with the SegFormer architecture in order to create a multitask model. GLPDepth is thus chosen as the monocular depth estimation model of this thesis.

DepthFormer

DepthFormer uses an encoder consisting of two separate branches, one Transformer-based and one CNN-based, in order to capture both global and local information of the image. This approach leads to impressive results on the KITTI dataset. The model is able to outperform GLPDepth, and would thus be interesting to investigate further. However, the official code for DepthFormer was not released until the start of April. By that time the experiments were already planned out, and it would be difficult and time-consuming to incorporate a new model. It was therefore decided to not include DepthFormer in the experiments.

BinsFormer

BinsFormer is currently the state-of-the-art model for monocular depth estimation on the KITTI dataset, and would thus be a natural choice as the depth estimation model of this thesis. Unfortunately, the paper was not released until the start of April, and the official code is still not available at the time of writing this thesis. Consequently, the model could not be used for the experiments.

3.1.3 Decision and Summary

SegFormer and GLPDepth are the chosen models for the experiments. SegFormer will be used for semantic segmentation, while GLPDepth will be used for monocular depth estimation. Both models utilizes the same Transformer backbone, which makes it convenient to combine the models into a multitask model. Additionally, both models use lightweight decoders that are suited for real-time applications. The architectural design of SegFormer and GLPDepth is described in Section 3.3 and 3.4, respectively. Section 3.5 explains how the architectures are combined into a multitask model.

3.2 Choice of Datasets

This section discusses the choice of datasets for this thesis. A total of three publicly available datasets will be chosen, two real-world datasets and one synthetic dataset. To be chosen, the dataset needs to be relatively big and have high quality annotations for both depth and segmentation. For the synthetic datasets the environmental variation will also be taken into consideration. In addition to these selected datasets, the dataset provided by NAPLab will be used for a small experiment. In this section, each of the datasets from Section 2.5 is presented together with an explanation of why the dataset was chosen/not chosen.

3.2.1 Real-World Datasets

Cityscapes

The Cityscapes dataset has a total of 5 000 images with fine annotations for semantic segmentation. The images also have corresponding disparity maps that can be converted to depth maps. The disparity maps are computed from stereo images, which means that they are less accurate than the depth maps generated by a LiDAR sensor. However, there are few public datasets for autonomous driving that provides both accurate LiDAR and semantic segmentation ground truth. Also, the depth maps computed from the stereo images are likely accurate enough to generate meaningful results. Thus, the Cityscapes dataset will be used for the experiments.

KITTI

KITTI is one of the most popular datasets for benchmarking of monocular depth estimation models. The dataset has over 90 000 available images with corresponding LiDAR depth maps. Unfortunately, the dataset only has 400 images with segmentation annotations, which is too few to train a segmentation model. Therefore the dataset will not be used for the main experiments. However, KITTI will be used to validate the accuracy of the chosen depth model GLPDepth, since the original paper provides results for this dataset.

KITTI-360

The KITTI-360 dataset has a lot of similarities with the original KITTI dataset. Both datasets are captured in the same area, and have a large number of images with corresponding depth data. A 64-channel LiDAR sensor was used to capture the depth data. This gives somewhat sparse depth maps, but they can still be used to train and evaluate the models. One key difference between the two datasets is that KITTI-360 provides a total of 61 280 images with corresponding segmentation annotations, while KITTI only provides 400. Because of its large number of segmentation and depth annotations, KITTI-360 is chosen as one of the datasets for the experiments.

Audi Autonomous Driving Dataset (A2D2)

The A2D2 dataset contains 41 277 images with segmentation annotations and corresponding LiDAR point clouds, which can be projected to 2D depth maps. This large amount of annotated data makes A2D2 an interesting candidate for the experiments. Upon closer inspection it was unfortunately discovered that the LiDAR data was collected using a 16-channel LiDAR sensor. This results in very sparse depth maps that only covers a small portion of the images. Consequently, the depth maps would likely not work well for training and evaluation of the models. Based on this knowledge the A2D2 dataset will not be used for the experiments.

NAPLab

The NAPLab dataset is an obvious choice for this thesis, as it is interesting to investigate how the model performs on local datasets. Unfortulately, the dataset was not available until the final week of thesis writing. Additionally, the dataset only contains 10 images with semantic segmentation annotations, which are too few for training and fine-tuning. Thus, the dataset will only be used for inference and evaluation, using a model trained on a different dataset.

3.2.2 Synthetic Datasets

Virtual KITTI

The Virtual KITTI dataset provides high quality photorealistic images with corresponding ground truth data for depth and segmentation. The dataset has a total of 21 000 images, which is quite few compared to the other synthetic datasets. Additionally, the dataset is a virtual recreation of the original KITTI dataset, and will thus have close resemblance to both KITTI and the KITTI-360 dataset, which is captured in the same area. Consequently, the Virtual KITTI dataset is not chosen as the synthetic dataset of this thesis.

SYNTHIA

The SYNTHIA dataset is a really strong candidate for the experiments. With a total of 213 400 images with segmentation and depth annotations captured during different seasons, weather and lighting conditions, the dataset has both the size and environmental variation that is desirable. The dataset was released in 2016, which makes it a little bit old compared to some of the other synthetic datasets. As the field of computer graphics evolves rapidly, it would be reasonable to think that the age of the dataset and the photorealism of the images could be connected. So even though the SYNTHIA dataset is a solid candidate, it will not be chosen for the experiments.

Apollo Synthetic Dataset

The Apollo Synthetic Dataset has a total of 273 000 images with annotations for both semantic segmentation and depth estimation. The images are captured in 7 different virtual environments with varying light, weather and road conditions. In addition the dataset was released in 2019, something that makes it relatively new. The large amount of annotated images and the environmental variation provided makes the Apollo Synthetic Dataset a solid candidate. Therefore the dataset is chosen as the synthetic dataset for the experiments.

All-In-One Drive

Another promising candidate is the All-In-One Drive dataset. With 500 000 images with both segmentation and depth ground truth, the dataset is the largest of the explored synthetic datasets. The dataset also provides environmental variations and rare traffic scenarios. The dataset would thus be interesting to explore in this thesis. Unfortunately, none of the download links on the official website worked at the time of writing this thesis. As there were other good synthetic datasets available, the All-In-One Drive dataset was not chosen.

3.2.3 Decision and Summary

The chosen datasets for the experiments are Cityscapes, KITTI-360 and Apollo Synthetic Dataset. The datasets have 5 000, 61 280 and 273 000 annotated images, respectively. Ground truth for both depth and segmentation is available for all the datasets. In addition to the selected datasets, the NAPLab dataset will be used for a small experiment.

3.3 SegFormer

This section describes how the selected segmentation model SegFormer works. The model uses a common encoder-decoder design. The encoder is a hierarchical Vision Transformer inspired by ViT [2] with multiple improvements for the task of semantic segmentation. The decoder is lightweight and consists of MLP layers exclusively. An overview of the entire architecture can be seen in Figure 3.1.

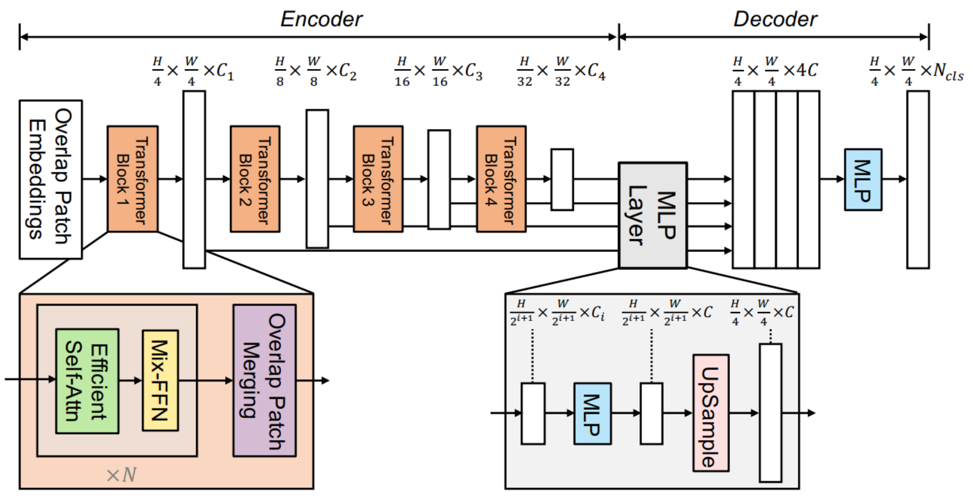

Encoder
Decoder
$\frac{H}{4}\times\frac{W}{4}\times C_1$
$\frac{H}{8}\times\frac{W}{8}\times C_2$
$\frac{H}{16}\times\frac{W}{16}\times C_3$
$\frac{H}{32}\times\frac{W}{32}\times C_4$
$\frac{H}{4}\times\frac{W}{4}\times 4C$
$\frac{H}{4}\times\frac{W}{4}\times N_{cls}$
Overlap Patch Embeddings
Transformer Block 1
Transformer Block 2
Transformer Block 3
Transformer Block 4
MLP Layer
MLP
Efficient Self-Attn
Mix-FFN
Overlap Patch Merging
$\times N$
$\frac{H}{2^{i+1}}\times\frac{W}{2^{i+1}}\times C_i$
$\frac{H}{2^{i+1}}\times\frac{W}{2^{i+1}}\times C$
$\frac{H}{4}\times\frac{W}{4}\times C$
MLP
UpSample

Figure 3.1: The SegFormer architecture. Image taken from Xie *et al.* [21].

3.3.1 Transformer Encoder

Similarly to ViT, the encoder takes images of size $H \times W \times 3$ as input and splits them into patches. Here a patch size of 4×4 is used to better capture finer details of the images. The patches are then sent through four Transformer blocks in order to create hierarchical feature maps. The Transformer blocks are based on the ones used in ViT, but multiple improvements are made. First, the complexity of the multi-head self-attention operation is reduced significantly. This is done by implementing the sequence reduction process proposed by Wang *et al.* [22], which reduces the length of the input sequence with a reduction ratio R. This lowers the computational complexity from $O(N^2)$ to $O(\frac{N^2}{R})$. Next, the positional encoding used in ViT is completely removed. Instead, a 3×3 convolution and a MLP is mixed into each feed-forward network. This approach provides the model with positional information, and also gives better performance for images with varying resolution during inference.

At the end of each Transformer block an overlapped patch merging process is applied. This is done in order to reduce the spatial resolution and produce hierarchical feature maps. The process is inspired by the none-overlapping patch merging process used in ViT, but operates on overlapping patches instead to preserve the local continuity around the patches. The final feature maps have a resolution of $\frac{H}{2^{i+1}} \times \frac{W}{2^{i+1}} \times C_i$, where i refers to the current Transformer block. Finally, the feature maps are sent to the decoder to generate the predicted segmentation mask.

The SegFormer architecture comes with a total of 6 different Transformer encoders, B0 to B5. The encoders have different sizes, but other than that they share the same design. B0 is the smallest model and suited for real-time applications, while B5 is the largest model and provides the highest accuracy.

3.3.2 MLP Decoder

The Segformer decoder is created using only MLP layers. This makes the decoder simple and lightweight compared to a lot of other decoders. However, the decoder is still able to produce high quality predictions. This is made possible by the large effective receptive field of the hierarchical Transformer encoder, which helps the model capture the global context of the image. The decoder goes through 4 main steps to make a prediction:

1. For each feature map F_i, an MLP layer is used to transform the individual channel dimension C_i into a common channel dimension C.
2. The feature maps are up-sampled to a resolution of $\frac{H}{4} \times \frac{W}{4}$, and then concatenated. The concatenated features F have a channel dimension of $4C$.
3. The concatenated features F are fused using a MLP layer, reducing the channel dimension from $4C$ to C.
4. The segmentation mask M is created by sending the fused features trough a new MLP layer. M has a resolution of $\frac{H}{4} \times \frac{W}{4} \times N_{cls}$, where N_{cls} is the number of classes.

3.4 GLPDepth

This section describes how the selected depth estimation model GLPDepth works. The model combines the hierarchical Vision Transformer encoder used in SegFormer with a novel lightweight decoder for the task of monocular depth estimation. Additionally, a depth-specific augmentation technique named Vertical CutDepth is proposed to further increase model performance. An overview of the architecture can be seen in Figure 3.2.

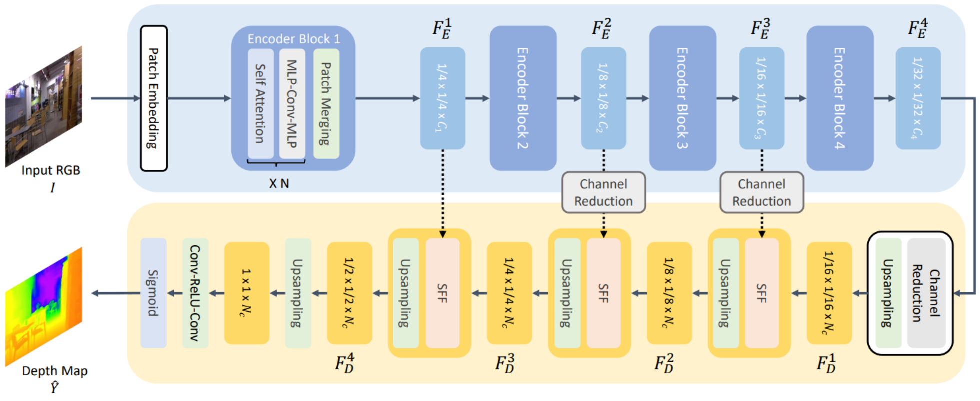


Figure 3.2: The GLPDepth architecture. Image taken from Kim *et al.* [32].

3.4.1 Transformer Encoder

The architecture implements the SegFormer encoder proposed by Xie *et al.* [21], which is described in Section 3.3.1. The encoder produces hierarchical feature maps from the input image that are passed on to the decoder. GLPDepth originally uses the SegFormer B4 backbone for the depth prediction task. However, in the experiments smaller backbone sizes will be tested to see how this affects performance.

3.4.2 Lightweight Decoder

The decoder takes the final feature map F_E^4 from the encoder as input. First, the feature map goes through a channel reduction step and a bilinear upsampling, which transforms the feature map size to $\frac{1}{16}H \times \frac{1}{16}W \times N_C$. Then, the feature map is passed through multiple consecutive layers with Selective Feature Fusion (SFF) and bilinear upsampling. The SFF module is proposed in order to selectively fuse global and local features and create rich feature maps. The module takes the current decoder feature map F_D and the encoder feature map F_E of corresponding size as input. F_E is provided through a skip connection from the encoder, and its channel dimension is reduced to N_C in order to match F_D. The feature maps F_E and F_D are concatenated along the channel dimension and sent through multiple convolutional layers. Then, each channel is multiplied with the corresponding original feature map F_E or F_D provided through a skip connection, and the channels are merged to produce a hybrid feature map H_D. Finally, H_D is upsampled and passed on to the succeeding layer. An illustration of the SFF module can be seen in Figure 3.3.

After multiple consecutive layers of SFF and upsampling, the decoder feature map F_D ends up with a size of $H \times W \times N_C$. In order to produce the final depth prediction, F_D is sent through two convolutional layers and a sigmoid function. The final depth map is then multiplied with the maximum depth value of the current dataset to get the predicted distance in meters.

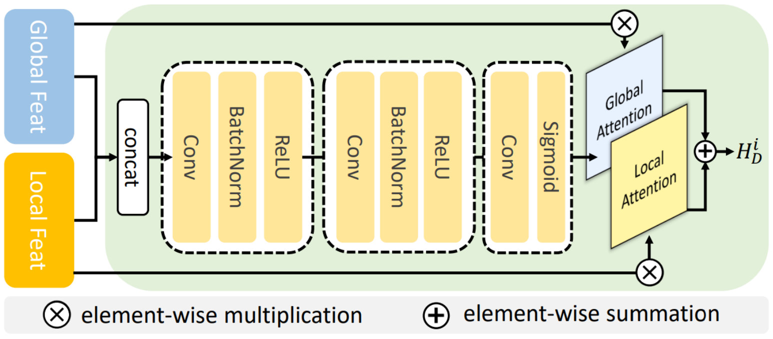


Figure 3.3: The GLPDepth SFF module. Image taken from Kim *et al.* [32].

3.4.3 Vertical CutDepth

GLPDepth proposes a new augmentation method, Vertical CutDepth, specifically designed for depth estimation. The method is inspired by CutDepth [33], which places a random crop of the ground truth depth map into the RGB image during training. This makes the dataset more diverse and leads to better performance. Vertical CutDepth, on the other hand, does not crop the depth map in the vertical direction. The vertical geometric information of the image, which is important for depth estimation, is thus better preserved. Vertical CutDepth is a depth-specific augmentation method, and it is uncertain how it affects other dense prediction tasks. Since the goal of this thesis is to design an architecture for both depth estimation and segmentation, it is decided to not use Vertical CutDepth.

3.5 Multitask Model

This section describes how a multitask model for monocular depth estimation and semantic segmentation is created using the selected models SegFormer and GLP-Depth. The model is able to create predictions for both tasks in a single forward pass.

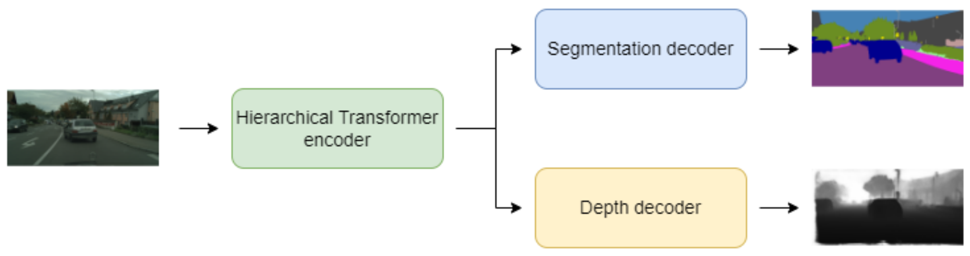


Figure 3.4: The proposed multitask architecture

3.5.1 Architecture

The multitask model uses the SegFormer encoder described in Section 3.3.1 to extract hierarchical features from the input image. The extracted feature maps are then passed on to two separate decoders: one for semantic segmentation and one for monocular depth estimation. The decoders produce the final predictions for their respective tasks. An overview of the proposed architecture can be seen in Figure 3.4. The segmentation and depth decoders are described in Section 3.3.2 and 3.4.2, respectively.

3.5.2 Loss Function

The model is trained for the tasks of semantic segmentation and monocular depth estimation simultaneously. This is done by using two separate task-specific loss functions. For the segmentation task the commonly used cross entropy loss is utilized:

$$L_{CE} = -\sum_{i}^{n} y_i \log(\hat{y}_i) \tag{3.1}$$

Where y_i and $\hat{y}_i$ is the ground truth label and the softmax probability for the i-th class, respectively. For the depth estimation task the scale-invariant log scale loss used in GLPDepth is adopted:

$$L_{SI} = \frac{1}{n}\sum_{i}^{n} d_i^2 - \frac{1}{2n^2}\left(\sum_{i}^{n} d_i^2\right) \tag{3.2}$$

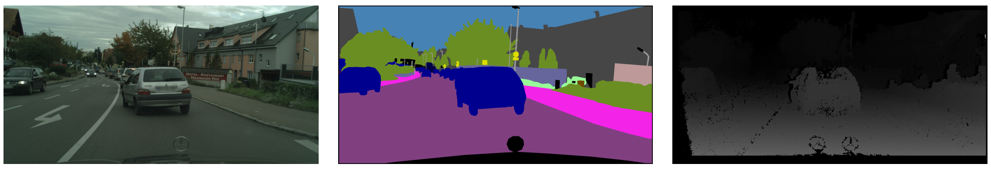
Where $d_i = \log y_i - \log \hat{y}_i$, and y_i and $\hat{y}_i$ is the ground truth and predicted depth value of the i-th pixel, respectively. During training the two loss functions are combined into a single one to find the total loss:

$$L = L_{CE} + L_{SI} \tag{3.3}$$

3.6 Dataset Preparation

This section describes the necessary dataset preparations for the chosen datasets.

3.6.1 Cityscapes



(a) Original **(b)** Segmentation **(c)** Disparity

Figure 3.5: Cityscapes example image with annotations

Download and Extraction

The Cityscapes dataset is available through the official website. The dataset consists of 5 000 frames with fine annotations, which are divided into a training set of 2 975 frames, a validation set of 500 frames and a test set of 1 525 frames. The annotations for the test set are held back for fair benchmarking, so this set will not be used for the experiments. This leaves a total of 3 475 frames for training and evaluation of the models. For each of these frames, RGB images from the left front-facing camera, annotations for semantic segmentation, precomputed disparity maps and intrinsic and extrinsic camera parameters are downloaded. In addition, blurred RGB images are downloaded for visualizations in this thesis.

The standard folder structure for Cityscapes will be used, which is *cityscapes/{type}/{split}/{city}/{filename}*. Here *type* refers to the type of data the folder contains, e.g. *leftImg8bit*, *gtFine*, *disparity* or *camera*. *split* is the dataset split that is contained within the folder, and can be *train* or *val*. *city* refers to the city the data was captured in, e.g. *aachen*, *bochum* or *frankfurt*. Additionally, the root folder contains the files *train.txt* and *val.txt* which contains the paths to the files in the training and validation set, respectively.

Image Size

The images in the Cityscapes dataset have a resolution of 2048x1024. This high resolution makes it infeasible to perform inference in real-time with the chosen models. In order to make inference in real-time possible the images have to be resized to a smaller resolution. Thus, a resolution of 1024x512 will be used in the experiments, which is half of the original image size.

Depth Maps

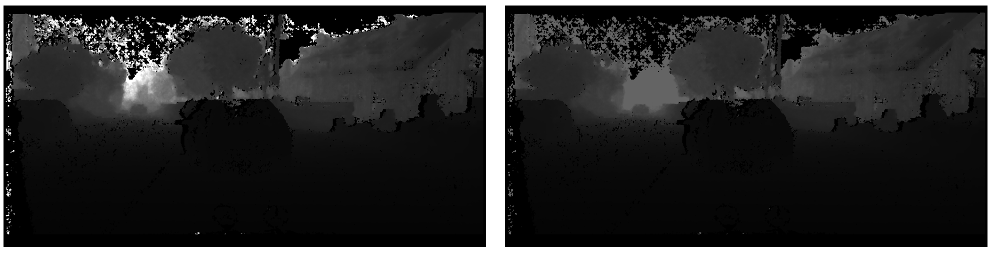
In order to utilize the disparity maps for training they have to be converted to depth maps. This can be done with some simple calculations. The first step is to calculate the disparity values from the raw 16-bit PNG files, which can be done using the following equation:

$$d = \frac{float(p) - 1.0}{256.0} \tag{3.4}$$

Where p is the raw pixel value from the 16-bit PNG file, and d is the calculated disparity value. The next step is to calculate the depth value using the obtained disparity value. For this calculation some of the intrinsic and extrinsic camera parameters is needed. These are found in the parameter file corresponding to the current image, and are located in the *camera* folder. The calculation of the depth values is done using the following equation:

$$D = \frac{B \times f_x}{d} \tag{3.5}$$

Where d is the disparity value, B is the baseline, f_x is the focal length, and D is the calculated depth in meters. The calculated depth maps have values ranging from 0 to approximately 470 meters. It was discovered that high depth values are quite noisy and inaccurate. This can be seen in Figure 3.6a. The depth maps were also tested for training, but the evaluation metrics did not converge as expected. Based on these observations it was decided to cut the depth maps at 100 meters. This means that values larger than 100 meters will be set to 100. The value of 100 meters was chosen since most of the interesting objects in the image appear within this range. The resulting depth map after clipping can be seen in Figure 3.6b.



(a) Depth **(b)** Depth clipped

Figure 3.6: Cityscapes depth map before and after clipping at 100m

Segmentation Annotations

The Cityscapes dataset is labelled with a total of 30 semantic classes. Usually, only 19 of these are used for training and evaluation, since the remaining classes are quite rare. This approach will also be used in this thesis. The 19 commonly used classes with corresponding colors can be seen in Table 3.1. In order to use the segmentation data for training, PNG images with values corresponding to the train IDs of the classes have to be generated. This is done using the official Cityscapes scripts.

Table 3.1: Cityscapes classes and color palette

Train ID	Name	Color
0	Road	🟪
1	Sidewalk	🟪
2	Building	⬛
3	Wall	🟪
4	Fence	🟫
5	Pole	⬜
6	Traffic Light	🟧
7	Traffic Sign	🟨
8	Vegetation	🟩
9	Terrain	🟩
10	Sky	🟦
11	Person	🟥
12	Rider	🟥
13	Car	🟦
14	Truck	⬛
15	Bus	🟦
16	Train	🟦
17	Motorcycle	🟦
18	Bicycle	🟥

3.6.2 KITTI-360

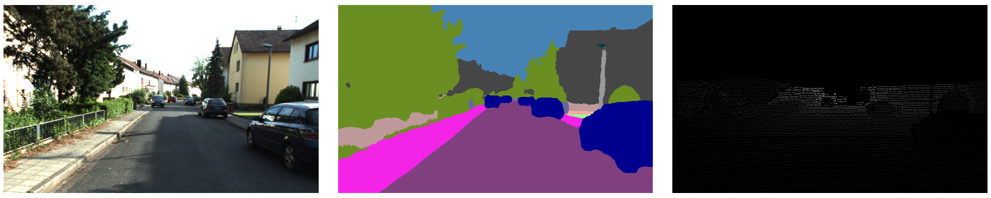

(a) Original **(b)** Segmentation **(c)** Depth

Figure 3.7: KITTI-360 example image with annotations. The images have been cropped for better visualization.

Download and Extraction

The KITTI-360 dataset is available through the official website. The raw perspective images, the 2D semantic labels, the raw LiDAR scans, the vehicle poses, and the extrinsic and intrinsic camera parameters are downloaded. The data is structured into folders with names corresponding to the type of data they contain, e.g. *data_2d_depth*, *data_2d_raw* and *data_2d_semantics*. The dataset contains a total of 61 280 annotated images. These images are split into a train and validation set using the official split. Text files with paths to the images of each split can be found inside the *data_2d_semantics* folder. There are a total of 49 004 frames in the train set and 12 276 frames in the validation set.

Image Size

The images in the KITTI-360 dataset have a resolution of 1408x376. The resolution is quite low, so it is possible to perform inference in real-time without resizing the images. However, the image height is not divisible by 32, which is a requirement due to the architectural design of the chosen models. Thus, the image height needs to be cropped. The cropping is done similar to the approach of Kim *et al.* [32] on the original KITTI dataset by cropping 24 pixels at the top of the image. This gives a final image resolution of 1408x352, which will be used in the experiments. During evaluation the crop proposed by Garg *et al.* [37] will be used.

Depth Maps

The raw LiDAR point clouds are stored in binary format. To utilize them for training they have to be projected to 2D depth maps. This is done by using the official KITTI-360 scripts. First, a transformation matrix from the LiDAR coordinate frame to the camera coordinate frame needs to be applied to all the points. This transformation matrix can be calulated with the following equation:

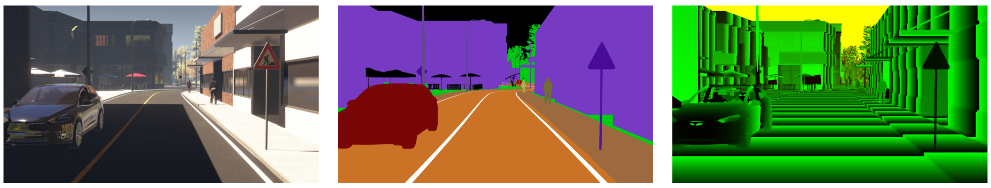
$$T_{L\to k} = T_{0\to k} \times T_{L\to 0} \tag{3.6}$$

Where $T_{L\to 0}$ is the transformation matrix from the LiDAR coordinate frame to the left perspective camera, and $T_{0\to k}$ is the transformation matrix from the left perspective camera to any other camera k. Only the left perspective camera will be used for the experiments, so $T_{0\to k}$ is set to the identity matrix during the calculations. Next, the points in the camera coordinate frame are projected to the image plane using the intrinsic camera parameters. The resulting depth maps have a range of 0 to 80 meters and are stored within the *data_2d_depth* folder.

Segmentation Annotations

The semantic classes for KITTI-360 are consistent with the classes used for Cityscapes (Table 3.1). Thus, the same 19 classes will be used for training and evaluation of KITTI-360 as well. In order to train the models, PNG images containing train IDs for the semantic classes have to be generated. A script for generating these files are created using Python.

3.6.3 Apollo Synthetic Dataset



(a) Original **(b)** Segmentation **(c)** Depth

Figure 3.8: Apollo Synthetic Dataset example image with annotations

Download and Extraction

The Apollo Synthetic Dataset was downloaded from the official website. The dataset has a total of 273 000 frames. It was decided to create a subset of the dataset for the experiments in order to achieve a better class and scene balance. The following list shows the folder structure for the dataset in hierarchical order from top to bottom. All the available frames for the current folder are used unless other is specified:

- ***type*** refers to the type of data contained within the folder: *Depth*, *RGB* or *Segmentation*.
- ***time_of_day*** refers to the time of day the data was captured, e.g. *00-00*, *09-00* or *13-00*.
- ***weather*** refers to the weather conditions for the data: *CLEAR_SKY*, *LIGHT_RAIN* or *HEAVY_RAIN*. Only the *CLEAR_SKY* frames will be used for the experiments.
- ***degradation*** refers to the degree of road surface damage: *NO_DEGRADATION*, *DEGRADATION* or *SEVERE_DEGRADATION*.
- ***pedestrian*** refers to whether or not there are pedestrians present in the frames: *With_Pedestrian* or *Without_Pedestrian*. Only the frames with pedestrians present will be used. In addition, the highway frames will be included, which do not contain pedestrians.
- ***traffic_barrier*** refers to whether or not traffic barriers are present in the frames: *With_TrafficBarrier* or *Witout_TrafficBarrier*.
- ***area*** refers to the virtual scene where the data was captured, e.g. *Downtown*, *Residential* or *Highway*. The frames from the indoor parking garage will not be used.
- ***sequence*** refers to the traffic sequence the data belongs to, e.g. *Traffic_066*. The highway scene has a large number of frames compared to the other

scenes. In order to avoid imbalance only half of the available highway sequences will be used.

In addition, 125 frames had to be excluded because of missing depth annotations. This leaves a total of 45 235 frames for the experiments. These frames have to be split into a training and validation set. In order to avoid validating on data that are too similar to the training data, the dataset was split on virtual scene level. The *Road_Loop_with_Intersections* scene will be used for validation, while the remaining scenes will be used for training. The resulting training set has a total of 40 195 frames and the validation set has 5 040 frames.

Image Size

The images in the Apollo Synthetic Dataset have a resolution of 1920x1080. Just like for Cityscapes, the resolution has to be reduced in order to make inference in real-time possible. A logical approach would be to resize the images to 960x540, which is half the size of the original images. However, due to the architectural design of the chosen models, the image size has to be divisible by 32. Consequently, an image size of 960x544 is selected instead.

Depth Maps

The depth data is encoded into 16-bit PNG files. To decode the depth values the following equation is used:

$$D = \left(R + \frac{G}{255.0}\right) \times 655.36 \tag{3.7}$$

Where D is the calculated depth value in meters, and R and G are the normalized float values of the pixel's red and green channels, respectively. This produces depth values in the range of 0 to 655.35 meters with 1 cm precision. As the depth values are highly accurate, it would be totally possible to use the full depth range during training and evaluation. However, since a much shorter depth range is used for Cityscapes and KITTI-360, it is decided to clip the depth values for Apollo Synthetic Dataset as well. The high accuracy of the depth maps makes it possible to use a larger max depth value here than for the Cityscapes dataset. The max value is thus set to 200 meters.

Segmentation Annotations

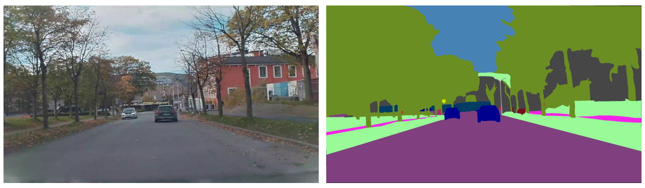
The Apollo Synthetic Dataset has a total of 24 classes for semantic segmentation. These classes are modified in order to better match the ones used in Cityscapes. Some of the classes are merged or ignored, while others are kept as they are. The result is a total of 14 classes that will be used for the experiments. These classes, together with the corresponding original classes, can be seen in Table 3.2. The color palette corresponds to the one used for Cityscapes. Finally, PNG images with train IDs have to be generated. Since the Apollo Synthetic Dataset does not provide a script for generating this, a new one is made using Python.

Table 3.2: Apollo Synthetic Dataset classes and color palette

Train ID	Class Name	Original Classes	Color
0	Road	Road	🟪
1	Sidewalk	Sidewalk	🟪
2	Building	Building	⬛
3	Pole	Pole, Street Light	⬜
4	Traffic Light	Traffic Light	🟧
5	Traffic Sign	Traffic Sign	🟨
6	Vegetation	Vegetation	🟩
7	Terrain	Terrain	🟩
8	Person	Pedestrian	🟥
9	Car	Coupe, SUV, Hatchback, Van	🟦
10	Truck	Truck, Pickup Truck	⬛
11	Bus	Bus	🟦
12	Motorcycle	Motorcyclist	🟦
13	Bicycle	Cyclist	🟥

3.6.4 NAPLab

The NAPLab dataset consists of 10 frames with semantic segmentation annotations only. The images have a resolution of 1920 × 1080, and are thus resized to 960 × 544 to make real-time inference possible. The dataset uses the same class definition as Cityscapes for semantic segmentation (Table 3.1). An example frame from the dataset with corresponding segmentation annotation can be seen in Figure 3.9.



(a) Original **(b)** Segmentation

Figure 3.9: NAPLab example image with segmentation annotation

3.7 Hardware

Powerful computational resources were needed to conduct the experiments of this thesis. Two different solutions were used: The Idun cluster [38] and a virtual machine provided by NTNU. Idun is a large computing cluster at NTNU that provides computational resources for research purposes. The cluster provides several A100, V100 and P100 GPUs that were used to train the models. This made it possible to train across multiple GPUs, something that was necessary due to the large memory consumption of the models. The virtual machine provided by NTNU was used for development, benchmarking and inference. The machine is equipped with an A10 virtual GPU with 23 GB RAM. It also provides a graphical user interface, which is convenient for development and visualization of results.

3.8 Training

The multitask model is trained on the three selected datasets. The number of training epochs are selected according to the dataset size: 300 epochs for Cityscapes, 30 epochs for Apollo Synthetic Dataset, and 20 epochs for KITTI-360. During training the Adam optimizer [3] is used with a learning rate of 1.0×10^{-4}. The batch size is set to 12. Augmentations are implemented using the Albumentations library [39], and the following augmentations are applied during training: *HorizontalFlip*, *RandomBrightnessContrast*, *RandomGamma* and *HueSaturationValue*. The models are initialized with ImageNet [26] pre-trained weights.

Chapter 4

Experiments and Results

This chapter presents the results from the experiments of this thesis. In order to answer the research questions, the following experiments were conducted:

Experiment 0 - Validating Results: The chosen models are trained and evaluated to verify that the results from the official papers are reproducible.

Experiment 1 - Multitask Training: The multitask training approach is compared to individual task training for segmentation and depth to investigate how it affects model performance.

Experiment 2 - Different Backbone Sizes: The multitask model is trained with different backbone sizes to investigate how choice of backbone affects performance.

Experiment 3 - Pre-training on Synthetic Data: The multitask model is pre-trained on a synthetic dataset in an attempt to increase the model accuracy.

Experiment 4 - Validating the Predicted Depth Values: The predicted distance to individual objects in the images are evaluated to investigate how accurately the model predicts depth.

Experiment 5 - Evaluating on NAPLab data: The multitask model trained on Cityscapes is evaluated on the NAPLab data.

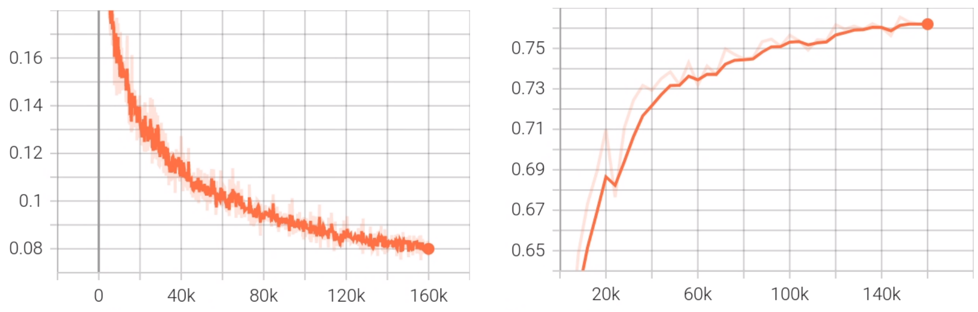
4.1 Experiment 0: Validating Results

This section presents the results from experiment 0. This is not really an official experiment, but more of an "warm-up" for the other experiments. The selected models SegFormer and GLPDepth are trained and evaluated on Cityscapes and KITTI using the official code implementations for the models. The purpose of the experiment is to validate the results reported in the official papers, and investigate how the training graphs converge during training.

4.1.1 SegFormer

Training

The SegFormer-B0 model was trained on the Cityscapes dataset for 160 000 iterations following the training strategy of Xie *et al.* [21]. Graphs for the training loss and validation mIoU during training can be seen in Figure 4.1. Unfortunately, the validation loss was not logged during training, so it is not included here. Both the training loss and mIoU converges smoothly throughout the entire training process. The graphs can thus be used as an ideal representation of how these values should evolve during training.



(a) Training Loss **(b)** Validation mIoU

Figure 4.1: Experiment 0: Cityscapes training graphs

Results

The evaluation metrics for the final model can be seen in Table 4.1. The achieved mIoU closely matches the one reported in the official paper (76.2 mIoU), and proves that the results are reproducible. The qualitative results in Figure 4.2 shows that the model is able to produce good and accurate segmentation predictions. When the inference speed is tested the model is able to work at 8 FPS, which is too slow to be able to perform in real-time. The reason for this is that inference is done on the full-sizes images with a resolution of 2048x1024, so the image processing becomes too computationally expensive. Consequently, the remaining experiments will use a smaller image size of 1024x512 for the Cityscapes dataset to make real-time segmentation achievable.

Table 4.1: Experiment 0: Cityscapes evaluation metrics

Model	FPS	mIoU	mAcc	aAcc
SegFormer-B0	8	76.19	83.81	95.89

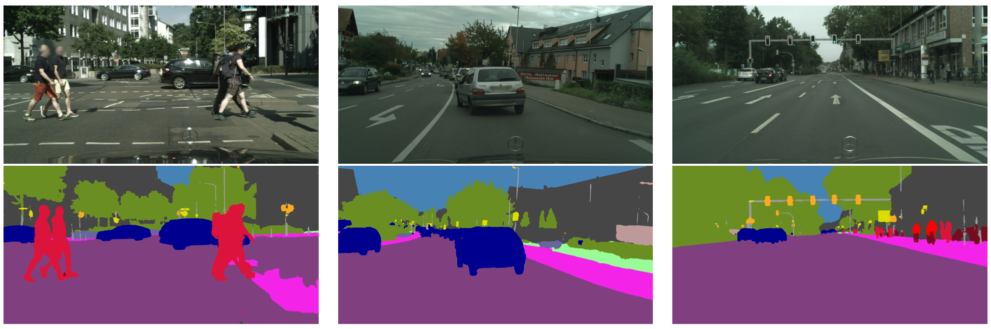
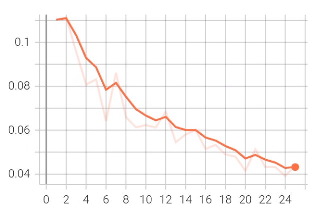


Figure 4.2: Experiment 0: Cityscapes qualitative results. First row: original image, second row: predicted segmentation mask.

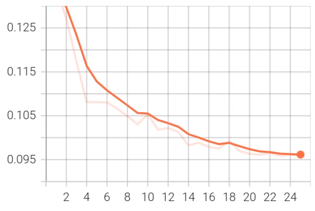
4.1.2 GLPDepth

Training

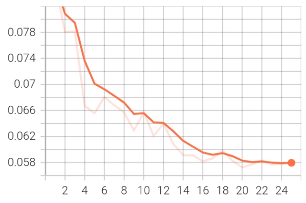
GLPDepth was trained on the KITTI dataset for a total of 25 epochs following the training strategy of Kim *et al.* [32]. Figure 4.3 shows the development of the training loss, validation loss and absolute relative error during training. All values converge smoothly, and the graphs thus give an indication of what an optimal training procedure should look like.



(a) Training Loss



(b) Validation Loss



(c) AbsRel

Figure 4.3: Experiment 0: KITTI training graphs

Results

The evaluation metrics for the final model can be seen in Table 4.2. The metrics are very similar to the ones reported in the official paper (0.057 AbsRel), and indicates that the results are reproducible. Figure 4.4 shows qualitative results for the model. In terms of inference speed the model is able to work at 19 FPS. This is faster than the segmentation model due to the low resolution of the KITTI images. However, GLPDepth is still not able to perform depth estimation in real-time, mostly because of the large and complex backbone that is used.

Table 4.2: Experiment 0: KITTI evaluation metrics

Model	FPS	AbsRel	SqRel	RMSE	RMSElog	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
GLPDepth	19	0.058	0.189	2.321	0.087	0.966	0.996	0.999

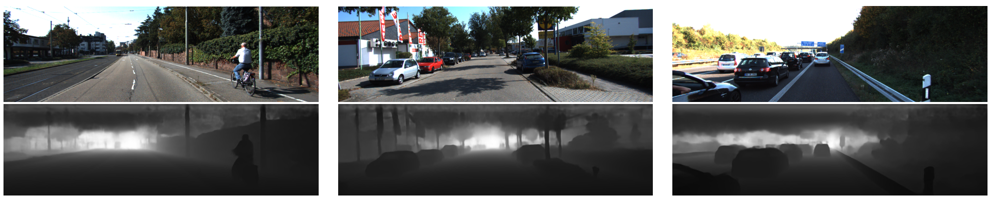


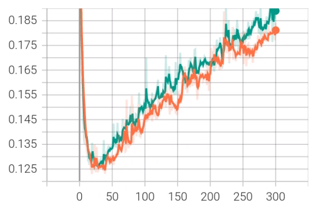
Figure 4.4: Experiment 0: KITTI qualitative results. First row: original image, second row: predicted depth map.

4.2 Experiment 1: Multitask Training

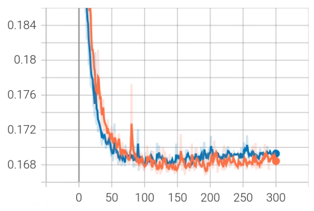
This section presents the results from experiment 1. The experiment investigates how a multitask training approach, where the model is trained for both semantic segmentation and monocular depth estimation simultaneously, affects performance. To investigate this, three different models are trained: one that performs segmentation only, one that performs depth estimation only, and a multitask model that performs both tasks. The models are trained and evaluated on the three selected datasets: Cityscapes, KITTI-360 and Apollo Synthetic Dataset. The SegFormer B2 backbone is used for all the models.

4.2.1 Training

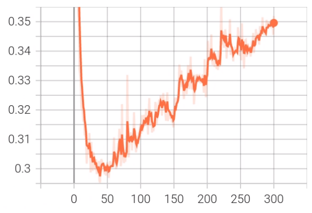
The models were trained according to the specifications in Section 3.8. The training graphs for Cityscapes, KITTI-360 and Apollo Synthetic Dataset can be seen in Figure 4.5, 4.6 and 4.7, respectively. The segmentation loss starts increasing after the first few epochs of training for all the datasets, which is a common sign of overfitting. However, the mIoU continues to improve even though the segmentation loss is increasing. The depth loss has a smoother convergence than the segmentation loss throughout the training process. The individual task models and the multitask model generally converge quite similarly, and no significant differences can be seen in the training graphs.



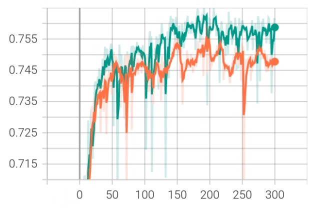
(a) Segmentation loss



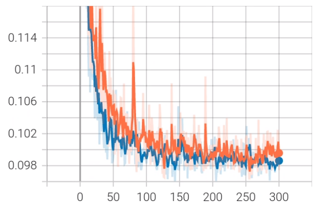
(b) Depth loss



(c) Total loss

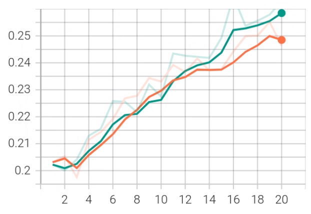


(d) mIoU

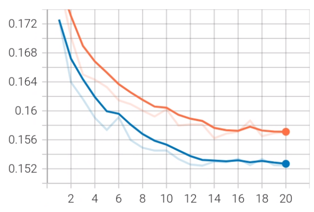


(e) AbsRel

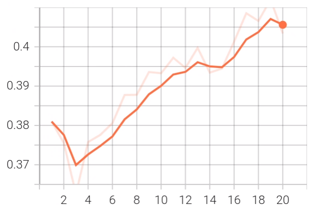
Figure 4.5: Experiment 1: Cityscapes training graphs. The graphs show the validation loss and evaluation metrics during training. Orange: multitask, blue: depth, green: segmentation.



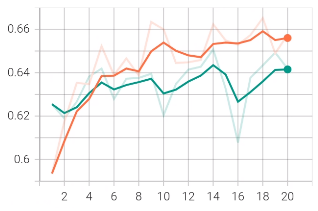
(a) Segmentation loss



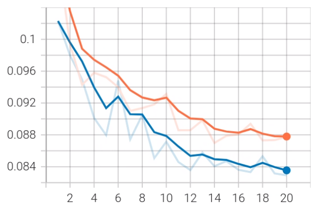
(b) Depth loss



(c) Total loss

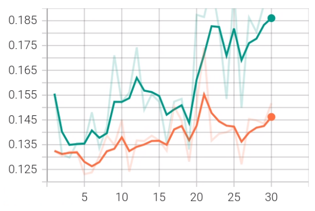


(d) mIoU

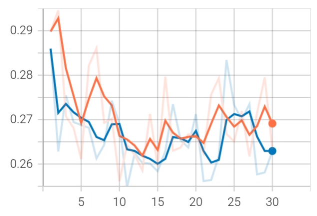


(e) AbsRel

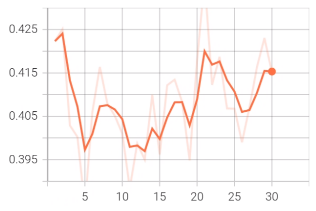
Figure 4.6: Experiment 1: KITTI-360 training graphs. The graphs show the validation loss and evaluation metrics during training. Orange: multitask, blue: depth, green: segmentation.



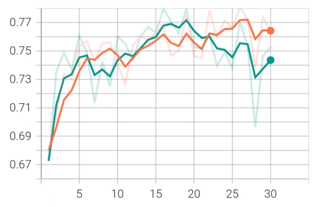
(a) Segmentation loss



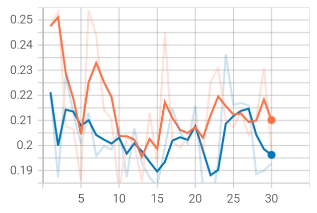
(b) Depth loss



(c) Total loss



(d) mIoU



(e) AbsRel

Figure 4.7: Experiment 1: Apollo Synthetic Dataset training graphs. The graphs show the validation loss and evaluation metrics during training. Orange: multitask, blue: depth, green: segmentation.

4.2.2 Quantitative results

The quantitative results can be seen in Table 4.3. For depth, the individual task model performs equal to or better than the multitask model across all datasets in terms of absolute relative error. However, the differences are quite small. For segmentation, the individual task model performs the best on Cityscapes, while the multitask model performs the best on the two other datasets. The inference speed is similar across all the datasets since the images have roughly the same number of pixels that need to be processed. The multitask model has comparable inference speed with the segmentation model, even though it performs an additional task.

Table 4.3: Experiment 1: quantitative results

						Cityscapes					
Model	FPS	AbsRel	SqRel	RMSE	RMSElog	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$	mIoU	mAcc	aAcc
Depth only	30	**0.096**	**1.213**	**6.289**	**0.162**	0.904	**0.977**	0.991	-	-	-
Seg only	21	-	-	-	-	-	-	-	**76.53**	**83.98**	**95.87**
Multitask	18	**0.096**	1.266	6.317	**0.162**	**0.905**	**0.977**	**0.992**	75.93	83.54	95.80
						KITTI-360					
Depth only	32	**0.083**	**0.372**	**2.905**	**0.147**	**0.912**	**0.972**	**0.989**	-	-	-
Seg only	21	-	-	-	-	-	-	-	65.07	73.33	**93.39**
Multitask	19	0.087	0.386	2.938	0.150	0.907	**0.972**	**0.989**	**66.52**	**74.23**	93.37
						Apollo Synthetic Dataset					
Depth only	30	**0.173**	6.993	18.550	**0.275**	**0.792**	**0.902**	**0.954**	-	-	-
Seg only	21	-	-	-	-	-	-	-	77.98	**85.23**	94.55
Multitask	18	0.181	**6.047**	**17.190**	0.277	0.783	0.897	**0.954**	**78.09**	84.59	**94.88**

4.2.3 Qualitative results

The qualitative results for Cityscapes, KITTI-360 and Apollo Synthetic Dataset can be seen in Figure 4.8, 4.9 and 4.10, respectively. No significant differences were observed between the predictions of the multitask model and the individual task models, so only the predictions of the multitask model is included here. The model is able to create convincing predictions across all the datasets. For Apollo Synthetic Dataset the model struggles to segment the sky region, since this area is not labeled in the ground truth annotations. The model is also unable to create accurate depth predictions for the top part of the KITTI-360 images, since the ground truth depth maps only cover the bottom part of the image.

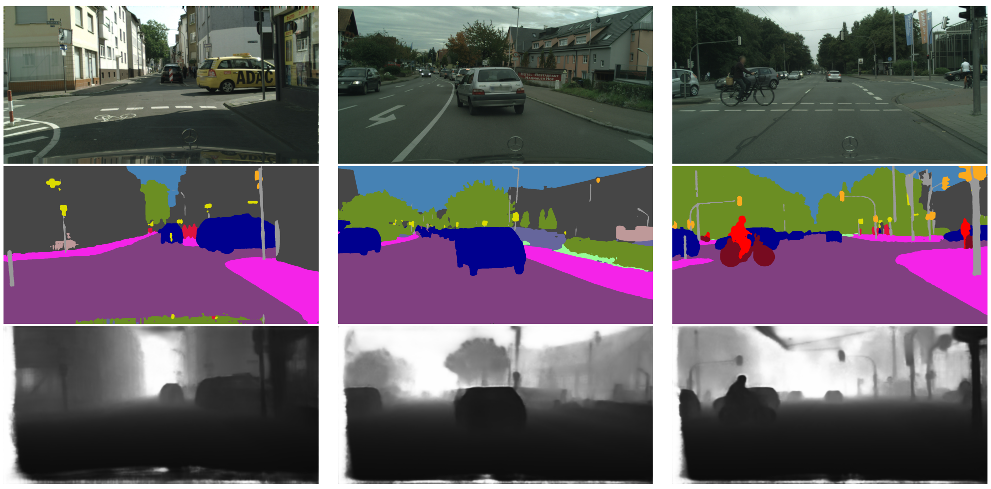


Figure 4.8: Experiment 1: Cityscapes qualitative results. The results are generated using the B2 multitask model. First row: original image, second row: predicted segmentation mask, third row: predicted depth map.

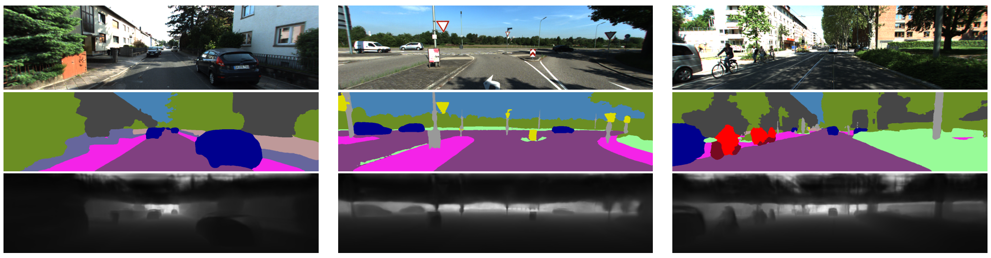


Figure 4.9: Experiment 1: KITTI-360 qualitative results. The results are generated using the B2 multitask model. First row: original image, second row: predicted segmentation mask, third row: predicted depth map.

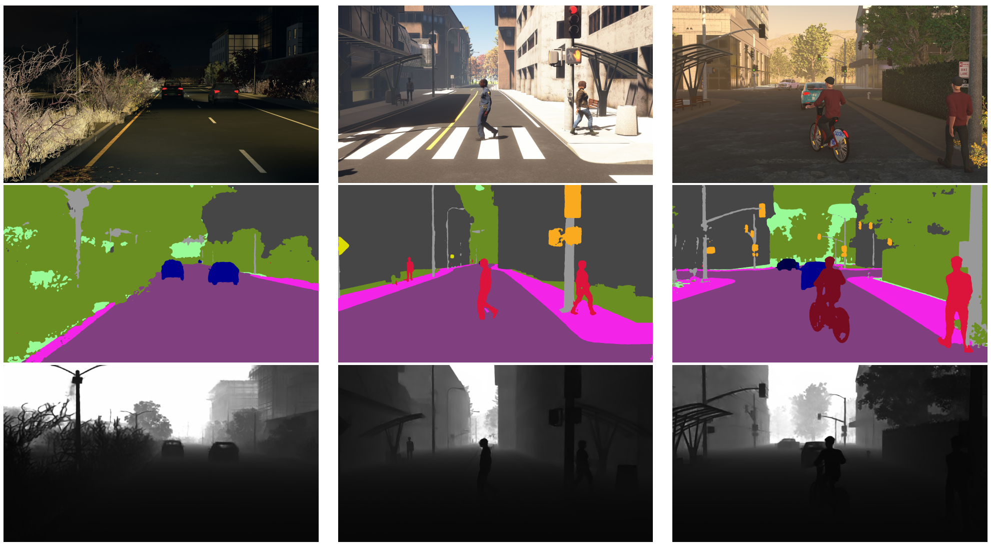


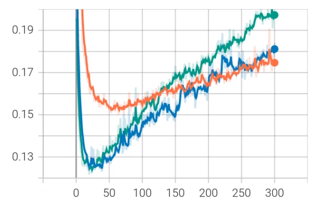
Figure 4.10: Experiment 1: Apollo Synthetic Dataset qualitative results. The results are generated using the B2 multitask model. First row: original image, second row: predicted segmentation mask, third row: predicted depth map.

4.3 Experiment 2: Different Backbone Sizes

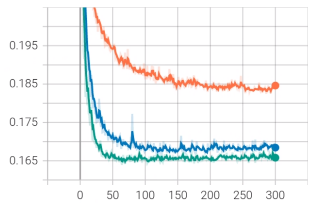
This section presents the results from experiment 2. The experiment investigates how the size of the Transformer backbone affects the model performance. Three different SegFormer backbone sizes, B0, B2 and B4, are combined with the multitask model. The different sized models are then trained and evaluated on the selected datasets: Cityscapes, KITTI-360 and Apollo Synthetic Dataset.

4.3.1 Training graphs

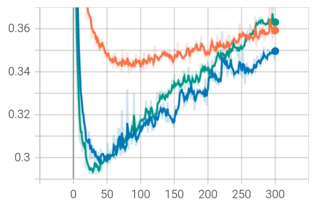
The models were trained according to the specifications in Section 3.8. The training graphs for Cityscapes, KITTI-360 and Apollo Synthetic Dataset can be seen in Figure 4.11, 4.12 and 4.13, respectively. Just like in experiment 1, the segmentation loss starts increasing after the first few epochs of training, indicating overfitting. The problem occurs for all backbone sizes, but is less prominent for the B0 backbone which has a slower convergence rate. The depth loss has a better convergence and is able to improve for a longer period of time. The accuracy graphs are generally as expected, with the B4 backbone performing the best, B2 the second best and B0 the worst.



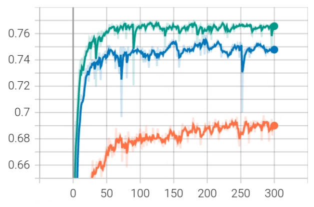
(a) Segmentation loss



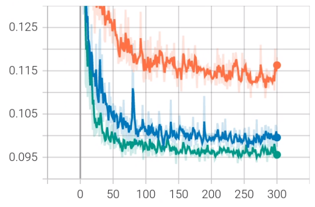
(b) Depth loss



(c) Total loss

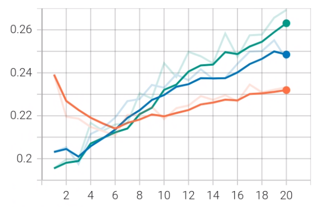


(d) mIoU

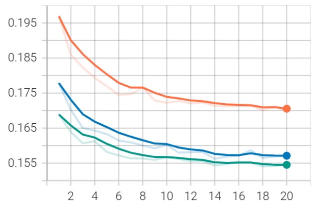


(e) AbsRel

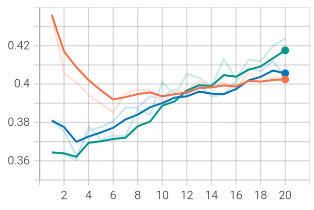
Figure 4.11: Experiment 2: Cityscapes training graphs. The graphs show the validation loss and evaluation metrics during training. Orange: B0, blue: B2, green: B4.



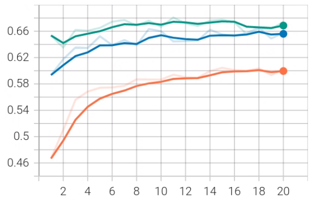
(a) Segmentation loss



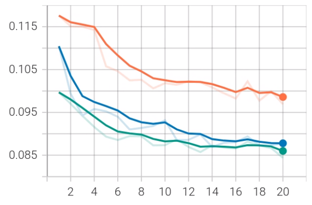
(b) Depth loss



(c) Total loss

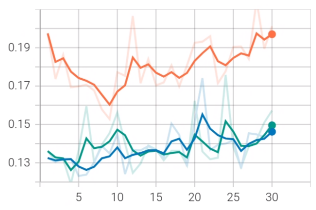


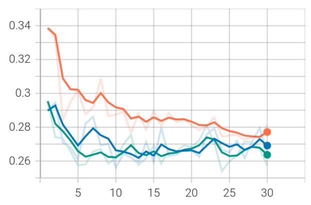
(d) mIoU

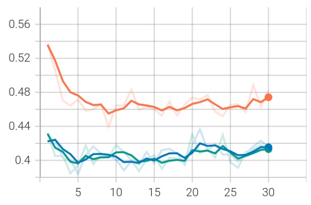


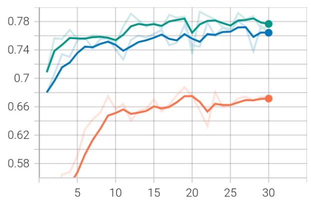
(e) AbsRel

Figure 4.12: Experiment 2: KITTI-360 training graphs. The graphs show the validation loss and evaluation metrics during training. Orange: B0, blue: B2, green: B4.


(a) Segmentation loss


(b) Depth loss


(c) Total loss


(d) mIoU

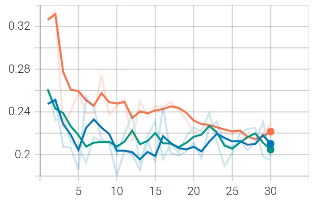

(e) AbsRel

Figure 4.13: Experiment 2: Apollo Synthetic Dataset training graphs. The graphs show the validation loss and evaluation metrics during training. Orange: B0, blue: B2, green: B4.

4.3.2 Quantitative results

The quantitative results can be seen in Table 4.4. As expected, the model with the largest backbone generally achieves the highest accuracy for both tasks. The only exception is depth estimation on the Apollo Synthetic Dataset, where the B2 model is able to outperform the B4 model. In terms of inference speed the lightweight B0 model is able to work at an impressive 56-61 FPS, which is in the real-time area by a large margin. The B2 model works at close to real-time with 18-19 FPS, while the heaviest B4 model works at 11-12 FPS.

Table 4.4: Experiment 2: quantitative results

					Cityscapes						
Backbone	FPS	AbsRel	SqRel	RMSE	RMSElog	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$	mIoU	mAcc	aAcc
B0	57	0.109	1.435	6.999	0.177	0.876	0.969	0.990	69.78	78.34	94.85
B2	18	0.096	1.266	6.317	0.162	0.905	0.977	**0.992**	75.93	83.54	95.80
B4	12	**0.093**	**1.263**	**6.251**	**0.160**	**0.909**	**0.978**	**0.992**	**76.94**	**84.83**	**95.94**
					KITTI-360						
B0	61	0.097	0.440	3.172	0.163	0.889	0.967	0.987	60.44	69.27	92.69
B2	19	0.087	0.386	2.938	0.150	0.907	**0.972**	**0.989**	66.52	74.23	93.37
B4	12	**0.084**	**0.377**	**2.925**	**0.149**	**0.909**	**0.972**	**0.989**	**68.08**	**75.75**	**93.58**
					Apollo Synthetic Dataset						
B0	56	0.208	7.096	17.750	0.298	0.761	0.881	0.945	68.76	77.16	92.93
B2	18	**0.181**	**6.047**	**17.190**	**0.277**	**0.783**	**0.897**	0.954	78.09	84.59	**94.88**
B4	11	0.190	6.960	18.390	0.278	0.775	0.896	**0.955**	**79.37**	**85.80**	94.69

4.3.3 Qualitative results

The qualitative results for segmentation and depth can be seen in Figure 4.14 and 4.15, respectively. For semantic segmentation, the two largest models are better at distinguishing between the driveable road surface and the sidewalk. Additionally, they are better at segmenting small objects far away from the camera and thin structures, such as poles, correctly. For depth estimation, the largest models are able to produce sharper depth maps than the smallest one. Small objects in the distance and thin structures are also better detected by the largest models.

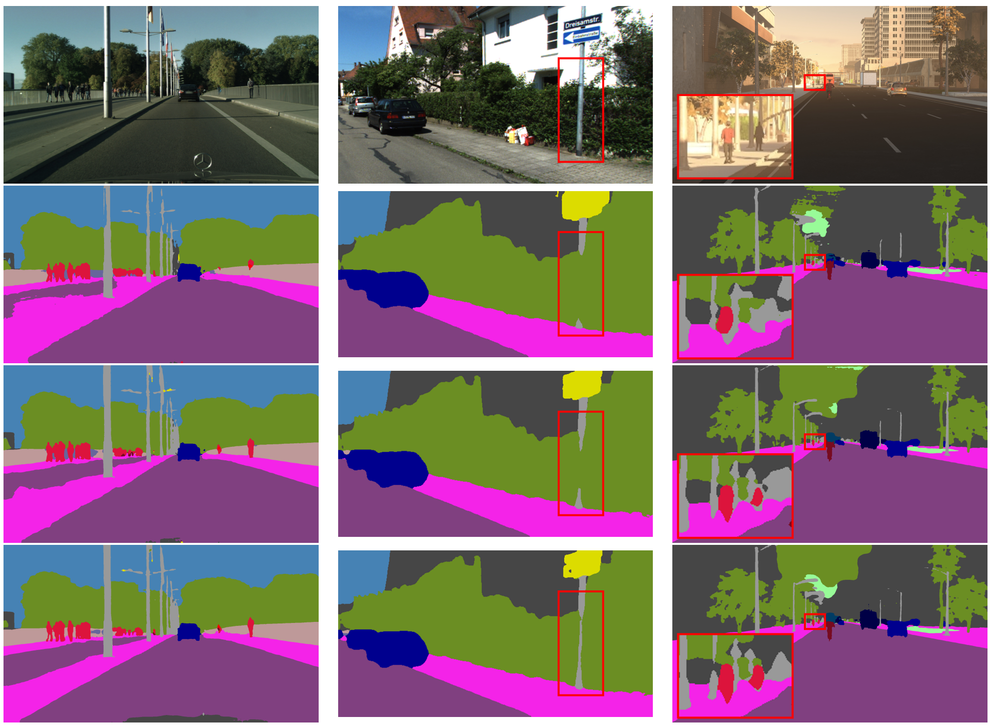


Figure 4.14: Experiment 2: Qualitative results for the segmentation task. The images have been cropped for better visualization. First column: Cityscapes, second column: KITTI-360, third column: Apollo Synthetic Dataset. First row: original image, second row: B0, third row: B2, fourth row: B4.

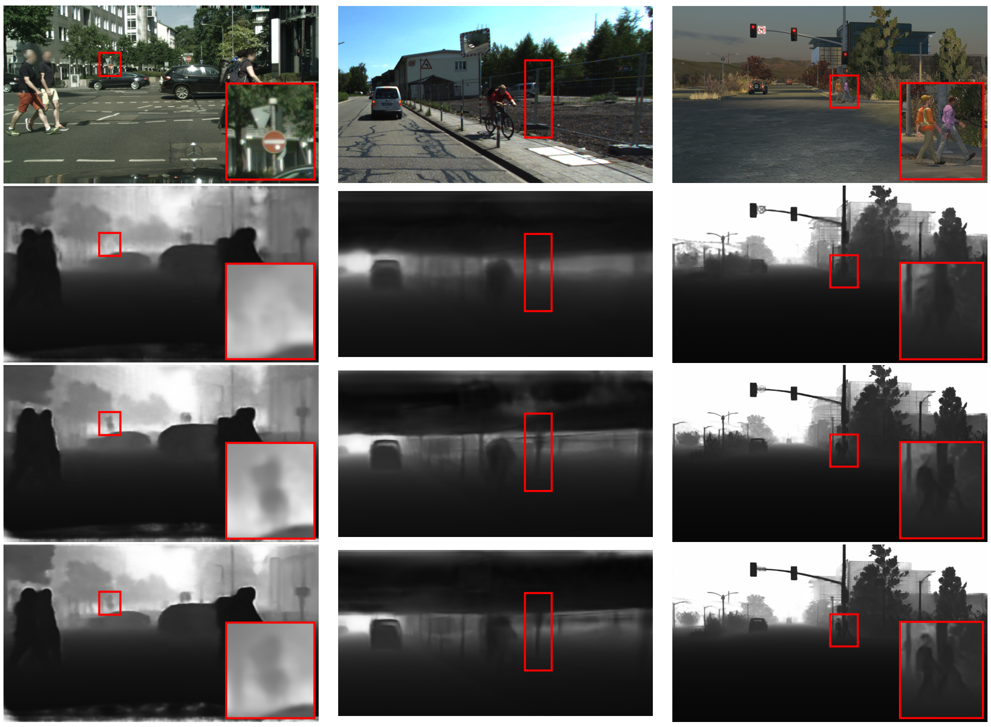


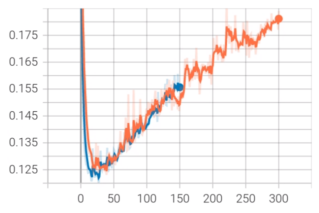
Figure 4.15: Experiment 2: qualitative results for the depth estimation task. The images have been cropped for better visualization. First column: Cityscapes, second column: KITTI-360, third column: Apollo Synthetic Dataset. First row: original image, second row: B0, third row: B2, fourth row: B4.

4.4 Experiment 3: Pre-training on Synthetic Data

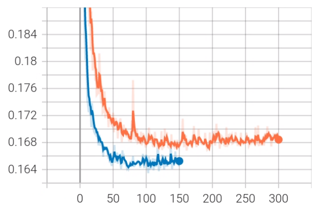
This section presents the results from experiment 3. The experiment investigates the effect of pre-training on a large synthetic dataset. The multitask model equipped with a SegFormer B2 backbone is pre-trained on the Apollo Synthetic Dataset, and then fine-tuned on Cityscapes. The performance of the model is compared to one trained only on Cityscapes.

4.4.1 Training

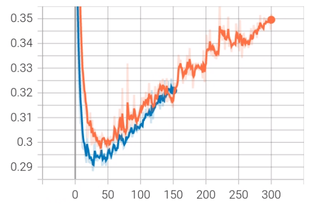
The models were trained according to the specifications in Section 3.8. The training graphs can be seen in Figure 4.16. For depth estimation, both the loss and the absolute relative error is improved by pre-training on the synthetic dataset. For semantic segmentation, no improvement is observed. The model pre-trained on synthetic data was unfortunately stopped after 150 epochs, so the model could potentially perform even better if it was trained for a longer period. However, since the improvement during the last phase of training usually is quite small, the difference would likely be minimal.



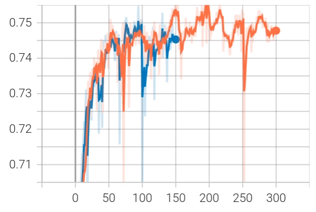
(a) Segmentation loss



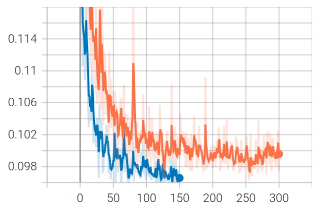
(b) Depth loss



(c) Total loss



(d) mIoU



(e) AbsRel

Figure 4.16: Experiment 3: Training graphs with and without pre-training on synthetic data. The graphs show the validation loss and evaluation metrics during training. Orange: without pre-training, blue: with pre-training.

4.4.2 Results

The quantitative results are presented in Table 4.5. The model pre-trained on synthetic data outperforms the original model across all depth evaluation metrics. For semantic segmentation, there is no improvement by pre-training on the synthetic dataset. No significant differences were observed when comparing the predictions of the models with and without synthetic dataset pre-training. Consequently, no qualitative results are provided for this experiment.

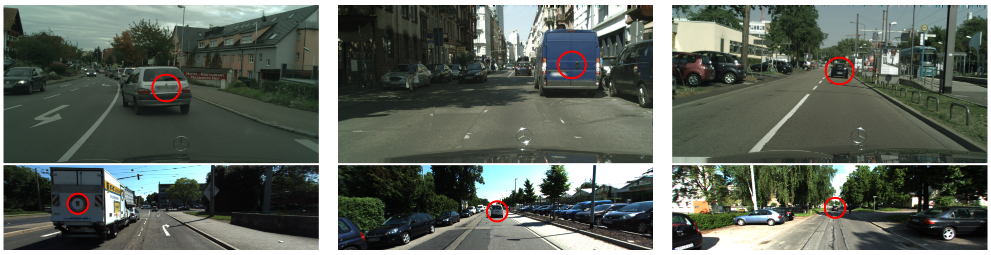
Table 4.5: Experiment 3: quantitative results

					Cityscapes					
Pre-trained	AbsRel	SqRel	RMSE	RMSElog	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$	mIoU	mAcc	aAcc
No	0.096	1.266	6.317	0.162	0.905	0.977	**0.992**	**75.93**	83.54	95.80
Yes	**0.094**	**1.185**	**6.227**	**0.159**	**0.908**	**0.978**	**0.992**	75.48	**83.64**	**95.83**

4.5 Experiment 4: Validating the Predicted Depth Values

This section presents the results from experiment 4. The experiment investigates how accurately the model is able to estimate the distance to different objects in an image. A total of six images are selected, three from Cityscapes and three from KITTI-360. These can be seen in Figure 4.17. The multitask model creates a depth prediction for each of the images. Then, the predicted depth of an object in each image, marked with a red circle, is compared to the ground truth depth. An average of multiple depth values in the same region is used. Two different versions of the multitask model is used: one trained on Cityscapes, and one trained on

KITTI-360. This makes it possible to investigate how accurate the model performs on the dataset it was trained on, and also how accurate it performs on new and unseen data.



(a) Image 1 **(b)** Image 2 **(c)** Image 3

Figure 4.17: Experiment 4: depth validation images. The predicted distance to an object in each image is validated (see Table 4.6). The chosen objects are marked with a red circle. First row: Cityscapes, second row: KITTI-360.

4.5.1 Results

The results can be seen in Table 4.6. As expected, the predicted depth is most accurate for the dataset the model was trained on. This is true for both datasets, and the prediction error is often less than a meter. However, when the model is tested on a different dataset than the one used for training, it fails to estimate depth accurately, and the prediction error is large. It seems like the Cityscapes model overestimates the distance on the KITTI-360 dataset, while the KITTI-360 model underestimates the distance on the Cityscapes dataset.

Table 4.6: Experiment 4: depth validation results. The predicted distance to individual objects in images from Cityscapes and KITTI-360 is evaluated (Figure 4.17). Two different models are used: one trained on Cityscapes and one trained on KITTI-360. The values are in meters.

		Cityscapes		
Trained on	Image	Ground truth	Prediction	Error
Cityscapes	1	9.26	9.50	0.24
	2	12.31	11.67	0.64
	3	35.09	36.40	1.31
KITTI-360	1	9.26	6.47	2.79
	2	12.31	6.64	5.67
	3	35.09	18.45	16.64
		KITTI-360		
KITTI-360	1	4.83	4.31	0.52
	2	17.13	16.57	0.57
	3	34.14	36.75	2.61
Cityscapes	1	4.83	9.31	4.48
	2	17.13	26.90	9.76
	3	34.14	67.89	33.76

4.6 Experiment 5: Evaluating on NAPLab Data

This section presents the results from experiment 4. The experiment investigates how the multitask model performs on the NAPLab dataset. As the dataset only contains 10 annotated frames, it is not possible to train or fine-tune the model on the dataset. Instead, a model trained on the Cityscapes dataset is used for evaluation. Three different model sizes, B0, B2 and B4, are tested, just like in experiment 2.

4.6.1 Results

The quantitative results can be seen in Table 4.7. The dataset does not provide any ground truth depth data, so only evaluation metrics for semantic segmentation are presented. The largest model achieves the highest mIoU score, while the smallest model achieves the lowest score, similar to the results of experiment 2. Qualitative results are provided in Figure 4.18. The predicted segmentation masks look fairly similar to the ground truth masks.

Table 4.7: Experiment 5: NAPLab quantitative results

Backbone	FPS	mIoU	mAcc	aAcc
B0	56	50.49	66.49	92.94
B2	18	55.36	**74.12**	94.15
B4	11	**59.97**	73.73	**94.36**

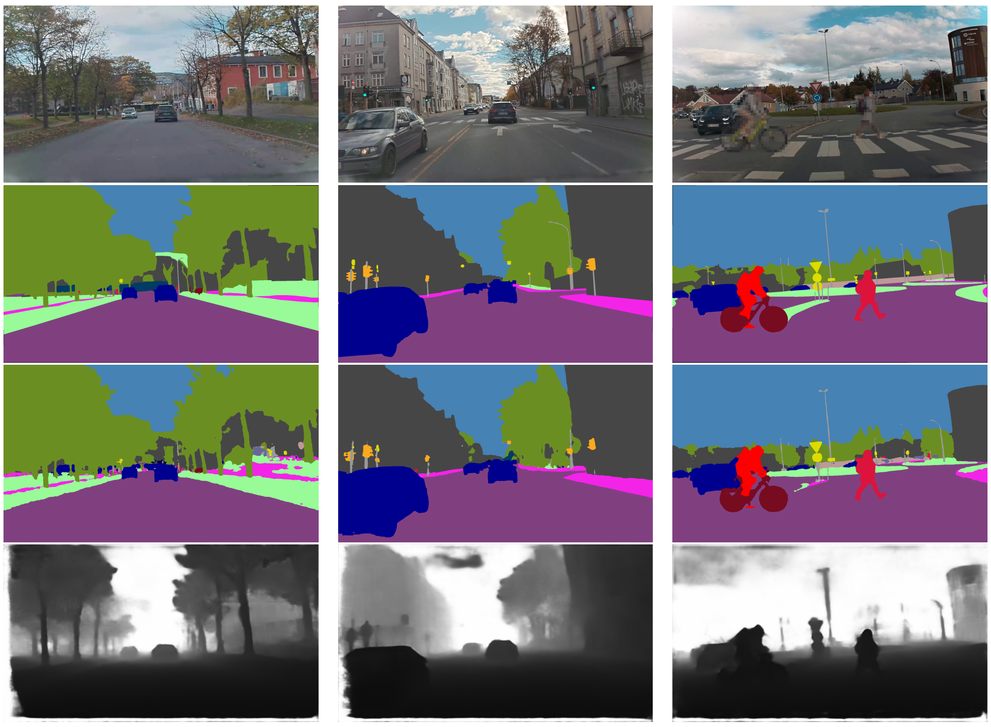


Figure 4.18: Experiments 5: NAPLab qualitative results. Pedestrians and license plates are blurred for anonymization. First row: original image, second row: ground truth segmentation mask, third row: predicted segmentation mask, fourth row: predicted depth map.

Chapter 5

Discussion

This chapter discusses the results of the experiments and their implications on the potential of the model in an autonomous driving setting. The chapter also reflects on the shortcomings of the thesis and makes an attempt at answering the research questions.

5.1 Potential in Autonomous Driving

An important requirement for deep learning architectures applied in autonomous vehicles is real-time performance. There is no exact definition of what real-time is, but a frame rate of approximately 30 FPS is usually considered real-time. In experiment 2, models with different backbone sizes were tested. The B0 model was able to achieve a frame rate of roughly 60 FPS, which is undoubtedly considered real-time. The B2 model is also close to real-time with a frame rate of roughly 20 FPS. However, it is important to note that the experiments were conducted on powerful GPUs that may not be available in an autonomous vehicle. In addition, only the pure inference time was measured. In a real-life scenario, dataloading and post-processing will take up additional time. The vehicle also needs time to make decisions based on the predictions. Thus, the actual frame rate in an autonomous vehicle will probably be lower than the one measured in the experiments.

One way to increase the inference speed of the model is to use TensorRT. This is a tool provided by NVIDIA that optimizes the model for fast inference. This is done by lowering the number precision used during inference, which significantly improves processing time. How significant the improvement is depends on the hardware being used. TensorRT can thus be a possible solution to speed up inference.

While inference speed is important in autonomous driving, it is also crucial that the model is able to produce accurate predictions. In experiment 2 the largest models, B2 and B4, clearly outperformed B0 in terms of accuracy. B0 is arguably the model best suited for real-time applications, so it would be beneficial to increase the performance of this model. One way of achieving better performance is to increase the resolution of the input images. This will help the model capture finer details in the images, and will thus lead to higher accuracy. However, the frame rate of the model will also decrease. Consequently, different images sizes should be tested to find the optimal balance between accuracy and inference speed.

5.2 Shortcomings of the Thesis

This section reflects on the shortcomings of the thesis and suggests possible solutions.

5.2.1 Early Overfitting

In experiment 1 and 2 it was observed that the segmentation loss starts to increase quite early in the training process. This usually indicates that the model is overfitting to the training data. However, the accuracy of the model continues to improve after this point. This is quite surprising, as there is usually a clear correlation between the loss and accuracy. This probably happens since the cross-entropy loss, which is used for the segmentation task, is calculated on the class probabilities of each pixel. The accuracy, on the other hand, is only calculated on the final prediction mask. Thus, if more pixels are predicted correctly but with less confidence, the accuracy will improve even though the loss increases.

One possible reason overfitting occurs could be that too few augmentations are applied during training. The original SegFormer model applies random cropping to the images during training. This technique was not implemented with the multitask model since it affects the depth estimation task negatively. Thus, the effective size of the dataset is reduced compared to SegFormer, which in turn could affect the generalization ability of the model. A possible solution could be to find alternative augmentation techniques that works well for both depth estimation and semantic segmentation.

There was spent little time experimenting with different hyperparameters during training of the models. Thus, the hyperparameters are probably not optimally tuned, and likely contributes to the overfitting problem. The learning rate is arguably the most important hyperparameter. In the experiments, a constant learning rate was used for simplicity. However, a learning rate scheduler or a manual decrease of the learning rate after the first initial training epochs would likely lead to better performance. Another interesting approach would be to use individual learning rates for each of the task-specific decoders. This would make it easier to achieve the best accuracy for both depth and segmentation after the same number of epochs.

5.2.2 Few Available Datasets

This thesis investigates the use of a multitask model for monocular depth estimation and semantic segmentation. It was thus important to find datasets that are annotated for both tasks. This was challenging, as most of the publicly available datasets mainly focus on one of these tasks. The three datasets chosen all contain annotations for both tasks, but they still have some issues. Cityscapes is a relatively small dataset with only 5 000 annotated frames. The depth maps are precomputed from stereo cameras, and is thus not as accurate as LiDAR depth maps. KITTI-360 is a much larger dataset with about 60 000 annotated frames and LiDAR ground truth. However, the depth maps are quite sparse and does not cover the entire input image.

The lack of annotated real-world data motivated the use of synthetic data for this thesis. Synthetic datasets are labeled automatically, and usually contain a large number of high-quality annotated frames for a wide range of vision tasks. The chosen dataset, Apollo Synthetic Dataset, contains a total of 273 000 annotated frames. About 45 000 of these were used in the experiments. The dataset proved useful for evaluation of the multitask model. However, synthetic datasets can not completely substitute real-world data as the domain shift between synthetic and real-world data is too large.

Semi-supervised and self-supervised methods were not explored in this thesis, but could potentially be interesting to look into. With a semi-supervised approach it would be possible to train the model for both tasks using only annotations for a single task. With a self-supervised approach no annotations would be needed at all during training. As the semi-supervised and self-supervised methods require less annotated data, it would be easier to find suitable datasets for this thesis. However, annotated data for both tasks would still be needed to evaluate the performance of the models.

5.3 Fulfillment of Research Questions

This section makes an attempt at answering the research questions of this thesis based on the findings from the experiments.

5.3.1 Research Question 1

How does a multitask Vision Transformer trained for both segmentation and depth estimation simultaneously perform compared to models trained for individual tasks?

This research question was investigated in experiment 1. For depth estimation, the results indicate a slight decrease in accuracy when using the multitask approach, but the differences are small. For semantic segmentation, no significant increase or decrease in accuracy was observed when using the multitask approach. It is important to note that the individual task models can increase their accuracy further by applying task-specific augmentations, such as random cropping and Vertical CutDepth, during training. These augmentations do not necessary work well for a multitask model, as the two prediction tasks are fundamentally different. Thus, the actual performance gap could possibly be bigger than the one reported in experiment 1.

In terms of inference speed, the multitask model has comparable inference speed to the individual segmentation model while performing an additional task. Compared to running the individual depth and segmentation models one after another, the multitask model increases the effective inference speed by almost 50 %. This proves that the multitask approach is an effective way to increase inference speed when multiple dense prediction tasks needs to be performed at the same time.

Conclusion

A multitask model is able to achieve comparable performance to individual task models, while lowering the effective inference time compared to running the two tasks individually.

5.3.2 Research Question 2

Can synthetic datasets be used to increase model performance when there is little real-world data available?

This research question was investigated in experiment 3. For depth estimation, the accuracy was significantly increased by pre-training on a synthetic dataset. However, for semantic segmentation no improvement was observed. This is quite surprising, as introducing a large synthetic dataset usually leads to better performance. The lack of improvement could be due to sub-optimal hyperparameter tuning and the lack of augmentations. This is discussed in Section 5.2.1.

In experiment 3, the models were pre-trained on a synthetic dataset and then fine-tuned on real-world data. This is only one of many possible methods for using synthetic data during training. Another possible approach is to mix the synthetic and real-world data during training and train on both datasets simultaneously. This approach is used by Ros *et al.* [16] and produces good results. As only one approach was tested in this thesis, it is not possible to say what the optimal way of utilizing synthetic data is.

It would be interesting to test multiple synthetic datasets to investigate how the choice of dataset affects performance. Properties such as dataset size and environmental variation could potentially have an influence on the results. However, due to time constraints only one synthetic dataset was used in this thesis.

Conclusion

Pre-training on a synthetic dataset can be used to increase model performance when there is litte real-world data available. However, it is crucial to optimize the training procedure in order to achieve the expected results.

5.3.3 Research Question 3

How accurate are the absolute depth predictions from the model?

This research question was investigated in experiment 4. The results show that the model is able to create fairly accurate depth predictions for the dataset it was trained on. The prediction error is often less than a meter. However, when tested on a different dataset than the one used for training, the model fails to create accurate depth predictions, and the prediction error is extremely large. This indicates that the model has learned features specific to the dataset used for training, and is unable to generalize to new and unseen datasets.

It is difficult to pinpoint exactly why the model struggles on new datasets. One possible reason could be that the sensor setup is different for different datasets. Thus, different image ratios and camera poses could potentially contribute to the error. Dijk and Croon [40] investigates what kind of information a neural network uses in order to estimate depth. Interestingly, they find that the network heavily relies on the vertical position of objects to estimate their depth. As the vertical position changes with different camara poses, this could potentially explain why the model is unable to estimate depth accurately for new datasets.

One possible solution could be to train the model on multiple depth estimation datasets simultaneously. This could potentially force the model to focus on other features than the vertical position and lead to better generalization abilities. However, this is just a hypothesis, and the effectiveness of this approach is not verified. It would be interesting to test this approach in the experiments, but it was not done due to time constraints.

Conclusion

The model is able to create fairly accurate depth predictions for the dataset it was trained on, but struggles on new and unseen data. It is thus necessary to fine-tune the model for the desired sensor setup and environment to ensure accurate predictions.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This thesis has investigated the use of Vision Transformers for dense prediction tasks in autonomous driving. The main goal was to design a Transformer-based multitask model that is able to perform both monocular depth estimation and semantic segmentation simultaneously. The model should also be able to operate in real-time. A literature review was conducted in order to find state-of-the-art Vision Transformers for the task. Two Transformer-based models, SegFormer and GLP-Depth, were chosen based on the literature review. These models were combined into a multitask architecture with a common hierarchical Transformer encoder and two lightweight, task-specific decoders.

In order to evaluate the effectiveness of the proposed model, it was tested with three different datasets: Cityscapes, KITTI-360 and Apollo Synthetic Dataset. First, the performance of the multitask model and the individual task models were compared. The results showed that the multitask model achieves comparable accuracy to the individual task models, while lowering the total inference time for both tasks significantly. Next, the model was tested with different backbone sizes. The results showed that the choice of backbone can be used to effectively control the ratio between inference speed and accuracy.

There was also conducted an experiment to investigate how pre-training on a large synthetic dataset affects model performance. The model was first pre-trained on Apollo Synthetic Dataset, and then fine-tuned on Cityscapes. This approach increased the depth estimation accuracy significantly, while the segmentation accuracy remained roughly the same. Finally, the models ability to estimate depth accurately was evaluated. This was done by examining the predicted distance to individual objects in the frames and comparing with the ground truth. While the model was able to estimate depth quite accurately for the dataset it was trained on, it struggled on new and unseen datasets.

6.2 Future Work

This section presents ideas related to Vision Transformers and autonomous driving that were not explored in this thesis, but could be interesting to investigate further in future work.

6.2.1 Semi-supervised or Self-supervised Learning

In this thesis a supervised learning approach was used to train the multitask model. This approach gives high accuracy, but requires large amounts of annotated data for both depth and segmentation. Annotating data for semantic segmentation is very time-consuming as labeling is done on pixel-level. Depth ground truth is usually gathered using a LiDAR sensor, which is both large and expensive. It would thus be beneficial to use methods that require less annotated data during training, as this would save both time and money. A semi-supervised model would only require ground truth data for one of the tasks, while a self-supervised model would not require any ground truth data at all. As semi-supervised and self-supervised methods have improved a lot in recent years, it would be interesting to test them with the multitask model.

6.2.2 Mixing Datasets for Depth Estimation

The proposed multitask model was able to create accurate depth predictions for the dataset it was trained on. However, the model failed to estimate depth correctly for new and unseen datasets. This means that the model has to be fine-tuned on each new dataset to ensure good performance. As data collection and training is expensive and time-consuming, it would be beneficial if the model could perform well across multiple datasets without additional fine-tuning. A possible solution is to use a mix of multiple depth estimation datasets during training. In theory, this would force the model to rely on more general features, and not features specific to a single dataset. Ranftl *et al.* [41] uses this approach to achieve good performance across multiple datasets without additional fine-tuning. As most of the recent work related to depth estimation focuses on performance on a single benchmark, it would be interesting to investigate this more general approach further.

Bibliography

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser and I. Polosukhin, *Attention is all you need*, 2017. DOI: `10.48550/ARXIV.1706.03762`. [Online]. Available: `https://arxiv.org/abs/1706.03762`.

[2] 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*, 2020. DOI: `10.48550/ARXIV.2010.11929`. [Online]. Available: `https://arxiv.org/abs/2010.11929`.

[3] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2014. DOI: `10.48550/ARXIV.1412.6980`. [Online]. Available: `https://arxiv.org/abs/1412.6980`.

[4] A. Kirillov, K. He, R. Girshick, C. Rother and P. Dollar, 'Panoptic segmentation,' in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2019.

[5] D. Eigen, C. Puhrsch and R. Fergus, *Depth map prediction from a single image using a multi-scale deep network*, 2014. DOI: `10.48550/ARXIV.1406.2283`. [Online]. Available: `https://arxiv.org/abs/1406.2283`.

[6] J. Devlin, M.-W. Chang, K. Lee and K. Toutanova, *Bert: Pre-training of deep bidirectional transformers for language understanding*, 2018. DOI: `10.48550/ARXIV.1810.04805`. [Online]. Available: `https://arxiv.org/abs/1810.04805`.

[7] A. Radford and K. Narasimhan, 'Improving language understanding by generative pre-training,' 2018.

[8] 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*, 2016. DOI: `10.48550/ARXIV.1604.01685`. [Online]. Available: `https://arxiv.org/abs/1604.01685`.

[9] A. Geiger, P. Lenz and R. Urtasun, 'Are we ready for autonomous driving? the kitti vision benchmark suite,' in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 3354–3361. DOI: `10.1109/CVPR.2012.6248074`.

[10] A. Geiger, P. Lenz, C. Stiller and R. Urtasun, 'Vision meets robotics: The kitti dataset,' *International Journal of Robotics Research (IJRR)*, 2013.

[11] Y. Liao, J. Xie and A. Geiger, *Kitti-360: A novel dataset and benchmarks for urban scene understanding in 2d and 3d*, 2021. arXiv: `2109.13410 [cs.CV]`.

[12] J. Geyer, Y. Kassahun, M. Mahmudi, X. Ricou, R. Durgesh, A. S. Chung, L. Hauswald, V. H. Pham, M. Mühlegg, S. Dorn, T. Fernandez, M. Jänicke, S. Mirashi, C. Savani, M. Sturm, O. Vorobiov, M. Oelker, S. Garreis and P. Schuberth, *A2d2: Audi autonomous driving dataset*, 2020. DOI: `10.48550/ARXIV.2004.06320`. [Online]. Available: `https://arxiv.org/abs/2004.06320`.

[13] NTNU. 'Ntnu autonomous perception laboratory (naplab).' (2022), [Online]. Available: `https://www.ntnu.edu/idi/naplab` (visited on 12/06/2022).

[14] A. Gaidon, Q. Wang, Y. Cabon and E. Vig, *Virtual worlds as proxy for multi-object tracking analysis*, 2016. arXiv: `1605.06457 [cs.CV]`.

[15] Y. Cabon, N. Murray and M. Humenberger, *Virtual kitti 2*, 2020. arXiv: `2001.10773 [cs.CV]`.

[16] G. Ros, L. Sellart, J. Materzynska, D. Vazquez and A. M. Lopez, 'The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes,' in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 3234–3243. DOI: `10.1109/CVPR.2016.352`.

[17] Baidu. 'Apollo synthetic dataset.' (2019), [Online]. Available: `http://apollo.baidu.com/synthetic.html` (visited on 21/05/2022).

[18] X. Weng, Y. Man, J. Park, Y. Yuan, D. Cheng, M. O'Toole and K. Kitani, 'All-In-One Drive: A Large-Scale Comprehensive Perception Dataset with High-Density Long-Range Point Clouds,' *arXiv*, 2021.

[19] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez and V. Koltun, *Carla: An open urban driving simulator*, 2017. DOI: `10.48550/ARXIV.1711.03938`. [Online]. Available: `https://arxiv.org/abs/1711.03938`.

[20] S. Zheng, J. Lu, H. Zhao, X. Zhu, Z. Luo, Y. Wang, Y. Fu, J. Feng, T. Xiang, P. H. S. Torr and L. Zhang, *Rethinking semantic segmentation from a sequence-to-sequence perspective with transformers*, 2020. DOI: `10.48550/ARXIV.2012.15840`. [Online]. Available: `https://arxiv.org/abs/2012.15840`.

[21] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez and P. Luo, *Segformer: Simple and efficient design for semantic segmentation with transformers*, 2021. DOI: `10.48550/ARXIV.2105.15203`. [Online]. Available: `https://arxiv.org/abs/2105.15203`.

[22] W. Wang, E. Xie, X. Li, D.-P. Fan, K. Song, D. Liang, T. Lu, P. Luo and L. Shao, *Pyramid vision transformer: A versatile backbone for dense prediction without convolutions*, 2021. DOI: `10.48550/ARXIV.2102.12122`. [Online]. Available: `https://arxiv.org/abs/2102.12122`.

[23] B. Cheng, A. G. Schwing and A. Kirillov, *Per-pixel classification is not all you need for semantic segmentation*, 2021. DOI: `10.48550/ARXIV.2107.06278`. [Online]. Available: `https://arxiv.org/abs/2107.06278`.

[24] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin and B. Guo, *Swin transformer: Hierarchical vision transformer using shifted windows*, 2021. DOI: `10.48550/ARXIV.2103.14030`. [Online]. Available: `https://arxiv.org/abs/2103.14030`.

[25] B. Cheng, I. Misra, A. G. Schwing, A. Kirillov and R. Girdhar, *Masked-attention mask transformer for universal image segmentation*, 2021. DOI: `10.48550/ARXIV.2112.01527`. [Online]. Available: `https://arxiv.org/abs/2112.01527`.

[26] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, 'Imagenet: A large-scale hierarchical image database,' in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255. DOI: `10.1109/CVPR.2009.5206848`.

[27] J. Jain, A. Singh, N. Orlov, Z. Huang, J. Li, S. Walton and H. Shi, *Semask: Semantically masked transformers for semantic segmentation*, 2021. DOI: `10.48550/ARXIV.2112.12782`. [Online]. Available: `https://arxiv.org/abs/2112.12782`.

[28] H. Yan, C. Zhang and M. Wu, *Lawin transformer: Improving semantic segmentation transformer with multi-scale representations via large window attention*, 2022. DOI: `10.48550/ARXIV.2201.01615`. [Online]. Available: `https://arxiv.org/abs/2201.01615`.

[29] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy and A. L. Yuille, *Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs*, 2016. DOI: `10.48550/ARXIV.1606.00915`. [Online]. Available: `https://arxiv.org/abs/1606.00915`.

[30] S. F. Bhat, I. Alhashim and P. Wonka, 'AdaBins: Depth estimation using adaptive bins,' in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, Jun. 2021. DOI: `10.1109/cvpr46437.2021.00400`. [Online]. Available: `https://doi.org/10.1109%2Fcvpr46437.2021.00400`.

[31] R. Ranftl, A. Bochkovskiy and V. Koltun, *Vision transformers for dense prediction*, 2021. DOI: `10.48550/ARXIV.2103.13413`. [Online]. Available: `https://arxiv.org/abs/2103.13413`.

[32] D. Kim, W. Ga, P. Ahn, D. Joo, S. Chun and J. Kim, *Global-local path networks for monocular depth estimation with vertical cutdepth*, 2022. DOI: `10.48550/ARXIV.2201.07436`. [Online]. Available: `https://arxiv.org/abs/2201.07436`.

[33] Y. Ishii and T. Yamashita, *Cutdepth:edge-aware data augmentation in depth estimation*, 2021. DOI: 10.48550/ARXIV.2107.07684. [Online]. Available: https://arxiv.org/abs/2107.07684.

[34] Z. Li, Z. Chen, X. Liu and J. Jiang, *Depthformer: Exploiting long-range correlation and local information for accurate monocular depth estimation*, 2022. DOI: 10.48550/ARXIV.2203.14211. [Online]. Available: https://arxiv.org/abs/2203.14211.

[35] Z. Li, X. Wang, X. Liu and J. Jiang, *Binsformer: Revisiting adaptive bins for monocular depth estimation*, 2022. DOI: 10.48550/ARXIV.2204.00987. [Online]. Available: https://arxiv.org/abs/2204.00987.

[36] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov and S. Zagoruyko, *End-to-end object detection with transformers*, 2020. DOI: 10.48550/ARXIV.2005.12872. [Online]. Available: https://arxiv.org/abs/2005.12872.

[37] R. Garg, V. K. BG, G. Carneiro and I. Reid, *Unsupervised cnn for single view depth estimation: Geometry to the rescue*, 2016. DOI: 10.48550/ARXIV.1603.04992. [Online]. Available: https://arxiv.org/abs/1603.04992.

[38] M. Själander, M. Jahre, G. Tufte and N. Reissmann, *EPIC: An energy-efficient, high-performance GPGPU computing research infrastructure*, 2019. arXiv: 1912.05848 [cs.DC].

[39] A. Buslaev, V. I. Iglovikov, E. Khvedchenya, A. Parinov, M. Druzhinin and A. A. Kalinin, 'Albumentations: Fast and flexible image augmentations,' *Information*, vol. 11, no. 2, p. 125, Feb. 2020. DOI: 10.3390/info11020125. [Online]. Available: https://doi.org/10.3390%2Finfo11020125.

[40] T. van Dijk and G. C. H. E. de Croon, *How do neural networks see depth in single images?* 2019. DOI: 10.48550/ARXIV.1905.07005. [Online]. Available: https://arxiv.org/abs/1905.07005.

[41] R. Ranftl, K. Lasinger, D. Hafner, K. Schindler and V. Koltun, *Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer*, 2019. DOI: 10.48550/ARXIV.1907.01341. [Online]. Available: https://arxiv.org/abs/1907.01341.