Evaluation of Multi Vs. Single-task Neural Networks for Autonomous Driving Perception

Dinesh Soudagar
Matriculation Number: 1534137
Masters in Mechatronics
Universität Siegen
16 June 2022

Supervisors:
Prof. Dr. Michael Möller, Universität Siegen.
Dipl-Ing Patrick Brückner, IAV GmbH, Dresden.

Abstract

A Multi-Task Neural Network (MTNN) is a type of network in the field of machine learning where a single large neural network learns to solve multiple tasks simultaneously. In this paper, the performance of MTNNs is compared with that of corresponding Single-Task Neural Networks (STNN). The primary focus of this paper is on four autonomous driving perception tasks: semantic segmentation, lane marking, drivable area, and object detection. The Audi Autonomous Driving Dataset (A2D2) is used for the experiments. The dataset contains semantic segmentation labels for 57 classes (including labels for lane marking and drivable area) and 2D bounding boxes for 14 classes for 12,497 images. Here, we first implement and train STNNs, then MTNNs for all the combinations of our four primary tasks. The MTNN used here has a shared encoder and one decoder for each task. Introduced new loss weighing and training techniques for better optimization of MTNNs. Based on the experiment results, multi-task learning outperforms single-task learning both in speed and accuracy. MTNNs are up to 33% faster, with improvement in accuracy of semantic segmentation up to 1%, lane marking up to 2.5%, drivable area up to 1.5%, and object detection up to 12%. Further, transfer learning method is used to train STNNs with MTNN's shared encoder weights. Results show improvement in STNNs accuracy up to 2%. Also, it shows that MTNN's shared encoder has learned better generalized feature representations on the A2D2 dataset. Finally, an MTNN with object detection as its primary task and semantic segmentation as an auxiliary task is trained on A2D2 dataset. The labels for auxiliary task were generated using DeepLabv3 architecture model pre-trained on CityScapes and BDD dataset. The results show that the object detection mAP score was improved by 6.94%.

Zusammenfassung

Ein neuronales Multiaufgaben-Netzwerk (MN) ist eine Art von Netzwerk im Bereich des maschinell -en Lernens bei dem ein einziges großes neuronales Netz lernt, mehrere Aufgaben gleichzeitig zu lösen. In dieser Arbeit wird die Leistung von MNs mit der von entsprechenden Einzelaufgabe - Netzwerk (EN) verglichen. Das Hauptaugenmerk dieser Arbeit liegt auf vier Wahrnehmungsaufgaben für autonomes Fahren: semantische Segmentierung, Fahrspurmarkierung, befahrbarer Bereich und Objekterkennung. Der Audi Autonomous Driving Dataset (A2D2) wird für die Experimente verwendet. Der Datensatz enthält semantische Segmentierungslabels für 57 Klassen (einschließlich Labels für Fahrspurmarkierung und befahrbaren Bereich) und 2d Bounding Boxes für 14 Klassen für 12.497 Bilder. Hier implementieren und trainieren wir zunächst ENs und dann MNs für alle die Kombinationen unserer vier Hauptaufgaben. Das hier verwendete MN hat einen gemeinsamen Kodierer und einen Decoder für jede Aufgabe. Einführung neuer Verlustwäge- und Trainingstechniken zur besseren Optimierung von MNs. Die Ergebnisse des Experiments zeigen, dass Multi-Task-Lernen dem Single-Task-Lernen überlegen ist sowohl hinsichtlich der Geschwindigkeit als auch der Genauigkeit. MNs sind bis zu 33% schneller, mit einer Verbesserung der Genauigkeit der semantischen Segmentierung um bis zu 1 %, der Fahrspurmarkierung um bis zu 2,5 %, der befahrbaren Fläche um bis zu 1,5 % und der Objekterkennung Erkennung um bis zu 12%. Außerdem wird die Methode des Transfer-Lernens verwendet, um ENs mit MNs gemeinsamen Kodiergewichten zu trainieren. Die Ergebnisse zeigen eine Verbesserung der Genauigkeit der ENs um bis zu 2 %. Außerdem zeigt es MNs geteilter Encoder hat bessere verallgemeinerte Merkmalsrepräsentationen im A2D2-Datensatz gelernt. Schließlich wird ein MN mit Objekterkennung als Hauptaufgabe und semantischer Segmentierung als Hilfsaufgabe Aufgabe auf dem A2D2-Datensatz trainiert. Die Beschriftungen für die Hilfsaufgabe wurden mit dem DeepLabv3 Architekturmodell generiert, das zuvor auf CityScapes und BDD-Datensätzen trainiert wurde. Die Ergebnisse zeigen, dass die Objekt Erkennung mAP-Score um 6,94% verbessert wurde.

Contents

List of Tables

List of Figures

1 Introduction

In recent years, there has been a huge development in the area of deep-learning-based perception methods for autonomous vehicles. An end-to-end self-driving car has four major components, namely perception and localization, high-level path planning, behavior arbitration (or low-level path planning), and motion controllers [6]. Perception is a very important property of a self-driving car. It helps the car to understand its surroundings, classify and locate objects, and make intelligent decisions. A self-driving car needs information about traffic lights, traffic signs, road signs, cars, pedestrians, etc. This can be achieved by using deep learning-based methods such as 2D and 3D object detection, semantic segmentation, lane marking, depth estimation, etc. All these are camera-based deep learning methods. In a self-driving car, these methods rely on camera data to provide the understanding of the surroundings necessary for the other three components.

In deep learning, we typically aim to optimize a Deep Neural Network (DNN) that is dedicated to solving one particular task. This can be referred to as a Single-Task Neural Network (STNN). Any STNN is capable of doing a task with a certain accuracy. Generally, this depends on a lot of factors, such as dataset size and quality, type of architecture, loss functions, etc. Even if all these factors are figured out properly, in most cases, the performance is not up to the desired accuracy. Then, to improve the performance further, several techniques like data regularization, data augmentation, hyper-parameter tuning, etc. are employed in the training of STNNs. In this way of focused training, an STNN can solve the dedicated task with acceptable performance. In contrast to autonomous driving perception, these tasks can be 2D object detection, semantic segmentation, lane marking, etc. All these tasks can be solved and implemented using dedicated STNNs in the self-driving car. Each trained STNN would have learned to extract features from the input image (camera input). It has also learned to use relevant features for prediction and ignore non-relevant ones. What if these non-relevant features are relevant for another task? These learned feature representations can be shared with another STNN for solving another task. Thus, the idea of Multi-Task Learning (MTL). MTL is an approach to improving a network's ability to generalize by sharing features or information between the tasks. A Multi-Task Neural Network (MTNN) can potentially be faster, more accurate, use fewer computing resources, and solve multiple tasks simultaneously [7] [8]. In this study, the main aim is to explore this idea of MTL in the field of autonomous driving perception. The general idea of this study is to 1. implement STNNs for some specific primary autonomous driving perception tasks and 2. implement an MTNN that solves all of our primary tasks at the same time. 3. Investigate the compatibility of our primary tasks in an MTL scenario. 4. Research the best methods for optimizing an MTNN. 5. Finally, compare the MTNN's performance with that of STNNs.

2 State of The Art

An STNN takes one input and provides one output at a time. For example, an STNN can be taught to classify images into predefined categories (known as "classification tasks"), to recognize objects in images and draw bounding boxes around them (known as "object detection"), and more. In the case of autonomous driving perception, STNNs are used to solve tasks like semantic segmentation, lane marking, depth estimation, object detection, object tracking, etc. The output of these STNNs is used dynamically for vehicle path planning. In this section, some perception tasks, architectures, loss functions, and their evaluation metrics are discussed.

2.1 Segmentation

Segmentation is a task in deep learning that aims to classify input image pixels into several categories. It is utilized in a variety of applications, including self-driving cars [6], intelligent robotic systems [9], damage detection [10], medical image segmentation [2] and so on. In the

case of autonomous driving, these tasks can be semantic segmentation, lane marking, motion segmentation, drivable area, etc. Some segmentation tasks related to autonomous driving are discussed below.

Semantic segmentation: It is a segmentation task used in self-driving cars to categorize the input image pixels into cars, trucks, pedestrians, bicycles, etc. Semantic segmentation helps provide autonomous vehicles with a deep understanding of their surroundings, which is necessary for motion planning. An example of semantic segmentation is shown in figure 1..

Figure 1: An example of an input image on the left and its semantic segmentation color map on the right. Image taken from A2D2 dataset [1].

Lane marking: It is a segmentation task of detecting road markings like painted driving instructions, crossroads, dashed lines, etc. It helps the vehicle to navigate through the different lanes, determine its relative position between the vehicle and the road, and analyze the vehicle's heading direction. An example of lane marking is shown in figure 2.

Figure 2: An example of an input image on the left and its lane marking color map on the right. Image taken from A2D2 dataset [1].

Drivable area: Drivable area is a segmentation task that helps the vehicle understand the road conditions and determine drivable and non-drivable areas, which is used for vehicle motion planning. An example of drivable area is shown in figure 3.

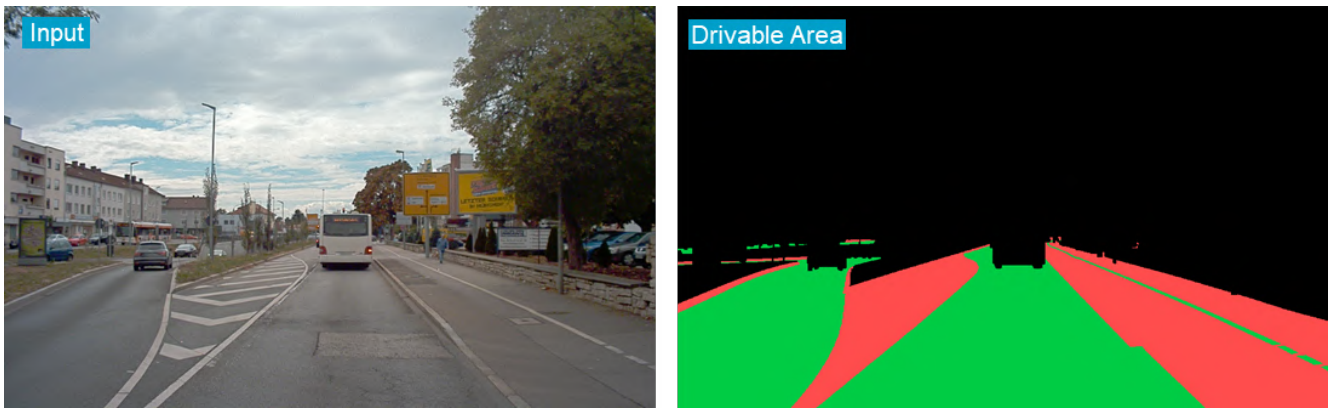

Figure 3: An example of an input image on the left and its drivable area color map on the right. Image taken from A2D2 dataset [1].

2.1.1 Architectures

There are several popular architectures that solve segmentation efficiently. Fully Convolution Network (FCN) [11], Mask RCNN [12], U-Net [2], DeepLab [13] and many more. U-Net is one of the most widely used segmentation models. The U-net is popular because it is simple and captures the spatial locations of objects as well as context at the same time. The U-net architecture is shown in figure 4. The architecture consists of an encoder and a decoder connected via a bottleneck layer. The encoder halves the input image spatial dimension and doubles the number of channels at four stages. Each stage in the encoder contains two convolution layers followed by a max-pool layer. The decoder takes input from the bottleneck layer. It doubles the input spatial dimension and halves the number of channels at four stages, and the output from the final stage is mapped to the segmentation map.

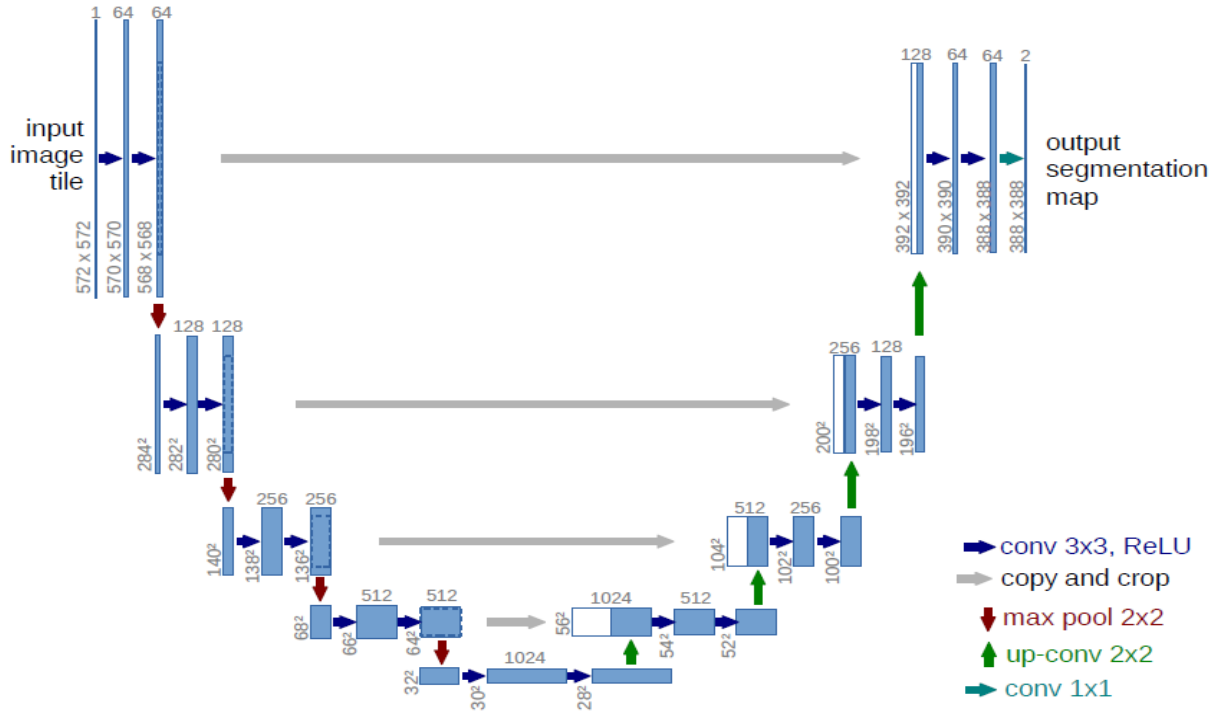

Figure 4: U-net Architecture [2]. The numbers on the top of each box indicates number of channels and numbers at the bottom indicates the spatial dimensions of the image at that stage.

2.1.2 Training and Loss Functions

Semantic segmentation can be treated as an image classification task at the pixel level. A segmentation model is trained with a set of images and their corresponding 2D class maps of the same spatial dimension. A 2D class map is an image where each pixel value represents a class ID to which the input image's corresponding pixel belongs. There are several loss functions that can be used to train a semantic segmentation network. The most popular loss functions are cross-entropy loss, dice loss, focal loss, etc. A detailed review of different loss functions available for image segmentation is described in this paper [14]. It is difficult to decide on a universal loss function because it greatly depends on the type of dataset (unbalanced, balanced, skewed, etc.). Loss functions like focal loss, sensitivity-specificity loss, and a few others work best for imbalanced datasets. Cross-entropy loss works best with balanced datasets, while weighted cross-entropy loss works best with skewed datasets [14].

2.1.3 Evaluation Metrics

Evaluation metrics help to evaluate the performance of the model. The most commonly used evaluation metrics are pixel accuracy, Intersection Over Union (IOU) [15], and dice coefficient [16]. The simplest one is pixel accuracy, which simply measures the percentage of correct pixels predicted by the model. However, in the case of an imbalanced dataset, it fails to give a meaningful score. For example, if a target label is covered with 90% of the background class and the rest is covered with the primary classes, even if the model has failed to predict any of the primary classes, the pixel accuracy score for that label will be 90% which is misleading. The IOU and dice coefficient are described in figure 5.

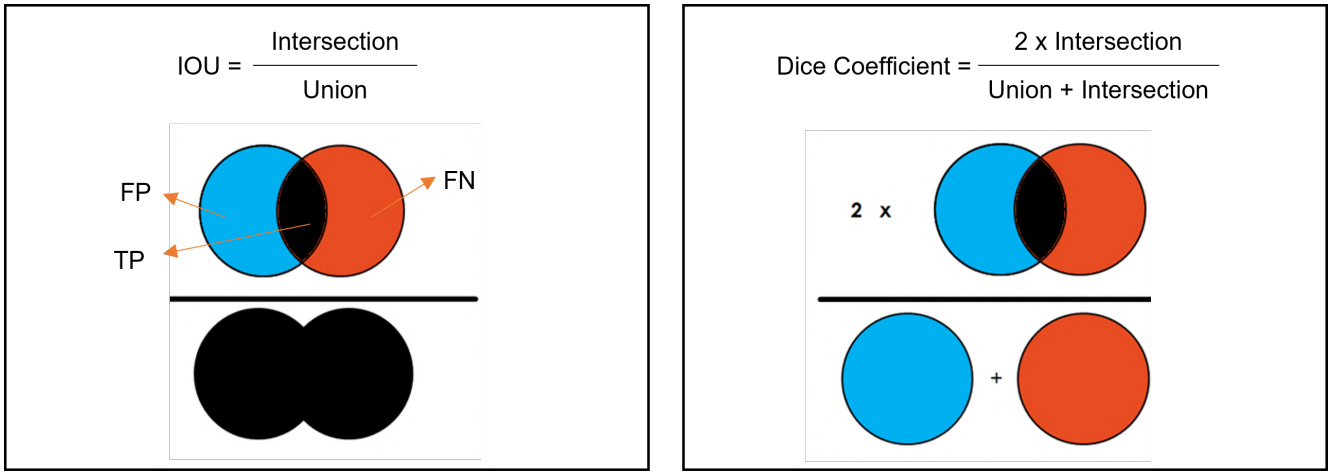

Figure 5: Evaluation metrics for segmentation task [3].

The IOU is simply the ratio of intersection over union, which focuses on True Positives (TP), False Positives (FP), and False Negatives (FN), as shown in the equation (1).

$$IOU = \frac{TP}{TP \ + \ FP + \ FN} \tag{1}$$

Both the IOU and the dice coefficient are calculated for each class and then averaged over all the classes for the final value. Since these two metrics are averaged over all the classes, it provides a meaningful score that captures the accuracy of all the classes, unlike pixel accuracy. The dice coefficient (equation (2)) is the harmonic mean of precision and recall (equation (3)).

$$Dice\ Coefficient = \frac{1}{\frac{1}{Precision} + \frac{1}{Recall}} = \frac{2 \cdot TP}{2 \cdot TP \ + \ FP + \ FN} \tag{2}$$

$$Precision = \frac{TP}{TP\ +\ FP} \qquad , Recall = \frac{TP}{TP\ +\ FN} \tag{3}$$

Precision focuses on FNs, whereas recall focuses on FPs. In the context of autonomous driving, both FNs and FPs need to be zero for safe motion planning of the vehicle.

2.2 Object Detection

Object detection is a computer vision application that is employed to detect certain instances of objects like cars, people, animals, etc. in an image. It has several applications, like self-driving cars, intelligent logistics, facial recognition, security, etc. In the case of autonomous driving applications, object detection is used to localize objects like cars, pedestrians, traffic signs, etc.

2.2.1 Architectures

YOLO [17] and its variants, Fast RCNN [18], Faster RCNN [19], SSD [20], RetinaNet [21], and others, are popular object detection architectures. YOLO is the most famous object detection algorithm due to its speed and accuracy, which makes it the best for autonomous driving applications. The comparison of YOLOv3 and other models is shown in figure 7. YOLOv3, YOLOv4, and YOLOv5 all have good detection accuracy and speed for real-time applications [22]. However, their performance varies with respect to the type of data and application. The architecture of YOLOv3 is shown in figure 6. It has darknet-53 as a feature extractor, and it predicts the bounding boxes at 3 different locations corresponding to different bounding box sizes (small, medium, and large) [23]. There are two skip connections employed in the model, as shown in figure 6. More details of YOLOv3 can be found in their paper [23].

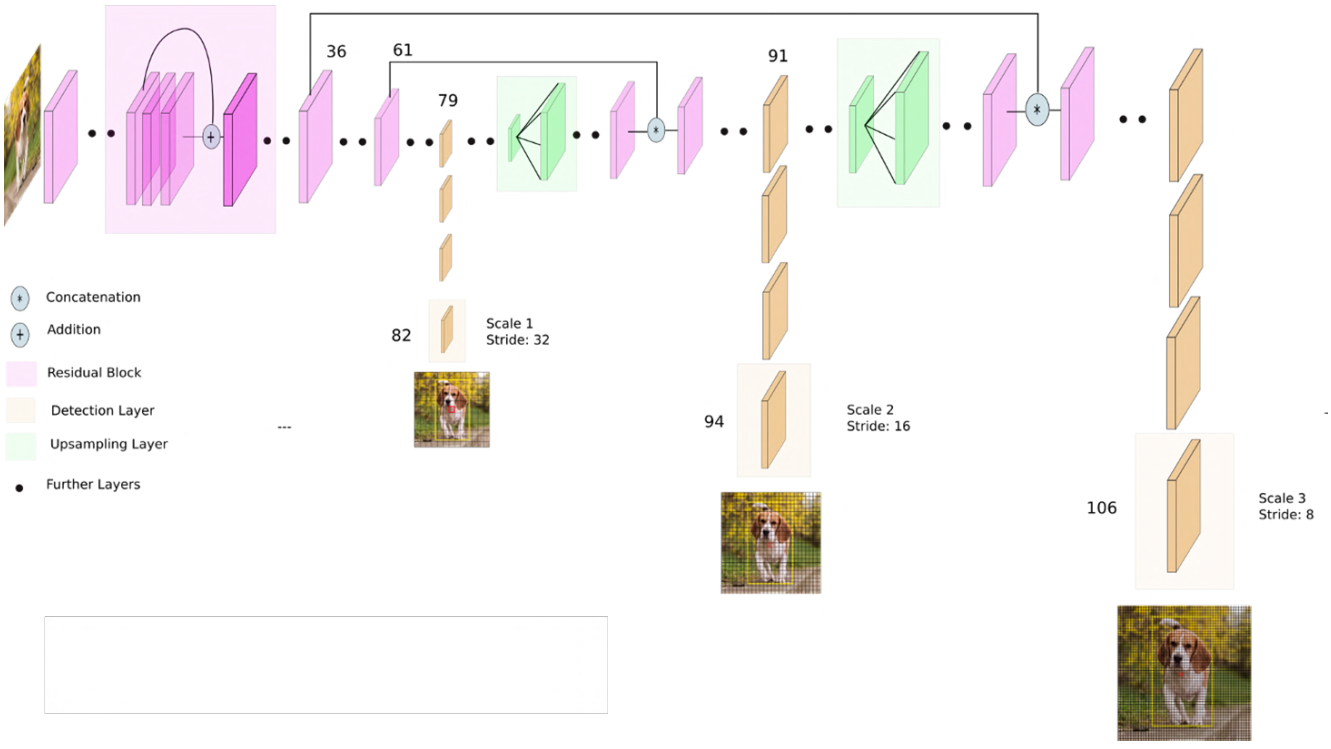

Figure 6: YOLOv3 architecture [4].

2.2.2 Training and Loss Functions

An object detection model is trained with a set of images and bounding box labels. A bounding box label consists of values describing class and bounding box coordinates for all the interested

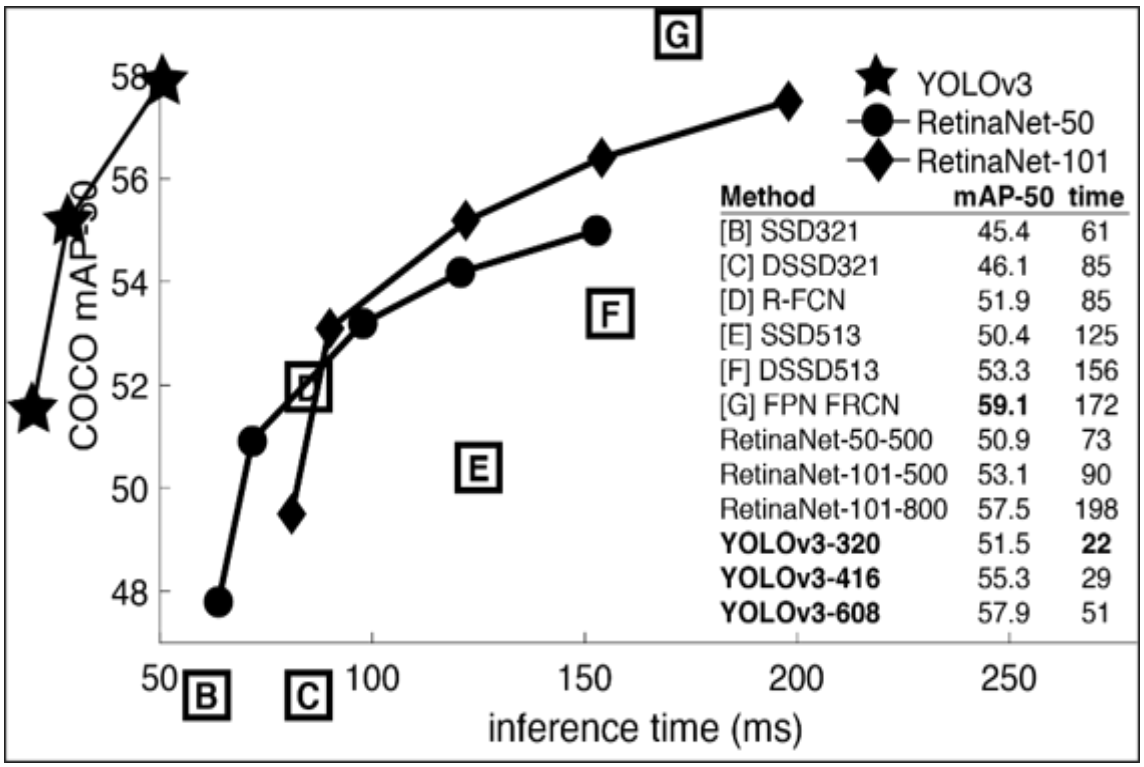

Figure 7: YOLOv3 Comparison with other detection architectures [5].

objects in the corresponding image. There are several different loss functions that are used to train object detection models. Generally, every loss function has two main parts. The first one is classification loss (Cross-Entropy, Focal loss, Hinge loss, etc.) for classifying the detected objects, and the second one is regression loss (MSE, MAE, etc.) for predicting the bounding box coordinates. Depending on the architecture, additional optimizing parameters are employed in the loss function. For example, YOLO uses a four-part loss function. The additional two parts are related to the model's confidence in predicting that there is an object and in predicting that there is no object.

2.2.3 Evaluation Metrics

There are two popular evaluation metrics used to evaluate object detection models: Average Precision (AP) and mean Average Precision (mAP) [24]. AP is nothing but the area under the Precision-Recall (P-R) curve at a predefined IOU(1) threshold. The P-R curve is constructed by calculating precision and recall (3) and plotting the precision against the recall for one class. The mAP is simply the mean of the APs across all the classes.

3 Multi-Task Neural Network

3.1 Concept

Multi-task learning is an approach to improving a neural network's ability to generalize by learning to solve multiple tasks simultaneously. By sharing the features between the tasks, MTL enables the model to generalize and perform better over single-task learning. A Multi-Task Neural Network (MTNN) takes in a single input and provides multiple outputs at the same time. In the case of autonomous driving, these tasks can be semantic segmentation, lane marking, object detection, etc. An MTNN can be trained with a single dataset having multiple labels with a lower amount of computing resources both during training and inference time. It can solve multiple tasks

with better accuracy compared to corresponding single tasks. In recent years, many approaches have been developed. One approach that is based on the features sharing idea (also called hard parameter sharing [25]) is the shared trunk MTNN. It consists of a global feature extractor made of several convolution layers, and these layers are shared between all the task heads (multiple decoders). The output of each decoder is then mapped into corresponding task outputs. Many other MTNN architectures follow the shared trunk type outline, such as Facial landmark detection by deep multi-task learning [26], Instance-aware semantic segmentation via multi-task network cascades [27], End-to-end multi-task learning with attention [28] and others. Detailed information on different types of MTNN architectures can be found in these papers here [25] [29]. Another approach to MTL called Cross-Talk is proposed in the papers Cross-stitch networks for multi-task learning [30] and Latent multi-task architecture learning [31]. In this approach, MTNN does not have a shared encoder, instead, each task has a separate network. Each task shares the information between the parallel layers of the other task networks. This information sharing is referred to as "cross-talk." There are numerous other approaches to MTL, more detailed information on various types of MTNN architectures is available in these papers [25] [29].
There are many approaches have been published for MTL for various applications, and few are specifically for autonomous driving perception tasks. Teichmann et al. [8] is one of the first approaches to solve classification, object detection, and semantic segmentation tasks using the KITTI vision benchmark dataset [32]. They propose a multi-task network called MultiNet, a simple encoder-decoder network similar to our implementation (discussed in the section 4.3). They demonstrated that MTL can improve task performance and be very efficient. Kumar et al. [7] propose the OmniDet multi-task network that aims to solve six autonomous driving perception tasks including depth estimation, semantic segmentation and object detection. They demonstrate that OmniDet network for six-task model on WoodScape [33] and five-task model on KITTI [32] and CityScapes [34] perform better than respective single task networks. The goal of the both above approaches are very similar to the goal of this study. Nekrasov et al. [35] demonstrate that MTL can achieve equivalent results compared to state-of-the-art single task networks. Sistu et al. [36] propose a NeurALL: unified CNN architecture for important visual perception task in automated driving. Feng et al. [37], Chowdhuri et al. [38], Ishihara et al. [39], and Maddu et al. [40] are some of the other approaches that focuses on MTL for Autonomous driving.

3.2 Datasets

Many autonomous driving datasets have been made available by many companies and research institutions. However, it is difficult to find the best dataset that is suitable for a particular goal. Audi Autonomous Driving Dataset (A2D2) [1], Berkely DeepDrive Dataset (BDD) [41], CityScapes Dataset [34], KITTI Vision Benchmark Suite [42], nuScenes Dataset [43], WoodScape [33], etc. are widely used for autonomous driving perception tasks. These datasets aim to solve tasks like instance segmentation, 2D and 3D object detection, semantic segmentation, lane marking, depth estimation, motion segmentation, and many more. A typical dataset contains data in the form of sets of images with corresponding single target labels. However, for an MTNN to provide multiple outputs for a single input, we need a dataset that has multiple labels for a single input image. In our case, these labels should correspond to our primary autonomous driving perception tasks. The datasets A2D2 [1], BDD [41] and WoodScape [33] are suitable for our MTNN training.

A2D2 is an open-source autonomous driving dataset released by Audi. The dataset contains 41,277 images with semantic segmentation labels for 57 classes (including labels for lane marking and drivable area), instance segmentation labels, point cloud labels, and 3D and 2D bounding box labels for 12,497 of the 41,277 images. BDD is a large-scale, diverse driving video dataset that contains 100K driving videos with comprehensive annotations for 10 tasks. In contrast to our requirement, it also has two sets of image data. Set one is 100K images with object detection, lane marking, and drivable area labels, and set two is 10K images with semantic segmentation labels.

4,235 out of 10K images are part of set one. WoodScape is an interesting dataset that provides fish-eye camera images with multiple labels that correspond to nine different tasks, including semantic segmentation, 2D object detection, depth estimation, etc. However, currently, the dataset contains labels only for semantic segmentation and 2D object detection corresponding to 10K fish-eye camera images.

3.3 Tasks Selected

The selection of the tasks directly depends on which available datasets are suitable for our goal. As discussed above, A2D2 and BDD datasets can be considered the best options. The A2D2 dataset contains labels for semantic segmentation, lane marking, drivable area, and object detection tasks corresponding to 12,497 images. And in BDD, there are 4,235 images with semantic segmentation, object detection, lane marking, and drivable area labels. The WoodScape dataset contains labels for semantic segmentation and 2D object detection corresponding to 10K fish-eye camera images. Based on the description of all the datasets, considering quantity, quality, and the total number of tasks, A2D2 seems to be the best fit for our goal. Hence, for this study, semantic segmentation, lane marking, drivable area, and object detection tasks are selected.

3.4 Architectures Selected

The idea of MTNN here is to have a simple network that should correspond to the core idea of MTL of sharing features between the tasks. Based on all the different types of MTNN architectures discussed in the previous section, the shared trunk approach is selected. It has a shared trunk (shared encoder) and multiple decoders attached to the shared trunk. In order to have a transparent and proper comparison with STNNs, it is necessary that they have an encoder and decoder structure or similar. In this way, each decoder of MTNN can be the same as the individual STNN decoders, and the shared encoder will have the same dimension across all the STNNs and MTNN. Hence, for semantic segmentation, lane marking, and drivable area, U-net [2] is the best fit, and YOLOv3 [23] is the best fit for object detection. To improve the architecture further and reduce training time, every encoder can be replace with a large pre-trained network instead of training completely from scratch. Further, the shared encoder in MTNN should be large enough such that it has enough feature representations to be optimized for individual decoders. Inception [44], Xception [45], VGG [46] and ResNet [47] are some of the widely used pre-trained networks. Considering the available resources and simplicity, resnet-34 is selected as the common encoder for all MTNNs and STNNs. The actual implementation details are described in section 4.

3.5 Goal of This Work

Any STNN is capable of doing a task with a certain accuracy. They do, however, present certain difficulties in terms of producing excellent results. To train an STNN, you'll need a task-specific dataset that's vast and diverse. Multiple STNN training necessitates a significant amount of computational power and time. In the case of autonomous driving, the output of these is used dynamically for vehicle path planning. Furthermore, they must be highly precise and fast, requiring a large amount of processing power in the vehicle. All these problems can be solved with MTL to some extent. Hence, in this study, the main goal is to implement, investigate, and improve the performance of MTNNs over STNNs. The following are the sub-goals: 1. Research the best optimization method for MTNN to solve our primary tasks. 2. Look into the loss functions that result in the best performing MTNN over single tasks. 3. Investigate task compatibility when working together in an MTNN. Finding an auxiliary task that can be used to improve the performance of the other task. 4. Compare and contrast the inference times of STNNs and MTNNs.

4 Implementation

4.1 Dataset Preparation

A2D2 is an open-source autonomous driving dataset released by Audi in the year 2020 [1]. As discussed in section 3.2, the dataset contains labels for semantic segmentation, lane marking, drivable area, and object detection tasks corresponding to 12,497 images. The 55 A2D2 base classes are distributed into three of our four primary tasks. For example, The classes 30, 31, 34, and 35 in the table 8 from A2D2 base classes belong to the lane marking task. These classes are also part of the class road in semantic segmentation and the class drivable area in the drivable area task. All the other classes that do not belong to the lane marking are categorized as background class. Also, the classes like car1, car2, car3, and car4 are variants of the same category "car". These types of classes are combined into a single class. The table 8 shows which of the 55 A2D2 base classes are combined and distributed into semantic segmentation: 20 classes, lane marking: 4 classes, and drivable area: 2 classes (excluding background class). An example of an A2D2 segmentation color map distributed into semantic segmentation, lane marking, and the drivable area is shown in figure 8. The 2D bounding boxes are distributed for the object detection primary task, refer to table 2. Also, here, similar classes like cars and VanSUV are combined into a single class of car. The 12,497 images are provided in the form of sets of images, and these sets are divided into training and testing sets based on the type of weather, the number of images, and instances, such that they follow 80% training and 20% testing data approximately, refer to table 3. Considering the size of the dataset, it is not partitioned for validation data.

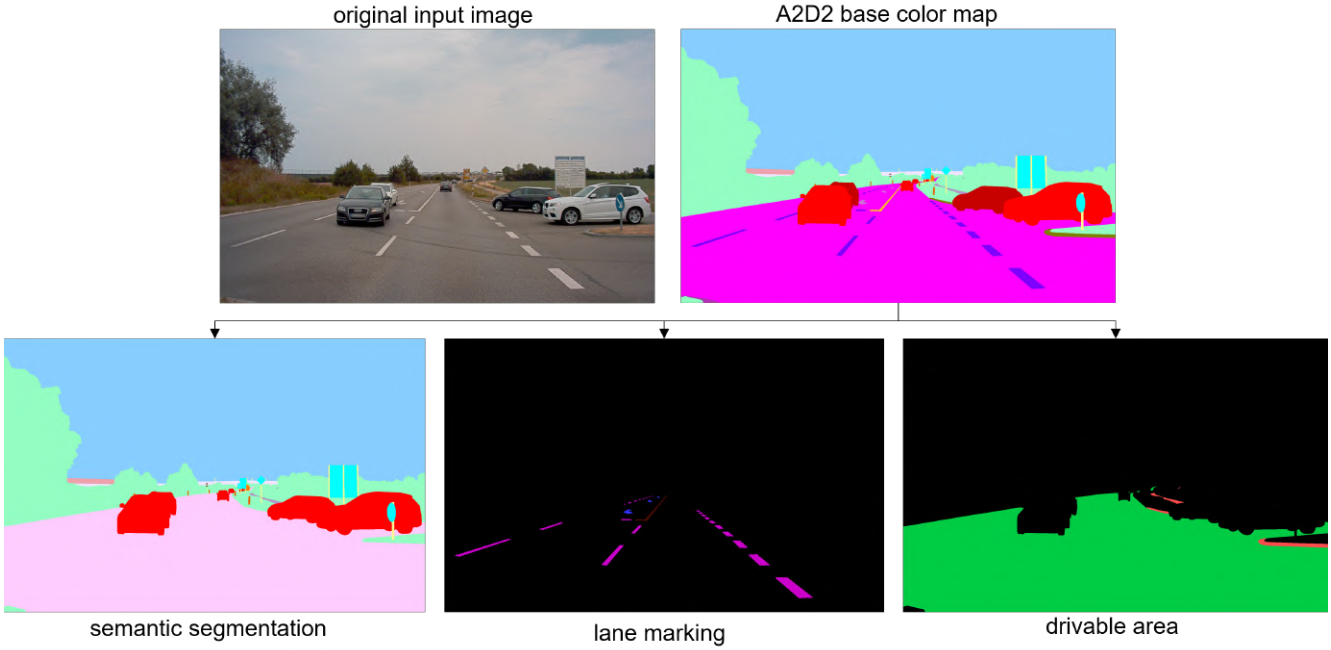

Figure 8: An example of an A2D2 base color map distributed into primary task labels.

4.2 Single-Task Neural Network Implementation

4.2.1 Segmentation Tasks

Model

The model architecture for primary tasks semantic segmentation, lane marking, and the drivable area is illustrated in figure 9. The model follows the U-net architecture [2], a U-shaped encoder-

Sl.No	A2D2 base classes	Semantic segmentation	Lane marking	Drivable area
1	Car 1	Car	Background	Background
2	Car 2			
3	Car 3			
4	Car 4			
5	Bicycle 1	Bicycle		
6	Bicycle 2			
7	Bicycle 3			
8	Bicycle 4			
9	Pedestrian 1	Pedestrian		
10	Pedestrian 2			
11	Pedestrian 3			
12	Truck 1	Utility vehicle		
13	Truck 2			
14	Truck 3			
15	Utility vehicle 1			
16	Utility vehicle 2			
17	Tractor			
18	Small vehicles 1	small vehicles		
19	Small vehicles 2			
20	Small vehicles 3			
21	Traffic signal 1	Traffic signal		
22	Traffic signal 2			
23	Traffic signal 3			
24	Traffic sign 1	Traffic sign		
25	Traffic sign 2			
26	Traffic sign 3			
27	Sidebars	Sidebars		
28	Speed bumper	Speed bumper		drivable area
29	Curbstone	Road		Non-drivable area
30	Solid line		Solid line	drivable area
31	Zebra crossing		Zebra crossing	
32	Drivable cobblestone		Background	
33	Slow drive area			
34	Painted driv. instr.		Painted driv. instr.	
35	Dashed line		Dashed line	
36	RD normal street		Background	
37	Road blocks	Road blocks		Background
38	Poles	Poles		
39	Animals	Animals		
40	Grid structure	Grid structure		
41	Signal corpus	Signal corpus		
42	Nature object	Nature object		
43	Parking area	Parking area		
44	Sidewalk	Sidewalk		Non-drivable area
45	Traffic guide obj.	Traffic guide obj.		Background
46	Sky	Sky		
47	Buildings	Buildings		
48	Irrelevant signs	Background		
49	Non-drivable street			Non-drivable area
50	Obstacles / trash			Background

Sl.No	A2D2 base classes	semantic segmentation	lane marking	drivable area
51	RD restricted area	Background	Background	Non-drivable area
52	Electronic traffic			Background
53	Ego car			
54	Blurred area			
55	Rain dirt			

Table 1: Distribution of A2D2 base semantic segmentation classes for primary tasks ; semantic segmentation, lane marking and drivable area.

Sl.No	A2D2 base classes	Number of instances	Object detection
1	Car	25041	Car
2	VanSUV	4389	
3	Pedestrian	7574	Pedestrian
4	Truck	2057	UtilityVehicle
5	UtilityVehicle	510	
6	Cyclist	521	Cyclist
7	Bus	354	Bus
8	MotorBiker	1560	MotorBiker
9	Bicycle	472	Bicycle
10	Motorcycle	238	Motorcycle
11	CaravanTransporter	34	Not considered due to low number of instances for training and testing
12	Animal	50	
13	Trailer	77	
14	EmergencyVehicle	27	

Table 2: Distribution of A2D2 base 2D bounding box classes for primary task; object detection.

Sl.No	A2D2 Image Set IDs	No. Of Images	Weather type	Category	Count
1	20180807145028	942	cloudy	Training data	9768
2	20180925101535	1060	sunny		
3	20180925112730	952	sunny		
4	20180925124435	976	sunny		
5	20181016125231	893	sunny		
6	20181107132300	570	sunny		
7	20181107132730	812	sunny		
8	20181108084007	643	cloudy		
9	20181108091945	862	cloudy		
10	20181108123750	1283	cloudy		
11	20181204135952	624	cloudy		
12	20181204154421	151	drizzling/cloudy		
13	20180810142822	563	cloudy	Testing data	2728
14	20180925135056	692	sunny		
15	20181008095521	727	cloudy		
16	20181107133258	117	sunny		
17	20181108103155	517	cloudy		
18	20181204170238	112	drizzling/cloudy		

Table 3: Distribution of A2D2 image sets for training and testing.

decoder network connected via a bottleneck layer as described in the paper [2]. The encoder part acts as a feature extractor and learns an abstract representation of the input image. The decoder part uses these representations and also additional information from the skip connections to generate a better segmentation mask. As discussed in the section 3.4, the encoder part of the network is replaced with resnet-34 [47] pre-trained on the Image-Net dataset [48]. The stride of conv1 of resnet is changed to 1 to match the skip connection dimensions for corresponding concatenation in the decoder. Each resnet layer doubles the number of input channels and halves the input's spatial dimension. A bottleneck layer acts as a bridge between the encoder and the decoder. In the decoder part, each double conv block consists of two 3x3 convolutions with stride 1 and padding 1, each followed by a ReLU and batch normalization. The double conv block halves the number of input channels, and a 2x2 up-sampling layer is used to double the input's spatial dimension before each concatenation. The final layer maps the input to the final output dimension (number of classes x H x W). The complete model is implemented in the PyTorch library; torch version 1.10.1, torchvision version 0.11.2, and cuda version 10.2.

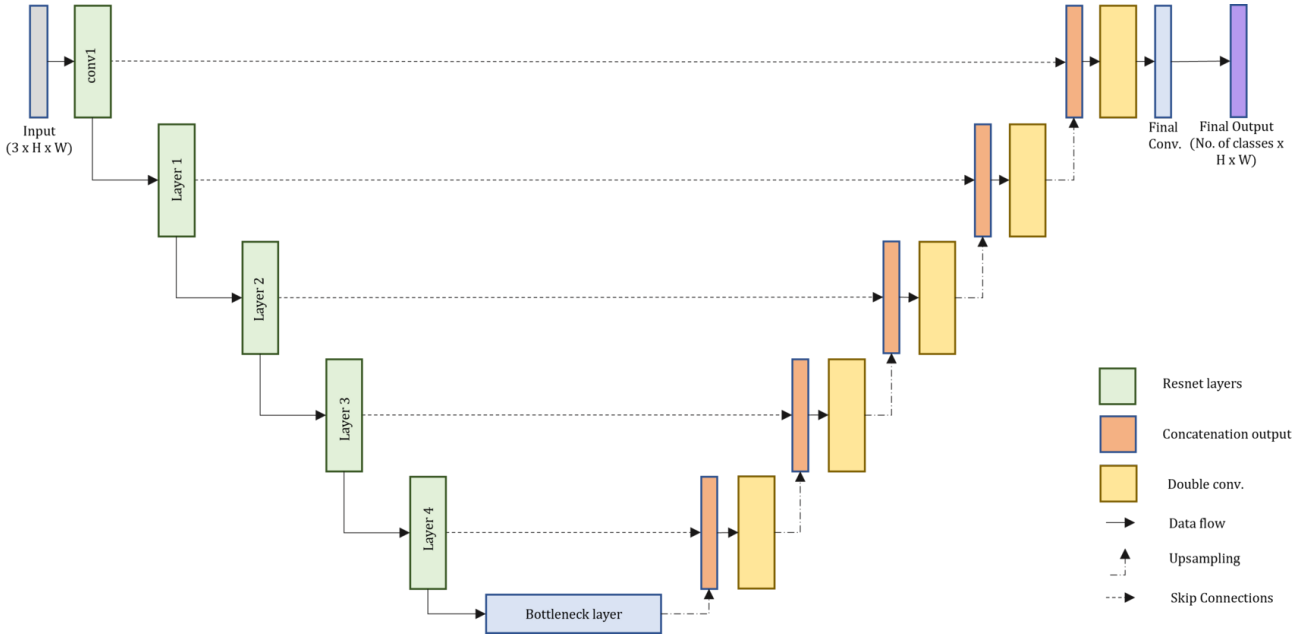

Figure 9: Segmentation model with resnet as encoder part of the U-Net.

Loss

The segmentation task can be considered as a pixel-wise classification problem. The model is optimized with weighted cross-entropy loss implemented in PyTorch library [49] as illustrated in equation (4).

$$Weighted\ Cross\ Entropy\ Loss : -\sum_{c=1}^{C}[y_{o,c}\log(p_{o,c})\cdot w_c] \tag{4}$$

In the equation, C is number of classes, log is the natural log, $y_{0,c}$ is binary indicator (0 or 1) if class label c is the correct classification for observation o. The $p_{o,c}$ is predicted probability of observation o is of class c and w_c is weight for class c. As discussed in section 2, to handle the imbalanced distribution of classes over the target label known as the skewed type of data as described in the paper [14], weights are applied to the loss function at every batch. The weights are calculated using the inverse class frequency method, which calculates the weight for each class inversely proportional to the number of pixels of that class covered in the target label. The formula

to calculate weights is illustrated in equations (5) and (6).

$$w_c = \frac{Total\ number\ of\ pixels\ in\ target\ label}{C \cdot number\ of\ pixels\ of\ class\ c} + 1 \tag{5}$$

$$weight = \frac{Total\ number\ of\ pixels\ in\ target\ label}{C \cdot number\ of\ pixels\ of\ class\ c} \tag{6}$$

During training, in a situation where for some training examples the weights calculated for one or more classes will be very large as they will be covering a few pixels (less than 5%) over the target label, the weights calculated for the remaining classes will be very low. During this situation, the classes having low weights will be partially learned (weak gradients) and other classes with large weights will be learned with stronger gradients, which may lead to a high number of false positives during testing. In other words, classes with larger weights are prioritized and learned effectively, whereas other classes are partially learned. To deal with this situation, each calculated weight is added with one, ensuring that each class has enough gradients and is not ignored throughout the training. To test this, semantic segmentation STNN was trained with weighted cross entropy loss with both weighting methods (equation (5) and (6)). The results are shown in table 4. It can be seen that the overall accuracy is little better when the classes are weighted with equation (5) compared to equation (6). As per results, some of the classes like car, road, road blocks, side walk, etc. which usually cover lot of area in the target label, have better accuracy. Figure 10 shows results for both STNNs. STNN trained with equation (5) shows good prediction on road (example a), vegetation (example b), etc. however bad at predicting small objects like pole in all examples, refer to prediction 2, figure 10. STNN trained with equation (6) shows good prediction for small objects like poles, but many false positives can be seen on road and sidewalk predictions (examples a and c). Refer to prediction 1, figure 10.

Class ID	Class names	Weighted with equation (5)	Weighted with equation (6)	Difference
0	Car	89.34	88.37	0.97
1	bicycle	53.52	56.31	-2.79
2	pedestrian	46.34	50.54	-4.20
3	Utility vehicle	48.77	51.73	-2.96
4	small vehicles	79.51	82.64	-3.13
5	Traffic signal	72.38	71.57	0.81
6	Traffic sign	58.08	58.14	-0.06
7	sidebars	86.39	84.17	2.22
8	road	96.64	96.63	0.01
9	road blocks	84.08	74.72	9.36
10	poles	43.30	44.90	-1.60
11	animals	96.33	96.33	0.00
12	grid structure	38.01	36.28	1.73
13	signal corpus	45.15	47.21	-2.06
14	nature object	90.19	90.31	-0.12
15	parking area	58.78	59.83	-1.05
16	sidewalk	65.40	64.32	1.08
17	traffic guide obj.	69.61	66.52	3.09
18	sky	97.23	97.15	0.08
19	buildings	77.23	77.67	-0.44
20	background	42.19	39.67	2.52
Average		68.50	68.33	0.16

Table 4: The table shows class wise semantic segmentation results trained on equation (4). Column three results are related to weighing equation (5) and column four results are related to weighing equation (6). The last column indicates the difference in the a class accuracy between column two and three results.

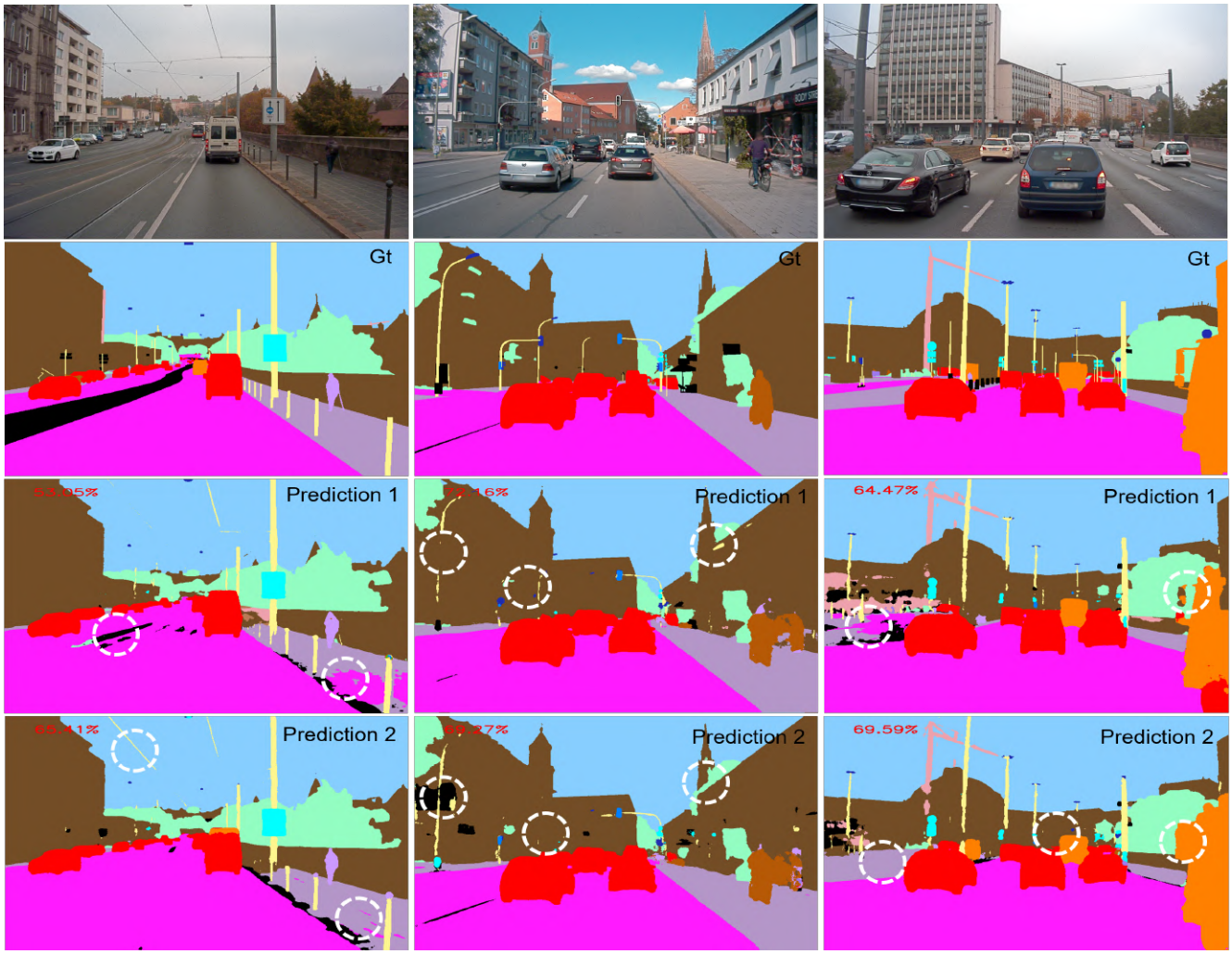

Figure 10: Figure shows examples of semantic segmentation prediction on A2D2. Prediction 1 belongs to STNN trained with equation (6) and prediction 2 belongs to STNN trained with equation (5). The white circles indicate area of interest, to compare predictions with respect to ground truth. The score written on each prediction image indicates the dice coefficient with respect to the ground truth.

Training Details

The input to the model is an RGB image of size 704x704, paired with a corresponding 2D class map of the same size as the target label. The model is trained using the Adam optimizer with a learning rate of 0.00001 and weight decay of 0.0005 to avoid over-fitting. This learning rate and weight decay were the initial values selected for training that gave good results. These values are used in all the other training in this study. Training data were augmented with image rotation, horizontal flip, vertical flip, random brightness, blur, and colorjitter using albumentations library [50]. A dice coefficient metric is used to measure the model's performance as it captures precision and recall in the measurement. During training, after each epoch, the model's performance on test data is measured, and the model with the highest dice coefficient is considered the final model. The semantic segmentation, lane marking, and drivable area models were trained for 110, 58, and 63 epochs, respectively. And the models with the best performance were achieved at epochs 97, 39, and 43 for semantic segmentation, lane marking, and drivable area, respectively. The training loss and accuracy graphs are shown in section 9, figure 38.

Results

The results of the primary tasks semantic segmentation, lane marking, and drivable area are shown in the table 5 below. The class-wise accuracies are shown in the appendix section 9, figure 34. The accuracy of the drivable area is highest as it attempts to classify the pixels into only 3 categories. Similarly, semantic segmentation and lane marking accuracy are lower with an increase in the number of classes. Figure 11, figure 12, and figure 13 shows predictions examples of semantic segmentation, lane marking, and drivable area respectively. To have an idea of the model's predictions at a different level of accuracy, each figure has three examples a, b, and c, belonging to the low, mid, and high range of the model's prediction accuracy. In all the examples, the quality of the model's prediction with respect to ground truth seems to be good. In figure 11, image "b", the model is able to predict small objects like poles, traffic signs, etc. correctly. However, it struggles with detecting small objects that are far away in the input image. In figure 12, the model predictions on all the lane markings are good and accurate. And in figure 13, the drivable area model's prediction of the non-drivable area seems to be poor.

Sl.No	Task	Dice coefficient
1	semantic segmentation	68.50%
2	lane marking	79.40%
3	drivable area	89.83%

Table 5: Dice coefficient of segmentation tasks

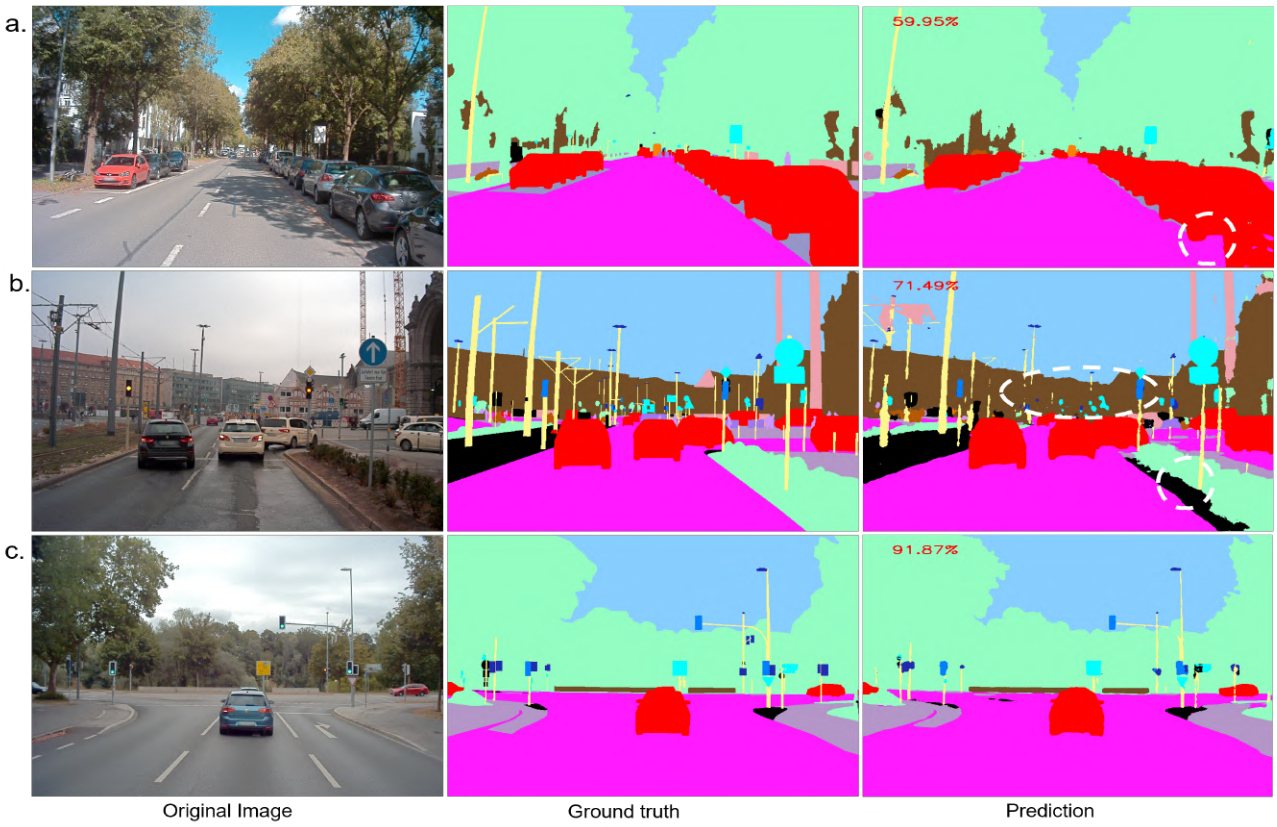

Figure 11: Semantic segmentation prediction examples on the A2D2 test set. The score written on each prediction image indicates the dice coefficient with respect to the ground truth. The white circles indicate some of the bad predictions with respect to ground truth.

Figure 12: Lane marking prediction examples on the A2D2 test set. The score written on each prediction image indicates the dice coefficient with respect to the ground truth. The red circles indicate some of the bad predictions with respect to ground truth.

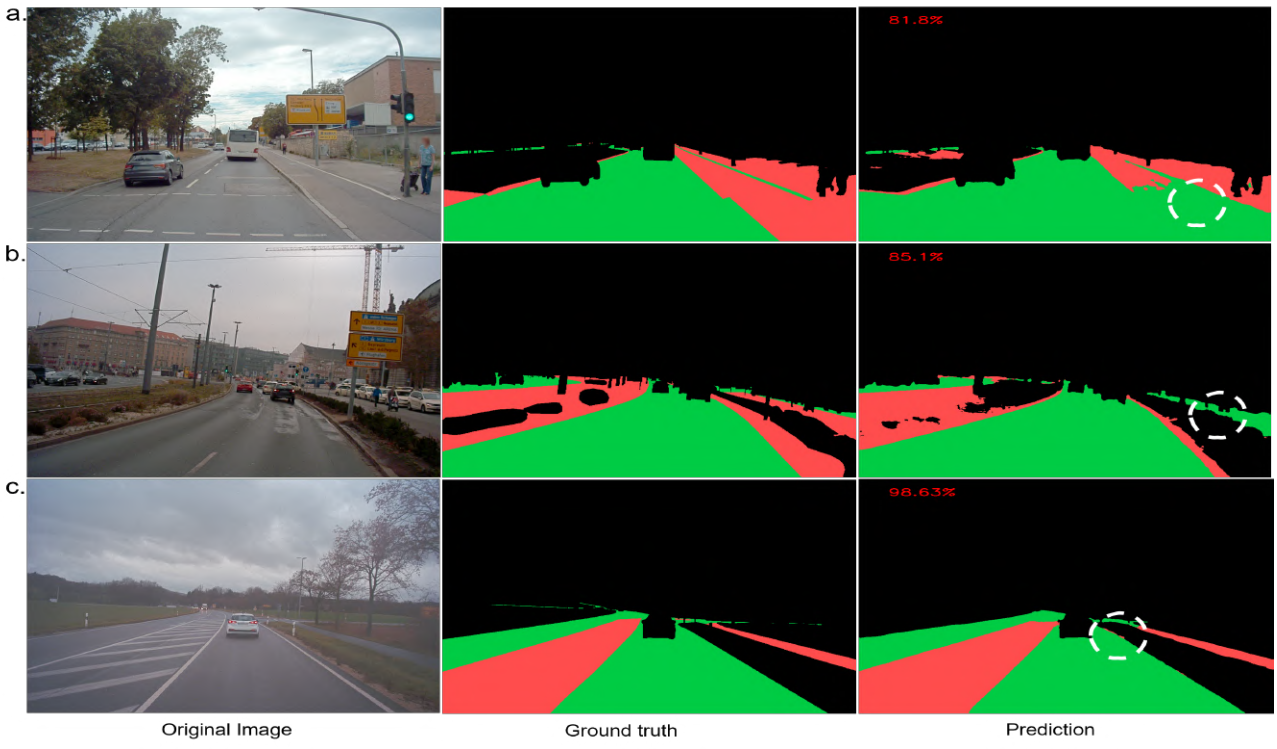

Figure 13: Drivable area prediction examples on the A2D2 test set. The score written on each prediction image indicates the dice coefficient with respect to the ground truth. The white circles indicate some of the bad predictions with respect to ground truth.

4.2.2 Object Detection

Model

The model architecture for object detection is illustrated in figure 14. The model is based on the YOLOv3 architecture [23]. YOLOv3 is one of the best object detection architectures and it is an improved version of YOLO [17] and YOLOv2 [51]. To have a common feature extractor across all the models, the original darknet-53 [4] feature extractor is replaced with resnet-34 [47] pre-trained on the Image-Net dataset [48] and the stride of conv1 of the resnet is changed to 1. To match the dimensions needed for scale predictions, the kernel size and stride of the first convolution layer after the resnet layer4 is changed to (2,2), refer to figure 14. For each input, the model outputs 3 predictions at 3 different scales (grid sizes) and 3 bounding boxes per grid cell. Each bounding box prediction has 5 + number of classes values. The first 5 values correspond to the confidence score, x, y, w, and h of the bounding box, and the remaining values are one hot prediction of the bounding box class. Here, 'x' and 'y' refer to the bounding box center, 'w' and "h" refer to the width and height of the bounding box. The different grid sizes at scale predictions ensure the model learns to predict well even for small target bounding boxes.

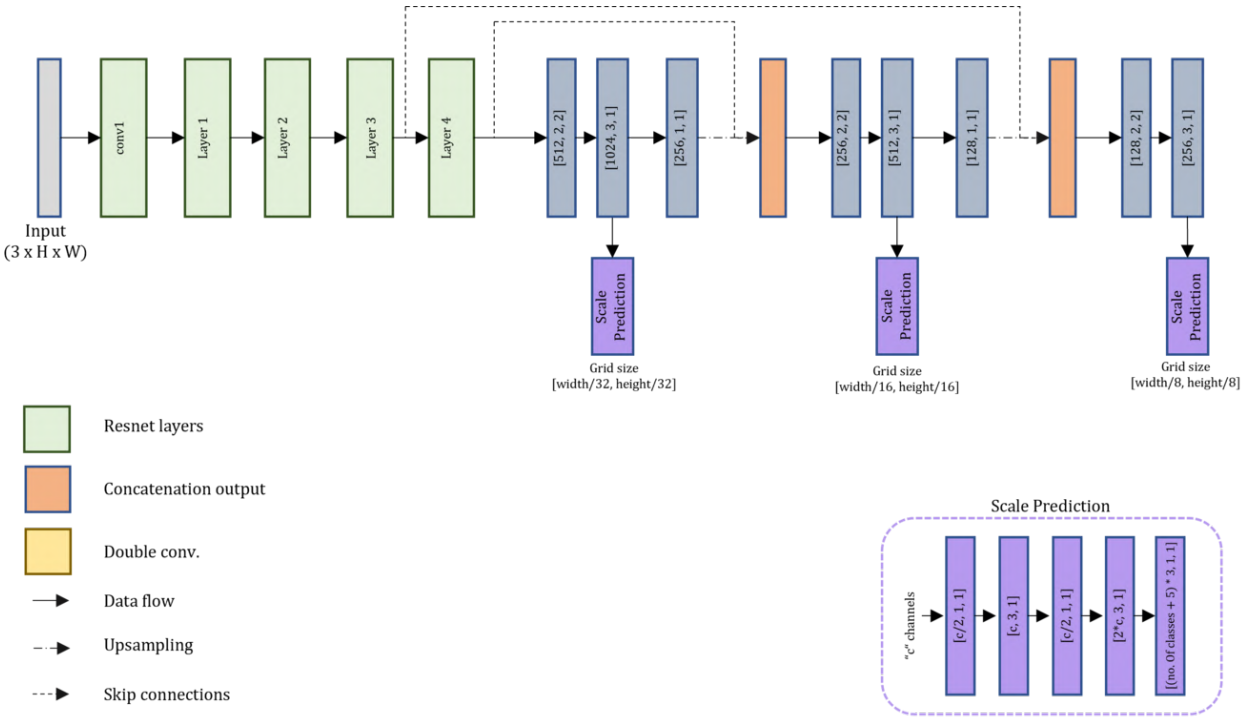

Figure 14: YOLOv3 model with resnet-34 as feature extractor. The parameters mentioned in some of the blocks correspond to [output channels, kernel size, stride].

Loss

The loss function for YOLOv3 is illustrated in the equation (8). YOLOv3 uses anchor boxes for the prediction of the width and height of the bounding box. There are a total of 9 anchor boxes (3 at each scale) corresponding to different bounding box sizes across the dataset, and each anchor box contains a predefined width and height. The anchor boxes help the model learn to predict bounding boxes as offsets from predefined values, which is easier to learn than learning to predict from scratch. Here, the anchor boxes used are from the original YOLOv3 implementation [23]. The model predicts p_x,p_y as bounding box center coordinates, p_w and p_h are width and height offset from the predefined anchor's width(w_{anchor}) and height(h_{anchor}). The final bounding box

coordinates are calculated using the equation (7).

$$
\begin{aligned}
b_x &= \sigma(p_x) \\
b_y &= \sigma(p_y) \\
b_w &= w_{anchor} \cdot e^{(p_w)} \\
b_h &= h_{anchor} \cdot e^{(p_h)}
\end{aligned}
\tag{7}
$$

The YOLO loss function has 4 major parts as illustrated in equation (8) and each part is weighted with constant λ. In our experiments, the values for $\lambda_{no,obj}, \lambda_{obj}, \lambda_{box}$ and λ_{class} were 1, 1.5, 1.5, and 1.5, respectively. These values were selected so as to give more importance to the model's class, confidence, and bounding box coordinates prediction. One can experiment with these values to prioritize the different parts of the loss function. In the original YOLOv3, all the constants were set to 1.

$$
\begin{aligned}
Yolo\ loss\ &= \lambda_{no,obj}\ [-(c_{gt}\log(c_{pred}) + (1-c_{gt})\log(1-c_{pred}))] && \rightarrow no\ object\ loss \\
&+ \lambda_{obj}\ [\textstyle\sum_{i=1}^{D} |c_{pred} - (IOU \cdot c_{gt})|] && \rightarrow object\ loss \\
&+ \lambda_{box}\ [\textstyle\sum_{i=1}^{D} |b_{pred} - b_{gt}] && \rightarrow box\ coordinates\ loss \\
&+ \lambda_{class}\ [-\textstyle\sum_{c=1}^{C} y_{o,c}\log(p_{o,c})] && \rightarrow class\ loss
\end{aligned}
\tag{8}
$$

The first part, *no object loss*, is the binary cross entropy loss between the model's confidence score c_{pred} and ground truth c_{gt} for all the grid cells where there is no object. The second part, *object loss*, is the Mean Squared Error(MSE) between the model's confidence score c_{pred} and ground truth c_{gt} for all the grid cells where there is an object. The ground truth c_{gt} is multiplied with IOU between the predicted and ground truth bounding box, so that the model's confidence score should correspond to IOU and not just the value 0 or 1 as in previous YOLO versions. The first two parts help the model learn to predict whether an object center is present or not in a grid cell. The third part, *box coordinate loss*, is MSE between predicted b_{pred} and ground truth b_{gt} bounding box coordinates. And the last part, *class loss*, is cross entropy loss between predicted probability $p_{o,c}$ and ground truth $y_{o,c}$.

Training Details

The input to the model is an RGB image of size 704x704, paired with corresponding 2D bounding box coordinates in YOLO format. The model is trained using the Adam optimizer with a learning rate of 0.00001 and weight decay of 0.0005 to avoid over-fitting. The same data augmentations were applied to the training data as in the segmentation task training, refer to section 4.2.1. During training, after each epoch, the model's mean Average Precision (mAP) on test data is calculated, and the model with the highest mAP score is considered the final model. The YOLOv3 was trained for 132 epochs, and the model with the best mAP was achieved at epoch 95. The training loss and accuracy graphs are shown in section 9, figure 38.

Results

The results of primary task object detection is shown in the table 6 below. The class-wise accuracies are shown in the appendix section 9, figure 34. In A2D2 dataset, not all the objects in the image are provided with bounding boxes, and some of the objects (usually far from the ego vehicle) are not labeled. In figure 15, example b, It can be noticed that two cars on the left side do not have ground truth bounding boxes, but the model has predicted those two cars and can be seen in the prediction image. These two predictions will be considered false positives, leading to a low mAP even though the model has learned to detect all the objects correctly. For better comparison with respect to the ground truth and multi-task model predictions, a bounding box accuracy is calculated as illustrated in the equation (9).

$$bb\_accuracy = \frac{TP}{FP \ + \ GT} \tag{9}$$

The bounding box accuracy provides a score for each prediction by taking into account the total number of ground truth (GT) boxes, false positives (FP), and true positives (TP).

Sl.No	Task	mAP@0.5
1	object detection	34.9%

Table 6: object detection model performance on A2D2 test set.

Figure 15: Object detection prediction examples on A2D2 test set. On each prediction, the total number of GT, FP, TP and bb_accuracy is mentioned. The red circles indicate some of the bad predictions with respect to ground truth.

4.3 Multi-Task Neural Network Implementation

4.3.1 Model

The multi-task model architecture is illustrated in figure 16. The model architecture is of the shared trunk type as described in the papers here [25] [29] and discussed in section 3.4. The model has a global feature extractor (shared trunk) as encoder followed by decoders, one for each task. The encoder and decoders have the same dimensions as used in STNN models (figure 14 9) to maintain uniformity and for proper comparison with STNN's performance. The skip connections from the encoder are used by each decoder to produce the corresponding task output. The model shown in figure 16 solves our four primary perception tasks simultaneously. However, to investigate and understand the idea of finding the best tasks to learn together and finding a task that favors every other task in a multi-task learning scenario as discussed in section 3.5, 11 multi-task models with different combinations of our primary tasks are implemented, refer to table 9. There are six multi-task models that solve two tasks, four multi-task models that solve three tasks, and one multi-task model that solves four tasks simultaneously. All the different combination models are implemented the same as shown in figure 16, with only those decoders that correspond to the tasks in the combination present and other decoders removed.

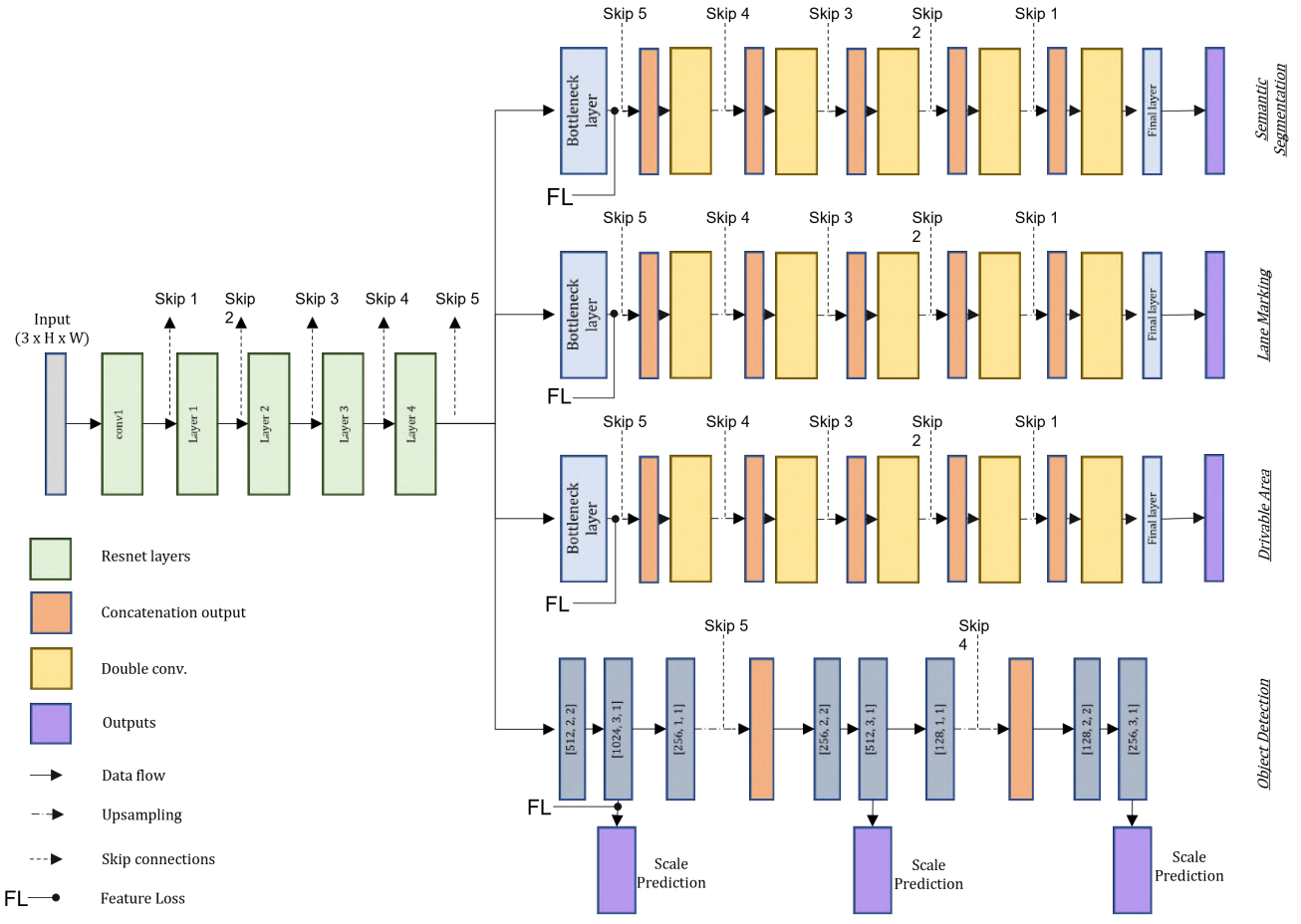

Figure 16: A shared trunk type multi-task model that solves semantic segmentation, lane marking, drivable area, object detection tasks simultaneously.

4.3.2 Loss

The loss function for our MTNN can be as simple as the sum of all the losses from individual decoders. The optimization of the encoder (global feature extractor) depends on the gradients

from each task (decoder) during training. This demonstrates the concept of different tasks in MTNNs working together for better generalization. In other words, the encoder optimization is influenced by all the tasks in the MTNN. The higher the loss of one task, the stronger the gradients, and the more significant influence of that task in optimizing the shared encoder. If one of the task losses is much higher than the losses of all the other tasks, the encoder's optimization is biased towards that task. Hence, loss weighing becomes the most important aspect of training MTNNs. To investigate the influence of losses on MTNN training, experiments were done on five different losses before deciding on the final loss function, refer to table 7.

Sl.No	**Task**	**Semantic segmentation**	**Object detection**	**Loss**	**Description**
1	sem+det	62.99(-5.5)	37.39(+2.49)	Loss1, equation (10)	Sum of individual losses
2		63.86(-4.64)	41.03(+6.03)	Loss2, equation (11)	Weighted sum of individual losses
3		63.98(-4.52)	41.03(+6.13)	Loss3 equation (13)	Weighted sum of individual losses
4		61.89(-6.61)	**41.86(+6.96)**	Loss4, equation (15)	Sum of individual losses + Feature loss
5		**64.67(-3.83)**	41.24(+6.34)	Final loss, equation (16)	Weighted sum of individual losses + Feature loss

Table 7: Table shows experiment results of semantic segmentation and object detection MTNN on different loss functions. Column two and three show the dice coefficient for semantic segmentation and mAP@0.5 for object detection respectively. Bold values indicate the highest accuracy. The deviation of each accuracy from its corresponding STNN is indicated in brackets.

The experiment was done on semantic segmentation and object detection MTNN because these two differed the most with respect to their loss functions. The first loss is the simplest of them all, it's the total sum of individual losses as illustrated in equation (10).

$$Loss1 = \sum_{t=1}^{T} (loss_t) \tag{10}$$

where T is the total number of tasks in the multi-task model, and loss_t is individual loss of task t. During our experiment, it was observed that object detection loss was higher when compared to semantic segmentation loss. Object detection loss starts at approximately 5 and semantic segmentation at approximately 1.5. As explained earlier, object detection has a greater influence on optimizing the encoder. The results also indicate the same, because the improvement in accuracy was observed only in object detection. Refer to table 7, serial number 1. To improve further, the individual losses were weighted using the equation (12), where w1_t is weight for task t. Loss2 is the weighted sum of individual losses as illustrated in the equation (11).

$$Loss2 = \sum_{t=1}^{T} (w1_t \cdot loss_t) \tag{11}$$

$$w1_t = \frac{\sum_{t=1}^{T} (loss_t)}{T \cdot \ loss_t} \tag{12}$$

There was an improvement in both semantic segmentation and object detection accuracy compared to loss1 training. Refer to table 7, serial number 2. However, the semantic segmentation accuracy was low when compared to its STNN accuracy. During our experiments, it was observed that some of the tasks (decoders) needed fewer epochs to be optimized when compared to other tasks. This depends on the complexity of the task that the decoder needs to solve. In the scenario where one of the tasks has already been optimized early during the training and other tasks still need more epochs to train, the individual loss value of this optimized task will always be very low,

leading to a large computed weight. The weights for the other tasks will be low as they are yet to be optimized. Due to this weighing scenario, only the task with a large weight is prioritized during the training. This impacts the learning of other tasks as well as the optimization of the shared encoder. To avoid this, in our next loss function, each calculated weight is added with 1 (equation (14)), ensuring that even tasks with low weights are optimized properly irrespective of the other task weights and also influence in optimization of the encoder throughout the training, refer to equation (13).

$$Loss3 = \sum_{t=1}^{T}(w2_t \cdot loss_t) \tag{13}$$

$$w2_t = \frac{\sum_{t=1}^{T}(loss_t)}{T \cdot \ loss_t} \ + \ 1 \tag{14}$$

The results show that weighing the loss with equation (14) yields slightly better results than equation (12), refer to table 7, serial number 3.

The feature representation from the encoder has a great influence on the individual task performance. The features required by each decoder differ depending on the type of task. However, all our primary tasks are similar to each other, and they can benefit from each other's feature representations. Hence, loss4 is a combination of feature loss and individual loss, as illustrated in equation (15).

$$Loss4 = \sum_{t=1}^{T}(loss_t) \ + \ \sum_{t=1}^{T}((\frac{\sum_{t=1}^{T}[f_t]}{T}) - f_t)^2 \tag{15}$$

The first part is the sum of individual task losses, and the second part is the feature loss. The feature loss is the total sum of MSE between the average and individual task features. These features are the intermediate outputs from individual decoders. For the segmentation tasks, the features are the output from the bottleneck layer, and for object detection, it is the features before first scale prediction, as illustrated in figure 16. At the end of the training, feature loss ensures that features from the encoder are a generalized version of the individual features required by each decoder. The results show that there is further improvement in object detection accuracy but a further reduction in semantic segmentation accuracy. Refer to table 7 serial number 4. To improve optimization further, the first part of equation (15) was replaced with the weighted sum of individual losses as illustrated in the equation (16).

$$Final\ Loss = \sum_{t=1}^{T}(w2_t \cdot loss_t) \ + \ \sum_{t=1}^{T}((\frac{\sum_{t=1}^{T}[f_t]}{T}) - f_t)^2 \tag{16}$$

There was further improvement observed in semantic segmentation accuracy. It was the best when compared to all the other experiment results as shown in the table 7, serial number 5.

4.3.3 Training Details

The input to the model is an RGB image of size 704x704, paired with corresponding single task target labels. The model is trained using the Adam optimizer with a weight decay of 0.0005 to avoid over-fitting. The data augmentation applied to training data is the same as across all the MTNN's training and augmentation tasks. Refer to section 4.2.1. During the training, it was observed that some of the task accuracies were lower with respect to corresponding STNN accuracy. To improve the accuracy of these tasks, the complete model is trained in two stages. The first stage focuses on optimizing the shared encoder, and the second stage focuses on fine-tuning the decoders based on the learned encoder.

Stage1 : Optimizing Feature Extractor.

The model is trained with final loss (equation 16) and learning rate of 0.00001. As described in the previous section, the final loss function will ensure that the encoder is dynamically optimized for every batch. After each epoch, the model performance on the test data is measured. The performance score of the MTNN will be the average of all the individual task performance scores. The metrics for calculating individual task scores are the same as those used in STNNs. The model with the highest average performance score will be considered as the final model.

Stage2 : Optimizing Task Decoders.

At this stage, the decoders are fine-tuned with a smaller learning rate. The learning rate is changed to 0.000002 based on the trial and error method. The idea was to select a learning rate significantly lower than the stage 1 learning rate. The shared encoder weights are frozen and the model is trained with loss1, as illustrated in the equation (10). This modification in the training will ensure that the learned encoder is intact and the decoders are fine-tuned based on the learned feature representations from the encoder. After each epoch, the model average performance score on the test data is measured, and the model with the highest average performance score will be considered the final model.

		stage 1 epochs		**stage 2 epochs**	
Sl.No	**Multi-Tasks**	**total**	**best**	**total**	**best**
1	sem+lane	143	122	37	29
2	sem+dri	142	121	28	12
3	sem+det	214	187	17	7
4	lane+det	156	136	23	3
5	dri+det	197	175	17	8
6	lane+dri	147	121	41	30
7	sem+lane+dri	134	114	20	1
8	sem+lane+det	165	137	21	4
9	sem+dri+det	193	171	19	10
10	lane+dri+det	180	160	54	30
11	sem+lane+dri+det	196	173	16	1

Table 8: Table shows the total number of epochs trained. The column best indicates the epoch at which the corresponding MTNN's performance was highest. sem: semantic segmentation, lane: lane marking, dri: drivable area, det: object detection

The number of training epochs depends on the combination and number of tasks of the MTNN. The total number of epochs and the best epoch when the MTNN's accuracy was highest are shown in the table 8. In general, in stage 1, the training was continued until there was no improvement in the MTNN's performance, which was observed for 20 epochs approximately. And in stage 2, it was approximately 10 epochs. The training loss and accuracy graph for some of MTNN training is show in the section 9.

4.3.4 Results

The results of 11 different MTNNs are shown in the table 9. The results clearly demonstrate the concept of multi-task learning, in which tasks learn from one another. As shown in the table 1, Some of the target labels are shared by semantic segmentation, lane marking, and drivable area. The semantic segmentation "road" class, for example, has all of the lane marking labels. As a result, if semantic segmentation learns to correctly detect roads, lane marking will find it easier to learn with common feature representations. Similarly, from semantic segmentation, object detection can learn to localize some objects such as cars, utility vehicles, people, and so on. In general, tasks having a common area of interest in feature space favor each other. The results

Sl.No	Multi-Tasks	Accuracy				Training stage
		Semantic segmentation	**Lane marking**	**Drivable area**	**Object detection**	
1	sem+lane	**69.07(+0.57)**	80.92(+1.52)			stage 2
2	sem+dri	68.02(-0.48)		**91.07(+1.24)**		stage 2
3	sem+det	65.09(-3.41)			**46.72(+11.82)**	stage 2
4	lane+det		77.86(-1.54)		35.60(+0.7)	stage 2
5	dri+det			89.29(-0.54)	40.42(+5.52)	stage 1
6	lane+dri		**81.95(+2.55)**	90.65(+0.82)		stage 2
7	sem+lane+dri	68.64(+0.14)	81.07(+1.67)	90.87(+1.04)		stage 2
8	sem+lane+det	64.15(-4.35)	78.80(-0.6)		41.63(+6.73)	stage 2
9	sem+dri+det	64.82(-3.68)		89.63(-0.2)	41.63(+6.73)	stage 2
10	lane+dri+det		79.04(-0.36)	89.05(-0.78)	40.10(+5.2)	stage 1
11	sem+lane+dri+det	63.63(-4.87)	79.68(+0.24)	89.62(-0.21)	41.39(+6.49)	stage 1

Table 9: The accuracy of MTNN for all combinations is shown in the above table. The green cells represent an increase in accuracy for a single task in that multi-task combination, whereas the red cells represent a decrease in accuracy. The deviation of each accuracy from its corresponding STNN is indicated in brackets. The accuracy mentioned in bold indicates the maximum improvement of that single task accuracy. The training stage from which the results are collected is shown in the last column. sem: semantic segmentation, lane: lane marking, dri: drivable area, det: object detection.

shown here are based on the highest accuracy averaged over all of the individual task accuracies. During stage 2 training of serial numbers 5, 10, and 11 MTNNs, an improvement was observed in one of the tasks when compared to stage 1 results. However, the average accuracy at stage 2 was lower than at stage 1, hence the results.

Semantic segmentation: Except for the semantic segmentation and lane marking MTNN combination, none of the combinations show a considerable improvement. One hypothesis for these results is that some parts of feature representations from the encoder required for semantic segmentation prediction are lost due to our feature loss function. The feature loss aids the MTNN in learning more generic feature representations as a result of combining all of the other features. In most cases, adding a semantic segmentation task to the combination helps improve the accuracy of other tasks, but it loses its own accuracy. Hence, semantic segmentation can be considered the best or auxiliary task to train within a multi-task learning scenario.

Lane marking: Performance improvement is observed whenever it is paired with semantic segmentation, drivable area, or both. The maximum improvement is 2.55%, when it is paired only with drivable area. Refer to serial numbers 1, 6, and 7 in the table 9.

Drivable area: Performance improvement is observed whenever it is paired with semantic segmentation, lane marking, or both. The accuracy is maximum when it is paired only with semantic segmentation. refer to serial numbers 2, 6, and 7 in the table 9.

Object detection: There has been a significant improvement in each case. The accuracy improves the most with semantic segmentation and the least when only lane marking is part of MTNN. It can be observed that other tasks lose their accuracy significantly whenever object detection is a part of the MTNN combination. Hence, object detection may be the worst task to train with, except where the focus is only on improving its own accuracy.

To summarize, MTL learning shows great promise in improving accuracy of individual tasks. Based on the results, there was a improvement in accuracy of semantic segmentation up to 1%, lane marking up to 2.5%, drivable area up to 1.5%, and object detection up to 12%.

Inference Time

Task	Inference time(in ms)
semantic segmentation	84
lane marking	83
drivable area	86
object detection	48.3

Table 10: STNNs inference time

Table 10 shows the inference time of each STNN. Table 11 shows the comparison of inference time between MTNN and STNNs. Each inference time calculated is averaged over 256 examples from A2D2. The tables clearly show that MTNNs are up to 33% faster than STNNs. The calculation was performed on an Intel (R) Core (TM) i7-10850H CPU, and the GPU was an NVIDIA Quadro RTX 5000 with Max-Q Design and 16GB of VRAM.

Inference time(in ms)			
Multi-Tasks	**multi-task**	**single task**	**percentage reduction**
sem+lane	135	167	19.2%
sem+dri	135	170	20.6%
sem+det	101	132.3	23.7%
lane+det	95	131.3	27.6%
dri+det	96.7	134.3	28.0%
lane+dri	136.5	169	19.2%
sem+lane+dri	194.2	253	23.2%
sem+lane+det	155.77	215.3	27.6%
sem+dri+det	152.3	218.3	30.2%
lane+dri+det	154.93	217.3	28.7%
sem+lane+dri+det	200.81	301.3	33.4%

Table 11: MTNNs inference time. Inference time mentioned under single task is the sum of inference time of individual task in the MTNN combination. The last column indicates total reduction in inference time when compared to corresponding STNNs

Predictions

The predictions of all MTNNs are shown in figures from 20 to 31.To have an idea of MTNN's predictions at a different level of accuracy, the predictions belong to the lower, middle, and higher sides of single task accuracy. The score written on each prediction image indicates the prediction accuracy with respect to the ground truth. For tasks semantic segmentation, lane marking, and drivable area, it is dice coefficient and bounding box accuracy (equation (9)) for object detection. Each prediction belongs to the MTNN combination indicated on the top-right of the image, except for object detection predictions, which are indicated on the bottom-center of the image. The white and red colored circles in each prediction image are indicated to compare the MTNNs predictions with respect to single task and GT. The prediction accuracy of each MTNN in these figures is appropriately higher or lower with respect to the results shown in table 9. In general, semantic segmentation performance is better with STNN, as most of the MTNNs accuracy for semantic segmentation is lower than STNN accuracy. Refer to figures 20, 21, and 22. The performance of the lane marking task is improved in terms of predicting fewer false positives compared to single task. Refer to red circles in figures 23, 24, and 25. For the drivable area task, non-drivable area prediction is improved in most of the MTNNs predictions compared to single task. Refer to white circles in the figures 26, 27, and 28. And object detection performance is improved in all the MTNN combinations. The number of FPs detected for each MTNN prediction is reduced compared to FPs in STNN prediction. Refer to FPs and bounding box accuracy of each prediction shown in the figures 29, 30, and 31.

(a) (b) (c)

Figure 17: Input images for MTNNs semantic segmentation prediction results shown in figures 20, 21, and 22 for (a), (b) and (c) respectively.

(a) (b) (c)

Figure 18: Input images for MTNNs lane marking prediction results shown in figures 23, 24, and 25 for (a), (b) and (c) respectively.

(a) (b) (c)

Figure 19: Input images for MTNNs drivable area prediction results shown in figures 26, 27, and 28 for (a), (b) and (c) respectively.

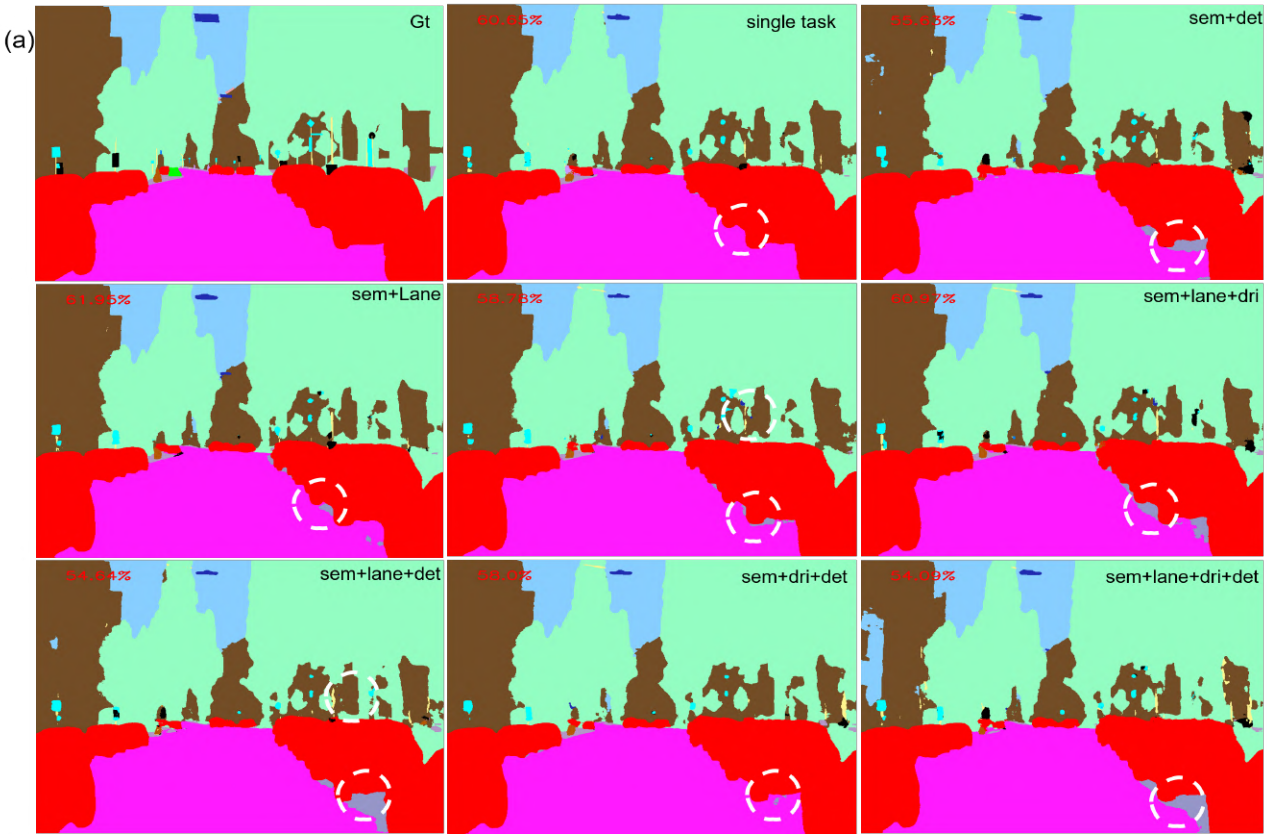

Figure 20: Semantic segmentation predictions of all MTNNs for input image (a) from figure 17. sem: semantic segmentation, lane: lane marking, dri: drivable area, det: object detection.

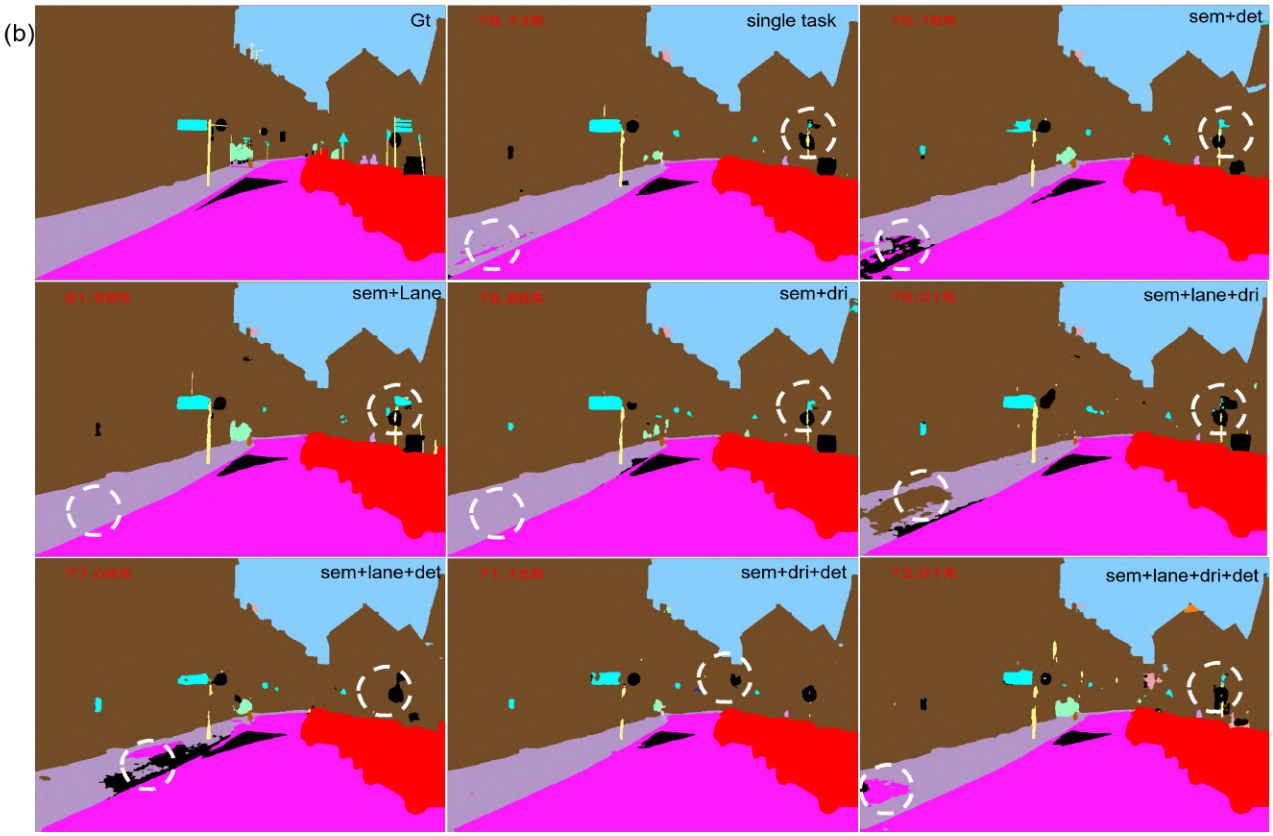

Figure 21: Semantic segmentation predictions of all MTNNs for input image (b) from figure 17. sem: semantic segmentation, lane: lane marking, dri: drivable area, det: object detection.

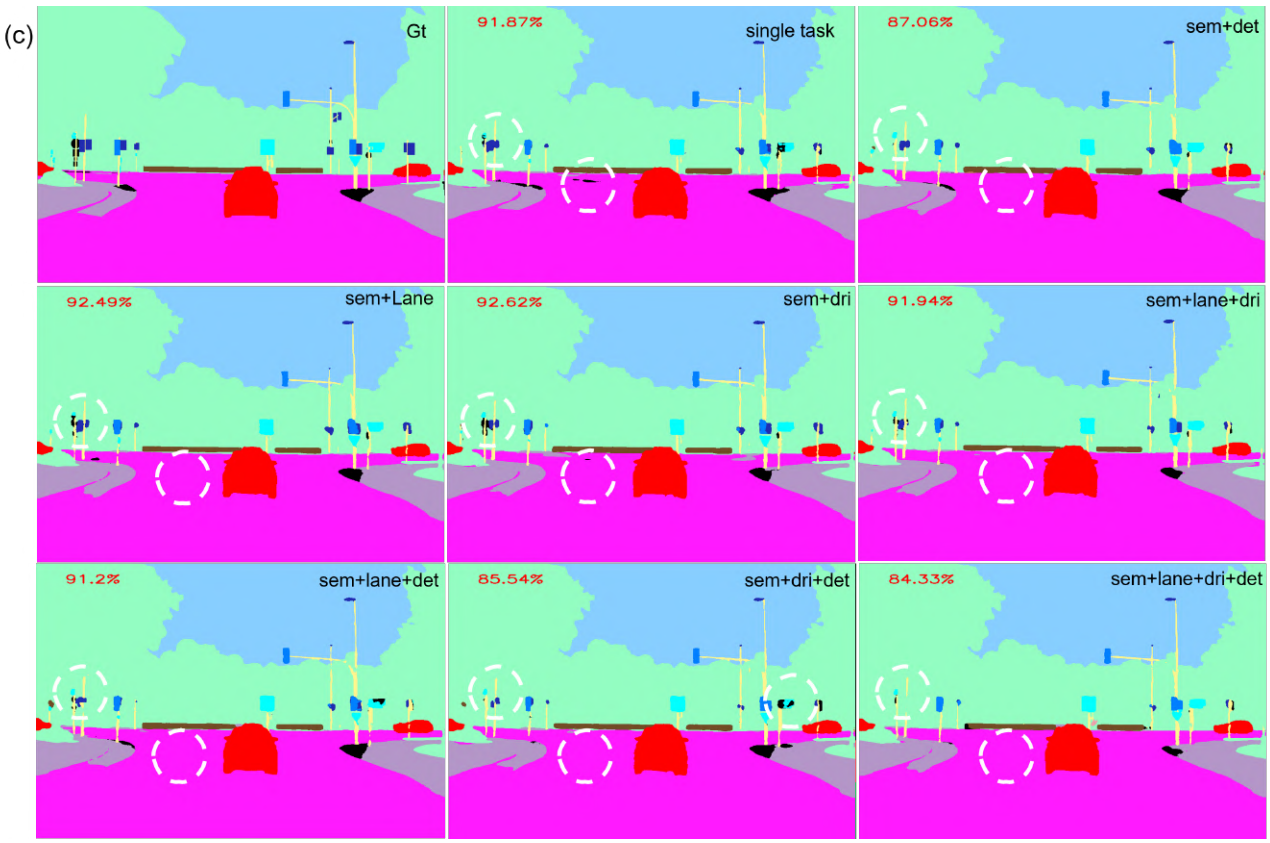

Figure 22: Semantic segmentation predictions of all MTNNs for input image (c) from figure 17. sem: semantic segmentation, lane: lane marking, dri: drivable area, det: object detection.

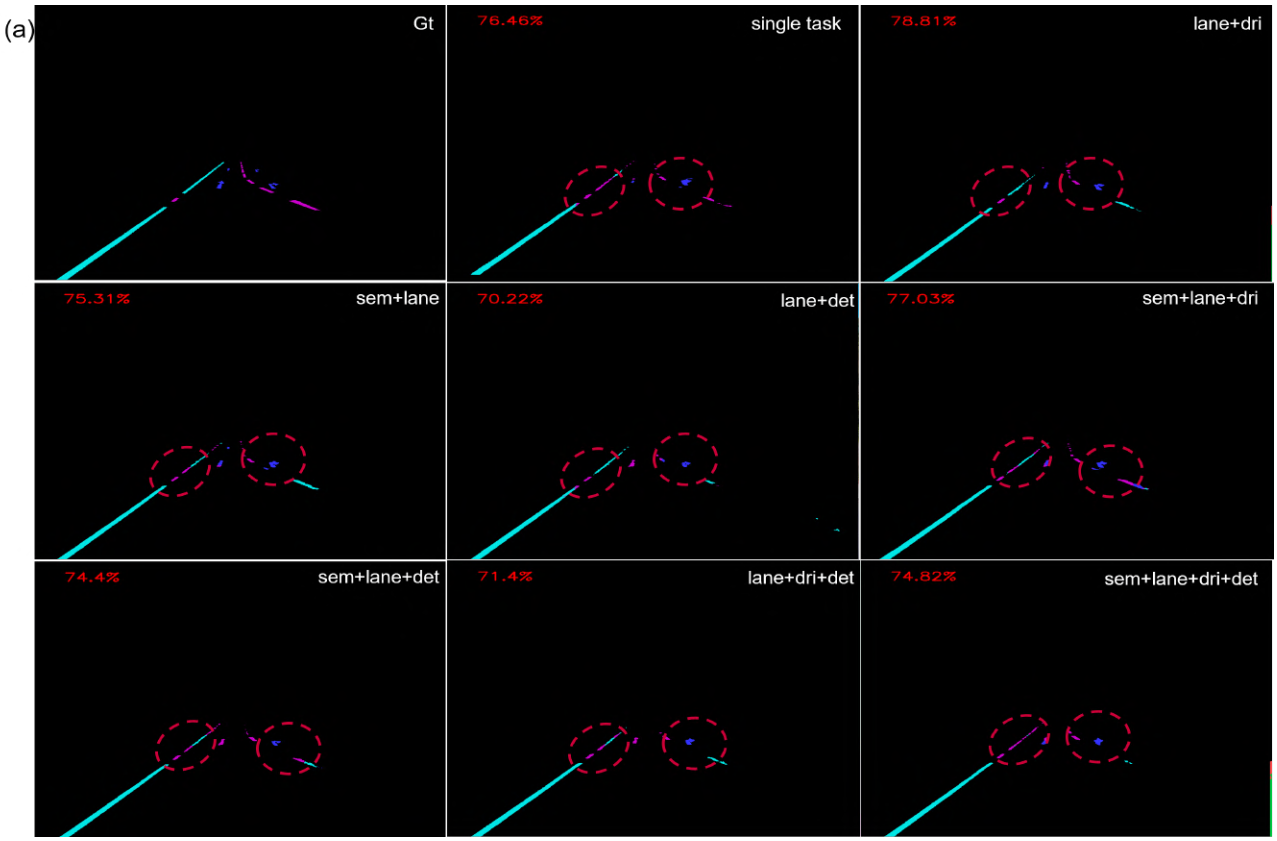

Figure 23: Lane marking predictions of all MTNNs for input image (a) from figure 18. sem: semantic segmentation, lane: lane marking, dri: drivable area, det: object detection.

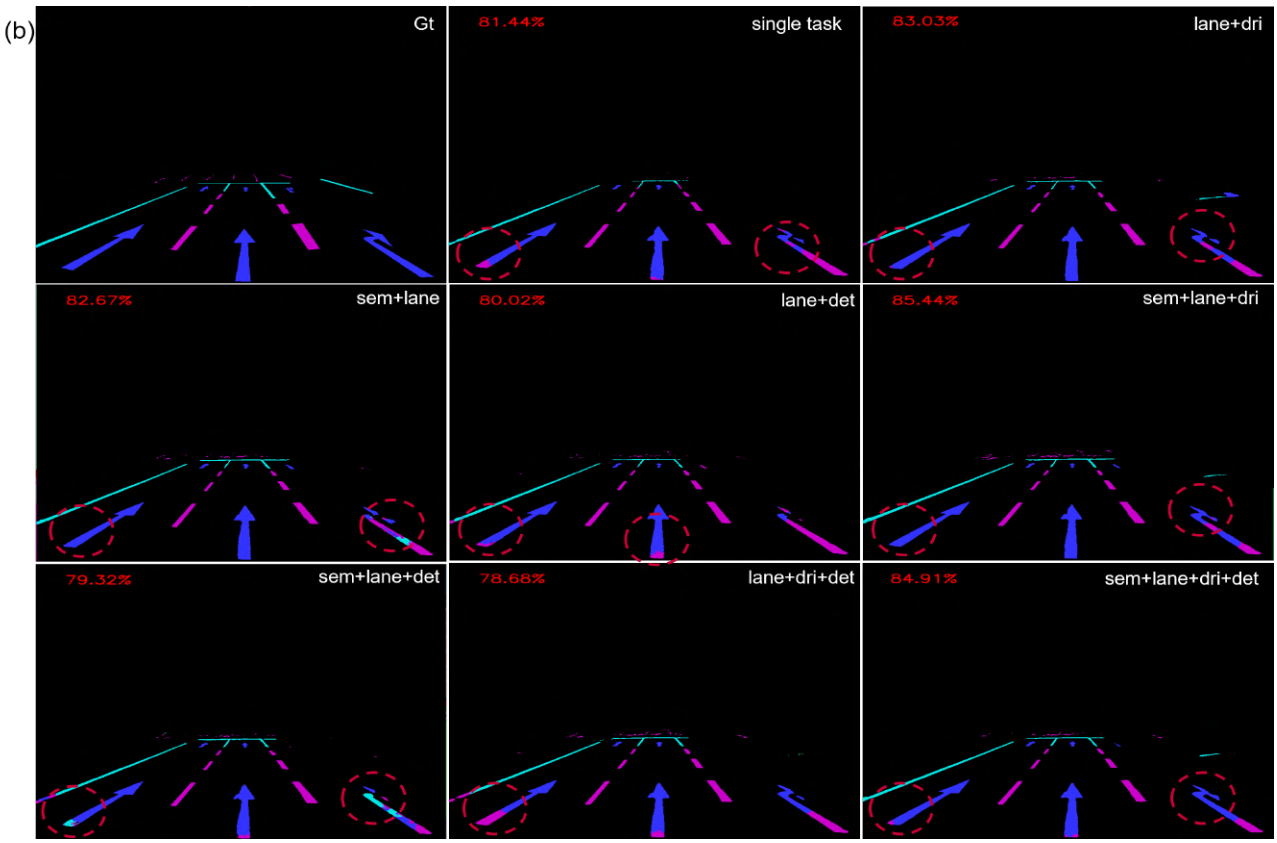

Figure 24: Lane marking predictions of all MTNNs for input image (b) from figure 18. sem: semantic segmentation, lane: lane marking, dri: drivable area, det: object detection.

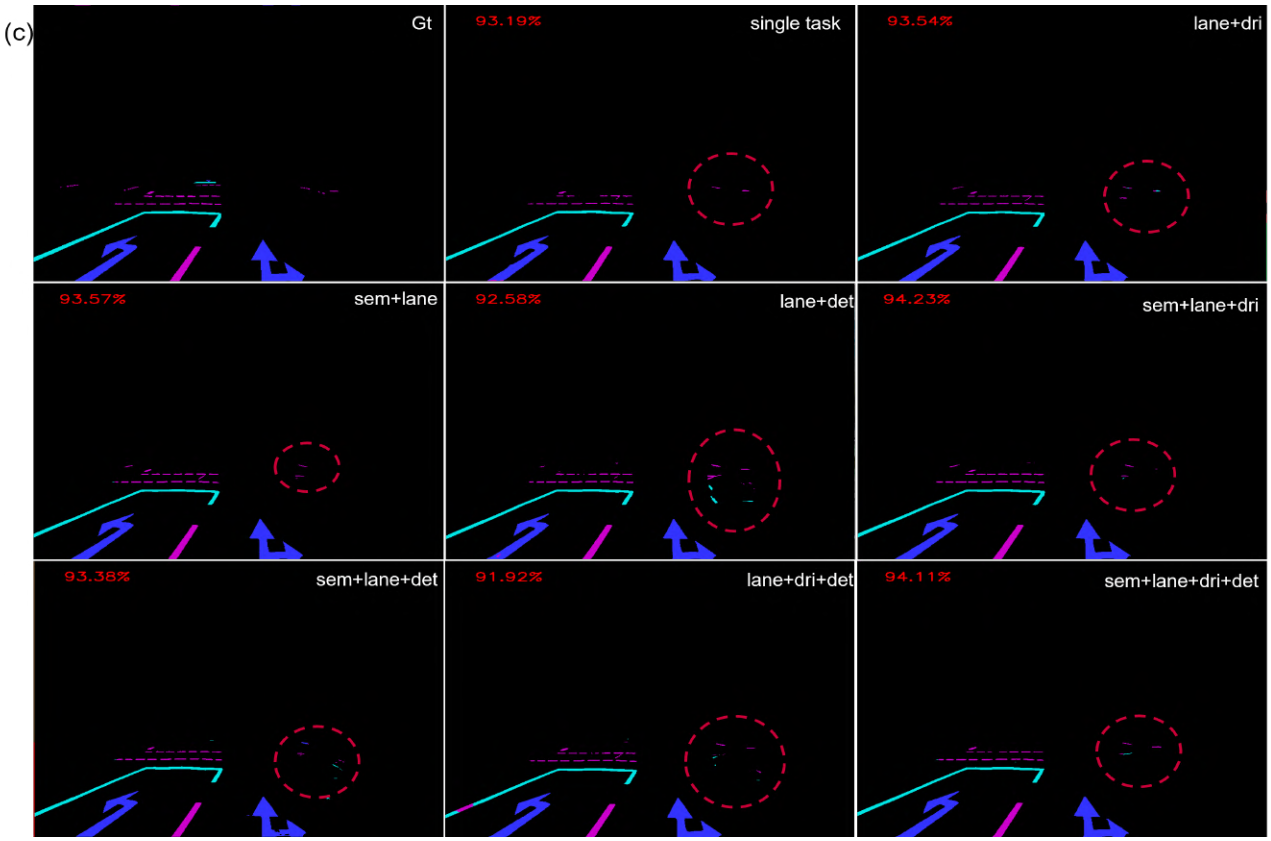

Figure 25: Lane marking predictions of all MTNNs for input image (c) from figure 18. sem: semantic segmentation, lane: lane marking, dri: drivable area, det: object detection.

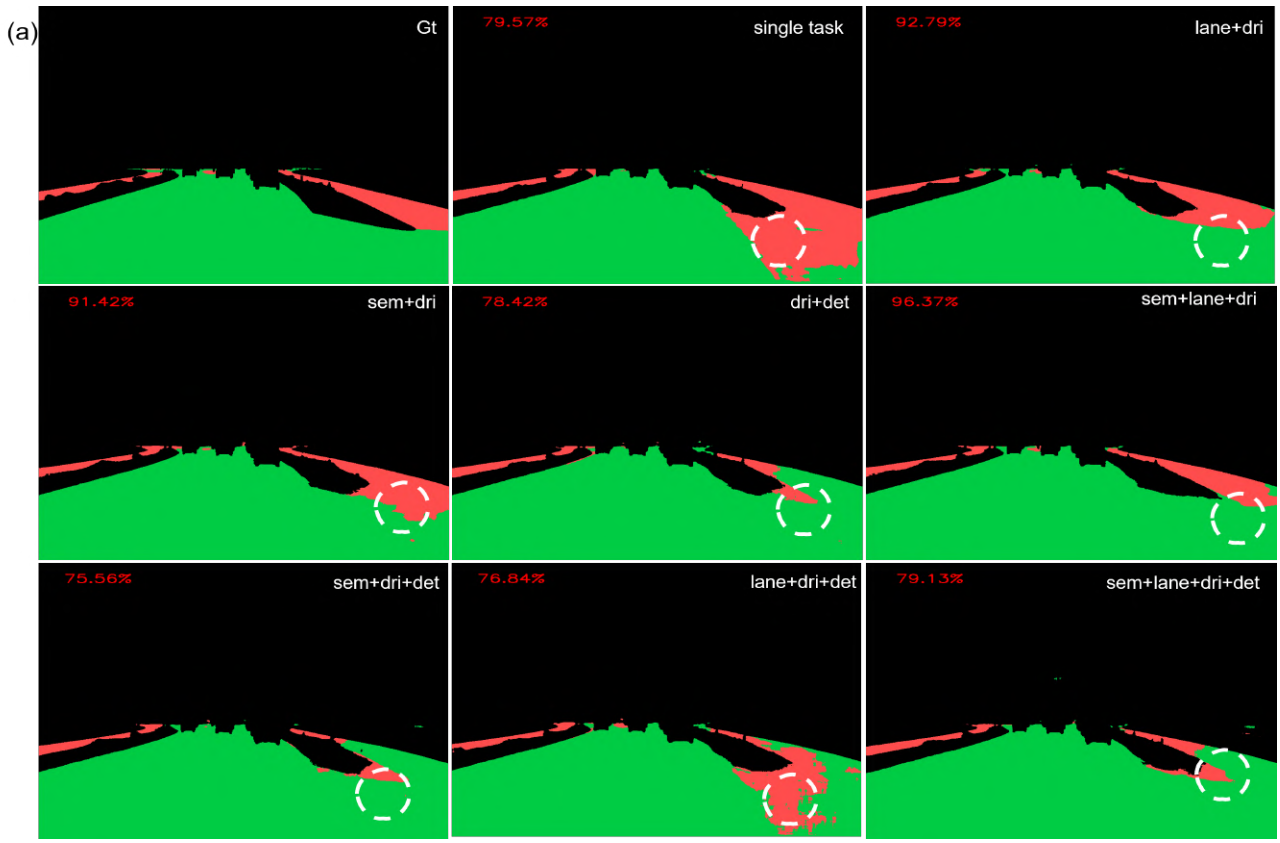

Figure 26: Drivable area predictions of all MTNNs for input image (a) from figure 19. sem: semantic segmentation, lane: lane marking, dri: drivable area, det: object detection.

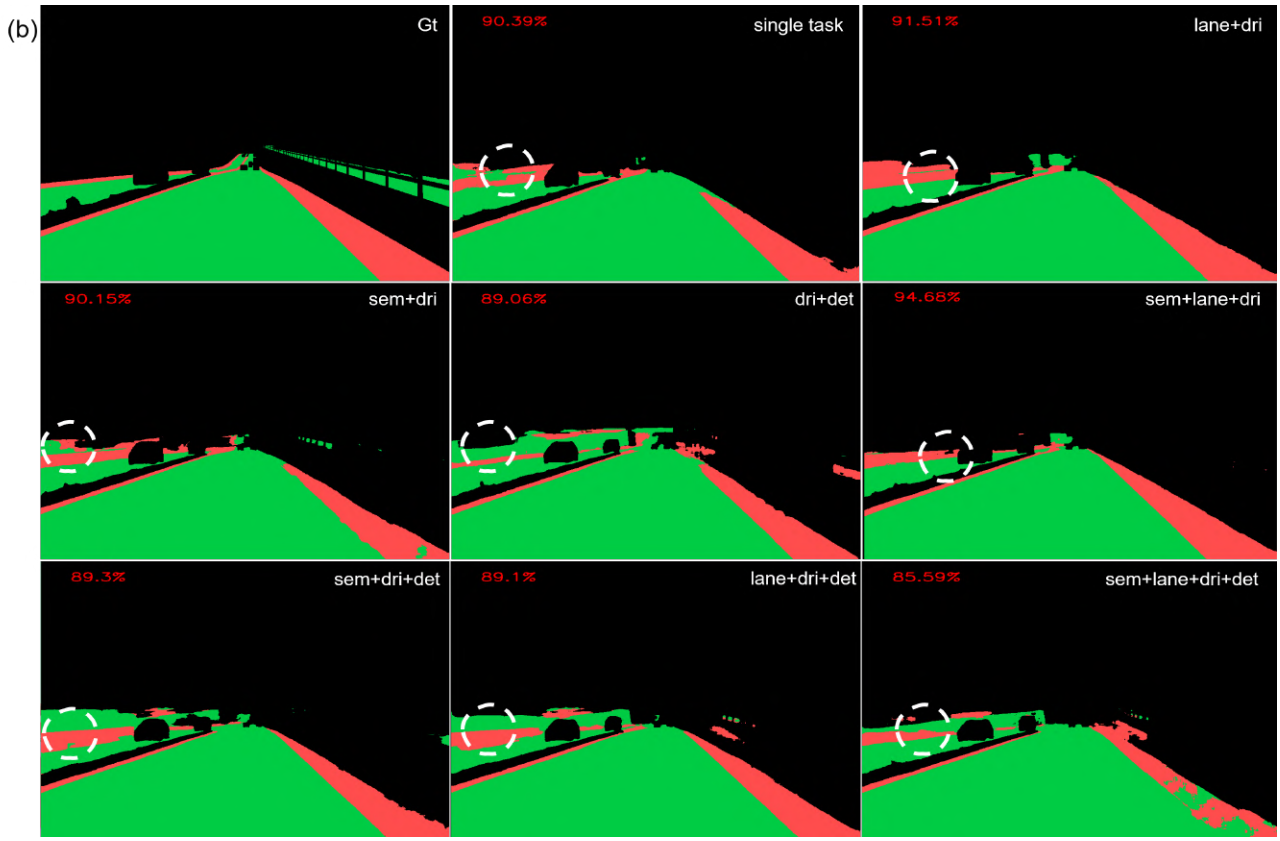

Figure 27: Drivable area predictions of all MTNNs for input image (b) from figure 19. sem: semantic segmentation, lane: lane marking, dri: drivable area, det: object detection.

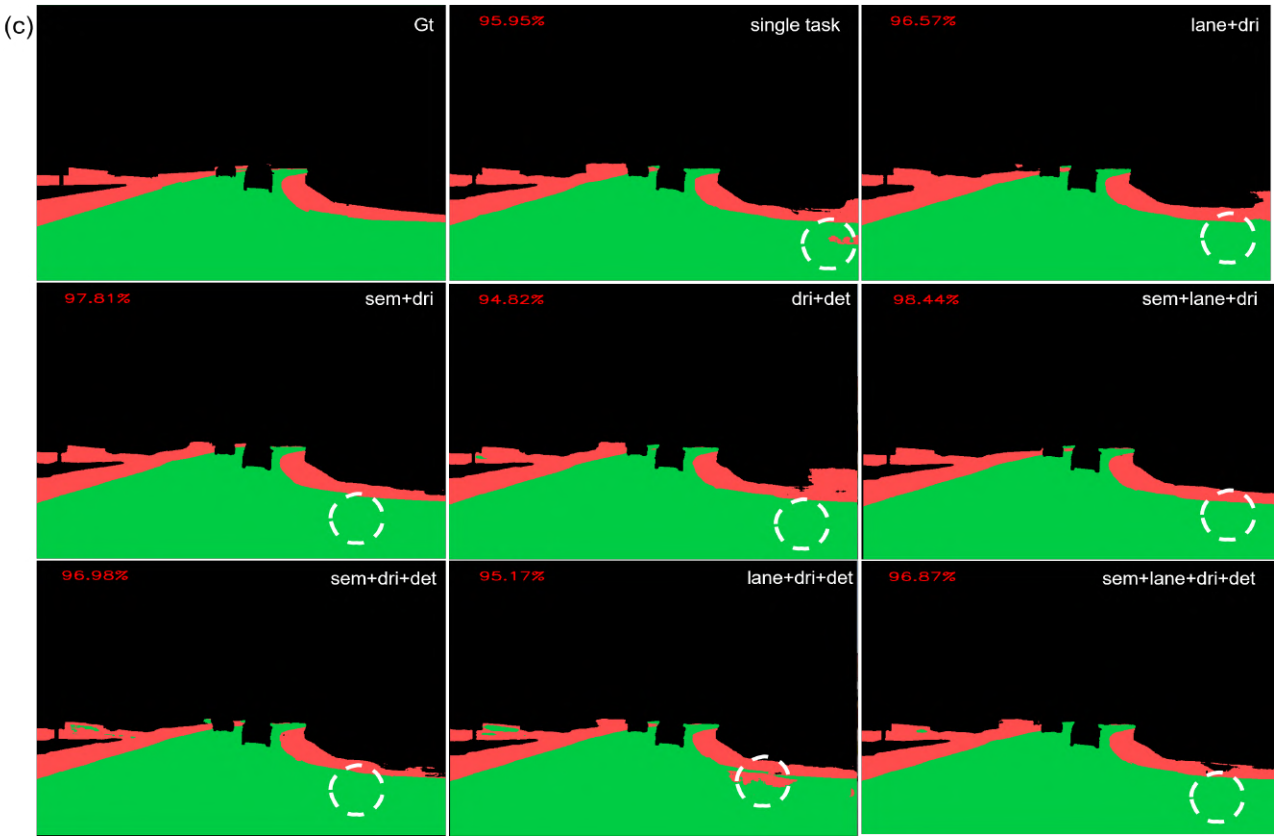

Figure 28: Drivable area predictions of all MTNNs for input image (c) from figure 19. sem: semantic segmentation, lane: lane marking, dri: drivable area, det: object detection.

Figure 29: An example of object detection predictions of all MTNNs. sem: semantic segmentation, lane: lane marking, dri: drivable area, det: object detection.

Figure 30: An example of object detection predictions of all MTNNs. sem: semantic segmentation, lane: lane marking, dri: drivable area, det: object detection.

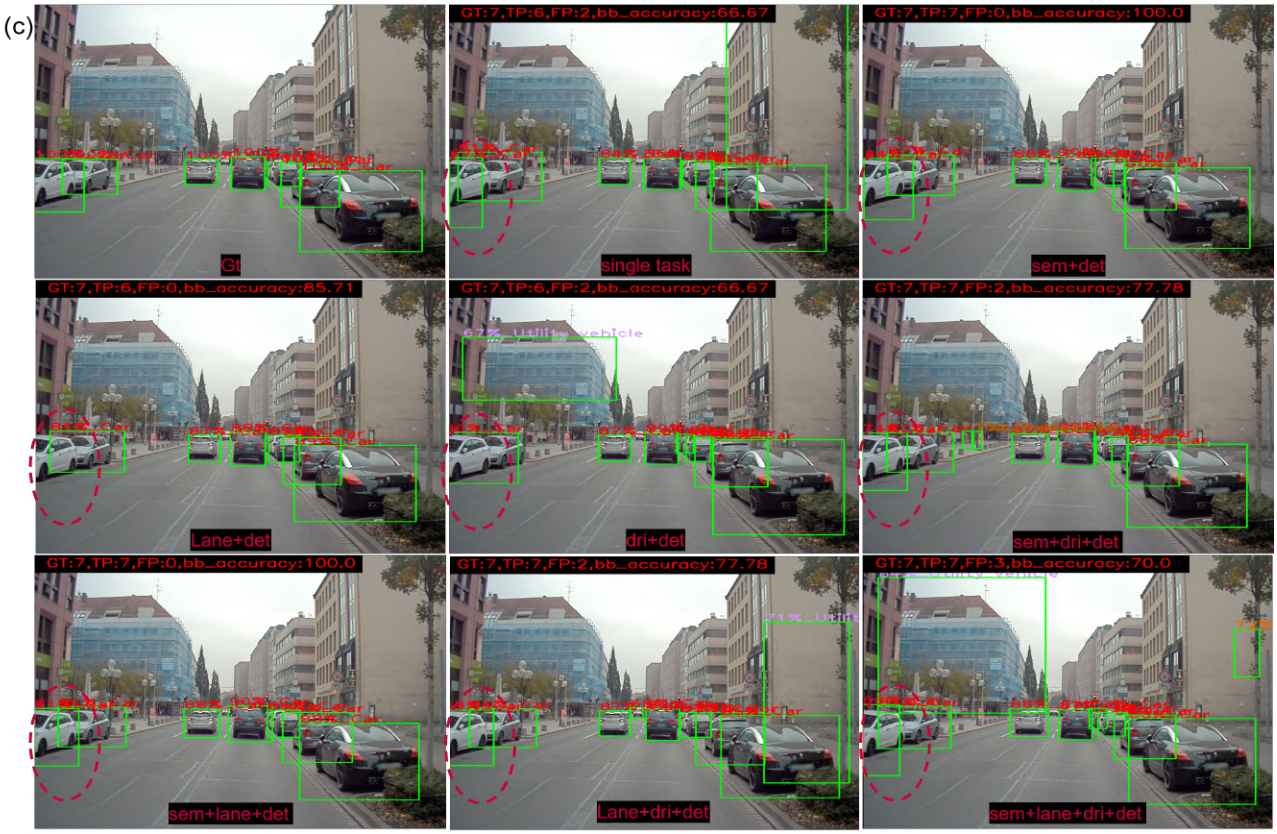

Figure 31: An example of object detection predictions of all MTNNs. sem: semantic segmentation, lane: lane marking, dri: drivable area, det: object detection.

5 Transfer Learning With Multi-Task Neural Network's Backbones

Transfer learning is a well-known method in deep learning that is applied to DNNs to improve training and performance. In this method, the weights of a pre-trained model DNN are used in the training of a new model. Generally, a pre-trained model is trained on a large dataset that is similar to the new task. This enables the new model to use generalized knowledge of a pre-trained model to improve its performance. In this study, all the STNNs and MTNNs were trained using this method. They were trained with the weights of a resnet-34 pre-trained on a large Image-Net classification dataset. The ImageNet is an image classification dataset that contains over a million training images for a thousand classes. In a classification task, a model learns to identify certain features in an image to classify it into a particular category. This knowledge of identifying features can be used in a localization problem, such as our primary tasks. Since our primary tasks aim to solve similar localization problems, the transfer learning method can be applied between the tasks. The idea here is that the shared encoders of MTNNs have more generalized knowledge when compared to the STNN encoders, and the shared encoder weights can be used to improve the performance of STNNs.

Tasks	Backbone	Result	Metric
Semantic segmentation	resnet-34	68.5	Dice Coefficient
	det encoder	68.22(-0.28)	
	Lane+dri+det	**68.69(+0.19)**	
Lane_marking	resnet-34	79.4	
	dri encoder	80.34(+0.94)	
	Sem+dri+det	**80.95(+1.55)**	
Drivable area	resnet-34	89.83	
	sem encoder	90.25(+0.42)	
	Sem+lane+det	**90.69(+0.86)**	
Object_detection	resnet-34	34.9	mAP@0.5
	sem encoder	36.43(+1.53)	
	Sem+lane+dri	**36.84(+1.94)**	

Table 12: The table shows the results of each STNN trained on different backbones (encoders). The cells in red indicate that the accuracy is lower compared to STTN's accuracy when trained with the resnet-34 backbone. And the green cells indicate an increase in the accuracy of that backbone. In brackets, the deviation of each accuracy from its corresponding STNN with resnet-34 backbone is indicated. sem: semantic segmentation, lane: lane marking, dri: drivable area, det: object detection

The table 12 shows the performance of each STNN when trained with two new backbones. One is from an STNN and the other one is from an MTNN. The pre-trained encoder is selected such that its original task is similar to the current task. For example, for semantic segmentation training, the object detection encoder from the section 4.2.2 is selected. And the MTNN shared encoder is simply the one that handles the other three primary tasks except for the current one. The results show that each task's accuracy improves the most when MTNN's shared encoder weights are used. It also proves that the MTNNs trained in this study have learned more generalized feature representations on the A2D2 dataset. The improvement in accuracy is also observed in training with STNN backbones except for semantic segmentation.

6 Semantic Segmentation as Auxiliary Task

6.1 Concept

Multi-task learning has shown promising results in improving the performance of the tasks by better generalization. It was observed that adding some of the tasks to the combination improves the performance of the other tasks. As discussed in the section 4.3.4, task semantic segmentation favors other task's performance in most cases. Adding the semantic segmentation task improves the MTNN's overall performance. It can be considered an auxiliary task to improve the performance of other tasks. One study that is closely related to this idea is "Auxiliary Tasks in Multi-task Learning" [52]. This was the first paper to introduce this idea. In this study, they used synthetic data called synMT for the training of MTNN to improve the performance of semantic segmentation. Refer to the paper [52]. The dataset contains labels for semantic segmentation, depth estimation, time estimation, and weather classification. They demonstrated that the performance of semantic segmentation can be boosted by using other auxiliary tasks in the MTNN. However, you need labels for all the auxiliary tasks for the training of the MTNN. An overview of auxiliary tasks in MTL is described in this paper [53].

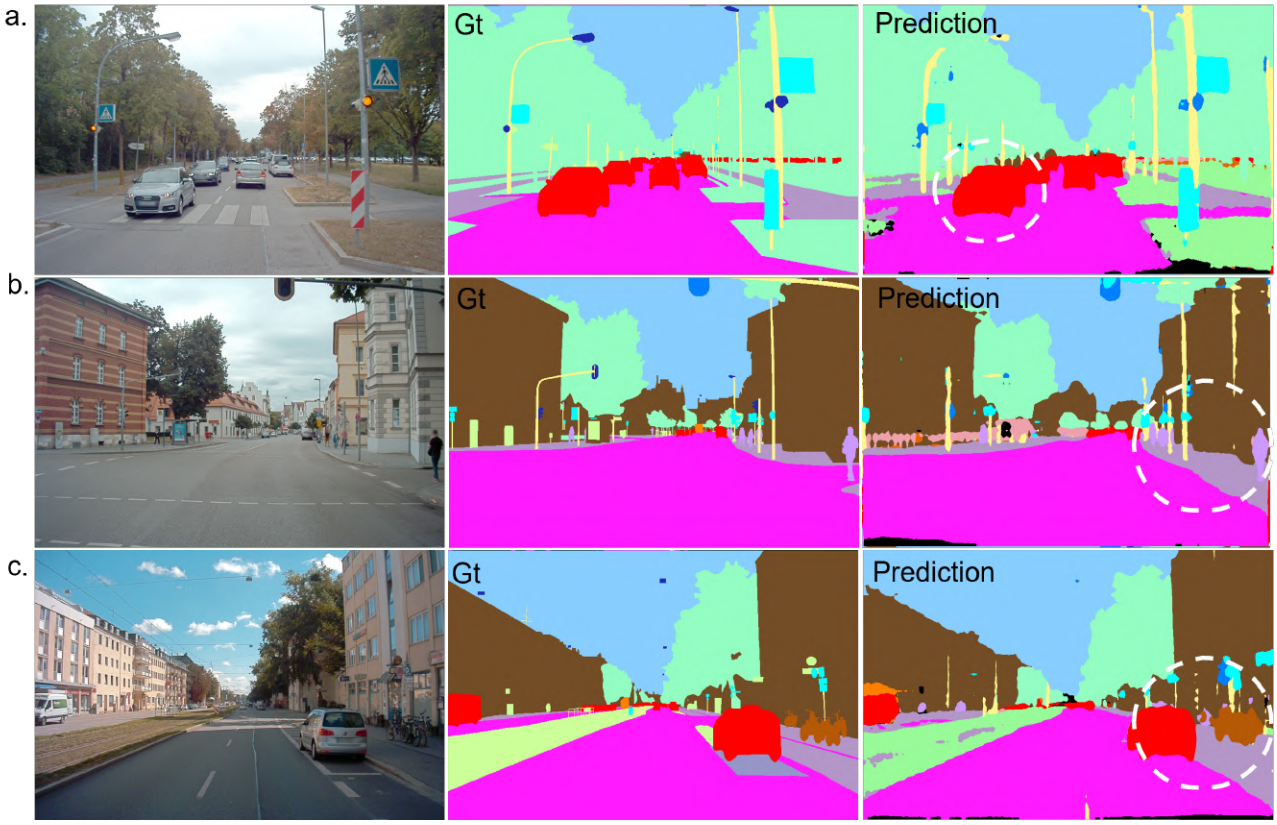

Figure 32: Examples of semantic segmentation labels generated from DeepLabv3 pre-trained on CityScapes and BDD. The white circles indicate some of the important objects detected.

In this study, this idea is extended to the situation where the dataset has labels only for one particular task and is not suitable for MTNN training, as discussed in section 3.2. The labels for the auxiliary task can be generated by using an STNN belonging to the same task as the auxiliary task, and it should be pre-trained on a similar dataset. Here, an MTNN that has semantic segmentation as an auxiliary task and another task from our primary tasks is trained. For example, an MTNN with semantic segmentation and object detection. To generate the labels for semantic segmentation, an STNN called DeepLabv3 [54] pre-trained on CityScapes [34] and BDD [41] dataset is used. The pre-trained weights are available in this repository [55]. Some

examples of generated labels for A2D2 dataset images are shown in figure 32. The Deeplabv3 is able to detect objects like cars (example a), trucks, pedestrians (example b), bicycles (example c), etc., which are necessary for object detection. It is important that the pre-trained model generates labels with good accuracy and that most of the objects are predicted.

6.2 Training Results

The semantic segmentation and object detection MTNN is trained with A2D2 training images. The labels for semantic segmentation are DeepLabv3 generated labels and original A2D2 labels for object detection. The MTNN architecture is the same as described in section 4.3.1. The training procedure and all the hyper-parameters are kept the same as in section 4.3.3. The training results are shown in the table 13 and these results are from stage 2 of the training procedure. As per the results, there is an improvement of 6.94 in object detection's mAP compared to STNN (refer to section 4.2.2). Hence, MTL can be used to improve STNN performance by adding a similar task as an auxiliary to the MTNN architecture.

Multi-task	**Semantic segmentation (auxiliary task)**	**Object detection (main task)**
semantic segmentation + object detection	56.37	41.86(+6.94)

Table 13: The table shows the result of MTNN training with semantic segmentation as a auxiliary task. Semantic segmentation score is dice coefficient and object detection score is mAP@0.5. The improvement in the object detection mAP with respect to STNN result (refer to section4.2.2) is shown in the brackets.

7 Scope of Improvement

The training of all STNNs and MTNNs is done using the pre-trained weights of resnet-34 [47]. Based on the results shown in table 9, it can be seen that the overall improvement in the performance of the MTNNs reduces as the number of tasks increases. In fact, the most improvement in the accuracy of a particular task comes from a two-task MTNN. One hypothesis is that, as the number of tasks increases, the ability of the shared encoder to accommodate feature representations required for all the tasks reduces. This could be solved by increasing the size of the shared encoder as the number of tasks increases. For example, for three-task and four-task MTNN, resnet-50 can be used as the shared encoder. However, it increases the overall size of the MTNN and may require longer training time.

8 Conclusion

In this study, multiple MTNNs were implemented and compared with their respective single task performances. The loss weighing technique and the training procedure developed for multi-task training significantly improve their performance. Based on the results, it has been successfully proved that MTNNs outperform STNNs both in terms of speed and accuracy. Furthermore, it has been demonstrated that using the transfer learning method, the MTNN shared encoder can increase STNN's performance. Based on the performance results of all MTNNs, it is clear that adding semantic segmentation helps to improve the performance of other tasks in an MTL scenario. Further, this was demonstrated by improved object detection performance, by training an MTNN with semantic segmentation as an auxiliary task and object detection as the main task.

References

[1] Jakob Geyer, Yohannes Kassahun, Mentar Mahmudi, Xavier Ricou, Rupesh Durgesh, Andrew S. Chung, Lorenz Hauswald, Viet Hoang Pham, Maximilian Mühlegg, Sebastian Dorn, Tiffany Fernandez, Martin Jänicke, Sudesh Mirashi, Chiragkumar Savani, Martin Sturm, Oleksandr Vorobiov, Martin Oelker, Sebastian Garreis, and Peter Schuberth. A2D2: Audi Autonomous Driving Dataset. 2020.

[2] O. Ronneberger, P.Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 9351 of *LNCS*, pages 234–241. Springer, 2015. (available on arXiv:1505.04597 [cs.CV]).

[3] Image of dice coefficient and iou. `https://towardsdatascience.com/metrics-to-evaluate-your-semantic-segmentation-model-6bcb99639aa2`.

[4] Image of yolov3 network architecture. `https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b`.

[5] Yolov3 comparison graph. `https://pjreddie.com/darknet/yolo/`.

[6] Sorin Grigorescu, Bogdan Trasnea, Tiberiu Cocias, and Gigel Macesanu. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 37(3):362–386, 2020.

[7] Varun Ravi Kumar, Senthil Yogamani, Hazem Rashed, Ganesh Sitsu, Christian Witt, Isabelle Leang, Stefan Milz, and Patrick Mäder. Omnidet: Surround view cameras based multi-task visual perception network for autonomous driving. *IEEE Robotics and Automation Letters*, 6(2):2830–2837, 2021.

[8] Marvin Teichmann, Michael Weber, Marius Zoellner, Roberto Cipolla, and Raquel Urtasun. Multinet: Real-time joint semantic reasoning for autonomous driving. In *2018 IEEE intelligent vehicles symposium (IV)*, pages 1013–1020. IEEE, 2018.

[9] Richard Bormann, Florian Jordan, Wenzhe Li, Joshua Hampp, and Martin Hägele. Room segmentation: Survey, implementation, and analysis. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 1019–1026. IEEE, 2016.

[10] Qinghui Zhang, Xianing Chang, and Shanfeng Bian Bian. Vehicle-damage-detection segmentation algorithm based on improved mask rcnn. *IEEE Access*, 8:6997–7004, 2020.

[11] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[12] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[13] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.

[14] Shruti Jadon. A survey of loss functions for semantic segmentation. In *2020 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*, pages 1–7. IEEE, 2020.

[15] Alberto Garcia-Garcia, Sergio Orts-Escolano, Sergiu Oprea, Victor Villena-Martinez, Pablo Martinez-Gonzalez, and Jose Garcia-Rodriguez. A survey on deep learning techniques for image and video semantic segmentation. *Applied Soft Computing*, 70:41–65, 2018.

[16] Agung W Setiawan. Image segmentation metrics in skin lesion: accuracy, sensitivity, specificity, dice coefficient, jaccard index, and matthews correlation coefficient. In *2020 International Conference on Computer Engineering, Network, and Intelligent Multimedia (CENIM)*, pages 97–102. IEEE, 2020.

[17] Joseph Redmon, 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*, pages 779–788, 2016.

[18] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[19] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[20] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[21] Yuanyuan Wang, Chao Wang, Hong Zhang, Yingbo Dong, and Sisi Wei. Automatic ship detection based on retinanet using multi-resolution gaofen-3 imagery. *Remote Sensing*, 11(5):531, 2019.

[22] Upesh Nepal and Hossein Eslamiat. Comparing yolov3, yolov4 and yolov5 for autonomous landing spot detection in faulty uavs. *Sensors*, 22(2), 2022.

[23] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.

[24] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[25] Sebastian Ruder. An overview of multi-task learning in deep neural networks. *ArXiv*, abs/1706.05098, 2017.

[26] Zhanpeng Zhang, Ping Luo, Chen Change Loy, and Xiaoou Tang. Facial landmark detection by deep multi-task learning. In *European conference on computer vision*, pages 94–108. Springer, 2014.

[27] Jifeng Dai, Kaiming He, and Jian Sun. Instance-aware semantic segmentation via multi-task network cascades. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3150–3158, 2016.

[28] Shikun Liu, Edward Johns, and Andrew J Davison. End-to-end multi-task learning with attention. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1871–1880, 2019.

[29] Trevor Standley, Amir Zamir, Dawn Chen, Leonidas Guibas, Jitendra Malik, and Silvio Savarese. Which tasks should be learned together in multi-task learning? In *International Conference on Machine Learning*, pages 9120–9132. PMLR, 2020.

[30] Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. Cross-stitch networks for multi-task learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3994–4003, 2016.

[31] Sebastian Ruder, Joachim Bingel, Isabelle Augenstein, and Anders Søgaard. Latent multi-task architecture learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4822–4829, 2019.

[32] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3354–3361. IEEE, 2012.

[33] Senthil Yogamani, Ciarán Hughes, Jonathan Horgan, Ganesh Sistu, Padraig Varley, Derek O'Dea, Michal Uricár, Stefan Milz, Martin Simon, Karl Amende, et al. Woodscape: A multi-task, multi-camera fisheye dataset for autonomous driving. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9308–9318, 2019.

[34] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016.

[35] Vladimir Nekrasov, 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)*, pages 7101–7107. IEEE, 2019.

[36] Ganesh Sistu, Isabelle Leang, Sumanth Chennupati, Senthil Yogamani, Ciarán Hughes, Stefan Milz, and Samir Rawashdeh. Neurall: Towards a unified visual perception model for automated driving. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 796–803. IEEE, 2019.

[37] Di Feng, Christian Haase-Schütz, Lars Rosenbaum, Heinz Hertlein, Claudius Glaeser, Fabian Timm, Werner Wiesbeck, and Klaus Dietmayer. Deep multi-modal object detection and semantic segmentation for autonomous driving: Datasets, methods, and challenges. *IEEE Transactions on Intelligent Transportation Systems*, 22(3):1341–1360, 2020.

[38] Sauhaarda Chowdhuri, Tushar Pankaj, and Karl Zipser. Multinet: Multi-modal multi-task learning for autonomous driving. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1496–1504. IEEE, 2019.

[39] Keishi Ishihara, Anssi Kanervisto, Jun Miura, and Ville Hautamaki. Multi-task learning with attention for end-to-end autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2902–2911, 2021.

[40] Pullarao Maddu, Wayne Doherty, Ganesh Sistu, Isabelle Leang, Michal Uřičář, Sumanth Chennupati, Hazem Rashed, Jonathan Horgan, Ciarán Eising, and Senthil Yogamani. Fisheyemultinet: Real-time multi-task learning architecture for surround-view automated parking system. 12 2019.

[41] Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan, and Trevor Darrell. Bdd100k: A diverse driving dataset for heterogeneous multitask learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2636–2645, 2020.

[42] Yiyi Liao, Jun Xie, and Andreas Geiger. Kitti-360: A novel dataset and benchmarks for urban scene understanding in 2d and 3d. *IEEE transactions on pattern analysis and machine intelligence*, PP, 2022.

[43] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11618–11628, 2020.

[44] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[45] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[46] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2015.

[47] Kaiming He, 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*, pages 770–778, 2016.

[48] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[49] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[50] Alexander Buslaev, Vladimir I. Iglovikov, Eugene Khvedchenya, Alex Parinov, Mikhail Druzhinin, and Alexandr A. Kalinin. Albumentations: Fast and flexible image augmentations. *Information*, 11(2), 2020.

[51] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6517–6525, 2017.

[52] Partoo Vafaeikia, Khashayar Namdar, and Farzad Khalvati. A brief review of deep multi-task learning and auxiliary task learning. 2020.

[53] Partoo Vafaeikia, Khashayar Namdar, and Farzad Khalvati. A brief review of deep multi-task learning and auxiliary task learning. *ArXiv*, abs/2007.01126, 2020.

[54] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *ArXiv*, abs/1706.05587, 2017.

[55] Pytorch implementation of deeplabv3, trained on the cityscapes dataset. `https://https://github.com/fregu856/deeplabv3`.

9 Appendix

Comparison of Multi-Task vs Single-Task Class Accuracies

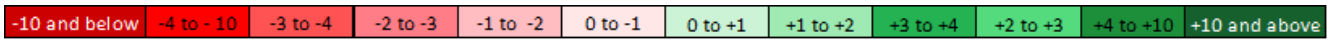

Figure 33: Color code specifying range of deviation for accuracies mentioned in figures 34, 35, 36, and 37.

Semantic segmentation									
classes		Single	Multi-Task Neural Networks						
class ID	class names	Task Network	sem+lane	sem+dri	sem+det	sem+dri +det	sem+lane +det	sem+lane +det	sem+lane +dri+det
0	Car	89.34	89.6	89.16	88.34	87.54	88.53	89.2	88.59
1	bicycle	53.52	59.68	54.64	52.5	56.24	54.34	57.87	57.1
2	pedestrian	46.34	54.51	53.16	51.76	51.37	51.64	52.8	52.24
3	Utility vehicle	48.77	47.6	42.35	45	41.14	42.67	47.56	40.37
4	small vehicles	79.51	81.99	81.63	81.5	82.66	81.84	83.17	83.61
5	Traffic signal	72.38	74.13	70.98	68.34	69.34	71.09	73.92	68.37
6	Traffic sign	58.08	61.52	59.62	56.27	53.17	54.09	57.79	55.26
7	sidebars	86.39	86.83	87.31	86.26	84.44	81.67	85.7	84.39
8	road	96.64	97	97.07	96.16	96.38	96.34	97.05	96.47
9	road blocks	84.08	73.56	72.72	62.34	63.66	53.68	71.37	59.85
10	poles	43.30	46.46	45.29	40.82	38.48	38.78	44.57	39.33
11	animals	96.33	96.33	96.33	96.33	96.33	96.33	96.33	96.33
12	grid structure	38.01	35.37	31.76	26.23	24.99	23.07	32.68	24.53
13	signal corpus	45.15	49.4	46.91	45.14	41.85	43.35	47.92	43.59
14	nature object	90.19	90.15	89.76	88.9	88.59	88.56	90.09	88.85
15	parking area	58.78	57.52	56.88	54.62	54.63	51.85	60.06	55.32
16	sidewalk	65.40	66.91	68.59	62.63	64.57	63.26	68.15	63.52
17	traffic guide obj.	69.61	61.85	65.05	52.98	57.8	55.4	68.2	53.35
18	sky	97.23	97.16	97.45	97.16	97.24	97.02	97.41	97.12
19	buildings	77.23	78.46	78.05	75.8	74.02	74.75	76.88	74.02
20	background	42.19	44.5	43.66	37.79	36.78	38.84	42.76	37.68
Average Accuracy		68.50	69.07	68.02	65.09	64.82	64.15	68.64	64.76

Figure 34: Comparison of MTNNs and STNN class accuracies for semantic segmentation. Each value is dice coefficient.

Lane marking									
classes		Single	Multi-Task Neural Networks						
class ID	class names	Task Network	sem+lane	lane+dri	lane+det	sem+lane +dri	sem+lane +det	lane+dri +det	sem+lane +dri+det
0	Solid line	56.28	58.95	61.64	52.78	62.47	56.08	58.46	59.84
1	Zebra crossing	98.46	97.9	98.46	98.45	98.41	98.46	98.46	97.68
2	Painted driv. instr.	70.46	72.05	74.73	68.24	69.09	65.38	66.4	68.93
3	Dashed line	72.01	75.87	75.09	70.04	75.56	74.28	72.05	74.77
4	background	99.81	99.83	99.82	99.8	99.83	99.81	99.81	99.82
Average Accuracy		79.404	80.92	81.948	77.862	81.072	78.802	79.036	80.208

Figure 35: Comparison of MTNNs and STNN class accuracies for lane marking. Each value is dice coefficient.

Drivable area									
classes		Single	Multi-Task Neural Networks						
class ID	class names	Task Network	dri+det	lane+dri	sem+dri	lane+dri +det	sem+lane +dri	sem+dri +det	sem+lane +dri+det
0	non_drivable	73.73	73.09	75.57	76.91	71.68	76.25	73.27	73.54
1	drivable	96.65	96.6	97.16	97.08	96.39	97.16	96.55	96.71
2	background	99.12	99.11	99.22	99.23	99.08	99.19	99.15	99.16
Average Accuracy		89.83	89.60	90.65	91.07	89.05	90.87	89.66	89.80

Figure 36: Comparison of MTNNs and STNN class accuracies for drivable area. Each value is dice coefficient.

Object detection									
classes		Single	Multi-Task Neural Networks						
class ID	class names	Task Network	sem+det	dri+det	lane+det	lane+dri +det	sem+dri +det	sem+lane +det	sem+lane +dri+det
0	Car	81.43	88.19	84.94	85.73	85.93	86.28	85.43	86.26
1	Pedestrian	23.32	35.4	29	24.94	25.25	30.13	29.75	30.68
2	Utility vehicle	35.09	48.1	42.74	41.57	41.27	46.45	41.21	44.39
3	Cyclist	17.96	15.82	26.88	14.06	29.46	31.03	31.68	17.9
4	Bus	13.67	22.89	22.96	7.1	17.46	23	20.89	17.85
5	MotorBiker	30	53.79	28.38	19.32	29.89	22.49	26.85	24.01
6	Bicycle	55.8	76.04	54.97	67.81	75.13	64.47	75.16	60.39
7	Motorcycle	22.06	33.57	26.56	24.3	16.48	29.22	22.12	23.64
Average Accuracy		34.92	46.73	39.55	35.60	40.11	41.63	41.64	38.14

Figure 37: Comparison of MTNNs and STNN class accuracies for object detection. Each value is mAP@0.5.

Training Graphs

Below are some of the training loss and accuracy graphs. Some of the values in the accuracy graphs do not start from the zeroth epoch because the accuracy was measured only after a few epochs of training to save time. Also, some of the object detection accuracy graphs are measured every fifth epoch at the beginning of the training.

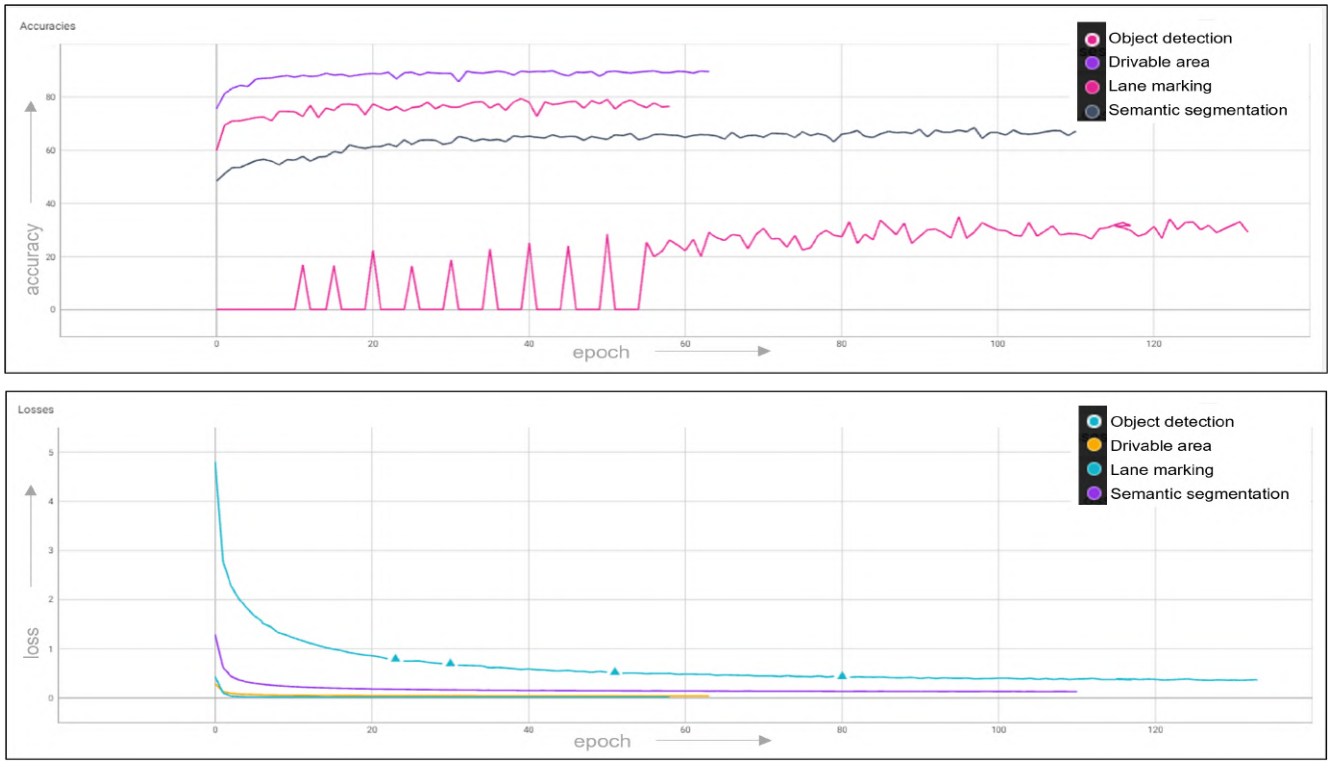

Figure 38: Training loss and accuracy graphs of all STNNs. Each Value in the accuracies graph is dice coefficient for semantic segmentation, lane marking and drivable area and mAP@0.5 for object detection.

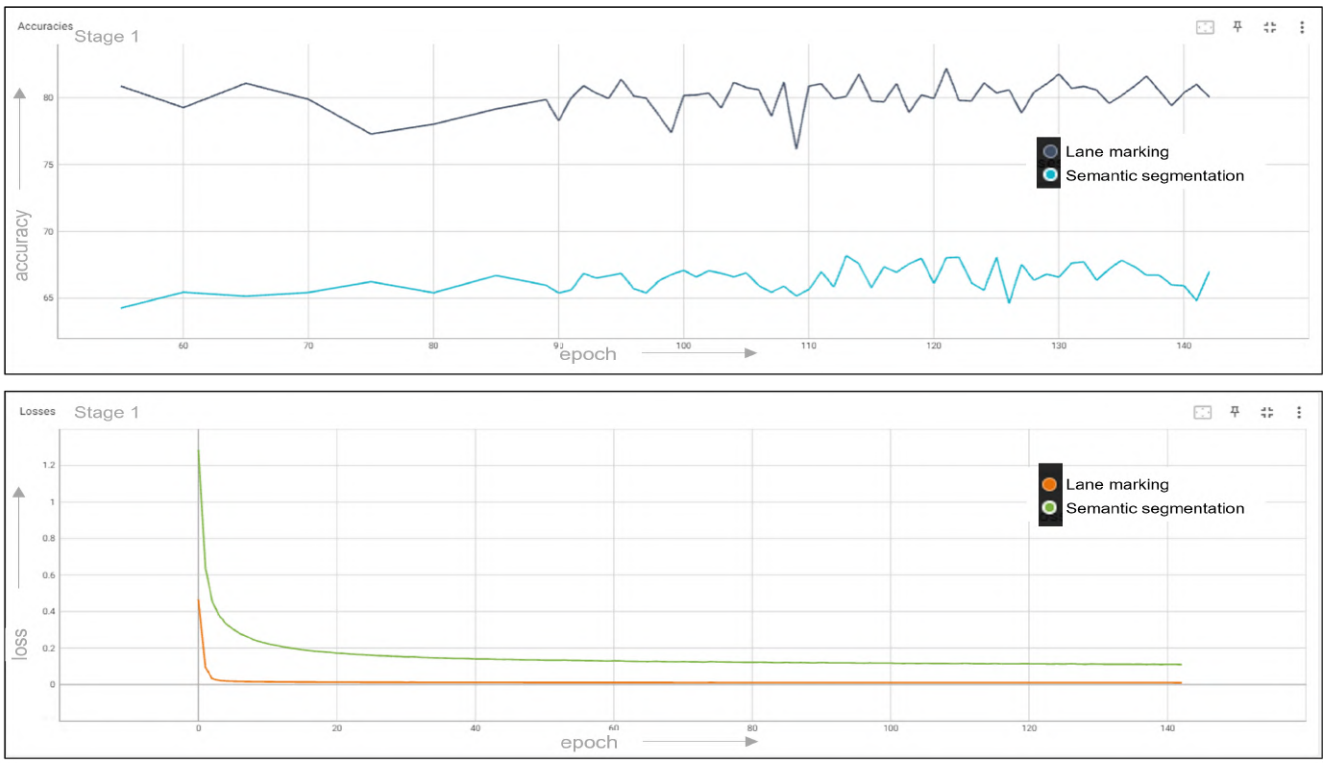

Figure 39: Stage1 training loss and accuracy graphs of semantic segmentation and lane marking MTNN. values in the accuracies graph are dice coefficient.

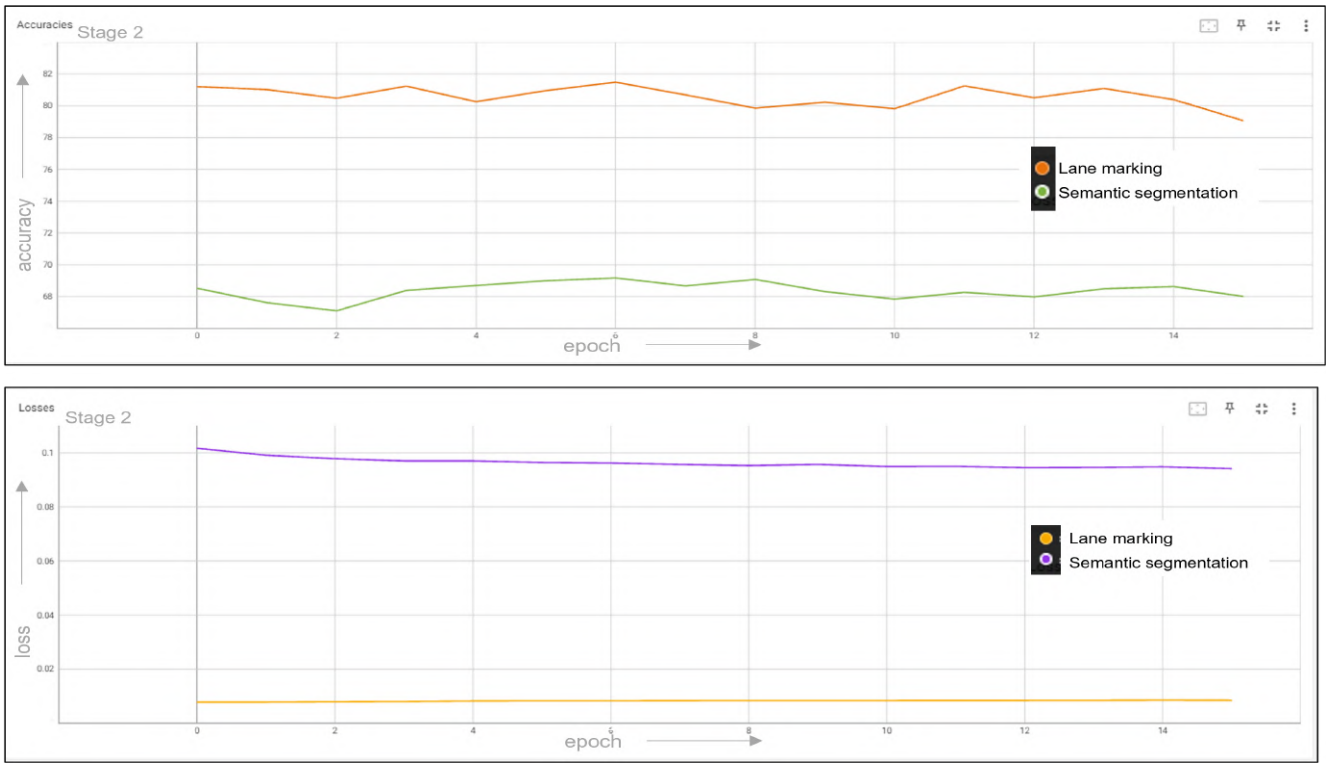

Figure 40: Stage2 training loss and accuracy graphs of semantic segmentation and lane marking MTNN. values in the accuracies graph are dice coefficient.

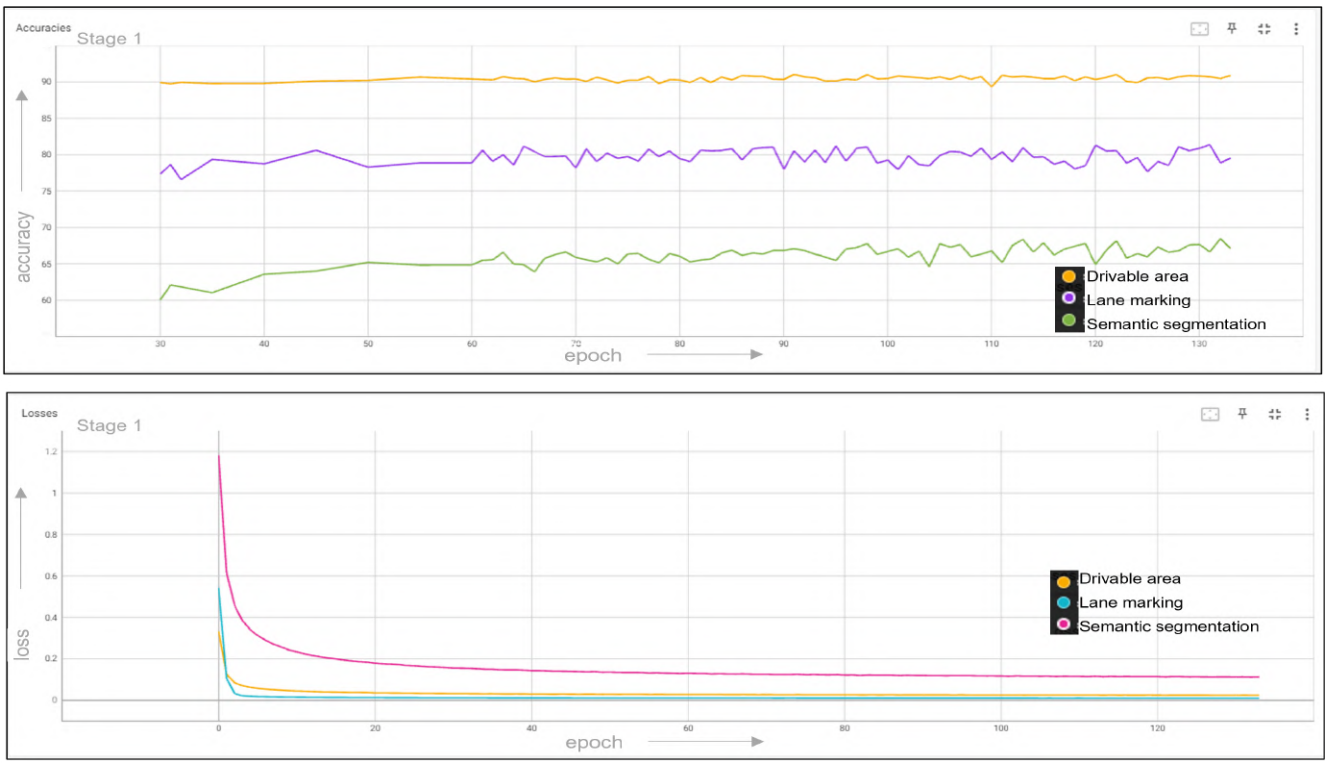

Figure 41: Stage1 training loss and accuracy graphs of semantic segmentation, lane marking and drivable area MTNN. values in the accuracies graph are dice coefficient.

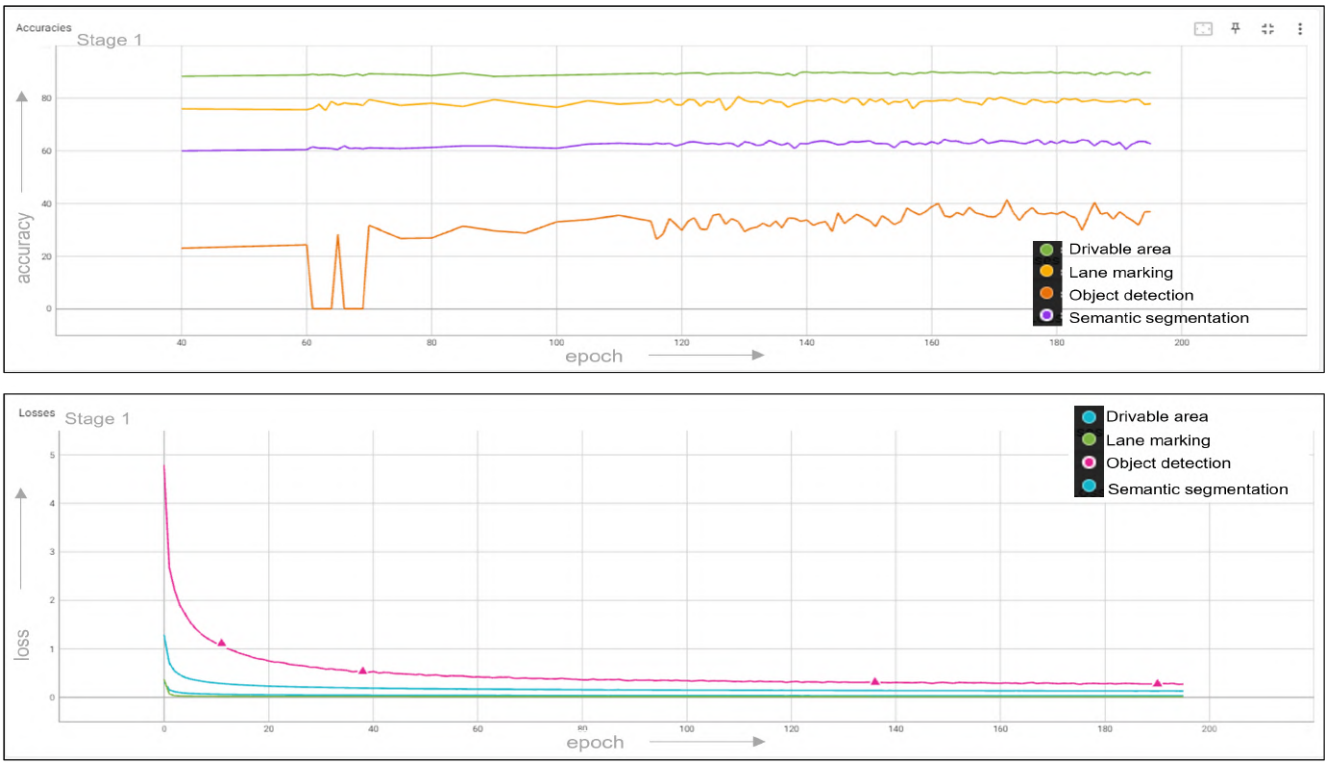

Figure 42: Stage1 training loss and accuracy graphs of semantic segmentation, lane marking, drivable area and object detection MTNN. Each Value in the accuracies graph is dice coefficient for semantic segmentation, lane marking and drivable area and mAP@0.5 for object detection.

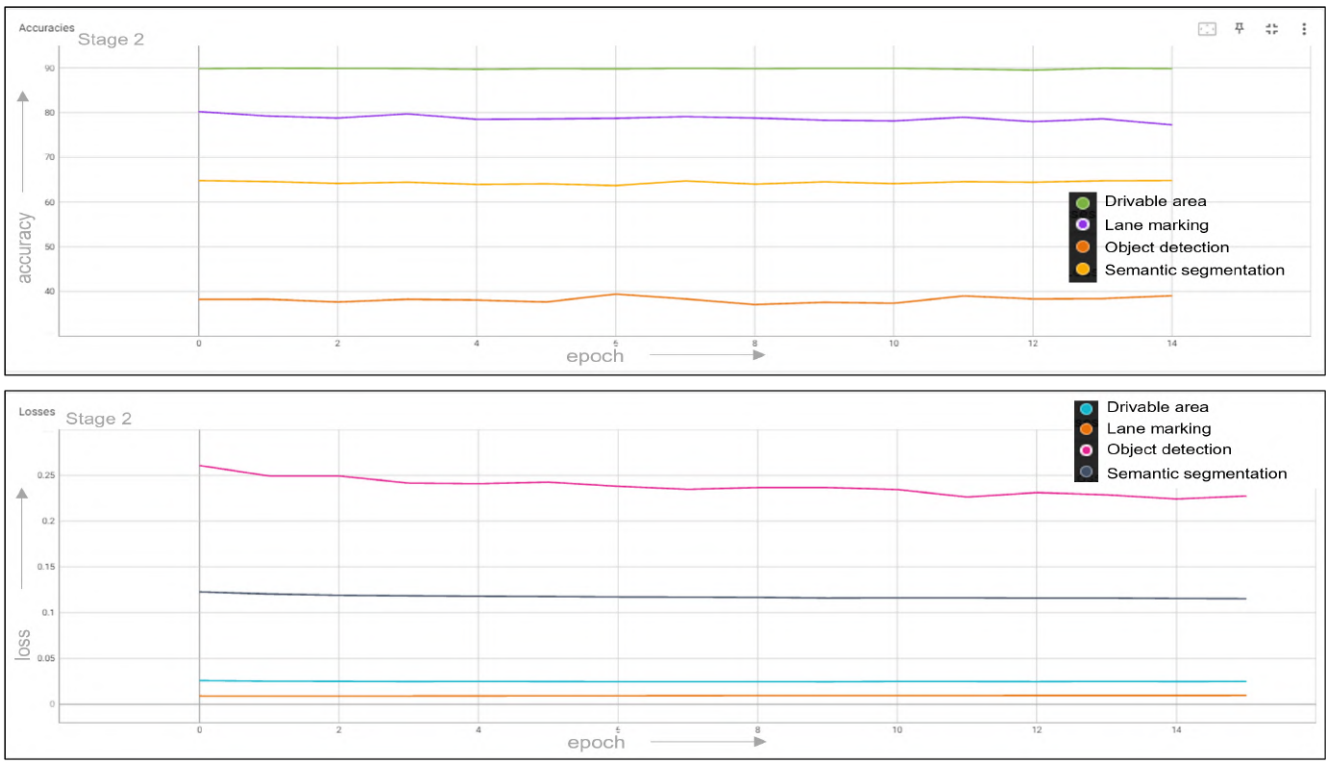

Figure 43: Stage2 training loss and accuracy graphs of semantic segmentation, lane marking, drivable area and object detection MTNN. Each Value in the accuracies graph is dice coefficient for semantic segmentation, lane marking and drivable area and mAP@0.5 for object detection.