Tartu University

Faculty of Science and Technology

Institute of Technology

Ali Maharramov

Asymmetric Deep Multi-Task Learning

Master's thesis (30 EAP)
Robotics and Computer Engineering

Supervisor:

MSc. Tambet Matiisen

Tartu 2024

Resümee/Abstract

Asümmeetriline mitmikülesande õpe

Kiire areng sügavate närvivõrkude valdkonnas on muutnud need oluliseks komponendiks isejuhtivate sõidukite lahendustes. Närvivõrke saab kasutada nii otse juhtimiskäskluste ennustamiseks isejuhtivas süsteemis, või ka konkreetsete alamülesannete lahendamiseks keerulisemas modulaarses süsteemmis. Kui süsteemis kasutatakse närvivõrke mitme erineva ülesande jaoks, siis ühise mitme väljundiga närvivõrgu treenimine võib olla efektiivsem kui eraldi närvivõrgu treenimine iga üksiku ülesande jaoks, seda nii nii töökiiruse kui mälukasutuse mõttes. Samuti võib üheks eeliseks olla närvivõrgu täiendav üldistusvõime tänu teadmuse ülekandumisele ühelt ülesandelt teisele. Samas ei ole selliste mitmikülesande närvivõrkude treenimine lihtne. Selle üheks põhjuseks on, et puuduvad sobivad andmestikud, mis sisaldaksid märgendeid kõikide vajalike ülesannete jaoks. Nende märgendite tekitamine, kas automaatselt või käsitsi, on täiendav aja- ja rahakulu. Seetõttu pakub huvi kuidas treenida mitmikülesande võrku asümmeetriliste andmestike peal, kus ühe alamhulga jaoks on olemas ühe ülesande märgendid ja teise alamhulga jaoks on olemas teise ülesande märgendid. Naiivne treenimine sellise andmestiku peab võib anda tulemuseks ebaühtlase täpsuse erinevate ülesannete vahel. Antud töös uuritaksegi võimalusi mitmikülesande võrkude paremaks treenimiseks asümmeetriliste andmestike peal ning võrreldakse tulemusi sümmeetriliste andmestike ning üksikülesande võrkudega.

CERCS: P176 Tehisintellekt; T111 Pilditehnika;

Märksõnad: asümmeetriline mitmikülesande õpe, sügavad närvivõrgud, gradientide akumuleerimine

Asymmetric Deep Multi-Task Learning

Recent developments make deep neural networks a valuable asset for autonomous driving. They can be deployed as an end-to-end system or part of more complex systems for specific tasks. If a system needs several tasks by neural networks, using multi-task learning (MTL) introduces few benefits compared to deploying several single-task learning (STL) models, such as better time and space complexity on deployment and potentially increased generalization on the backbone network. However, MTL often faces unique challenges. Many existing MTL datasets have limited labels or lack the required labels for specific tasks, and generating labels for these tasks leads to resource and time consumption for researchers. Training the model on an asymmetric labeled dataset, a dataset where labels for specific tasks are unavailable for a subset, can cause a biased gradient, reflecting an unbalance in the accuracy of tasks. In this thesis, asymmetric MTL were investigated and compared to symmetric MTL and STL methods.

CERCS: P176 Artificial intelligence; T111 Imaging, image processing;

Keywords: asymmetric multi-task learning, deep learning, gradient accumulation

Contents

List of Figures

List of Tables

Lühendid, konstandid, mõisted

NN - Neural Network

DNN - Deep Neural Network

DL - Deep Learning

CNN - Convolutional Neural Network

STL - Single Task Learning

MTL - Multi Task Learning

FLOPs - Floating Point Operations

YOLO - You Only Look Once

3D - 3 Dimensional

2D - 2 Dimensional

RGB - Red Green Blue

ALVINN - Autonomous Land Vehicle In a Neural Network

LIDAR - Light Detection and Ranging

FPN - Feature Pyramid Networks

A2D2 - Audi Autonomous Driving Dataset

PNG - Portable Network Graphics

GPU - Graphics processing unit

PC - Personal Computer

HPC - High Performance Computing

IOU - Intersection over Union

1 Introduction

Autonomous driving researchers have been interested in deep neural networks since the 1990s when research like ALVINN was conducted [18]. Recent advancements in deep learning networks have made them reliable enough to solve complex tasks [39, 40, 42, 43]. Moreover, the trend of increasing computing performance [37] has made deep learning suitable for real-time applications. Deep learning can be implemented as a core element of end to end systems [4] or can be used as a part of more complex autonomous driving systems to extract information about the surroundings; for example, the deep neural network can be implemented for segmentation tasks for road line detection [6, 7] or object detection task for pedestrian and traffic sign detection [24, 25].

In order to gain a more comprehensive understanding of the environment, it is sometimes necessary to employ multiple representations as input. For example, applications shown in Figure1.1 combining semantic segmentation, sparse depth estimation and RGB image can enable accurate depth estimation [45], while object detection and semantic segmentation can be combined to achieve panoptic segmentation [49], which provides higher-level information from semantic segmentation [44]. Similarly, object detection and depth estimation can be combined to obtain 3D bounding box information [48].

All mentioned applications may serve as a reason for deploying multiple deep neural networks to obtain input representations. However, rather than deploying several networks, each dedicated to a single task, adopting a single multi-task deep neural network capable of simultaneously handling all tasks may prove more advantageous. By sharing parameters across tasks, the model inherently reduces the number of parameters, leading to fewer Floating Point Operations (FLOPs) required for computation [1]. This reduction in parameters and FLOPs translates to faster inference times.

In addition to the advantage of inference time, some studies claim that because of sharing knowledge among tasks, multi-task learning may achieve better generalization, which increases model performance on unseen data [46, 47].

Despite its benefits, using multi-task learning (MTL) models presents challenges, particularly in securing symmetric datasets in which labels for all tasks exist. In scenarios where symmetric training is impossible, symmetric datasets are not available; asymmetric training, using asymmetric datasets, where labels of one or several tasks are not present for all tasks, for the training process is one option. However, if there is an imbalance in the quantity of labels, that imbalance may represent itself in the final result, where the trained deep learning model performs one task better than others [15]. Additionally, if the loss functions of different tasks have different ranges, that may cause an imbalance in gradient updates, resulting in the trained network performing one task better than the others. It is important to note that this last challenge exists for both symmetric and asymmetric training cases.

This thesis aims to train deep neural network models asymmetrically to get enough performance for real-world applications and analyze and evaluate that model by comparing several other methods. First, single-task neural network models are trained for various tasks, such as

PilotNet for steering angle prediction [4], U-Net for segmentation [5], YOLO with Darknet backbone for 2D bounding box prediction [14], and DenseDepth for depth estimation [20]. In addition to single-task models, the multi-task model was also trained symmetrically to compare the investigated training method to the traditional one. Finally, multi-task architecture was trained for a single task.

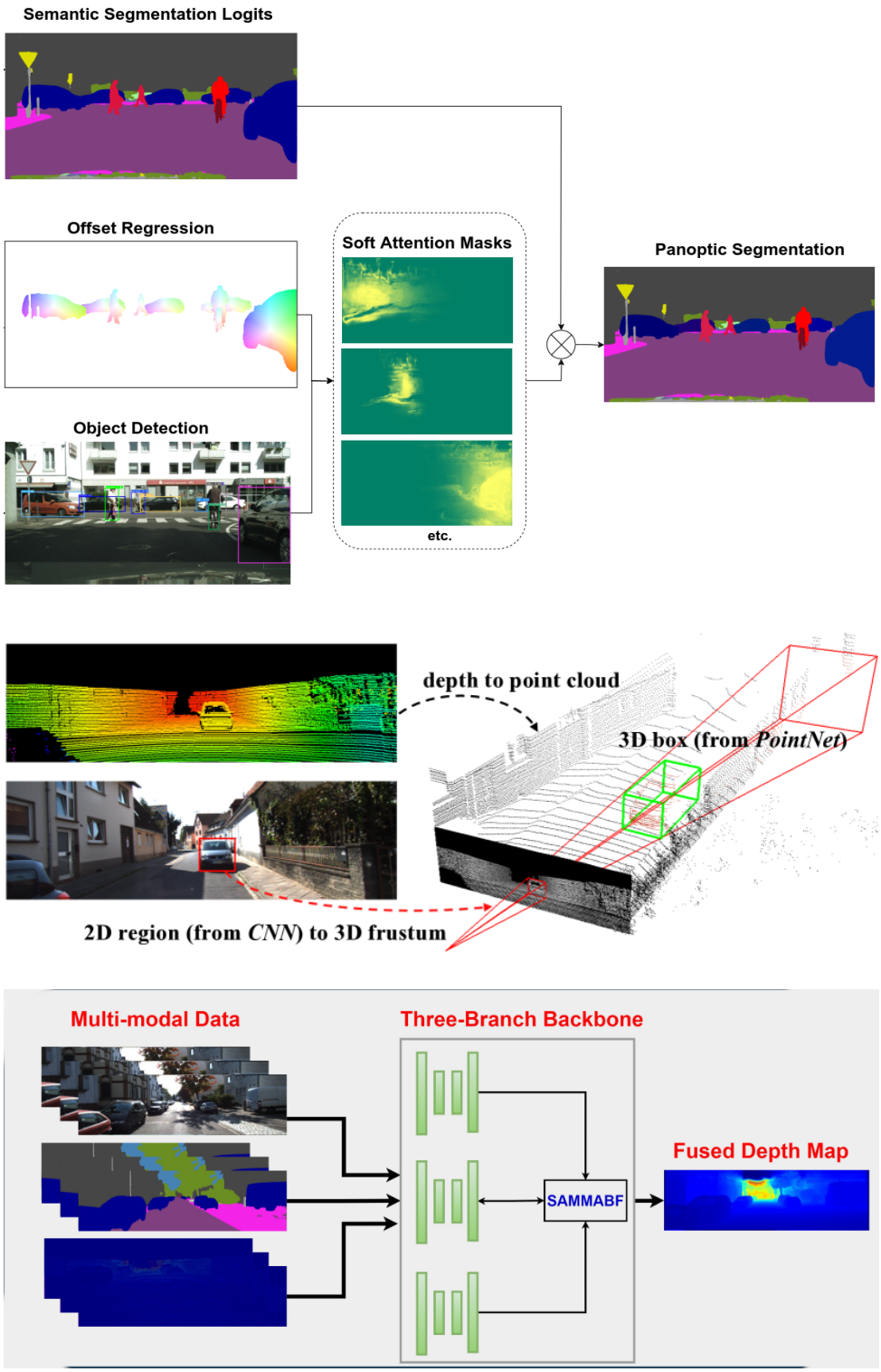


Figure 1.1: **Applications that combine different representations of the environment and output representation have higher levels of information.** Top: DL takes 2D bounding boxes, semantic segmentation and pixel offset to generate panoptic segmentation [49]. Middle: Application which generates 3D bounding box from sparse depth mask and 2D bounding box [48]. Bottom: The model generates a dense depth mask from semantic segmentation and sparse depth [45].

2 Background

2.1 Tasks

Understanding the surrounding environment is essential for autonomous driving. As mentioned above, semantic segmentation, depth estimation, and 2D bounding box tasks can be fused to achieve a better understanding of the environment. However, these representations hold enough information to support autonomous driving even without fusing. All tasks in this work are chosen for that reason. Semantic segmentation allows understanding of the road environment; with enough samples, it is possible to distinguish driveable areas, pot-holes [33], puddles, etc. Depth estimation gives a 3D understanding, which can be crucial for safety measurements. By combining depth estimation with object detection, it is possible to detect pedestrians and other cars and plan driving strategies for them. Steering angle prediction can be used directly for control steering or processed through a hierarchical system [4, 38].

2.1.1 Segmentation

Semantic segmentation is the task of assigning one class for each pixel [16]. That class may be an object like a car, bus, traffic sign, including background elements like the sky, road, or building. When the task is implemented using neural networks, labels are a stack of masks, where each mask represents one class, and if a pixel belongs to that class, it is represented as one, if not zero as shown in Figure2.1. During inference, the values of every pixel among masks were compared, and the highest number for each pixel was assigned to the class.

2.1.2 Sparse depth estimation

The depth estimation task involves assigning distance values from 0-255 to each pixel in an image. The lower the assigned value, the closer the object associated with the pixel is to the vehicle. The maximum value represents the maximum distance of the LIDAR sensor. When depth information is available for all pixels, it is known as dense depth estimation. However, in some cases, depth information may not be available for all pixels, which results in sparse depth estimation as one example shown in Figure2.2.

However, the depth estimation tasks should not be confused with the depth completion tasks. Depth completion means filling in missed information from sparse depth masks. Meanwhile, depth estimation means predicting depth information from RGB images [45].

2.1.3 Object detection

Object detection is the task of estimating a 2D bounding box, drawing boxes around objects, and classifying those objects. Successful object detectors should be able to differentiate different instances of the same objects like in Figure2.3. One of the most common implementations

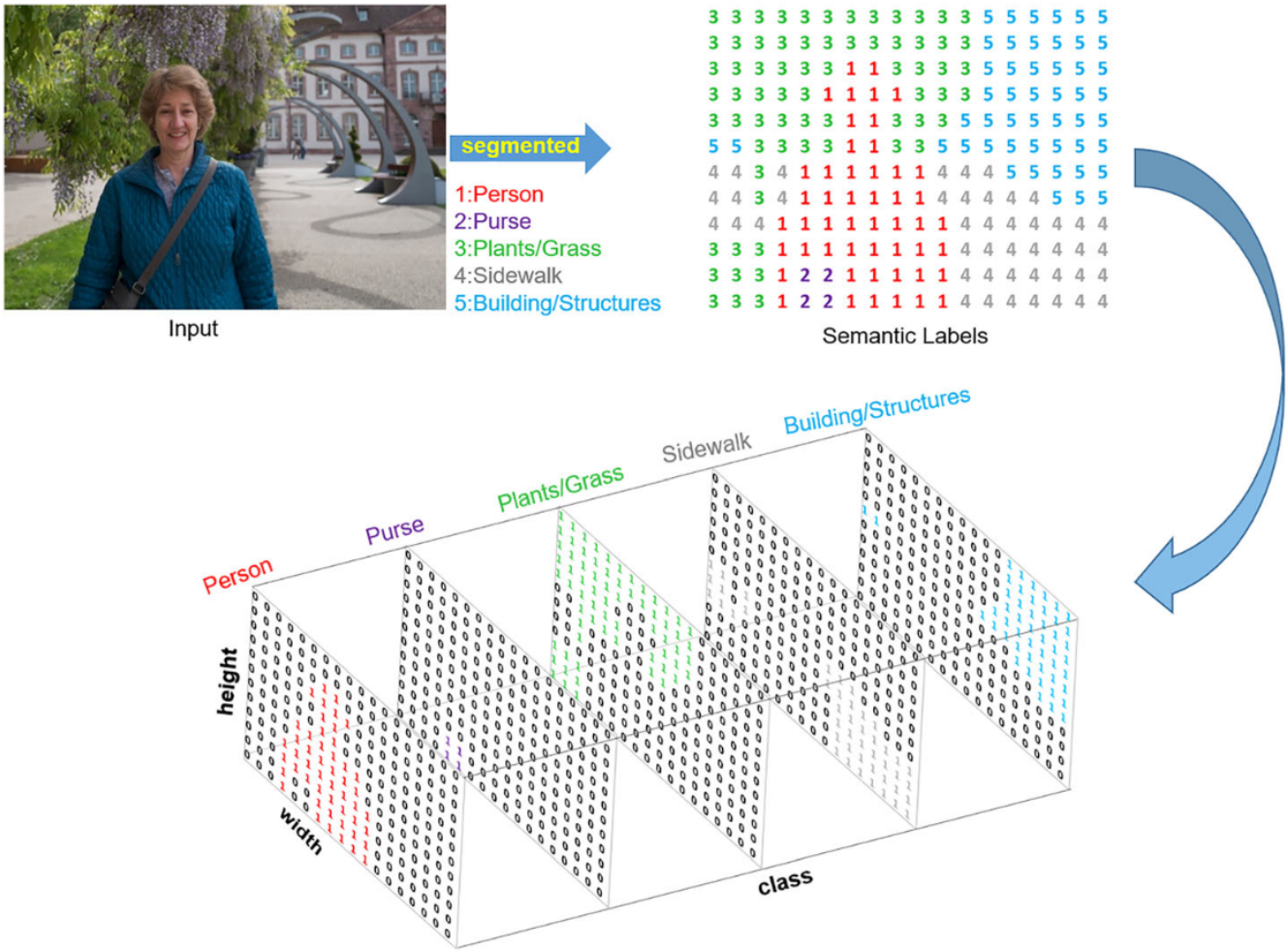


Figure 2.1: **Implementation of semantic segmentation task for DL:** A segmentation task that can represent information for 5 classes [16].

of object detection is YOLO; when labels are represented by YOLO [14], there are 8x8 or 16x16 grids, and for each grid, there are bounding box parameters that represent the presence or absence of objects.

Depending on the implementation, there are several bounding box labels for each grid cell. For each bounding box, the first value represents whether there is an object present or not. The second, third, fourth, and fifth values represent the center of the bounding box in the grid and the size of the box. The remaining values represent probabilities of classes for those boxes. One example is illustrated in Figure2.4.

2.1.4 Steering angle prediction

Waypoint and steering angle predictions are the most used control systems for autonomous driving. Waypoint prediction is the task of building a route for a given time; meanwhile, steering angle prediction predicts one or several values that represent the maneuver vehicle. These values may be the angle of the ego vehicle's frontal wheels or steering wheel at a given time. Compared to waypoint steering angle has a lower computational cost

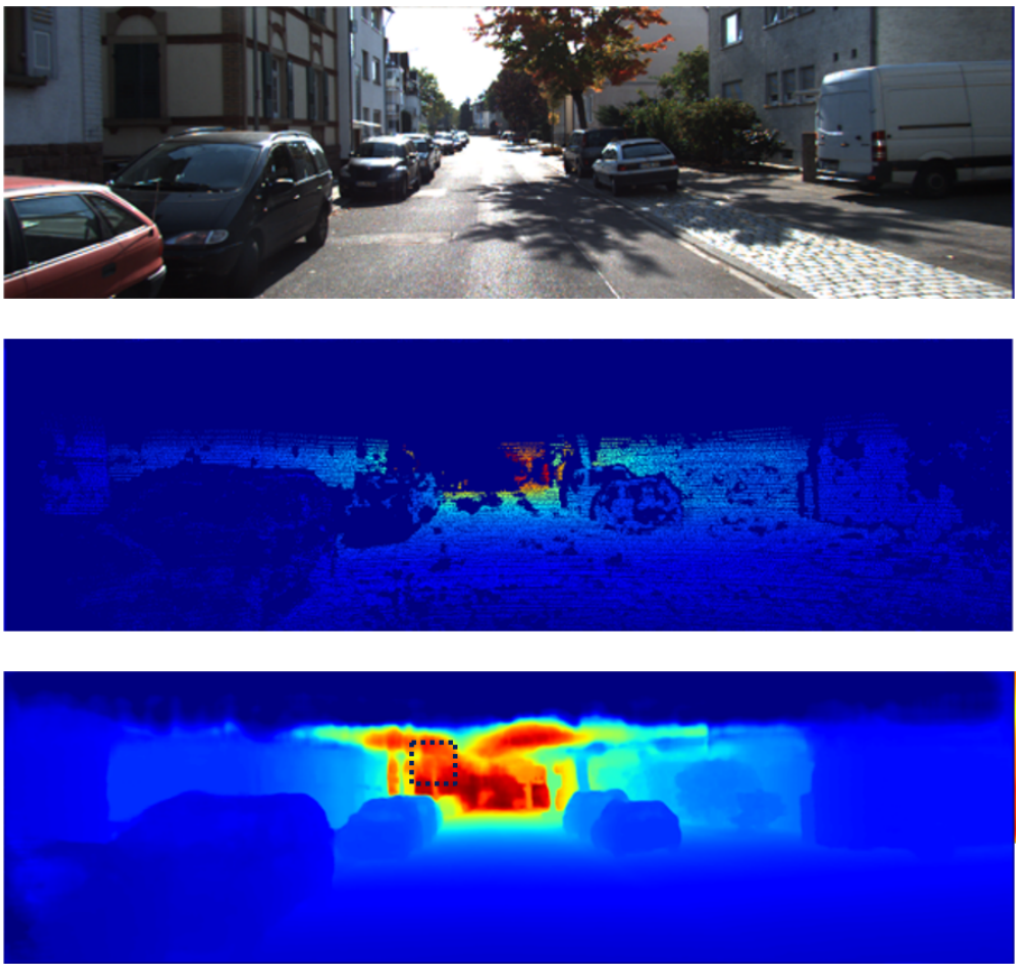


Figure 2.2: **Comparison of sparse and dense depth mask:** From top to bottom: Input image, sparse depth mask, and dense depth mask. For ease of understanding, the pixel is represented with a colormap in this visualization. The black pixel in the sparse mask represents the absence of depth information [9].

2.2 Architectures

2.2.1 ResNET

The process of combining pre-existing architectures with task-specific decoders is a popular approach. One of the most commonly used architectures for this purpose is ResNet [34]. ResNet is highly effective in learning informative representations of images, thanks to its skip connections and well-designed building blocks. According to the authors' experiments, skip connections performed better to solve the vanishing gradient problem, where the calculated gradient starts to not affect the model's result. Although ResNet architectures come in different sizes, they all use the same building blocks, as shown in Figure2.5.

2.2.2 Feature Pyramid Networks

FPN is a deep learning method that extracts several multi-scale feature maps from input images [32]. Generally, FPN and similar architectures consist of bottom-up and top-down pathways, which can be seen in Figure2.6. The bottom-up pathway is a feature map generated by applying convolutional layers one after the other. With each layer, we get a deeper feature map. After the

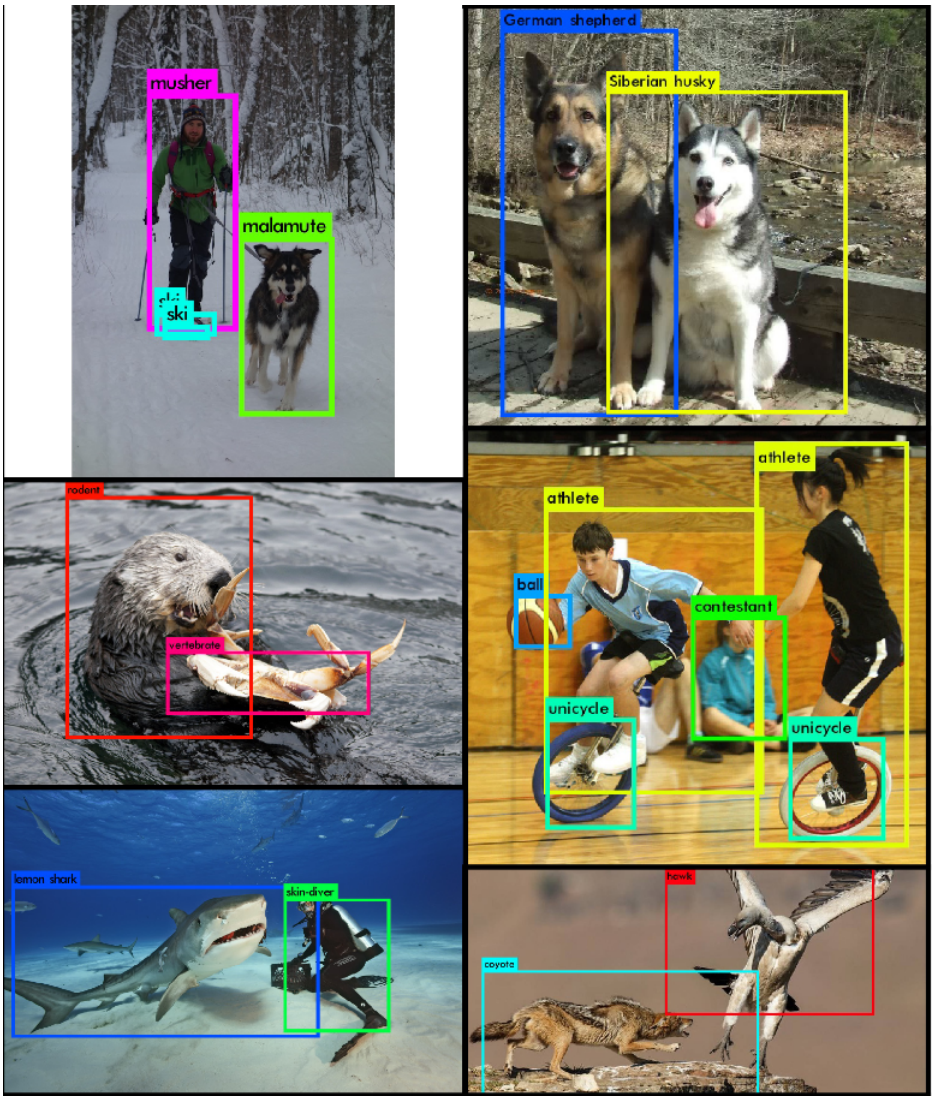


Figure 2.3: **Example of bounding box tasks:** As seen in the right middle example, two athletes and two unicycles for each instance of these classes have their own separate labels [19].

bottom-up path is completed, starting from the deepest layer, every feature map is upsampled and fused with features of the same size from the bottom-up path. These generated features are the top-down path of FPN architecture. Feature maps from the top-down layer can be fused or used separately for further operations.

2.2.3 Depthwise seperable convolution

Standard convolutions can become computationally expensive, especially when dealing with a large number of channels in the feature maps. Depthwise separable convolutions address this challenge by offering a more efficient alternative. Depthwise separable convolutions break down traditional convolution operation into two operations: depthwise operation illustrated in Figure2.7, one channel convolutional kernel applied to every channel of the input feature map, and pointwise convolution 1x1 kernel that produces the desired output channel [23]. Compared to traditional convolutional operations, depthwise separable convolutions have fewer parameters but can achieve remarkable performance.

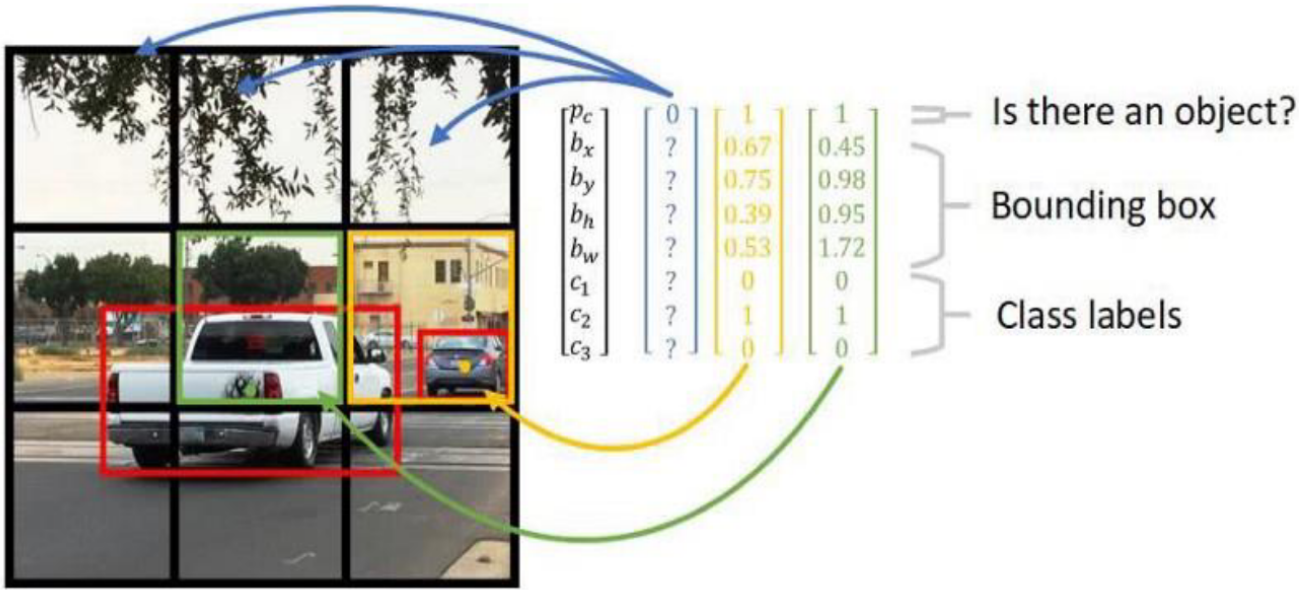


Figure 2.4: **Explanation of bounding box labels with 3 class examples:** The first value represents the presence of an object. If no object is presented, it is marked with 0, and other values of that box don't participate in the loss calculation process. The next four values represent the position of the bounding box in the grid and the size of the bounding box. The last three values represent which class that object belongs to [2].

2.2.4 BiFPN

BiFPN is an FPN-like novel architecture first used in EfficientDet architecture [3]. It is inspired by PANet [30] and Nas-FPN [31] architectures, which are shown in Figure2.8. PaFPN introduced an additional bottom-up path to FPN to robust performance with bidirectional information flow. Meanwhile, Nas-FPN introduced new fusing methods to increase performance. In the work EfficientDet, they took these two features from the mentioned FPN architectures, proposed BiFPN architecture and paired it with EfficientNet architecture [41]. BiFPN takes a set of hierarchical feature maps with the same number of channels as input. These maps represent different resolutions of the input image that is processed by backbone architecture. First, the smallest feature map is upscaled by interpolation, then multiplied to learnable coefficients, and then fused with the multiplication of the next input feature map with another set of learnable coefficients. This fusion process involves summation followed by a depthwise separable convolution. Then, the output of the last operation was upscaled again, multiplied with learnable coefficients, and summed with multiplication with the next input. This operation is repeated iteratively, upsampling and fusing the smaller maps with the progressively larger ones until the largest feature map is reached. For the top-down path starting from the largest output, it downscaled interpolation multiplied with learnable weights and fused with the multiplication of the next smaller input layer and learnable weight and multiplication of the sized feature map generated during the bottom-down path. This operation goes on till the smallest layer. For generating the smallest output, the multiplication in upscaled production and parameters fused with the multiplication of the smallest input and parameters.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 2.5: **Building blocks of ResNET:** Each building block contains convolution operations [34].

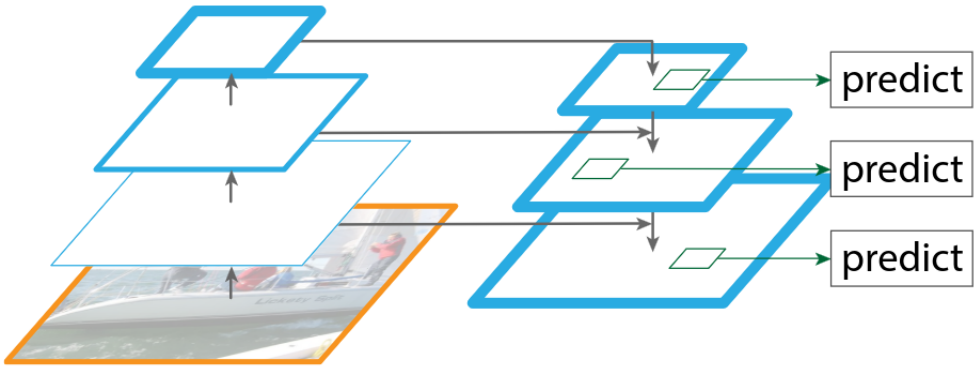


Figure 2.6: **Simple representation of FPN:** The bottom-up map on the left side of the image and the top-down map on the right side. Each feature map in the top-down path can either be used for output or fed to the next FPN layer [32].

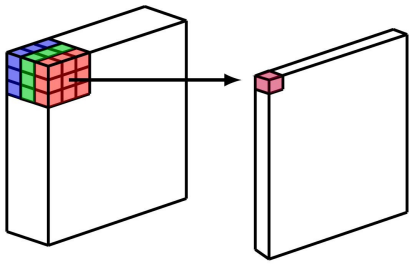
2.3 Gradient accumulation

Traditionally, during backpropagation, the gradients of the loss function are calculated with respect to the model's parameters for each mini-batch of data presented to the network. These gradients are then used to update the parameters in gradient descent. However, gradient accumulation takes a different approach. Instead of updating the parameters after each mini-batch, the gradients are accumulated over several mini-batches [35]. After a predefined number called accumulation steps, the accumulated gradients are used to update the model's parameters in a single step. This method helps to simulate the effects of bigger batch sizes by accumulating them.

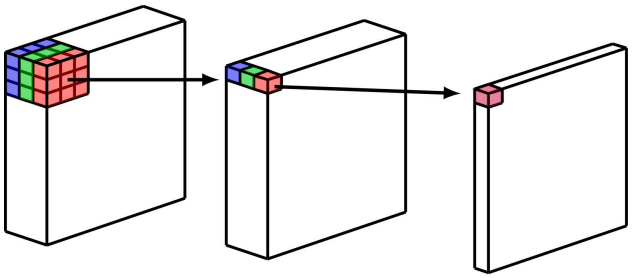
2.4 Multi task learning

2.4.1 Traditional

Multi-task models have been interested in multiple domains for a long time, even before the advent of deep learning. In a 2005 research paper [8] researchers proposed the "Obj cut" method



(a) Conventional Convolutional Neural Network



(b) Depthwise Separable Convolutional Neural Network

Figure 2.7: **Comparison of traditional convolution and depthwise separable convolution:** In traditional convolution, the channel size is changed in one process by applying multichannel kernels. On the other hand, in depthwise separable convolution, a single channel kernel is first applied to each channel individually, followed by a 1x1 multichannel kernel to get the final result [26]

.

for performing segmentation and object classification on a dataset containing images of cows and horses. The researchers achieved this by extracting pictorial structures and applying probabilistic methods. Similar research conducted in the same period focused on classification and segmentation tasks [29]. Computational statistical methods like expectation maximization and algorithms based on graph theory were used to perform these tasks on datasets containing human faces and cars. However, both methods rely on manual feature extraction, it may not be suitable for autonomous driving tasks.

2.4.2 Symmetric DL

After advances in deep learning studies, several types of research have been conducted on multi-task models that utilize CNN and NN. For example, research that performs depth prediction, surface normal estimation, and semantic segmentation simultaneously [22] or Hyperface algorithm that specializes for human faces and is able to perform face detection, landmarks localization, pose estimation and gender recognition tasks [21]. Both architectures have satisfying results, and with the correct hardware, it is possible to benefit from these architectures for real-world applications. However, these architecture rely on symmetric labels for training.

2.4.3 Asymmetric DL

One promising research in asymmetric multi-task learning is asymmetric deep learning training in 2018 [36]. In this research, the authors used the knowledge distillation method. During

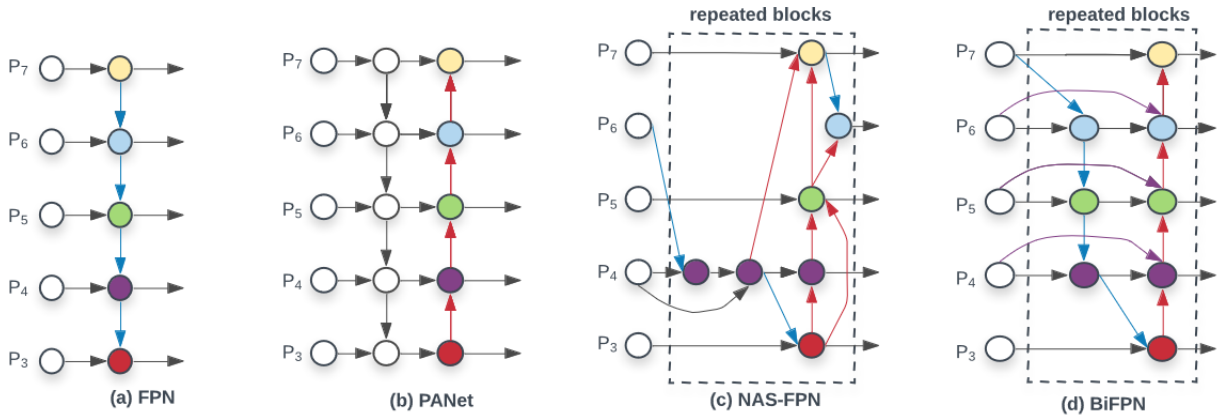


Figure 2.8: **BiFPN architecture and architectures inspired that architecture:** From left to right, FPN, PANet, NAS-FPN, and Bi-FPN. NAS-FPN implemented different types of connections, while PaNET added an additional bottom-up path. BiFPN, on the other hand, implemented advanced connection techniques similar to NAS-FPN and added a bottom-up path [3].

training, if a specific data sample lacks a label for one of the tasks, the sample is still fed into the pre-trained, single-task network for that specific task. Using this method, they were able to apply semantic segmentation and depth estimation to the autonomous driving dataset. However, this method incurs additional computational cost during the training process because of additional DL architectures.

3 Methodology

3.1 Dataset

As mentioned above, semantic segmentation, object detection, depth estimation, and steering angle estimation are among the essential tasks for autonomous driving; the generated results of these tasks contain good information about the surrounding environment and can be fused for higher representation. One of the thesis aims is to compare symmetric MTL with asymmetric MTL. Therefore, the objective dataset should contain all labels, and A2D2 is one of the few autonomous driving datasets that meet conditions for label existence [10].

The automotive manufacturer Audi AG published A2D2 in 2020. The sensor suite was set up on an Audi Q7 e-tron. It consisted of six cameras, five LiDAR sensors that surrounded 360 degrees of the vehicle, and a bus gateway, which recorded the trip information, like vehicle speed, steering angle degree, brake pressure and more.

Data was recorded in three cities south of Germany: Gaimersheim, Munich, and Ingolstadt. The data was collected on urban, suburban, and highway roads during clear weather. On official source data is split by labels. For the thesis ”3D Bounding Boxes” branch and ’Bus Data’ from ”Semantic Segmentation” downloaded, data is published in tar files. The extracted directory, ”3D Bounding Boxes,” consists of directories named by date. Every folder with a date name consists of folders named ’camera,’ ’lidar,’ ’label’, and ’label3D’, which contain image, LIDAR information, semantic segmentation, and bounding box information, respectively. The extracted file from ’Bus Data” also follows a similar folder structure.

3.1.1 Camera images

Camera images are RGB images with 1920 x 1208 resolution. However, images are resized to 1024 x 1024 resolution to decrease computational cost while preserving enough quality to capture features of far-away objects. During train and validation time before feeding to the model, images are normalized to range 0-1 for all channels by dividing all values by 255. Such normalization ensures values are more manageable and prevents specific input features from dominating the model.

3.1.2 Semantic segmentation labels

Semantic segmentation labels are PNG images of the same size as the input. There are 55 possible colors, but not all colors represent a class; there are 38 classes. The reason for that difference is that the dataset maintainers decide to make objects that share the same class and boundaries distinguishable on segmentation labels, which can be observed in the sample shown Figure3.1.However, for aimed tasks, there is no need for such distinction. Therefore, the replicates of most presented classes in Figure 3.2 are ignored.

PNG images are unsuitable for deep learning, and converting these images to multichannel masks is resource-intensive. Therefore, all segmentation images are converted into NumPy arrays and stored as files in a directory called 'multi_label' in directories named by date.

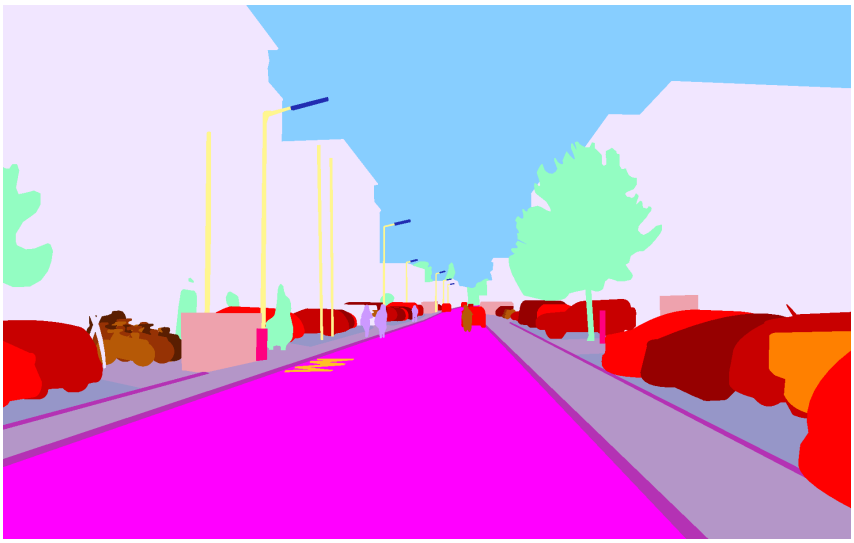


Figure 3.1: **Semantic segmentation label from the dataset:** As seen in the sample, cars are annotated differently, even if they are classified as the same object. However, that does not indicate that the labels belong to different types of segmentation tasks. Annotation differences for the same object class only exist if the objects in view share a boundary and annotations are repeated through the other instances of objects presented in view.

3.1.3 Depth labels

The depth information comes from LIDAR sensors placed in front of the vehicle, which can capture distances up to 100m. Dataset providers shared code for marking depth points with colors on original images. The provided code first takes LIDAR values, which represent distance in meters, and brings them to the 0-1 range by dividing them into maximum distances. Then, it represents these normalized distances as colors on the original input image, like in the example in Figure3.3a. For the thesis, two main changes were made to the original code: As a base, instead of input images, a black empty background with the same size as the original input has been created, and instead of normalized distances in the color scheme, they multiplied with 255 and stored in grayscale images in PNG files, one stored image shown in Figure3.3b. However, during the training process, these images were normalized in the 0-1 range for the same reason as the input images.

3.1.4 Bounding box labels of dataset

Both 2D and 3D bounding box annotations were stored in the same JSON files. There are 14 bounding box classes in the dataset, all shown in Figure3.4. Model output class probabilities and loss include these probabilities for calculations, but metric evaluation focused on the quality of the 2D bounding box. 2D bounding boxes are represented with coordinates of 4 corners of boxes in the image's coordinate frame. Converting these four values to YOLO format in data loader is suitable, as processing this process does not require high resources. For the implementation, a grid size of 8x8 was used because this number is closer to the implemented grid size in the original paper, and when it divides the height and width of the input image, it creates an integer.

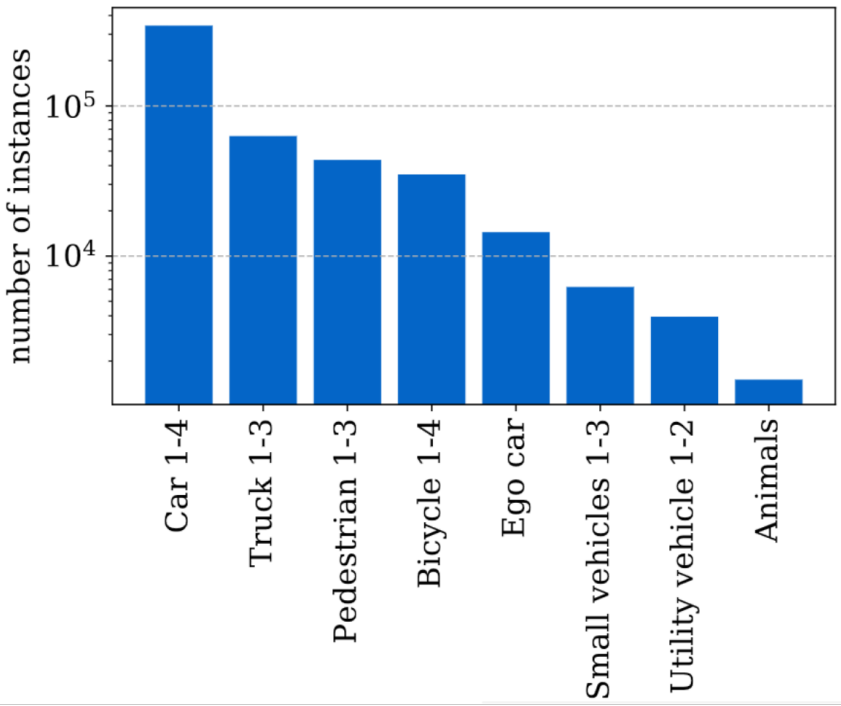


Figure 3.2: **Number of instances of classes in the dataset for segmentation task [12]**

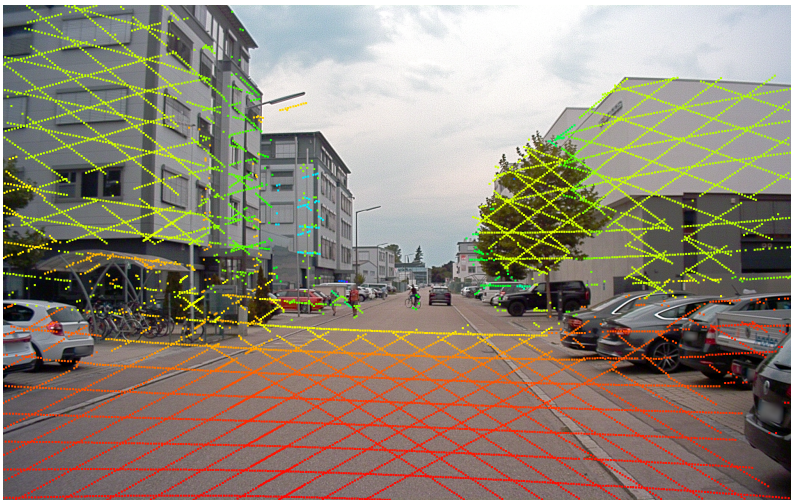
After some data analysis, it appears that there are a maximum of 9 objects for one grid in the dataset. Therefore, it implemented ten bounding boxes output per grid to capture all those objects. Considering that every bounding box has five values to represent confidence and the bounding box attributes and 14 values to represent class probabilities, there are 190 values for one bounding box output. Thus, considering there are 8x8 grids, the size of the bounding box output is 190x8x8.

Unfortunately, the general quality of 2D boundary box labels is low compared to other bounding box datasets. Sometimes, bounding box labels are missing even if there are objects present in that grid, like the case in Figure3.5b or for small classes like pedestrians or bicycles, the location of 2D bounding boxes is not accurate, like in Figure3.5a.

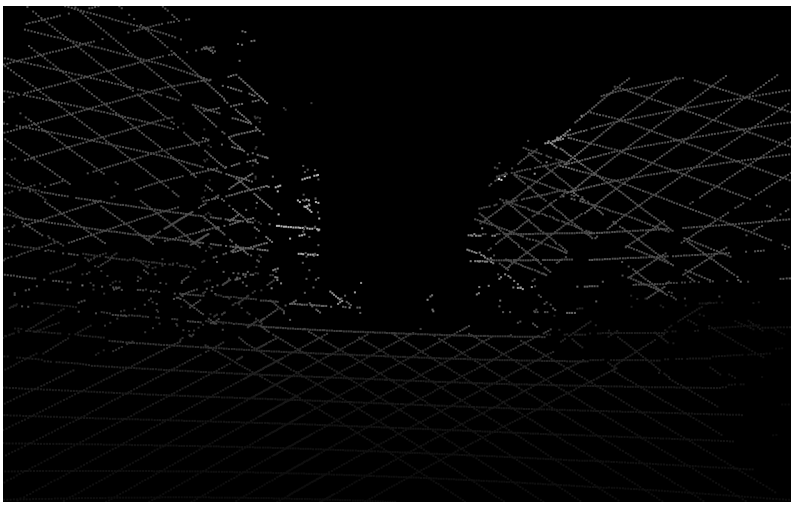
3.1.5 Steering angle labels of dataset

The dataset maintainer published bus gateway data of each date in one big JSON folder. Having all bus gateway information in one file is inconvenient during training when loading data, opening, and closing big files is resource-intensive. Initializing these files during the initialization step is also not convenient, as keeping a file that size may cause instability during the training process. For that reason, bus gateway information for each sample is extracted and stored in separate files. During training, these small files were opened, and the data was loaded into the GPU and then closed.

Each camera image has one dictionary containing bus data with the same name. Each dictionary has 21 data points for all types of bus gateway data. However, these data points are not synchronized with the camera image. The timestamp of camera images is in the range of the first and last timestamp in the contradicting bus data dictionary; however, there is no exact match. After looking through the data, it appears that the last 10 timestamps in each dictionary are always greater than the timestamp of the camera image; therefore, each dictionary's last 10 data points are loaded into the GPU.



(a) Before



(b) After

Figure 3.3: **Changes on depth labels:** Top: depth labels generated by code provided by dataset maintainers; Bottom: depth labels after mentioned changes made on code.

The steering angle in A2D2 is expressed in two variables per sample. The first variables store information on the direction of the front, while the second values store the positive turning angle of the vehicle's front wheels. By simple operations, left turns are represented by negative steering angles, and right turns are positive. In addition, values are normalized and brought to the 0-1 range.

3.2 Research Tools

The chunk of the A2D2 dataset downloaded during research is about 50GB in a .tar file. Pre-processing and training this amount of data is time-consuming on consumer-grade PCs. For that reason, rocket clusters from Tartu HPC were utilized for research [50]. This cluster provides 1 PB of storage space and several computing unit options, enough to store and process datasets. Instances with 64 CPU cores with Nvidia-A100 GPU were utilized for all training process instances.

The code base for research was created using the PyTorch framework [51] in Python pro-

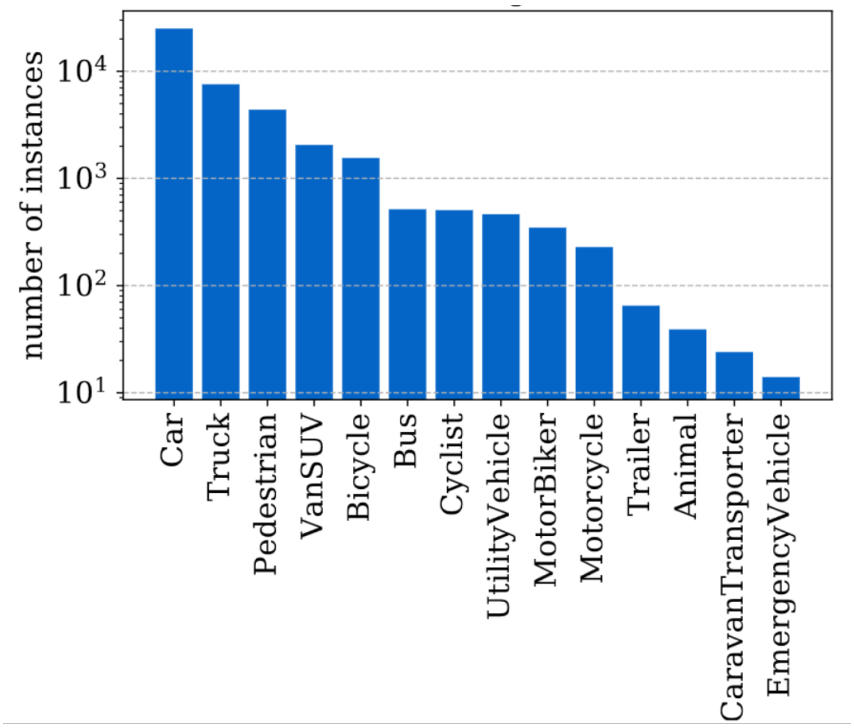


Figure 3.4: **Number of instances of classes in the dataset for bounding box task [12]**

gramming language. OpenCV [52] and Pillow [53] frameworks are utilized during different image processing operations. NumPy [54] is used during dataloading and some image processing operations. Matplotlib [55] is used for visualization.

In codes for STL models [20, 57–59], scale-invariant loss [56] and BiFPN [3] were taken from already implemented projects but modified according to the needs of the thesis.

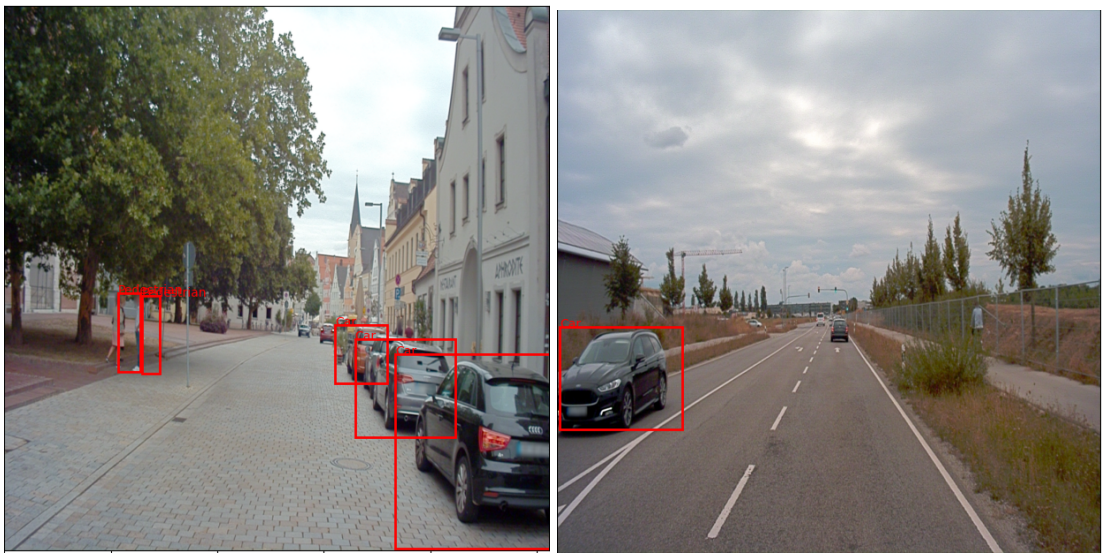
During debugging, ChatGPT 3.5 [60] was used for debugging errors in code, and Gemini [61] and Grammarly [62] were used for improving the quality of writing of paper.

3.3 Architecture

As the first step, input images are fed through ResNet50 architecture, feature maps after every stage stored, conv_1 being the exception. The end backbone produces four feature maps with different widths, heights, and channel sizes. Depthwise separable convolution operations were applied to all four feature maps with an output channel size of 256. The resulting feature maps will be inputted into the BiFPN layers, which are illustrated in detail in Figure3.6

In EfficientDet, they did not use BiFPN architecture for multi-task learning; instead, they combined generated feature maps by BiFPN and fused feature maps used for the object detection task. However, in this thesis, it is implemented for multi-task learning. As mentioned, BiFPN architecture can provide a bi-directional flow of information, and this feature may be beneficial for multi-task learning. For that reason, each generated output of BiFPN is used for different tasks so that during backward propagation, signals from each task would extract unseen features from another task, improving the model's generalization and accuracy.

Depth and semantic segmentation are different tasks that operate at the pixel level. Depth is a regression task, whereas semantic segmentation is a classification task where the output should have the same width and height as the input image to represent each pixel. There is research that shows that deconvolution layers may cause repeated patterns in output [27]. To



(a) (b)

Figure 3.5: **Low quality 2D bounding box labels:** Left: locations of bounding boxes on pedestrians are off. Right: the car closer to the ego vehicle was marked. However, the further car was not labeled.

avoid that problem, both tasks were placed after the largest feature maps to avoid deconvolution operations that change the size. As a result, for depth estimation two, and for semantic segmentation three deconvolution layers were used.

The smallest output is used for steering angle prediction, as it stores the highest spatial information, which is beneficial for this task. In addition, as the output size for this task is relatively smaller than another task, this decoder will use fewer parameters.

The output size is much smaller than the input feature map size for the bounding box task. Therefore, the decoder using only convolution operations was sufficient to process the feature map and get the desired output.

All decoders consist of 4 layers. The steering angle decoder is the only decoder that utilizes linear layers. As for the activation function, LeakyReLU was utilized for all decoders.

3.4 Loss and metrics

3.4.1 Cross entropy loss

As discussed in previous paragraphs, the segmentation map consists of stacked maps on top of each other. The values of one pixel across every map can be considered a probability distribution that presents the probability of a pixel belonging to a class. That is why implementing cross-entropy loss, the loss function that measures the difference between probability distributions, is appropriate for that case [13].

$$\begin{gathered} q_i - \text{prediction} \quad p_i - \text{target} \\ -\sum_{i=1}^{C} p_i \log(q_i) \end{gathered} \tag{3.1}$$

Luckily, this loss function exists in the PyTorch framework. They apply softmax operation for the output of the network and then implement the loss function. Doing so means there is no

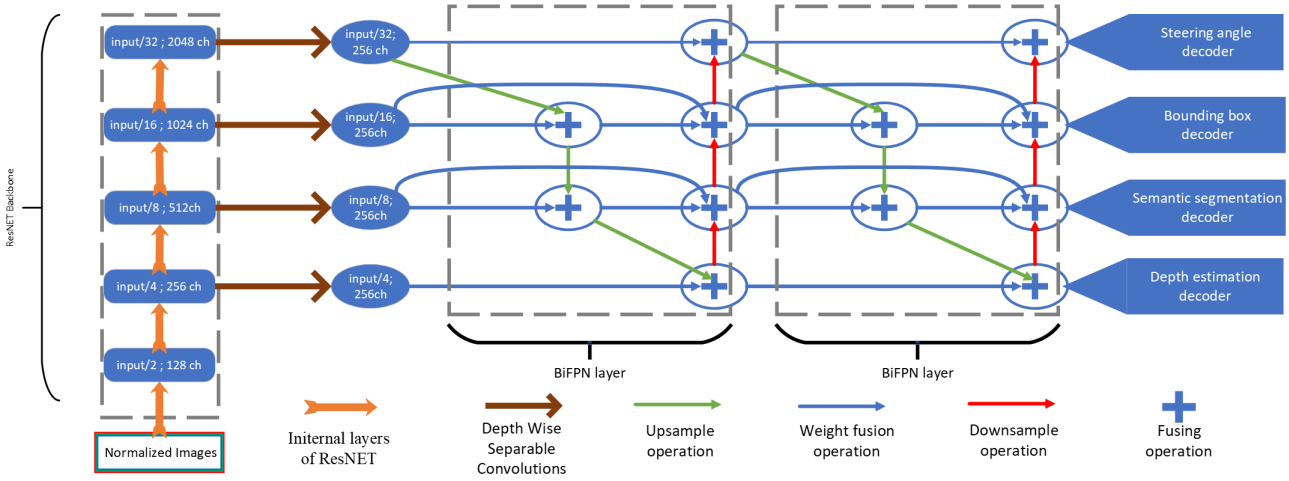


Figure 3.6: **Structure of MTL architecture:** the orange arrows show the operations of ResNET architecture. Rounded squares and ellipses demonstrate feature maps; the text inside maps shows how much height and width decreased from input and channel size. The brown arrows represent depthwise separable convolution operations, which convert feature maps from different channel sizes to the same channel size. The green arrows indicate the downsample operation, which involves upscaling the feature map by interpolation and multiplying trainable coefficients. Similarly, the downsample operation downsamples by interpolation and applies learnable coefficients, as the red arrows demonstrate. Weight fusion is the operation of multiplying the feature map with learnable coefficients. The '+' sign represents the fusion operation of inputs that are directly connected to that sign. Fusing operation sum inputs and apply depthwise seperable concolution to sum.

need to implement softmax during inference time.

$$-\sum_{i=1}^{C} \log \frac{\exp(x_n, y_n)}{\sum_{j=1}^{C} \exp(x_n, y_n)} \quad (y_n \neq ignoreindex) \tag{3.2}$$

3.4.2 Mean absolute error

Mean Absolute Error is a simple loss function that calculates the difference between the target and prediction.

$$\frac{1}{N} \sum_{i=1}^{N} |y_i - \hat{y}_i| \tag{3.3}$$

This error was used to calculate the loss function of steering angle prediction during the training process. It is also used as a metric, but to make it easy to comprehend results de-normalized during metric calculations.

3.4.3 Scale invariant loss

For depth estimation tasks, the relations of pixels is the most crucial aspect. For example, being able to understand that one object is closer than another object is more important than knowing the average distance of both objects. That point is the reason for utilizing scale-invariant loss [11]. Unlike the root mean square error scale invariant loss is able to keep relations among

pixels.

$$
\begin{aligned}
& y_i - prediciton - \hat{y}_i - target \\
& d_i = \log y_i - \log \hat{y}_i \\
& \frac{1}{n}\sum_{i=1}^{n} {d_i}^2 - \frac{1}{n^2}\left(\sum_{i=1}^{n} d_i\right)^2
\end{aligned}
\tag{3.4}
$$

The loss first calculates the difference of logarithms of the target and prediction for each pixel. These relative errors are then combined on a global scale to obtain the final result.

Scale-invariant loss does not exist among loss functions of the PyTorch framework. However, other researchers have already implemented this loss function for medical applications [56]. Therefore, the loss was taken from their public repositories and modified. The loss is only calculated for pixels where depth information exists.

3.4.4 Intersection over union

Intersection over union is the ratio of the intersecting area between prediction and target over the union of prediction and target.

$$
\frac{\text{Predicted area} \cap \text{Target area}}{\text{Predicted area} \cup \text{Target area}}
\tag{3.5}
$$

IOU is unsuitable during training because it does not capture necessary information for the model, like direction or size. However, it is suitable for metrics as they are easier for humans to understand. In addition, IOU is used during the loss calculation of bounding box prediction for a process called non-max suppression. During training, the network generates multiple bounding boxes for a single grid. Non-max suppression decides which bounding box to use for loss calculation by comparing the IOU values of the predicted boxes and target label and selecting the box with the highest IOU for loss calculation.

3.4.5 Bounding bos loss

Output for the bounding box task is arranged in YOLO style. Therefore, the loss function is also implemented in the same way in the original work.

$$
\begin{aligned}
& \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \\
& + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] \\
& + \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{noobj} (C_i - \hat{C}_i)^2 \\
& + \sum_{i=0}^{S^2} 1_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2
\end{aligned}
\tag{3.6}
$$

The loss function is the sum of several summation operations. The first summation calculates the error between the center of the target and the predicted boxes. The second summation operation is for the size of the boxes. The third and fourth summations calculate confidence

error, the third summation calculates the error for selected predicted boxes, and the fourth summation calculates confidence error where there is no object. Finally, the last summation process calculates class error. All summation operations are regularized by λ coefficient.

This loss function did not exist in PyTorch, which is why this loss was implemented from scratch by following the original paper. Some minor changes were made during implementation to make the process faster, but the main principle was kept the same.

3.4.6 Accuracy

In the context of machine learning, accuracy means the percentage of true prediction among all labels.

TP - True Positives	TN - True Negatives
FP - False Positives	FN - False Negatives

$$\frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{3.7}$$

Accuracy does not contain information confidence of the model's prediction, which is why it is unsuitable for the training process. However, accuracy is easy to comprehend, so it is implemented as a metric for semantic segmentation.

3.5 Training method

Single-task models and the multi-task model with symmetrical labels were trained using traditional methods. Static batch size was defined, and data was loaded to GPU in batches, forward passed, and backpropagated. The weights of architecture were updated right after backpropagation.

However, the method used to train the multi-task architecture with asymmetric labels vastly differs from the traditional method. This method is implemented in UberNet research [17]. The method rejects using batches or mini-batches instead of there are n+1 counters for n tasks. For every task decoder, one counter is assigned, and one remaining is assigned to a shared part of the architecture. For each training step, a single input and its corresponding label are loaded onto the GPU. Input feedforward through the network and loss is calculated for only presenting the label and output of the contradicting label. The backpropagation happens from that calculated loss function. This backpropagation only affects the decoder of the task with existing labels and the common parts of the network. After that process, the value of counters corresponding to the network's backpropagated part increases. The weights of the network are updated partially. If the counter corresponding to part of the network reaches the threshold value, the weights corresponding to only that part of the network are updated, which can be seen in Figure3.7. It is also worth mentioning that in the PyTorch framework, different optimizers should be used for different parts of networks, which makes it possible to use different learning rate for different part of the network.

However, depending on train parameters, this method may introduce stale gradients. As seen in the Figure 3.7 for the 'k+1'th step, after one of the task decoders gets updated, the backpropagated gradients from that decoder are accumulated with gradients that backpropagated from the previous state of the same task decoder. Another example is during the 'n+1' step in the Figure3.7. After the gradient of the shared part of the network is updated, some accumulated gradients remain on the decoder before the k th and n th training process. However, with correct train parameters, this feature may improve the generalization capabilities of the architecture.

3.6 Experiment setup

During the implementation of STL, all input samples with corresponding labels to the task were used. For symmetric MTL, the same input samples with all labels were used. In the case of asymmetric MTL, all samples were loaded into the model 4 times, each time with a different task label. The purpose of loading all input samples for each task is to ensure that no specific unseen advantages are given for any of the tasks. For example, 1/4 of train data is selected for steering angle prediction, but if this data does not contain left or right turns, it may cause an 'unfair' advantage for the train. However, there is still an imbalance because the model feeds forward four times more input data as the backbone gets backpropagation at every step. It experiences four times more backpropagation for asymmetric MTL. To balance it, asymmetric MTL trained four times less epoch than another training process.

As one loss function is present during STL, there is no need to balance loss functions. However, symmetric and asymmetric MTL loss functions are balanced to avoid task bias. Before balancing loss functions, each loss function is stored in separate values for logging purposes. Doing so makes it easier to compare individual loss functions to STL during the train. Calculated loss functions were divided by coefficients before backpropagation to balance loss functions. For initial coefficients, loss functions of STL are taken as reference, and loss functions of MTL are divided so they become as close to each other as possible. However, during the later stage of fine-tuning, this coefficient changed depending on the results. In the case of learning rates, the same learning rate value is initially assigned for all network parts and tuned to the next stages of the experiment.

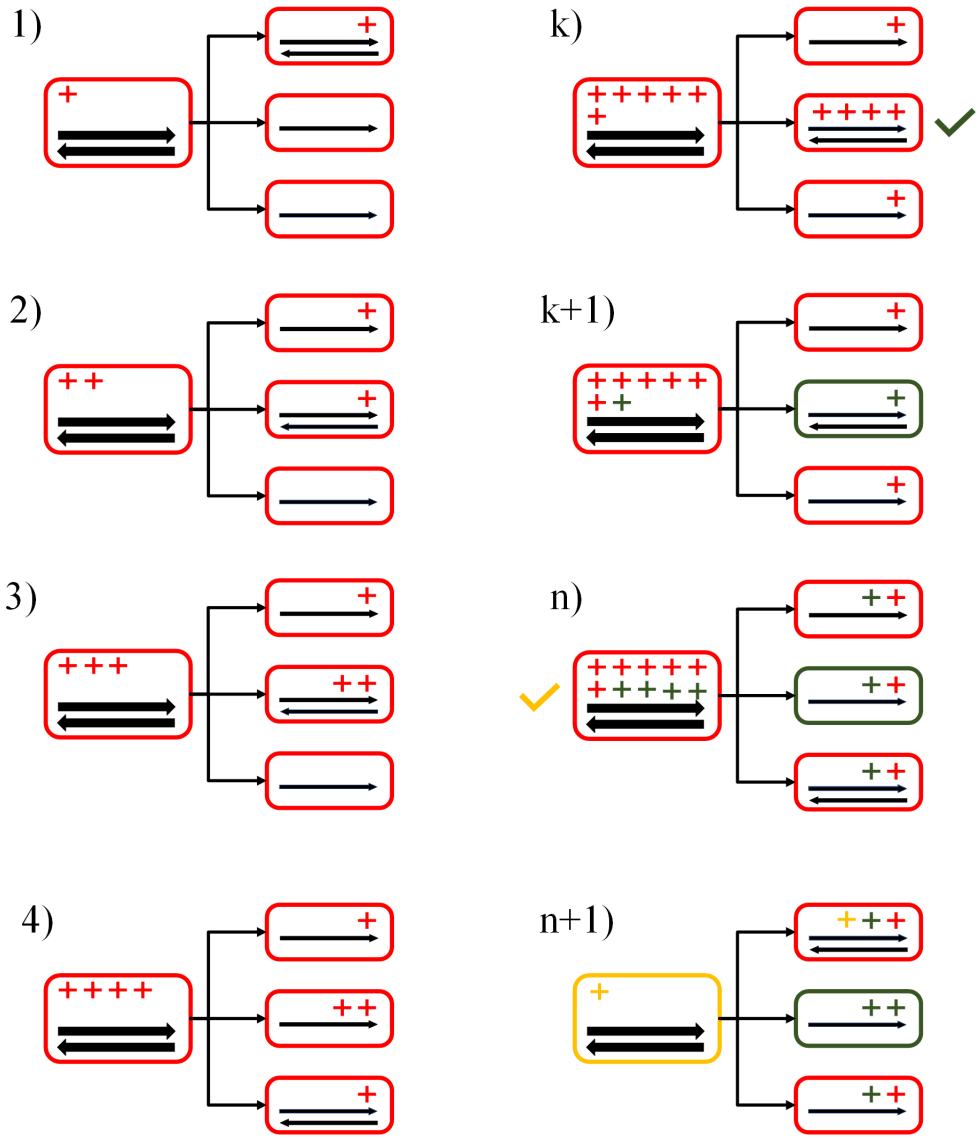


Figure 3.7: **Train process on generic simplistic three task MTL network:** The MTL network in this example consists of 4 parts: Bigger rounded squares demonstrate the backbone or shared part of the network, smaller rounded squares for task decoders. Arrows pointing to the right indicate the feedforward process, while those pointing to the left illustrate the backpropagation process. '+' signs used to show accumulated gradients. Check marks at the side of part of the networks indicate that part of the network reached the accumulation step. The colors associated with ticks show the part of a network that was updated and the calculated gradients after that update.

4 Results

	Asymmetric MTL	Symmetric MTL	STL	MTL trained on single task
2D Bounding box IOU ↑	27%	19.09%	19.43%	2%
Segmentation accuracy ↑	90.40%	82.43%	90.46%	92.99%
Steering angle error(degree) ↓	0.5°	0.08°	0.019°	0.016°
Depth error(m) ↓	7.04m	7.02m	7.12m	6.76m

Table 4.1: **Metrics of architectures from the validation set:** An upward arrow indicates that higher is better, while a downward arrow indicates the opposite.

According to the results of metric calculations in Table4.1, Architecture trained in STL performed better than other training processes, except for being a 2D bounding box task. Interestingly, during training, the difference between validation loss asymmetric MTL and MTL architecture trained on a single task was smaller than others, as seen visually in Figure4.1. First, look at the metric table. Asymmetric MTL shows promising results. It is important to remember that in the case of real-world applications, MTL architecture has a smaller footprint compared to the aggregation of STL architectures. Details regarding network sizes are shown in Table4.2.

During the training process, individual loss functions of tasks demonstrated different characteristics and decreasing rates. For example, the STL model for the depth estimation task got an average validation loss of for the first epoch 1.003 and for the last epoch 0.9396, meaning that there is approximately a 6% decrease during the training process. Another example is the STL model UNet for the segmentation task. It gets 1.389 and 0.3263 average validation loss for the first and last epoch. The largest change for validation loss among tasks is steering angle loss. It gets 0.6 average validation loss for the first epoch and around 0.001 during the train, meaning it decreased 600 times during the train. It is important to remember that these decrease rates are a general characteristic and do not imply that the loss decreases sharpest is best performing.

When checking the visual results for the validation set in Figure4.2, it is seen that the archi-

Model	**Task**	**Parameters**
PilotNet	Steering angle prediction	140,696,494
DenseDepth	Depth estimation	44,322,689
UNet	Semantic segmentation	31,040,363
YOLO V1	Bounding box prediction	226,607,102
Aggregation of models	All above	442,666,648
Multi task	All above	61,656,914

Table 4.2: **Number of parameters for each model**

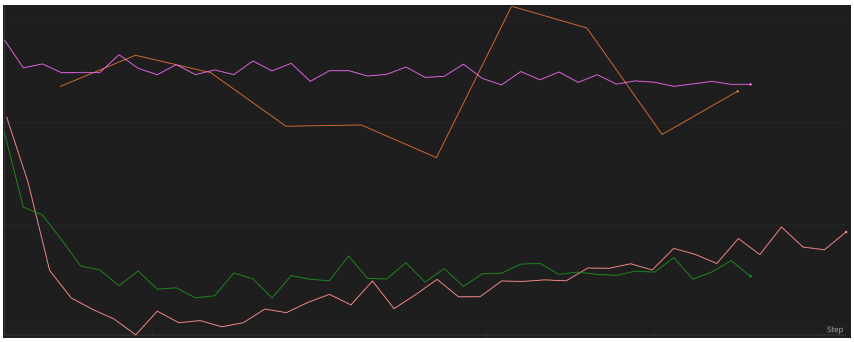


Figure 4.1: **Bounding box validation loss:** Pink-STL, Purple-Architecture-STL, Green-Symmetric STL, Orange-Asymmetric MTL

tecture trained in the STL method for the bounding box task is unsuitable for any application. It either outputs the same pattern all over the image or outputs nothing. Meanwhile, STL architecture predicted many false positive bounding boxes for most validation samples. MTL architecture trained symmetrically demonstrated a better fit on validation data than the STL. MTL architecture trained asymmetrically demonstrated surprising results by detecting objects not labeled on the dataset but still exist in the image view. This may indicate that knowledge from one task of asymmetric MTL improves the performance of other tasks.

When looking at the visual results of segmentation tasks in Figure4.3, it is seen that with symmetric MTL architecture being an exception, all models successfully captured the boundary lines of most objects, and MTL architecture trained on a single task. The symmetric MTL model can still detect some pixels of objects; however, it fails to capture the shape of objects. For MTL trained asymmetrically, there are noisy pixels for sky and buildings; however, the model shows significant results in capturing small objects like traffic signs and light poles compared to other models.

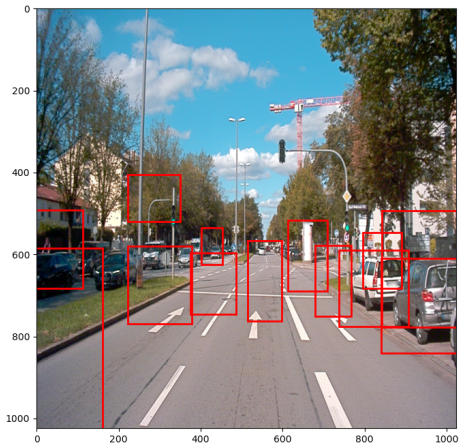
Because ground truth labels on depth estimation are sparse, it is challenging to evaluate visually. However, specific observations can be made in Figure4.4. STL models tend to have horizontal repeated patterns for most of the samples. However, in the case of the STL DenseDepth model, it is possible to see that a depth map can preserve the shape of objects to some extent. Symmetric MTL trained model tend to have repeated patterns instead of horizontal like there are repeated square patterns all over the output. Compared to others, the asymmetric MTL model has fewer repeated patterns on output images and can capture objects' shapes.

Compared to other tasks in real-life applications, the steering angle task implemented as the primary control mechanism tends to accumulate loss. This means that for an error caused at n_{th} timeframe, that error will affect the vehicle's state for the next timeframe. If this error is repeated, it can cause the vehicle to get out of the way. For that reason, simulations are more effective ways to evaluate the performance of the steering angle task. Unfortunately, for A2D2, the publishers have not provided any kind of instruction for simulations.

(a) Ground truth

(b) Asymmetric MTL

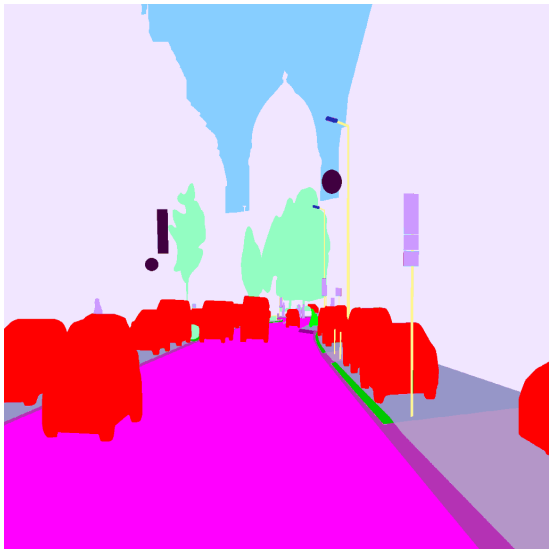
(c) Symmetric MTL



(d) Single Task Architecture

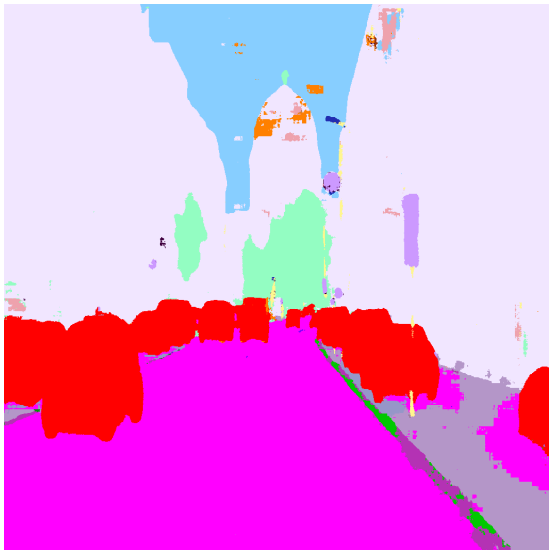
(e) MTL trained on single task

Figure 4.2: **Visual result for bounding box detection task**

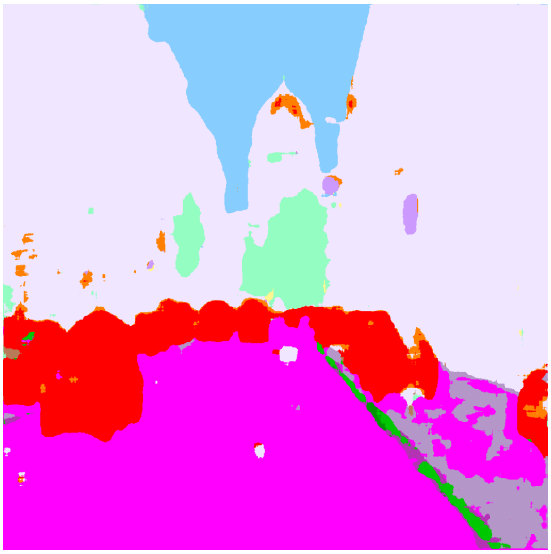


(a) Ground truth

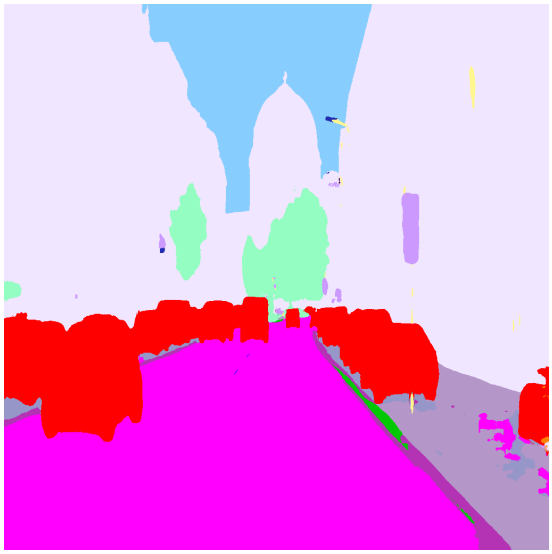
(b) Input Image



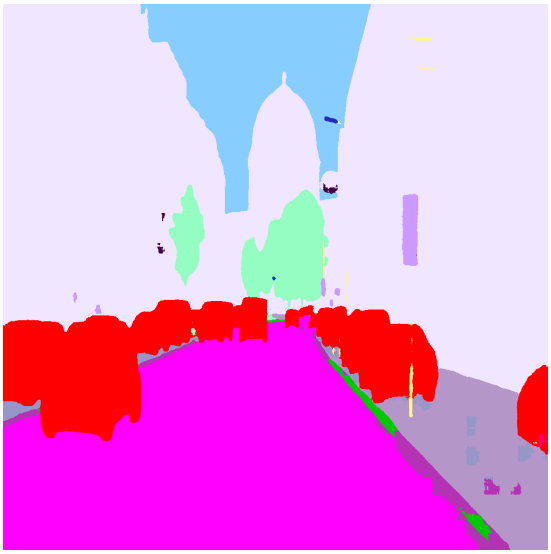
(c) Asymmetric MTL



(d) Symmetric MTL



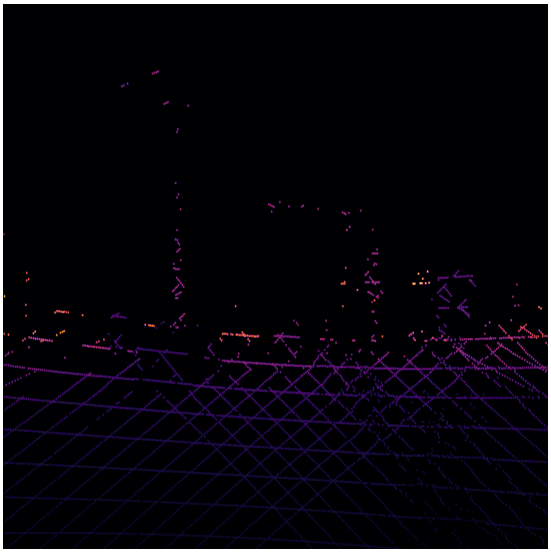
(e) Single Task Architecture



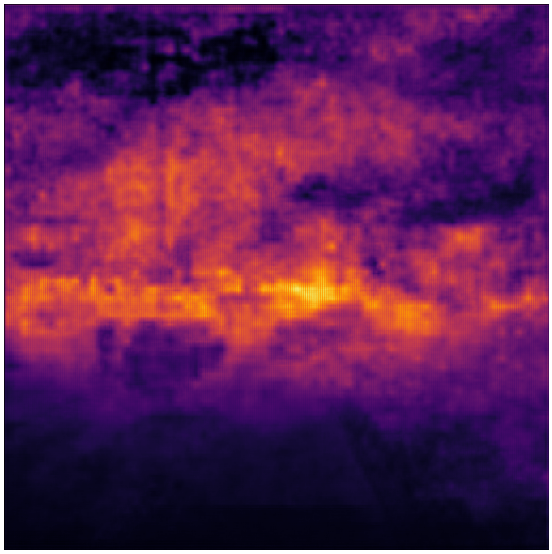
(f) MTL trained on single task

Figure 4.3: **Visual result for semantic segmentation task**

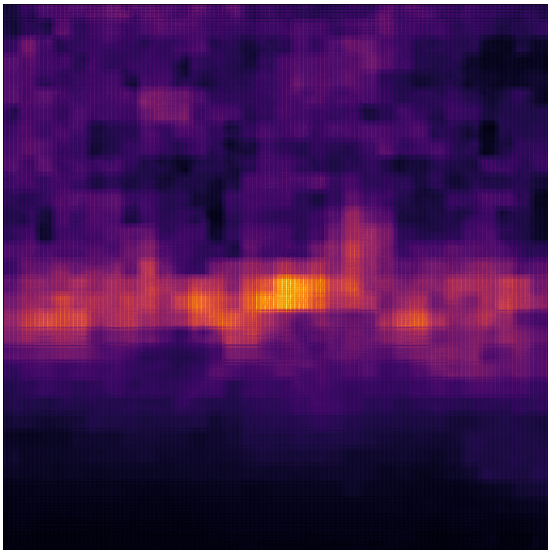
(a) Input Image



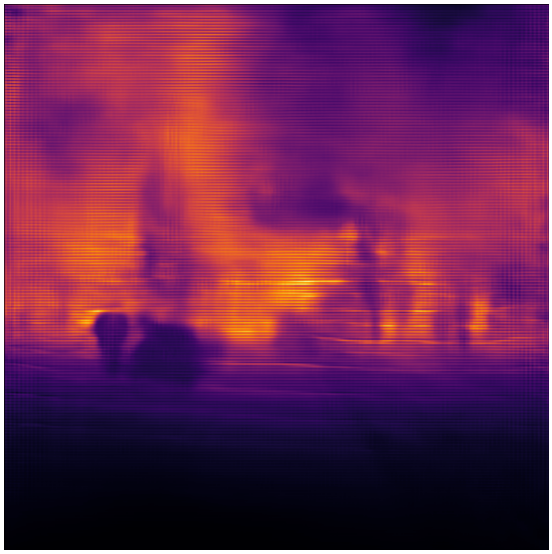
(b) Ground truth



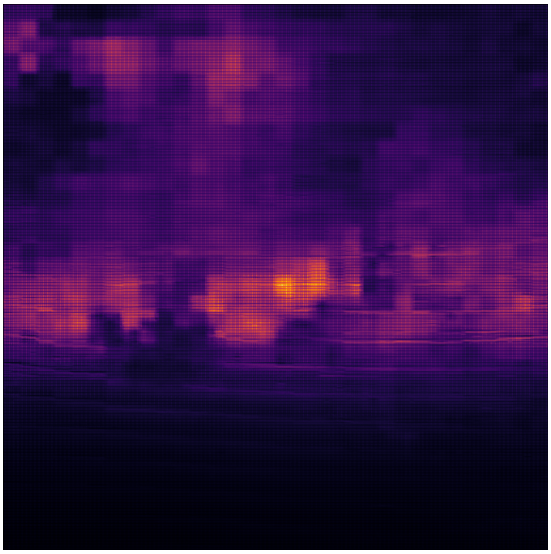
(c) Asymmetric MTL



(d) Symmetric MTL



(e) Single Task Architecture



(f) MTL trained on single task

Figure 4.4: **Visual result for depth estimation task**

5 Discussion

The results for asymmetric MTL are impressive, considering that the architecture has fewer parameters than other models and is memory efficient during training due to gradient accumulation. Asymmetric MTL captured missing items in the dataset for the bounding box task, detected small objects in the semantic segmentation task, and generated less repeated patterns for depth estimation. Thus, the network can be deployed for segmentation and depth estimation tasks because it can extract robust information about the surrounding environment. Semantic segmentation tasks can be used for road line detection [6], while depth estimation can be deployed for safety applications. Asymmetric MTL does not achieve the same amount of success for bounding box tasks as it can be used as a task to support others' tasks, but for mission-critical applications, it still needs to be more accurate. However, considering that asymmetric MTL performed better than other models for bounding box tasks, it most likely will achieve satisfactory results if the dataset had bounding box labels with better quality. In addition to performance during inference, because the implemented train method utilizes gradient accumulation, it is more suitable to train in lower memory environments.

However, there are still some downsides of asymmetric MTL. As mentioned for MTL models, loss functions need to be balanced coefficients used to increase or decrease loss functions, which can be considered train parameters that add complexity to the training process. Specifically, the asymmetric MTL has additional parameters, accumulation steps, and learning rates for decoders and shared parts of the network, which makes the architecture fine-tuning process even more challenging than the symmetric MTL.

5.1 Future works

In this thesis, compared to other tasks, the steering angle task showed the lowest performance compared to other methods. It is most likely not a coincidence that the decrease in the rate of loss function of the steering angle task is different from other tasks. Future research could be conducted to mitigate this problem. One possible solution is rearranging loss balance coefficients after every epoch using average loss functions as a reference. This method also has the potential to decrease the complexity of choosing fine-tuning parameters.

In addition, during that thesis, all images were taken in the same settings. Further research could be conducted to evaluate the effectiveness of asymmetric MTL when trained on datasets with varying camera parameters, such as different focus and placement on vehicles, or weather conditions, such as snow, rain, and fog. This research would provide valuable insights into the generalizability and robustness of the proposed approach in real-world scenarios.

6 Conclusion

In this thesis, MTL architecture, which has fewer parameters than the aggregation of four STL architectures, is trained on the asymmetric dataset to perform four different tasks. In addition to that model, four STL models were trained in a single task dataset, the same MTL model was trained on the symmetric dataset, and the single label dataset for each task. All models were evaluated on single-label validation sets using four different metrics, and visual outputs for all models were demonstrated side by side. Among these tasks, semantic segmentation and depth estimation showed satisfactory results to be deployed in real-life applications; bounding box estimation can be deployed under certain conditions.

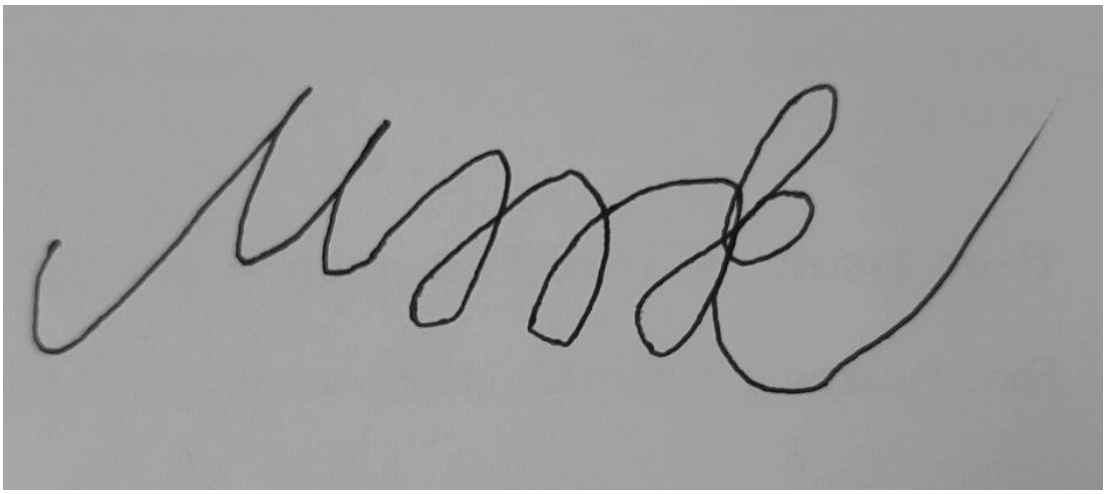
The implemented MTL architecture is more efficient in terms of parameters than other evaluated methods, making it suitable for resource-constrained environments. It is essential to mention that as MTL can perform four tasks more effectively, these four tasks can be used as input for more complex tasks.

Acknowledgements

First, I want to thank my supervisor, Tambet Matiisen, who continuously gave feedback to me. His feedback helped me to overcome my confusion and settle the thesis progress in the correct direction. I also want to express my gratitude to all the lecturers at Tartu University who helped me learn new things and develop myself.

I am also very grateful to the developers and maintainers of all open-source projects, publicly available datasets, and everyone who takes part in the development of artificial intelligence.

I want to thank my friends in Tartu, who supported me emotionally. Lastly, I want to thank my family for raised me to be who I am today.



Bibliography

[1] Almeida, Mario, Stefanos Laskaridis, Ilias Leontiadis, Stylianos I. Venieris, and Nicholas D. Lane. "EmBench: Quantifying performance variations of deep neural networks across modern commodity devices." In The 3rd international workshop on deep learning for mobile systems and applications, pp. 1-6. 2019.
URL: https://dl.acm.org/doi/pdf/10.1145/3325413.3329793

[2] Burić, Matija, Goran Paulin, and Marina Ivašić-Kos. "Object detection using synthesized data." ICT innovations 2019 web proceedings (2019): 110-124.
URL: https://proceedings.ictinnovations.org/attachment/paper/517/object-detection-using-synthesized-data.pdf

[3] Tan, Mingxing, Ruoming Pang, and Quoc V. Le. "Efficientdet: Scalable and efficient object detection." In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 10781-10790. 2020.
URL: https://openaccess.thecvf.com/content_CVPR_2020/papers/Tan_EfficientDet_Scalable_and_Efficient_Object_Detection_CVPR_2020_paper.pdf

[4] Bojarski, Mariusz, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel et al. "End to end learning for self-driving cars." arXiv preprint arXiv:1604.07316 (2016).
URL: https://arxiv.org/pdf/1604.07316

[5] Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation." In Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18, pp. 234-241. Springer International Publishing, 2015.
URL: https://arxiv.org/pdf/1505.04597

[6] Tran, Le-Anh, and My-Ha Le. "Robust u-net-based road lane markings detection for autonomous driving." In 2019 International Conference on System Science and Engineering (ICSSE), pp. 62-66. IEEE, 2019.
https://www.researchgate.net/publication/331646621_Robust_U-Net-based_Road_Lane_Markings_Detection_for_Autonomous_Driving

[7] Abouelyazid, Mahmoud. "Comparative Evaluation of VGG-16 and U-Net Architectures for Road Segmentation." Eigenpub Review of Science and Technology 6, no. 1 (2022): 75-91.
URL: https://studies.eigenpub.com/index.php/erst/article/view/69

[8] Kumar, M. Pawan, P. H. S. Ton, and Andrew Zisserman. "Obj cut." In 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), vol. 1, pp. 18-25. IEEE, 2005.
URL: https://www.robots.ox.ac.uk/~vgg/publications/2005/Kumar05/kumar05.pdf

[9] Liu, Jiade, and Cheolkon Jung. "Nnnet: New normal guided depth completion from sparse lidar data and single color image." IEEE Access 10 (2022): 114252-114261.
URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9923926

[10] Tampuu, Ardi, Tambet Matiisen, Maksym Semikin, Dmytro Fishman, and Naveed Muhammad. "A survey of end-to-end driving: Architectures and training methods." IEEE Transactions on Neural Networks and Learning Systems 33, no. 4 (2020): 1364-1384.
URL: https://arxiv.org/abs/2003.06404

[11] Eigen, David, Christian Puhrsch, and Rob Fergus. "Depth map prediction from a single image using a multi-scale deep network." Advances in neural information processing systems 27 (2014).
URL: https://proceedings.neurips.cc/paper_files/paper/2014/file/7bccfde7714a1ebadf06c5f4cea752c1-Paper.pdf

[12] Geyer, Jakob, Yohannes Kassahun, Mentar Mahmudi, Xavier Ricou, Rupesh Durgesh, Andrew S. Chung, Lorenz Hauswald et al. "A2d2: Audi autonomous driving dataset." arXiv preprint arXiv:2004.06320 (2020).
URL: https://arxiv.org/pdf/2004.06320 Dataset: https://www.a2d2.audi/a2d2/en.html

[13] Mao, Anqi, Mehryar Mohri, and Yutao Zhong. "Cross-entropy loss functions: Theoretical analysis and applications." In International Conference on Machine Learning, pp. 23803-23828. PMLR, 2023.
URL: https://proceedings.mlr.press/v202/mao23b/mao23b.pdf

[14] Redmon, Joseph, Santosh Divvala, Ross Girshick, and Ali Farhadi. "You only look once: Unified, real-time object detection." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 779-788. 2016.
URL: https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Redmon_You_Only_Look_CVPR_2016_paper.pdf

[15] Caruana, Rich. "Multitask learning." Machine learning 28 (1997): 41-75.
URL: https://link.springer.com/article/10.1023/a:1007379606734

[16] Gao, Yan, Xiangjiu Che, Quanle Liu, Mei Bie, and Huan Xu. "SFSM: sensitive feature selection module for image semantic segmentation." Multimedia Tools and Applications 82, no. 9 (2023): 13905-13927.
URL: https://link.springer.com/content/pdf/10.1007/s11042-022-13901-0.pdf

[17] Kokkinos, Iasonas. "Ubernet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 6129-6138. 2017.

URL: https://openaccess.thecvf.com/content_cvpr_2017/papers/Kokkinos_Ubernet_Training_a_CVPR_2017_paper.pdf

[18] Pomerleau, Dean A. "Alvinn: An autonomous land vehicle in a neural network." Advances in neural information processing systems 1 (1988).
URL: https://proceedings.neurips.cc/paper/1988/file/812b4ba287f5ee0bc9d43bbf5bbe87fb-Paper.pdf

[19] Redmon, Joseph, and Ali Farhadi. "YOLO9000: better, faster, stronger." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 7263-7271. 2017.
URL: https://openaccess.thecvf.com/content_cvpr_2017/papers/Redmon_YOLO9000_Better_Faster_CVPR_2017_paper.pdf

[20] Alhashim, Ibraheem, and Peter Wonka. "High quality monocular depth estimation via transfer learning." arXiv preprint arXiv:1812.11941 (2018).
URL: https://arxiv.org/pdf/1812.11941

[21] Ranjan, Rajeev, Vishal M. Patel, and Rama Chellappa. "Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition." IEEE transactions on pattern analysis and machine intelligence 41, no. 1 (2017): 121-135.
URL: https://arxiv.org/pdf/1603.01249

[22] Eigen, David, and Rob Fergus. "Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture." In Proceedings of the IEEE international conference on computer vision, pp. 2650-2658. 2015.
URL: https://openaccess.thecvf.com/content_iccv_2015/papers/Eigen_Predicting_Depth_Surface_ICCV_2015_paper.pdf

[23] Kaiser, Lukasz, Aidan N. Gomez, and Francois Chollet. "Depthwise separable convolutions for neural machine translation." arXiv preprint arXiv:1706.03059 (2017).
URL: https://arxiv.org/pdf/1706.03059

[24] Zhu, Yanzhao, and Wei Qi Yan. "Traffic sign recognition based on deep learning." Multimedia Tools and Applications 81, no. 13 (2022): 17779-17791.
URL: https://link.springer.com/content/pdf/10.1007/s11042-022-12163-0.pdf

[25] Brunetti, Antonio, Domenico Buongiorno, Gianpaolo Francesco Trotta, and Vitoantonio Bevilacqua. "Computer vision and deep learning techniques for pedestrian detection and tracking: A survey." Neurocomputing 300 (2018): 17-33.
URL: https://www.sciencedirect.com/science/article/pii/S092523121830290X

[26] Kamal, K. C., Zhendong Yin, Mingyang Wu, and Zhilu Wu. "Depthwise separable convolution architectures for plant disease classification." Computers and electronics in agriculture 165 (2019): 104948.
URL: https://www.sciencedirect.com/science/article/pii/S0168169918318696

[27] Odena, Augustus, Vincent Dumoulin, and Chris Olah. "Deconvolution and checkerboard artifacts." Distill 1, no. 10 (2016): e3.

URL: https://distill.pub/2016/deconv-checkerboard/?ref=mlq-ai

[28] Ayachi, Riadh, Yahia Said, and Abdessalem Ben Abdelaali. "Pedestrian detection based on light-weighted separable convolution for advanced driver assistance systems." Neural Processing Letters 52, no. 3 (2020): 2655-2668.
URL: https://link.springer.com/article/10.1007/s11063-020-10367-9

[29] Kokkinos, Iasonas, and Petros Maragos. "An expectation maximization approach to the synergy between image segmentation and object categorization." In Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1, vol. 1, pp. 617-624. IEEE, 2005.
URL: https://www.researchgate.net/publication/4193820_An_expectation_maximization_approach_to_the_synergy_between_image_segmentation_and_object_categorization

[30] Liu, Shu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. "Path aggregation network for instance segmentation." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 8759-8768. 2018.
URL: https://openaccess.thecvf.com/content_cvpr_2018/papers/Liu_Path_Aggregation_Network_CVPR_2018_paper.pdf

[31] Ghiasi, Golnaz, Tsung-Yi Lin, and Quoc V. Le. "Nas-fpn: Learning scalable feature pyramid architecture for object detection." In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 7036-7045. 2019.
URL: https://openaccess.thecvf.com/content_CVPR_2019/papers/Ghiasi_NAS-FPN_Learning_Scalable_Feature_Pyramid_Architecture_for_Object_Detection_CVPR_2019_paper.pdf

[32] Lin, Tsung-Yi, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. "Feature pyramid networks for object detection." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2117-2125. 2017.
URL: https://openaccess.thecvf.com/content_cvpr_2017/papers/Lin_Feature_Pyramid_Networks_CVPR_2017_paper.pdf

[33] Rateke, Thiago, and Aldo Von Wangenheim. "Road surface detection and differentiation considering surface damages." Autonomous Robots 45, no. 2 (2021): 299-312.
URL: https://link.springer.com/content/pdf/10.1007/s10514-020-09964-3.pdf

[34] He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770-778. 2016.
URL: https://openaccess.thecvf.com/content_cvpr_2016/papers/He_Deep_Residual_Learning_CVPR_2016_paper.pdf

[35] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In Advances in neural information processing systems, pages 1223–1231, 2012
URL: https://arxiv.org/abs/1712.01887

[36] Nekrasov, Vladimir, Thanuja Dharmasiri, Andrew Spek, Tom Drummond, Chunhua Shen, and Ian Reid. "Real-time joint semantic segmentation and depth estimation using asymmetric annotations." In 2019 International Conference on Robotics and Automation (ICRA), pp. 7101-7107. IEEE, 2019.
URL: https://arxiv.org/pdf/1809.04766

[37] Shalf, John. "The future of computing beyond Moore's Law." Philosophical Transactions of the Royal Society A 378, no. 2166 (2020): 20190061.
URL: https://royalsocietypublishing.org/doi/full/10.1098/rsta.2019.0061

[38] Beregi, Sándor, Dánes Takács, Chaozhe R. He, Sergei S. Avedisov, and Gábor Orosz. "Hierarchical steering control for a front wheel drive automated car." IFAC-PapersOnLine 51, no. 14 (2018): 1-6.
URL: https://www.sciencedirect.com/science/article/pii/S2405896318309352

[39] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).
URL: https://arxiv.org/pdf/1409.1556

[40] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet classification with deep convolutional neural networks." Communications of the ACM 60, no. 6 (2017): 84-90.
URL: https://dl.acm.org/doi/pdf/10.1145/3065386

[41] Tan, Mingxing, and Quoc Le. "Efficientnet: Rethinking model scaling for convolutional neural networks." In International conference on machine learning, pp. 6105-6114. PMLR, 2019.
URL: http://proceedings.mlr.press/v97/tan19a/tan19a.pdf

[42] Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need." Advances in neural information processing systems 30 (2017).
URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[43] Kim, Sungwoong, Daejin Jo, Donghoon Lee, and Jongmin Kim. "Magvlt: Masked generative vision-and-language transformer." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 23338-23348. 2023.
URL: https://openaccess.thecvf.com/content/CVPR2023/papers/Kim_MAGVLT_Masked_Generative_Vision-and-Language_Transformer_CVPR_2023_paper.pdf

[44] Kirillov, Alexander, Kaiming He, Ross Girshick, Carsten Rother, and Piotr Dollár. "Panoptic segmentation." In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 9404-9413. 2019.
URL: https://openaccess.thecvf.com/content_CVPR_2019/papers/Kirillov_Panoptic_Segmentation_CVPR_2019_paper.pdf

[45] Nazir, Danish, Alain Pagani, Marcus Liwicki, Didier Stricker, and Muhammad Zeshan Afzal. "Semattnet: Toward attention-based semantic aware guided depth completion."

IEEE Access 10 (2022): 120781-120791.
URL: https://arxiv.org/pdf/2204.13635

[46] Maurer, Andreas, Massimiliano Pontil, and Bernardino Romera-Paredes. "The benefit of multitask representation learning." Journal of Machine Learning Research 17, no. 81 (2016): 1-32.
URL: https://www.jmlr.org/papers/volume17/15-242/15-242.pdf

[47] Tripuraneni, Nilesh, Michael Jordan, and Chi Jin. "On the theory of transfer learning: The importance of task diversity." Advances in neural information processing systems 33 (2020): 7852-7862.
URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/59587bffec1c7846f3e34230141556ae-Paper.pdf

[48] Qi, Charles R., Wei Liu, Chenxia Wu, Hao Su, and Leonidas J. Guibas. "Frustum pointnets for 3d object detection from rgb-d data." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 918-927. 2018.
URL: https://openaccess.thecvf.com/content_cvpr_2018/papers/Qi_Frustum_PointNets_for_CVPR_2018_paper.pdf

[49] Petrovai, Andra, and Sergiu Nedevschi. "Fast panoptic segmentation with soft attention embeddings." Sensors 22, no. 3 (2022): 783.
URL: https://www.mdpi.com/1424-8220/22/3/783

[50] University of Tartu High Performance Computing centre's (UTHPC) official public documentation site
URL: https://docs.hpc.ut.ee/public/

[51] PyTorch documentation
URL: https://pytorch.org/docs/stable/index.html

[52] OpenCV Open Source Computer Vision, OpenCV-Python Tutorials
URL: https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html

[53] Pillow (PIL Fork) Documentation
URL: https://readthedocs.org/projects/pillow/downloads/pdf/latest/

[54] NumPy documentation
URL: https://numpy.org/doc/stable/

[55] Matplotlib Tutorials
URL: https://matplotlib.org/devdocs/tutorials/index.html

[56] Scale Invariant Loss Implementation
URL: https://github.com/lppllppl920/EndoscopyDepthEstimation-Pytorch/blob/master/losses.py

[57] YOLO V1 Implementation URL: https://github.com/motokimura/yolo_v1_pytorch/blob/master/yolo_v1.py

[58] UNet Implementation
URL: https://github.com/milesial/Pytorch-UNet/blob/master/unet/unet_model.py

[59] PilotNet Implementation
URL: https://github.com/UT-ADL/e2e-rally-estonia/blob/master/pilotnet.py

[60] OpenAI, ChatGPT 3.5 URL: https://chatgpt.com/chat

[61] Google, Gemini
URL: https://gemini.google.com/app

[62] Grammarly, Official website
URL: https://app.grammarly.com/

Appendices

I Code base

The code base of the thesis can be found in the repository mentioned below.
URL: https://github.com/ali-609/Asymmetric-Deep-Multitask-Learning
To download repository, use this command: 'git clone https://github.com/ali-609/Asymmetric-Deep-Multitask-Learning'

II Loss graphs

Graphs demonstrating loss functions during training are publicly available in the following Wandb project.
URL: https://wandb.ai/ali-maharramov/thesis_final?nw=nwuseralimaharramov

III Weights

The weights discussed in the results section can be downloaded using the link below.
URL: https://owncloud.ut.ee/owncloud/s/dn2t2NzgAaq6dXz

Non-exclusive licence to reproduce thesis and make thesis public

I, Ali Maharramov

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   "Asymmetric Deep Multi-Task Learning"

   supervised by Tambet Matiisen

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Ali Maharramov
20.05.2024