Scalable Multi-Task Learning R-CNN for Classification and Localization in Autonomous Vehicle Technology

Sonam Rinchen

A thesis submitted in partial fulfillment of the requirements for the
Master of Applied Science in Electrical and Computer Engineering

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

Multi-task learning (MTL) is a rapidly growing field in the world of autonomous vehicles, particularly in the area of computer vision. Autonomous vehicles are heavily reliant on computer vision technology for tasks such as object detection, object segmentation, and object tracking. The complexity of sensor data and the multiple tasks involved in autonomous driving can make it challenging to design effective systems. MTL addresses these challenges by training a single model to perform multiple tasks simultaneously, utilizing shared representations to learn common concepts between a group of related tasks, and improving data efficiency.

In this thesis, we proposed a scalable MTL system for object detection that can be used to construct any MTL network with different scales and shapes. The proposed system is an extension to the state-of-art algorithm called Mask RCNN. It is designed to overcome the limitations of learning multiple objects in multi-label learning. To demonstrate the effectiveness of the proposed system, we built three different networks using it and evaluated their performance on the state-of-the-art BDD100k dataset. Our experimental results demonstrate that the proposed MTL networks outperform a base single-task network, Mask RCNN, in terms of mean average precision at 50 (mAP50). Specifically, the proposed MTL networks achieved a mAP50 of 66%, while the base network only achieved 53%. Furthermore, we also conducted comparisons between the proposed MTL networks to determine the most efficient way to group tasks together in order to create an optimal MTL network for object detection on the BDD100k dataset.

Acknowledgements

I would like to express my sincere appreciation and gratitude to my supervisor, Professor Hussein Mouftah, for his invaluable guidance, mentorship, and support throughout the course of this research. His expertise and experience in the field of autonomous vehicles and computer vision have been instrumental in shaping the research problem and guiding me in the right direction. I am particularly grateful for his patience and unwavering support during times of my illness, including the challenging period of the COVID-19 outbreak.

I would also like to extend my thanks to Dr. Binod Vaidya for his valuable insights and guidance throughout the research. His constructive feedback and comments have been extremely helpful in shaping the research and improving the quality of my work. It has been a pleasure to work with him in the SecCharge lab and I am grateful for the opportunity to have learned from him.

Finally, I would like to acknowledge the significant contributions of the other members of the SecCharge lab, who have provided me with valuable feedback and support throughout the research. I am grateful for the opportunity to have been a part of this research community and I am confident that the experiences and knowledge gained during this research will be invaluable in my future endeavors.

Contents

List of Acronyms

List of Figures

List of Tables

CHAPTER 1

Introduction

There is abundant proof throughout history to suggest that the concept of a car that can operate without a human driver has been floating about for quite some time. In the Middle Ages, hundreds of years before the first automobile was ever built, people were imagining what it would be like to have a vehicle that could operate itself without human intervention. The proof for this comes from a sketch that was made by Leonardo da Vinci, and it was intended to be a basic plan for a self-propelled cart. What he had in mind at the time, which involved using coiled-up springs as a means of propulsion, was quite straightforward in comparison to the more complicated navigation systems that are currently being developed. Even though the concept of autonomous vehicles was invented throughout the Middle Ages, the first vehicle to ever have some level of autonomy was not observed until 1925. It was Francis Houdina who came up with the idea of a car that could drive itself for the first time. The vehicle could be driven via radio control, and in addition to blowing its horn, it could also start the engine, shift gears, and change the ignition. It was proven by Francis when he drove his radio-controlled car around the streets of Manhattan with no one else in control of the vehicle.

Since then, due to advancements in various technology, the concept of driverless cars is no longer a far-fetched notion. In particular, the advancement in artificial intelligence has been a significant factor in moving this concept from the realm of fantasy to that of possible future use. Large corporations such as Google, Uber, Tesla, and other major automotive manufacturers have already developed vehicles with at least some level of autonomy. These vehicles have features such as lateral control, adaptive cruise control, lane detecting, self-parking, and semi-autonomous driving capabilities. Companies such as Tesla, Google, Waymo, and ZOOX are making significant progress toward developing completely autonomous vehicles.

It is reasonable to assume that these businesses have already accomplished the requirements for level 3 autonomy. The advancement of artificial intelligence has brought us substantially closer to self-driving automobiles. Particularly, the progress that has been made in deep learning and computer vision has brought us one step closer to having cars that can drive themselves. Thus, the thesis is initially motivated by a desire to enhance a computer vision task for autonomous vehicle systems.

1.1 Motivations and Objectives

The field of autonomous driving has gained tremendous attention in recent years, with significant investments being made by both industry and academia. A self-driving vehicle is a complex system that is comprised of multiple subsystems, including perception, localization, path planning, decision-making, and vehicle control. Among these subsystems, perception comprises computer vision tasks that are critical to the successful operation of self-driving vehicles.

A self-driving vehicle's vision system must process all objects on the road in the same manner as a human vision system. Consequently, a self-driving vehicle must utilize a multitasking system in order to process all the objects on the road. Multitask learning is one of the most essential aspects of a self-driving vehicle, as it allows a single model to perform multiple tasks simultaneously.

The importance of multitasking learning in the field of autonomous driving is supported by Dr. Andrej Karpathy, Senior Director of AI at Tesla. He elevated the importance of multi-task learning in his June 2019 presentation at the International Conference on Machine Learning [7]: "I think multi-task learning is one of the most important and understudied subfields of machine learning. Most real-world problems are multi-task."

While the concept of multitasking is crucial, there are still many challenges left to investigate in the field of multi-task learning. Therefore, the motivation for this thesis is inspired by the need for research in the field of multi-task learning in computer vision for self-driving cars. The primary objective of this thesis is to build a scalable multi-task learning system for object detection in autonomous vehicle technology.

This research will explore the potential benefits of the proposed scalable multi-task learning system, including the ability to leverage shared knowledge and reduce the need for additional data or computation over its state-of-the-art single-task counterparts. Additionally, the research will examine how multiple objects can be grouped together in the proposed multi-task system and can lead to more generalizable and accurate representations. The goal is to develop a more effective and efficient multi-task network using the proposed system, ultimately leading to more accurate object detection over single-task networks in multiple object detection tasks.

In conclusion, this research will investigate the potential benefits of the proposed multi-task systems

over single-task systems in self-driving vehicle technology, with the goal of improving the performance and efficiency of multi-task network models. By leveraging and grouping related tasks or objects, the research aims to develop a more effective and efficient multitask network using the proposed system that can lead to more accurate object detection.

1.2 Thesis Contributions

The primary contribution of this thesis is the development of a novel, a scalable multi-task learning platform for computer vision tasks, with a specific emphasis on object detection in autonomous vehicles. The proposed system offers several significant contributions that advance the field of multiple object detection systems.

First, the system addresses a common problem in multi-task learning, the issue of negative learning, where the performance of a model on one task is negatively impacted by the learning of another task. This phenomenon is particularly problematic in computer vision tasks that require the detection of multiple objects simultaneously. To overcome this challenge, the proposed system introduces three distinct grouping methods based on object sample size and visual features to group similar objects. These grouping methods enable the model to focus on related tasks and avoid negative learning, thus improving the overall performance of the system.

Second, the proposed system addresses a well-known issue of imbalance labels, usually seen in single-task networks for multi-label object detection where some labels are much more prevalent than others, leading to poor performance for minority labels and making accurate predictions challenging. To address this issue, the proposed system utilized a multi-task network that represents each object or group of objects with a separate output layer. This enables the network to learn to recognize and classify minority classes more accurately, improving the overall performance of the model on imbalanced datasets.

Third, the proposed platform offers a significant benefit in its scalability property, allowing for the construction of multiple multi-task networks of desired size and facilitating a comprehensive analysis of their performance. This property is particularly advantageous in real-world scenarios, where computer vision systems need to process large amounts of data and must be adaptable to different contexts.

The proposed system also offers several benefits for object detection in autonomous vehicles. It outperforms single-task systems in multiple object detection tasks, providing a more accurate and reliable system. The proposed grouping methods and multi-task network can also be extended to other computer vision tasks, making them applicable to a wide range of applications beyond autonomous vehicles.

In conclusion, this thesis offers a significant contribution to the development of more effective and reliable computer vision systems for object detection in autonomous vehicles and beyond. The proposed system's scalability property, coupled with its ability to address the challenges of negative learning and

imbalanced labels, offers promising prospects for future research and applications in the field of computer vision.

1.3 Thesis Outlines

The thesis is structured into five chapters, each with a specific focus. Chapter 1 of this thesis provides an introduction to the research topic, highlighting the motivation behind the study. The chapter begins with a discussion on the importance of multi-task learning (MTL) in machine learning, followed by a presentation of the contributions of the thesis to the field. The chapter concludes with an outline of the thesis.

Chapter 2 presents an overview of related work on MTL in computer vision. The chapter starts with an introduction to MTL, followed by an explanation of what a task is. The chapter then explores two methods in MTL, namely parameter sharing and hard parameter sharing. The concept of data fidelity in MTL is also discussed in the chapter. Additionally, the chapter reviews recent work on MTL in computer vision, with a focus on driving perception.

Chapter 3 describes the system architecture design and implementation of a scalable multi-task R-CNN for object detection. The chapter starts with an introduction to the system, followed by an explanation of the base system and its architecture. The chapter then introduces the proposed scalable multi-task system and discusses its implementation, network parameters, model parameters assignment, loss function, separator functions for inputs and outputs separation, and configuration file for the system.

Chapter 4 presents the experiments conducted to evaluate the proposed scalable multi-task system for object detection. The chapter explores the base network and the proposed networks based on object visual feature similarities, objects falling within the same sample sizes range, and both principles at once as a hybrid branch. The chapter presents the model training and results for each network type, followed by a results analysis that examines the effect of the learning rate, momentum, weight decay, and comparison to related work.

Finally, Chapter 5 offers concluding remarks on the proposed system, summarizing the contributions of this thesis and the implications of the results. This chapter also outlines potential future directions for research, building on the proposed system and its features.

CHAPTER 2

Related Work on Multi-Task Learning

2.1 Introduction: Multi-Task Learning

Multi-task learning (MTL) [8] is a rapidly evolving field of research that aims to capitalize on the synergies between different tasks to reduce the amount of data or computational resources required for multiple-task learning. It is thought to closely mimic the human learning process, as the human mind relies on the ability to synthesize information from a variety of domains. The central idea of MTL is to train a single model to perform multiple tasks simultaneously. This is achieved by utilizing shared representations that learn common concepts across a group of related tasks. These shared representations are designed to improve data efficiency and accelerate the learning process for related or subsequent tasks, addressing some of the known challenges of deep learning, such as the need for large amounts of data and high computational power.

Deep MTL models have been shown to outperform their single-task counterparts in computer vision tasks. MTL is generally preferred over single-task learning because it utilizes more data from a variety of learning tasks, enabling the training of more robust and universal representations for various tasks. This leads to increased information sharing between tasks, improved performance in each task, and a reduced risk of overfitting each task. On top of that, MTL networks have a smaller memory footprint because they share layers organically, and they have shorter inference times because they only need to calculate the features in the common layers once, rather than once for each task. It can also improve overall performance if the associated tasks complement each other or function as a regularizer for one another.

There are many potential applications for MTL, and the tasks that can be addressed are diverse. Some examples of tasks that may be addressed using MTL include classification, regression, translation, segmentation, clustering, ranking, and detection tasks. In the context of object detection, MTL can be used for tasks such as object detection, object segmentation, object tracking, object pose estimation, and object depth estimation. Our thesis focuses on the object detection perspective of computer vision tasks involves in autonomous vehicle technology. Object detection basically comprises the classification of objects and localization of objects by refining their bounding box.

In conclusion, MTL is a powerful technique that can improve the performance and efficiency of machine learning models by allowing them to learn multiple tasks simultaneously. It addresses the problem of data sparsity, allows for the reuse of previously acquired knowledge, and reduces the time spent on manual labeling. MTL has a wide range of potential applications, and its effectiveness has been demonstrated in several computer vision tasks. It is a promising area of research that has the potential to improve the performance and efficiency of machine learning models in a variety of applications.

2.2 Definition of MTL

One of the earliest papers published about MTL is written by Rich Caruana. In his paper[8], he defined it as "an inductive transfer mechanism whose principle goal is to improve generalization performance. MTL improves generalization by leveraging the domain-specific information contained in the training signals of related tasks."

Whereas, According to Zhang et al. in [9], the definition of MTL is given as "Given m learning tasks $\{T_i\}_{i=1}^{m}$ where all the tasks or a subset of them are related but not identical, multi-task learning aims to help improve the learning of a model for T_i by using the knowledge contained in the m tasks." As for the mathematical formulation of MTL, Thung et al. explain in [1] that The typical formulation for a conventional MTL algorithm [10, 8, 9] is given in the following form:

$$\min_{w=[w^1 w^2 ... W^M]} \sum_{m=1}^{M} L(X^m, y^m, w^m) + \lambda Reg(W) \tag{2.1}$$

where $X^m \in \mathbb{R}^{N_m \times D}$ denotes the input matrix of the m-th task, $y^m \in \mathbb{R}^{N_m \times 1}$ denotes the corresponding m-th task output vector, and $w^m \in \mathbb{R}^{D \times 1}$ denotes the weight vector (or regression parameters) for the m-th task that maps X^m to y^m, e.g., $y^m \approx X^m w^m$ (for regression problem). The scalars N_m, D, and Mdenote the number of samples for the m-th task, the number of features for each input matrix, and the number of tasks, respectively. Thung et al. further explained that at this stage, they assume that all the input matrices in $X^m, m = 1, 2, , M$ are having the same dimensionality of features (but can have a different number of samples for each task), and the features of all the tasks are corresponding, so that they can concatenate all the weight vectors in w^m together to obtain $W = [w^1 w^2 w^M]$(i.e., features in

each row of W are corresponding). Based on the prior knowledge of the data and different assumptions on the relationship among tasks, it can design different constraints for W, which is normally implemented in the regularizer of W, denoted by $Reg(W)$. In addition, λ is the regularization parameter that controls the balance between the loss function (first term) and the regularizer (second term) in equation 2.1.

2.3 What is a Task?

As stated before, multi-task learning is a machine learning approach in which a single model is trained to perform multiple tasks simultaneously. These tasks can vary in different ways, such as the type of data being processed (e.g., images, text), the objective function (e.g., classification, regression), or the characteristics of the data (e.g., lighting conditions, vocabulary). One of the main benefits of multi-task learning is that it allows a model to learn more effectively by leveraging the shared information between tasks. This can be especially useful when the tasks are related or have some overlap, as the model can use the shared knowledge to improve its performance on each individual task.

In multi-task learning, tasks are typically defined in terms of the learning objective or goal that the model is trained to achieve. This can involve a variety of different types of learning, such as classification, regression, or generation, depending on the nature of the task.

For example, in the context of computer vision and image processing, tasks might include image classification (predicting the class label for an image), object detection (identifying and locating objects in an image), scene segmentation (predicting the class label for each pixel in an image), or other types of image analysis. In Natural Language Processing (NLP), tasks might include language translation, sentiment analysis, or language modeling. In other fields, tasks might involve predicting numerical values (regression)or generating output based on input data (generation).

Different tasks can vary in different ways. For example, tasks might involve different objects (e.g., cars, pedestrians, buildings), different people (e.g., different age groups or genders), or different objective functions (e.g., different loss functions in the case of multi-objective optimization). Tasks might also correspond to different lighting conditions, different works, or different languages.

Overall, tasks in multi-task learning are defined as specific goals or objectives that the model is trained to achieve. These tasks can exhibit a wide range of variations, characterized by a multitude of factors such as the nature of the data under consideration, the objective function of the analysis, or the specific features of the data itself. Essentially, multi-task learning can refer to different tasks in the traditional sense of the word, as well as different domains or objectives that correspond to different machine learning problems. According to Zhang [9], there are various approaches to designing and implementing multi-task learning systems, depending on the specific goals and objectives of the system, as well as the characteristics of the training data.

In this thesis, the representation of a task is defined as an object or a group of objects that are to be learned by the proposed multi-task learning (MTL) networks. The grouping of objects into tasks is an essential aspect of the MTL approach, as it allows for the sharing of information and representations between related tasks. This improves the efficiency of the learning process, as the network can leverage the information acquired from one task to improve the performance of other related tasks.

In order to demonstrate the flexibility and adaptability of the MTL approach, a variety of methods were utilized to group objects in the experiments conducted in this thesis. The specific method used for grouping objects varies depending on the network architecture being evaluated. The criteria for grouping objects into tasks include visual similarity, sample size, and task relatedness.

In summary, the representation of a task in this thesis is defined as an object or a group of objects that are to be learned by the proposed MTL networks. The method for grouping objects into tasks varies depending on the network architecture being evaluated and is discussed in further detail in the thesis. The experiments carried out in this thesis demonstrate the flexibility and adaptability of the MTL approach and its potential to improve performance in various applications.

2.4 Methods in Multi-Task Learning

In the early days, multi-task learning was generally classified into two distinct categories based on parameter sharing. They are hard parameter sharing and soft parameter sharing.

Soft parameter sharing is a method of training a single model to perform multiple tasks simultaneously by allowing the model to learn shared representations that are relevant to multiple tasks, as well as task-specific representations that are only relevant to a single task. It is a type of parameter sharing in which each task is given its own set of parameters, also known as task-specific networks, and feature-sharing algorithms are responsible for handling cross-task communication as shown in Figure (2.1).

The idea behind soft parameter sharing is to allow the model to learn common features that are shared across tasks, while also allowing it to learn task-specific features that are unique to each task. This is achieved by introducing task-specific parameters that are only activated for a particular task and not for other tasks. The task-specific parameters are often referred to as “task gates” or “task masks.”

For example, consider a multi-task learning model that is trained to perform both image classification and object detection tasks. The model might learn shared features such as edge detection and texture features that are relevant to both tasks, as well as task-specific features such as specific object classes or bounding box coordinates that are only relevant to one task. The task-specific parameters would be used to weigh the importance of these task-specific features, allowing the model to focus on the features that are most relevant to each task.

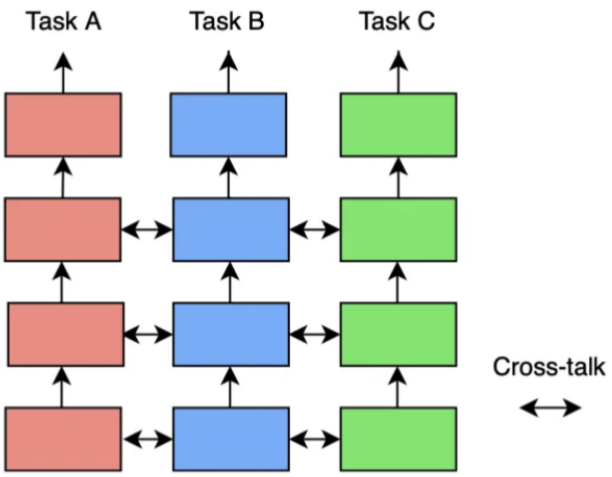

Figure 2.1: Soft Parameter Sharing Method

An advantage of soft parameter sharing is that it allows the model to learn more efficient and compact representations of the data since it can reuse shared features across multiple tasks. This can help to reduce the risk of overfitting since the model is able to learn from a larger and more diverse set of data. In addition, soft parameter sharing can improve the generalization performance of the model, since it allows the model to learn more robust and universal representations that are applicable to multiple tasks.

Soft parameter sharing has been explored in various studies, including the use of cross-stitch networks [11] for soft feature fusion through the linear combination of activations in each layer of task-specific networks, sluice networks [12] for the acquisition of knowledge on selective sharing of layers, subspaces, and skip connections, and NDDR-CNN [13] for the inclusion of dimensionality reduction strategies in feature fusion layers. Lui[14] employed an attention mechanism to distribute a generic feature pool among multiple task-specific networks, resulting in improved performance. However, one disadvantage of soft parameter sharing is that the size of the multi-task network tends to increase linearly with the number of tasks, which can impact scalability.

Hard parameter sharing is a method for multi-task learning in which a single model is trained to perform multiple tasks simultaneously using a single set of shared parameters. It involves dividing the parameter set into operations that are shared and those that are exclusive to the task at hand as shown in Figure (2.2) This means that the model has a single set of weights that is used across all tasks, and the model is optimized to perform all tasks simultaneously. According to Sener et al. in [15], "In hard parameter sharing, a subset of the parameters is shared between tasks while other parameters are task-specific."

Here is an example of how hard parameter sharing might be used in a multi-task learning scenario: Imagine that we want to train a multi-task model to perform both image classification and object detection. We could use a Convolutional Neural Network (CNN) as the base architecture for the model, with shared convolutional and pooling layers for extracting features from the images. We could then add

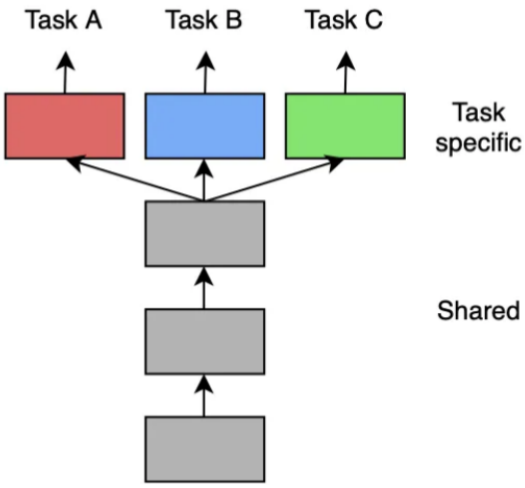

Figure 2.2: Hard Parameter Sharing Method

separate output layers for each task, one for classification and one for detection. The shared layers would be trained to extract features that are relevant for both tasks, while the output layers would be trained to perform the specific task they are designed for.

This approach has the advantage of being simple and straightforward to implement, as it involves training a single model with a single set of weights. It can also be effective when the tasks are related and have similar input and output structures, as the shared layers can learn to extract features that are relevant to both tasks.

UberNet [16] was one of the first architectures that hosted a multi-head design across a variety of network tiers and scales. In spite of this, the architecture that is most representative of hard parameter sharing is one in which there is a shared encoder that divides into task-specific decoding heads [17]. This concept was expanded upon by multilinear relationship networks [18], which did so by putting tensor normal priors on the parameter set of the fully connected layers. The branching points in the network are established on an as-needed basis in these works, which can result in less-than-ideal job groups. Tree-based solutions [19] have been developed to help address this problem.

As mentioned before, sharing model weights across several tasks in order to train each weight to jointly minimize numerous loss functions is an example of the method known as "hard parameter sharing." In the case of soft parameter sharing, various tasks will each have their own particular task-specific models with their own weightings; however, the distance between the model parameters of the various tasks will be added to the combined objective function. In spite of the fact that there is no explicit exchange of parameters, there is an incentive for task-specific models to have parameters that are comparable to one another.

2.5 Data Fidelity in MTL

In the field of multi-task learning (MTL), data fidelity refers to the ability of a model to accurately fit the training data for multiple tasks. This concept is crucial in MTL as it allows the model to perform well on multiple tasks, rather than just one. Kim et al. in [1] formalized the concept of data fidelity by classifying MTL problems into three distinct categories based on the number of distinct input matrices and output vectors for each task. These categories are multi-input single-output multi-input single-output (MISO), single-input multi-output single-input multi-output (SIMO), and multi-input multi-output multi-input multi-output (MIMO) problems.

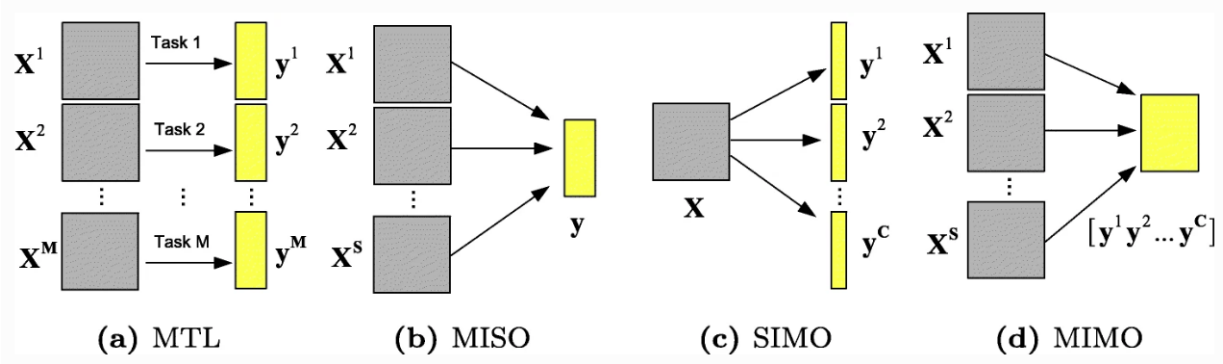

Figure 2.3: MTL in terms of Data fidelity [1]

MISO networks are characterized by having multiple inputs for a single output task as shown in Figure 2.3(b). An example of this is a multi-class classification problem, where multiple sources of data are used to predict a single label. This means that there are multiple input matrices (e.g. images, text, audio) and a single output vector (e.g. class label) for each task.

SIMO, on the other hand, is the networks that define by a single input for multiple output tasks. An example of this is a multi-label classification problem, where a single image is used to predict multiple labels. Here, there is a single input matrix (e.g. an image) and multiple output vectors (e.g. labels) for each task. A network workflow of SIMO is shown in Figure 2.3(c).

MIMO networks have multiple inputs and multiple outputs for multiple tasks as depicted in Figure 2.3(d). An example of this is a multi-modal problem, where multiple modalities of data are used to predict multiple targets. Here, there are multiple input matrices and multiple output vectors for each task.

This classification is important as it can help understand the relationship between inputs and outputs for each task and how they are shared among tasks. It also helps to understand the type of data fitting term that would be required for each case. Furthermore, in real-world applications, it is common to have some input matrices or output vectors shared among tasks. Understanding this sharing of inputs and outputs is important in designing the MTL model and in determining the appropriate data-fitting term. The general form of MTL is depicted in Figure 2.3(a) along with its three distinct special cases,

which are distinguished by the degree of sharing of inputs or outputs among the various tasks. Each of these special cases has a unique form of data fitting term and a slightly altered definition of the task and relationship.

2.6 Recent Work on MTL in Computer Vision

Although MTL is used in many disciplines of deep learning, such as natural language processing, reinforcement learning, and computer vision, our thesis focuses on developing an MTL system for computer vision problems in autonomous vehicles. Consequently, we have compiled and analyzed recent work on MTL in computer vision for autonomous vehicles.

2.6.1 Joint Semantic Understanding with a Multilevel Branch for Driving Perception

In the paper [2] by Lee et al., the authors propose a multi-task learning framework for simultaneous traffic object detection, drivable area segmentation, and lane line segmentation in the context of autonomous driving as shown in Figure (2.4). The authors argue that this approach can provide efficient learning and improved prediction accuracy, as different tasks in the context of autonomous driving often have mutual information and a shared representation can capture this information.

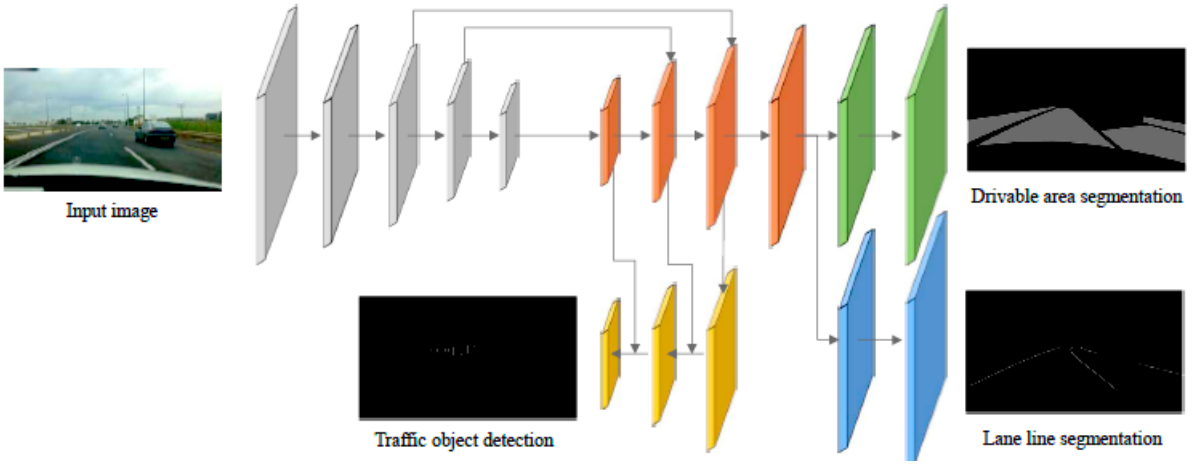

Figure 2.4: The overall architecture of JSUMBDP. The input image is fed into the encoder part, then the three decoders share the representations for different tasks at each branch. [2]

The proposed framework consists of an encoder and three decoders, with the encoder based on the CSPDarknet [20]. The CSPDarknet is a variant of the DenseNet [21] architecture that preserves the advantage of feature reuse, which refers to the ability of the network to reuse features learned from previous layers while preventing excessively duplicate gradient information. This leads to faster computation and fewer parameters, which is important for real-time applications such as autonomous driving. In addition

to the CSPDarknet, the encoder also uses a feature pyramid network and a spatial pyramid pooling module to extract rich image features from the input image. The feature pyramid network [22] fuses features at different semantic levels with multiple pieces of semantic information in multiple scales, which allows for the capture of multiple levels of information in the image. The spatial pyramid pooling module [23] generates and fuses different scale features, which allows for the capture of information at different scales in the image.

The decoders are structured based on the similarity of the tasks, with the drivable area and lane line segmentation decoders sharing a separate feature map to learn the bottom region of the image. This is motivated by the fact that drivable areas and lane lines share the bottom region of the image of the road area, while traffic objects are typically located a little above. The traffic object detection decoder, on the other hand, uses a structure based on the path aggregation network [24]. The path aggregation network is a one-stage object detector that fuses multi-scale feature maps of semantics and positioning and allows for predictions at multiple scales by assigning multiple anchors to each grid cell. This enables the network to make predictions at different scales, which is important in the context of autonomous driving where objects can be at different distances from the camera. The detection head in the traffic object detection decoder predicts the scaling of the height and width, offset of position, and the corresponding probability of each category for each anchor.

To evaluate the performance of the proposed framework, the authors conducted experiments on the BerkeleyDeepDrive100K (BDD100K) dataset. The BDD100K dataset consists of 100,000 images of diverse driving environments, including urban, suburban, and highway scenes. The authors compared the proposed method with multi-task and single-task methods in terms of accuracy and real-time inference. In terms of accuracy, the authors found that the proposed method outperformed the competing methods in all three tasks. For traffic object detection, the proposed method achieved an average precision of 82.5%, which was higher than the competing multi-task method 80.6%, and the single-task method 77.7%. For drivable area segmentation, the proposed method achieved a mean intersection-over-union (IoU) of 92.3%, which was higher than the competing multi-task method 91.5% and the single-task method 90.9%. For lane line segmentation, the proposed method achieved a mean IoU of 96.7%, which was higher than the competing multi-task method 96.1% and the single-task method 95.8%.

In terms of real-time inference, the authors found that the proposed method maintained a frame rate of more than 37 frames per second, which is fast enough for real-time applications. The authors attribute the fast inference time to the efficient design of the network, which uses the CSPDarknet to reduce the number of parameters and calculations and the nearest interpolation method for upsampling feature maps instead of deconvolution.

Overall, the results of the experiments suggest that the proposed multi-task learning framework for simultaneous traffic object detection, drivable area segmentation, and lane line segmentation is effective

and efficient for autonomous driving applications. The proposed method outperforms competing methods in terms of accuracy and maintains a fast frame rate for real-time inference. The use of the CSPDarknet [20] and the multi-level branch structure in the proposed method likely contribute to its improved performance.

In conclusion, the study by Lee and Kim presents a multi-task learning framework for the driving perception that simultaneously performs traffic object detection, drivable area segmentation, and lane line segmentation. The proposed method utilizes a shared encoder and three decoders, with each decoder handling a specific task. The decoders share feature maps with more similar tasks for joint semantic understanding, and multiple loss functions are automatically weighted and summed to learn multiple objectives simultaneously. The proposed method is trained end-to-end and was tested on the BDD100K dataset, achieving improved accuracy and real-time inference compared to competing multi-task and single-task methods. The authors suggest that the efficient design of the network, including the use of the CSPDarknet and the nearest interpolation method for upsampling, contribute to the improved performance of the proposed method. The results of this study provide valuable insights into the development of visual perception systems for autonomous driving and highlight the potential of multi-task learning approaches in this domain.

2.6.2 YOLOPv2: Better, Faster, Stronger for Panoptic Driving Perception

The paper [3] by Cheng Han et al. presents a multi-task learning network for object detection, drivable area segmentation, and lane detection tasks in the context of autonomous driving. These tasks are crucial for autonomous vehicles to achieve a comprehensive understanding of their surroundings and make accurate and timely decisions during the driving stage. While many approaches have been proposed for these tasks individually, it is often impractical to run separate models for each task in a real-time autonomous driving system due to the computational cost. Multi-task learning networks provide a potential solution for this problem by sharing an encoder among different tasks and having task-specific decoder heads.

The authors propose a multi-task learning network, YOLOPv2, that is inspired by the YOLO [25] family of networks and the HybridNet [26] approach. They design a more efficient network architecture with a shared encoder for feature extraction and three separate decoder heads for specific tasks. They also use various optimization strategies, such as data augmentation techniques and a hybrid loss function, to improve the performance of their network. In their experiments, the authors compare YOLOPv2 to several state-of-the-art (SOTA) multi-task models on the challenging BDD100K dataset and show that their network performs better in all three tasks while also being faster.

The proposed network, shown in Figure 2.5, consists of a shared encoder and three decoder heads for specific tasks. The encoder is based on the design of E-ELAN in [27] and uses group convolution to

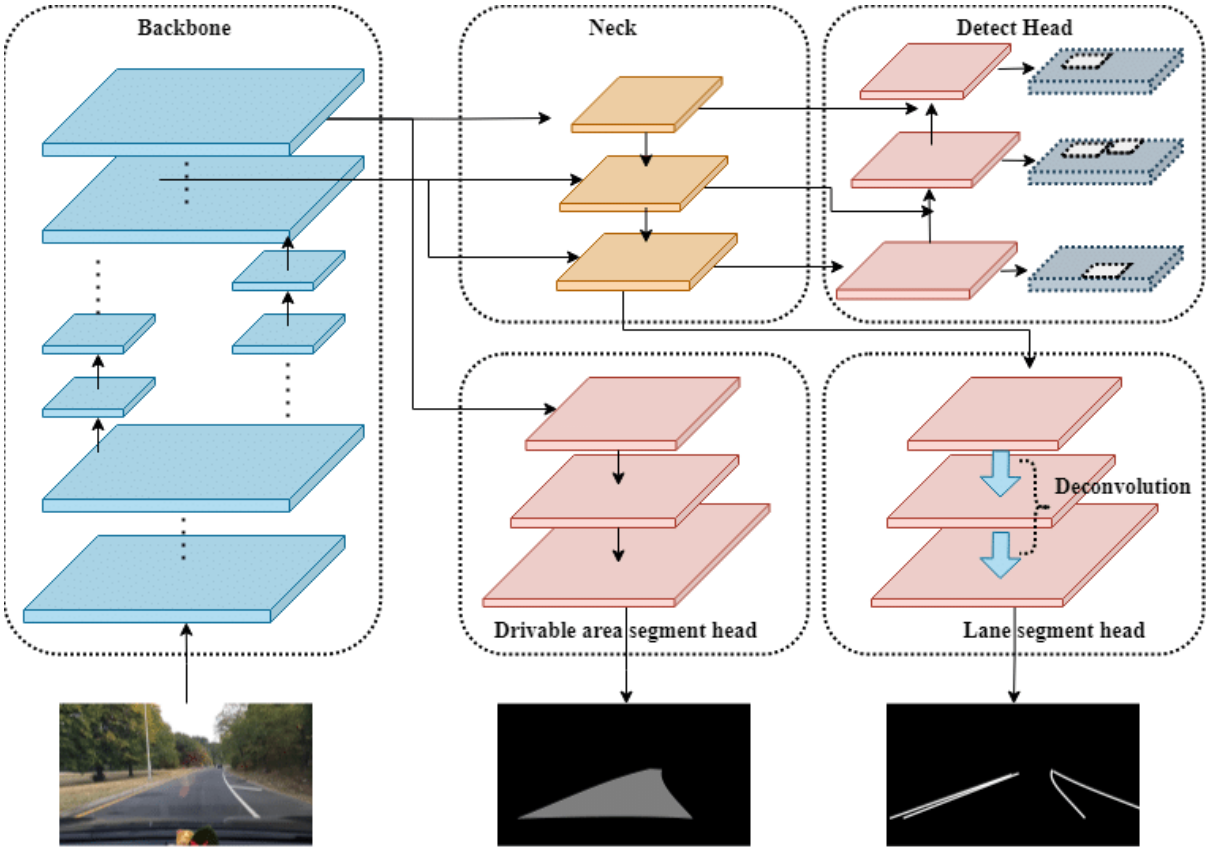

Figure 2.5: The Network of YOLOPv2 [3]

extract diverse features from the input image. It also includes a neck part where features generated from different stages are collected and fused by concatenation. The encoder also includes a Spatial Pyramid Pooling (SPP) module [23] and a Feature Pyramid Network FPN module [22] to fuse features at different scales and semantic levels, respectively.

The decoder heads are designed specifically for each task. For the object detection task, the decoder head is similar to the YOLOv7 [27] approach, using an anchor-based multi-scale detection scheme with a Path Aggregation Network (PAN)[24] for better localization feature extraction. The drivable area segmentation and lane detection decoder heads are also designed differently to suit the specific requirements of each task. The drivable area segmentation head is connected to the FPN module and includes additional upsampling layers to compensate for the loss of deeper features. The lane detection head is connected to the end of the FPN module to extract features at a deeper level and includes a lane context block for global context information.

In terms of optimization strategies, the authors use several data augmentation techniques, such as Mosaic and Mixup, for data preprocessing and a hybrid loss function that combines the standard cross-entropy loss for classification tasks with the Dice loss for segmentation tasks. They also use a cosine annealing learning rate schedule with a warm restart in the first 3 epochs and a total of 300 training epochs.

The paper presents a comparison of YOLOPv2 to several SOTA multi-task models on the BDD100K dataset. The authors compare the model parameters and inference speed, as well as the performance on the three tasks of object detection, drivable area segmentation, and lane detection.

In terms of model parameters and inference speed, the authors show that YOLOPv2 has a stronger network structure and more parameters compared to the other models, but performs faster due to its efficient network design and sophisticated memory allocation strategy. In the object detection task, YOLOPv2 achieves a higher mean average precision @ IoU = 50% (mAP50) at 50% and competitive recall compared to the other models. In the drivable area segmentation task, YOLOPv2 achieves the best performance with a mean intersection-over-union (mIOU) of 0.93. In the lane detection task, YOLOPv2 achieves the highest accuracy and a decent lane IoU compared to the other models.

In conclusion, the paper "YOLOPv2: Better, Faster, Stronger for Panoptic Driving Perception" presents a multi-task learning network for object detection, drivable area segmentation, and lane detection in the context of autonomous driving. The authors propose a more efficient network architecture with a shared encoder and task-specific decoder heads, as well as various optimization strategies to improve the performance of their network. In experiments on the challenging BDD100K dataset, YOLOPv2 performs better in all three tasks and is faster than several SOTA multi-task models. Overall, the work in this paper is a significant contribution to the field of autonomous driving perception and can provide valuable insights for future research in this area.

2.6.3 Multi-task Learning with Attention for End-to-end Autonomous Driving

In the paper [4] by Keishi Ishihara et al., the authors propose a novel multi-task attention-aware neural network for vision-based end-to-end autonomous driving. The architecture design of the proposed network is given in Figure (2.6). The network learns to predict control signals, semantic segmentation, depth estimation, and the color state of traffic lights from a monocular RGB camera image in an end-to-end manner. The authors also introduce an attention mechanism to improve control performance by focusing on salient regions in the extracted features.

The authors adopt the Conditional Imitation Learning (CIL) framework, like the one proposed by Codevilla et al. in [28], to train the proposed network in an end-to-end manner. The CIL framework involves training a deep network to predict control signals (steering, throttle, and brake) from a given image and high-level command (indicating the direction the agent should proceed from its current position). The authors also incorporate multi-task learning, like the one proposed by li et al. in [29], by including the tasks of semantic segmentation and depth estimation in the training process, as well as traffic light classification. This is intended to encourage the network to learn generalizable scene representations that are effective for decision-making.

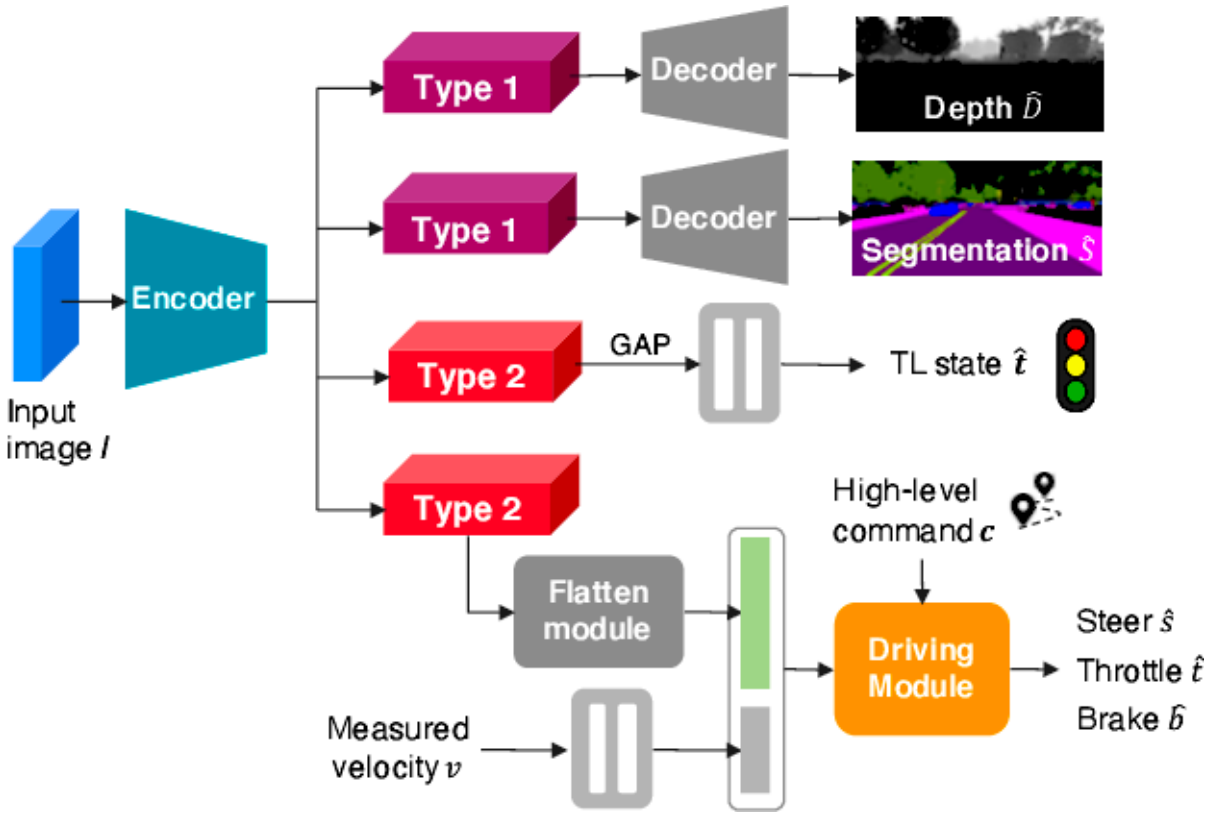

Figure 2.6: The overall architecture of MTL with Attention. The input image is fed into the encoder part, then the three decoders share the representations for different tasks at each branch. [4]

The proposed network architecture consists of an encoder, two decoders, a traffic light state classifier, a flatten module, a velocity encoder, and a driving module, as shown in Figure 2.6. The main goal of the network is to predict control signals from a monocular RGB camera in an end-to-end manner.

The encoder component, as shown in Figure 2.7, of the network is responsible for extracting useful features from the input image and implementing an attention mechanism [14, 30]. It generates two types of attention-weighted latent feature maps, which are used as input for the decoders and the traffic light state classifier. The flatten module is responsible for flattening the feature maps into a 1D vector, which is then fed into the velocity encoder. The velocity encoder processes the velocity measurement and concatenates it with the flattened feature maps to create a final feature vector. This feature vector is then fed into the driving module, which predicts the control signals (steering, throttle, brake) for the vehicle.

The two decoders in the network are responsible for predicting the semantic segmentation and depth maps, respectively. These sub-tasks are meant to encourage the network to learn generalizable scene representations that are effective for decision-making as was done in [29]. The traffic light state classifier is a separate component of the network that is trained to predict the color state of traffic lights based on the input image. This allows the operator to easily debug the model performance by looking at its predictions.

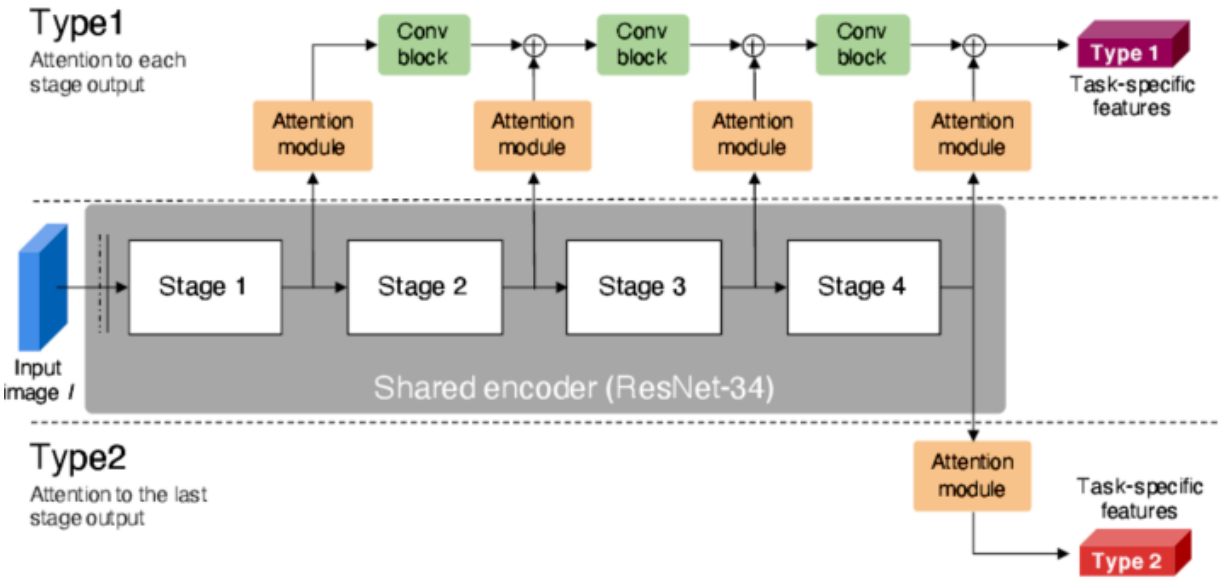

Figure 2.7: Visualization of the ResNet34-based encoder network with two attention mechanisms.[4]

One of the key contributions of this paper is the introduction of multi-task learning to improve the generalizability of the network for autonomous driving. By learning additional tasks such as semantic segmentation and depth estimation alongside the main task of predicting control signals, the network is able to learn more meaningful and generalizable scene representations. The authors also demonstrate the effectiveness of the attention mechanism in improving control performance and provide visualization of the saliency maps to show how the attention layers change the focus of the network.

To evaluate the effectiveness of the proposed approach, the authors evaluate the proposed approach on the original CARLA benchmark (CoRL2017) and the NoCrash benchmark. The CoRL2017 benchmark consists of four different driving conditions, including lane-keeping and navigation tasks, and measures the success rate of the agent in reaching a goal point within 2 meters. The NoCrash benchmark consists of more complex scenarios and measures the ability of the agent to handle collisions and traffic light violations, as well as the percentage of human interventions per kilometer.

The authors, collected their own dataset using the CARLA simulator, which consists of front-faced monocular RGB camera images, auto-generated semantic segmentation maps, depth maps, and measurements including steering, throttle, brake, speed, high-level command, and ground-truth traffic light state. The dataset is collected under four different weather conditions in the Town01 environment and is split into a training set and a validation set. To improve the generalization performance of the network, the authors apply various data augmentation techniques, including gaussian noise, blurring, pixel dropout, contrast normalization, and PCA color augmentation. They also perform undersampling based on steering values to balance the skewed distribution of the dataset. The network is trained using the Adam optimizer with a learning rate of 10^{-4} and a batch size of 32. The authors also use early stopping and model ensembling to further improve the performance.

The results showed that the proposed approach outperforms or is comparable with the state-of-the-art models on both benchmarks. The results showed that the proposed method outperformed the baseline approaches on the CoRL2017 benchmark, with a particularly strong performance in the "new town" and "new town & weather" conditions. The proposed approach is able to maintain a high success rate across all driving tasks and generalization contexts, including in previously unobserved environments and weather conditions. In the NoCrash benchmark, which measures the ability of the agent to handle more complex events and traffic congestions, the proposed approach also demonstrated superior performance compared to the baselines. The authors also perform an analysis of traffic light infractions and find that the proposed approach is able to significantly reduce the number of infractions compared to the baseline models.

In conclusion, the proposed multi-task attention-aware neural network shows promising results in improving the performance of end-to-end autonomous driving systems. The inclusion of additional tasks and an attention mechanism helps the network learn generalizable and interpretable representations for decision-making, leading to better performance in previously unobserved environments. Overall, the authors have made a significant contribution to the field of autonomous driving with their proposed multi-task attention-aware network, and their results suggest that this approach could be a promising solution for vision-based end-to-end [29, 31] control in autonomous vehicles.

2.6.4 On Generalizing Detection Models for Unconstrained Environments

In the paper [5] by Prajjwal Bhargava, the author presents a method for adapting a model trained on one data distribution to perform well on a new, different data distribution without forgetting the information it has already learned. The authors evaluate their method on two datasets: the India Driving Dataset (IDD), which represents unstructured environments, and the BDD100K dataset, which represents structured environments [32]. Although multi-task learning is not the central focus of this research in its entirety, there are other components of the study that are very relevant to our thesis.

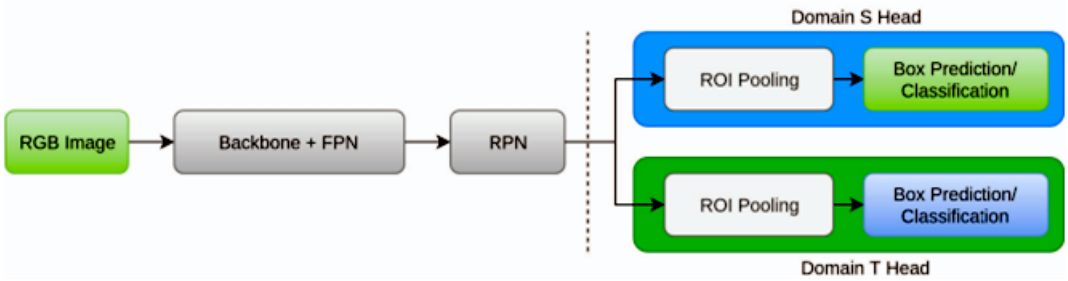

Figure 2.8: Overview of System Architecture [5]

As shown in Figure (2.8), the author uses a region proposal-based approach for object detection, with the ResNet50 network pre-trained on the Common Objects In Context (COCO) dataset as the feature extractor network (FPN). The network is trained using the Stochastic gradient descent (SGD) optimizer

with momentum and weight decay set to 0.9 and 0.00004, respectively. The learning rate is initially set to 0.001 and is optimized using the cyclical learning rate scheduler. The authors use random horizontal flipping for data augmentation and set the batch size to 4 for 5 epochs per camera orientation.

To adapt the model to the new target domain, the authors use a technique called domain-specific heads, where they add additional classifiers (called ROI heads) to the model that are specific to the target domain. The weights of the feature extractor and the RPN are shared across all of the domain-specific classifiers, which allows the model to learn common features across all domains while keeping the number of parameters the same.

To fine-tune the model and avoid catastrophic forgetting, as suggested in [33], the authors use a technique called discriminative fine-tuning [34], where they use different learning rates for different layers of the network. They use a lower learning rate for the shared layers of the network (such as the FPN and RPN) and a higher learning rate for the domain-specific layers (such as the ROI heads). This allows them to inhibit the loss of learned information in the shared layers while allowing the domain-specific layers to learn more quickly.

The authors also use a technique called gradual unfreezing, where they gradually unfreeze and fine-tune the different components of the network, starting with the domain-specific ROI head and gradually adding more components as the training progresses. This allows the model to adapt to the new data distribution while minimizing the loss of information learned from previous tasks.

To optimize the network, the authors use a technique called cyclical learning rate (CLR), which involves cycling the learning rate between lower and upper bounds instead of gradually decreasing it as the training converges. This helps to prevent the network from converging at poor local minima in the loss landscape and allows it to oscillate towards a higher learning rate when necessary. The authors use a triangular variation of CLR for their experiments.

S and *T*	Epoch	Active components (with LR)	LR Range	mAP (%) at specified epochs
BDD→IDD	5	+ROI Head(1e-3)	1e-3, 6e-3	24.3
IDD→BDD	Eval		-	45.7
BDD→IDD	5,9	+RPN (1e-4)	1e-4, 6e-4	24.7, 24.9
IDD→BDD	Eval	+ROI head (1e-3)	-	45.3, 45.0
BDD→IDD	1,5,6,7	+RPN (1e-4)	1e-4, 6e-3	24.3, 24.9, 24.9, 25.0
IDD→BDD	Eval	+ROI head (1e-3)	-	45.7, 44.8, 44.7, 44.7
BDD→IDD	1,5,10	+ROI head (1e-3)	1e-4, 6e-3	24.9, 25.4, 25.9
IDD→BDD	Eval	+RPN (4e-4) +FPN(2e-4)	-	45.2, 43.9, 43.3

Figure 2.9: Change in mAP with varying learning rates for different active components. Results reported on validation sets of T (IoU=0.5) [5]

As stated before, the author evaluated the method on both the IDD and BDD100K datasets and find that it leads to a considerable performance improvement on the unstructured data (IDD) while retaining good performance on the structured data (BDD100K). Figure 2.9 shows the results for the IDD dataset,

with the baseline model denoted as BDD→IDD and the proposed method denoted as IDD→BDD. The authors find that the use of domain-specific heads and discriminative finetuning with gradual unfreezing leads to a significant increase in performance, with an mAP of 24.3% for the baseline model and an mAP of 35.6% for the proposed method.

Figure (2.9) also shows the results for the BDD100K dataset, with the baseline model denoted as IDD→BDD and the proposed method denoted as BDD→IDD. The authors find that the use of domain-specific heads and discriminative finetuning with gradual unfreezing leads to a slight decrease in performance on the structured data, with an mAP of 45.7% for the baseline model and an mAP of 44.8% for the proposed method. However, this decrease is small compared to the increase in performance on the unstructured data, and overall, the proposed method demonstrates good performance on both structured and unstructured data.

The authors also investigate the effect of varying the learning rate range and the choice of active components during training on the performance of the model. They find that the learning rate range plays a crucial role in determining the rate at which weights change in the active components and that a larger range leads to a larger performance improvement on the target data (IDD). They also find that adding more active components during training, such as the FPN and the RPN, can further improve the performance of the model on the target data.

The authors also compare their method to other approaches for adapting object detection models to new data distributions, including fine-tuning the entire model, using domain adaptation techniques such as weight adaptation and feature adaptation, and using knowledge distillation to transfer knowledge from a larger model to a smaller model. They find that their method outperforms these other approaches, demonstrating the effectiveness of their approach for adapting object detection models to new, unconstrained environments.

Overall, the proposed approach presents a promising solution for adapting object detection models to new, unconstrained environments. By using domain-specific heads, discriminative fine-tuning, gradual unfreezing, and cyclical learning rate, the authors are able to adapt the model to the new data distribution while minimizing the loss of information learned from previous tasks and demonstrate good performance on both structured and unstructured data. The results of their experiments suggest that their method is an effective approach for adapting object detection models to new data distributions and has the potential to be applied to a wide range of tasks in the field of object detection.

2.6.5 Simultaneous vehicle and lane detection via MobileNetV3

In the paper [6] by Tianmin Deng et al., the authors propose a deep learning network architecture for real-time vehicle detection and lane line segmentation in the context of a car following scenarios. The network consists of four main parts as shown in Figure (2.10): a feature extraction block, a feature processing

block, a task-specific processing block for vehicle detection, and a task-specific processing block for lane line segmentation.

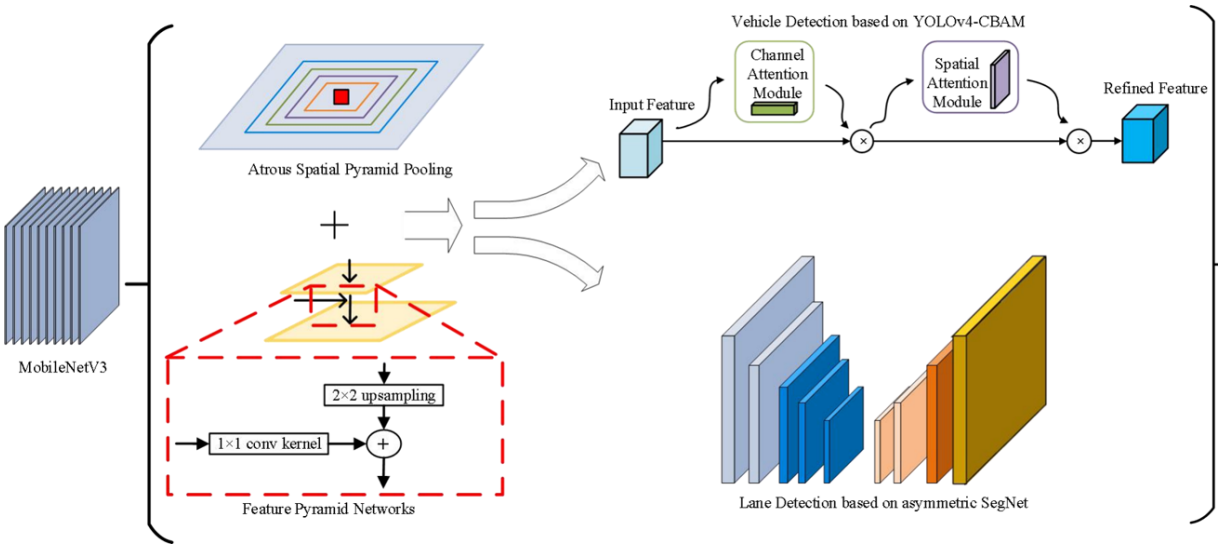

Figure 2.10: Architecture of proposed vehicle and lane detection. [6]

The feature extraction block is based on MobileNetV3, a lightweight and efficient network architecture for image classification tasks. To improve the feature extraction ability and real-time performance of MobileNetV3, they introduce two modules: a multi-scale receptive field module based on Atrous Spatial Pyramid Pooling (ASPP) and a feature fusion module based on FPN.

The multi-scale receptive field module aims to capture contextual information from road images at different scales, which is useful for both vehicle detection and lane line segmentation. It does so by using full convolutions with different void ratios, allowing a single pixel to acquire multiple receptive fields and thus gather information at multiple scales.

The feature fusion module, on the other hand, is designed to improve the detection accuracy of vehicles and lane lines of different scales. It achieves this by combining feature maps from both a bottom-up and a top-down branch, using 2x2 up-sampling to increase the size of the top-down feature maps. The resulting combined feature maps are then used for vehicle and lane line detection.

The task-specific processing block for vehicle detection is based on YOLOv4 , a state-of-the-art object detection model, shown in Figure (2.11). To further improve the performance of YOLOv4, an attention mechanism called convolutional block attention module CBAM is introduced. YOLOv4 is an end-to-end object detection method that has gained attention due to its high real-time performance and precision. However, as the input size increases, the scale of the model and the computation required also increase significantly. To improve the performance of feature extraction and detection accuracy while still maintaining a small model size and low computational requirements, the CBAM is used into YOLOv4. CBAM divides the attention process into two independent parts: a channel attention module and a spatial attention module. This allows for a reduction in parameters and computation while also

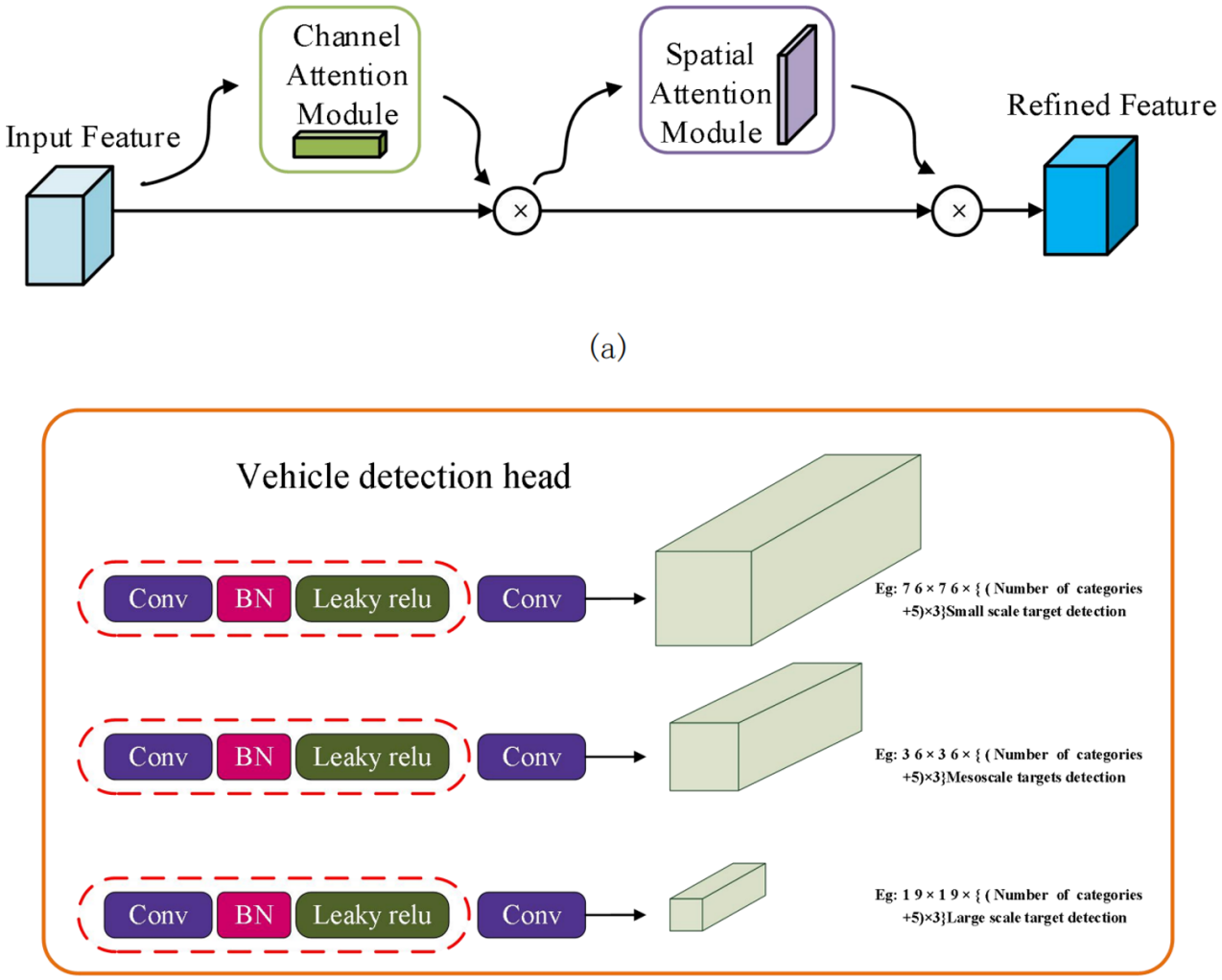

Figure 2.11: Vehicle detection based on YOLOv4-CBAM [6]

making it possible to easily integrate the module into existing network architectures.

The vehicle detection head block in the authors' proposed network architecture consists of three branches: convolution, batch normalization, and leaky rectified linear activation unit (ReLU). Each position in this block predicts 3 anchor boxes of different scales for the detection of large, medium, and small vehicles, and the output dimensions are 76x76x(number of categories+5)x3, 38x38x(number of categories+5)x3, and 19x19x(number of categories+5)x3. The output includes 4 regression parameters for the position of the target vehicle in the road image (i.e., the x and y coordinates of the center point of the anchor point box and the length and width of the anchor point box) as well as a confidence level. Overall, the use of YOLOv4-CBAM in the vehicle detection task of the proposed network is intended to improve the real-time performance and accuracy of vehicle detection.

The task-specific processing block for lane line segmentation is based on SegNet, a network architecture for semantic segmentation tasks, as shown in Figure (2.12). To improve the real-time perfor-

mance of SegNet for this task, the authors modify it to an asymmetric network architecture with "more encoding-less decoding", which reduces the number of image channels and the number of hyperparameters in the model. They also include 1x1 convolution and batch normalization layers to reduce computation and accelerate convergence.

The lane detection block in the authors' proposed network architecture includes two 2x2 upsampling operations on the maximum feature map output by the FPN and sequential feature map merging to obtain features of the same size as the original image. The first feature fusion merges the feature map with the feature map after standard convolution and average pooling/upsampling, while the second fusion merges the FPN output (1/8 the size of the original image) with the feature map before output. These fusions are intended to better extract semantic information about lane lines and improve lane line detection accuracy.

To handle multi-scale input, the authors also include a multi-scale receptive field block in the network, which performs convolution at different scales to capture features at multiple scales.

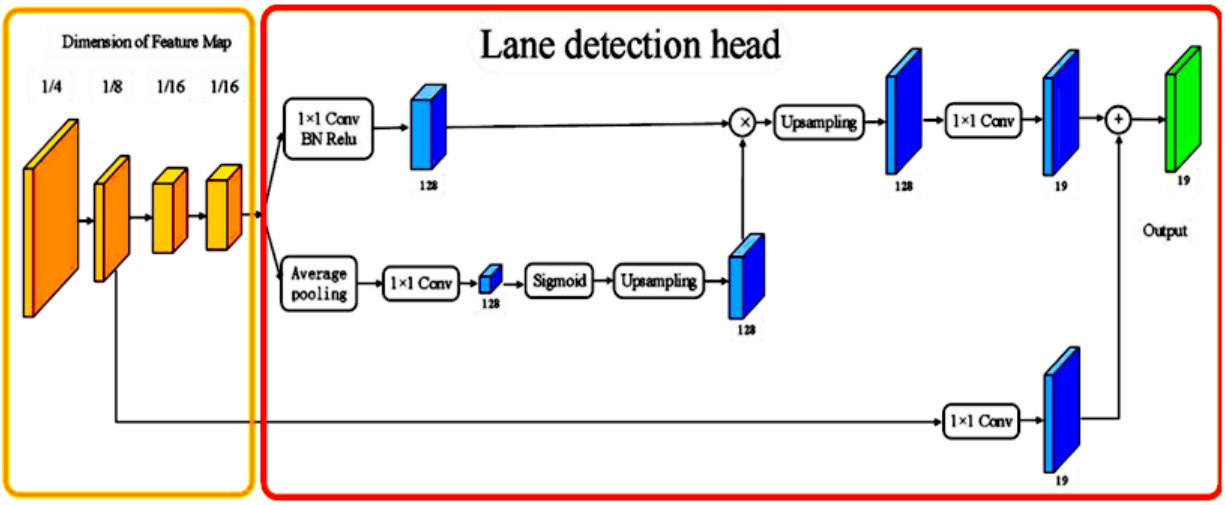

Figure 2.12: Lane Detection head [6]

The final model is trained using a loss function that combines the loss functions of both task-specific processing blocks, with a weighting coefficient applied to balance their relative importance. the authors describe the use of an improved focal loss function for the vehicle and lane line joint detection task in their proposed network architecture for real-time vehicle detection and lane line segmentation in the car following scenarios. The loss function is designed to measure the loss of both vehicle and lane line detection at the same time and is divided into three parts: the loss of the regression box, classification, and confidence.

For the classification task, the authors use the Focal Loss function, which is based on the cross-entropy of binary classification and is designed to address imbalanced sample proportions. For the loss of the regression box, they use the Distance-IoU loss function, which minimizes the center distance between the detection frame and the target frame by penalizing the distance between their center points and speeding

up convergence. They use the Dice loss function, commonly used for semantic classification in medical image segmentation, for lane line detection due to the similarity to blood vessel segmentation.

The total loss function is a combination of these three individual loss functions, with a certain degree of competition between the detection and segmentation tasks in the optimization direction. To address this, the authors design a dynamic adjustment loss function to dynamically adjust the scale coefficients of the two tasks based on the mAP (mean average precision) and IoU (intersection over union) of the current model and the optimal values for single task detection and segmentation. The dynamic coefficients are regularized into a unit vector to be used in the total loss function. This helps to prevent the optimization from falling into a local maximum and allows for better performance on both tasks.

To evaluate the performance of the proposed method, the authors use the BDD100K dataset for training and the Karlsruhe Institute of Technology and Toyota Technological Institute (KITTI) and Chongqing traffic road image datasets for testing. Data augmentation methods, including cuts, scales, level flips, and the addition of noise, are applied to the training data in order to improve the robustness of the model. The Adam algorithm is used to optimize the model parameters, with a learning rate decay applied at each round of training. The authors also report on the performance of the model on the KITTI dataset, as well as on the Chongqing traffic road image dataset under both day and night conditions, in order to assess the generalization performance of the improved vehicle and lane line joint detection model.

The authors conducted a loss function analysis to evaluate the effectiveness of the dynamic adjustment loss function. They found that using the dynamically adjusted loss function for training improved the detection accuracy and segmentation accuracy of the model compared to using a fixed configuration ratio. Specifically, they observed improvement in the AP50, Recall, F1, Precision, and IoU metrics.

In addition, the authors conducted an ablation study to better understand the contributions of the multi-scale receptive field module and the FPN to the performance of the multi-task joint algorithm. They found that the introduction of both the ASPP and FPN significantly improved the performance of the algorithm, with the ASPP leading to increases in Precision, Recall, AP50, F1, and IoU of +3.2, +2.6, +2.6, +1.9, and +3.7, respectively, and the FPN leading to increases of +3.8, +3.4, +3.5, +2.7, and +2.9, respectively. When both the ASPP and FPN were introduced, the algorithm saw further improvements, with increases of +5.8, +5.3, +6.1, +4.7, and +5.9, respectively. The authors also noted that the introduction of the FPN led to a larger increase in time overhead compared to the ASPP, but that the improvements in accuracy made the trade-off worthwhile.

The authors also compared the performance of their proposed multi-task joint algorithm to that of the YOLOv4 and SegNet algorithms on the BDD100K dataset. They found that the multi-task joint algorithm outperformed both the YOLOv4 and SegNet in all performance metrics, with improvements of +1.1 AP50, +0.9 Recall, +0.7 F1, and +0.3 Precision compared to the YOLOv4, and an improvement

of +1.2 IoU compared to the SegNet. In terms of detection speed, the multi-task joint algorithm was 1.7 times and 3.2 times faster than the YOLOv4 and SegNet, respectively.

Finally, the authors compared the performance of their proposed algorithm to several state-of-the-art models on the BDD100K and KITTI datasets. They found that their algorithm performed competitively or outperformed these other models in terms of target detection and semantic segmentation. For example, on the BDD100K dataset, the multi-task joint algorithm achieved the highest F1 score of 0.908 for target detection and the second-highest IoU score of 0.812 for semantic segmentation. On the KITTI dataset, the algorithm achieved the second-highest mAP score of 0.678 for target detection and the highest IoU score of 0.746 for semantic segmentation.

Overall, the results of the experiments suggest that the proposed multi-task joint algorithm is effective at simultaneously performing target detection and semantic segmentation and that it performs competitively with state-of-the-art models in both tasks. In particular, the model was able to achieve a high level of accuracy in both target detection and semantic segmentation, as demonstrated by its strong performance on the BDD100K and KITTI datasets. Additionally, the model was able to maintain a high level of performance across a variety of different lighting and weather conditions, further demonstrating its robustness and adaptability. The dynamic adjustment loss function was also shown to be effective at improving the convergence speed of the model and enhancing its overall performance. In terms of implementation, the use of MobileNetV3 as the backbone network allowed for a significant reduction in the number of network parameters, resulting in a faster detection and segmentation speed compared to YOLOv4 and SegNet. Overall, the proposed multi-task joint algorithm represents a promising approach for performing simultaneous target detection and semantic segmentation in real-time applications.

CHAPTER 3

System Architecture Design and Implementation

In this chapter, we will introduce our proposed scalable multi-tasking system. We begin by introducing the base system, Mask R-CNN, the foundational technology that allows us to develop our proposed system. Then, we discuss the design and implementation of our system architecture. Finally, towards the very end, we went over the loss function considerations for our system.

3.1 Introduction

As stated before, our proposed system is a scalable multi-tasking system called Multi-Tasks Learning R-CNN, MTL R-CNN. It allows the construction of any multi-task learning networks at different scales for computer vision tasks, such as object detection and instance segmentation. Our system is an enhancement of the base algorithm, called Mask R-CNN. Mask R-CNN is a state-of-the-art computer vision algorithm, and the workflow of its architecture is given in Figure 3.1. Further details are provided in the next section.

3.2 Base System

According to Kaiming He et al., the Mask RCNN [35] network model is an extension of Faster R-CNN, capable of precise object detection and instance segmentation. It consists of two stages as shown in this study[36]: 1) The first stage is for feature extraction and region of interest proposal. It does so by using two distinct networks: A backbone network composed of ResNet (elaborated in Appendix A.3.6) in conjunction with a feature pyramid network FPN for the feature extraction. And a region proposal

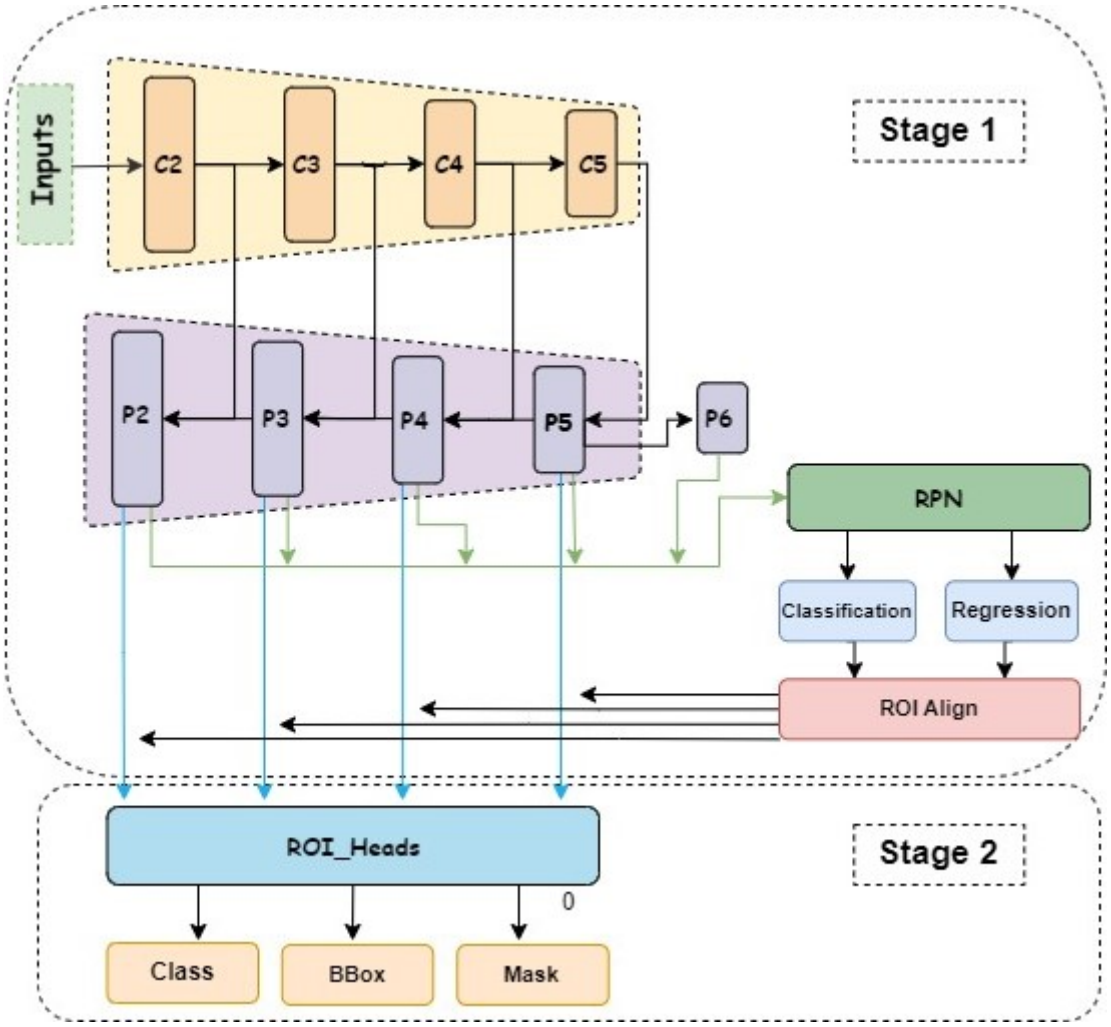

Figure 3.1: Workflow of Mask RCNN Architecture

network that proposes regions in the extracted feature map that contains the object. 2) In the second stage, the network predicts bounding boxes, object classes, and object masks for each of the proposed regions obtained in stage 1. Also, it introduces the region of interest (RoI) Align strategy to overcome the problem of misalignment by enhancing RoI pooling. Figure 3.1 depicts the workflow of the Mask RCNN method. There are three major changes from faster R-CNN to mask R-CNN: First, the Use of FPN's Feature pyramid networks for feature extraction. FPN is a feature extractor that generates multi-scale feature maps. FPN composes of a bottom-up and a top-down pathway. The bottom-up pathway is the usual convolutional network for feature extraction and the top-down pathway constructs higher-resolution layers from a semantic-rich layer. Second, replacement of the RoI Pool in faster R-CNN with RoI Align in Mask R-CNN. RoI Align solves the problem of area mismatch caused by RoI pooling in two quantization processes by using bilinear interpolation. It improves object detection accuracy and lays the foundation for pixel-level instance segmentation. Third, the introduction of an additional branch of FCN to generate masks. The FCN solves the problem of image segmentation at the semantic level by classifying images at the pixel level. Mask R-CNN uses FCN in the mask branch and uses full convolution to generate mask

regions to implement instance segmentation.

3.3 Base System Architecture

The architecture of the base model, Mask R-CNN, can be described more thoroughly by explaining the code implementations of Mask R-CNN in the Torch vision library of Pytorch [37]. In Torch vision, Mask R-CNN is defined as a child class of the Faster R-CNN class. Additionally, the Faster R-CNN class inherits from the superclass called the Generalized R-CNN class. The child classes initialize the respective network heads, detection heads, and mask heads, before, calling the forward function in the superclass.

3.3.1 Generalized RCNN

The Generalized R-CNN serves as a fundamental building block for the Mask R-CNN algorithm. It is a composite architecture that comprises of three key components, which are sequentially arranged as illustrated in Figure 3.2: the backbone network, the RPN, and the RoI Head.

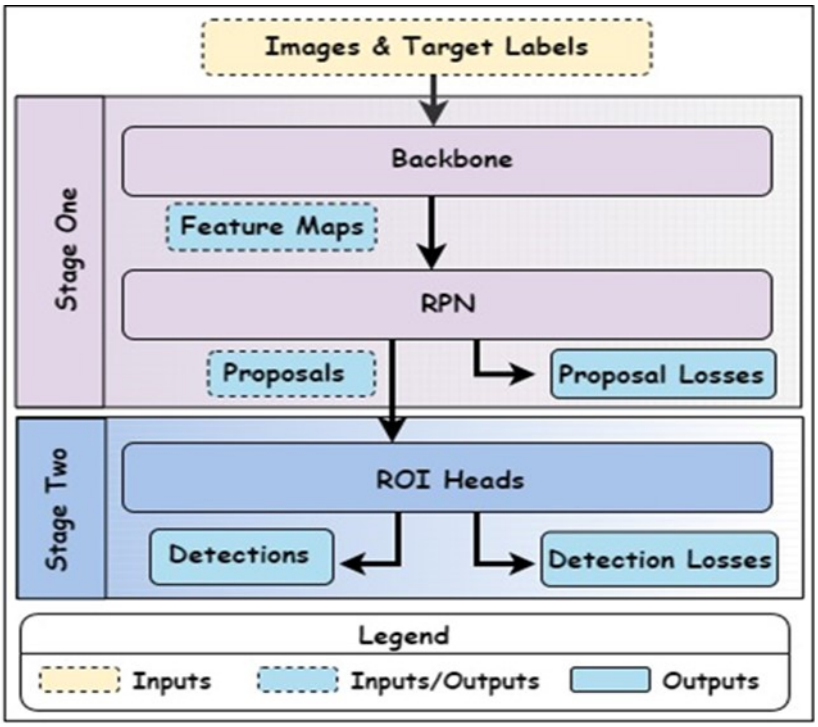

Figure 3.2: Workflow of Generalized RCNN Class

The backbone network is responsible for extracting feature maps from the input images. These feature maps are then used as input for the RPN, which generates a set of proposals. These proposals are used to train the model by calculating the proposal losses. The RPN is used to identify the regions of the images that are most likely to contain objects.

The generated proposals are then passed on to the RoI Head network along with the extracted feature maps and input labels. The RoI Head network then uses this information to generate the final output of the model, which includes the detection and detection losses. The detection losses are used to train the model and the detection output is used to detect the objects in the images.

To summarize, the Generalized R-CNN serves as a base class for Mask R-CNN, where the input images are processed by the backbone network to extract feature maps, which are then used as input for the RPN to generate proposals. Then the generated proposals are passed on to the RoI Head network together with the extracted feature maps and input labels, which returns the final detection output.

3.3.2 Backbone

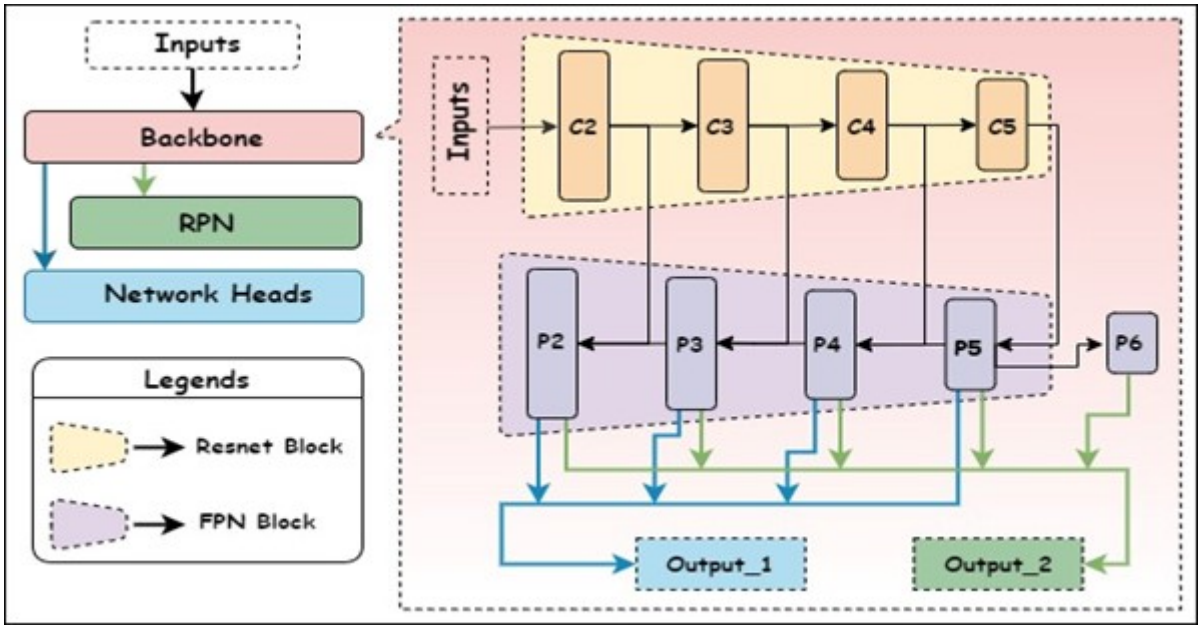

Figure 3.3: Backbone: ResNet-FPN

Mask Region-Based CNN (R-CNN) uses the ResNet-FPN backbone for feature extraction. It uses the standard ResNet architecture for image encoding. Fur layers of feature maps are extracted from ResNet50 architecture as shown in Figure3.3. There are several other backbones that are compatible with the Mask R-CNN architecture as shown in these studies [38, 39]. However, ResNet50 is the default backbone used with Mask R-CNN.

As mentioned before FPN uses a top-down path approach is used to generate the feature map. According to Bhatti et al. in [40], "FPN uses a top-down pathway to build higher resolution layers from a semantic rich layer and then we add lateral connections between reconstructed layers and the corresponding feature maps to help the detector to predict the location accurately" It starts with the smallest feature map and works progressively to larger feature maps by upscale operation. First, a 1×1 convolution filter is applied to reduce C5 channel depth to 256-d to create a P5 layer. This becomes the

first feature map layer used for object prediction. It is then added elementwise to the up-sampled output from the previous iteration. All outputs of this process go through 3 X 3 convolutional layers to produce the final 4 feature maps (P2, P3, P4, P5). A fifth feature map (P6) is generated from the maximum merge operation at P5.

3.3.3 Region Proposal Network

Ren et al introduced the RPN in [41]. They came up with the idea to eliminate the slow selective search algorithm and let a separate network predict the region proposals. It uses generated features from the backbone to generate regions of interest as proposals for the next stage. Each generated feature map from the backbone goes through a 3 X 3 convolutional layer and the results are passed into two branches, one for feature estimation and the other for bounding box regression as shown in Figure 3.4. Since there are already different size functions in the pyramid to handle different size objects, therefore, only one anchor stride is used for the function pyramid and 3 anchor ratios. Hence, there are 3 channels for objectness score and 3x4 channels for bounding box regression.

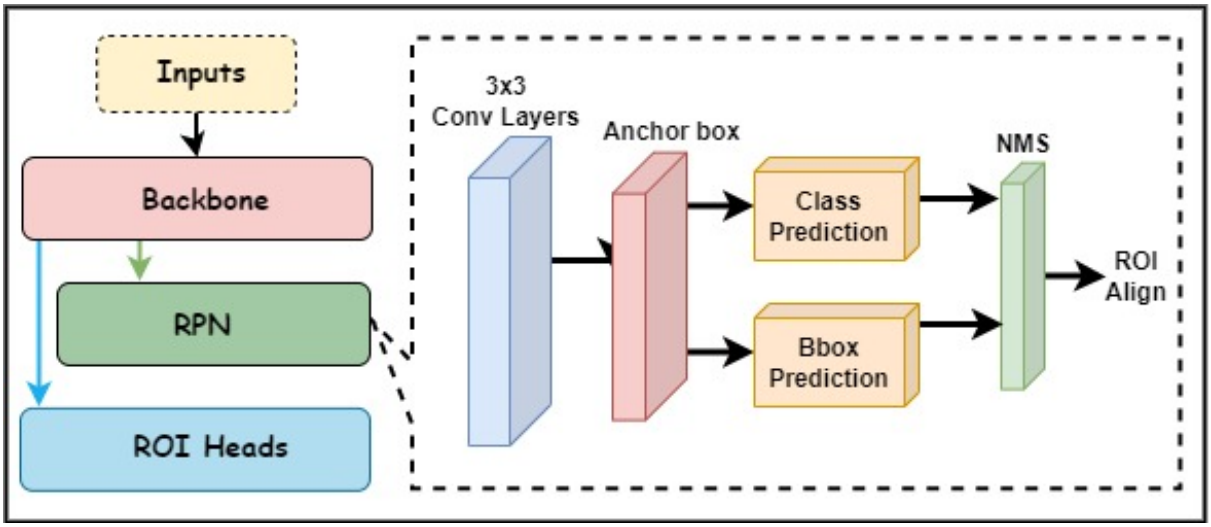

Figure 3.4: Region Proposal Network RPN

As for the Region proposal network class, the following important steps were taken to generate proposal losses and proposals:

1. Adds simple RPN Heads for classification and regression, which returns objectness as a classification score and pred-bbox-deltas as bounding box regression value.

2. Anchors are generated for a set of feature maps and image sizes. The module supports computing anchors at multiple sizes and aspect ratios per feature map. This module assumes aspect ratio = height/width for each anchor. The sizes and aspect ratios should have the same number of elements and correspond to the number of feature maps. The sizes and aspect ratios can have an arbitrary

number of elements, and the anchor generator will output a set of sizes * aspect ratios anchors per spatial location for the feature map.

3. The pred-bbox-deltas are applied to the anchors to obtain the decoded proposals.

4. Once the decoded proposals have been generated, they are filtered in several ways to ensure that the final output is accurate and useful. The first step in this process is selecting the top boxes independently per level. This means that the best proposals are chosen from each individual level, rather than considering all proposals across all levels at once. Next, non-maximum suppression (NMS) [42] is applied independently per level. This technique is used to eliminate any overlapping boxes, ensuring that only distinct and separate proposals are included in the final output. Finally, only the top-scoring predictions are kept, further reducing the number of proposals and increasing the overall accuracy of the output. After these filters have been applied, the final boxes and classification scores are returned.

5. However, during the training process, several separate steps are taken to ensure that the model is accurate and able to make accurate predictions. The first step is to assign targets to anchors in order to generate matched ground truth boxes and their labels. This allows the model to learn the relationship between the anchors and the actual objects in the image. Next, a set of proposals is encoded with respect to reference boxes. This step is used to create a standardized representation of the proposals that can be used by the model during training. Finally, the loss for objectness and boxes is computed. This step involves calculating the difference between the predicted and actual output and using this information to update the model's parameters and improve its accuracy. These steps are repeated for many iterations during the training process to make the model more accurate.

3.3.4 Region of Interest Heads

Region of Interest RoI Heads consists of two separate heads as shown in Figure 3.5: Box head and Mask head. While the box head is used to generate the final box and class prediction, whereas the mask head is used to generate the final mask prediction. The process of feature extraction of masks is similar to boxes, one key difference is the absence of fully connected layers that were present in the box head as reshaping before the fully connected layers loses the spatial structure information necessary to generate masks. Like they used RoI proposals as input for the box head feature extractor, they use the detected bounding boxes as input here. The FPN RoI mapping is similar to the box head again. The RoI align outputs are passed through a series of 3 X 3 convolutional layers, followed by ReLU, whose number of out channels of each layer and number of layers is a hyperparameter. For each detection, we get an output tensor of the same dimensions as RoI align output.

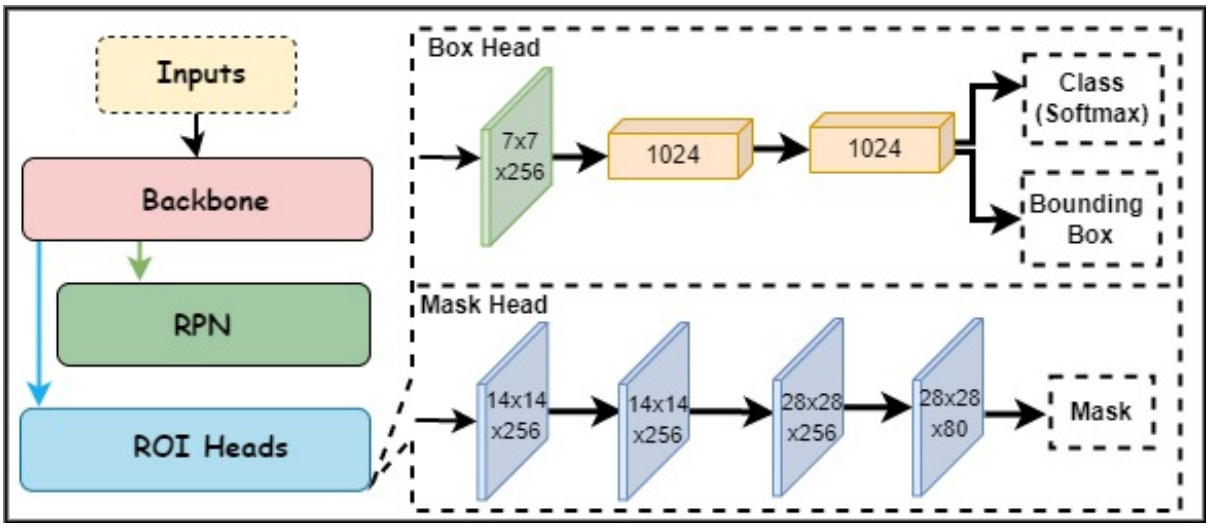

Figure 3.5: Region of Intrest Heads (RoI Heads)

Box Head

As stated before, the box head is used for object classification and object box regression tasks to predict the objectness score and box coordinate as shown in Figure 3.6. The following processes are performed in Box head:

1. FPN-RoI mapping: As the name suggests, the FPN-RoI mapping is used for mapping the associated appropriate feature map from RPN to a particular RoI based on its area. The Mask R-CNN algorithm employs the Resnet-50-FPN which generates 5 feature maps. All 5 of them are used in RPN to generate proposals, however, only P2, P3, P4, and P5 are used while associating with RoIs.

2. RoI Align: RoI Align is a replacement of the RoI Pool function in Faster NMS. The RoI Align layer removes the harsh quantization of the RoI Pool, by properly aligning the extracted features with the input.

3. Fully connected layers: Each RoI's output is altered such that it can be sent via a fully connected layer with a default representation size of 1024. This is transmitted to a second fully connected (FC) layer with the same number of input and output channels. We obtain distinct 1024-length vectors for each RoI. The RoI vectors are then processed through a predictor with two FC layer branches. One for classifying objects and the other for bounding box regression values. The classification of objects is complete at this stage but there are still numerous post-processing processes left for bounding box coordinate prediction.

Post-processing of Bounding Boxes: There are several post-processing steps involved in the process of detecting bounding boxes coordinate. The first step is the Pre-NMS threshold, where class probability scores are used as a threshold value to filter out detections before Non-Maximum Suppression (NMS) is applied. By default, the value for this threshold is set to 0.5. The next step is Decoding, where the

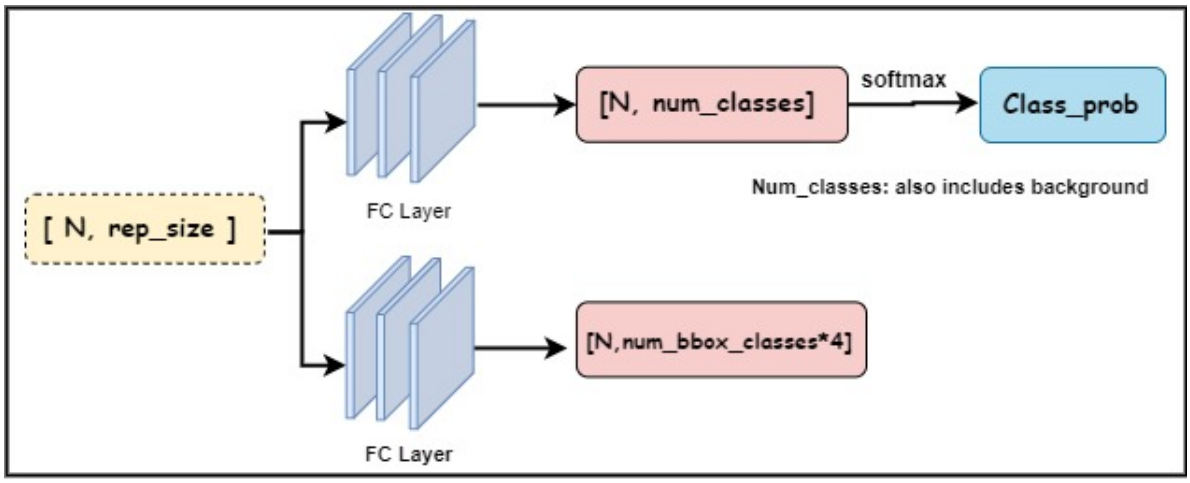

Figure 3.6: Box Head

encoded proposal is decoded for each class and proposal. This step is done separately for each class and proposal. After decoding, the next step is the Removal of invalid boxes, where any bounding boxes that have coordinates that lie outside of the image or have negative height and width are removed. Then, NMS is applied to remove boxes that have an overlap greater than the specified threshold. The default threshold value for NMS is set to 0.7. Finally, the last step is Filtering top detections, where the maximum number of detections for all classes in an image is specified. This ensures that there will never be more than this number of detections in an image.

Mask Head

The process of feature extraction of masks is like boxes, one key difference is the absence of fully connected layers as is visible in Figure 3.7. This is because the Mask R-CNN predicts an $n \times n$ mask from each region of interest using a fully-convolutional network (FCN). Since the same padding is used in FCN the output mask would have the same dimensions as the input, hence, there is no need for fully connected layers. The RoI align outputs are sent through a series of 3×3 convolutional layers, followed by ReLU.

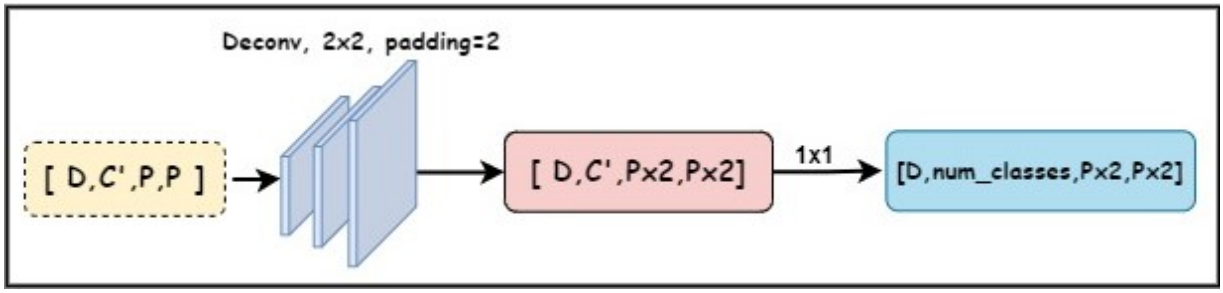

Figure 3.7: Mask Head

For the Resnet-50-FPN backbone, the number of out channels of each layer and the number of layers is set to [256,256,256,256]. The dimensions for the output tensor are the same as the RoI align output for each detection. Then, the extracted features are passed through the predictor, which involves a deconvolution

operation. The deconvoluted features were then subjected to a 1×1 convolutional layer, with the number of out channels being equal to the number of classes for each detection. Since the class prediction for each proposal is already done previously, therefore the output channel is selected according to the class and reduces the dimensions to [D, 1, 2P, 2P].

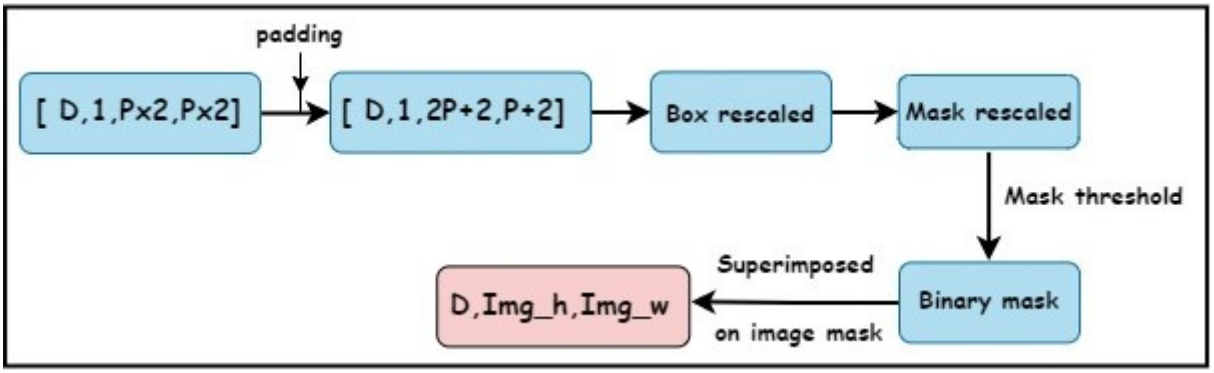

Figure 3.8: Post Processing of Masks

Post-processing of masks: As shown in Figure 3.8, the masks obtained are resized according to the input image. The mask tensors are padded to eliminate boundary effects produced by the up-sample operation we performed previously. Then the bounding box coordinates are re-scaled according to the new mask and converted to the nearest integer. The mask is also re-scaled according to the image size by using the bi-linear interpolation method. Finally, the rescaled mask is filtered using a mask threshold value with the default value of 0.5 to generate the final masks. This means that for each pixel, the object is assumed to be present if the value is greater than 0.5 and absent otherwise.

3.4 Scalable Multi-Task R-CNN System

As mentioned in section 3.2, the base network is designed as a two-stage framework. The first stage is responsible for generating object proposals, which are regions of the image that may contain objects of interest. This stage is a more generalized network that extracts features from the input image and proposes regions of interest as a shared layer. The second stage is a more class-specific network that takes these proposals as input and classifies them to generate bounding boxes and masks for the objects in the image.

The primary objective of our proposed system is to enhance multitask network performance by exploiting the distinctions between two stages. To achieve this objective, we propose the introduction of multiple RoI Heads, which are networks that are tailored to be task- or class-specific for an object or group of objects. These RoI Heads share the same Backbone and RPN networks, or solely the Backbone network, as shared layers across all tasks or groups of objects, as depicted in Figure 3.9. By following this approach, the system can acquire both general-purpose and task-specific representations, ultimately

leading to improved performance on a broad spectrum of tasks.

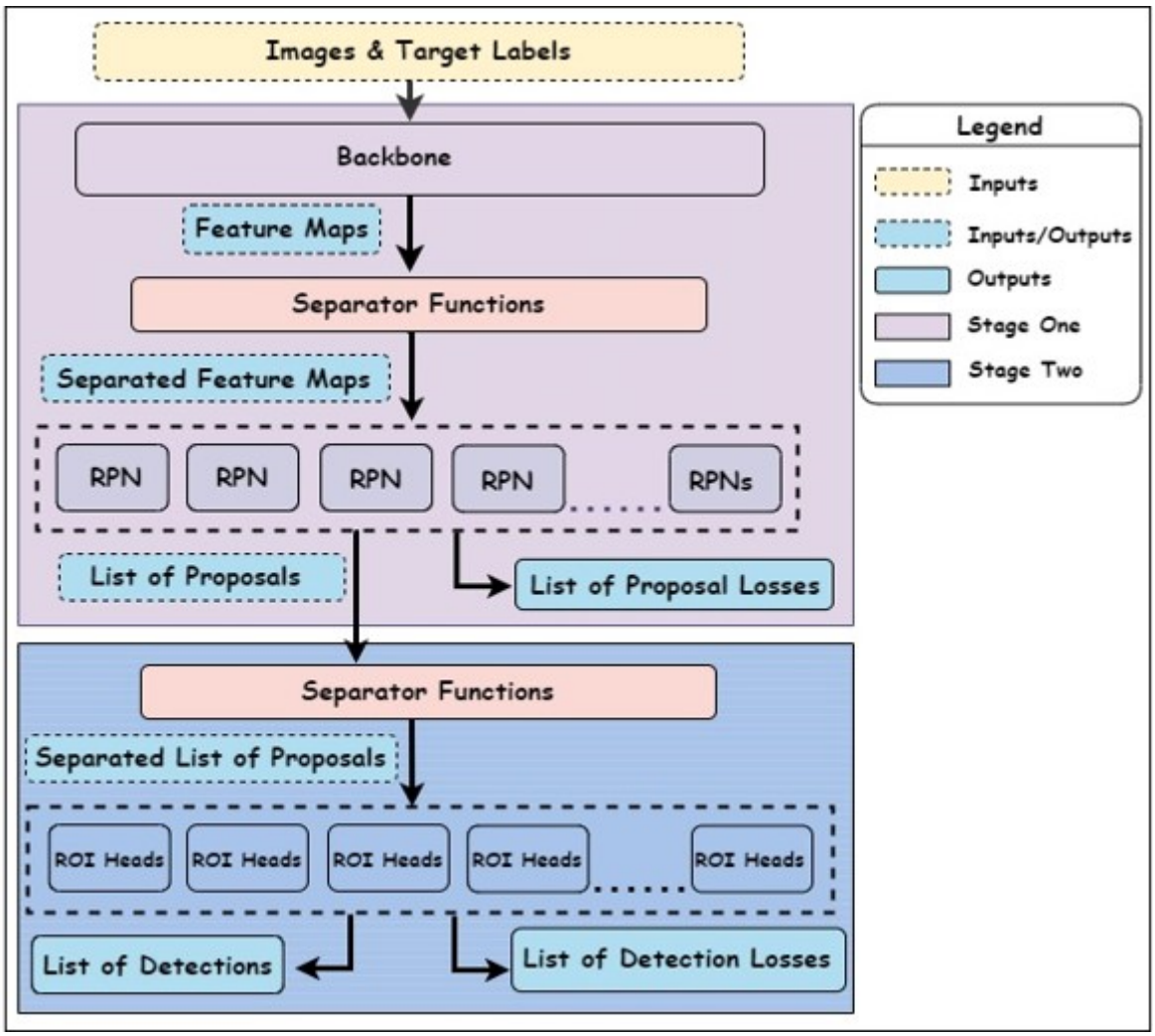

Figure 3.9: High-Level System Architecture Design of MTL R-CNN

To implement this approach, we use the hard parameter-sharing method like in [17, 18]. As mentioned in chapter 2, hard parameter sharing is a type of model architecture used in multitask learning where a single set of parameters is shared across all tasks. This means that the same weights and biases from the shared layers of the network are used for all tasks or groups of objects, resulting in a multi-task network that can perform multiple tasks or classify multiple groups of objects using a single set of parameters. This can lead to a more compact network with fewer parameters and faster training times, as the shared parameters can learn a general-purpose representation that is applicable to all tasks.

The aim of this thesis is to construct a scalable multi-task learning system by utilizing a shared generalized network with multiple class-specific networks using the base network. To expand the functionality of the proposed system for future work, we have incorporated additional features. For instance, we have included multiple regional proposal networks or RPN networks in the system design. Although the first stage of the system is a more generalized block, the RPN performs more class-specific tasks than the backbone network in the first stage. Hence, we propose a scalable multi-task region-based convolutional

neural network, abbreviated as MTL R-CNN, for object detection and instance segmentation. However, it is important to note that the focus of this thesis is on the object detection part of the proposed system.

3.4.1 System Implementation

As discussed in previous section, the proposed system is an enhancement over the base system with multiple RPNs and RoI Heads. We will refer to RoI Heads as a class-specific network head from here on class-specific network heads (CSN-Heads). Therefore, in order to integrate these functionalities, we developed special layers, called separator functions, to separate the respective features and labels. As shown in Figure 3.9, these separator functions are placed before the RPN and CSN-Heads networks to separate image, labels, and features maps before being fed into the respective networks. For example, in the first stage, the feature maps from the backbone, the target labels, and the input images are all separated before being fed into respective RPN networks. Likewise, in the second stage, the proposal from the RPN, the feature from the backbone, and the target labels are all separated being fed into respective CSN-Heads networks. Thus, due to these new functionalities, one can create various multi-task networks suited to their needs using our system. For example, for a facial feature detection like this [43], one can use our system to build a multi-task network with one RPN for whole face feature maps and three CSN-Heads branches for each respective face feature: eyes, nose, and mouth. Similarly, for full body detection, one can create a multi-task network with two RPNs and seven CSN-Heads branches, where the first RPN can be tasked to extract whole face features and fed the features to three CSN-Heads branches to predict the receptive facial features like the previous example, and the second RPN to extract feature of remaining body parts and fed it to the remaining CSN-Heads to do class-specific tasks.

3.4.2 Network Parameters

Similar to Mask R-CNN, in our proposed system, the parameters of the network model are stored in the weights and biases of the network layers. Each layer has its own set of weights and biases that are used to make predictions about the presence and location of objects in an image. The weights represent the strength of the connections between neurons in the current layer and the previous layer, while the biases represent the offset added to the input of each neuron in the current layer. Following are list of model parameters at each stage of our system:

Backbone Network: Our system uses ResNet50 as the backbone, and the ResNet50 backbone is composed of several layers, including 50 convolutional layers, and many residual blocks. The convolutional layers use a set of learnable kernels as weights to convolve the input feature map and produce an output feature map. In the residual blocks, which are a key component of ResNet architecture, the weights and biases are used to learn the residual mapping between the input and output of the block. A ResNet50

block can be represented by the equation (3.1) as suggested by He et al. in [44]:

$$y = F(x, \{W_i\}) + x \tag{3.1}$$

Where x and y are the input and output vectors, and the W_i is the weight, learnable filter kernel, of the layers considered. Finally, the function $F(x, \{W_i\})$ represents the residual mapping to be learned.

RPN Network: The RPN layer is typically composed of a set of fully convolutional layers and fully connected (FC) layers. The convolutional layers use a set of learnable kernels to convolve the input feature map and produce an output feature map. The FC layers are used to predict the scores and bounding box coordinates of the region proposals. The model parameters in the fully convolutional layer include the weights and biases; the weights are typically represented by a 4D tensor W with dimensions (filter_height, filter_width, input_channels, output_channels) and biases by a 1D tensor b with dimensions (output_channels). The model parameters in two FC layers can be categorized as classification parameters and regression parameters. According to Ming Zhou et al. [45], the classification parameters given as follows:

$$\begin{aligned} w_1 &= -\mathcal{E}(\rho w_1 + w_1^*) + w_1 \\ w_2 &= -\mathcal{E}(\rho w_1 + w_2^*) + w_2 \\ b_1 &= -\mathcal{E} b_1^* + b_1 \\ b_2 &= -\mathcal{E} b_1^* + b_2 \end{aligned} \tag{3.2}$$

Where $\mathcal{E}$ denotes the learning rate and ρ is the regularization penalty coefficient value. Additionally, w_1 represents the weight from the input to the hidden layer and w_2 represents the weight from the hidden layer to the output, whereas b_1 denotes the deviation from the input to the hidden layer and b_2 denotes the deviation from the hidden layer to the output. Finally, b_1^*, b_2^*, w_1^*, and w_2^* represent the gradient change of b_1, b_2, w_1, and w_2 respectively.

As for the bounding box regression, Kaiming He et al.[35] mentioned that they adopt the parameterization of the four coordinates as following:

$$\begin{aligned} t_x = \frac{(x - x_a)}{w_a}, t_y = \frac{(y - y_a)}{h_a}, \\ t_x^* = \frac{(x^* - x_a)}{w_a}, t_y^* = \frac{(y^* - y_a)}{h_a}, \\ t_w = \log(\frac{w}{w_a}), t_h = \log(\frac{h}{h_a}), \\ t_w^* = \log(\frac{w^*}{w_a}), t_h^* = \log(\frac{h^*}{h_a}) \end{aligned} \tag{3.3}$$

Where x, y, w, and h denote the box's center coordinates and its width and height as shown in

equation (3.3). Also, variables x, x_a, and x^* are for the predicted box, anchor box, and ground truth box respectively. The same holds true for the variables$y, y_a, y^*, h, h_a, h^*, w, w_a$, and w^*. Thus,t_x, t_y, t_w,and t_h are the four coordinates for predicted boxes and t_x^*, t_y^*, t_w^*, and t_h^* are the four coordinates for the ground truth boxes.

CSN-Heads Networks: CSN-Heads consist of two network heads namely, classification and regression heads, and mask heads. The "classification head" and "regression head" are used to generate the final object class and bounding box predictions, respectively. In the context of the "mask head", the weights and biases are used to generate the final mask predictions for each object in the image.

The classification head is a fully-connected layer that takes the proposed Region of Interest (RoI) from the RPN network as input and generates the class scores for each RoI. The class scores are then passed through a SoftMax activation function to get the final class probabilities. The Regression Head is also a fully connected layer that takes the proposed bounding box as input and generates the bounding box regression values for each RoI. The network model parameter for the classification and regression head is similar to that in equation 1 and equation 2 respectively.

The mask head is a fully-convolutional network FCN that takes the region of interest (RoI) features from the RoI pooling or align layer as input and generates a binary mask for each object instance in the image. The weights and biases in the mask head are the parameters of the FCN that generate the binary mask for each object instance. The weights are the values assigned to the filters in the convolutional layers of the FCN, and the biases are the values added to the output of each convolutional layer before passing it through the activation function. FCN of the mask head is defined as follows in TorchVision version 8.1:

$$O(N_i, C_o) = B(C_o) + \sum_{l=0}^{C_i - 1} X(C_o, l) \star I(N_i, l) \tag{3.4}$$

Where, in equation (3.4), I represents as input vectors with size $= (N, C_i, H_i W_i)$ or (C_i, H_i, W_i) and O denotes as output vectors with size $= (N, C_o, H_o, W_o)$ or(C_o, H_o, W_o). Additionally, star $\star$ is the 2D cross-correlation operator,N is the batch size, C denotes the number of channels, H is the height of input planes in pixels, and W is the width in pixels. Finally, X denotes the weights, and the weights in the FCN layers are given as out_channels, in-channelsgroups, kernel_size[0], and kernel_size[1]; where out_channels are the output of each proposed RoIs from RoI-Align layer, as for in_channels/groups, groups is always equal to 1 by default and in_channels is set to (256, 256, 256, 256) as mask layers in [35]. Finally, kernel_size[0] and kernel_size[1] are simply the length and breadth of the size of the kernel or filter, and the size is defined as 3x3. Last but not the least, B denotes the biases and the biases in the FCN layers given as out_channels if the biases option is set to True, where the out_channels represent the output from previous layers as suggested before in the weights section.

3.4.3 Model Parameters Assignments

Given that the proposed system is a scalable multitask learning network that provides the flexibility to create multiple RPN or multiple CSN-Heads layers, we have devised a novel algorithm, as presented in Algorithm 1, to assign the model parameters based on the desired network structure. The algorithm takes the following steps to assign the model parameters according to the network structure:

Algorithm 1 Model Parameter Separator for each Task

Input: *model*
Output: $ListofPara$
1: $BackbonePara \leftarrow model$ ▷ Extracts Backbone parameters from the model
2: **for** `<NumofRPNs>` **do**
3: $RPNPara \leftarrow model$ ▷ Extracts RPN parameters for each task
4: **end for**
5: **for** `<NumofROIs>` **do**
6: $ROIPara \leftarrow model$ ▷ Extracts ROI parameters for each task
7: $ListofPara \leftarrow BackbonePara + RPNPara + ROIPara$ ▷ Concatenate the shared layers-
8: **end for** ▷ -parameters with task-specific parameters into ListofPara

First, the algorithm extracts the parameters of the various subnetworks that make up the MTL R-CNN model. These subnetworks include the backbone network, which is responsible for extracting high-level features from the input image, and the region proposal network RPN, which is responsible for generating a set of region proposals for object detection. Additionally, there may be one or more class-specific networks (CSN-Heads), which are task-specific networks that perform object classification and bounding box regression for each class of interest.

Second, it rearranges the extracted parameters according to the structure of the MTL R-CNN network. In an MTL network, some of the parameters are shared across tasks, while others are specific to each task. By rearranging the parameters according to the network structure, it is possible to train the model to perform multiple tasks simultaneously, while sharing the parameters of the shared layers across tasks and adjusting the parameters of the task-specific layers for each task.

For example, consider an MTL R-CNN network with a backbone and an RPN as shared layers, and two CSN-Heads as task-specific layers for each task. Then, the algorithm will return two sets of parameters, designated for each task, as follows:

- Task 1 Parameter set = Backbone parameters + RPN_1 parameters + CSN-Heads_1 parameters
- Task 2 Parameter set = Backbone parameters + RPN_2 parameters + CSN-Heads_2 parameters

During training, the parameters in the shared layers (i.e., the backbone and the RPN in this example) are adjusted based on the loss computed over all samples, regardless of the sample type. The parameters in the task-specific layers (i.e., the CSN-Heads in this example) are adjusted only when the task-specific sample is learned, based on the loss computed over the samples of that specific task.

This approach allows our MTL model to learn multiple tasks simultaneously while sharing the parameters of the shared layers across tasks and adjusting the parameters of the task-specific layers for each task.

3.4.4 Loss Functions

In Mask R-CNN, the loss function is typically a combination of different losses, including a classification loss, a bounding box regression loss, and a mask loss. Thus, our proposed system also uses these losses to measure the error between the predicted output of the model and the ground truth labels.

The classification loss measures the error between the predicted class probabilities and the ground truth labels. The class probabilities are the output of the model's classification head, which is a fully connected layer that takes the feature maps from the convolutional layers as input and outputs a probability for each class. The classification loss L_c is typically calculated using the cross-entropy loss function and it is defined by the following equation (3.5):

$$L_c = \frac{1}{N_c} \sum_i L_x(P_i, P_i^*) \tag{3.5}$$

where P_i is the predicted probability of anchor i being an object, P_i^* is the ground truth label of whether anchor i is an object, N_c is the Normalization term, set to be mini-batch size (256) in the paper [35], and L_x is the log loss function over two classes and it is defined by following equation(3.6):

$$L_x(P_i, P_i^*) = -P_i^* \log P_i - (1 - P_i^*) \log(1 - P_i) \tag{3.6}$$

This loss function penalizes the model for making confident predictions that are incorrect, and it encourages the model to output probabilities that are close to the ground truth labels. The cross-entropy loss can be thought of as a measure of the distance between the predicted and ground truth probability distributions.

The bounding box regression loss measures the error between the predicted bounding box coordinates and the ground truth bounding boxes. The bounding box coordinates are the output of the model's bounding box regression head, which is a fully connected layer that takes the feature maps from the convolutional layers as input and outputs the coordinates of the bounding box for each class.

There are several loss functions that can be used for bounding box regression, including the smooth $L1$ loss and the mean squared error (MSE) loss. The smooth $L1$ loss is commonly used in object detection models because it is less sensitive to outliers than the MSE loss. It is calculated as follows in the equation (3.7):

$$L_b = \frac{\theta}{N_b} \sum_{i \in \{x,y,w,h\}} L1(T_i - T_i^*) \tag{3.7}$$

where T_i is predicted four parameterized coordinates, T_i^* is ground truth coordinates, N_b is a normalization term, set to the number of anchor locations (2400), and θ is balancing parameter, set to be 10 in [41] so that both L_c and L_b terms are roughly equally weighted. The smooth L1 loss $L1$ is defined as shown in the equation (3.8):

$$L1 = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases} \tag{3.8}$$

This loss function penalizes the model for making large errors in the bounding box coordinates, and encourages the model to output coordinates that are close to the ground truth values. The smooth L1 loss can be thought of as a smooth approximation to the L1 loss, which is the absolute value of the error.

The mask loss measures the error between the predicted masks and the ground truth masks. The masks are binary maps that indicate the pixels that belong to each object instance in the image. The mask loss is typically calculated using the binary cross-entropy loss function in conjunction with the sigmoid activation function.

The Binary cross-entropy loss is a loss function that is commonly used for classification tasks with binary labels (i.e., labels that are either 0 or 1). It is often used in conjunction with the sigmoid activation function, which maps input values to the range of 0 to 1. A cross-entropy loss is defined as follows in equation (3.9):

$$L = -\sum_{i=1}^{n}(Y_i \times \log(P_i) + (1 - Y_i) \times \log(1 - P_i)) \tag{3.9}$$

where Y_i is the ground truth label, and P_i is the predicted probability. The loss function penalizes the model for making confident predictions that are incorrect, and it encourages the model to output probabilities that are close to the ground truth labels. Since the Binary cross-entropy loss is often calculated as the average of cross-entropy across all the data samples, thus, the mask loss in Mask R-CNN is defined as follows in the equation (3.10). It is confirmed by Mahmoud et al. in their paper [46] and they define Mask lost, L_m, as:

$$L_m = -\frac{1}{X^2}\sum_{m=1}^{X} Y_m \times \log(P_m) + (1 - Y_m) \times \log(1 - P_m))) \tag{3.10}$$

where Y_m is the ground truth mask value for pixel m, and P_m is the predicted mask value for pixel m (a value between 0 and 1). The sum is taken over all pixels in the mask and the sum is averaged using the size of the true mask region, $X \times X = X^2$.

This loss function penalizes the model for making confident predictions that are incorrect, and it encourages the model to output mask values that are close to the ground truth masks. The binary cross-entropy loss can be thought of as a measure of the distance between the predicted and ground truth mask distributions.

The overall loss function for our proposed system model is typically a sum of these individual losses, for example, the loss function may be defined as in equation (3.11):

$$L_t = L_c + L_b + L_m \tag{3.11}$$

Where L_c is the classification loss, L_b represents box regression loss, L_m represents mask loss, finally L_t represents total loss of a task or a branch.

During training, the model is optimized to minimize the overall loss by adjusting the model's parameters. The weights of the model are updated using an optimization algorithm, such as stochastic gradient descent SGD or Adam. The optimization process continues until the loss reaches a satisfactory level or a predefined number of training iterations has been reached.

3.4.5 Separator Functions

As stated before, we developed special separator functions to facilitate the need to separate the inputs for RPN and CSN-Heads branches. On top of that, we created several other separator functions for model evaluation and loss function parameter separation. The following are the characteristics of each special separator function:

1. gnr_train_separator: The gnr_train_separator is one of most important functions created in generalized_rcnn class. It uses a separation map defined in a configuration file to separate target labels and returns a list of separated target labels and a new set of separation maps called f_map. Since these functions are applied twice, before and after the RPN network, it takes an extra attribute called "pre" to define whether the separation is happening "pre" or "post" RPN network. Note that the initial separation map from the configuration file is also different for pre and post-RPN networks. There are two additional internal functions created in this function. First, a function gnr_eval_separator is created to assign the target value to "None" so that gnr_train_separator would be able to generate a new separation map with the target value being None during the evaluation stage. This is because during the evaluation only the image is used inside the network to check the accuracy of a model. Second, a function pre_RPN is created to process the pre and post-RPN tasks separately.

2. gnr_feature_separator: This function is responsible for separating the extracted feature set from the backbone network. It separates the feature set using the new separation map, f map, generated by gnr_train_separator. It is utilized both before and after the RPN Network operation.

3. gnr_image_separator: Similar to the previous function, this one is responsible for splitting the input images. It separates the images using the newly formed separation map, f map, which was produced

by gnr_train_separator. Additionally, it is utilized prior to and after the RPN Network operation as well.

4. gnr_proposal_separator: This function separates RPN-generated proposals. Using the new separation map, f map, generated by gnr_train_separator, it separates the proposals, like the previous two functions. However, it is utilized only once the RPN network operation has been completed.

5. eval_separator and coco_eval_separator: These two functions were designed for the purpose of evaluation. Both parties are accountable for the separation of target labels. These target labels were separated using a separation map specified in configuration files. The eval_separator is utilized in the engine file whereas the coco_eval_separator is utilized in the coco_eval file.

6. para_separator: This function is responsible for creating respective loss functions for each branch. It does so by separating each parameter individually and rearranging the parameters according to the network branch designs.

7. Data loader preparations: All PyTorch-based computer vision jobs need to use the Data loader class so that a dataset can be combined and an iterable over the given dataset can be generated. Therefore, a little preprocessing of the dataset is required before it is supplied to the data loader. For the data preparation, we utilized all the standard methods, and in addition, we added an additional strategy by appending an annotation to the dataset with the name "branch" in addition to the labels that were given. This branch label is utilized throughout the various stages of the network to serve the goal of separating the data.

3.4.6 Configuration Files

This section discusses the description and formats of all parameters in the configuration file that facilitate the scalable and multi-task nature of the proposed system. One can simply use this configuration file to design and evaluate any type of multi-task network by using the system. It is like a one-stop shop, where one can find the parameters for a network definition or the hyperparameters for a network training. The details for each parameter in the configuration file are given as follows:

- n_branch: The parameter "n_branch" is used for network definition and represents the total number of branches to be generated, where each branch denotes the number of Region Proposal Network (RPN). The parameter's value must be an integer and equal to or greater than 1. For example, assigning n_branch = 2 would instantiate two RPN networks.

- s_branch: The parameter "s_branch" is also used for network definition and represents the total number of sub-branches per branch or RPN. This parameter enables one to define the total number of CSN_Heads per RPN. The value of this parameter cannot be zero, and it must be in a square

bracket. If multiple branches exist, assign the sub-branches appropriately. For instance, when s_branch is assigned as [1, 2], it indicates that there are three total sub-branches or CSN_Heads. The first sub-branch is assigned to the first branch, and the remaining two sub-branches are assigned to the second branch.

- class_cats: The parameter "class_cats" is used for network training purposes and is utilized for defining class categories. The classes are defined within three recursive square brackets, where the classes in the same second square bracket are grouped together in one branch and the classes defined in the same third square bracket are grouped together in one sub-branch. For instance, class_cats = [[["bike", "person"]], [["t_light", "t_sign"], ["truck", "train", "car"]]], where "bike" and "person" are defined in one group and the remaining classes are defined in another group. In the second group, the classes are further sub-grouped into two groups.

- class_ids: The parameter "class_ids" represents the actual class IDs per sub-branches and is used for network training purposes. This parameter value is utilized in assigning new class category IDs during branch separation. For example, class_ids = [3, 5, 8] implies that the class IDs from 1 to 3 are in the first sub-branch, 4 to 5 are in the second sub-branch, and 6 to 8 are in the third sub-branch.

- num_class: The parameter "num_class" is used in defining the number of classes in the respective CSN_Heads. This parameter is used during network training, and its value must be 1 more than the desired number of classes. This is because the algorithm assigns an extra value for the background class. The parameter value must be assigned in the format num_class = [6, 3, 7], where in this example, there are three CSN_Heads, and their respective numbers of classes per CSN_Heads are defined in the square bracket.

- num_epoch: The parameter "num_epoch" is used to initialize the number of epochs during training, enabling the model to run for the specified number of epochs. It can be assigned as num_epoch = 20, indicating that the model will be trained for 20 epochs.

- num_roi: The parameter "num_roi" serves to specify the number of branches to be instantiated for the CSN-Heads class during network training. This variable is analogous to "s_branch," but it is used specifically for the instantiation of the CSN-Heads class. To assign a value to "num_roi," the following format must be utilized: num_roi = 3. This signifies that the model will instantiate three CSN-Heads classes.

- num_rpn: Similarly, the parameter "num_rpn" is used to determine the number of RPNs to be created during network training, analogous to "n_branch." This variable is employed during the

instantiation of the RPN class. To assign a value to "num_rpn," the following format must be utilized: num_rpn = 1. This indicates that the model will instantiate one RPN class.

- Rpns: The parameter "Rpns" is used for features assignment per RPN during the network training. The value of the parameter is assigned within a round bracket with two terms, where the first term of the variable represents the current RPN, and the second term defines which features must be assigned to the current RPN. Here is an example of how this parameter value is assigned: Rpns = [('rpn0', 0)], This shows that the first set of features will be assigned to the first RPN branch. Note that branch numbering starts from 0 because the 0 is first index value.

CHAPTER 4

Experimentation using MTL R-CNN

4.1 Introduction

In this chapter, we present a comprehensive discussion of the experimental procedures conducted using the proposed MTL R-CNN system introduced in Chapter 3. The primary focus of the chapter is the dataset used for the investigations. Since the dataset required minimal preprocessing, we provide an overview of the measures taken to clean it. Subsequently, we delve into the intricacies of each experiment, including the model training, evaluation, and construction. Finally, we assess the performance of our top-performing model in comparison to other relevant studies.

4.2 BDD100K Dataset

We evaluate our proposed method on the Berkeley DeepDrive 100k (BDD100K) Dataset [47] developed by Fisher Yu et al. BDD100K is a large-scale dataset that has been widely used for training and evaluating self-driving car algorithms. It was created by the UC Berkeley DeepDrive project and contains over 100,000 videos recorded from self-driving cars in a variety of real-world scenarios. The videos are annotated with a variety of labels, including object bounding boxes and semantic labels for different types of objects, such as pedestrians, vehicles, traffic signs, and lane markings. The dataset also includes additional annotations, such as lane markings, drivable areas, and traffic lights.

BDD100K is intended to be a benchmark for the development and evaluation of algorithms for tasks such as object detection, tracking, and scene understanding in autonomous driving. To achieve this goal,

the dataset is designed to be diverse and challenging, with a wide range of scenarios including diverse weather conditions, different times of day, and various traffic and pedestrian densities. The dataset has been widely used in academic and industry research to evaluate the performance of various algorithms for self-driving car applications and has played a significant role in advancing the state of the art in these areas.

BDD100K is comprised of ten computer vision tasks: image tagging, lane detection, drivable area segmentation, road object detection, semantic segmentation, instance segmentation, multi-object detection tracking, multi-object segmentation tracking, domain adaptation, and imitation learning. Since our research is about object detection, we will only focus on the road object detection task of the BDD100K dataset.

For the road object detection tasks, BDD100K has a total of 10 objects or classes, namely: car, traffic sign, traffic light, pedestrian, truck, bus, bicycle, rider, motorbike, and train. To comprehend the distribution of the objects and their placements, they identify the object bounding boxes for common objects that appear on the road overall 100,000 keyframes. Figure 4.1 illustrates the item counts for each object, the varied range of objects that are included in the dataset, as well as the scope of the dataset, which includes more than one million different types of cars.

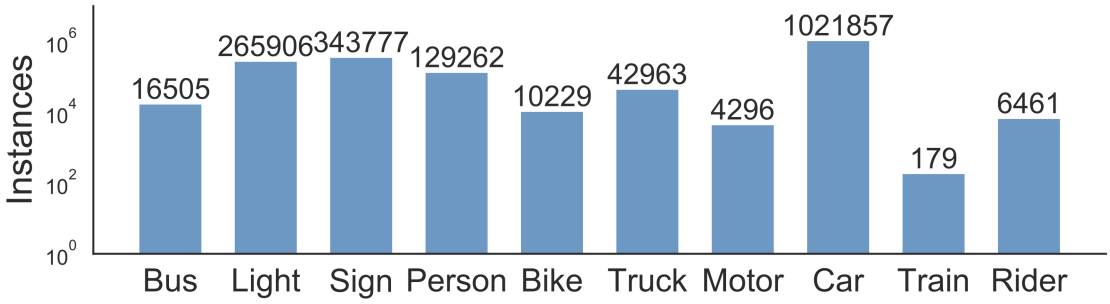

Figure 4.1: Class Instances Distribution in BDD100K Dataset

The reason why BDD100K is used to evaluate our networks is that, as stated before it is a high-quality and challenging dataset, with a wide range of scenarios including diverse weather conditions, different times of day, and various traffic and pedestrian densities. These characteristics make BDD100K well-suited for testing the robustness and generalization of self-driving car algorithms.

There are several other self-driving car datasets that have been developed and used for similar purposes, such as the KITTI dataset and the Waymo Open Dataset. These datasets also contain a variety of real-world scenarios and have been widely used in academic and industry research. However, BDD100K is unique in its scale, with over 100,000 videos and 8 million annotated frames, and in its annotation quality and diversity, which includes a wide range of objects and features relevant to self-driving car applications.

In terms of its performance as a benchmark for self-driving car algorithms, BDD100K has been shown

to be competitive with other datasets in terms of the quality of its annotations and the diversity of its scenarios. It has also been widely used in academic and industry research, with a large number of papers and articles reporting results on the BDD100K dataset [47].

BDD100K is one of the largest and most diverse self-driving car datasets available, which makes it well-suited for evaluating algorithms that are intended to perform well in a wide range of real-world scenarios. However, it is important to note that no single dataset is perfect and that different datasets may be more or less suitable for different tasks or applications. For example, the KITTI dataset is known for its high-quality annotations and has been widely used for object detection and 3D scene understanding tasks. The Waymo Open Dataset, on the other hand, is notable for its large scale and has been used for a variety of tasks related to autonomous driving, including object detection, tracking, and prediction.

Overall, it is important to consider the specific characteristics and limitations of each dataset when using them for research or development purposes, and to choose the dataset that is most appropriate for the task at hand. As for our thesis work, BDD100K serves as the perfect dataset for evaluating our proposed networks because of all the above reasons.

4.3 Data Preprocessing

Prior to utilizing the BDD100K dataset for the training or evaluation, it is essential to undertake a series of data preprocessing steps in order to prepare the data for use with the intended machine learning model. The thesis carried out several experiments, and in all cases, three key data preprocessing steps were implemented.

Firstly, as is typical in large datasets, there were a number of data images without corresponding labels. These images were eliminated. Secondly, new data labels that were not specified in the BDD100K manual were identified. These new labels were also removed, as their sample size was deemed insufficient to generate accurate results.

Lastly, it was observed that the sample sizes of the various object classes were vastly different, as demonstrated in the aforementioned Figure 4.1. For instance, if we compare the sample size of the object "car" to that of the object "train", the sample size of the "train" object was found to be exponentially smaller. In fact, the sample size of the object "train" is just 179 instances and the object "car is in millions of instances. Therefore, it is understandable that training a model using only 179 instances of object "train" with other instances of objects would result in bad accuracy of the overall network. As a result, the "train" object was excluded from all experiments.

4.4 Hardware and Software Specifications

All our experiments were conducted on a computer system that was equipped with an NVIDIA RTX A4000 graphics processing unit (GPU) with a memory capacity of 16 GB and a central processing unit (CPU) RAM of 44 GB. The operating system of the computer system was Ubuntu 16.04. In terms of software specifications, we utilized Anaconda 4.11 to establish a software environment. This environment included the following software components: Python 3.7, Pytorch 1.7, Torchvision 0.8.2, and OpenCV 3.4.2. These software components were chosen based on their compatibility with the tasks at hand and their ability to effectively support the implementation of the machine learning models. The use of a specific version of the software was chosen for stability and compatibility reasons.

4.5 Evaluation *Matrices* Metrics

In order to thoroughly evaluate the performance of the models presented in this thesis, a comprehensive evaluation strategy was employed utilizing various metrics. Specifically, the following metrics were used to quantitatively measure the performance of the models: Intersection over Union IoU, Average Precision (AP) and Mean Average Precision mAP.

Intersection over Union (IoU) is a commonly used evaluation metric for object detection tasks. It is a measure of similarity between the predicted bounding box and the ground truth bounding box. According to Hou et al. in [48], IoU is calculated as the ratio of the intersection of the predicted and ground truth boxes to their union. The formula for IoU is given as follows in equation(4.1):

$$IoU = \frac{Areaofintersection}{AreaofUnion} \tag{4.1}$$

Where the area of intersection is the area of the region where the predicted bounding box and the ground truth bounding box overlap, and the area of union is the total area of both bounding boxes combined.

The **Average Precision** (AP) is defined as the area under the precision-recall curve. According to Katirci et al. in [49], the precision-recall curve is a graph that plots the precision of the model against its recall. Precision is the ratio of true positive detections to the total number of positive detections, and recall is the ratio of true positive detections to the total number of actual positive instances.

$$\begin{aligned} AP &= \int P(x)d(R(x)), \\ P(x) &= \frac{TP}{TP+FP}, \\ R(x) &= \frac{TP}{TP+FN}, \end{aligned} \tag{4.2}$$

Where,$P(x)$ represents the function of precision and $R(x)$ represents the function of recall. Additionally,

TP stands for true positive, FP stands for false positive, and FN stans for false negative. Therefore, the functions$P(x)$ and $R(x)$ can be given as it is shown in the equation (4.2).

Mean Average Precision (mAP) is a measure of the performance of the model across all classes and at differentIoUthresholds. It is calculated by taking the mean of the Average Precision (AP) at different IoUthresholds (usually 0.5, 0.55, 0.6, 0.65, 0.7, 0.75) over all classes. mAP can be calculated by averaging over all the classes as shown in the following equation (4.3):

$$mAP = \frac{1}{N} \times \sum(AP, class_i) \tag{4.3}$$

Where N is the total number of classes and $\sum$ is the summation of all the classes. In this thesis, we compared most of our results using AP50 andAP[50:95] values.

4.6 Base Network Training and Result

Prior to evaluating our proposed methods, we initially assessed the efficacy of the base network, Mask R-CNN, on the BDD100K dataset. Figure 4.1 provides a visual representation of the dataset's extensive scope, encompassing a diverse assortment of objects, and the associated item counts for each object. As outlined in section 4.2, the BDD100K dataset comprises 70,000 training samples and 20,000 validation sample images, which were subsequently partitioned to generate 10,000 validation images and 10,000 test samples. However, following data preprocessing, the sample sizes were reduced to 54,000 training samples, 4,900 test samples, and 4,900 validation samples.

The first model was trained using a Stochastic Gradient Descent (SGD) optimizer with a momentum value of 0.9 and weight decay of 0.00001. An input batch size of 4 was used, and the model was initially trained with a learning rate of 0.01, which was gradually decreased with a gamma value of 0.1 on every third epoch. The training process took approximately 4 days to complete 20 epochs, although the model converged after only 11 epochs with a training loss of 0.72 and an AP@50 value of 53.3% on the validation sample. Several other models were built using different optimizers, such as Adam, but none of them performed better than the first model trained with the SGD optimizer. Therefore, the hyperparameters of the first model were further tuned to produce a better base network model. The same SGD parameters were used for tuning, but this time, the initial learning rate was set to 0.001, and the learning rate was decreased with a gamma value of 0.1 every 3 epochs until 20 epochs. Similar to the first model, the newly tuned model converged early, after the 11th epoch, with a training loss of 0.60% on the training sample and an AP@50 value of 57.9% on the validation sample.

4.7 Proposed Networks Designs using MTL R-CNN

As discussed in Chapter 3, our proposed system is a scalable multi-tasking system that allows us to create multiple networks at different scales for the same computer vision task. For multi-task object detection learning, at most, a network could have the same number of class-specific network heads CSN-Heads as a number of objects or tasks. Since the main idea for using multi-task learning is to share the learning among the tasks by having shared representation to reduce the computational cost. Therefore, having fewer CSN-Heads branches by grouping similar objects or learning correlated tasks together would be a more efficient multi-task learning model. Thus, the main objective behind all the experiments done in this thesis is to create the most efficient object detection multi-task learning network using the BDD100K dataset.

One of the key challenges of multi-task learning is how to group the tasks and the corresponding objects in order to optimize the performance of the system. One factor to consider when grouping objects in a multi-task learning system is the similarity of their visual features. If the objects for different tasks have significantly different visual features, this can affect the ability of the system to learn each task effectively. For example, if one task involves objects with similar visual features, such as color and texture, and another task involves objects with more diverse visual features, the system may have an easier time learning the task with the more similar visual features.

In addition to visual features, the size of the object sample for each task can also have an impact on the performance of the system. If one task has a much larger object sample than another task, the system may be able to learn the task with the larger object sample more effectively due to the increased amount of training data.

To address the above mentioned issues, one approach is to balance the sample sizes of the objects for the different tasks. This can help to ensure that the system has a sufficient amount of diversity in the training data for each task, as well as a sufficient amount of data overall, which can improve the overall performance of the system.

Another approach is to use feature-based weighting, where the loss function for each task is weighted based on the visual features and sample size of the objects. This can help to ensure that the system is able to learn all tasks effectively, even if the visual features and sample sizes of the objects are significantly different.

Overall, it is important to carefully consider the visual features and sample sizes of the objects when grouping tasks in a multi-task learning system, as these factors can have a significant impact on the performance of the system.

Therefore, we used the above two main principles to group the objects or the tasks to learn together. First, the objects or tasks with similar visual features are grouped together. For example, in our dataset, the object car, bus, and truck share similar visual features like four wheels, four doors, and multiple

lights. Similarly, the object bike and bicycle share similar visual features like two wheels and one handle. We utilize the visual features of the objects to determine which ones belong together in a group because there is a lot of evidence in [50, 51] that shows that the objects that share visual characteristics produce better results when they are learned together.

Second, the objects or tasks are grouped together if their sample sizes lie within the same range, where the range is determined by taking the rounded-down common log (RDCL) of sample size and using it as a threshold value. For example, objects traffic signs, traffic lights, and persons can be learned together as their sample size RDCL value is the same. The application of RDCL value will be elaborated on later in section 4.5.2. The main reason for using the sample size to group the objects or tasks into a cluster is to mitigate the class imbalance issue. As described in [52, 53] studies, when training any model with multi-object deep learning tasks, imbalanced data samples between the objects result in subpar performance. Thus, our assumption is that if objects with comparable sample sizes are grouped together, there will be no class imbalance problems. These two principles allowed us to create three distinct types of networks.

- Type 1 (T1): in T1 networks the tasks or the objects with similar visual features are grouped together.
- Type 2 (T2): in T2 networks the tasks or the objects that fall within the same range of sample size are grouped together.
- Type 3 (T3): in T3 networks the tasks or objects are grouped together by considering both principles at once as a hybrid branch.

4.7.1 T1 Networks Design:

In T1 networks, the objects with similar features are learned or grouped together where each group is assigned to a single CSN-Heads branch. The type 1 network would have the same first stage as the base Mask R-CNN architecture. As for the second stage, the network would have multiple CSN-Heads branches where each branch is assigned to an object or a group of objects with similar feature sets.

Network Settings: As mentioned in section 4.2, the BDD100K dataset contains nine class objects: traffic light, traffic sign, rider, pedestrian, bicycle, motorcycle, truck, bus, and car. In this section, we experimented with several T1 networks to find the most efficient way to group the given objects with similar feature sets. Following is the network description of the two of the best T1 networks:

A) A T1 network named T1A has been devised, incorporating four CSN-Heads branches with specific assignments. The first branch is designated to predict the traffic light and traffic sign in combination, the second branch to forecast the rider and pedestrian jointly, the third branch to predict the bicycle and motorcycle concurrently, and the fourth branch to anticipate the truck, bus, and car simultaneously, as illustrated in Figure 4.2. It is obvious why the objects are grouped together in each of the last three

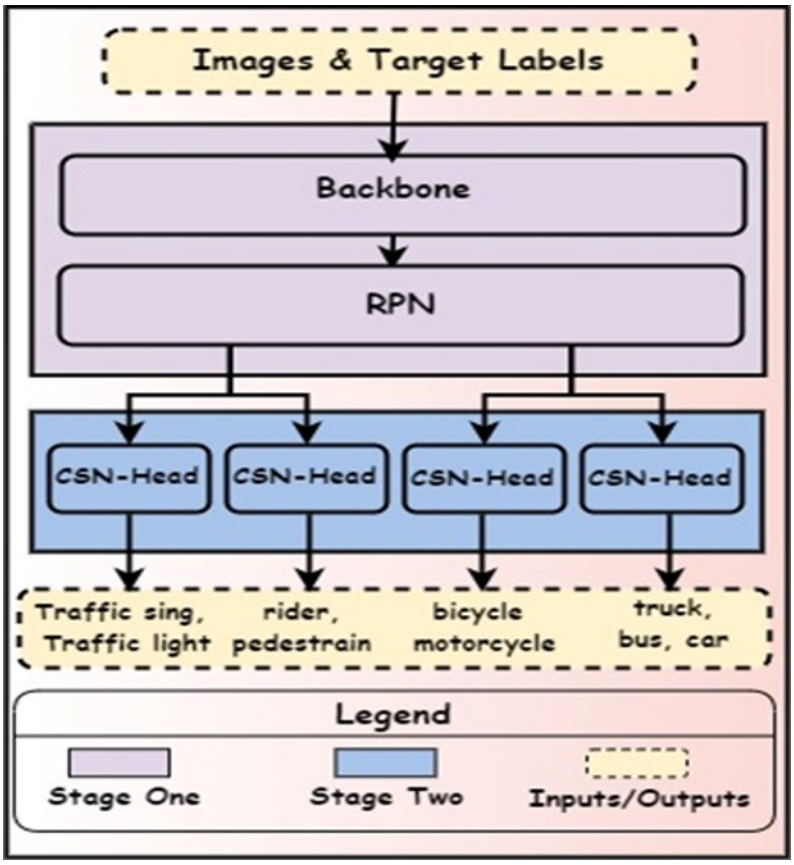

Figure 4.2: T1A Network Design

branches as the objects are kind of the same besides the little distinctions. For instance, in the third CSN-Heads branch, both the bicycle and motorcycle are the same types of objects as both are two-wheeler vehicles. Hence, the objects share lots of features like both having two wheels, two handles, etc. However, the objects in the first branch do not appear to be the same or alike, but these objects do share features in terms of their geometrical shapes. For example, both the traffic light and traffic sign are made from similar geometrical shapes like circles, rectangular, etc.

B) A further T1 network, denoted as T1B, was built, incorporating three CSN-Heads branches. These branches include the first CSN-Heads branch is tasked to predict the traffic light and traffic sign together, the second to predict the rider, pedestrian, bicycle, and motorcycle together, and the third to predict the truck, bus, and car together as shown in Figure 4.3. The idea of having the objects grouped together in each of the first and third branches is the same as it is in the T1A network. However, in the second branch, the rider and pedestrian are grouped together with the bicycle and the motorcycle because the object rider is seen to be riding on the object motorcycle often in the dataset, and the bounding boxes of both objects seem to partially overlap each other. Hence, it is safe to say that all the class objects in the second branch share their features.

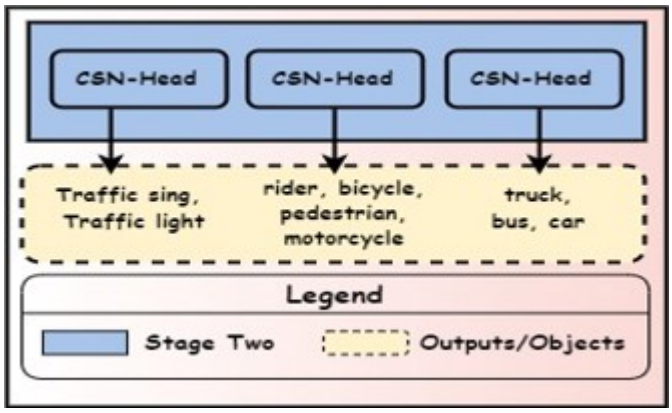

Figure 4.3: T1B Network Design

4.7.2 T2 Networks Design:

As stated previously, the tasks or objects that fall within the same range of sample size are grouped in the T2 network, with each group being assigned to a single CSN-Heads branch. Like the T1 network, the T2 consists of the same first stage and the second stage with multiple CSN-Heads branches. In this instance, however, each CSN-Heads branch is either designated for an object or a group of objects that fall within the same sample size range.

Network Settings: We experimented with multiple T2 networks to determine the most effective method for grouping the given 9 objects that fall within the same sample size range. The descriptions of the top two T2 networks are provided below:

A) A T2 network, denoted as T2A, has been built, encompassing four distinct CSN-Heads branches with specific duties. The initial branch is assigned to predict the traffic light, traffic sign, and pedestrian concurrently, while the second branch is responsible for predicting the rider and motorcycle jointly. The third branch is tasked with forecasting the bicycle, truck, and bus in combination, whereas the fourth branch is solely accountable for predicting the car, as illustrated in Figure 4.4. In T2A, the range of the

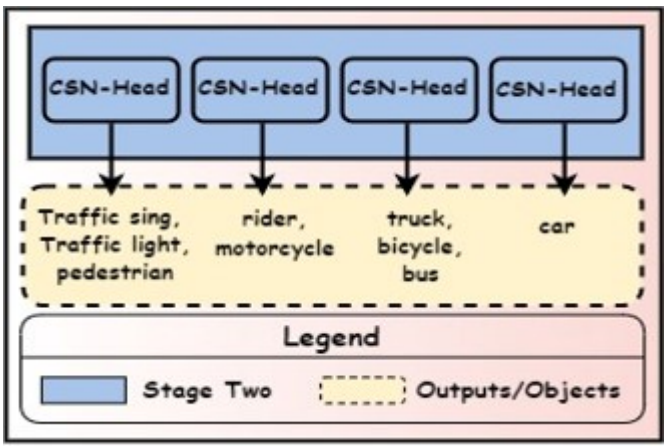

Figure 4.4: T2A Network Design

sample sizes is categorized by their common log rounded down to the nearest whole number, Let's call

it rounded down common log RDCL value. For example, in Figure 4.1, the object car has a sample size of 1,021,857 Images. Then, the log base 10 of the sample size of the car is rounded down to its nearest

Table 4.1: Rounded Down Common Log (RDCL)
Values of the Difference in Object Sample Sizes

Object A – B	Difference	RDCL
1,021,857	N/A	N/A
343,777-265906	77,871	4
265,906-129,262	136,644	5
129,262-42,963	86,299	4
42,963-16,505	26,458	4
16,505-10,229	6,276	3
10,229-6,461	3,768	3
6,461-4,296	2,165	3

whole number value, which is 6. Therefore, if any of the other objects have the same RDCL value, then, all these classes are grouped with the object car and vice versa. The following rationale underlies the grouping arrangement per group: 1) since all the objects in the first CSN-Heads branch have the same RDCL value of 5, hence, they are grouped. 2) Also, in the second branch, the sample size of all the objects has the same RDCL value of 3. 3) Similarly, in the third branch, the sample size of all the objects has a RDCL value of 4. 4) However, the object car is placed alone in the fourth branch because there are no other classes that have same RDCL value.

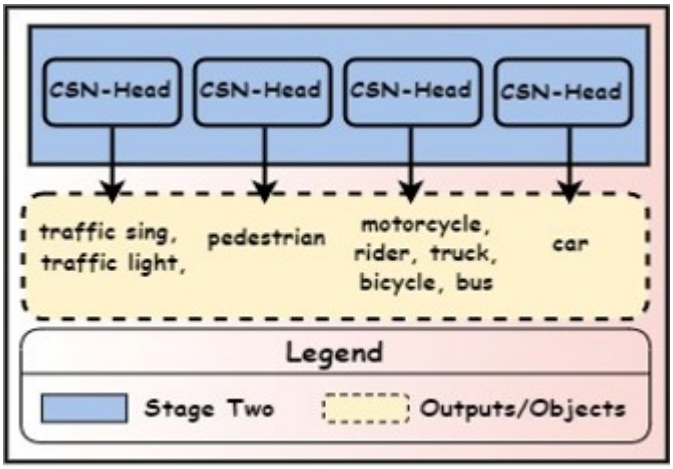

Figure 4.5: T2B Network Design

B) Another T2 network, designated as T2B, was constructed with four CSN-Heads branches, as exhibited in Figure 4.5. The first branch is responsible for the simultaneous prediction of the traffic light and traffic sign; the second for predicting the object pedestrian; the third for forecasting the bicycle, truck, bus, rider, and motorcycle; and the fourth for solely predicting the car. In T2B, the initial grouping from T1B was maintained, and subsequently ordered the objects by their respective sample size within each group. Then, the largest sample size was selected and subtracted from the second largest sample size. If the subtracted value yields a RDCL value less than 5, the objects are grouped together. This process

was iterated for all the groups, and the outcome is demonstrated in Table 4.1. Hence, the grouping arrangements in the network T2B are as follows: 1) Since the object car is assigned alone to a single branch in the T2A network, there is no need for further grouping. 2) Now, in group 2, the RDCL value of the difference between the object traffic sign and the traffic light is under 5, hence they grouped together. 3) However, the RDCL value of the difference between the object traffic sign and the pedestrian is not under 5. Thus, the object pedestrian is assigned alone to a single branch. 4) Although we could have groups 3 and 4 in two separate groups, we realize that the RDCL value of the difference between object bike (smallest sample in group 3) and object rider (most significant sample in group 4) is under 5. Thus, we decided to group them all together in one branch.

4.7.3 T3 Networks Design:

Lastly, in the T3 network type, objects or tasks that fall within the same sample size range and share comparable visual characteristics are grouped together and assigned to a single CSN-Heads branch. The T3 network, like the T1 and T2 networks, would have identical first and second stages, with the second stage consisting of several CSN-Heads branches designated for each group. **Network Setting:**In order to

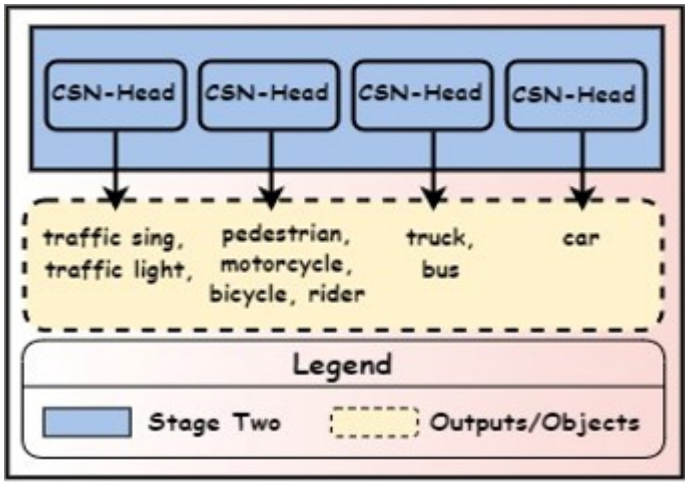

Figure 4.6: T3A Network Design

build a T3 network, we make use of the knowledge that we obtained from training T1 and T2 networks. For instance, to determine the most effective T3 network, the methods that were used to cluster the objects in T1 and T2 networks are combined and used together in this section. The following are descriptions of the two most prominent T3 networks:

A) A T3 network, named T3A, has been constructed, consisting of four CSN-Heads branches. The first branch is assigned to predict the traffic light and traffic sign jointly, the second to predict the rider, motorcycle, bicycle, and pedestrian collectively, the third to predict the truck and bus jointly, and the fourth to predict the car alone. These associations are illustrated in Figure 4.6. The T3A network's grouping order is justified by the following reasoning: 1) The best T1 network, T1B, will serve as the

first foundation. Then, the first two branches of the T1B network are preserved for the T3A network. 2) However, the third branch of the T1B network is subdivided further based on the RDCL value of the objects' sample sizes. For example, in the network T1B, the third group consists of the object car, bus, and truck, with respective sample sizes of 6, 4, and 4 RDCL values. Since the RDCL values for the truck and bus sample sizes are equivalent, they are grouped together in the third branch. 3) The object car is then placed in a separate branch, completing the T3A network with its fourth and final branch.

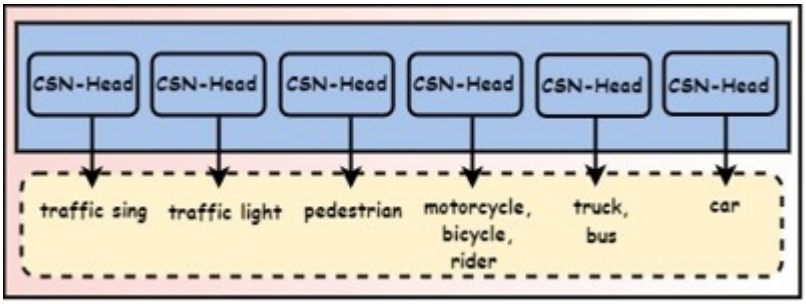

Figure 4.7: T3B Network Design

B) A second T3 network, known as T3B, was constructed with six CSN-Heads branches, as illustrated in Figure 4.7. The first CSN-Heads branch is tasked to predict the object traffic light, the second to predict the traffic sign, the third to predict the pedestrian, the fourth to predict the motorcycle, bicycle, and rider together, the fifth to predict the truck, and bus together, and the sixth to predict the car. The classification of the objects into six groups was determined based on their RDCL value of a random subset of their data. Objects with a value greater than or equal to 5 were assigned to their own CSN-Heads branch. For instance, the sample RDCL value of the objects traffic light, traffic sign, car, and pedestrian were all higher than 5, resulting in the assignment of each of these objects to a distinct CSN-Heads branch. The clustering of the first three branches and the last branch of the T3B network was thus explained, while the remaining two branches were considered as mere extensions of the T3A network."

4.8 Networks Training and Results Analysis

This section presents an in-depth discussion of the training process and outcomes achieved after training all six networks mentioned in section 4.7. Initially, we compared and analyzed the results obtained from utilizing the same type of network. Subsequently, we investigated performance differences between the top-performing network of each type. Ultimately, we make a conclusive decision by selecting the network type that exhibited superior performance in object detection using the BDD100K dataset.

4.8.1 T1 Networks Training and Results:

The training and evaluation of two proposed networks, T1A and T1B, on the BDD100K dataset are investigated here. The training process for T1A and T1B follows a similar approach to that of the base

network. However, due to the varying number of CSN-Heads branches per network, the hyperparameters are adjusted N times, where N represents the number of CSN-Heads branches. For example, the T1A network, which comprises four CSN-Heads branches, requires four optimizers and four learning rate values to be adjusted. The models of both T1A and T1B networks are trained for a predetermined epoch value of 20 using the SGD optimizer and an input batch size of 8.

T1A Training and Results:

In this subsection, the focus is on the training and evaluation of two T1A models. For the first T1A model, an initial learning rate of 0.01 was set for all four CSN-Heads branches. In order to avoid overfitting, the learning rates were gradually decreased after every third epoch, using a gamma value of 0.1. The training process involved the use of SGD optimizer, with a momentum value of 0.9 and weight decay value of 1^{-10^5}, for all four CSN-Heads branches. The model's performance was observed to converge by the 6th epoch, with the best results being obtained by the 15th epoch. The evaluation results showed that the

Table 4.2: Average Precision of each branch of T1A Network with its respective optimizer parameters Values

Branch	AP@0.50	AP@0.50:0.95	Momentum	Weight Decay
1	0.684	0.710	0.9	1^{-10^5}
2	0.569	0.540	0.99	3^{-10^5}
3	0.435	0.338	0.9	1^{-10^5}
4	0.573	0.566	0.99	3^{-10^5}

first CSN-Heads branch had the highest AP@50 score of 59.2% and an AP@50:95 score of 60.7%. The third branch performed the second best with an AP@50 score of 49.9% and an AP@50:95 score of 39.4%. The second and fourth branches ranked third and fourth, respectively, indicating that the first and third branches are more effective at object detection and classification.

To further enhance the performance of the T1A model, hyperparameters tuning was conducted, where the parameter values of each SGD optimizer for each of the four CSN-Heads branches were varied. The momentum value and weight decay value were adjusted for the first and third optimizers, while the second and fourth optimizers had different momentum and weight decay values. The momentum and weight decay values of each branch are shown in Table 4.2. The learning rate for all four CSN-Heads branches was kept constant at 0.01 to avoid negative learning.

The newly-tuned model exhibited an earlier convergence by the 3rd epoch and achieved the best results by the 9th epoch. The evaluation results indicated that the first CSN-Heads branch produced the highest AP@50 score of 68.4% and an AP@50:95 score of 71%. The fourth branch exhibited the second-best performance with an AP@50 score of 56.9% and an AP@50:95 score of 54%. The third branch scored an AP@50 score of 57.3% and an AP@50:95 score of 56.6%, ranking third. The second and fourth branches ranked fourth and fifth, respectively, indicating that the first, fourth, and third branches

are more effective at object detection and classification than the second and fourth branches. The results are summarized in Table 4.2.

Table 4.3: Average Precision of each branch of T1B Network with its respective optimizer parameters Values

Branch	AP@0.50	AP@0.50:0.95	Momentum	Weight Decay
1	0.611	0.706	0.9	1^{-10^8}
2	0.491	0.448	0.99	3^{-10^6}
3	0.633	0.631	0.99	3^{-10^6}

T1B Training and Results:

The T1B model, akin to its predecessor T1A, was initially trained with a learning rate of 0.01 for each of its three CSN-Heads branches. Subsequently, the learning rates were progressively decreased every third epoch with a gamma value of 0.1. Furthermore, the model was optimized with the SGD optimizer, utilizing a momentum value of 0.9 and a weight decay value of 1^{-10^5} for every four CSN-Heads branches. The convergence of the model was achieved at the sixth epoch, and its best results were produced at the end of the eleventh epoch, culminating in an overall training period of approximately seven days. The evaluation scores revealed that the highest score of AP@50 = 56.1% and AP@50:95 = 59.6% was attained by the 1st CSN-Heads branch, followed by the 3rd branch with an AP@50 of 53.6% and AP@50:95 of 50.1%, while the 2nd branch trailed behind with an evaluation score of AP@50 = 46.3% and AP@50:95 = 41.5%.

To further enhance the accuracy of the T1B network model, adjustments were made to its hyperparameters. Specifically, the SGD optimizers were modified by assigning a momentum value of 0.9 and a weight decay value of 1^{-10^8} to the first optimizer, a momentum value of 0.99 and a weight decay value of 3^{-10^6} to the second optimizer, and a momentum value of 0.99 and a weight decay value of 3^{-10^6} to the third optimizer. To prevent negative learning, the initial learning rate of 0.01 was retained for all three CSN-Heads branches and gradually decreased every three epochs with a gamma value of 0.1. The refined T1B model achieved convergence at the end of the seventh epoch, and its optimal results were produced towards the end of the tenth epoch. The final evaluation results for the optimized model are presented in Table 4.3, with the highest score of AP@50 = 61.1% and AP@50:95 = 70.6% being attained by the 1st CSN-Heads branch, followed by the 3rd branch with an AP@50 of 63.3% and AP@50:95 of 63.1%, and the lowest evaluation score of AP@50 = 49.1% and AP@50:95 = 44.8% being attained by the 2nd branch.

T1 Networks: Results Analysis

Here, a comparison was made between the T1A and T1B models of a multi-task object detection network against a single network with multiple object detection. As anticipated, both T1A and T1B models

Table 4.4: Average Precision of all the T1 networks

AP	**IoU = 0.50**		**IoU = 0.50:0.95**	
Branch	**T1A**	**T1B**	**T1A**	**T1B**
1	0.684	0.611	0.710	0.706
2	0.569	0.491	0.540	0.448
3	0.435	0.633	0.338	0.631
4	0.573	N/A	0.566	N/A

exhibited superior performance as compared to the base network, as indicated by the AP@50 results presented in Table 4.8. This finding lends support to the thesis that a multi-task object detection network is better than a single network with multiple object detection.

However, within the T1 network, it was observed that the T1B model had better overall performance compared to T1A, owing to the even results across each branch and the absence of one branch. Therefore, the T1B model required fewer computer resources, and the model exhibited generalization by not being biased toward any particular group. In T1A, the highest AP was significantly higher, while the lowest AP was considerably lower than in T1B. Thus, in T1A, the difference between the lowest and highest AP@50 was around 25%, as evidenced in Table 4.4. In contrast, in T1B, the difference was only 12%. As a result, T1B was deemed to be the superior model in this instance.

4.8.2 T2 Networks Training and Results:

The T2A and T2B networks, which were evaluated on the same BDD100K dataset, are examined in this section. The valuable insights gleaned during the T1 network training process were applied in the evaluation of the T2 networks. The T2 networks were trained for a predetermined epoch value of 20 with SGD optimizers and an input batch size of 8, similar to the T1 networks. It is worth noting that, during the T1 network training process, it was found that modifying the optimizer parameter at the group level rather than the learning rate is preferable. As a result, the same value of the learning rate was allocated to each branch in the T2 networks, and it was only modified at the model level. This implies that each branch's learning rates were all set to the same value.

T2A Training and Results:

Table 4.5: Average Precision of each branch of T2A Network
with its respective optimizer parameters Values

Branch	**AP@0.50**	**AP@0.50:0.95**	**Momentum**	**Weight Decay**
1	0.564	0.497	0.9	1^{-10^7}
2	0.498	0.455	0.99	3^{-10^5}
3	0.571	0.543	0.99	3^{-10^5}
4	0.737	0.699	0.9	1^{-10^7}

In terms of the T2A models' training process, an initial learning rate of 0.01 was assigned for all

four CSN-Heads branches. Subsequently, the learning rates were gradually decreased with a gamma of 0.1 on every third epoch. In contrast to the T1 networks, different optimizer parameter values were assigned to the four CSN-Heads branches based on their group size. This was due to the discovery during T1 network training that adjusting optimizer parameters leads to superior outcomes. Therefore, several additional T2A models were trained, with most hyperparameter values held constant except for the optimizer parameter values, which were modified for each T1 model. The best-performing T2A network had the following parameter configuration for each SGD optimizer: a momentum value of 0.9 and a weight decay value of 1^{-10^7} for the first optimizer; a momentum value of 0.99 and a weight decay value of 3^{-10^5} for the second and third optimizers, while the fourth optimizer had a momentum value of 0.9 and a weight decay value of 1^{-10^7}. Different parameter values were assigned for each branch based on the optimal outcome of training the model. The T2A model was completed in roughly seven days, with the best results obtained at the end of the tenth epoch.

Regarding the evaluation of the T2A model, the highest evaluation score was achieved by the 4th CSN-Heads branch, with AP@50 and AP@50:95 values of 73.7% and 69.9%, respectively. The second-best evaluation score was obtained by the 3rd CSN-Heads branch with AP@50 = 57.1% and AP@50:95 = 54.3. The 1st CSN-Heads branch secured the third position with an evaluation score of AP@50 = 56.4% and AP@50:95 = 49.7%, while the 2nd CSN-Heads branch obtained an evaluation score of AP@50 = 49.8% and AP@50:95 = 45.5%, securing the fourth position, as depicted in Table 4.5.

T2B Training and Results:

Table 4.6: Average Precision of each branch of T2B Network with its respective optimizer parameters Values

Branch	AP@0.50	AP@0.50:0.95	Momentum	Weight Decay
1	0.646	0.613	0.9	3^{-10^5}
2	0.622	0.609	0.9	1^{-10^7}
3	0.517	0.483	0.9	3^{-10^5}
4	0.783	0.765	0.9	1^{-10^7}

In a similar manner to the T2A network, the T2B models were trained with an initial learning rate of 0.01 for each of the three CSN-Heads branches. The learning rates were gradually decreased by a factor of 0.1 every six epochs to allow for longer training periods between the values of 0.01 and 0.001. We observed that this method improves convergence during the training process. We optimized the models' optimizers' parameters, including momentum and weight decay, to identify the optimal T2B model. The best T2B model was obtained using different parameter values for each of the four SGD optimizers. Specifically, the first SGD optimizer employed a momentum value of 0.9 and a weight decay of 3^{-10^5}, while the second utilized a momentum of 0.9 and a weight decay of 1^{-10^7}. The third SGD optimizer employed a momentum of 0.9 and a weight decay of 3^{-10^5}, and the fourth utilized a momentum of 0.9

and a weight decay of 3^{-10^5}. The model began converging during the seventh epoch and produced the best results at the end of the eleventh epoch. Among the four CSN-Heads branches, the fourth branch achieved the best evaluation score with an AP@50 value of 78.3% and an AP@50:95 value of 76.5%. The first CSN-Heads branch achieved the second-best evaluation score with an AP@50 value of 64.6% and an AP@50:95 value of 61.3%. The second CSN-Heads branch achieved the third-best evaluation score with an AP@50 value of 62.2% and an AP@50:95 value of 60.9%. The third CSN-Heads branch had the lowest evaluation score, with an AP@50 value of 51.7% and an AP@50:95 value of 48.3%. These results are summarized in Table 4.6.

T2 Networks: Results Analysis

Table 4.7: Average Precision of all the T2 networks

AP	**IoU = 0.50**		**IoU = 0.50:0.95**	
Branch	**T2A**	**T2B**	**T2A**	**T2B**
1	0.564	0.646	0.497	0.613
2	0.498	0.622	0.455	0.609
3	0.571	0.517	0.543	0.483
4	0.737	0.646	0.699	0.765

As previously mentioned in section 4.7, the T2 network utilizes a grouping strategy based on the similarity of the sample size of objects to address the issue of class imbalance. This issue can cause a model to be biased towards the dominant class, as reported by Krawczyk and colleagues [52] and Patel and colleagues [53]. As expected, the T2 network proves effective in overcoming this issue, as evidenced by the improved performance of both T2A and T2B compared to the base network, particularly in their respective AP@50 values. The results presented in Table 4.7 demonstrate that the average AP@50 of the T2B network is 60%, which is three percentage points higher than that of the base model. In comparison, the T2B network outperforms T2A in terms of overall AP@50 values, indicating superior generalization ability. Notably, the difference between the lowest and highest AP@50 scores for T2B is 13%, while for T2A, it is 24%. Thus, based on these results, it is apparent that T2B outperforms T2A, making it the preferred choice in the T2 network.

4.8.3 T3 Networks Training and Results:

This section aims to assess the efficacy of T3A and T3B networks on the BDD100K dataset by training multiple models with SGD optimizers over 20 epochs. To ensure uniformity across the networks, a consistent learning rate was allocated to each branch, as in the T1 and T2 networks, and modified only at the model level. This ensured that the learning rates of each branch were all tuned to the same value.

The SGD optimizer parameters were then adjusted at the group level to create the optimal T3 network. As such, the parameters for each CSN-Heads branch may differ according to the number of objects in

the group. This approach was adopted to identify the most effective T3 network configuration for object detection.

T3A Training and Results:

Table 4.8: Average Precision of each branch of T3A Network with its respective optimizer parameters Values

Branch	AP@0.50	AP@0.50:0.95	Momentum	Weight Decay
1	0.651	0.679	0.9	1^{-10^8}
2	0.533	0.470	0.99	3^{-10^6}
3	0.658	0.656	0.99	3^{-10^6}
4	0.788	0.787	0.9	1^{-10^8}

Multiple T3A models were subjected to training using an initial learning rate of 0.01, followed by a gradual reduction in the learning rate with a gamma value of 0.1 after every sixth epoch. In order to optimize these T3A models, SGD optimizers were utilized, with the parameters of the optimizer individually calibrated. The optimal configuration of the SGD optimizer for the best-performing T3A model was determined as follows: the first optimizer was allocated a momentum value of 0.9 and a weight decay value of 1^{-10^8}, the second optimizer had a momentum value of 0.99 and a weight decay value of 3^{-10^6}, the third optimizer was assigned a momentum value of 0.99 and a weight decay value of 3^{-10^6}, and the fourth optimizer was allocated a momentum value of 0.9 and a weight decay value of 1^{-10^8}.

The model began to converge from the 7th epoch and achieved its highest performance at the end of the 12th epoch. Our evaluation results indicate that the 4th CSN-Heads branch yielded the highest AP@50 score of 78.8% and AP@50:95 score of 78.7%. The 1st CSN-Heads branch obtained the second-best evaluation score with an AP@50 of 65.1% and AP@50:95 of 67.9%. The 3rd CSN-Heads branch secured the third spot with an evaluation score of AP@50 = 65.8% and AP@50:95 = 65.6%. Lastly, the 2nd CSN-Heads branch ranked fourth, having obtained an evaluation score of AP@50 = 53% and AP@50:95 = 47%, as presented in Table 4.8)

T3B Training and Results:

Table 4.9: Average Precision of each branch of T3B Network with its respective optimizer parameters Values

Branch	AP@0.50	AP@0.50:0.95	Momentum	Weight Decay
1	0.651	0.560	0.9	1^{-10^8}
2	0.692	0.673	0.9	1^{-10^8}
3	0.684	0.626	0.9	1^{-10^7}
4	0.536	0.440	0.99	3^{-10^6}
5	0.660	0.654	0.99	3^{-10^6}
6	0.790	0.782	0.9	1^{-10^8}

In a manner similar to T3A models, all T3B models were trained using an initial learning rate of 0.01 for each CSN-Heads branch, followed by a gradual reduction of learning rates with a gamma of 0.1 for every 6 epochs. The hyperparameter settings were nearly identical, except for the consideration of parameter values for the six optimizers in T3B models. The optimizers' parameter values for each T3B model were fine-tuned to determine the optimal model. The optimizers' parameter values for the best T3A model are shown in Table 4.9. Specifically, the optimizers' parameters for the first, second, and final optimizers had momentum values of 0.9 and weight decay values of 1^{-10^8}. The momentum values for the third, fourth, and fifth optimizers were 0.9 and 0.99, while the weight decay values were 1^{-10^8} and 3^{-10^6}, respectively. The model training took about seven days, with the optimal results obtained at the end of the eleventh epoch, despite the model starting to converge well from the fifth epoch.

The evaluation scores of the best T3B network were computed for each CSN-Heads branch, with the first branch generating AP@50 and AP@50:95 values of 65% and 56%, respectively; the second branch generating an evaluation score of AP@50 = 69% and AP@50:95 = 67%; the third branch generating an evaluation score of AP@50 = 68.4% and AP@50:95 = 62.4%; the fourth branch generating an evaluation score of AP@50 = 53.6% and AP@50:95 = 44%; the fifth branch generating an evaluation score of AP@50 = 66% and AP@50:95 = 65.4%; and the final branch generating an evaluation score of AP@50 = 79% and AP@50:95 = 78.2%, as presented in Table 4.9.

T3 Networks: Results Analysis

Table 4.10: Average Precision of all the T3 networks

AP	**IoU = 0.50**		**IoU = 0.50:0.95**	
Branch	**T3A**	**T3B**	**T3A**	**T3B**
1	0.651	0.651	0.679	0.560
2	0.533	0.692	0.470	0.673
3	0.658	0.684	0.656	0.626
4	0.788	0.536	0.787	0.440
5	N/A	0.660	N/A	0.654
6	N/A	0.790	N/A	0.782

In section 2.7, it was noted that the T3 networks utilize a grouping mechanism that clusters objects based on their sample size falling within the same given range and their visual features resembling one another. By integrating the principles of T1 and T2 networks, the T3 networks demonstrate characteristics of both types, which enables them to overcome the negative learning and class imbalance issues observed in the T1 and T2 networks. Therefore, it is unsurprising that both T3 networks outperform the winners of the T1 and T2 networks. In terms of comparison within the two T3 networks, T3B outperforms T3A in all aspects. Table 4.10 demonstrates that T3B's average of all the AP@50 scores surpasses that of T3A. Furthermore, T3B demonstrates superior generalization performance relative to T3A, making it the superior type of T3 network.

4.8.4 Result Analysis Between All the Network Types:

In the above analysis, it was determined that T1B, T2B, and T3B were the winners of their respective network types. The present study conducted a comparison and analysis of these networks to identify the final network with higher individual branch accuracy and a more even distribution of accuracy between each branch. It was observed that grouping objects together based on their visual features in T1 networks contributed to building a better MTL network compared to the base network (single-task network). The base network's AP@50 value was 53.3%, while the average AP of all branches in the T1B network was around 58%, with the highest AP@50 being 63.3% and the lowest AP@50 being approximately 50%.

Similarly, it was observed that using the sample size of objects to group them together in T2 networks also improved the MTL network's performance over the base network. The average AP of all branches in T2B was around 61%, significantly higher than the base network's result. Additionally, grouping objects based on their sample size was found to be more effective than grouping objects based on their visual features for the dataset under investigation. As shown in Table 4.11 the AP values in T2B were more evenly distributed between the groups than in the T1B network, with T2B having the highest and lowest AP@50 values of 64.4% and 51.7%, respectively. In contrast, the T1B network had the highest and lowest AP@50 values of 63.3% and 49.1%, respectively.

Table 4.11: Average Precision of All the Network Types

	AP@IoU = 0.50						**AP@IoU = 0.50:0.95**					
Branch	**T1A**	**T1B**	**T2A**	**T2B**	**T3A**	**T3B**	**T1A**	**T1B**	**T2A**	**T2B**	**T3A**	**T3B**
1	0.684	0.611	0.564	0.646	0.651	0.651	0.710	0.706	0.497	0.613	0.679	0.560
2	0.569	0.491	0.498	0.622	0.533	0.692	0.540	0.448	0.455	0.609	0.470	0.673
3	0.435	0.633	0.571	0.517	0.658	0.684	0.338	0.631	0.543	0.483	0.656	0.626
4	0.573	N/A	0.737	0.646	0.788	0.536	0.566	N/A	0.699	0.765	0.787	0.440
5	N/A	N/A	N/A	N/A	N/A	0.660	N/A	N/A	N/A	N/A	N/A	0.654
6	N/A	N/A	N/A	N/A	N/A	0.790	N/A	N/A	N/A	N/A	N/A	0.782

Finally, using both sample size and visual features of objects to group them together produced the best MTL network model, namely T3, which outperformed all other network types, including the base network. Thus, it is evident that MTLnetworks perform better than single-task networks. Regarding the most efficient and accurate MTL network, T3 was found to outshine both the T1 and T2 networks comprehensively. As shown in Table 4.11, the average AP@50 of all branches in T3B was around 66%, with the highest AP@50 value of 79% and the lowest AP@50 value of 53.6%. In contrast to the T1 and T2 networks, the lowest AP@50 value of T3B was actually higher than the AP@50 value obtained using the base network. This suggests that when objects in an MTL network are grouped efficiently, it generates much higher and more accurate results compared to a single-task network like the base network. In other words, the present model managed to mitigate the data imbalance issue in a single-task network of multi-label learning.

4.9 Important Observations

This section discusses the improtant obersavations carried out during the training process. Specifically, the effect of learning rate, momentum, and weight decay on the MTL R-CNN networks in general.

4.9.1 Effect of Learning Rate:

The learning rate is a hyperparameter that determines the step size at which the optimizer makes updates to the model weights during training. In a multi-task network, the learning rate can have a significant impact on the performance of the model. Gyoung confirms its' significance in [54]" In the gradient methods, learning rate (or step size) is one of the most important hyper-parameters that determines the overall optimization performance." If the learning rate [11] is too high, the model may overshoot the optimal weights and converge to a suboptimal solution, or even diverge and fail to converge at all. On the other hand, if the learning rate is too low, the model may take too long to converge or may get stuck in a local minimum.

In a multi-task network, it is important to carefully tune the learning rate to ensure that the model is able to effectively learn from the data and perform well on all tasks. One way to do this is to use a learning rate schedule, which adjusts the learning rate over the course of training based on the performance of the model. We did so in all the experiments by using a step size-based learning rate schedular called, StepLR. It decays the learning rate of each parameter group by gamma, 0.1 in our cases, for every step size epoch. We tested various step sizes for each task and found a step size of 5 or 6 works best when the initial learning rate is set to 0.01 for our training setups.

It is also possible to use different learning rates for different tasks in a multi-task network, depending on the relative importance of the tasks or the difficulty of learning each task. Using different learning rates for different tasks in a multi-task network can be an effective way to optimize the model's performance on each task. This can be particularly useful if the tasks have different levels of complexity or require different amounts of data to learn effectively.

Table 4.12: Average Precision(IoU@0.50) of Two T3A Models

Branch	Model 1	Model 2
1	0.651	0.607
2	0.533	0.199
3	0.658	0.572
4	0.788	0.801

However, we observed that utilizing different learning rates for various tasks in multi-task learning MTL can prove to be challenging, as it can result in a negative transfer. Negative transfer [55, 56], also known as negative learning, is a phenomenon that can occur in MTL, where the performance of a model on one task is negatively impacted due to variations in learning rate between tasks leading to a biased

network.

Our investigation found that utilizing a higher learning rate for a specific task can cause the model to rapidly update its parameters for that task while neglecting other tasks. As a result, the model may not converge to an optimal solution for all tasks, leading to over-fitting on one task and under-fitting on the others. To mitigate this issue, we determined that a model-level tuning approach for the learning rate should be employed. This approach involves utilizing the same learning rate for all tasks, thus ensuring that the model updates its parameters at a consistent pace for all tasks. For example, Table 4.12 shows a comparison of AP@50 values between model 1 and model 2, where model 1 is a T3A network model and its learning rate is tuned at model level and model 2 is another T3A model, however, its learning rate is tuned at the task level. The results demonstrate that Model 1 outperforms Model 2 by having a more balanced AP value among the tasks. Whereas in Model 2, only one branch has high AP, and the remaining were with subpar values. This shows the presence of negative transfer in Model B as it seems to over-fit task 4 and under-fit the other tasks. This is because of the difference in learning rate between tasks which causes the model to prioritize one task over the other. Thus, by tuning the learning rate at the model level in model 1, we managed to mitigate the negative transfer issues in model 3.

4.9.2 Effect of Momentum:

Momentum is a hyperparameter that determines the extent to which the optimizer takes into account the previous updates when making new updates to the model weights. It is commonly used in conjunction with gradient descent-based optimization algorithms, such as stochastic gradient descent (SGD) and its variants. The momentum [57, 58] term helps the optimizer to escape from local minima and accelerate convergence by adding a fraction of the previous update to the current update. This can help the model to make more consistent and efficient updates, especially when the gradients are noisy or have a lot of variances.

In a multi-task network, the momentum term can have a significant impact on the performance of the model. Similar to the learning rate, if the momentum is too high, the optimizer may overshoot the optimal weights and converge to a suboptimal solution. On the other hand, if the momentum is too low, the optimizer may take longer to converge or may get stuck in a local minimum. The default value for momentum in the SGD optimizer is given as 0.9, however, the optimal momentum value can vary depending on the specific characteristics of the data and the tasks being learned and may require some experimentation to determine.

It is possible to use different momentum values for different tasks in a multi-task network like ours, depending on the specific characteristics of the tasks and the data being used to learn them. For instance, while training a model using the T3A network, the tasks that were assigned to predict multiple objects together shows signs of loss oscillation when the momentum for all the tasks is assigned with a value of

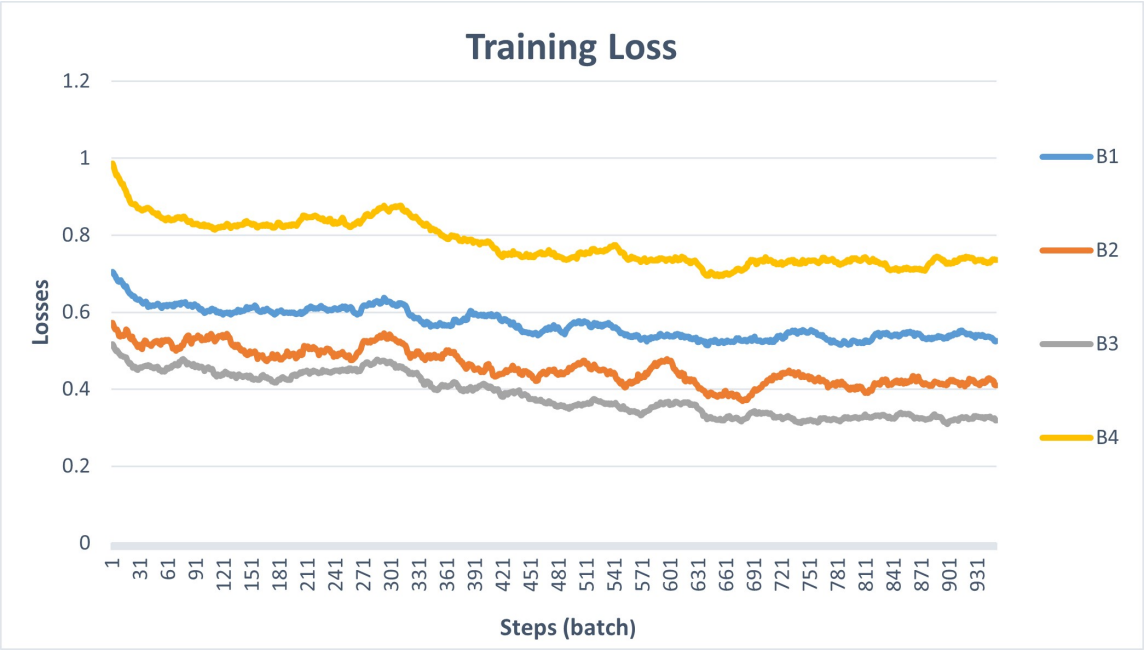

Figure 4.8: Training Loss of each Task of T3A Network

0.9. The oscillation of losses can happen when the network is trying to learn multiple objects at the same time, such as in multi-task learning or multi-label classification. In these cases, the network may struggle to find the optimal solution for all the labels, resulting in oscillations in the loss function. The main cause of this phenomenon is that the network is trying to find a balance between different objectives, each with its own gradient direction. This can lead to the network oscillating between different local minima, resulting in an unstable training process.

One method to address the issue of loss oscillation in multi-task networks is to assign different momentum values to the optimizers for different tasks, based on the specific characteristics of the tasks. This was clearly evidenced during the repeated training of multiple T3A network models. For instance, in our T3A network, the tasks that were assigned to predict multiple objects together showed signs of loss oscillation when using the same default momentum value of 0.9 for all the tasks. To mitigate this, we assigned higher momentum values for the optimizers of these tasks. Specifically, we assigned a momentum value of 0.99 for the first branch (B1), 0.999 for the second branch (B2) and third branch (B3), and a default value of 0.9 for the fourth branch (B4). Therefore, by assigning higher momentum we managed to avoid the loss oscillation to a great extent as shown in Figure 4.8.

This approach of assigning different momentum values to different tasks works as it allows the optimizers to make more consistent and efficient updates for each task, particularly when the gradients are noisy or have a lot of variances. The higher momentum values for tasks that predict multiple objects together can help the optimizer to escape from local minima more effectively and accelerate convergence, thus avoiding loss oscillation.

4.9.3 Effect of Weight Decay:

Weight decay is a regularization technique that is commonly used in machine learning to prevent overfitting and improve the generalization ability of the model, which is confirmed in [59, 60]. It works by adding a penalty term to the loss function that encourages the model weights to be small and close to zero. This helps to reduce the complexity of the model and prevents it from memorizing the training data too closely.

In a multi-task network, weight decay can be an effective way to improve the performance of the model on each task by reducing overfitting and increasing the model's ability to generalize to new data. However, the optimal weight decay value can vary depending on the specific characteristics of the tasks and the data being used to learn them.

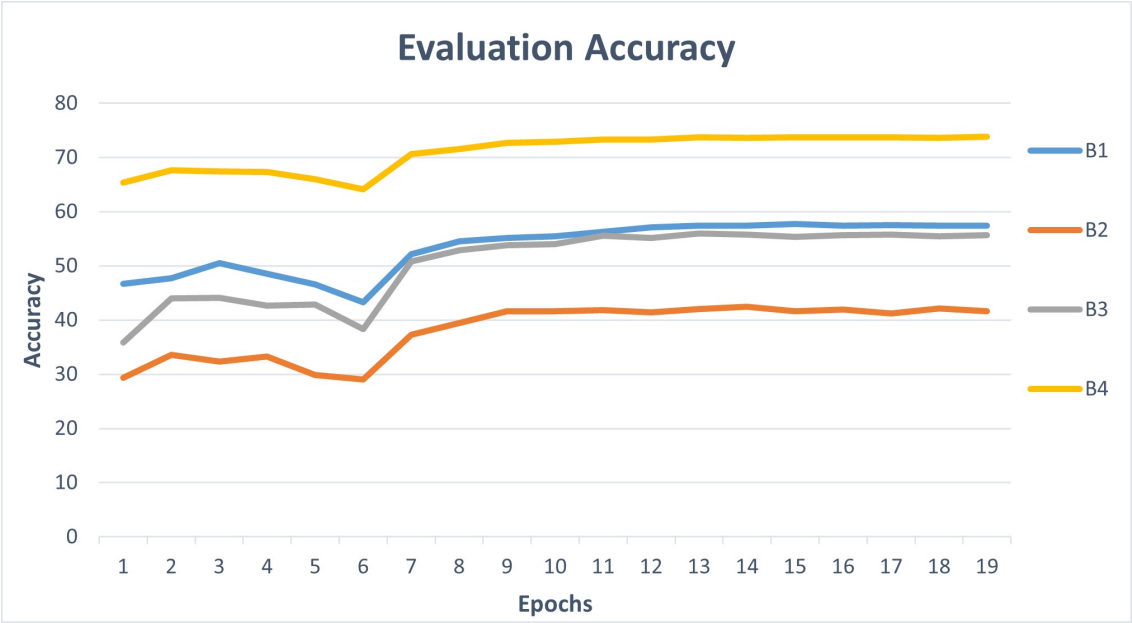

Figure 4.9: Evaluation Accuracy of each Task of T3A Network

If the weight decay value is too high, it may cause the model weights to become too small, which can lead to underfitting and poor performance on the training data. On the other hand, if the weight decay value is too low, it may not effectively prevent overfitting and may result in poor generalization performance. It is generally recommended to use a relatively small weight decay value, such as 0.001 or 0.01 when training a multi-task network. However, the optimal weight decay value may require some experimentation to determine and may depend on the specific characteristics of the tasks and data being used.

Similar to momentum, it is possible to use different weight decay values for different tasks in a multi-task network, depending on the specific characteristics of the tasks and the data being used to learn them. For example in the case of the T3A network, we noticed that tasks with larger data samples tend to influence the result of tasks with smaller samples because of the overfitting issues. To mitigate this,

we used a higher weight decay value for tasks with a larger sample, namely task or branch one (B1) and branch four (B4), to encourage the model to use simpler and more generalizable representations. On the other hand, we used a lower weight decay value for the tasks that have a small amount of data to learn, namely branch two (B2) and branch three (B3), in order to avoid underfitting. Figure 4.9 illustrates that evaluation accuracy increases for all the tasks as the training loss decreases as shown in Figure 4.8. This shows that our model generalizes well, and the model is not overfitting because if the training loss is decreasing but the evaluation accuracy is decreasing, it indicates that the model is overfitting to tasks with higher sample data. *Note: The slight dip in the evaluation accuracy in Figure 4.9 at epoch six shows the transition of change in learning rate as we used a step size of 6 for all the learning rate schedulers.*

4.10 Results Comparison to Related Work

This section undertakes an evaluation of the efficacy of the proposed multi-task learning system, MTL R-CNN, by conducting a comparison of the validation accuracy of its most superior performing network, T3B, against recent object detection methodologies. The comparison is executed through two main investigations. Firstly, the accuracy of the T3B network model is compared to other multi-task learning approaches that incorporate object detection tasks. Secondly, the accuracy of the T3B network model is juxtaposed against object detection approaches that exclusively employ single-task methodologies. The objective of this investigation is to establish that the proposed MTL system has led to better accuracy outcomes in multiple object detection tasks compared to single-task networks.

For the first comparison, recent approaches [25, 2] to multi-task learning on computer vision tasks using BDD100K have been selected. It should be noted that multi-task learning specifically for object detection, like our proposed system, is not performed by any other system. Therefore, the selection criteria for these approaches primarily rely on their multi-task nature and the designation of at least one branch of the network for the object detection task.

Table 4.13: Result Comparisons to Recent Multi-Task Approaches

Approaches	**AP@IoU=50**
YOLOP [25]	76.50
JSUMBDP [2]	76.16
MTL R-CNN(T3B's branch 6)	**78.20**

The papers [25, 2], propose a multi-task learning framework for autonomous driving. Specifically, the framework concurrently performs traffic object detection, drivable area segmentation, and lane line segmentation. Both frameworks adopt a shared encoder and three decoders, with each decoder handling a specific task. Notably, for the object detection task, both approaches utilize the same decoder algorithm, leading to a similar AP50 of 76.16% for [2] and 76.5% for [26]. It should be noted that object detection in both frameworks is limited to a single object, which is the vehicle object. Therefore, to compare the

performance of the proposed system, we directly compare the accuracy of branch 6 in the T3B network, which is also responsible for detecting vehicle objects, with the object detection decoder accuracy of [25, 2], rather than evaluating the overall network accuracy of T3B. The findings demonstrate that our proposed system outperforms the two aforementioned approaches, with T3B's branch 6 accuracy value of 78.2%, as shown in Table

For the second comparison, a selection of recent papers [61, 62] that employ single-task techniques with state-of-the-art algorithms for object detection has been included. The selection criteria for these approaches are primarily based on their recent publication date and competitive performance on the BDD100K dataset. In [61], Woo et al compare the performance of models trained on BDD100K and MS COCO datasets using the state-of-art YOLOv4 algorithm. Additionally, qualitative evaluation was performed, and the reliability of the bounding box and the ability to detect traffic-related objects were analyzed. Since the T3B is evaluated on BDD100K, the accuracy of the YOLOv4 model trained on the same dataset is considered for the comparison. As shown in Table 4.14 our proposed network, T3B, outperforms [61] with mAP50 values of 66.0% and 48.79% respectively.

Table 4.14: Result Comparisons to Recent Single-Task Approaches

Approaches	**mAP@IoU=50**
YOLOv4 [61]	48.80
STA-FRCNN [62]	41.10
MTL R-CNN (Ours)	**66.00**

In paper [62], Ikeda et al propose three methods to enhance the state-of-the-art algorithm object detection Faster R-CNN: SA-FRCNN; STA-FRCNN; and a lambda layer refines feature representation. The proposed methods were evaluated on all 9 objects of the BDD100K dataset, and all achieved improved accuracy compared to the original Faster R-CNN. SA-FRCNN achieved 40.4% mAP; STA-FRCNN achieved 41.1% mAP; and the lambda layer achieved 39.9% mAP. However, all three methods underperform when compared to the T3B network accuracy of 66.0%.

In conclusion, the T3B network demonstrates superior performance when compared to the above multi-task learning and single-task learning approaches. In multi-task learning comparison, the T3B network achieved a lot higher AP50 value in detecting vehicle objects than that of [2, 26]. Whereas in single-task learning, the T3B network demonstrates superior performance compared to the three papers [61, 62]. Specifically, the T3B network achieved a mAP50 value of 66.0%, which is significantly higher than the values achieved by [61, 62]. The T3B network's excellent performance can be attributed to its multi-task nature, where the network can leverage the knowledge learned from one task to improve its performance on another related task. Thus, this proves that multi-task learning triumphs over its single-task learning counterparts.

CHAPTER 5

Conclusion

Our concluding comments on the proposed system are offered in this chapter. In addition, several possible future directions for our suggested system are presented below.

5.1 Concluding Remarks

This study proposes a new multi-task learning system called MTL R-CNN for computer vision tasks, which is designed to simultaneously perform multiple related tasks while sharing a common feature space. The study introduces three network architectures, namely T1, T2, and T3 networks, that group objects based on different criteria. T1 networks group objects based on their visual features, T2 networks group objects based on their sample size, and T3 networks combine both criteria to group objects based on their visual features and sample size.

To evaluate the performance of the proposed system and the three network architectures, six different networks were constructed by creating two different networks for each of the three network types T1A, T1B, T2A, T2B, T3A, and T3B. The study then conducted experiments on these six networks using the BDD100K dataset, which contains annotated images of nine classes of objects, namely traffic light, traffic sign, rider, pedestrian, bicycle, motorcycle, truck, bus, and car. The results of the same type of networks were compared and analyzed, and subsequently, the performance differences between the top-performing networks of each type were investigated. Finally, the network type that exhibited superior performance using the BDD100K dataset was selected.

The study found that all six MTL network types outperformed the single-task network (base network) in terms of AP50 scores. The performance of each network type was compared, and the winners of their

respective network types were T1B, T2B, and T3B. These networks were compared and analyzed to find the final network with higher individual branch accuracy and more even accuracy between each branch, and it was concluded that T3B outperforms T1B and T2B comprehensively. The average AP@50 of all the branches in T3B is around 66%, with the highest AP@50 value of 79% and the lowest AP@50 value of 53.6%. The study observed that T3 networks exhibit characteristics of both T1 and T2 networks, enabling them to overcome the negative learning and class imbalance issues observed in T1 and T2 networks.

The study also evaluated the proposed system for its efficacy in multiple object detection tasks by comparing it to recent state-of-the-art approaches. The evaluation was carried out by comparing the accuracy of the T3B network, the most superior performing network of the MTL R-CNN system, against recent object detection methodologies. Two main investigations were conducted. Firstly, the accuracy of the T3B network model was compared to other multi-task learning approaches that incorporate object detection tasks, such as YOLOP and other Joint Multi-task Learning. The findings demonstrated that the proposed MTL system outperformed these methods with T3B's accuracy value of 78.2%. Secondly, the accuracy of the T3B network model was juxtaposed against object detection approaches that exclusively employ single-task methodologies, such as YOLOv4 [61], and STA Faster RCNN [62]. The T3B network achieved a mAP50 value of 66.0%, which is significantly higher than the values achieved by the YOLOv4 [61], and STA Faster RCNN [62].

In conclusion, this study highlights the benefits of using multi-task network architectures for multi-object detection systems, which can mitigate class imbalance and negative transfer issues in computer vision tasks. The study demonstrates the effectiveness of the proposed MTL system and the T3B network architecture, which outperforms other approaches in terms of accuracy. However, the study notes the importance of considering the resource requirements and trade-offs of network architecture when determining the appropriate number of branches for specific tasks. The findings of this study can guide the development of more effective multi-task learning systems for computer vision tasks.

5.2 Future Directions

Although we have created several multi-task networks using our proposed system, we still have not tested all the capabilities of our system. So far in this thesis, we have exhaustively evaluated our system with networks containing multiple CNS Head branches but only a single shared RPN network. In addition, we solely conducted experiments on the object detection aspect of computer vision. Therefore, there are numerous aspects and areas of study that we have not yet evaluated with our networks. The following are potential tasks that we could perform with our system to evaluate its full capabilities:

- Evaluate our system with multiple RPN networks: One of the important observations we made during our experiments is that the RPN network serves a very crucial purpose in a multi-learning

setup. We noticed that due to the multi-task nature of our system, our RPN network needed more than 2000 proposals, which is the default value in the base network. Since there were multiple CSN-Heads in all our experiments, we noticed that keeping 2000 proposals per instance of an image was not enough to generalize the model to all the respective tasks properly. We noticed that having only 2000 proposals from the RPN network would limit the model's ability to generate generalized proposals, meaning all the proposals would at least have enough representation for each CSN-Heads. Therefore, to counter this issue, we changed the 2000 proposals to 3500, and the results were great, as we have seen in the previous section. However, I believe that we could have further improved the result if we had created multiple RPN networks and assigned each RPN network to a single CSN-Heads. This would enable objects with smaller samples to have fair representation because of the designated RPN network. We could not carry out this test for this thesis because the extra RPN networks make our network too large for our system to train its model, despite having 16 GB of GPU RAM. However, in the future, we would like to try an open-source supercomputer, like Compute Canada, to build such a network and test our system to its maximum possible size.

- Evaluate our system for joint learning on computer vision tasks: As mentioned before our system does have the ability to carry out instance segmentation tasks as well. There is plenty of open-source datasets that one can use to evaluate the instance segmentation part of our system. However, it would be more interesting if we create a joint learning system using our proposed system.

  Joint learning[63] is a form of multi-task learning where multiple tasks are learned simultaneously, rather than independently. In joint learning, a single model is trained to perform multiple tasks together, and the model's parameters are shared among all tasks. This allows the model to learn useful features and representations that can be used to improve the performance of all tasks.

  One of the main advantages of joint learning is that it can improve the performance of individual tasks by leveraging the information learned from other tasks. For example, if two tasks are closely related, such as object detection and semantic segmentation, training a model to perform both tasks jointly can improve the performance of both tasks by leveraging the information shared between them.

  Joint learning can also be more efficient than training individual models for each task, as it reduces the number of parameters that need to be learned. Additionally, it can also be more robust to changes in the data distribution, as the model is able to learn a more general representation of the data.

  Thus, similar to the systems [3, 2] we have seen in chapter 2, we could create a joint learning network for instance segmentation and object detection tasks. We could do so by creating a network using our system with a single stage 1 and multiple CSN-Heads or stage 2 designated for each task.

We can evaluate the network using BDD100K dataset since it has data labels for both instance segmentation and object detection for the same set of images.

- Exploring the use of MTL R-CNN for semi-supervised learning: Semi-supervised learning [64, 65] is a machine learning technique that uses a small amount of labeled data and a large amount of unlabeled data to train a model. The idea is that the model can learn from both labeled and unlabeled data to improve its performance. The labeled data is used to train the model to make predictions, while the unlabeled data is used to learn more general features and representations of the data.

  Semi-supervised learning can also be applied in the context of multi-task learning (MTL). In this case, the model is trained using a combination of labeled and unlabeled data for multiple tasks. The goal is to leverage the information shared among the tasks to improve the performance of each task, even when labeled data is scarce or expensive to obtain.

  One common approach in semi-supervised MTL is to use the labeled data for one or a few tasks to train the model, and then use the unlabeled data for the remaining tasks to improve the performance. This is known as transductive transfer learning, where the model is transferred from one task to another.

  Another approach is to use the unlabeled data to learn a shared representation of the data that can be used to improve the performance of all tasks. This is known as inductive transfer learning, where the model is trained to learn a general representation of the data that can be used for multiple tasks.

  Semi-supervised learning is useful in situations where labeled data is scarce or expensive to obtain. Hence, BDD100K is a perfect dataset to test semi-supervised learning using our system. This is because although BDD100K consists of 70K training labels for object detection, however, it only has 7K images with instance segmentation labels as stated in BDD100K.

  The semi-supervised learning network can be implemented using our system as follows; we create can an MTL network with one CSN-Heads designated to learn the object detection label and another CSN-Heads will learn the instance segmentation label. Both branches would have a shared first stage and they are learned together in joint learning way as suggested in the previous example. Only this time, the whole image set of 70k inputs is used for training the whole network instead of an image set of 7K with balanced labels for both object detection and instance segmentation. With some minor changes in the network design, we can easily execute the training in both inductive transfer learning and transductive transfer learning way.

- Exploring the use of MTL R-CNN in Reinforcement learning: One potential future research direction using MTL R-CNN and Reinforcement Learning (RL) could be in the field of autonomous

driving. Autonomous driving is a complex task that requires the vehicle to perform multiple tasks simultaneously, such as object detection, path planning, and decision-making.

The research could focus on developing a MTL R-CNN network that can perform multiple tasks simultaneously using RL [66, 67]. The model could be trained on a large autonomous vehicle dataset like BDD100K, which comprise of different driving scenarios, such as different road conditions, traffic patterns, and weather conditions. The tasks would include object detection, lane detection, obstacle avoidance, and decision-making.

Like in [66, 67], the reward function for the RL algorithm could be designed to encourage safe and efficient driving. For example, the model could receive a higher reward for avoiding collisions, staying in the correct lane, and reaching the destination quickly. The model architecture could be designed to take advantage of the shared information between tasks. For example, the object detection task could provide information to the obstacle avoidance and decision-making tasks, improving their performance.

The potential benefits of this research include improving the safety and efficiency of autonomous driving systems, reducing the need for human intervention, and increasing the adoption of autonomous vehicles. The research could also have broader applications in robotics and other domains where multiple tasks need to be performed simultaneously.

- Since our system performs object detection, it is pertinent to evaluate its performance on established benchmark datasets such as the COCO dataset, the PASCAL VOC dataset, or the ImageNet dataset. These datasets encompass a diverse range of objects, and comprehensive annotations, including bounding boxes, segmentation masks, and key points, are provided for each object.

  By subjecting the system to evaluation on these datasets, researchers can gain insights into its ability to accurately detect and classify objects under diverse conditions, such as variations in scale, orientation, and occlusion. Furthermore, the performance of the system can be compared with that of other state-of-the-art models, enabling the identification of areas that require improvement.

  The evaluation of the systems using benchmark datasets plays a critical role in advancing the field of computer vision. It allows for a standardized comparison of the performance of different models and facilitates the identification of strengths and weaknesses of individual models under different conditions. Ultimately, such evaluations help to foster the improvement of the proposed system.

Bibliography

[1] K.-H. Thung and C.-Y. Wee, "A brief review on multi-task learning," *Multimedia Tools and Applications*, vol. 77, no. 22, pp. 29705–29725, 2018.

[2] D.-G. Lee and Y.-K. Kim, "Joint semantic understanding with a multilevel branch for driving perception," *Applied Sciences*, vol. 12, no. 6, p. 2877, 2022.

[3] C. Han, Q. Zhao, S. Zhang, Y. Chen, Z. Zhang, and J. Yuan, "Yolopv2: Better, faster, stronger for panoptic driving perception," *arXiv preprint arXiv:2208.11434*, 2022.

[4] K. Ishihara, A. Kanervisto, J. Miura, and V. 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*, pp. 2902–2911, 2021.

[5] P. Bhargava, "On generalizing detection models for unconstrained environments," in *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pp. 0–0, 2019.

[6] T. Deng and Y. Wu, "Simultaneous vehicle and lane detection via mobilenetv3 in car following scene," *Plos one*, vol. 17, no. 3, p. e0264551, 2022.

[7] A. Karpathy, "Multi-task learning in the wilderness," *ICML. url: https://slideslive.com/38917690/multitask-learning-in-the-wilderness*, 2019.

[8] R. Caruana, "Multitask learning," *Machine learning*, vol. 28, no. 1, pp. 41–75, 1997.

[9] Y. Zhang and Q. Yang, "A survey on multi-task learning," *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[10] A. Argyriou, T. Evgeniou, and M. Pontil, "Convex multi-task feature learning," *Machine learning*, vol. 73, no. 3, pp. 243–272, 2008.

[11] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert, "Cross-stitch networks for multi-task learning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3994–4003, 2016.

[12] S. Ruder, J. Bingel, I. Augenstein, and A. Søgaard, "Latent multi-task architecture learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 4822–4829, 2019.

[13] Y. Gao, J. Ma, M. Zhao, W. Liu, and A. L. Yuille, "Nddr-cnn: Layerwise feature fusing in multi-task cnns by neural discriminative dimensionality reduction," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 3205–3214, 2019.

[14] S. Liu, E. Johns, and A. J. Davison, "End-to-end multi-task learning with attention," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 1871–1880, 2019.

[15] O. Sener and V. Koltun, "Multi-task learning as multi-objective optimization," *Advances in neural information processing systems*, vol. 31, 2018.

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

[17] A. Kendall, Y. Gal, and R. Cipolla, "Multi-task learning using uncertainty to weigh losses for scene geometry and semantics," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7482–7491, 2018.

[18] M. Long, Z. Cao, J. Wang, and P. S. Yu, "Learning multiple tasks with multilinear relationship networks," *Advances in neural information processing systems*, vol. 30, 2017.

[19] Y. Lu, A. Kumar, S. Zhai, Y. Cheng, T. Javidi, and R. Feris, "Fully-adaptive feature sharing in multi-task networks with applications in person attribute classification," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5334–5343, 2017.

[20] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "Scaled-yolov4: Scaling cross stage partial network," in *Proceedings of the IEEE/cvf conference on computer vision and pattern recognition*, pp. 13029–13038, 2021.

[21] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

[22] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, pp. 2980–2988, 2017.

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, no. 9, pp. 1904–1916, 2015.

[24] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, "Path aggregation network for instance segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8759–8768, 2018.

[25] D. Wu, M.-W. Liao, W.-T. Zhang, X.-G. Wang, X. Bai, W.-Q. Cheng, and W.-Y. Liu, "Yolop: You only look once for panoptic driving perception," *Machine Intelligence Research*, pp. 1–13, 2022.

[26] D. Vu, B. Ngo, and H. Phan, "Hybridnets: End-to-end perception network," *arXiv preprint arXiv:2203.09035*, 2022.

[27] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," *arXiv preprint arXiv:2207.02696*, 2022.

[28] F. Codevilla, E. Santana, A. M. López, and A. Gaidon, "Exploring the limitations of behavior cloning for autonomous driving," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9329–9338, 2019.

[29] Z. Li, T. Motoyoshi, K. Sasaki, T. Ogata, and S. Sugano, "Rethinking self-driving: Multi-task knowledge for better generalization and accident explanation ability," *arXiv preprint arXiv:1809.11100*, 2018.

[30] Q. Fan, W. Zhuo, C.-K. Tang, and Y.-W. Tai, "Few-shot object detection with attention-rpn and multi-relation detector," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4013–4022, 2020.

[31] L. Cultrera, L. Seidenari, F. Becattini, P. Pala, and A. Del Bimbo, "Explaining autonomous driving by learning end-to-end visual attention," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 340–341, 2020.

[32] X. Huang, X. Cheng, Q. Geng, B. Cao, D. Zhou, P. Wang, Y. Lin, and R. Yang, "The apolloscape dataset for autonomous driving," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 954–960, 2018.

[33] J. Howard and S. Ruder, "Universal language model fine-tuning for text classification," *arXiv preprint arXiv:1801.06146*, 2018.

[34] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," *Advances in neural information processing systems*, vol. 27, 2014.

[35] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, pp. 2961–2969, 2017.

[36] H. Almubarak, Y. Bazi, and N. Alajlan, "Two-stage mask-rcnn approach for detecting and segmenting the optic nerve head, optic disc, and optic cup in fundus images," *Applied Sciences*, vol. 10, no. 11, p. 3833, 2020.

[37] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[38] M. Wu, H. Yue, J. Wang, Y. Huang, M. Liu, Y. Jiang, C. Ke, and C. Zeng, "Object detection based on rgc mask r-cnn," *IET Image Processing*, vol. 14, no. 8, pp. 1502–1508, 2020.

[39] G. Ciaparrone, F. Bardozzo, M. D. Priscoli, J. L. Kallewaard, M. R. Zuluaga, and R. Tagliaferri, "A comparative analysis of multi-backbone mask r-cnn for surgical tools detection," in *2020 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, 2020.

[40] H. M. A. Bhatti, J. Li, S. Siddeeq, A. Rehman, and A. Manzoor, "Multi-detection and segmentation of breast lesions based on mask rcnn-fpn," in *2020 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pp. 2698–2704, IEEE, 2020.

[41] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[42] A. Neubeck and L. Van Gool, "Efficient non-maximum suppression," in *18th International Conference on Pattern Recognition (ICPR'06)*, vol. 3, pp. 850–855, IEEE, 2006.

[43] X. Wan, Y. Wu, and X. Li, "Learning robust shape-indexed features for facial landmark detection," *Applied Sciences*, vol. 12, no. 12, p. 5828, 2022.

[44] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[45] M. Zhou, J. Wang, and B. Li, "Arg-mask rcnn: An infrared insulator fault-detection network based on improved mask rcnn," *Sensors*, vol. 22, no. 13, p. 4720, 2022.

[46] A. Mahmoud, S. Mohamed, R. El-Khoribi, and H. AbdelSalam, "Object detection using adaptive mask rcnn in optical remote sensing images," *Int. J. Intell. Eng. Syst*, vol. 13, no. 1, pp. 65–76, 2020.

[47] F. Yu, H. Chen, X. Wang, W. Xian, Y. Chen, F. Liu, V. Madhavan, and T. Darrell, "Bdd100k: A diverse driving dataset for heterogeneous multitask learning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 2636–2645, 2020.

[48] F. Hou, W. Lei, S. Li, J. Xi, M. Xu, and J. Luo, "Improved mask r-cnn with distance guided intersection over union for gpr signature detection and segmentation," *Automation in Construction*, vol. 121, p. 103414, 2021.

[49] R. Katırcı, E. K. Yılmaz, O. Kaynar, and M. Zontul, "Automated evaluation of cr-iii coated parts using mask rcnn and ml methods," *Surface and Coatings Technology*, vol. 422, p. 127571, 2021.

[50] K. B. Antonelli and C. C. Williams, "Task-relevant perceptual features can define categories in visual memory too," *Memory & Cognition*, vol. 45, no. 8, pp. 1295–1305, 2017.

[51] S. Chen, A. Kocsis, H. R. Liesefeld, H. J. Müller, and M. Conci, "Object-based grouping benefits without integrated feature representations in visual working memory," *Attention, Perception, & Psychophysics*, vol. 83, no. 3, pp. 1357–1374, 2021.

[52] B. Krawczyk, "Learning from imbalanced data: open challenges and future directions," *Progress in Artificial Intelligence*, vol. 5, no. 4, pp. 221–232, 2016.

[53] H. Patel, D. Singh Rajput, G. Thippa Reddy, C. Iwendi, A. Kashif Bashir, and O. Jo, "A review on classification of imbalanced data for wireless sensor networks," *International Journal of Distributed Sensor Networks*, vol. 16, no. 4, p. 1550147720916404, 2020.

[54] G. S. Na, "Efficient learning rate adaptation based on hierarchical optimization approach," *Neural Networks*, vol. 150, pp. 326–335, 2022.

[55] Z. Meng, X. Yao, and L. Sun, "Multi-task distillation: Towards mitigating the negative transfer in multi-task learning," in *2021 IEEE International Conference on Image Processing (ICIP)*, pp. 389–393, IEEE, 2021.

[56] W. Zhang, L. Deng, L. Zhang, and D. Wu, "A survey on negative transfer," *arXiv preprint arXiv:2009.00909*, 2020.

[57] A. Gulli and S. Pal, *Deep learning with Keras*. Packt Publishing Ltd, 2017.

[58] V. Tiwari, S. Gupta, P. Roy, C. Karda, S. Agrawal, M. S. Rathore, S. Jain, and A. Pal, "Soybean crop non-beneficial insect identification using mask rcnn," in *Information and Communication Technology for Competitive Strategies (ICTCS 2020)*, pp. 301–311, Springer, 2022.

[59] Z. Xie, I. Sato, and M. Sugiyama, "Stable weight decay regularization," 2020.

[60] A. Krogh and J. Hertz, "A simple weight decay can improve generalization," *Advances in neural information processing systems*, vol. 4, 1991.

[61] J. Woo, J.-H. Baek, S.-H. Jo, S. Y. Kim, and J.-H. Jeong, "A study on object detection performance of yolov4 for autonomous driving of tram," *Sensors*, vol. 22, no. 22, p. 9026, 2022.

[62] R. Ikeda and A. Hidaka, "Improvement of on-road object detection using inter-region and intra-region attention for faster r-cnn," in *Frontiers of Computer Vision: 28th International Workshop, IW-FCV 2022, Hiroshima, Japan, February 21–22, 2022, Revised Selected Papers*, pp. 211–226, Springer, 2022.

[63] Q. Xu, Y. Zeng, W. Tang, W. Peng, T. Xia, Z. Li, F. Teng, W. Li, and J. Guo, "Multi-task joint learning model for segmenting and classifying tongue images using a deep neural network," *IEEE journal of biomedical and health informatics*, vol. 24, no. 9, pp. 2481–2489, 2020.

[64] Y. Ouali, C. Hudelot, and M. Tami, "An overview of deep semi-supervised learning," *arXiv preprint arXiv:2006.05278*, 2020.

[65] X. Yang, Z. Song, I. King, and Z. Xu, "A survey on deep semi-supervised learning," *arXiv preprint arXiv:2103.00550*, 2021.

[66] S. Sodhani, A. Zhang, and J. Pineau, "Multi-task reinforcement learning with context-based representations," in *International Conference on Machine Learning*, pp. 9767–9779, PMLR, 2021.

[67] K.-H. Lee, T. Xiao, A. Li, P. Wohlhart, I. Fischer, and Y. Lu, "Pi-qt-opt: Predictive information improves multi-task robotic reinforcement learning at scale," *arXiv preprint arXiv:2210.08217*, 2022.

[68] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[69] M.-L. Zhang and Z.-H. Zhou, "A review on multi-label learning algorithms," *IEEE transactions on knowledge and data engineering*, vol. 26, no. 8, pp. 1819–1837, 2013.

[70] Y. LeCun, L. D. Jackel, L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, U. A. Muller, E. Sackinger, P. Simard, *et al.*, "Learning algorithms for classification: A comparison on handwritten digit recognition," *Neural networks: the statistical mechanics perspective*, vol. 261, no. 276, p. 2, 1995.

[71] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[72] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[73] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*, pp. 818–833, Springer, 2014.

[74] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[75] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[76] Z. Wu, C. Shen, and A. Van Den Hengel, "Wider or deeper: Revisiting the resnet model for visual recognition," *Pattern Recognition*, vol. 90, pp. 119–133, 2019.

[77] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587, 2014.

[78] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448, 2015.

[79] J. Dai, Y. Li, K. He, and J. Sun, "R-fcn: Object detection via region-based fully convolutional networks," *Advances in neural information processing systems*, vol. 29, 2016.

[80] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2117–2125, 2017.

[81] Z. He and L. Zhang, "Multi-adversarial faster-rcnn for unrestricted object detection," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 6668–6677, 2019.

[82] R. Kukreja, S. Rinchen, B. Vaidya, and H. T. Mouftah, "Evaluating traffic signs detection using faster r-cnn for autonomous driving," in *2020 IEEE 25th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, pp. 1–6, IEEE, 2020.

[83] D. Wang and D. He, "Fusion of mask rcnn and attention mechanism for instance segmentation of apples under complex background," *Computers and Electronics in Agriculture*, vol. 196, p. 106864, 2022.

[84] V. Mistry, S. Rinchen, B. Vaidya, and H. T. Mouftah, "Investigating drivable space instance segmentation for connected and autonomous vehicles," in *2022 International Wireless Communications and Mobile Computing (IWCMC)*, pp. 437–442, IEEE, 2022.

APPENDIX A

Appendix

A.1 Paradigms Related to MTL

Multi-task learning (MTL) is a machine learning paradigm that aims to improve the performance of multiple related tasks simultaneously by exploiting the shared representations and correlations among them. In MTL, all tasks are treated equally, and the objective is to improve performance on all tasks by sharing information and knowledge across tasks. MTL has been widely used in various fields such as computer vision, natural language processing, and bioinformatics, among others.

It is often confused with other machine learning paradigms such as transfer learning [68] multi-label learning [69], and multi-view learning. However, there are significant differences between MTL and these other paradigms that are worth noting.

In transfer learning, the objective is to improve the performance of a specific target task with the help of one or more source tasks. This is done by transferring the knowledge learned from the source tasks to the target task. The knowledge flow in transfer learning is unidirectional, flowing from the source task(s) to the target task. In contrast, in MTL, there is no distinction between different tasks and the objective is to improve the performance of all tasks. The knowledge flow in MTL is bidirectional, flowing between any pair of tasks.

Multi-view learning is another machine learning paradigm that is different from MTL, where each data point is associated with multiple views each of which consists of a set of features. Even though different views have different sets of features, all the views are used together to learn for the same task, and hence multi-view learning belongs to single-task learning with multiple sets of features, which is

different from MTL.

Multi-label learning is different from MTL as well because in this paradigm each data point is associated with multiple labels which can be categorical or numeric. If they treat each of all the possible labels as a task, multi-label learning and multi-output regression can be viewed in some sense as a special case of multi-task learning where different tasks always share the same data during both the training and testing phases. Such characteristic in multi-label learning and multi-output regression leads to different research issues from MTL. For example, the ranking loss, which enforces the scores of labels associated with a data point to be larger than those of absent labels, can be used for multi-label learning but it does not fit MTL where different tasks possess different data. On the other hand, this characteristic in multi-label learning and multi-output regression is invalid in MTL problems. For example, in an MTL problem where each task is to predict the disease symptom score of cancer for a patient based on multiple bio-medical features, different patients/tasks should not share the bio-medical data.

In summary, MTL is a machine learning paradigm that aims to improve the performance of multiple related tasks simultaneously by exploiting the shared representations and correlations among them. It is different from transfer learning, multi-view learning, and multi-label learning, as these paradigms have different objectives and characteristics. Understanding the distinction between MTL and other machine learning paradigms is crucial for the correct application and development of machine learning models.

A.2 Deep Convolutional Neural Networks

Deep Convolutional Neural Networks, or CNNs, have proven to be highly effective in a variety of applications that are related to computer and machine vision. Image Classification and Segmentation, Object Detection, Active Recognition, Video Processing, and Natural Language Processing are some of the application fields that CNN can be used for. Other application fields include Video Processing. The utilization of a large number of feature extraction stages that are able to automatically learn representations based on the data is the primary reason why deep CNN is capable of such tremendous learning. The accessibility of vast amounts of data as well as the development of new hardware technologies has both contributed to the acceleration of research on CNNs. Recent years have seen the discovery of extremely effective deep CNN architectures, which have also proven to be beneficial to the field of object recognition research.

A.2.1 Convolutional neural networks (CNN)

In recent years, deep convolutional neural networks (DEEP CNNs) have become a crucial component of many computer vision applications. This section examines the core CNN components and representative CNN designs that are frequently used as the backbone networks in the object identification pipeline.

ResNet and VGG, among others, are examples of these designs. The accuracy of a detector is strongly reliant on the network for feature extraction.

A.2.2 Basic CNN components

The structure of a CNN consists of convolutional layers, non-linear processing units, and subsampling layers. Convolutional neural networks (CNNs) are a kind of feedforward multilayered hierarchical network in which each layer employs a bank of convolutional kernels to perform multiple transformations. The Convolution 6 procedure simplifies the extraction of important features from locally related data. By allocating the output of the convolutional kernels to a nonlinear processing unit, nonlinearity is included in the feature space (activation function). This nonlinearity, which creates unique activation patterns for varied responses, facilitates the learning of semantic distinctions between pictures. Subsampling is often used after the output of a non-linear activation function to help in summarising the findings and make the input insensitive to geometrical distortions. Using a CNN reduces the requirement for a specialist feature extractor due to the network's intrinsic ability to extract features automatically. This allows a CNN to acquire accurate internal representations from raw pixels. Essential components of a CNN include hierarchical learning, autonomous feature extraction, parallel processing, and weight sharing.

In a CNN's design, convolution and pooling layers are often arranged one after the other, followed by one or more fully linked layers. In certain implementations, a global average pooling layer is used as opposed to a completely connected layer. In addition to their array of mapping functions, they apply a variety of regulatory units, such as batch normalization and dropout, to improve CNN's functionality. LeNet-5 is a straightforward illustration of CNN's fundamental architecture.

A.2.3 Convolutional Layer

A convolutional layer consists of a collection of convolutional kernels, each of which carries out its function by convolutionally slicing the input image into a number of discrete areas that are referred to as receptive fields. A convolutional layer is one of the layers that can be found in a neural network. The photos are convolved using a kernel that has a specified set of weights, which allows for the production of a wide range of feature maps. This is performed by multiplying the components of the kernel by the elements in the picture receptive field that are a match for those elements in the kernel. The use of the convolution process is associated with the following three key benefits:

1. The weight-sharing technique in the same feature map can reduce the number of parameters
2. Local connection can learn correlations among surrounding pixels
3. Invariance to the position of the object can be achieved

A.2.4 Pooling Layer

A pooling layer is often inserted after a convolutional layer in a neural network. This layer may cut down on the amount of space required for feature maps as well as network parameters. Once the features have been retrieved, the exact location of those features with respect to the other features is of less importance since the estimated position of those features is kept. The pooling approach is effective for obtaining a combination of characteristics that are immune to translational shifts and tiny distortions since its calculation takes into consideration neighboring pixels. This makes the pooling technique one of the most useful image-processing techniques. This is inevitable given the characteristics of the pooling process. It gathers information of the same general category as that which surrounds the receptive field and then produces the response that is most prevalent in this neighborhood. Keeping the size of the feature map constrained to an invariant feature set is one way to preserve the network's inherent complexity. This not only improves generalization, but it also improves generalization by reducing overfitting, which is a common pitfall in data analysis. CNN makes use of a variety of pooling formulations, some of which include (but are not limited to) maximum pooling, average pooling, spatial pyramid pooling, and others.

A.2.5 Activation Function

One way to think of the activation function is as a choice function that helps in the learning of complicated pattern knowledge. The pace of the learning process may be increased by selecting an appropriate activation function. In order to generate non-linear combinations of features, various activation functions such as sigmoid, tanh, rectified linear unit (ReLU), and variants of ReLU such as leaky ReLU, exponential linear unit (ELU), and parametric rectified linear unit (PreLU) are utilized. Other activation functions include the following: exponential linear unit (ELU), and parametric rectified linear unit (PreLU). On the other hand, ReLU and its derivatives are selected because of their capability of avoiding the gradient issue. During the gradient descent training of machine learning algorithms, you could run across an issue known as the vanishing gradient problem. The use of partial derivatives in order to calculate the gradient becomes an increasingly important characteristic as the layer is buried deeper inside the network. Since gradients dictate how much a network learns during training if gradients are extremely tiny or zero, very little or no training may occur, which can result in poor predicting performance. This is because gradients control how much a network learns during training.

A.2.6 Batch Normalization

Using batch normalization as a remedy for internal covariance shifts inside feature maps is one way to correct the issue. Convergence is slowed down as a result of the internal covariance shift, and careful setting of parameters is required because of this change in the distribution of values for hidden units. The distribution of feature map values may be unified using batch normalization, which has the benefit

of setting them to zero means and one standard deviation each. In addition to this, it controls the gradient flow and smoothes its flow, which together contributes to an increased degree of generalization throughout the network.

A.2.7 Dropout

Dropout contributes to the establishment of regularisation within the network, which ultimately helps to improve generalization. This is accomplished by arbitrarily skipping certain units or connections with a probability that has been calculated in advance. This method of randomly removing certain connections or units from the network provides a variety of different topologies for the network, from which a representative network with low weights is selected. Because of this, the chosen design is considered to be a reasonably accurate representation of all of the suggested networks.

A.2.8 Fully Connected Layer

After the very last layer of pooling and at the very end of a network is the normal location for the implementation of a fully linked layer for the purposes of classification. In contrast to convolution and pooling, this process has a global scope. It does a global analysis of the results of analyzing the output of all the layers that came before it and gets input from later levels of feature extraction. It accounts for something in the neighborhood of ninety percent of a CNN's parameters. It generates a non-linear combination of the chosen characteristics that are required for the data classification process.

A.3 Evolution of CNN backbone architectures

With the advent of deep learning came refinements to CNN's learning technique and architecture that made it more adaptable to big, heterogeneous, complicated, and multiclass issues. In this part, you will learn about the fundamentals of many different CNN backbone designs.

A.3.1 LeNet

With their proposal of LeNet, LeCun et al [70]. created the first CNN architecture to achieve state-of-the-art performance on a hand-written digit recognition test. Five layers of convolution and pooling, followed by two fully connected layers, make up the feed-forward neural network known as LeNet-5 [71]. LeNet-5 made use of the premise that neighboring pixels are associated with one another and spread uniformly over an image as the foundation for its representational model. So, it changed the conventional view of training in which each pixel was considered a separate input feature from its neighborhood, and the correlation between them was ignored, revealing that convolution with learnable parameters is an efficient method for extracting similar features at multiple locations with a small number of parameters.

LeNet-5 shows that a CNN can not only learn features automatically from raw pixels but also reduce the number of parameters.

A.3.2 AlexNet

When it comes to picture categorization, AlexNet was the first convolutional neural network (CNN) developed by Krizhevesky et al. [72] LeNet was the first deep CNN, however, it could only recognize hand-written digits and performed badly with any kind of picture. AlexNet was a major step forward for picture classification and recognition applications since its learning capacity was increased by elongating the CNN and adopting various parameter optimization algorithms. To broaden CNN's applicability to other types of pictures, the depth of AlexNet was extended from 5 (LeNet) to 8 (AlexNet) layers, and big-size filters were used in the network's early levels (11 x 11 and 5 x 5). In addition, the convergence rate was enhanced by using a nonsaturating activation function in the form of a rectified linear unit (ReLU) to help with the vanishing gradient issue. Overfitting was decreased with overlapping subsampling and local response normalization to enhance generalization. Furthermore, AlexNet was the first convolutional neural network (CNN) to be trained using GPUs (GPUs).

A.3.3 ZFNet

ZFNet [73] is an abbreviation that refers to Zeiler and Fergus networks. It is a multilayer Deconvolutional Neural Network (DeconvNet) that was introduced by Zeiler and Fergus in the year 2013. The provision of accurate and quantitative visualization of network performance was the motivation for the development of ZDNet. The DeconvNet network performs operations that are similar to those of the forward pass CNN; however, the order in which its convolutional and pooling processes occur is switched around. This back-projection mapping interprets the learned feature representation internally at the neuron 10 level by projecting the output of a convolutional layer back to visually observable image patterns. Back-projection mapping is a technique that was developed by Google DeepMind. While keeping the same number of features in CNN's first two convolutional layers, Zeiler and Fergus were able to gain the maximum potential learning from the network by reducing the filter size and the stride. This allowed them to achieve their goal of maximizing the network's learning potential. This increase in performance after changing the CNN topology showed that feature visualization may be used to identify design flaws and enable quick parameter modification.

A.3.4 VGG

In 2014, Simonyan and Zisserman presented their proposal for a strong CNN architecture, which they referred to as VGG [74]. Its depth was far larger than that of AlexNet and ZDNet, both of which had 8 layers since it had 19 levels. This was done in order to approximate the link between the network's depth

and its representational capability. As an example, VGG carried out an experiment to demonstrate that the simultaneous application of a huge number of small-size (3x3) filters may have the same impact as a single big-size filter. In order to demonstrate this, a set of 3x3 filters, rather than 11x11 or 5x5 ones, was employed in an experiment, which was then documented (5x5 and 7x7). It is important not to undervalue the computational gains that come from using tiny filters. They accomplish this objective by lessening the number of adjustable parameters that each of the individual filters has. VGG also did quite well when it comes to categorizing photographs and determining where they were located geographically. The 2014 ImageNet Large Scale Visual Recognition Challenge (ILSVRC) was won by ResNet-50; nevertheless, VGG's popularity skyrocketed as a direct result of the competition due to its better depth, simplicity of use, and consistent topology. This was the case. The most significant drawback associated with using VGG is the need for a staggering 138 million parameters. It is difficult to implement on low-resource systems because of the significant computational cost associated with it.

A.3.5 GoogleNet

2014 marked the beginning of Google Inc.'s development of the model family that they refer to as the GoogleNet CNN model. "Inception"[75] is another term that's been used for this particular series of models. It was shown that it is possible to lessen the amount of work that must be done on the computer while maintaining the same level of precision. In the field of convolutional neural networks, which are more often referred to as CNNs, GoogleNet was the first to pioneer the introduction of the inception block. This part of the code uses filters of several sizes (1x1, 3x3, and 5x5) to gather spatial data at fine-grained and coarse-grained resolutions, respectively. GoogleNet was created to handle the difficulty of learning the various variations that occur within a particular category of photographs but across a variety of resolutions. It achieves this by making use of the principles of divide, transform, and merge. In order to do this, many photographs were blended into a single image. Before it employs large-sized kernels, GoogleNet inserts a bottleneck layer that is comprised of a 1x1 convolution filter. This layer places a cap on the number of computations that may be performed. In addition, the sparse connections helped in addressing the problem of duplicate data and saved money by deleting unneeded feature maps. This was made possible because of the money saved from not having to create those maps. Why? mainly because not every potential input feature map is related to every single possible output feature map. The most significant step forward that the Inception family took was the development of the factorizing convolution and batch normalization algorithms. They were able to see a considerable improvement as a direct consequence of modifying these factors.

A.3.6 ResNet

In 2015, He et al. presented ResNet [44], which introduced residual learning in CNNs and an effective approach for training deep neural networks. ResNet's residual block was twenty and eight times deeper than those of AlexNet and VGG, respectively. However, its computational complexity was smaller than that of previously suggested networks. The 2015-ILSVRC champion was ResNet with 152 layers. The exceptional performance of ResNet on picture identification and localization challenges shows that depth is crucial to many visual recognition tasks. ResNet's disadvantage is that it explicitly retains information via additive identity transformations, which may result in many layers contributing little or no information. Like this research [76], ResNet is widely used and researched in computer vision tasks.

A.4 CNN Based Two-Stage Object Detectors

A.4.1 RCNN

The process of the first Region-based Convolutional Neural Network (R-CNN) [77], which was shown to the public for the very first time in 2014, begins with the extraction of a collection of object suggestions via the use of selective search (SS). After that, each recommendation is shrunk down to a certain size and then put into a CNN model that has been trained on the ImageNet dataset (such as AlexNet) in order to extract features from the images. Last but not least, linear support vector machine (SVM) classifiers are used in order to distinguish item categories and to create predictions about the existence of objects inside each area. These predictions are made using the data collected from the regions. When used with the PASCAL-VOC07 dataset, R-CNN generates performance that is noticeably better. R-CNN has made significant progress, but it still has significant flaws, the most obvious of which is that it performs redundant feature computation on a large number of overlapped proposals (over 2000 boxes from one image), which results in an extremely sluggish detection speed. Although R-CNN has made significant advancements, it still has significant flaws (14s per image with GPU).

A.4.2 SPPNet

Later on in the same year, He and his colleagues came up with the idea of using Spatial Pyramid Pooling Networks (SPPNet) [23] to solve this problem. The key improvement made by SPPNet is the incorporation of a Spatial Pyramid Pooling (SPP) layer between the last two convolutional layers. It does away with the network's fixed-size limitation, making it possible for a CNN to construct a representation of a given length regardless of the size of the picture or the area of interest being processed. In contrast, earlier CNNs need inputs of a certain size, such as a picture of 224 by 224 pixels for AlexNet. The feature maps may be calculated from the whole of the picture just once when using SPPNet for

object identification. After that, fixed-length representations of random sections can be constructed for training the detectors, which eliminates the need to repeatedly compute the convolutional features. When compared to R-CNN's detection speed, SPPNet's is almost 20 times quicker without compromising accuracy. Despite the fact that SPPNet has significantly improved the detection time, there are still certain restrictions. To begin, there are still numerous levels to the training process. Second, SPPNet ignores all of the layers that came before it and only makes adjustments to the layers that are completely linked.

A.4.3 Fast RCNN

In 2015, Girshick presented the Fast R-CNN [78], which is an extra enhancement of both the R-CNN and the SPPNet algorithms. He did so in order to speed up the processing time of both models. It was only able to train a detector and a bounding box regressor concurrently using the same network setup because FastR-CNN made it possible. It boosted the speed of detection to almost 200 times quicker than what R-CNN was able to achieve. Even though Fast R-CNN was able to effectively combine the advantages of R-CNN and SPPNet, the speed of detection is still limited by the proposal detection. This is despite the fact that Fast R-CNN was able to successfully integrate the benefits of both networks.

A.4.4 Faster RCNN

In the same year, Ren et al. came up with the idea for the Faster R-CNN detector [41]. It is the first near-real-time deep learning detector and the first end-to-end deep learning detector. The invention of a Region Proposal Network, also known as an RPN, which makes use of a CNN model to create object suggestions is the fundamental contribution provided by Faster R-CNN. From R-CNN to Faster R-CNN, the bulk of the different building blocks of an object identification system has been progressively merged into a cohesive, end-to-end learning framework. These building blocks include proposal detection, feature extraction, bounding box regression, and so on. This procedure was first started using R-CNN, and it is now being carried out with Faster R-CNN. Even though Faster R-CNN does away with the detection speed bottleneck that afflicted Fast R-CNN, there is still computation duplication in the phases of detection that follow.

A.4.5 R-FCN

The concept of region-based fully convolutional networks (R-FCN) was proposed by Dai et al. in 2016 [79]. This was done to avoid the requirement to execute the time-consuming and costly region of interest (RoI)-wise subnetwork in Faster-R-CNN hundreds of times, which is equivalent to running it once for each proposal. R-FCN came up with the concept of position-sensitive feature maps as their contribution to the conversation. The production of scores for a specific region of the target class, such as the top left, center,

or bottom right, etc., is the responsibility of each feature map on its own in this approach. For example, the top left, the center, or the bottom right. The components are differentiated from one another by employing RoI-Pooling cells, which are scattered throughout a specific feature map near each individual component. Calculating the average number of votes obtained for each component of the RoI while applying the proper filter is what's used to get the final scores. By using this implementation strategy, a little amount of extra translational variation was introduced into structures that, as a result of their development, had translation invariance for the most part. The fluctuation in object recognition that takes place throughout the process of translation might help gain an understanding of the representations used in localization. Although this pipeline provides the idea of being more exact, in terms of performance, it is not necessarily better than its Faster R-CNN cousin. The total inference time speed of two-stage detectors was able to be improved thanks to this technology, even though it had somewhat worse performance.

A.4.6 FPN

Lin et al. presented the concept of Feature Pyramid Networks (FPN) [80] in 2017, which was conceived on the foundation of Faster R-CNN. Before the invention of FPN, the great majority of object detectors that were based on deep learning could only perform detection on the very first layer of a network. Even though the characteristics in deeper levels of a CNN are beneficial for category 19 classification, it is not favorable to localizing objects. This is because the deeper layers of a CNN include more information. The FPN project has designed a top-down architecture with lateral connections to develop high-level semantics at all sizes to achieve this objective. In doing so, the project hopes to realize its overall objective. Because a CNN will always generate a feature pyramid as a consequence of its forward propagation, the FPN displays tremendous gains when it comes to the identification of objects that cover a broad range of scales. This is the case because the CNN will naturally construct the feature pyramid. By incorporating FPN into a foundational Faster R-CNN system, it can achieve single model identification results on the MS-COCO dataset that are considered to be state-of-the-art. FPN has evolved into a vital component that is used in the assembly of many of the most modern detectors. FPN-based Faster R-CNN is currently widely used in various object detection types of research like these [81, 82].

A.4.7 Mask RCNN

The Faster R-CNN serves as the basis for the Mask R-CNN [35], which expands upon it by adding a second branch that does pixel-level object instance segmentation in tandem with the first branch's operations. The branch is a fully linked network that is applied to the ROIs. Its objective is to segment each pixel while incurring the least amount of total compute cost as is humanly feasible. It uses an architecture for object proposal that is comparable to that of Faster R-CNN in its most fundamental form; however, in addition to the classification mask head and the bounding box regressor head, it also

has a parallel mask head that is used for object proposal. The RoIAlign layer, rather than the RoIPool layer, is employed in place of the RoIPool layer to avoid a misalignment at the pixel level that would be induced by spatial quantization. This is one of the most important distinctions between the two. The authors concluded that the ResNeXt-101 [48] would serve as the backbone of their system and would be combined with the feature Pyramid Network (FPN) to achieve greater accuracy and throughput. In a way that is comparable to that of FPN, the loss function of Faster R-CNN has been modernized to include mask loss, and it employs five anchor boxes with an aspect ratio of three to one. In most cases, training with faster R-CNN or with Mask R-CNN produces results that are equivalent to one another. The performance of the Mask R-CNN was much better than that of the most advanced single-model architectures that are currently in use as shown in these studies [83, 84]. In addition to this, it introduced the feature of instance segmentation while imposing just a little extra demand on the computing resources. It is quick to train, versatile, and generalizes well in applications such as keypoint recognition and human position estimation, amongst others. Additionally, it is simple to train. Despite this, it is slower than the performance in real-time, which is more than 30 frames per second.