UNIVERSITY OF OTTAWA

MASTER'S THESIS

Sensor Fusion for 3D Object Detection for Autonomous Vehicles

Author:
Yahya MASSOUD

Supervisor:
Dr. Robert LAGANIÈRE

A thesis submitted in fulfillment of the requirements
for the degree of Master of Computer Science

in the

VIVA Research Lab
School of Electrical Engineering and Computer Science

UNIVERSITY OF OTTAWA

Abstract

Engineering

School of Electrical Engineering and Computer Science

Master of Computer Science

Sensor Fusion for 3D Object Detection for Autonomous Vehicles

by Yahya MASSOUD

Thanks to the major advancements in hardware and computational power, sensor technology, and artificial intelligence, the race for fully autonomous driving systems is heating up. With a countless number of challenging conditions and driving scenarios, researchers are tackling the most challenging problems in driverless cars. One of the most critical components is the perception module, which enables an autonomous vehicle to "see" and "understand" its surrounding environment. Given that modern vehicles can have large number of sensors and available data streams, this thesis presents a deep learning-based framework that leverages multimodal data – i.e. sensor fusion, to perform the task of 3D object detection and localization.

We provide an extensive review of the advancements of deep learning-based methods in computer vision, specifically in 2D and 3D object detection tasks. We also study the progress of the literature in both single-sensor and multi-sensor data fusion techniques. Furthermore, we present an in-depth explanation of our proposed approach that performs sensor fusion using input streams from LiDAR and Camera sensors, aiming to simultaneously perform 2D, 3D, and Bird's Eye View detection. Our experiments highlight the importance of learnable data fusion mechanisms and multi-task learning, the impact of different CNN design decisions, speed-accuracy tradeoffs, and ways to deal with overfitting in multi-sensor data fusion frameworks.

"And in no way is my success with anyone except with Allah; in Him I have put my trust, and to Him I turn penitent."

Acknowledgements

First and foremost, all praises be to Allah for all His blessings upon me.

I would like to thank my supervisor and mentor, Professor Robert Laganière. I am humbled by his trust in my abilities and his confidence in my potential. His time, guidance, knowledge, kindness, and unconditional financial support, were key for me since my Day-1 at the VIVA Research Lab. Professor Robert has provided me with amazing research and industrial opportunities that would better position me in my career. I will always be grateful.

I would like to thank Dr. Ahmad Al-Kabbany for helping me taking the first steps in pursuing my graduate studies at the University of Ottawa, and for always encouraging and motivating me. I am glad that I know such an inspiring human being as Ahmad.

It has been a tough year and a half during the pandemic; the only thing that would make me want to keep going is a phone call with my family. I would like to thank my father, mother, both sisters, and both grandmothers, for always being there for me, for their unconditional emotional and financial support, for always cheering me up, and for being a kind and amazing family.

This thesis would have been much harder without the support of my friends and lab-mates. I thank Wassim Al-Ahmar for being a great friend, and for his continued support, which started even before I arrived in Canada. I thank Hamza Hajar for our great conversations, and our refreshing coffee breaks. I also thank Sadik Al-Nasif and Selameab Demilew.

I thank my friend Aly Gemae. It is very rare to meet such an inspiring and humble person, who is as successful as Aly.

I would like to thank Shehab El-Salamony, Mahmoud El-Tanbouli, Abdelrahman Ahmad, Loay Hafez, Ali Sobhy, Mohamed Essam, for their unmatched emotional support, and for being amazing friends.

My sincere thanks to Canada, and the University of Ottawa, for providing me with thousands of dollars, unlimited resources of knowledge, and a top-notch learning quality. Indeed, it was a life-changing experience.

Contents

List of Figures

List of Tables

List of Abbreviations

ADAM	**ADA**ptive **M**oment Estimation
AI	**A**rtificial **I**ntelligence
AP	**A**verage **P**recision
AV	**A**utonomous **V**ehicle
AVOD	**A**ggregated **V**iew **O**bject **D**etection
BEV	**B**ird's **E**ye **V**iew
BN	**B**atch **N**ormalization
CAN	**C**ontrol **A**rea **N**etwork
CE	**C**ross **E**ntropy
CIFAR	**C**anadian **I**nstitute **F**or **A**dvanced **R**esearch
CNN	**C**onvolutional **N**eural **N**etwork
COCO	**C**ommon **O**bject in **CO**ntext
CUDA	**C**omputer **U**nified **D**evice **A**rchitecture
CuDNN	**Cu**DA **D**eep **N**eural **N**etwork
CV	**C**omputer **V**ision
FCN	**F**ully **C**onnected **N**etwork
FoV	**F**ield **o**f **V**iew
FP	**F**alse **P**ositive
FPN	**F**eature **P**yramid **N**etwork
FPS	**F**rames **P**er **S**econd
FV	**F**rontal **V**iew
GD	**G**radient **D**escent
GN	**G**roup **N**ormalization
GPS	**G**lobal **P**ositioning **S**ystem
GPU	**G**raphical **P**rocessing **U**nit
GT	**G**round **T**ruth

ICS **I**nternal **C**ovariate **S**hift

ILSVRC **I**mageNet **L**arge **S**cale **V**isual **R**ecognition **C**hallenge

IMU **I**nertial **M**easurement **U**nit

IoU **I**ntersection **o**ver **U**nion

KITTI **K**arlsruhe **I**nstitute of **T**echnology and **T**oyota **T**echnological **I**nstitute

KNN **K**-**N**earest **N**eighbours

LiDAR **Li**ght **D**etection **A**nd **R**anging

MAE **M**ean **A**bsolute **E**rror

mAP **m**ean **A**verage **P**recision

MFB **M**ulti-modal **F**actorized **B**ilinear Pooling

MLB **M**ulti-modal **L**ow-rank **B**ilinear Pooling

MLP **M**ulti **L**ayer **P**erceptron

MSE **M**ean **S**quared **E**rror

MV3D **M**ulti **V**iew **3D** Object Detection

NHTSA **N**ational **H**ighway **T**raffic **S**afety **A**dministration

NLP **N**atural **L**anguage **P**rocessing

NMS **N**on **M**aximum **S**uppression

OS **O**perating **S**ystem

PASCAL **P**attern **A**nalysis, **S**tatistical Modeling, and **C**omputational **L**earning

PIXOR **OR**iented 3D object detection from **PIX**el-wise neural network predictions

PR **P**recision-**R**ecall

PRED **P**rediction

RADAR **RA**dio **D**etection **A**nd **R**anging

RCNN **R**egion-based **C**onvolutional **N**eural **N**etwork

ReLU **Re**ctified **L**inear **U**nit

RGB **R**ed **G**reen **B**lue

RGB-D **R**ed **G**reen **B**lue **D**epth

RMSProp **R**oot **M**ean **S**quare **Prop**agation

RoI **R**egion **o**f **I**nterest

RPN **R**egion **P**roposals **N**etwork

SAE **S**ociety of **A**utomotive **E**ngineers

SECOND	**S**parsely **E**mbedded **Con**volutional **D**etection
SOTA	**S**tate **O**f **T**he **A**rt
SSD	**S**ingle **S**hot **D**etector
SVM	**S**upport **V**ector **M**achines
Tanh	**Tan**gent **H**yperbolic
TP	**T**rue **P**ositive
UAT	**U**niversal **A**pproximation **T**heorem
VFE	**V**oxel **F**eature **E**ncoding
VGG	**V**isual **G**eometry **G**roup
ViT	**Vi**sion **T**ransformer
YOLO	**Y**ou **O**nly **L**ook **O**nce

Chapter 1

Introduction

We, humans, always have the impulse to seek new information, reach new heights, and explore new possibilities that people before us would label as "impossible" or "fictional". That's how we've been able to make breakthrough discoveries and remarkable inventions throughout history; starting from the industrial era, going through the electricity era, and leading to our information age, where everything is digitized and everyone is connected. Curiosity and problem solving are two fundamental human attributes that drove our exploration throughout history. We are witnessing now a new era that will transform our society, industry, and our future: the *Artificial Intelligence* era. Artificial Intelligence is often referred to as "*the new electricity*", as it is a transformative technology, just like how electricity was 100 years ago.

Artificial Intelligence is concerned with machines that can imitate or mimic human behaviour and actions in a set of tasks, and it is based on giving the machine the ability to "learn" from examples, or by try-and-error. The usage of AI fits best when it comes to tasks that fall in (but not limited to) these categories: (1) *redundant tasks*, where AI can be more productive and efficient, (2) *unsafe tasks*, where intelligent bots could be used to protect labour's safety, (3) *requires constant attention and concentration*, where AI can eliminate emerging human errors, as the case in driving.

Driving is a task that requires constant concentration, quick responses, and close attention. The National Highway Traffic Safety Administration (NHTSA) attributes 94% of car crashes to human errors [1], and this is where AI can be transformative. Autonomous vehicles (AV) can increase safety on the road, and reduce, or even eliminate errors, and can save a lot of lives. In fact, many governments are starting to

support the development and progress of the AV technology for more safety on the roads, and for a potential improvement in the overall traffic, as [2] predicts more than 250 million hours of consumers' commuting time per year will be freed in the most congested cities in the world.

As a result of the huge progress in the last ten years in the fields of: *artificial intelligence*, *hardware and sensor technology*, *compute and processing power*, *Internet of Things (IoT) and connectivity*, the self-driving vehicles technology emerges and attracts a large number of technology giants, startups, as well as researches all around the world. While self-driving technology is full of challenges and interesting research problems yet to be solved, it also presents a big business opportunity in the future, and a large number of innovative applications and services will emerge as self-driving technology evolves.

In this thesis, we present our work in advancing a critical task in autonomous vehicles: the perception system. More specifically, the task of detection, localization, and classification of objects and obstacles in the vehicle's surrounding environment by leveraging the power of sensor fusion techniques and deep learning algorithms.

1.1 Autonomous Vehicles

An autonomous vehicle (AV) is one that incorporates a fully automated driving system, that can drive and navigate on the road by itself, without requiring any human intervention, and can respond to external conditions and driving situations as a human would do. Most importantly, a fully automated driving system aims to eliminate human driving errors, which were categorized by [1]: *recognition errors* (e.g. lack of concentration, internal or external distractions), *decision-making errors* (e.g. false assumption of others' actions, driving too fast), *reflexive performance errors* (e.g. poor directional control), and *non-performance errors* (e.g. sleeping).

"Levels of Driving Automation" is an international standard developed by the Society of Automotive Engineers (SAE), and which categorizes vehicles' autonomy levels from Level 0 (human driver has full control) to Level 5 (fully automated driving system). Levels 1 and 2 require full supervision from the driver, while the driving assistant system may offer some support features. Level 3 does not require constant

supervision from the driver unless the system requests from the driver to take over. Levels 4 and 5 will not require the driver to take over driving at any time. The key difference between Levels 4 and 5, is that a Level 4 driving system can drive the vehicle only in the presence of limited conditions, while Level 5 autonomy will be able to drive the vehicle under all possible conditions.

In the next two subsections, we will provide an overview of the following: (1) the various functional modules incorporated within an autonomous vehicle (subsection 1.1.1), and (2) the different types of sensors used by an autonomous vehicle (subsection 1.1.2).

1.1.1 Autonomous Driving System's Functional Modules

For a vehicle's driving system to achieve full autonomy and eliminate human driving errors, such a system must incorporate several functional modules [3], and these modules can be classified as follows:

- **Sensing**: Leverages various types and categories of sensors to provide a reliable sensing of the vehicle's state and its surrounding environment.

- **Environment Perception**: Builds a contextual understanding of the surrounding driving environment of the vehicle through sensing. Perception relies on inputs coming from different types of sensors (e.g. camera, LiDAR, radar), and performs several critical tasks in real-time, such as: (1) detecting and tracking surrounding objects while predicting their distance from the car, and their corresponding three-dimensional shape and size, (2) detecting road signs, (3) detecting road lanes, and (4) detecting other vehicles' speed. The perception module leverages sophisticated computer vision algorithms intending to reduce human drivers' *recognition errors*.

- **Mapping and Localization**: The mapping module is responsible for constructing a map of the surrounding driving environment that contains information about roads, lanes, traffic rules, etc. The localization module determines the vehicle's position with respect to the driving environment. These two modules focus on reducing human drivers' *non-performance errors*.

- **Prediction**: Estimates the behaviours and trajectories of the objects that fall within the autonomous vehicle's surrounding environment. The prediction module contributes as well in the reduction of human drivers' *recognition errors*.

- **Planning**: Based on the knowledge provided by the previous modules, the *planning* module is responsible for decision-making regarding the vehicle's path and trajectory towards its destination, and this includes tasks such as: route (mission) planning, behaviour planning, and motion planning. The goal of this module is to eliminate human drivers' *decision-making errors*.

- **Control**: Takes the planned trajectory and starts executing it by triggering the appropriate vehicle controls (e.g. steering, brake, signals and lighting, etc.) An automated and accurate control unit will significantly reduce human drivers' *reflexive errors*.

1.1.2 Autonomous Vehicle's Sensors

Sensing is a critical module in a fully automated driving system, it must be highly reliable, as the vehicle's perception system heavily relies on its inputs. Sensing in an autonomous vehicle can be categorized into:

- **Self-sensing**: Constant computation of the vehicle's state (e.g. velocity, acceleration, steering angle, etc.). Uses sensors such as: odometers, gyroscopes, inertial measurement unit (IMU), and the control area network (CAN) bus.

- **Localization**: Determining the vehicle's local and global positions. Uses sensors such as the global positioning system (GPS).

- **Surrounding sensing**: Capturing information about the surrounding driving environment in order to give the vehicle the ability to "perceive" the road, the lanes, the surrounding objects, and the traffic signs.

In order to build a robust and reliable understanding of the surrounding environment, a fully automated driving system incorporates various sensor modalities to deliver a rich and complete representation of the environment. We discuss here

the three mostly used sensor modalities in the task of perception for autonomous vehicles.

- **Cameras**
  - **Monocular Camera** provides rich and detailed information about the textures and the two-dimensional shapes of the objects lying in the camera's limited *field of view* (FOV). However, monocular cameras lack the presence of depth information, which is critical for accurate position estimation in three-dimensional world coordinates. In addition, monocular cameras are subject to issues in adverse light and weather conditions.
  - **Stereo Camera** is a setup where two or more monocular lenses are used together, then their outputs are fed into a correspondence matching algorithm to recover the depth information. However, depth estimation algorithms are computationally expensive and have poor performance in some conditions (e.g. night-time).
  - **RGB-D Camera** provides synchronized depth and colour within the camera's *field of view* (FOV).
- **LiDARs**
  - LiDAR stands for *Light Detection and Ranging*, and it is categorized as an *active sensor*, as it operates by emitting and receiving laser beams, then uses the timing information between the emission and the reception of the beams to determine the distance from the sensor to the obstacles.
  - LiDARs usually use an array of laser beams that rotate in 360 degrees and covers a specific radius range. The LiDAR sensor outputs the following: (1) a set of points (coordinates) in the three-dimensional space, and these points form a representation that is called a *Point Cloud*, (2) a *reflectance* value for each three-dimensional point in the *point cloud* describing the strength of the received pulse.

  - LiDAR has many advantages: it has a large *field of view* (FOV), it provides accurate depth information, and it is not affected by light conditions. However, it does not provide any texture information, and the output *point cloud* is highly sparse and computationally expensive to process.

- **Radars**

  - Radar stands for *Radio Detection And Ranging*, and it is categorized as an *active sensor*. It transmits and receives radio waves to determine the angle, range, and velocity of objects, and is usually used to detect objects in mid to long-range distances. However, it is difficult to detect stationary objects using radar.

Figure 1.1 show a practicable setup of the previously mentioned set of sensors on an autonomous vehicle.

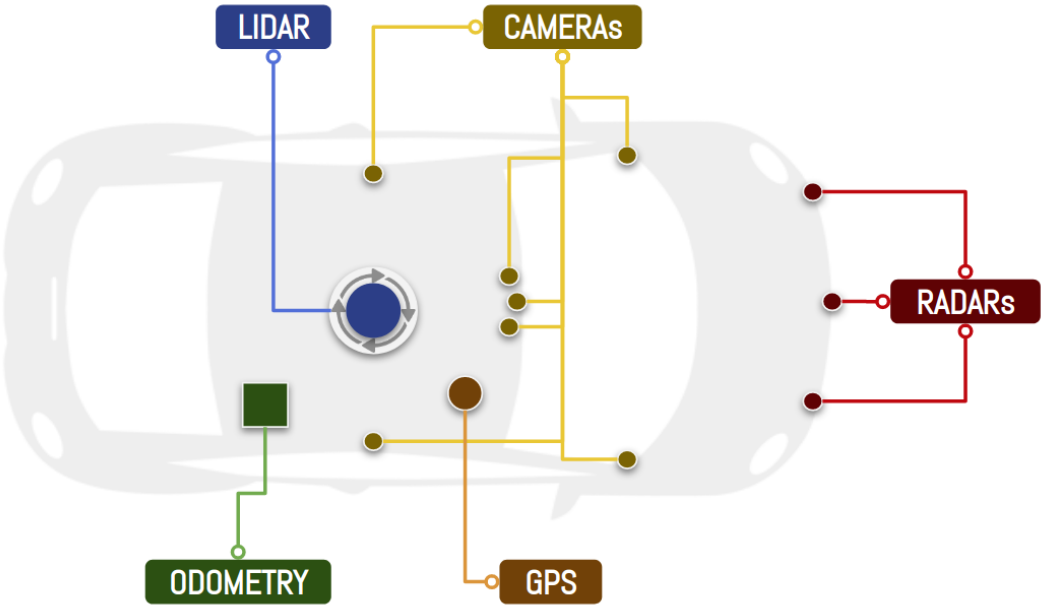


FIGURE 1.1: Sensors' setup on an autonomous vehicle.

1.2 Perception of Autonomous Vehicles

Perception is a critical module in a fully automated driving system; a vehicle must have perceptual capabilities to become aware of its surrounding environment to plan

and execute driving decisions that will result in a very high level of safety and efficiency on the road. A fully automated driving system leverages the various input modalities coming from different sensors mounted all around the vehicle, as they provide a representation of the surroundings, then acquires *computer vision* techniques to detect, localize, and classify the objects of interest around the vehicle. Perception could either be done by using an individual sensor modality, or by combining several sensor modalities to overcome the shortcomings of individual sensors. In the upcoming subsections, we will give an overview of *computer vision*, and the tasks of *object detection and recognition* in both two- and three-dimensional spaces.

1.2.1 Compute Vision

Computer Vision (CV) is the field of computer science that focuses on enabling computers to gain an understanding and extract useful information from the contents of visual digital data (e.g. images). As perception might seem a trivial and effortless task for us to do, however, perception in a dynamic environment is a highly complicated process to be implemented in a computer. Computer vision researchers aspire to enable computers to make sense of visual data represented in form of *bits*, as a human would do through his *physical sensation*.

Back in the sixties, leading experts in artificial intelligence thought that making computers "see" could be at the level of difficulty of a student's summer project. Sixty years later, *computer vision* is still a field full of challenges and open problems, here are some of the challenges facing a computer to be able to process visual data as humans do:

- *Limited understanding of how humans' biological vision works*. The human brain-eye coordination is still a mystery to be solved; it is fascinating how we can conceptualize what we see, and use our previous knowledge and experiences to help us process and understand visual data. Computer vision research will surely accelerate once we get to know precisely how *human perception* works.

- *Inherit complexity of the visual world.* A given object might be observed in a variety of sizes and aspect ratios, different orientations, under several light conditions, from multiple views, or it might even be occluded. However, a

computer must be able to extract meaningful information consistently in all of these situations.

- *Lack of context.* For us, humans, our brain helps us understand the context of what our eyes see in a magical way, however, a machine can only have access to the *bits* of a two-dimensional digital image. One of the most challenging problems to date in computer vision is *scene understanding*, where a computer's task is to understand a scene's context, as well as understanding different concepts in the scene along with their structure.

Computer vision is already transforming various aspects in our society and life, as it has become a critical component in many industries, such as: (1) *Security and safety*: which was enabled by the advancements in object detection and tracking algorithms, and was critical during the time of the COVID-19 pandemic. Advanced detection and tracking algorithms were used to ensure that people are following the precautionary measures (e.g. wearing face masks, applying social distancing, etc.) (2) *Healthcare*: during the pandemic, medical image analysis algorithms were used to detect the presence of COVID-19 virus given a chest x-ray scan. (3) *Automotive industry*: where *computer vision* plays a critical role in the advancement of fully autonomous vehicles' technology.

Traditionally, solving computer vision problems involved scientists and engineers who would design different filters to recognize various types of patterns found in images, this process is known as *feature engineering*. A field named *image processing* emerged to deal with various challenges found in visual digital information (e.g. light conditions, noise, distortions, etc.) Both, *image processing* and *feature engineering* were essential steps in the *computer vision* pipeline. Then traditional machine learning algorithms were used to process the extracted features to solve a specific problem (e.g. object recognition, face detection).

In 1989, the term *Convolutional Neural Networks* (CNN) was introduced by Yann LeCun et al. in [4]; it provided a new architecture and learning technique, that is inspired by the primary visual cortex, and that would advance the performance of the object recognition task. It was not until 2012 when Alex Krizhevsky et al. have used convolutional neural networks and deep learning [5] to win first place in the

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) which have inspired a series of advancements in the fields of computer vision and deep learning. A brief introduction about convolutional neural networks will be provided in the next subsection 1.2.2.

1.2.2 Convolutional Neural Networks

A *convolutional neural network* is a class of deep neural networks, that aims to automate the process of *feature engineering* by acquiring a deep neural network architecture that learns to automatically extract the traditionally *engineered features* from a given input image by leveraging a gradient-based learning algorithm. Figure 1.2 shows the difference between traditional machine learning and deep learning paradigms. In the *computer vision* field, convolutional neural networks are the architectures that are used within the deep learning paradigm.

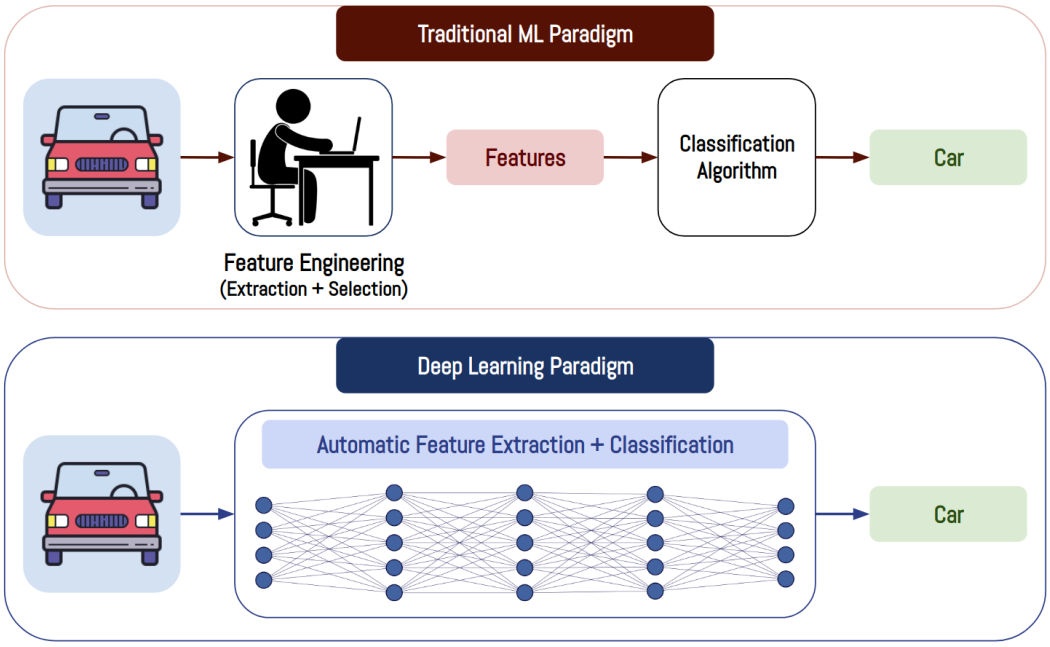


FIGURE 1.2: Difference between traditional machine learning (top) and deep learning paradigms (bottom).

Another family of neural networks is *multi-layer perceptrons* (MLPs), also known as *fully connected artificial neural networks*. MLPs simulate the structure of several layers of neurons' connections. The MLP's task is to learn the appropriate weight of each neuron using a gradient-based learning algorithm. One issue arises when

using MLPs to process or learn from visual data: an MLP would react differently to an input and its shifted version. MLPs are not shift-invariant architectures, which means that they are not good candidates when it comes to dealing with visual data.

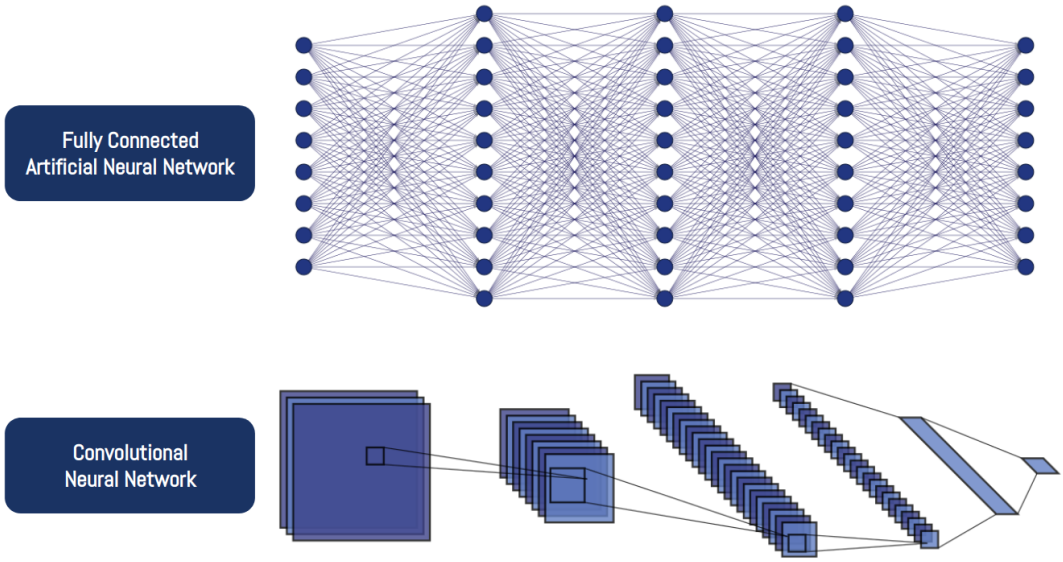


FIGURE 1.3: Architectural difference between a fully connected artificial neural network (top) and a convolutional neural network (bottom).

On the other side, a *convolutional neural network* has a set of characteristics that enables it to achieve the *feature extraction* task in a very efficient, performant, and automated way, compared to either traditional methods, or *multi-layer perceptrons*. CNNs have a shared-weights architecture, which means that for each layer, there consist a number of weight matrices named *filters*, and these filters learn their weights by being moved across the whole input of that layer, and by not just targeting a specific region in the input. This shared-weights architecture provides CNNs with a *shift-invariant* characteristic, which will enable the CNN to adapt to displacements found in an input image and will react indifferently from the original image. Figure 1.3 shows the architectural difference between an MLP and a convolutional neural network.

Another issue with MLPs is that when either depth or width of the network increase, the number of learnable parameters in the network increases significantly, as each layer is fully connected to the next one, and each connection represents a learnable weight. In contrast, since CNN is a shared-weights architecture, we can

increase either the width or depth of the network while still having a reasonable number of parameters and being able to process the input efficiently.

CNNs have different types, where each type is used to analyze and extract useful information from various forms of input data: (1) *1D convolutions* deal with signals to be processed and analyzed, (2) *2D convolutions* deal with image representation of visual digital data, (3) *3D convolutions* deal with volumes or three-dimensional representation of data, such as three-dimensional medical imaging. Computer vision researchers have used CNNs to achieve huge advancements in many computer vision challenges, including two critical tasks to the problem of perception in autonomous vehicles: *object detection* and *recognition*. Both will be briefly introduced in the next two subsections.

1.2.3 Two-Dimensional Object Detection and Recognition

Both object detection and recognition are two important sub-problems in computer vision, but also critical components when it comes to building robust and reliable perception systems. *Object detection* is the task of identifying the presence of certain objects in a given scene along with their corresponding positions. *Object recognition* is the ability to "label" the detected objects, and map them to their corresponding classes or categories.

When acquired in the two-dimensional object detection and recognition tasks, CNNs achieve state-of-the-art (SOTA) performance both in terms of detection accuracy and speed. Such network architectures are trained to take an RGB image as an input, and produce a *bounding box* around each detected object. This *bounding box* provides information such as: width, length, and center of the object as observed from the input image. Additional information is provided for each box such as: (1) how confident the network is about that detection (e.g. 80% confidence score), (2) the class label of that detection (e.g. pedestrian). Figure 1.4 shows a visualization of *bounding boxes* around vehicles in a scene from KITTI dataset [6]. Detections obtained from such architectures lack information about the distance of the detected objects from the sensor, which is referred to as the *depth information*.

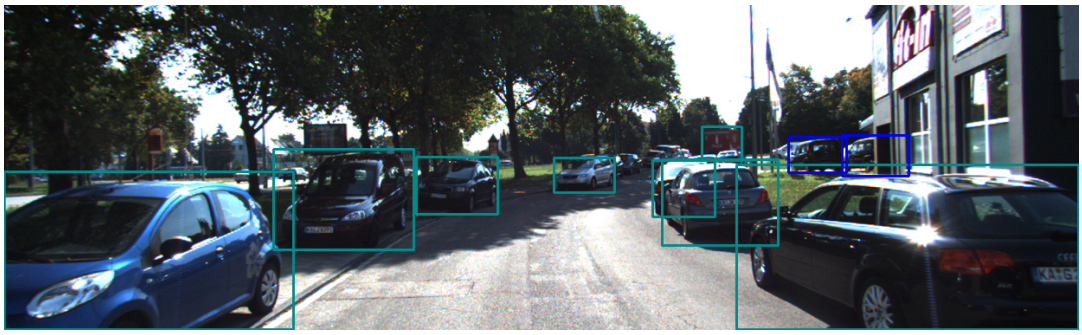


FIGURE 1.4: Example of two-dimensional vehicle detection. Image from KITTI dataset [6].

1.2.4 Three-Dimensional Object Detection

In the space of autonomous driving, the self-driving vehicle must construct an understanding of all the obstacles lying within its surrounding three-dimensional environment (e.g. vehicles, pedestrians, cyclists, etc.). Performing that task reliably will result in a much safer decision-making process. *Three-dimensional object detection algorithms* benefit from the robustness of CNNs and deep learning to detect and classify objects in a given scene, along with predicting their corresponding distance from the vehicle, their three-dimensional shape and size, as well as their orientation. For such algorithms to achieve high detection accuracy, sensor modalities that offer depth information are preferably used as input(s) for these architectures (e.g. stereo camera, RGB-D camera, LiDAR). In Figure 1.5, three-dimensional *bounding boxes* around vehicles are visualized within a scene from KITTI dataset [6].

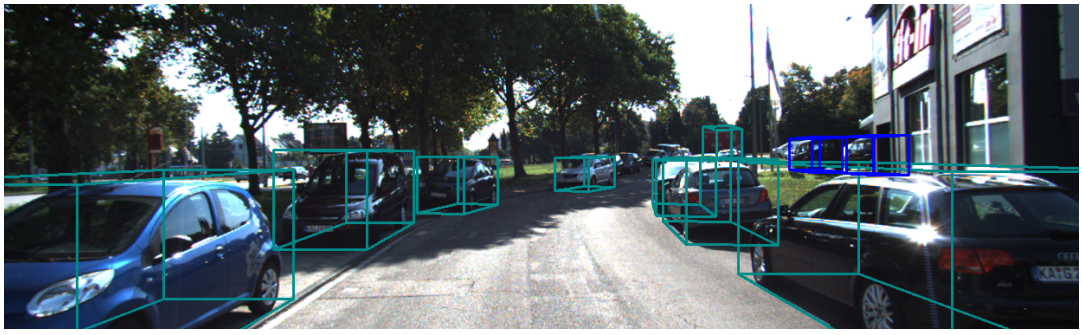


FIGURE 1.5: Example of three-dimensional vehicle detection. Image from KITTI dataset [6].

1.3 Challenges

Amounts of data. Collecting and annotating multi-modal data for the task of 3D object detection is a non-trivial and time-consuming task, as it requires the following: (1) mounting the self-sensing, localization, and surrounding sensors on top of a vehicle to start collecting the data, (2) determining intrinsic and extrinsic calibration parameters for the mounted sensors, (3) synchronizing data frames collected from different sensor modalities, (4) using either proprietary or open-source software for multi-modal data annotation, (5) validating the annotated data and (6) potentially providing evaluation code and metrics for benchmarking purposes. As with any other deep learning architecture, 3D object detection algorithms require large amounts of data to train in order to achieve high detection accuracy. Recently, the number of technology giants and startups who are working hard to provide the community with driving data with a variety of conditions and driving scenarios are increasing.

Reliability of the system. Some sensor modalities might have failure cases in certain conditions, such as cameras in adverse light conditions, radars in dealing with stationary objects, and LiDARs and cameras in adverse weather conditions. Having a fully automated driving system that is highly reliable and aware of possible failure cases is crucial for driving safely on the roads. Additionally, and most importantly, the perception system of the autonomous vehicle must be highly accurate in all the perception tasks including detection, recognition, and segmentation. Perception algorithms must be trained to perform accurately in various driving scenarios, different types of roads, and surrounded by a variety of obstacles' types.

Processing speed and computational cost. Driving decisions must be taken in real-time, with no-delay of any kind, as the driving environment is highly dynamic. When designing the perception system for an autonomous vehicle, the *computational cost* must be one of the evaluation metrics of such a system, as it should be as low as possible, so that the perception could be done in real-time without any considerable latency. This might affect the choice of sensor modalities to be used for perception. For example, although using LiDAR yields high localization accuracy, the processing of 3D point clouds is a challenging and computationally expensive task, while

camera images could be processed with a relatively low computational cost, however, they lack the presence of depth information. An autonomous vehicle usually has its own dedicated specialized hardware unit mounted within the car to perform all the required tasks, which is called *edge computing*.

1.4 Problem Definition

In this thesis, we address a critical component in fully automated driving systems' perception module, more specifically, the task of 3D detection and localization of objects in the surrounding environment of the autonomous vehicle. Most of the current detection techniques rely only on one sensor modality for this task, usually, LiDAR is used for its high localization accuracy. While using LiDAR is very advantageous for this task, we believe that fusing different sensor modalities would help overcome the shortcomings of individual ones, specifically, leveraging LiDAR's high localization accuracy with the richness of camera images to build robust deep learning-based 3D object detection algorithms.

In the presence of a similar sensors' setup as in 1.1, we leverage the multi-modal data inputs to be able to accurately detect and localize objects in the surroundings of the autonomous vehicle. We use both LiDAR and RGB camera sensors to accurately detect oriented bounding boxes in three different views: 2D, 3D, and top-view (Bird's Eye View). We are able to produce such detections simultaneously by employing a multi-modal frame-based robust deep learning architecture that is responsible for generating probabilities for potential surrounding objects, as well as identifying accurate bounding box information for every candidate detection.

1.5 Contributions

In this thesis, we addressed a critical component in autonomous vehicles' perception systems, which is the object detection and localization task. We develop a robust Camera-LiDAR deep-learning based sensor fusion framework that simultaneously generates BEV, 3D, and 2D oriented bounding boxes. Also, it is worth noting that while the proposed techniques are designed to tackle the perception of autonomous

vehicles, they could also be extended to other fusion-based object detection architectures.

We provide an in-depth analysis of the most important factors that contribute in building robust multi-modal object detectors. For this matter, we propose, develop, and analyze the following:

- We propose a sensor fusion framework that leverages feature maps' level fusion – i.e. *intermediate fusion*, along with *early fusion* of RGB images and important frontal view LiDAR features, such as: depth, intensity, and height maps.
- We show that integrating multi-task training within multi-view sensor fusion architectures significantly boosts detection accuracies, by training our models on multi-view supervised learning tasks aiming to produce high quality feature maps that would then be used for intermediate fusion.
- We provide a detailed analysis of the impact of using robust convolutional blocks within feature extractors on both performance and inference speed. We also study the impact of applying heavy data augmentations during the training process.
- We propose exchanging the Element-wise Addition fusion technique that has been used in the literature by Element-wise Multiplication, while showing the impact of such modification in a detailed quantitative analysis.
- We adapt a learnable fusion module from the field of temporal activity detection to the object detection field. We show how learnable fusion techniques can significantly boost the detection accuracy, and outperform both Element-wise Addition and Multiplication. Using such robust fusion mechanisms might give a new hope for multi-modal based deep learning architectures.

1.6 Thesis structure

Chapter 2 provides a comprehensive background on convolutional neural networks and the usage of deep learning to solve computer vision problems and presents the building blocks of the methods used or discussed in later chapters. **Chapter 3** is

dedicated to reviewing the literature on two- and three-dimensional object detection methods, including both uni- and multi-modal architectures. For each method, a discussion of the recently-proposed techniques will be provided. **Chapter 4** will provide an in-depth explanation of the implemented methods and techniques to address the problem of 3D object detection. **Chapter 5** is dedicated to explaining the details of our proposed deep learning network architecture used in the sensor fusion framework. **Chapter 6** will show the extensive experiments we have designed, along with the obtained results and detailed analysis and discussion around these results. The **last chapter** will provide a conclusion for this thesis, along with our suggestions for potential future research directions.

Chapter 2

Convolutional Neural Networks

The perception ability of an autonomous vehicle is critical to its reliability and safety of the roads. That's why researchers in the field of *computer vision* have spent years of research trying to come up with new techniques and algorithms to increase the performance the *perception systems*. Traditionally, *computer vision* challenges were solved by two sequential steps: (1) *feature engineering*, where scientist and engineers would extensively study the different expected scenarios of the given problem, and based on their study, they design filters that would help recognize patterns and extract useful features from inputs images, (2) then traditional machine learning algorithms are used to process the extracted features, and these algorithms are required to solve the problem in-hand. An illustration of the traditional computer vision process is shown in the top part of Figure 1.2.

As the complexity of the domain increases, it gets harder to design filters that can generalize and perform well in all different cases of a given computer vision problem, and this is mainly because of the inherit complexity of the visual world. In the case of autonomous vehicles, a perception system should be able to detect multiple objects (e.g. pedestrians, cyclists, vehicles) with various sizes, orientations, and distances from the autonomous vehicle. The *deep learning paradigm* (Figure 1.2 (bottom part)) has proved to be much more effective in such domains, and has pushed limits of what was possible or achievable in the field of computer vision.

Convolutional neural networks are the heart of modern perception systems. The main idea behind the deep learning paradigm is to give a network the ability to learn the previously "*engineered features*" from the data related to the problem in-hand. The more the data provided, the better the system at the task of learning

various patterns and filters. In 2012, Alex Krizhevsky et al. have won the first place in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [7] by using convolutional neural networks and deep learning. Their priceless contribution have inspired a series of huge advancements in the fields of computer vision and deep learning.

The utilization of convolutional neural networks represents a large part of our work in advancing the perception system of self-driving vehicles through sensor fusion, so were are dedicating this chapter to present a detailed overview of these architectures. **Section 2.1** provides an overview of the basic building blocks of a CNN. **Section 2.2** explains the different design decisions in such architectures. **Section 2.3** presents classical CNN architectures. **Section 2.4** presents a number of modern CNN architectures.

2.1 Basic Building Blocks

2.1.1 Convolutional Layer

The convolutional (layer) is the core building block of a convolutional neural network. This layer is used for the *feature extraction* step, and it comprises of two important concepts: (1) learnable filters, and (2) convolution operation. A **filter**, or also known as a *kernel*, represents a spatially small image that depicts a particular feature. For example, a specific filter was designed to detect horizontal and vertical edges within an image, namely Sobel filter. Both Sobel filter's components are defined in Equation 2.1, where G_x is the vertical edge detector, and G_y is the horizontal edge detector. These filters are also called *gradient matrices*, as they are used to approximate the gradient of the image.

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}, G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{2.1}$$

A **convolution operation** is a simple linear operation, which takes a spatially small filter G, and operates on an input image I (assume I is a grayscale image for simplicity). The output of this operation is computed by sliding G horizontally and

vertically over each block of the input I. At each block, the element-wise product of both G and the block at position (i, j) in I is being computed and stored in the corresponding position in an output matrix O. The resulting output of this operation is called a *feature map*. The formal definition of the convolution operation is defined in Equation 2.3.

$$O[m, n] = I * G[m, n] = \sum_j \sum_k G[j, k] * I[m - j, n - k] \tag{2.2}$$

To be able to detect horizontal and vertical edges within a grayscale image, both Sobel filters (G_x, G_y) are convolved over the whole inputs image, creating two new output images, one for each filter. In each image, the pixels where an edge was found, its color would be brighter (higher pixel value) than other places where no edges were found. In other words, these filters detect features by matching themselves with every local structure within the input image. The presence of a match results in high-value in the output feature map, and vice versa. Figure 2.1 illustrated the effect of applying Sobel filter to a simple grayscale image that contains a cube. A point that is both vertical and horizontal is by definition a diagonal, and that is the reason why we notice the diagonal lines between the vertices are detected by both filters.

A convolutional layer relies on both concepts, filters and convolution operation, but instead of using predefined filter values to extract and detect features, a convolutional layer has a number learnable filters, and the objective is the learn the values for these filters by the help of an optimization algorithm. Figure 2.2 illustrates the difference between the traditional process and the mode of operation of a convolutional layer.

Convolutional neural networks are usually used to solve complex problems that involve processing and learning from colored images. Colored images consist of three channels: red, green, and blue, and such images are known as RGB images. An RGB image has spatial dimensions (width and height), but also has a third dimensions: the depth, which has a value of three, one for each color channel. The dimensions of an RGB could be represented as ($W \times H \times C$), where W represents the width, H represents the height, and C represents the number of channels. When

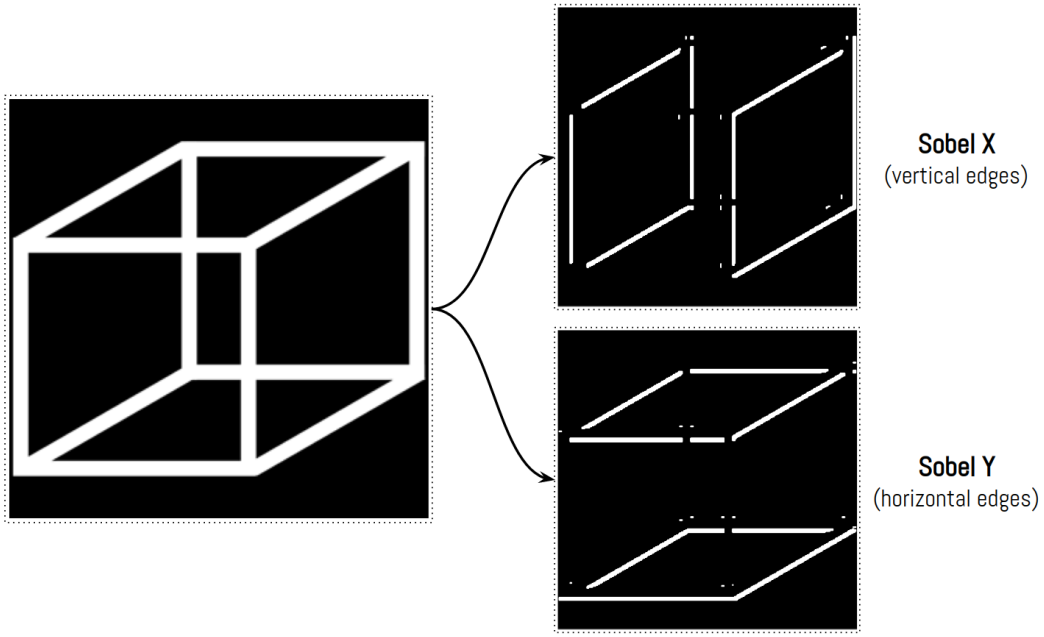


FIGURE 2.1: Illustration of applying Sobel filter to a simple grascale image. "Sobel X" represents the output when the filter G_x is applied, and the detection of vertical edges. "Sobel Y" represents the output when the filter G_y is applied, and the detection of horizontal edges. The outputs generated by using OpenCV library. [8]

the input to a convolutional layer is volume-shaped, as the case of an RGB image, the learnable filters within that layer will still have small spatial dimensions, however, each filter will have the same number of depth channels as the layer's input. Assuming an input volume of shape $W \times H \times C$, and squared-shaped filters are used, then each of the learnable filters within that layer will have the shape $K \times K \times C$, where $K \times K$ represents a filter's spatial dimensions.

$$includeniceequationhereO[m,n] = I * G[m,n] = \sum_j \sum_k G[j,k] * I[m-j, n-k] \quad (2.3)$$

The output of a convolutional layer is controlled by several hyperparameters:

- **Filter size (K).** Controls the size of the filters within the layer, and its value is usually preferred to be an odd number. The reason behind that odd-sized filters can be anchored on a center point, and that center have equal number of neighbour pixels in all directions, which is not the case for an even-sized filter. Using even-sized filter would result distortion and ailiasing errors in the

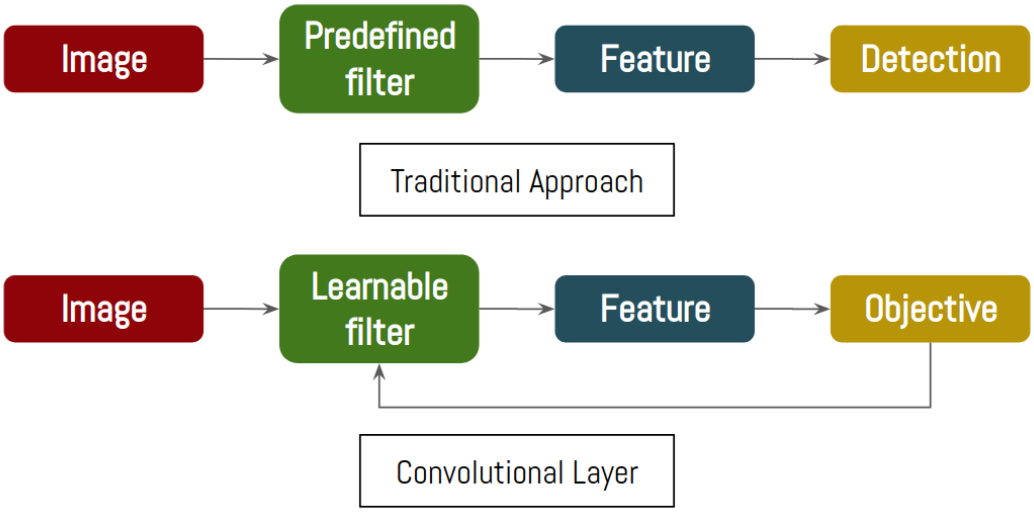


FIGURE 2.2: Illustration of the difference between the tradition approach for feature extraction compared to the mode of operation of a convolutional layer within a deep neural network.

output feature map. Having an odd-sized, sqaure-shaped filters works best for the task of feature extraction in a convolutional layer.

- **Number of filters (F).** Controls the total number of patterns that could be extracted by a specific layer. All filters within a single layer should have the same spatial dimensions. The input volume is convolved with each filter seperately to produce an output feature map. Then all output feature maps are concatenated to form the final output volume of the layer.

- **Stride (S).** Controls the step size we use while we are sliding the filters over the input volume. For example, when $S = 1$ the filters move one step at a time, when $S = 2$ the filters move two steps at a time, etc. The affects the spatial dimensions of the output feature map, larger stride values will produce smaller feature maps.

- **Padding (P).** When consecutive convolutions are applied sequentially, the spatial dimensions of the output feature maps will reduce at each convolution operation, which will prevent us from designing deep convolutional networks (with many consecutive layers). The padding will allow us to control the spatial dimensions of the output feature map. Usually *zero-padding* is used to preserve the spatial dimensions of the input volume, so that both the input and

output will have the same width and height. Also, the the convolution operation does not give an equal weight to the corner pixels compared to pixels in the center of an input volume. When zero-padding is applied, the corner pixels are being accounted for equally as other pixels.

Assuming an input volume of $W \times H \times C_{in}$ fed to a convolutional layer having F filters, each filter has the size K, the filters are being moved with a stride of S, and padding P is applied to the input volume. The output dimensions can be calculated using the formulas in Equation 2.4, where $W' \times H' \times C'$ represents the dimensions of the output feature map after applying the convolution operation. A visual illustration of the convolution operation being applied on an RGB image can be found in Figure 2.3.

$$\begin{aligned} W' &= \frac{(W-K+2 \cdot P)}{S} + 1 \\ H' &= \frac{(H-K+2 \cdot P)}{S} + 1 \\ C' &= F \end{aligned} \tag{2.4}$$

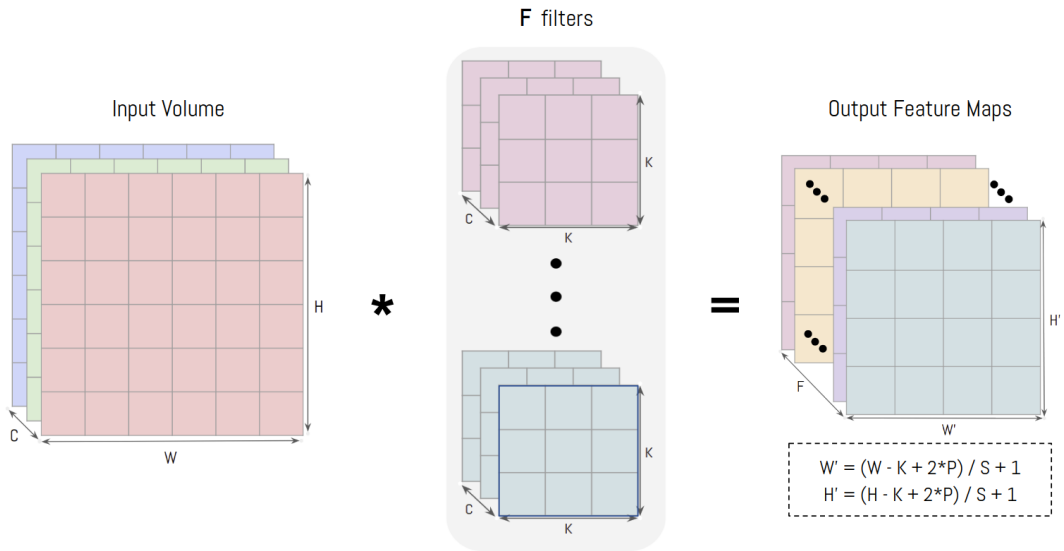


FIGURE 2.3: Demonstration of a convolution operation with $\mathcal{F}$ filters applied to an input RGB image with 3 channels, outputting $\mathcal{F}$ feature maps.

Like other types of neural networks' layers, a convolutional layer contains a *bias* term, which is learnable as well, and is useful in the training process of such architecture. A single bias term is initialized for each filter within a convolutional layer.

The total number of **learnable parameters** in a standard convolutional layer could be computed as in 2.5, the first term in the addition is denoting the weights, while the second is denoting the biases.

$$\begin{aligned} &F \times K \times K \times C + F \\ &\text{where } F \text{ is the number of filters,} \\ &K \text{ is the kernel size,} \\ &C \text{ is the number of input channels} \end{aligned} \tag{2.5}$$

2.1.2 Non-linearity Operation

Universal Approximation Theorem [9] (UAT) is the theorem behind the huge success of deep neural networks in many complex domains. UAT proves the potential of deep neural network architectures to be *universal learning machines*, which means they are capable of approximating any function. More specifically, UAT suggests that an artificial neural network with a single hidden layer that consists of sufficiently many hidden neurons, where each hidden neuron is incorporated with a *continuous*, *bounded*, and *nonconstant* activation function, can be a universal approximator. UAT also suggests that the smoother the activation function, the more accurate it can approximate a function along with its derivatives. Since a *linear activation function* (Equation 2.6) is considered to be an *unbounded function*, it will not be suitable for designing universal learning machines. Instead, A family of non-linear activation functions were designed to address the constraints stated by UAT, and aiming to achieve universality for deep neural networks.

The most widely adopted non-linear activation functions in the early stages of deep neural network research were: **Sigmoid** (Equation 2.7) and **Hyperbolic Tangent** (Tanh) (Equation 2.8). Both activation functions are *non-linear*, and satisfy the constraints of being *continuous*, *bounded*, and *nonconstant*. However, there are a couple of challenges when these two non-linear activations are used in neural networks. (1) It is computationally expensive to computing the derivative for these activation functions, as seen in both equations 2.7 and 2.8. (2) Slow convergence when training deep neural networks due to the problem of *vanishing gradients*. As more layers

are stacked, the gradients with respect to the inputs gets smaller and smaller, to the point it will vanish, and the network will be no longer be able to learn.

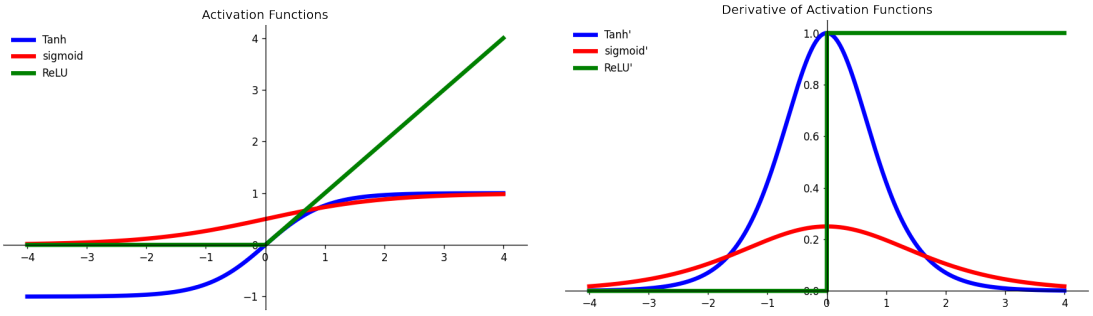
$$\begin{aligned} \textbf{identity}(x) &= x \\ \textbf{identity}^{'}(x) &= 1 \end{aligned} \tag{2.6}$$

$$\begin{aligned} \textbf{sigmoid}(x) &= \frac{1}{1+e^{-x}} \\ \textbf{sigmoid}^{'}(x) &= \textbf{sigmoid}(x)(1-\textbf{sigmoid}(x)) \end{aligned} \tag{2.7}$$

$$\begin{aligned} \textbf{tanh}(x) &= \frac{2}{1+e^{-2x}} - 1 \\ \textbf{tanh}^{'}(x) &= 1 - \textbf{tanh}(x)^2 \end{aligned} \tag{2.8}$$

In order to be able to design deep networks that are trained using gradient-based optimization algorithms, more robust non-linear activation functions were required. [10] have shown that artificial neural networks incorporating *unbounded* activation functions are still capable of being universal approximators. The idea if using *unbounded* activation functions within deep neural networks has led to the design of deeper and more complex neural network architectures, which resulted in huge advancements in the field of artificial intelligence. Even after validating the usage of *unbounded* activation functions in deep neural networks, the *linear activation function* is still not a suitable candidate. When stacking multiple linear layers to construct a deep neural network, the composition of each two consecutive linear layers is itself one linear layer, which means a deep neural network with N layers with linear activations corresponds to a network with a single linear layer. Also, the back-propagation algorithm cannot be applied as the gradient of a linear function is always 1 and is no longer dependant on the input, so we are not going to be able to change the network's weights to improve the model.

Rectified Linear Unity (ReLU) activation function was first used in the domain of neural networks in [11], where ReLU have been proved to be more efficient at learning patterns and features compared to either Sigmoid or Hyperbolic Tangent activations. ReLU (Equation 2.9) activation is very simple, yet very powerful. (1) It is computationally efficient to compute, no need to compute the exponential function of the input as in both Sigmoid or Tanh. (2) ReLU has a linear behaviour, which



(A) Non-linear activation functions.

(B) The corresponding derivatives for the non-linear activation functions.

FIGURE 2.4: Plot visualization of *Tanh*, *Sigmoid*, and *ReLU* activation functions in (A), along with their corresponding derivatives in (B).

means it does not saturate, the gradient is always equal to 1 when a neuron is activated, and 0 otherwise. That is why ReLU has a fast rate of convergence. Both Sigmoid and Tanh saturate when the inputs are large, the gradient values are small, which slows down the learning process. (3) ReLUs are great candidates when designing very deep neural network architectures, as they do not suffer from the *vanishing gradients* issue, due to their linear behaviour. Figure 2.4a shows a plot of the three different activation functions, and Figure 2.4b show the derivative of the same activation functions.

$$\begin{aligned} \textbf{ReLU}(x) &= \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{if } x \geq 0 \end{cases} \\ \textbf{ReLU}^{'}(x) &= \begin{cases} 0, & \text{if } x < 0 \\ 1, & \text{if } x \geq 0 \end{cases} \end{aligned} \tag{2.9}$$

Applying a non-linear activation function after the convolution (2.1.1) is also sometimes known as the *detector stage*. After getting the feature maps from the convolution operation, the activation function will decide for each pixel whether or not it is *activated*. In other words, if a pattern was matched with a local structure in the input image, the the convolution operation will produce a large value in the output feature map (> 0), and this large value will be preserved by applying the non-linear activation function (e.g. ReLU). Otherwise, if there were no matched patterns, the convolution operation will yield a small value (≤ 0), which means that this neuron will not be activated, and the corresponding value in the output feature map will be

set to 0. Figure 2.5 demonstrates both convolution and activation stages on a simple grayscale input image.

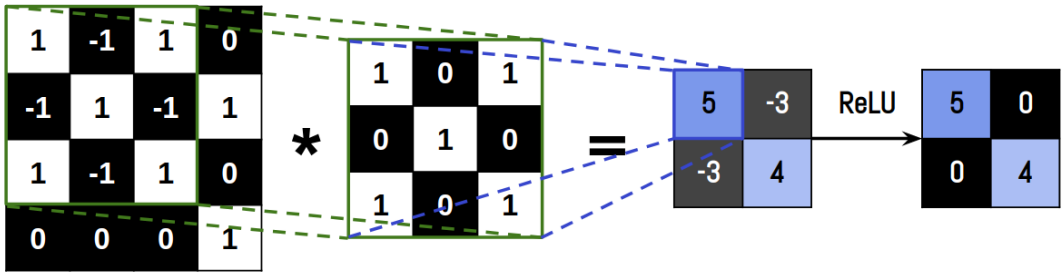


FIGURE 2.5: The convolution operation in action, followed by a ReLU activation function to preserve only important features within the feature map.

2.1.3 Downsampling Operation

Downsampling is the operation responsible for reducing the spatial dimensions of the input of a convolutional layer. Applying downsampling within deep convolutional neural networks is important mainly for two reasons: (1) preserve the relevant information detected by the the activation layer, and (2) reducing the computation complexity of the network, as it will be operating on smaller sized inputs, hence, reducing the overall number of parameters required to train the CNN model. Downsampling is performed using either **Pooling Layers** or **Strided Convolutions**.

Pooling layer. Pooling operation is a simple operation, that is performed by applying a sliding window on the input with a specific *step size* (S), and a *filter size* (K). While iterating over local structures in the input, a predefined operation is applied to each local structure, and produces a single-valued output that represents this specific local structure. The *predefined operation* is usually either *Average*, or *Maximum*. When comparing the performance of both *average* and *max* pooling operations, it turns out *max pooling* yeilds better results, and is usually used in modern CNN architectures for the prupose of downsampling. The advantages of a pooling layer that it does not have any learnable parameters, an it is a computationally inexpensive operation.

Assuming an input volume of $W \times H \times C_{in}$. A pooling layer is applied with step size of S and a filter size of K. The output of such a layer can be calculated using the formulas in Equation 2.10, which are the same for the spatial dimensions as the

output of a convolution operation, however, the depth dimension is preserved and not changed by a pooling layer. Figure 2.6 illustrates the process of applying max pooling operation to an input image.

$$\begin{aligned} W' &= \frac{(W-K+2\cdot P)}{S} + 1 \\ H' &= \frac{(H-K+2\cdot P)}{S} + 1 \\ C' &= C_{in} \end{aligned} \tag{2.10}$$

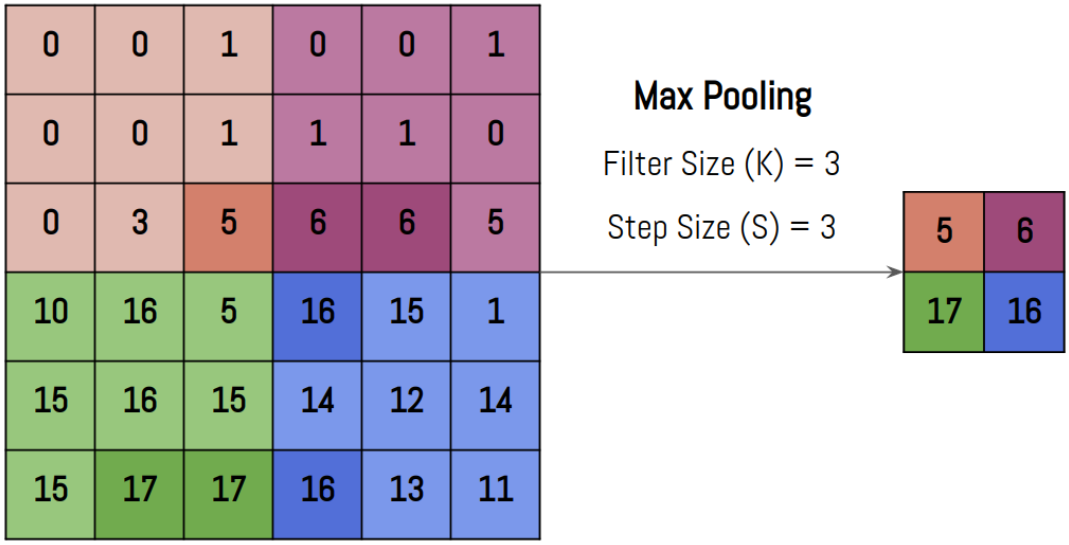


FIGURE 2.6: Demonstration of the max pooling operation applied on a grid of size 6×6, and with both filter size and stride equal to 3. Max pooling's usefulness comes from "extracting" important features with the largest values from the input image.

Strided Convolution. By increasing the stride hyperparameter in a convolutional layer, the spatial dimensions of the output feature maps could be downsampled (Equation 2.4). The drawback of this downsampling technique is that extra learnable paramterers are introduced within the CNN architecture. However, strided convolutions are usually used in the first layer of a CNN architecture [12]. The reason is to significantly reduce the amount of computation needed in the subsequent layers, especially if the size of the input image is large.

2.2 CNN Architecture Design

In this section we will cover the most critical design decisions one should make when designing convolutional neural network architecture.

2.2.1 Objective Function

The choice of the objective function of a neural network architecture depends mainly on two factors: (1) the type of data set available for training, and (2) the type of the task we are aiming to solve. If a data set D consists of observation pairs (X, y), where X is a sample input, and y represents the corresponding expected output, then this data set D could be useful for supervised learning algorithms, where a model is trained to learn a function representation that maps an input X to a corresponding output y - i.e. $f(y|X, \theta)$, by optimizing a set of learnable parameters θ based on the training data. There are other types of data sets where only a percentage of the training data is unlabelled. In this case, semi-supervised learning algorithms could be incorporated. Unsupervised learning algorithms are used when the data set contains no information about the expected labels for input data samples, and the goal is to learn more about the data by modelling its underlying structure or distribution. In our work, we focus on incorporating supervised learning algorithms to solve the three-dimensional object detection and localization task.

Objective functions, or also known as loss functions, are the main factor that derives the optimization of the learnable parameters θ towards learning a specific task T. In the scope of supervised learning algorithms, an objective function is as simple as computing the error value between the model's predicted output $\hat{y} = f(X, \theta)$, and the expected output denoted in the data set y - i.e. the groundtruth. Gradient-based optimization algorithms are then used to minimize the error of such objective functions by modifying the model's learnable parameters θ.

Supervised learning tasks can be categorized into *classification* and *regression* tasks. Classification is when a neural network aims to learn a mapping between an input X and a class label c, chosen out of a set of C predefined classes. More formally, the neural network predicts the probability of input X being assigned to a class label c, where $c \in \mathbb{Z} : c \in [1, C]$, and c could be either binary ($C = 2$) or categorical ($C > 2$).

On the other hand, regression involves in predicting a real-valued output quantity. For example, given an input image of a car: a binary classification task would be to predict whether or not a car exists in the image, a categorical classification task would be to predict the car's brand out of C different brands, while a regression task would be to estimate the car's price.

For each of these supervised learning tasks, a different activation function should be employed in the final layer of the model. *Sigmoid* activation is used for binary classification tasks, as it maps an input to a value lying in the range between $\{0, 1\}$, which will represent the probability of the positive class. *Softmax* activation is used for categorical classification tasks, as it produces a set of output class probabilities. For regression tasks, there is no need for applying a non-linear activation, as the goal is to map an input to a real-valued output.

Classification objective functions. There exist various objective functions used for classification tasks. The cross-entropy (CE) loss has proven its success as the core loss of classification models whose output is a probability value between 0 and 1. The general form of the CE loss is defined in Equation 2.11, where C is the number of output classes, y_i and $\hat{y}_i$ are the ground-truth value and the value predicted by the model for the i^{th} class, respectively. In a binary classification task where $C = 2$, the CE loss could also be defined as in Equation 2.12, and it could be summarized as in Equation 2.13. The CE loss value increases when the model's predicted probability $\hat{y}_i$ is different from the expected value y_i, and gets closer to 0 otherwise. Gradient-based algorithms optimize this objective function by minimizing the *negative log-likelihood* (Equation 2.11), and this optimization objective is equivalent to *maximizing the log-likelihood* of the model.

$$\mathbf{CE}(y, \hat{y}) = -\sum_{i=1}^{C} y_i \log(\hat{y}_i) \tag{2.11}$$

$$\begin{aligned} &\mathbf{CE}_{binary}(y, \hat{y}) = -\sum_{i=1}^{C=2} y_i \log(\hat{y}_i) \\ &= -y_1 \log(\hat{y}_1) - y_2 \log(\hat{y}_2) \\ &= -y_1 \log(\hat{y}_1) - (1 - y_1) \log(1 - \hat{y}_1) \end{aligned} \tag{2.12}$$

where y_1 denotes the positive class, y_2 denotes the background class

$$\mathbf{CE}_{binary}(y, \hat{y}) = \begin{cases} -\log(\hat{y}), & \text{if } y = 1 \\ -\log(1 - \hat{y}), & \text{otherwise} \end{cases} \tag{2.13}$$

Another widely used variant of the binary classification CE loss is called the *focal loss* (FL), and is defined in Equation 2.14. FL was designed to address scenarios where the training data suffers from extreme class imbalance between the foreground and background classes, and was initially used to train robust single-stage object detectors [13]. When performing object detection from image, the target pixels assigned to the background class are usually a lot more compared to the foreground (positive class), which affects the performance of such single-stage detectors negatively. FL solves this issue by introducing a modulating factor to the CE loss, with a goal to down-weight easy examples (background), and to drive the model to focus more on hard or misclassified examples (foreground).

$$\begin{aligned} p &= \begin{cases} \hat{y}, & \text{if } y = 1 \\ 1 - \hat{y}, & \text{otherwise} \end{cases} \\ \mathbf{FL}(p) &= -\alpha(1 - p)^{\gamma} \log(p) \\ &\text{where } \alpha \text{ is a weight factor, } \gamma \text{ is a modulating factor} \end{aligned} \tag{2.14}$$

Regression objective functions. A number of objective functions were designed especially to give machine learning models the ability to regress real-valued output. One basic objective function used in regression tasks is the *mean absolute error* (MAE), or also known as the L_1 *loss* (Equation 2.15), and it is computed by measuring the average of the absolute differences between the predicted and target values. MAE loss is robust to outliers, however, its derivatives is not continuous which makes it hard to optimize. Another objective functions is the *mean squared error* (MSE), or also known as the L_2 (Equation 2.16). The squared term in the L_2 loss gives more weight if the error is greater than 1, which makes it sensitive to outliers. On the other hand, it provides a more stable behaviour during optimization as it is differentiable everywhere.

$$\mathbf{L1} = \frac{1}{N} \sum_{i=1}^{N} |y_i - \hat{y}_i| \tag{2.15}$$

$$\mathbf{L2} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{2.16}$$

Huber loss, or also known as *smooth* L_1 *loss*, was designed to combine the properties of both L_1 and L_2 loss functions. The smooth L_1 loss is more robust to outliers than L_2 loss, at the same time, it is differentiable at 0. The behaviour of smooth L_1 loss is determined by a tunable hyperparameter β. In other words, it behaves linearly when the error is larger than β, and quadratically when the error is smaller than β. Figure 2.7 shows plots for the three discussed regression loss functions.

$$\mathbf{L1}_{\text{smooth}} = \begin{cases} 0.5(y_i - \hat{y}_1)^2/\beta, & \text{if } |y_i - \hat{y}_1| < \beta \\ |y_i - \hat{y}_1| - 0.5 * \beta, & \text{otherwise} \end{cases} \tag{2.17}$$

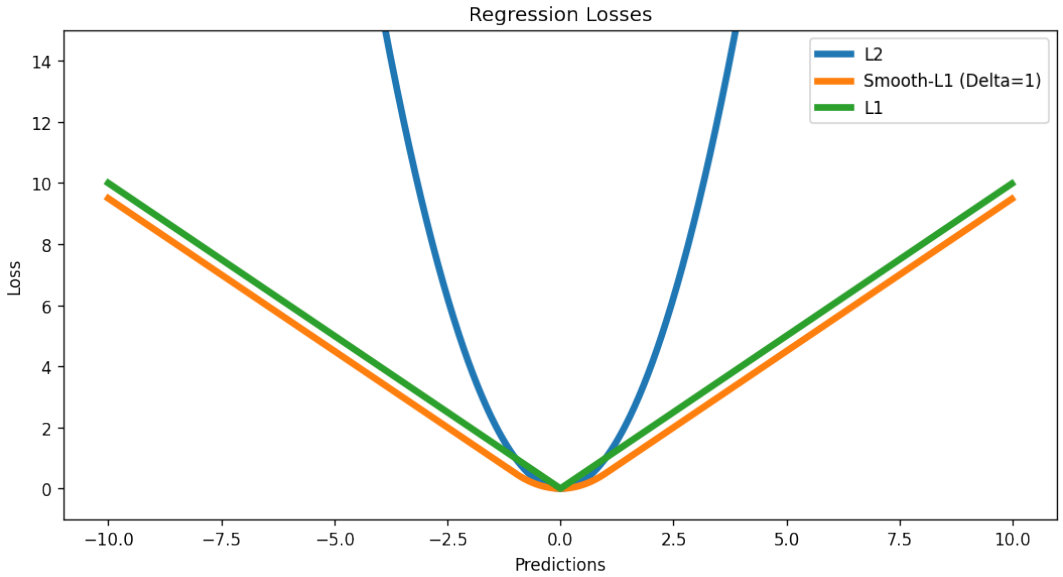


FIGURE 2.7: Visualizing three commonly used regression loss functions. Smooth l_1 loss is combining both properties of l_1 and l_2 losses: the robustness to outliers, and the smoothness at 0, which makes it differentiable everywhere.

2.2.2 Optimization

Training large neural networks is mainly about optimizing their learnable weights θ with the following objective in mind: finding the set of model weights θ that will result in the smallest value of the loss function - i.e. finding the global minima of the objective function. We can find this set of weights through an iterative process called

the *gradient descent algorithm*. Initially, a model starts with a randomly initialized set of weights. First, we compute the model's loss value $J(\theta)$. Then, we compute the gradient of the loss function, which basically measures how much the output of a function changes if the input is slightly changed. This is done by computing the partial derivatives of the loss function with respect to the model's weights. We then use these partial derivatives to update the model's weights in the direction that minimizes the loss value - i.e. the direction of the steepest descent. Figure 2.8 visualizes a simplified version of the gradient descent algorithm.

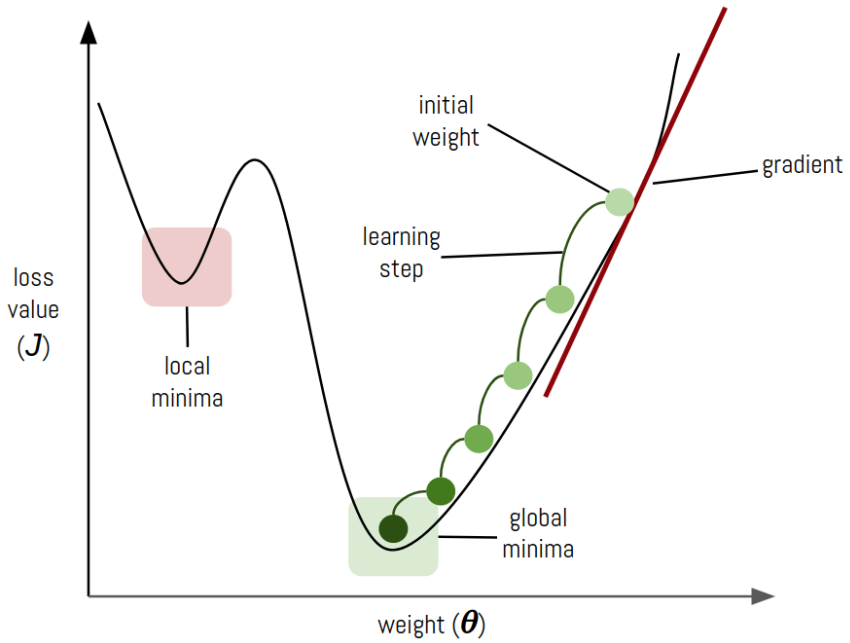


FIGURE 2.8: Simple visualization for consecutive steps taken by a gradient-based optimization algorithm to reach the global minima, while highlighting the learning step (learning rate), and the local minima which should be avoided.

The weight update rule is denoted in Equation 2.18, where J is the loss function, θ represents the model's weights, ∇ represents the gradient of the loss function with respect to θ, and γ represents the step size, or also known as the *learning rate*. The learning rate is a hyperparamter which controls the size of the step the algorithms is going to take towards the steepest descent. Tuning this hyperparamter is critical to a successful convergence of the optimization algorithm; if it is set to a high value, it

will be likely to suffer from divergence, where it never reaches the global minima.

$$\theta_{\text{new}} = \theta_{\text{old}} - \gamma \nabla_{\theta_{old}} J(\theta_{\text{old}}) \tag{2.18}$$

The gradient descent (GD) algorithm can perform weight updates in three different ways: batch updates, stochastic updates, or mini-batch updates. **Batch GD** updates the model's weights after calculating the error and the gradient for every example in the training set. Batch GD provides a stable convergence. However, its drawback is that it requires the whole training set to be in-memory. On the other hand, **stochastic GD** (SGD) performs an update for each training sample after evaluating its gradient. Although SGD is less likely to being stuck in a local-minima due to its stochastic behaviour, however, it might result in noisy gradients and an unstable convergence, which will take more training time compared to batch GB. **Mini-batch GD**, combines the stability and efficiency of batch GD with the robustness of SGD. It splits the training data into small batches, and performs an update after evaluating the gradients of each of these batches.

Several versions of the GD update rule were designed with the goal to (1) stabilize and speed-up the convergence, (2) choose an ideal step size (learning rate) in the direction of the steepest descent, and (3) avoid being stuck in a local minima. **Momentum** is a method where past steps are taken into consideration when determining the direction of the next update step - i.e. momentum includes an exponentially weighted average term of the previous gradients, which will quicken the convergence towards to global minima. **RMSProp** uses an adaptive learning rate scheme to adjust the learning rate for each weight based on the decaying average of past squared gradients. **Adam** combines the advantages of both Momentum and RMSProp, it accelerates and decelerates according to the momentum value, while keeping an adaptive learning for the weights based on the exponential weighted average of the past squared gradients.

2.2.3 Regularization

A common goal when training large neural networks is to have not just high performance on training data, but also on new unseen test data - i.e. the ability to generalize. However, a neural network with large capacity often suffers from overfitting, which leads to a having a low training error rate and a high test error rate. Regularization techniques were designed to avoid large neural networks from overfitting [14]. One type of regularization is the **parameter norm penalties**, where a penalty term is added to the objective function. L^1 and L^2 Regularization are applied by adding the terms $\Omega(\theta) = \lambda \sum_i |w_i|$, and $\Omega(\theta) = \lambda \sum_i w_i^2$ to the objective function, respectively. λ is a hyperparamter called the *regularization coefficient*. The higher the value of λ, the more the model's weights will be penalized towards obtaining less complex solutions, which will help decrease the generalization (test) error rate.

Data augmentation is another technique used to regularize neural networks with large capacities. Neural networks could have better generalization if provided with sufficiently large number of diverse training examples. Data augmentation aims to increase the number of training examples by generating more synthetic or "fake" data. When dealing with imagery data, data augmentation can be done by transforming original data samples to create new ones. Examples of transformations used are: rotation, translation, scaling, flipping, cropping, adding noise, etc. By doing that, we make sure that the neural network always sees new data examples during the training process, which will reduce overfitting.

Dropout [15] is another powerful technique that addresses the issue overfitting. It works by randomly dropping out (deleting) neurons during training, while each neuron has a probability of being dropped (e.g. 50%). Dropout is applied during training. In each training step, the networks is forced to achieve low error rate with only a subset of the original weights of the network. For a neural network with u learnable units, training with dropout is equivalent to simultaneously training a 2^u smaller networks on the same task, while all 2^u networks share the same set of parameters.

2.2.4 Normalization

Deep neural networks usually suffer from a phenomenon called *internal covariate shift* (ICS). ICS happens as an effect of the distributional change of a layer's inputs caused by weight updates in earlier layers. In the case of deep neural networks, the occurrence of this phenomenon impacts both training performance and convergence speed negatively, regardless of the optimization algorithm. Gradient-optimization algorithms are highly sensitive to the choice of both learning rate and the weight initialization method, which might leads to one of two undesirable phenomenons: vanishing gradients (being on a flat region in the loss function landscape) or exploding gradients (stuck in local minima).

Batch Normalization [16] (BN) layer was designed to overcome the challenges of training deep neural networks. BN layer applies a normalization step before the non-linearity operation. The normalization step uses per mini-batch statistics (mean and standard deviation) to fix the layer inputs to having a zero-mean and a unit-variance. To maintain the network's flexibility of choosing the appropriate weights to learn complex representations, γ and β are introduced as learnable scale and shift parameters, respectively, and are applied after the initial normalization step. BN is robust to initial weight initialization, prevents vanishing/exploding gradients, and results in much faster convergence rates. It is worth noting that BN is used only during the training process, and the normalization is not required during the inference.

Although the core concept behind the design of BN was to reduce ICS, [17] suggest that the reason behind the practical success of BN is not the reduction of a network's ICS, instead, the usage of normalization results in a significantly smoother optimization landscape, which improves the convergence speed. [17] also suggest that this smoothness effect is not tightly coupled with BN, and it could be achieved by applying other normalization techniques as well. BN yields poor performance when small batch sizes are used, due to the inaccurate estimation of per mini-batch statistics. Group Norm [18] (GN) was developed to address this issue. GN is independent of batch size, as the normalization operation is applied on grouped channels for each training example. GN offers consistent and stable performance over various batch size settings.

2.2.5 Width and Depth

Usually, deep CNNs are designed by stacking learnable blocks, where a block represents a set of similar and repeated operations. A basic CNN block consists of (1) convolution operation, followed by (2) a normalization step, then (3) a non-linear activation, and (4) an optional pooling layer. One more thing to consider when designing a CNN is the choice of two hyperparameters, namely the *depth* and *width* of the network. The network's depth corresponds to the number of CNN blocks, while the width corresponds to the number of learnable filters used in each block.

Deeper networks are able usually to learn more complex functions compared to shallow networks. However, training deep networks might be difficult due to the problem of vanishing gradients. The width of a each CNN block is chosen depending on the number of features we are aiming to extract at a specific layer. Earlier layers a CNN architecture learn low-level features (e.g. edges, diagonal), hence, small number of learnable filters are used. Deeper layers are usually initialized with larger number of filters, as they tend to learn more complex representations that are more related to the learning task.

2.3 Classical Architectures

2.3.1 AlexNet

The work of Alex Krizhevsky et. al. in 2012 [5] was indeed a turning point for computer vision and deep learning research. By winning the ILSVRC challenge [7], [5] depicted the first major success of incorporating CNNs to solve large-scale image recognition tasks, after achieving state-of-the-art performance on ImageNet data set's [19] 1000-classes object classification task. Along with achieving more than 10% reduction in the top-5 error rate compared to the second best entry in ILSVRC-2012, [5] highlighted potential challenges of training large CNNs, while providing details on how they have approached these issues.

AlexNet is a 8-layer neural network. The input is expected to be a square-shaped RGB image. The input RGB image is processed by a sequence of five convolutional

layers, followed by three fully-connected layers. Since AlexNet aims to solve a multi-class classification task, a softmax layer with 1000 neurons is used as the final layer. The first and second convolutional layers used 11 and 5 as filter size, respectively, while the other three layers use 3×3 filters. Max-pooling is used as the main down-sampling operation. The total number of learnable parameters in the network is $\sim$60 million.

AlexNet designers provided empirical results supporting the importance of using non-saturating non-linear activation functions (e.g. ReLU (Equation 2.9)) when training deep neural networks, which resulted in six times faster convergence compared to using saturating activation functions (e.g. *Tanh* (Equation 2.8)).

Although the number of labelled training examples available in ImageNet is large (1.2 million), the large CNNs suffered from overfitting. To approach this issue, they have applied two label-preserving data augmentation techniques, along with applying *dropout* [15] to fully-connected layers.

2.3.2 VGG

VGG [20] is a CNN architecture that has secured the first and second places in classification and localization tasks, respectively, in ILSVRC-2014 challenge. [20]'s authors studied two important point: (1) the power of having similarly structured CNN blocks throughout the architecture, and (2) the effect of increased depth on the performance of CNNs. The two best performing variants of VGG consist of 16 and 19 layers, respectively. The same 3×3 filter size is used across all the network's convolutional layers. The authors suggest that two consecutive 3×3 convolutional layers have an effective receptive field of one 5×5 convolutional layer, while three correspond to one 7×7 layer. Using two consecutive 3×3 convolutions instead of one 5×5 allows the model to learn more complex representations, while reducing the overall number of parameters of the network.

All variants of VGG consist of five convolutional blocks, followed by three fully-connected layers. Max-pooling downsampling is applied to the output of each block. The number of learnable filters in the first block is equal to 64, and it is multiplied by $2\times$ in each subsequent block, until reaching 512. The performance of six architecture with varying depths were compared based on the performance of the classification

task, and deeper variants (16 and 19 layers) have yielded best results. The total number of learnable parameters of VGG-16 and -19 is 134 and 138 million, respectively.

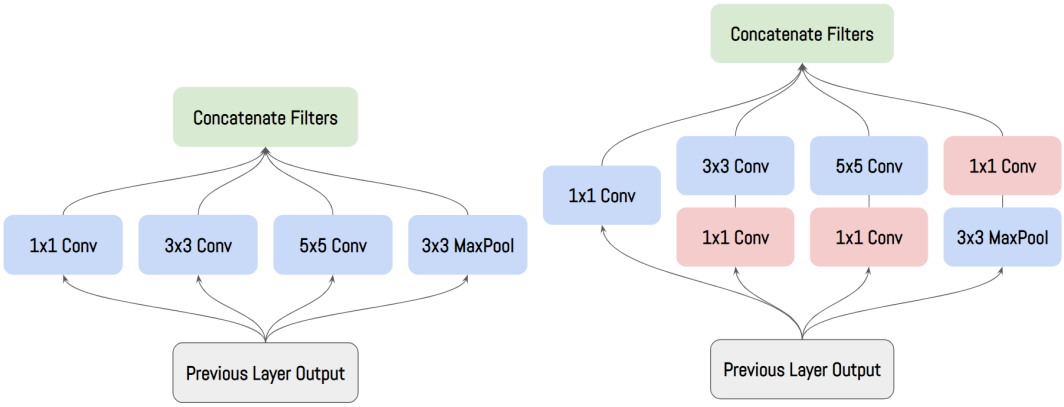
2.4 Modular Architectures

2.4.1 Inception

Inception-v1 [21]. One challenging architectural design decision in CNNs is the filter size within a convolutional layer. A small filter size learns to capture features that are concentrated in small local regions, while larger filters sizes might detect features located within larger patches. A special module was designed to address this type of issue. Such module would leverage the power of various filters sizes, {1x1 Conv, 3x3 Conv, 5x5 Conv, 3x3 Max-pooling} within the same layer, as shown in Figure 2.9a. The resulting outputs of all four layers is then concatenated in the channel dimensions to form the input for the next module. The idea then is to stack many modules on top of one another to build a deep classifier.

A downside to the module in 2.9a is the computational cost: (1) 5x5 convolutions are computationally expensive if performed in every single module, (2) filter concatenation will further increase the channels dimension, which will further increase the computational complexity when increasing the number of filters in deeper layers. To overcome such challenge, 1x1 convolutions [22] were applied before both 3x3 and 5x5 convolutions, and after the 3x3 max-pooling operations. The 1x1 convolution has is used to reduced the dimensionality of input tensors before applying convolution operation - i.e. compressing the depth dimension. Inception module (Figure 2.9b) is the name given by the authors to such module.

Nine stacked Inception modules, followed by a global average pooling layer and a final softmax classification layer, were used to build the GoogLeNet architecture, which secured the first place in ILSVRC-2014 [7] image classification challenge. Since deeper architectures are harder to train, because of the potential issue of vanishing gradients, two auxiliary softmax classifiers were added to intermediate layers, and each contributed to the total loss function by a factor of 0.33. Adding these auxiliary classifiers helped the network to converge, and reduced overfitting. The overall



(A) Naive Inception module. (B) Efficient Inception module.

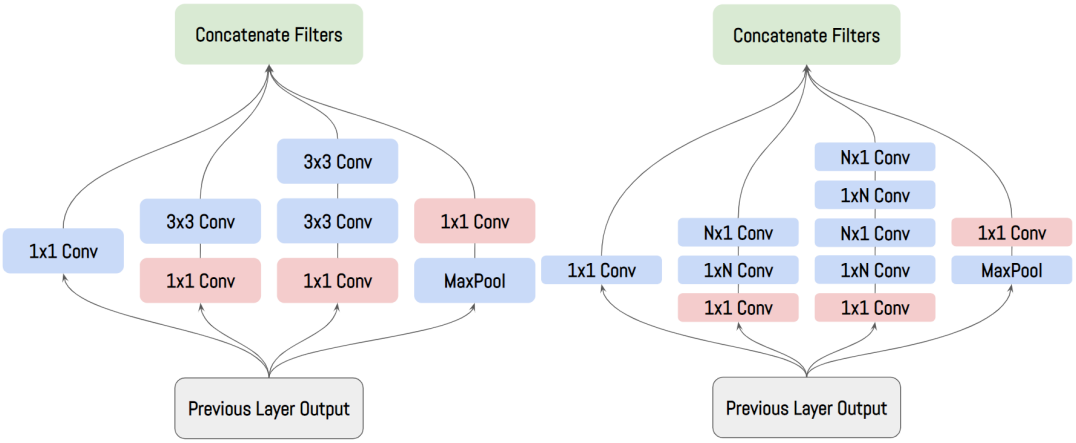
FIGURE 2.9: Demonstrating the design process leading from (A) to (B) aiming to reduce the computational complexity of the Inception [21] module by employing 1×1 convolutions to reduce the channels' dimensions.

number of learnable paramters in GoogLeNet is ∼5 million, which is 12x fewer compared to AlexNet [5].

Inception-v2 [23]. The main contribution of the Inception-v2 architecture is the factorization of convolution operation within the original Inception module, with a goal to reduce computational complexity. Similar to the idea introduced by [20], expensive 5x5 convolutions were factorized into two 3x3 consecutive convolutions as shown in Figure 2.10a, which resulted in reduced computation. Another factorization is introduced, which is splitting an NxN convolution into two consecutive operations: 1xN followed by a Nx1 convolution. This type factorization, illustrated in Figure 2.10b, yields faster computation of the convolution operations, while maintaining the same representational power. **Inception-v3** architecture was introduced in the same paper, [23], where the authors suggest that the auxiliary classifiers act as regularizers, and the overall performance of the architecture could be improved by adding either batch normalization [16] or dropout [15] to the classification layers.

2.4.2 ResNet

Very deep neural networks are hard to train due to the vanishing gradient problem that might occur during the optimization process. [12] introduce a *residual learning framework* for training significantly deep architectures with over 100 or even 1000



(A) Replacing costly 5×5 convolutions by 3×3 convolutions to reduce the computational complexity.

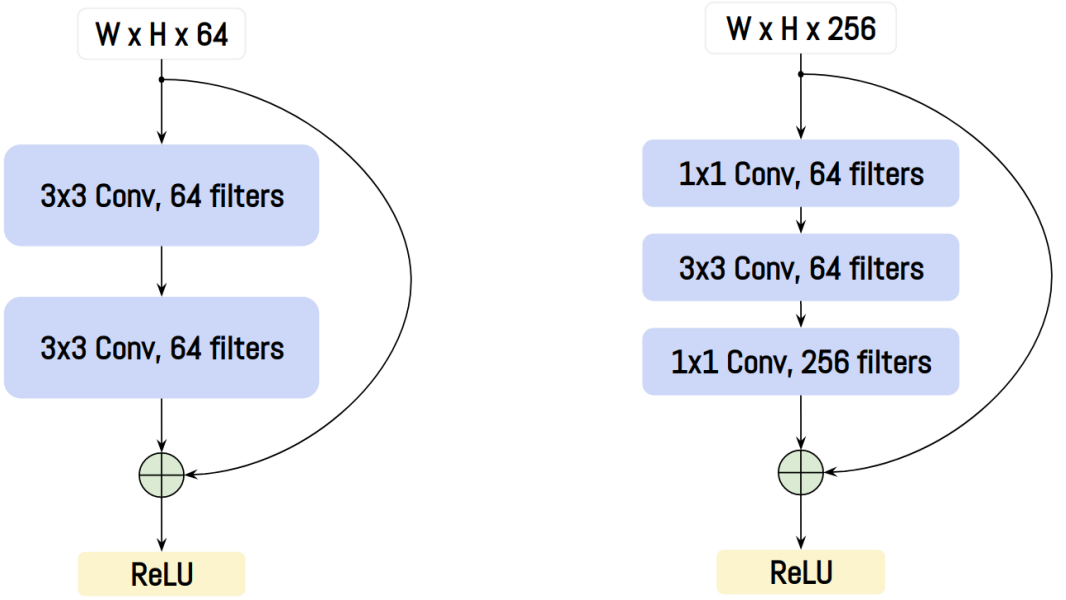
(B) Further factorized Inception [23] module.

FIGURE 2.10: Demonstration of two types of factorization applied to the original Inception module [21] in [23] aiming to reduce the computation complexity.

stacked layers, without neither optimization difficulty nor performance degradation. In fact, [12]'s groundbreaking work in 2015, namely ResNet architectures, was been able to secure the first places in ImageNet classification, detection, and localization tasks, as well as COCO detection and segmentation tasks.

The *residual learning framework* modifies the original target mapping from a collection of stacked nonlinear layers ($\mathcal{F}(x)$) to ($\mathcal{F}(x) + x$), which is the residual mapping. The residual mapping ($\mathcal{F}(x) + x$) could be implemented using an "identity skip connection", which simply skips one or more nonlinear layers, then then perform an identity mapping by adding the original input to the output of the stack of nonlinear layers ($\mathcal{F}(x)$). This simple modification to the convolutional architecture introduces no extra learnable parameters to the network. Figure 2.11a shows the structure of the basic residual block.

The authors of [12] argue that applying the identity skip connections to neural network architectures have multiple advantages: (1) support training neural networks with greatly increased depth by mitigating vanishing gradient problem, as the skip connection makes sure that the gradient will always flow from layer $\mathcal{L}$ to the layer $\mathcal{L} - t$, where t is the number of skipped layers. (2) Residual networks are easier to optimize, compared to networks with stacked nonlinear layers. Assuming the case that the optimal mapping of a collection of stacked layers $\mathcal{F}$ is the identity



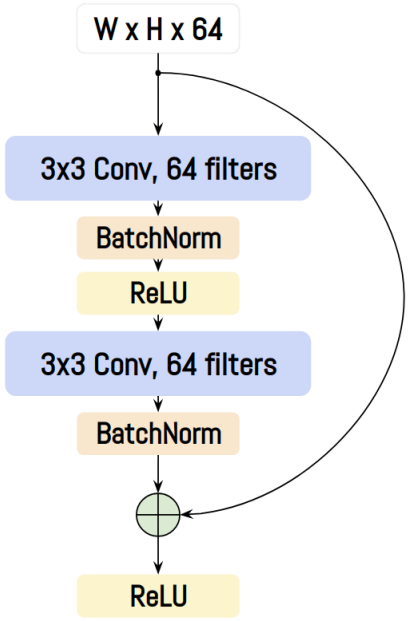
(A) Basic residual block [12] design. (B) Bottleneck residual block [12] design.

FIGURE 2.11: Demonstrating the concept of residual connections for convolutional layers. (B) shows a computationally efficient variation of the residual block, achieved by used 1×1 convolutions for dimensionality reduction.

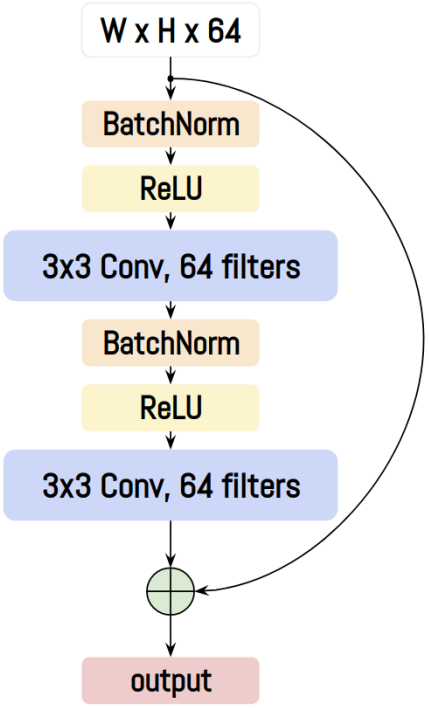
mapping, then it is easier to fit the mapping with a skip connection than a stack of nonlinear layers.

The authors of [12] introduce a family of networks with various depths, namely ResNets. The authors introduce the basic residual block in Figure 2.11a. In order to training very deep networks in reasonable training time, they have modified the block's structure to the *bottleneck* residual block in Figure 2.11b. The 1x1 convolutions are useful to reduce and increasing the channels' dimension. After reducing the number of channels, a normal 3x3 convolution is applied with a reduced number of filters. Resnets with {50, 101, 152} layers achieve higher performance compared to a 34-layers ResNet. ResNet-152 has less computational complexity compared to VGG-19, while having significantly more layers. Training an 1202-layers ResNet achieved slightly lower test error compared to {50, 101, 152}, however, it showed no issue during the optimization process.

The residual framework have then been heavily studied by multiple papers. [24] suggest that modifying the residual block's structure by reordering the batch normalization and activation operations, could result in an increased performance, especially with greatly increase depth (>1000 layers). Instead of the basic residual



(A) Original ordering of the layers within a residual block.



(B) *Pre-activation residual block.* A reordering scheme of the layers within the residual block to achieve high performance with very deep architectures.

FIGURE 2.12: The main difference between both CNN block design is the ordering of operations. (B) shows that having the right order of these operations could lead to increased performance using very deep architectures (1000+ layers).

block in Figure 2.12a, the reordering the the layers as in Figure 2.12b, namely *pre-activation* residual block, resulted in lower test error for both ResNet-101 and -164 on CIFAR-10 dataset. When incorporating the new pre-activation residual block, ResNet-1001 consistently outperform shallower counterparts, on both CIFAR-10 and -100 datasets, which was not possible with the original residual block structure in Figure 2.12a.

Further exploration of the resitual blocks was done by [25], as they've studied the possibility of a residual block with increased width (number of filters), within shallower architectures. WideResNets [25] with only 16 layers and 8 widening factor have outperformed both original [12] and pre-activation [24] ResNets on both CIFAR-10 and -100 datasets. Another variant called ResNeXt [26] have introduced the idea of "split-merge-transform" within the residual block. In each residual block, the input is split into d paths, namely the *cardinality* hyperparamter, then each path's

channels are reduced using 1x1 convolutions, a normal 3x3 convolution followed by a channels' restoration (1x1 convolution) are applied. This method aggregates different paths to the output, which resembles the idea of ensembles. ResNeXt architecture outperformed the original ResNet [12], while also scoring lower test error than Inception-ResNet [27], an architecture designed to leverage both Inception modules and the residual learning framework. When experimented on CIFAR dataset, ResNeXt yielded lower test error compared to WideResNet [25], while maintaining similar model complexity. Figure 2.13 contains a visualization of the design components of a ResNeXt module.

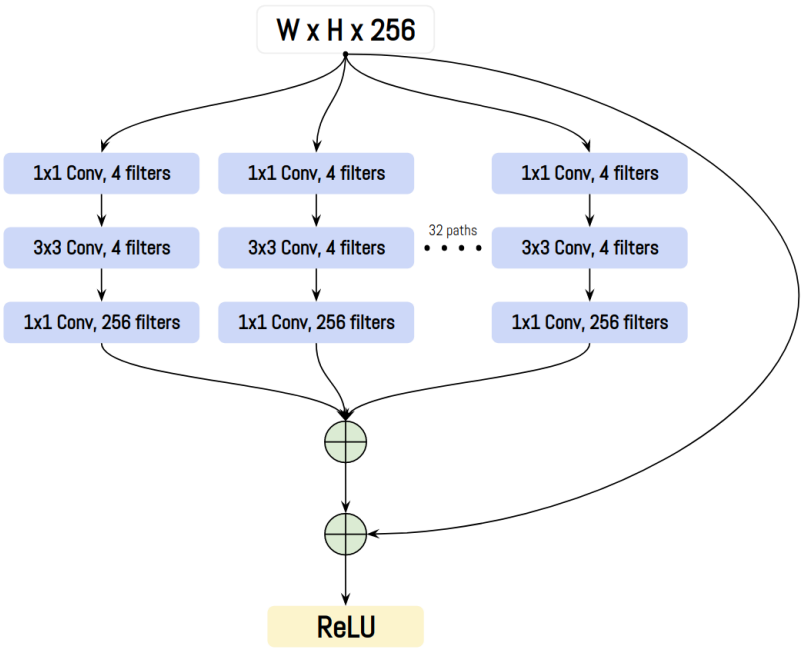


FIGURE 2.13: ResNeXt module [26]. A module that follows both "split-merge-transform" as well as the "ensembles" principle to build a family of robust classifiers.

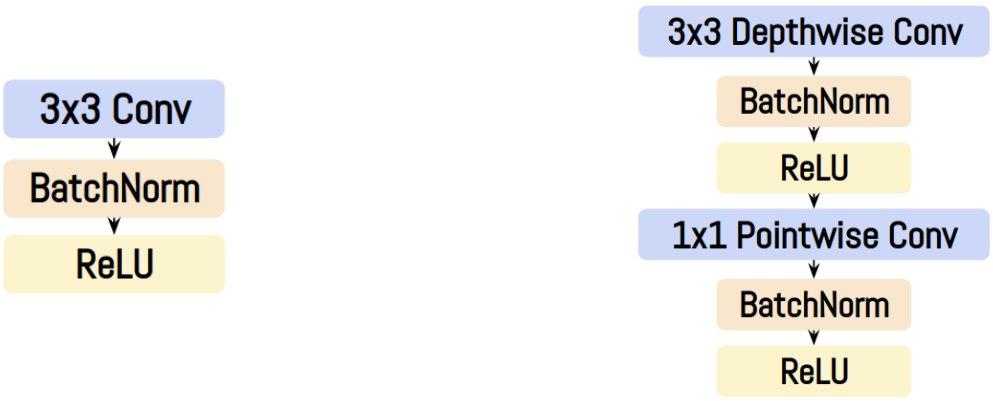
2.4.3 MobileNet

The authors of [28] introduced a new more computationally efficient type of convolution, namely *depthwise separable convolution*. The depthwise separable convolution operation splits the standard convolution into two sequential operations: (1) a channel-wise convoltuion (KxK convolution), followed by (2) a pointwise convolution (1x1 convolution). These two separated operations yields a significant reduction in the computational complexity. In a standard convolution layer, the number

of learnable parameters is computed as defined in Equation 2.5. In a separable convolution, since the first operation is applied channels-wise, the number of learnable parameters is $C \times 1 \times K \times K$, the second operation has $F \times C \times 1 \times 1$. Total number of leanable parameters in a depthwise separable convoltuion layer is denoted in Equation 2.19.

$$\begin{aligned} &C \times 1 \times K \times K + F \times C \times 1 \times 1 \\ &\text{where } F \text{ is the number of filters,} \\ &K \text{ is the kernel size,} \\ &C \text{ is the number of input channels} \end{aligned} \tag{2.19}$$

MobileNets [28] is a family of lightweight architectures designed using the depthwise separable convolution module. When trained on the ImageNet dataset, MobileNet achieved only 1% lower in classification accuracy compared to a fully convolutional architecture, while having around 7x reduction in the number of parameters, and around 8x reduction in the number of Million Multiplication-Addition operations. MobileNet [28] performed similarly compared to the state-of-the-art methods in multiple tasks (e.g. classification, detection), while having a tremendous reduction in the computational complexity, which enabled it to be used in for applications that require real-time inference with high accuracy. Figure 2.14 demonstrates the design difference between a normal convolutional block and a separable convolution.



(A) Basic convolutional block.

(B) Separable convolution operation [28].

FIGURE 2.14: A tremendous reduction in computational complexity could be achieved by successfully separating (B) the operations of a basic convolutional block (A).

Chapter 3

Literature Review

In this chapter, we provide an extensive and detailed review of the previous work found in the literature about: 2D object detection (Section 3.1), available data set and benchmarks (Section 3.2), LiDAR-only object detection methods (Section 3.3), camera-only object detection methods (Section 3.4), and fusion-based object detection methods (Section 3.5).

3.1 Two-dimensional Object Detection

A robust object detector is one of the most critical components of autonomous vehicles' perception systems. The problem of object detection comes with a huge complexity, as objects can be found in different aspect ratios, various colors, or even occluded. The object detectors deployed on autonomous vehicles' perception systems are usually expected to run in real-time, with high detection accuracy, detecting all the objects in the surrounding environment and drawing a bounding box around each of them, while also identifying the class and the corresponding confidence score of each bounding box.

One naive approach would be to build a multi-classes classifier for the target classes (e.g. car, pedestrian), and train such classifier on well-cropped and centered images that contain the target classes. Then performing multi-scale sliding window operations (potentially with various stride values) on the original input image, where each window would be cropped and passed to the trained classifier to know whether or not one of the target classes is found, and its corresponding confidence score. One huge disadvantage of such approach is the computational complexity.

Many regions are being cropped and processed independently through a CNN classifier. Aiming to reduce the computational cost by using smaller stride value would directly hurt the performance and detection accuracy of such model. In the next subsections, we will discuss the ideas behind building robust deep learning-based object detection algorithms.

3.1.1 Two-stage Methods

The two-stage object detection framework has resulted in state-of-the-art results for object detection, with a main focus to reduce the number of regions of interest (RoI) of that are cropped and passed through a CNN for classification, while also aiming to optimize the quality of those RoIs. The two stages refer to (1) the generation of region proposals, followed by (2) classification and regression of these proposals to output the final detections.

R-CNN [29] suggested the use of selective search to extract a fixed number of region proposals per image (e.g. $\sim$2K), then combining similar proposals together and passing them through a CNN feature extractor. Finally, the extracted feature maps are fed into an SVM to decide both the presence of an object, and the class of the potential object within each RoI. While achieving good detection accuracy, R-CNN's training takes a lot of time, as each of the $\sim$2K proposals are being processed and fed to a CNN followed by an SVM. For the same reason, R-CNN was not used suitable for real-time applications. The other disadvantage would be the use of a fixed algorithm (selective search) rather than giving the network the ability to learn the object detection task.

Fast R-CNN [30] introduced a faster object detection framework, compared to R-CNN, by aggregating input image and the RoI (obtained using selective search) into one pre-trained classification CNN, where all the features are shared between both inputs. Then, for each RoI, we pass the output of the CNN to an RoI Pooling layer, followed by a fully connected layer. The RoI pooling layer extracts the important features from the output of the CNN, based on which the final detection of the RoI is done. The final detection step consists of a softmax layer for classification, and a linear layer for regressing the bounding box information. The training is done in a single-stage, and both the training and testing speed of Fast R-CNN are much better

compared to the original R-CNN [29]. The main contributor to the boost in the speed is sharing the CNN inference between the image and region proposals.

Faster R-CNN [31] came up with a framework that would learn to generate region proposals instead of generating them using selective search algorithm. Similar to Fast R-CNN [30], the input image is passed through a pre-trained CNN feature extractor (e.g. VGG16 [20]). Then the extracted features are used as input to the RPN (Region Proposal Network) to generate region proposals. Regional proposals generated by RPN are simply bounding boxes that have high probability of containing objects. Finally, the output of the RPN, along with the extracted feature maps from the pre-trained CNN, are fed into a final detection layer, which is similar to the one used in Fast R-CNN [30]. An architectural overview of all three members of the R-CNN family are demonstrated in Figure 3.1.

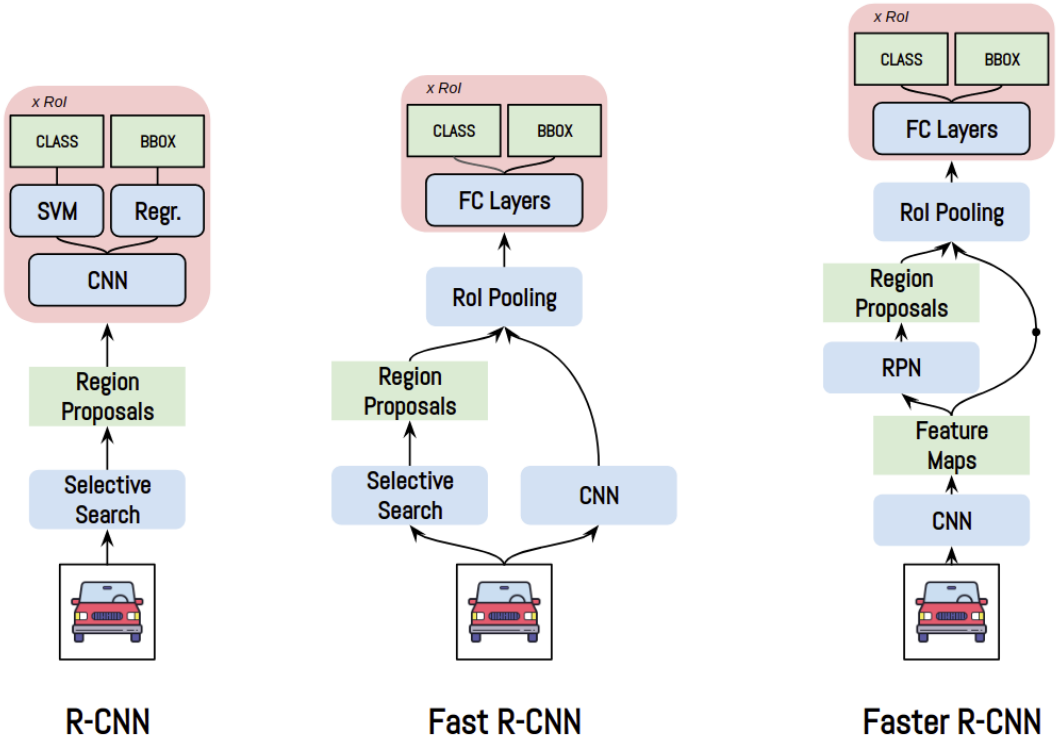


FIGURE 3.1: Demonstration of R-CNN family of two-stage object detectors. R-CNN [29] (left), Fast R-CNN [30] (middle), and Faster R-CNN [31] (right).

3.1.2 Single-stage Methods

The single-stage detectors (SSD) are mainly targeting applications where real-time processing and detection is required. A fully convolutional neural network is used in an end-to-end setting for both training and inference modes, where the input of the detector is an image, and the output is a list of bounding boxes along with their corresponding classes. The term "fully-convolutional" is used when the fully-connected layers in the final stages of a CNN are replaced by 1x1 convolutions, which have the same effect, but computationally more efficient. An SSD would map the input image to an output grid map of size ($\mathcal{S} \times \mathcal{S}$), where each grid cell is responsible for detecting a fixed number of bounding boxes with their corresponding classes. In theory, the SSD setting is similar to sliding a CNN classification network through an image $\mathcal{S} \times \mathcal{S}$ times. However, in practice, the SSD setting is a robust end-to-end learner, that would find meaningful relations between features, and learn the shapes and aspect ratios of different objects.

YOLOv1 [32] (You Only Look Once) is a popular single-stage, real-time object detector, based on the idea of dividing the output map into 7×7 grid cells, where each grid cell predicts a fixed number ($\mathcal{B}$) of bounding boxes, but only the box with the highest confidence score out of the $\mathcal{B}$ boxes is detected. For each bounding box detection, YOLOv1 [32] predicts (1) an objectness score, which denotes whether or not there is an object in the grid cell, (2) the class probability of the potential object, (3) bounding box information, such as: width, height, center coordinates offsets to the corresponding grid cell. All bounding box information are normalized to have values between $0 - 1$. Instead of employing a fully-convolutional architecture, YOLOv1 passes the original image through 24 convolutional layers, followed by 2 fully-connected layers, whose output tensor is then reshaped to have a spatial dimension of $\mathcal{S} \times \mathcal{S}$.

As an initial step, the convolutional layers are pre-trained on ImageNet [19] dataset for the classification task at half of the input image resolution, then the input image resolution is doubled for object detection training. During training, the detection accuracy of the predicted bounding boxes is computed based on the overlapping area, also known as Interstion over Union (IoU), between the groundtruth

and the predicted bounding boxes. Higher IoU values denote more accurate detections. During inference, since YOLOv1 [32] architecture can predict multiple detections for the same object, a post-processing step called Non-maximum Suppression (NMS) is applied. NMS post-processing is applied to preserve detections having high confidence scores by removing other detections with an overlapping area (IoU) higher than a specific thershold (e.g. IoU > 0.5). Being a single-stage object detector, YOLOv1 [32] can be trained in an end-to-end fashion, while having a real-time inference speed (~45 FPS). YOLOv1's complete object detection architecture is visualized in Figure 3.2.

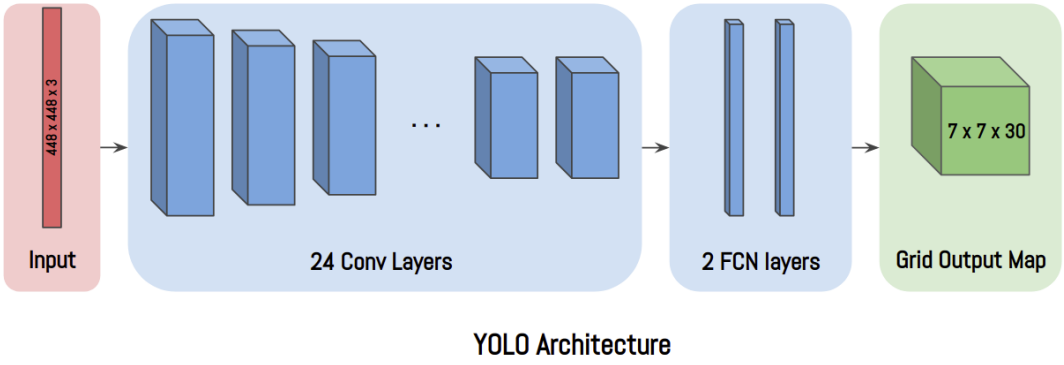


FIGURE 3.2: Architecture overview for YOLOv1 [32].

SSD (Single Shot Multibox Detector) [33] is another architecture that has outperformed both Faster R-CNN [31] and YOLOv1 [32] in term of detection accuracy and inference speed. SSD consists of a CNN feature extractor (e.g. VGG16), followed by a number of multi-scale feature maps. Each of the multi-scale feature maps' output is divided into grid cells, similar to YOLOv1 [32]. However, in SSD implementation, each grid cell has a fixed number of predefined (anchor) boxes (e.g. 4) which are obtained from the training set, hence the architecture name includes the keyword *multibox*. The *multi-box* idea is further illustrated in Figure 3.3. Similar to YOLOv1 [32], a set of values are being learnt for each of the predefined boxes: (1) width and height of the bounding box, (2) offset values from the center, (3) probability of each output class. The purpose of using multi-scale feature maps is to be able to detect object at different scales. Using the multibox method, and multi-scale feature maps, SSD is able to generate 8732 bounding boxes per class in a single stage. Bounding

boxes with low confidence scores are discarded, then NMS post-processing is applied to remove multiple detections of the same object.

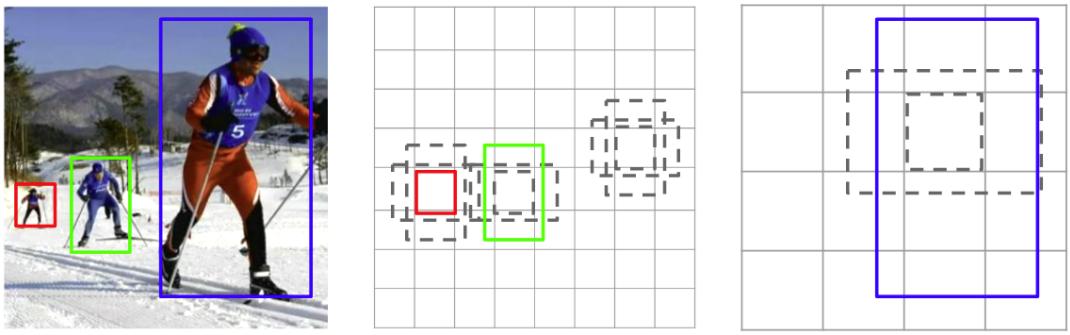


FIGURE 3.3: Overview of how SSD [33] detection algorithm leverages both multi-scale feature maps and multiple pre-defined (anchor) boxes to be able to detect objects with variant scales.

SSD [33] also employs data augmentation techniques during training, which improves both the performance and the model's generalization capability. In order to overcome the issue of huge class imbalance between background and foreground pixels in the object detection problem, SSD uses hard negative mining, which is a technique that would sort all the output boxes by confidence score, then picks only top-K for training and back-propagation. An architectural overview of SSD is demonstrated in Figure 3.4.

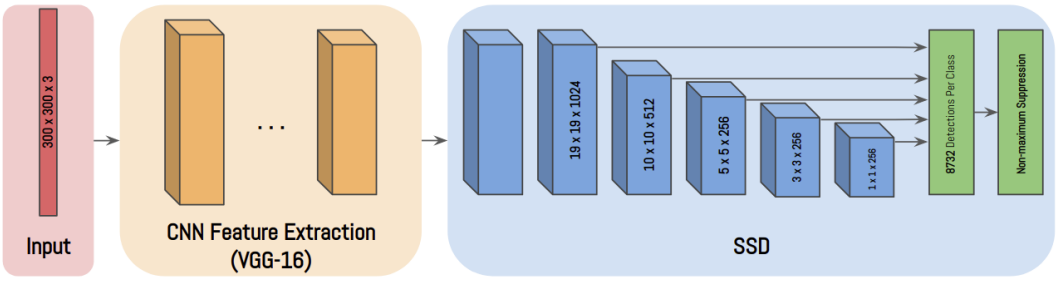


FIGURE 3.4: Architecture overview for SSD [33]

YOLOv2 [34] comes with further modifications on YOLOv1 [32] to outperform SSD [33]. Some of the reasons behind the increase in detection accuracy are: (1) a more robust backbone network (DarkNet), (2) a fully-convolutional architecture (both feature extractor and detector), (3) the use of batch-normalization [16], and (4) employing anchor (predefined) boxes (similar to SSD).

Object Detector	mAP (%)	FPS
R-CNN [29]	66.0	0.02
Fast R-CNN [30]	70.0	0.5
Faster R-CNN [31]	73.2	7
YOLOv1-448 [32]	63.4	45
YOLOv1 (VGG-16) [32]	66.4	21
SSD-300 [33]	74.3	46
SSD-512 [33]	76.8	19
YOLOv2-416 [34]	76.8	**67**
YOLOv2-480 [34]	77.8	59
YOLOv2-544 [34]	**78.6**	40

TABLE 3.1: Comparison of both performance and inference speed on PASCAL VOC 2007 [35] dataset. Inference time is computed on a Geforce GTX Titan X GPU.

RetinaNet [13] authors have introduced a strong backbone network based on the idea feature pyramids (originally inspired by image pyramids), namely FPN [36] (Feature Pyramid Network). RetinaNet employs a RestNet [12] as its feature extractor. Then a bottom-up pathway is constructed by using the last layer of each convolutional block in the feature extractor. Bottom-up pathway's feature maps are then up-scaled and merged (element-wise summation) to a top-down pathway through lateral connections. Both the bottom-up and top-down pathways build the FPN. FPN solves the issue of detecting objects with various scales, without adding up extra computational complexity. Two additional fully-convolutional networks are attached to each of the FPN levels, namely classification and regression subnets. The output feature maps of the subnets is divided into grid cells, where each grid cell has predefined (anchor) boxes with various sizes and scales.

The key contribution by RetinaNet [13] authors is the focal loss 2.14. Focal loss is used mainly to deal with the issue of the huge class imbalance between foreground and background pixels, which has a great impact on the performance of object detectors. Focal loss employs two hyperparameters to a modified version of the cross-entropy loss: (1) a modulating factor γ, which drives to model to focus more on hard examples (foreground) and down-weight the easy examples (background), and (2) a weight factor α, which is used to address class imbalance. RetinaNet significantly outperforms SSD [33] which uses hard negative mining to deal with class imbalance, as well as YOLOv2 [34] which uses a specially designed backbone architecture. RetinaNet also outperformed some high performing variants of Faster R-CNN, which

could be counted as a proof that single-stage detectors can outperform two-stage detectors in performance if the class imbalance problem is addressed during training, while at the same time having a real-time inference speed. An architectural overview of RetinaNet is demonstrated in Figure 3.5. Quantitative results for both two- and single-stage detectors on PASCAL VOC [35] and COCO [37] object detection datasets could be found in Tables 3.1 and 3.2, respectively.

Object Detector	**mAP (%)**	**FPS**
SSD-321 [33]	45.4	16
SSD-513 [33]	50.4	8
YOLOv2-608 [34]	48.1	**40**
Retinanet-50-500 [13]	50.9	14
Retinanet-101-500 [13]	53.1	11
Retinanet-101-800 [13]	**57.5**	5

TABLE 3.2: Comparison of both performance and inference speed on COCO [37] dataset.

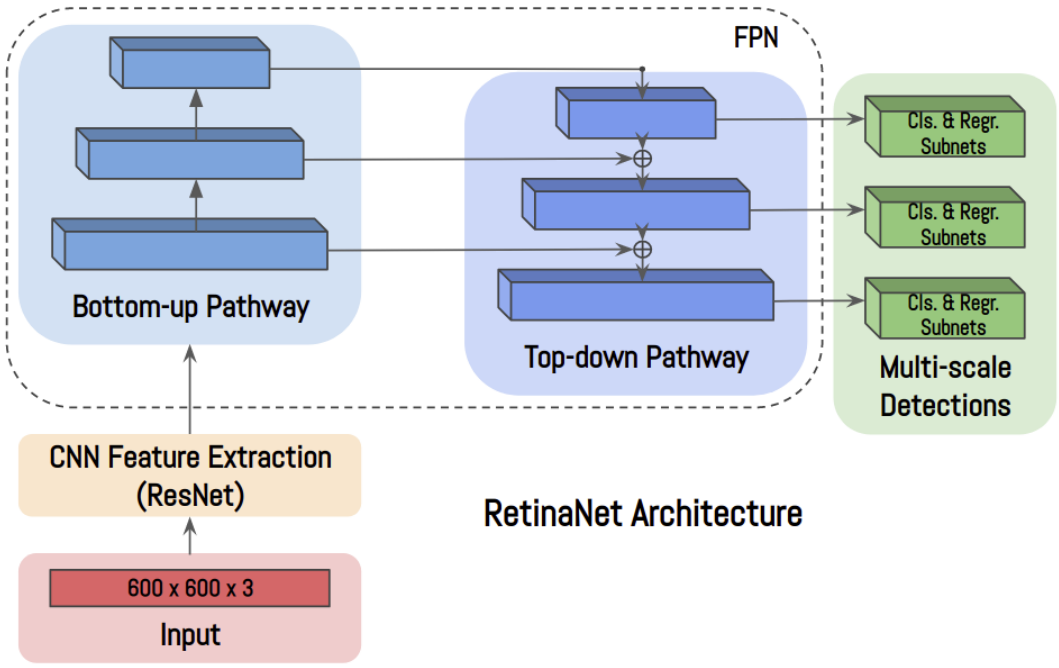


FIGURE 3.5: Architecture overview for RetinaNet [13]

3.2 Datasets

While there exist multiple publicly available datasets in the context of the autonomous driving, the KITTI [6] dataset stands out to be the most well-established benchmark

in the field. One major drawback of KITTI is the dataset size, which is composed of 14999 scenes, divided equally between training and testing. Deep learning models usually require alot of training data to produce accurate results, hence, it is common that data augmentation are used to increase the models' generalization when benchmarked on KITTI. Another publically available data set is NuScenes' [38], which is a large-scale dataset, with more than 100 thousand examples for training, and provides very rich multimodal sensory data. Each frame contains the following: a full 360 degrees LiDAR sweep, radar point clouds collected from five radars all around the car, as well as six camera images covering the surrounding area of the autonomous vehicle. NuScenes provides a larger range of object classes, along with some useful attributes. Although NuScenes' dataset is still not widely adopted as a benchmark in the object detection field, it has a great potential in the future as it solves many of KITTI's shortcomings, such as: data set size, variety of annotated objects, objects' attributes, and adding radar sensor readings.

3.2.1 KITTI Dataset

To train and validate our sensor fusion framework, we use the KITTI dataset [6], a well-established benchmark for 2D, 3D, and Bird's Eye View (BEV) detection tasks in the context of autonomous driving. KITTI [6] provides multiple publically available datasets for a number of critical tasks for autonomous vehicles' perception systems, including: object detection, object tracking and segmentation, pixel and instance level segmentation, depth estimation and completion, and more. KITTI also provides a benchmarking platform for 2D, 3D, and BEV object detection tasks for three different classes: cars, pedestrians, and cyclists.

The data available for the object detection task is divided into 7481 frames for training, and 7518 for testing and benchmarking, containing a total of $80K$ labelled objects. KITTI provides calibrated, multi-modal data, including: (1) colored RGB images obtained from two cameras (may be used in either a stereo or a monocular setup), (2) raw LiDAR point clouds, (3) accurate camera-LiDAR calibration parameters. The annotated labels are only available for the training set. The annotations are divided into three main difficulty levels: easy, moderate, and hard. An annotated

object is included in a specific category based on the size of the object, and its occlusion level. The test data is only available on KITTI's evaluation server, thus, the training data is usually split nearly in half to produce training and validation splits, of size 3712 and 3769, respectively, according to [39].

3.2.2 KITTI Performance Metrics

In our work, we use KITTI's [6] performance and evaluation metrics for 2D, 3D, and BEV object detection. A bounding box is considered a detection - i.e. *true positive* (TP) if the intersection-over-union (IoU) between the prediction and the groundtruth is greater than or equal to a specific value, otherwise, it is considered as a *false positive* (FP). Then, average precision (AP%) with 40 recall points is used to generate final detection accuracy for *easy*, *moderate*, and *hard* cases.

Precision is a measurement 3.1 that indicates how accurate is our detection model in generating predictions - i.e. the percentage of correct bounding boxes. *Recall* is a measurement 3.2 that indicates how accurate is our model in detecting all groundtruth boxes. After applying IoU to know the correct detections, the bounding boxes are sorted decreasingly according to their confidence score. A *precision-recall* (PR) curve can then be constructed by iterating over the final detections and computing both precision and recall values at each data point.

$$\begin{aligned} &Precision = \frac{TP}{TP+FP} \\ &\text{where } TP \text{ is the number of true positives, and} \\ &FP \text{ is the number of false positives} \end{aligned} \tag{3.1}$$

$$\begin{aligned} &Recall = \frac{TP}{TP+FN} \\ &\text{where } TP \text{ is the number of true positives, and} \\ &FN \text{ is the number of false negatives} \end{aligned} \tag{3.2}$$

The definition for *average-precision* (AP) 3.3 is to find the area under the *precision-recall* (PR) curve. Since both precision and recall have values between 0 and 1, AP will also have a similar value range. Interpolation 3.3 is used to compute AP in order to account for variations found in the PR curve. To calculate AP, recall is divided into R equally spaced levels (e.g. $R_{11} = \{0, 0.1, 0.2, ..., 1.0\}$). As [40] proposed, using AP_{40}

results in a more fair comparison of the detection models evaluated on the KITTI dataset, compared to AP_{11}, which had a glitch at the lowest recall bin. [40] suggested using 40 recall levels as well as not using 0 as a recall level to overcome the issue.

$$\begin{aligned} AP_R &= \frac{1}{R} \sum_{r \in R} p_{interp}(r) \\ p_{interp}(r) &= \max_{r':r' \geq r} p(r^{'}) \end{aligned} \tag{3.3}$$

3.3 LiDAR-only Object Detection

In this section, we review a number of the proposed methods for the problem of 3D object detection based only on LiDAR point-clouds as the main input sensor modality. Such methods use different approaches in order to deal with the unstructured and sparse nature of LiDAR point-clouds. In subsection 3.3.1, two popular point-cloud processing methods are briefly explained. Then, both volume-based and projection-based methods are reviewed in subsections 3.3.2 and 3.3.3, respectively.

3.3.1 Point-cloud Processing Methods

PointNet [41]. PointNet is one of the first methods proposed to directly process and learn from raw point-clouds for the tasks of classification and segmentation, without the need to transform the point-cloud's original unstructured representation into a structured format. In order to deal with the permutation and transformation invariance of the coordinates of a point-cloud, PointNet relies on the maxpooling operation, which is a simple, yet effective, symmetric function. Although the simplicity of the method, PointNet is robust to both outliers and missing data. It is worth noting that the idea of PointNet was experimented in a setting where $\sim 1K$ points only where sampled from a point cloud to perform either classification or segmentation. On the other hand, a typical LiDAR scene in the autonomous driving domain would contain $\sim 100K$ points.

PointNet++ [42]. PointNet++ paper was introduced to address an important issue with PointNet [41], which is the lack of capturing local structures from the input point-cloud. PointNet architecture is similar to applying multiple consecutive fully-connected layers to process a specific input. The idea behind PointNet++ is

to process overlapped partitions of the input point-cloud through PointNet, which would give the framework the ability to capture hierarchical structure within the input point-cloud.

3.3.2 Volume-based Methods

VoxelNet [43]. PointNet's [41] work was a huge step towards learning effectively from raw point-cloud data. However, and as mentioned in 3.3.1, the framework was intended to small-sized point clouds ($\sim 1 - 5K$ points). To be able to extend such robust learners to the task of 3D object detection, VoxelNet has proposed a single-stage detector that operates on LiDAR point-clouds. VoxelNet discretize the 3D space into equally spaced 3D voxels, where each voxel is either empty, or contains a group of LiDAR points. Then, only the non-empty voxels are passed through a sequence of Voxel Feature Encoding (VFE) layers, which is responsible for learning hierarchical feature encodings of each non-empty voxel. The encoded voxel-wise features are then passed through a sequence of 3D convolutional blocks. Finally, the output feature map of the 3D CNN is passed through an Region Proposal Network (RPN) that aggregates high- and low-level feature maps. The output objective of VoxelNet's RPN is similar to the one defined in Faster R-CNN [31]. The output targets are being predicted from the BEV perspective, where it is easier detect objects given that all of the target objects lie on the same ground, and none of them have overlapping areas.

VoxelNet employed the idea anchors for the output targets. Two anchors were defined per grid-cell, while varying only the orientation ($\theta_{k_1} = 0°, \theta_{k_2} = 90°$). The regression head included: width, length, height, Euclidean coordinates (x, y, z), and orientation of target bounding boxes. Binary cross-entropy loss was used as objective for the classification head, and smooth L1 loss for regression head. VoxelNet architecture was trained on the small-sized KITTI data set. In order to overcome overfitting issues, the authors have applied three types of data augmentation to LiDAR point-clouds: (1) translation per groundtruth box, (2) global scaling, (3) global rotation. Both global augmentation operations are applied to all the boxes, as well as the point-cloud. Even though VoxelNet had superior detection accuracy compared to other LiDAR- and camera-only methods, it had high computational complexity

due to the use of 3D convolutions for processing the sparse encoded features. Figure 3.6 contains a high-level overview of the VoxelNet network architecture.

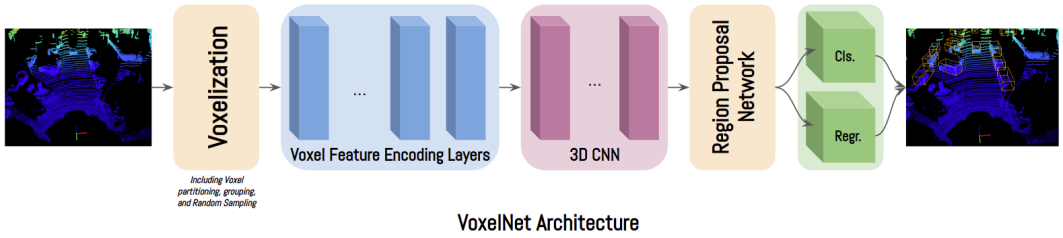


FIGURE 3.6: Architecture overview for VoxelNet [43].

PointPillars [44]. After VoxelNet's success in building an end-to-end 3D object detection framework that would mainly rely on learning feature encodings based on sparse point-clouds, PointPillars introduced another novel point-cloud encoding technique, where the focus is more on both the computational efficiency and the robustness of the detector. VoxelNet's bottleneck in inference speed was the 3D convolutions that are required in order to extract features from sparse encoded voxel-wise features.

Firstly, PointPillars' end-to-end framework is based on 2D convolutions, instead of the computationally expensive 3D convolutions. In order to be able to represent sparse point-cloud in a structured format so that it could be easily processed by 2D convolutions, the authors divide the 3D space into pillars in the $x - y$ plane instead of partitioning it into 3D voxels. Encoding the point-cloud into this pillar representation eliminates the need to choose a binning factor for the vertical dimension of the 3D space. Then, a maximum of P non-empty pillars, where each pillar has a maximum on N points, are processed through a PointNet [41] to learn the corresponding pillar encoding. The pillar-wise encodings are then combined to form a 2D top-view pseudo-image. The 2D top-view pseudo-image is then fed into a 2D CNN backbone network for feature extraction (Figure 3.7), followed by a 2D SSD-like [33] network to produce the final bounding box detections.

The objective functions used to train PointPillars' [44] end-to-end framework are: (1) the focal loss [13] for the classification head, and (2) the smooth l_1 loss for the regression head. Predefined anchors are used, similar to [43] in terms of the orientation, while other regression values are set as the mean values from all the

groundtruth boxes. Data augmentation is key in in this work, heavy data augmentation techniques are applied, including: individual boxes' rotation and translation, global rotation, translation, and scaling, as well as global random mirrored flipping along the x-axis. One of the most important augmentations they have applied is to randomly add points of other groundtruth boxes extracted from different scenes into the current scene. The latter augmentation technique is very effective, and was initially mentioned in a framework called SECOND [45], which is another voxel-based technique aiming to increase the inference speed of VoxelNet by replacing 3D convolutions by sparse convolutions.

PointPillars [44] has been experimented on KITTI [6] benchmark, and outperformed other voxel-based method [43], [44], while having the fastest inference speed among 3D object detectors, including projection-based methods. The main contributor to this boost in inference speed is using NVIDIA TensorRT for optimizing the deep learning architecture for GPU inference. PointPillars achieved state-of-the-art performance on KITTI benchmark, while running at 62 Hz., which was three times faster compared to the second fastest benchmarked method at the time of publishing PointPillars' paper. Table 3.3 contains a comparison between [43]–[45] BEV detection accuracy on KITTI [6] test benchmark.

Object Detector	**Input Modality**	**Easy**	**Moderate**	**Hard**	**Speed (Hz)**
VoxelNet [43]	LiDAR (*Voxels*)	**89.35**	79.26	77.39	4.4
SECOND [45]	LiDAR (*Voxels*)	88.07	79.37	77.95	20
PointPillars [44]	LiDAR (*Pillars*)	88.35	**86.10**	**79.83**	**62**

TABLE 3.3: Results for Car detection from BEV on KITTI [6] test benchmark ($AP_{0.7}$).

3.3.3 Projection-based Methods

Bird's Eye View Projection

PIXOR [46]. PIXOR is a single-stage three-dimensional object detector, which attained fairly high detection accuracy, while having real-time performance, by representing the LiDAR point-cloud only from the top-view perspective, by generating a *3D occupancy grid*. A 3D occupancy grid discretizes the physical space ($L \times W \times H$)

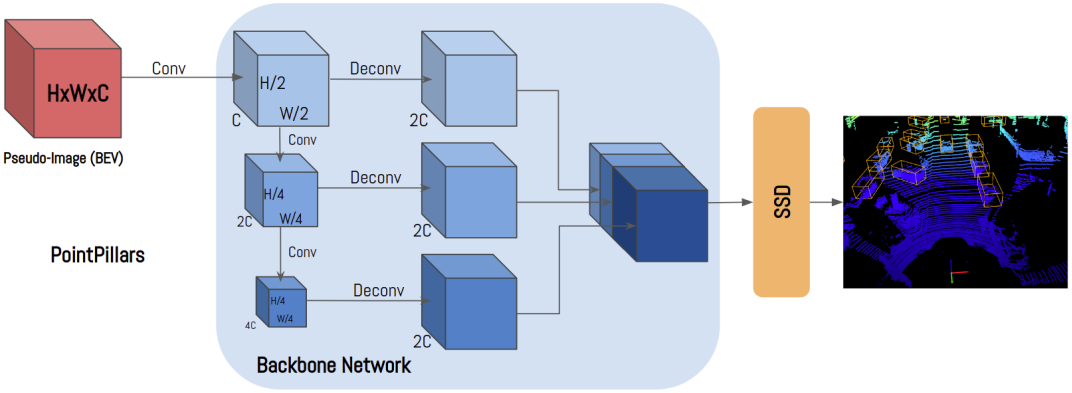


FIGURE 3.7: Backbone network architecture overview for PointPillars [44]. Each *Conv* operation is followed by batch normalization and ReLU non-linearity.

using a discretization factor ($d_L \times d_W \times d_H$), where the value of each grid cell depends on the presence of LiDAR points inside it (0 for empty, and 1 for non-empty cells). The output of the discretization operation yields a 3D grid of shape ($\frac{L}{d_L} \times \frac{W}{d_w} \times \frac{H}{d_H}$), where $\frac{L}{d_L}$ and $\frac{W}{d_w}$ are both representing the spatial dimensions of the discretized physical space, the third dimension of the grid encodes the height information, and would be processed by a 2D CNN similar to RGB channels in a 2D image. Projecting LiDAR point-clouds into BEV representation is computationally more efficient compared to voxelization, while also simplifying the learning task as all objects are non-overlapping and lie on the same ground.

PIXOR [46] neither depends on region proposals nor pre-defined anchors. Fully-convolutional architecture is employed to perform end-to-end learning. The convolutional network is composed of two parts: a backbone network for feature extraction, followed by a header network for generating the final detections. The backbone network has FPN-like [36] design consisting of a bottom-up branch with a downsampling factor of 16, followed by a top-down branch which upsamples the features map to get a a total downsampling factor of 4. All convolutional blocks have residual connections [12]. The header network consists of a sequence of convolutions, followed by classification and regression heads. The classification head is responsible for predicting the pixel-wise objectness, and is optimized using the focal loss [13] to address the foreground-background class imbalance. The regression head is optimized using the smooth L1 loss. The heading angle of the boxes has a valid

range of $[-\pi, \pi]$, and is encoded into $(\cos(\theta), \sin(\theta))$. The width and length of the boxes are log-scaled, then all the output regression values are normalized to have zero-mean and unit-variance. Also, when constructing the target objectness map, boxes' boundaries are ignored, which stabilizes the training process and improves the performance.

After decoding the output objectness and regression maps, NMS post-processing is applied to keep the final, non-overlapping detections. PIXOR outperform other BEV-based methods (at the time of the publication) in AP on KITTI benchmark, while running at $\sim 28Hz$, ~ 7 times faster than the second best inference time [47], and ~ 10 times faster than the second best detection accuracy [47]. It is worth noting that PIXOR [46] has real-time performance without being optimized by NVIDIA TensorRT library, as opposed to PointPillars [44]. PIXOR's high-level architectural overview is demonstrated in Figure 3.8.

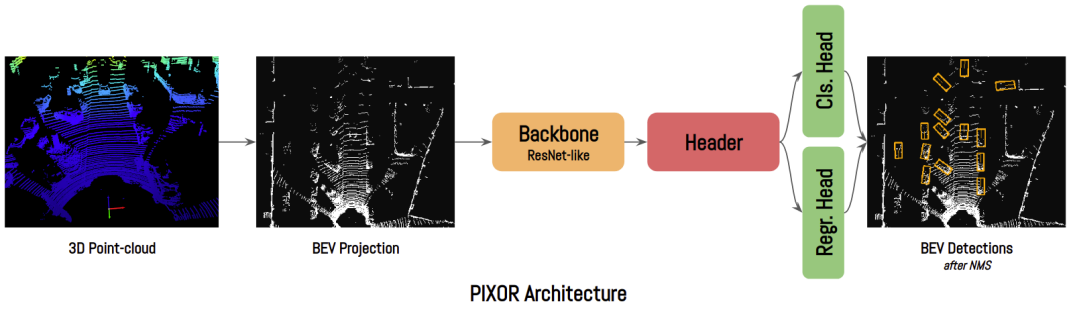


FIGURE 3.8: High-level overview for PIXOR [46] end-to-end BEV detection framework.

HDNet [48]. PIXOR++ provided improvements based on the work of PIXOR [46], and is introduced in the HDNet paper. HD maps were used as priors to the end-to-end detector, by being concatenated with the discretized input point-cloud. The detector benififed from ground information, as well as semantic road masks provided by the HD maps. In order to deal the challenging scenarios where HD maps are either noisy or unavailable, data-dropout was applied on these priors to ensure the model's robustness, even in the absence of priors. The backbone network has been modified to be a sequence of convolutional blocks with a downsampling factor of 16. Then multi-scale feature aggregation is used to retain a total downsampling factor of 4, similar to [46]. The aggregated multi-scale features are then fed into the

header network, which is followed by classification and regression heads. The target encoding is similar to PIXOR [46], except for the orientation encoding which is encoded as $(\cos(2\theta), \sin(2\theta))$. The baseline PIXOR++ had an increased detection accuracy when compared to PIXOR [46], while having the same real-time performance ($\sim 28Hz$). Employing HD maps helped further increase the detection accuracy on KITTI benchmark, while running at ($\sim 20Hz$). The modified PIXOR++ architecture is demonstrated in Figure 3.9.

Object Detector	Input Modality	Easy	Moderate	Hard	Speed (Hz)
PIXOR [46]	LiDAR (*BEV*)	86.79	80.75	76.60	35
PIXOR++ [48]	LiDAR (*BEV*)	**89.38**	83.70	77.97	35
PointPillars [44]	LiDAR (*Pillars*)	88.35	**86.10**	**79.83**	**62**

TABLE 3.4: Comparison between [46], [48] and [44] performance. Results for Car detection from BEV on KITTI [6] test benchmark ($AP_{0.7}$).

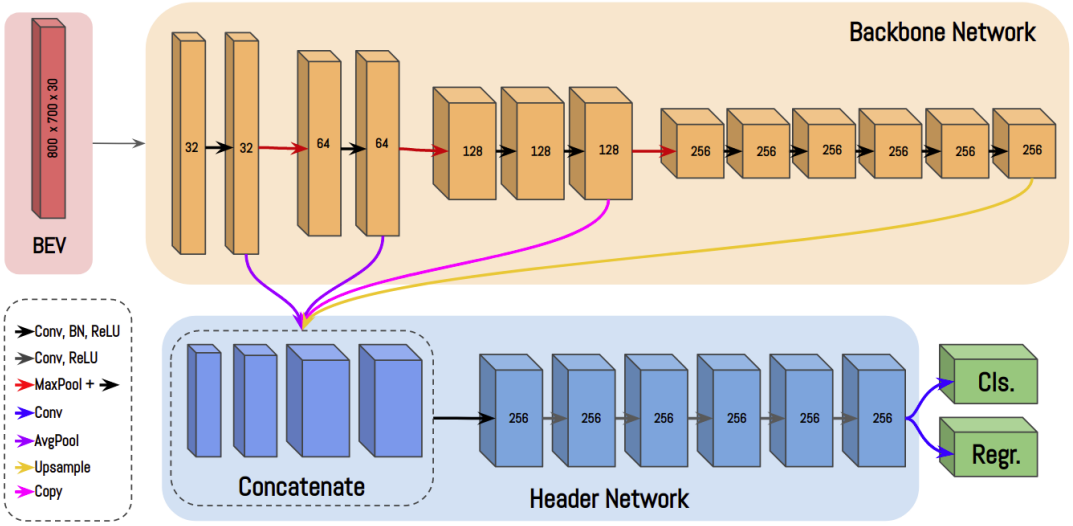


FIGURE 3.9: Architectural design for PIXOR++ [48] end-to-end BEV detection framework. All convolution operations have kernel size $k = 3$.

Frontal View Projection

FVNet [49]. FVNet is a two-stage 3D object detector, that operates on raw LiDAR point-clouds. FVNet's input is generated by projecting the point-cloud into a cylindrical frontal-view representation, which is used to directly generate 3D region proposals. FVNet uses a modified version of an region proposal network (RPN) that extends the 2D targets into 3D by leveraging the depth information provided by the

LiDAR point-cloud. Along with predicting the box's size and position, two extra values are regressed to denote the distances to the front and back of the bounding box within the LiDAR point-cloud. Then, for each of the proposed 3D regions, the corresponding points are cropped from the original point-cloud, and each cropped region is then passed through a variant of PointNet [41] to directly estimate the bounding box information such as position, size, and orientation. To address the scale-variance issue introduced due to using frontal-view maps, FVNet employs multi-scale CNN architecture to capture objects of all sizes.

Object Detector	Input Modality	Easy	Moderate	Hard	Speed (Hz)
FVNet [49]	LiDAR (*FV*)	65.43	57.34	51.85	**83**
PointPillars [44]	LiDAR (*Pillars*)	**79.05**	**74.99**	**68.30**	62

TABLE 3.5: Comparison between [49] and [44] performance. Results for Car detection from 3D on KITTI [6] test benchmark ($AP_{0.7}$).

3.4 Camera-only Object Detection

Mono3D [50]. Mono3D is a two-stage monocular 3D object detector. By leveraging different image features such as: context, semantic and instance segmentation, shape and location priors, Mono3D generates 3D region proposals, which are then fed into a scoring CNN for filtering the top proposals and to provide accurate 3D bounding boxes. In the proposal generation stage, the 3D groundtruth boxes are projected into the 2D image to help generate all the required features and priors. Then, exhaustive search is used to generate 3D candidate boxes that lie on the same ground. Finally, regarding the proposal generation stage, NMS post-processing is applied to get top non-overlapping proposals. The 3D proposals generated from the first stage are further processed by a Fast R-CNN-like [30] architecture, which is responsible for estimating the class, bounding box information (position and size), and the orientation for the final detections.

Mono3D [50] has been experimented on KITTI object detection benchmark [6]. Although Mono3D outperformed other available monocular- and stereo-based proposed methods at the time of publishing the paper, it has two major drawbacks: (1) Mono3D is not an end-to-end learner, therefore its performance is dependant on the

quality of other extracted features and priors, (2) it is not suitable for real-time applications due to its two-stage nature, and mostly the computationally demanding regions' proposal stage.

Pseudo-LiDAR [51]. The main idea is to make use of advanced LiDAR-based 3D object detectors while requiring only a single monocular image as input. The latter is only feasible if there is a method to generate a *pseudo*-LiDAR point-cloud out of an image. Such method would have the advantage of efficient computation as it only requires a single image as input, while also having high 3D detection accuracy by operating on *pseudo* point-clouds. The authors of [51] propose an interesting framework that consists of four major components. (1) Depth estimation module that gets a monocular image as input, then outputs a dense depth map, which would then be transformed into a *pseudo*-LiDAR point-cloud using the camera calibration parameters. (2) Proposal generation module that operates on the input monocular image to generate segmented 2D instance proposals. (3) 3D Instance segmentation, which takes as inputs the *pseudo*-LiDAR point-cloud and the 2D instance masks, and is trained to segment only the corresponding points for each proposal (inspired by Frustum PointNets [52]). (4) Finally, the 3D points obtained from each segmented instance are used to train a model to estimate the target bounding box information: position, size, orientation.

Since the generation of *pseudo*-LiDAR point-cloud depends solely on the quality of depth maps generated by the depth estimation algorithm, inaccuracies of the depth estimation module are propagated to the 3D detection task. In addition, building *pseudo*-LiDAR from dense depth maps results in noisy point-clouds, which suffer two main issues, namely long-tail and local-misalignment. These issues are addressed by using a image-based instance segmentation step before cropping the 3D proposal, and employing a bounding box consistency loss which is then solved as an optimization problem. Inaccuracies from the 2D instance segmentation module are directly affecting the 3D detection accuracy as well.

Pseudo-LiDAR [51] has been experiment on KITTI [6] object detection benchmark, and has shown significant improvements over other monocular-based methods (at the time of publishing the paper). Although the framework is dependent on having robust depth estimation and instance segmentation models, it introduced a

clever way of detecting objects from monocular images, which are easily obtainable in most of the modern vehicles, and are significantly cheaper compared to LiDAR sensors.

3.5 Fusion-based Object Detection

3.5.1 What, How, and When to Fuse?

What to Fuse

Designing fusion-based 3D object detectors should include decisions regarding the types of sensors which will be fused together, and the exact representation of the sensory information as well. Is is also important to think in light of the effect of such decisions on both the detection accuracy and the performance in real-time.

LiDAR. LiDAR sensor 1.1.2 readings are obtained in the form of an unordered list of three-dimensional coordinates, and each has its own reflectance value. This format is called point-cloud, and it is unstructured and highly sparse. When visualized in the 3D space, point-clouds can represent the surrounding environment of the autonomous vehicle. As opposed to camera RGB images, LiDAR does not provide any information about texture, shape, or color. However, point clouds provide highly accurate depth information of the surrounding objects, which is critical for the task of three-dimensional object detection. Various point-cloud representations have been suggested to address the sparsity issue, and aiming to transform point-clouds' unstructured representation into usable inputs to CNN-based object detection architectures.

Voxel Representation of LiDAR Point Clouds. The idea of voxelizing a point cloud is to divide the three-dimensional space into equally-spaced partitions, namely voxels, where each voxel can contain 0 or more LiDAR points. The voxelization operation results in a volumetric shape, which could be further processed using 3D convolutions.

Projection-based Representations of LiDAR Point Clouds. Another way to deal with point clouds' sparsity, and at the same time reduces the required computation for processing the point cloud is to project the points into a 2D plane.

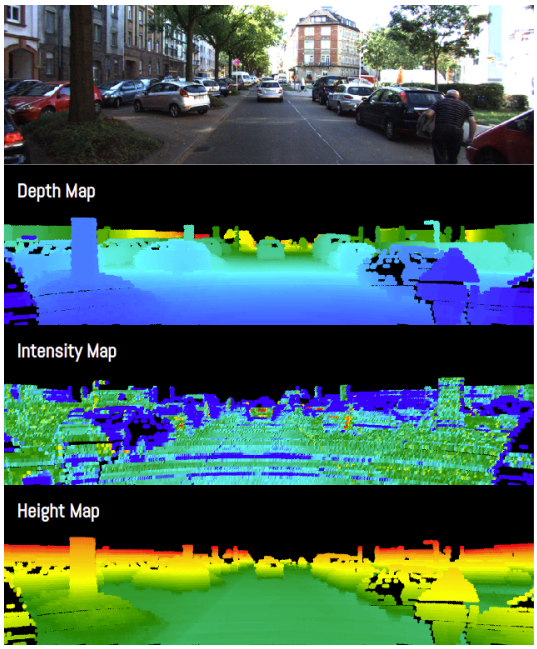
- **Bird's Eye View (BEV) representation.** Projecting LiDAR point cloud into the top view has a number of advantages. First, the 3D point-cloud is represented as a 2D image, where the width and height of the projected image both represent the spatial dimensions of the 3D space, and the channels' dimensions discretizes the height information from the point-cloud. The BEV representation of a point-cloud could be used as input to 2D CNN-based object detectors, and would significantly decrease the required computational cost for processing the point-cloud, compared to voxelization. Second, in the domain of autonomous vehicles, the objects in the surrounding environment are expected to lie on the same ground. Projecting the point-cloud into the top view would simplify the learning task of detection, as all the potential objects' bounding boxes are non-overlapping. A visualization of the BEV representation of point-clouds can be found in Figure 6.4b.

- **Frontal View (FV) representation.** Projecting LiDAR point cloud into the frontal view would result in an image, as if the LiDAR sensor is used as a camera. However, the frontal view projection of the point-cloud does not have information about texture or color, instead, it contains other features such as: *depth*, *intensity*, and *height*. The frontal view projection can give us the ability to accurately generate (1) sparse or dense depth maps given the depth information within the point cloud, (2) intensity maps based on the reflectance values of the 3D points, and (3) accurate height maps. All of these maps are extra features that could be used to enhance RGB images obtained from normal camera sensors. A visualization of the frontal-view representation of point-clouds can be found in Figure 6.4a.

Since LiDAR provides unstructured and sparse point-clouds. The representation of LiDAR point-clouds is important as it will have a direct impact on the inference speed of fusion-based methods. All of the discussed representation could either be used alone, or combined with other representations as well to give the model the ability to capture rich information during the fusion process.

Camera. Cameras are cheap and easy-to-have in any modern vehicle. There are multiple camera representations that are considered to be used in fusion-based

(A) Bird's Eye View (BEV) projection of a LiDAR point-cloud. Original point-cloud and RGB image are obtained from KITTI [6] dataset.



(B) Frontal-view projection of a point-cloud. Original point-cloud and RGB image are obtained from KITTI [6] dataset.

FIGURE 3.10: Different point-cloud projection-based representations.

method. Monocular colored RGB images are the most commonly used data representation, however, we can also use grayscale images. In the top row of Figure 3.11, we can see the same image in both RGB and grayscale represetantions. Clearly, RGB images would provide more meaningful information about color and texture in the context of feature extraction when compared to grayscale images. When the sensors' setup on an autonomous vehicle has two cameras, stereo images can also be used as input. The main advantage of stereo vision is the ability of estimating depth information, however, the drawback is that depth estimation algorithms are computationally expensive, which adds up more computational overhead.

As in [50], features that are extracted from RGB images can also be used as input for fusion modules, such as: instance semantic segmentation masks. Such decision might also lead to extra computational complexity, as in the use of stereo images, thus, it is a trade-off between four main pillars: the usefulness of the input modality, the easiness of obtaining such modality, the detection accuracy, and the real-time performance.

Various input modality combinations were proposed by different object detection architectures. PointFusion [53] uses monocular RGB images and raw LiDAR

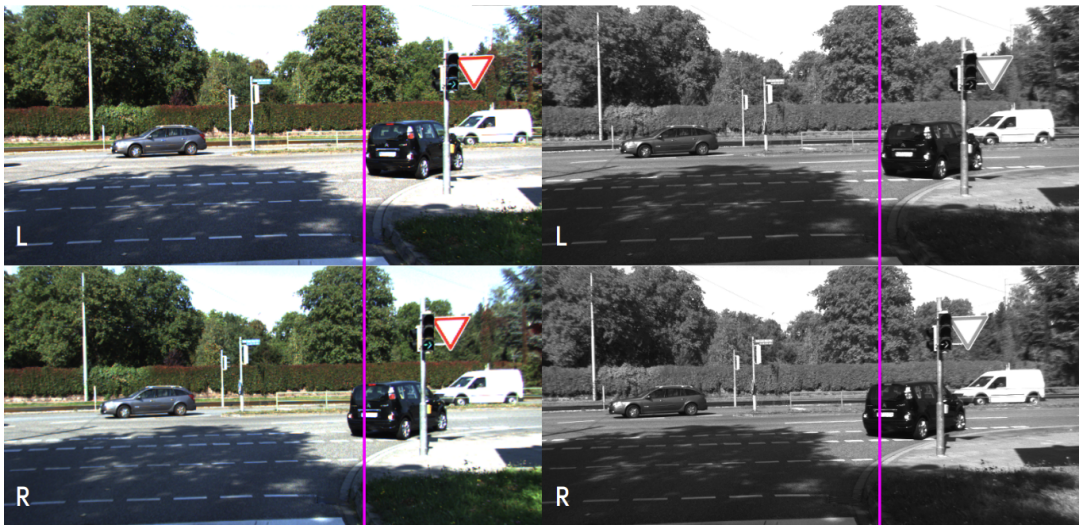


FIGURE 3.11: *Left column:* Colored RGB stereo images of the same scene. *Right column:* Grayscale stereo images of the same scene. $\mathcal{L}$ denotes the left image, and $\mathcal{R}$ denotes the right image. All images obtained from KITTI [6] dataset.

point-clouds as input. The image is fed into a CNN for feature extraction, while the raw point-cloud is processed through a PointNet [41] architecture, then the outputs of both streams is fused to generate the detections. AVOD [54] two-stage architecture uses monocular RGB images along with a BEV representation of the point-clouds. Frustum-PointNets [52] fuse RGB image along with depth maps obtained from RGB-D data.

ContFuse [55] perform the 3D object detection task by having both monocular RGB and BEV LiDAR representation as input, and by fusing both modalities on a feature level. MV3D [47] leverages three different inputs: BEV point-cloud representation, FV point-cloud representation, and monocular RGB image. Our proposed method uses the following inputs: monocular RGB images, BEV representation of point-clouds, frontal-view features extracted from point-cloud, such as intensity, depth, and height maps. The inputs are processed through two seprate streams, and feature-level fusion inspired by [55] is performed to generate 2D, 3D, and BEV bounding boxes.

In order to train robust fusion models, there are multiple important factors, including the precise calibration of different sensors, as well as accurately annotated 3D bounding boxes. These factors are well considered in modern publically available autonomous driving datasets, as in Lyft's [56], NuScenes' [38], as well as KITTI

[6]. While both Lyft and NuScenes data sets acquire 4 (or more) cameras, and high-quality LiDAR sweeps with a large number of accurate 3D annotations, both datasets have not been yet established as the benchmark for object detection. For that reason, in thesis, we are using KITTI dataset, as it provides the aforementioned factors, and it has been well established and used as a benchmark for the object detection task in a large number of publications.

How to Fuse

The operation used to fuse two or more modalities is critical as it is responsible for enabling interactions between the different available features. The fusion is either done by applying fixed mathematical operations or by employing learnable layers. Examples of fixed mathematical fusion operations are: element-wise summation 3.4, mean 3.5 or max 3.6. While such operations are computationally efficient, they are not robust enough to capture related features between different modalities. Another method relies on channel-wise concatenation of multiple modalities, followed by a learnable convolution operation.

$$\mathcal{F}_{sum} = \mathcal{F}_{inp1} \oplus \mathcal{F}_{inp2} \tag{3.4}$$

$$\mathcal{F}_{avg} = \frac{\mathcal{F}_{inp1} \oplus \mathcal{F}_{inp2}}{|\mathcal{F}_{inp1}|} \tag{3.5}$$

$$\mathcal{F}_{max} = \max(\mathcal{F}_{inp1}, \mathcal{F}_{inp2}) \tag{3.6}$$

More robust fusion operators were introduced in [57], which make use of the idea of bilinear-pooling. Bilinear pooling is another name for the full outer product of two vectors or matrices ($\mathcal{Z} \leftarrow \mathcal{X} \otimes Y$). The outer product of two input matrices means that all the element of each of the input matrices will be multiplied with each other, resulting in a new representation of the input matrices (modalities). Researchers aimed to use bilinear pooling as a form of fusion within deep learning architectures for its expressiveness, but the main issue is its impracticality from a computational perspective. The aforementioned challenge has pushed the research

community to develop techniques that would enable bilinear pooling in a computationally acceptable setting. The high-level approach was to factorize the bilinear pooling into multiple and consecutive computationally inexpensive operations.

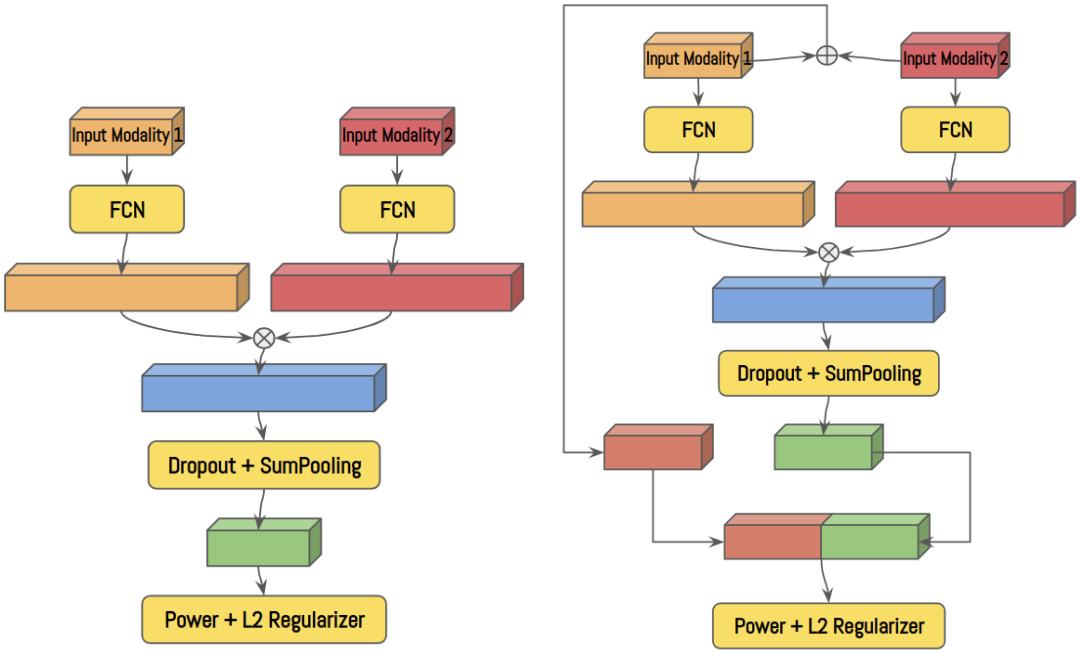
Multi-model Low-rank Bilinear Pooling (MLB) [58] first applies a non-linear projection for each modality into a common embedding space, then element-wise multiplication is applied for fusion, followed by linear projection. Fully-connected layers are being used as the projection operators. Multi-modal Factorized Bilinear Pooling (MFB) [59] starts by projecting the input modalities to a higher dimensional space, then uses element-wise multiplication for fusion, followed by dropout and sum pooling operations, finally, power ($z \leftarrow \text{sign}(z)\sqrt{|z|}$) and l_2 normalization are applied. The aforementioned factorized representation of bilinear pooling is computationally efficient and can be employed within sensor fusion architectures to perform mid-level (feature map level) fusion.

Further improvements are made to MFB [57] so that input modalities are fused through element-wise summation then concatenated with the output of the dropout and sum pooling operations. This modification enables high interaction between the input modalities and the fused outputs. Moreover, this acts as a residual connection that simplifies training deep architectures having this special type of bilinear fusion mechanism. The output is then fed through a convolution operation to reduce the dimensions of the concatenated feature to the desired output dimensions. Finally, and similar to original MFB, power and l_2 normalization are applied. Both MFB and its modified variant are demonstrated in Figure 3.12.

The use of power and l_2 normalization techniques is of a huge important, mainly, to deal with the magnitude variation in the output of the multiplication fusion mechanism. [59] showed that the impact of l_2 normalization is superior to the power normalization, however, the combination of both normalization techniques results in the best performing fusion module.

When to Fuse

After deciding which data representation to use, and how these modalities are going to be fused, we have to take a design decision of *when* to actually apply the fusion



(A) Basic version of the Multi-modal Factorized Bilinear Pooling [57].

(B) Modified variant of the Multi-modal Factorized Bilinear Pooling [57] (addition of a residual connection between input modalities and the fused vectors).

FIGURE 3.12: Demonstration of both original and modified variants of a multi-modal fusion technique, namely Multi-modal Factorized Bilinear Pooling [57]. Fully-connected Network is denoted as *FCN*.

operation between the different input modalities. There are three ways fusion can be applied to input modalities: *early*, *intermediate*, and *late* fusion.

Early Fusion. Early fusion is to fuse the input modalities before proceeding to the feature extraction step. This fusion technique is rarely used, specially when the input modalities are obtained from multiple views (e.g. top-view LiDAR and frontal view monocular image). Usually, fusing raw input modalities would simply produce a combined representation that might not contain more meaningful or useful information for the deep learning architecture. Instead, many fusion-based method like [47], [52]–[55] directly feed the multi-modal raw inputs to the feature extraction networks.

Intermediate Fusion. Intermediate fusion is when the fusion is performed on a feature map level. Intermediate fusion can be the most effective fusion strategy, because it enables the interaction between the feature maps of the multi-modal feature extraction streams. Both [47], [55] leverage intermediate fusion between LiDAR and camera streams to obtain high detection accuracy.

Late Fusion. Late fusion is where the modalities are fused after being separately processed through separate feature extraction streams, and right before the final detection stage. Compared to early fusion, late fusion benefits from the high-level features extracted from the multi-modal streams to perform the fusion, which contains more meaningful information compared to the raw inputs. However, late fusion does not use low- and mid-level features that might boost the final detection accuracy as well, as in the intermediate fusion strategy.

3.5.2 Camera-LiDAR Sensor Fusion Methods

MV3D [47]. MV3D stands for multi-view 3D object detection, and it introduces a robust sensor-fusion framework which operates based on camera and LiDAR sensory inputs. The reason behind fusing specifically these two sensors is to be able to overcome the shortcomings of each sensor alone; camera images contain rich information about color and texture, but lacking depth information, while LiDAR provides accurate depth information, but in form of sparse and unstructured point-clouds. MV3D consists of two main stages: 3D region-proposal generation, followed by a multi-view fusion network. The inputs to MV3D are three-fold: (1) BEV projection of the LiDAR point-cloud (similar to 3.3.3), in addition to intensity and density features of the point-cloud concatenated on the channels' dimension; (2) cylindrical frontal-view projection of the LiDAR point-cloud (similiar to 3.3.3); (3) camera RGB image. Each of the three input modalities is fed into its own CNN feature extractor followed by a RoI pooling layer [30].

Firstly, 3D region-proposals are generated using the point-cloud's BEV representation. The 3D proposals are then projected into multiple views (BEV, frontal-view, camera-view) and added to the output of the RoI pooling layer. In the second stage, the multi-view region proposals are averaged together, then fed into a *deep fusion* network. While early fusion is to fuse input modalities before starting to extract features, late fusion is to fuse the high-level feature maps of each feature extractor (e.g. concatenation). Deep fusion is applied by sequentially averaging N feature maps using the element-wise mean operation, then branching out to N paths with non-shared convolution operations. Finally, the fused multi-view feature maps are fed into two header branches for classification and regression. MV3D [47] has been

experimented on the KITTI [6] object detection benchmark, and has outperformed other 3D proposal-based methods operating on either camera or stereo [50], [60], as well as LiDAR-based method [61]. Figure 3.13 contains a high-level overview of MV3D's end-to-end sensor fusion framework.

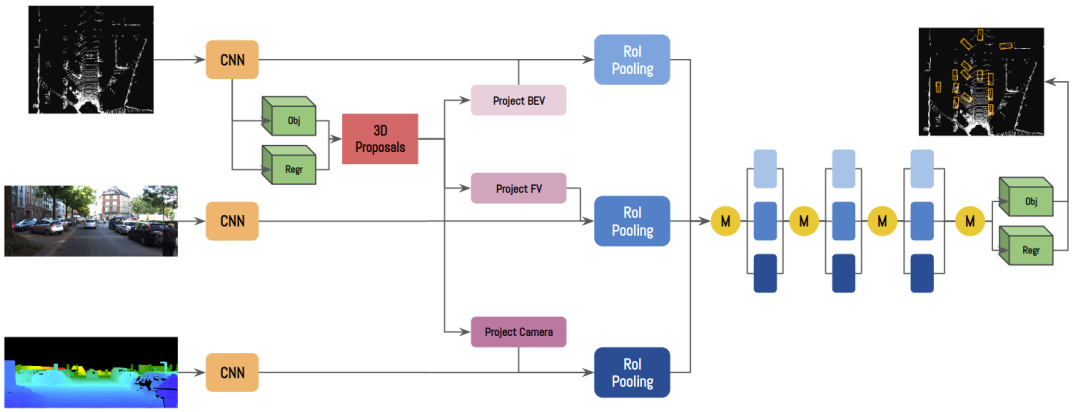


FIGURE 3.13: MV3D [47] sensor fusion framework architecture.

Frustum-PointNets [52]. Frustum-PointNets is a fusion architecture that uses RGB-D data as input to accurately perform 3D object detection and localization. In order to reduce the computational complexity resulting from processing 3D points clouds, and to reduce this large search space for candidate objects, Frustum-PointNets leverage mature 2D object detection networks to generate region proposals, then it uses depth information obtained from RGB-D data to detect amodal oriented 3D bounding boxes.

Firstly, monocular RGB image is fed into a robust FPN-based [36] 2D object detection network, that has been pretrained on COCO [37] data set, then fine-tuned on KITTI [6] 2D object detection task. The output of the first step are 2D region proposals. Using the camera projection matrix, and based on the 2D region proposals, frustums are extracted from the depth data are extended to the 3D space, forming a *3D frustum point cloud*. Each frustum point cloud contains only one object, and this object's class is known from the 2D region proposal step, and is fed as extra information to following the segmentation network. After that, a 3D instance segmentation network is employed to segment only the point of the target object from the frustum point cloud. This instance segmentation task is performed using a PointNet [41] architecture.

After segmenting the target object from the frustum point cloud, a masking operation is performed on that object. It is worth noting that normalizing such frustum point clouds is crucial in this algorithm. Without normalization, these frustum point cloud generation step would result in a large number of variations in the position and orientation of such point cloud. The first normalization step is performed after extracting the frustum point cloud. In addition, after performing the masking operation, the local coordinate system is positioned in the center of the masked object. Finally, an amodal 3D bounding box estimation module is employed to accurately estimate the size, position, and orientation of the bounding boxes using a regression PointNet [41] architecture. Frustum-PointNet [52] method is trained by optimizing multi-task losses: instance segmentation, bounding box information, as well as a novel *corner loss*, that aims to optimize the network to generate best 3D detection accuracy by minimizing the distance between the corners of predicted boxes compared to the groundtruth boxes.

PointFusion [53]. PointFusion proposes a simple method for camera-LiDAR sensor fusion. Instead of performing late fusion after processing both modalities, this method performs intermediate fusion right after extracting meaningful features from the input modalities. The method consists of two consecutive stages. The first stage is to employ a state-of-the-art 2D object detection network such as Faster R-CNN [31] to produce cropped images of the objects in the scene. The second stage is to perform the sensor fusion using the cropped image, as well as the corresponding 3D points extracted from the raw LiDAR point cloud. In the presence of other alternative representations of LiDAR point clouds, this method is using raw points to mitigate any loss of information or computational complexity from projection or voxelization operations. Other methods like [47] use the top-view projection of LiDAR point clouds. However, the top-view could yield good results for large objects like cars and trucks, while being less performant on smaller objects like pedestrians and cyclists.

After generating cropped RoI from an image using a Faster R-CNN [31] model pretrained on COCO [37] and fine-tuned on KITTI [6], the corresponding 3D points from the raw point cloud are obtained by querying only the 3D points that are visible

inside the 2D bounding box. First, the cropped image is fed into a ResNet [12] feature extraction network pretrained on ImageNet [19]. In parallel, the cropped raw point cloud is fed into a modified version of PointNet [41]. Employing the original architecture of PointNet [41] did not yield accurate 3D bounding box estimation. The reason, as stated by the authors of the paper, was the use of Batch Normalization [16] layers which affected negatively the 3D regression task. Based on this observation, they have removed all Batch Normalization layers from their modified PointNet network.

PointFusion proposes two intermediate fusion strategies. The first one is a vanilla global fusion network. It works simply by concatenating the output features of both the ResNet and the global features of the modified PointNet, producing a feature vector of size (1×3072), then applying a number of consecutive fully connected layers, followed by a direct 3D regression task of the 8 corners of the target bounding box. The loss function use for regression is the smooth L1 loss 2.17. The other strategy is called dense fusion network, and it uses the same concatenated features, in addition of the point-wise features from the modified PointNet. Then, for each point, offsets to the target bounding box corners are estimated after a series of fully connected layers as well.

AVOD [54]. AVOD tackles the challenge of extending 2D region proposal networks (RPN) to the 3D domain. Robust 2D-based RPNs are intended to deal with dense RGB images, thus, using RPNs for the task of 3D object detection is not a trivial task. AVOD proposes an architecture that performs 3D object detection by employing two consecutive sub-networks, and by leveraging both LiDAR and monocular images in that process. The first sub-network is responsible for region proposals from both LiDAR point clouds and RGB images. Then, a detection network is used, along with fusion, to produce final 3D oriented bounding boxes. AVOD's method has high detection accuracy for both large and small objects (e.g. cars and pedestrians).

In the first stage of this architecture, two identical feature extractor networks are used to process both BEV projection of the point-cloud, as well as the monocular RGB image. In order to generate high resolution feature maps, an encoder-decoder architecture is used, which is inspired by FPN [36]. Using the output feature maps,

and with the help of pre-defined anchor boxes, a crop-and-resize step is performed to generate region proposals. Then, the cropped regions are fused using an element-wise mean operation, and fed into two streams of fully connected layers (one stream for each modality). Each stream is trained to produce objectness scores, as well as regression values for the bounding box information. The detections obtained from the first stage are sorted, and the top K are selected to be used in the second stage of the architecture.

In the second stage of AVOD, the top K selected boxes are used to generated crops for the high resolution feature maps, which will result in final 3D proposals. 300 proposals are used for the *car* class, and 1024 are used for the classes *pedestrian* and *cyclist*. The resized crops proposals are then fused using element-wise mean, then passed through a series of fully connected layers, to generate final bounding box estimations for each proposal. Regarding the bounding box encoding, instead of using 8 corners for regression, AVOD used another encoding technique, which represents the bounding box with 4 (lower) corners, along with 2 height values: the height from the ground to both the higher and lower surfaces of the bounding box. Similar to [46], AVOD used the angle factorization into two terms, $(\cos(\theta), \sin(\theta))$, which yielded better accuracy in the orientation estimation learning task.

ContFuse [55]. ContFuse is an end-to-end 3D object detection model that fuses multimodal sensory information coming from both camera and LiDAR. More specifically, it focuses on feature-level fusion of both modalities, instead of adopting either early or late fusion techniques. The end-to-end framework consists of two main streams, one for each sensor modality. The LiDAR stream operates on BEV representation of point-clouds, similar to 3.3.3. The fusion happens between multi-scale feature maps from both streams via simple element-wise summation. The main contribution for this paper, and which solves a challenging task, is the way front-view feature maps (extracted from camera image) are transformed to a BEV representation, for which they use a new layer named *continuous fusion* layer, inspired by the work of [62]. In simple terms, for each feature vector in the spatial dimensions of an image's feature map $\mathcal{F}_{img}$, a multi-layer perceptron is employed to estimate the corresponding feature vector $\mathcal{F}_{img \to bev}$ when $\mathcal{F}_{img}$ is projected into the BEV representation.

Object Detector	**Input Modality**	**Easy**	**Moderate**	**Hard**	**Speed (Hz)**
MV3D [47]	Camera + LiDAR	86.02	76.90	68.49	2.8
ContFuse [55]	Camera + LiDAR	88.81	**85.83**	77.33	16.7
PIXOR++ [48]	LiDAR (BEV)	**89.38**	83.70	**77.97**	**35**
AVOD [54]	Camera + LiDAR	88.53	83.79	77.90	10
Frustum-PointNets [52]	RGB-D	88.70	84.00	75.33	6

TABLE 3.6: Comparison between [47], [52], [54], [55] and [48] performance. Results for Car detection from BEV on KITTI [6] test benchmark ($AP_{0.7}$).

ContFuse [55] feeds the camera input to a ResNet-like [12] feature extraction stream to obtain multi-scale feature maps. BEV representation of the LiDAR point-cloud is fed into a parallel stream of residual blocks as well. Constructing multi-scale feature maps in the camera stream is done based on FPN [36]. The multi-scale feature map is then transformed into BEV using the continuous fusion layer, followed by an element-wise summation with the corresponding (in size) feature map from the BEV stream. Finally, after fusing camera stream feature maps, FPN [36] multi-scale feature map grouping is applied to the BEV stream as well. The final feature map is fed into two headers to estimate objectness map and regressed bounding box information. ContFuse method has been evaluated on KITTI [6] object detection benchmark, and has shown competitive results compared to other fusion methods, while having real-time performance. Since our proposed method is an extension to ContFuse, a more detailed explanation will be provided in the next chapter. Table 3.6 has comparisons between fusion-based and projection base BEV detection accuracy.

Chapter 4

Methodology

In this chapter, we delve into the details of our approach to LiDAR-camera sensor fusion, while breaking our method down to three main sections: Dataset and Data Representation 4.1, Training details 4.2, and the Inference stage 4.3.

4.1 Input & Output Representation

In this section, we provide an in-depth details of how the inputs/outputs of our 3D object detection deep learning-based framework are encoded/decoded.

4.1.1 Input Representations

Camera Sensor

Most of the state-of-the-art object detectors were initially designed to deal with dense input images. In this thesis, we aim to leverage the richness of color and texture of monocular RGB camera images. In addition, RGB images are easily obtainable in almost any modern vehicle, due to its easy setup, and the cheap price of the camera sensor. The size of the input RGB images is $L \times W \times 3$, where L and W represent the spatial dimensions of the image, and the third dimension contains three color-specific channels: red, green, and blue. The size of the images obtained from KITTI dataset is: $1242 \times 375 \times 3$. We apply min-max normalization to the input RGB image as a preprocessing step, so that all pixel values are in the range 0–1.

Despite the previously mentioned advantages of RGB camera images, this sensor modality suffers for a number of issues: (1) the lack of accurate depth information, (2) easily impacted by adverse light and weather conditions. These issues drive the

main reason behind developing robust sensor fusion frameworks to overcome the shortcomings of individual sensors.

LiDAR Sensor

As discussed in 3.5.1, LiDAR point-clouds are unstructured and sparse, and there are various ways to deal with both of these issue, including: voxelization and projection. In our proposed approach, we use multiple representations of the LiDAR point cloud: top-view (BEV) and frontal view.

Bird's Eye View Representation. We employ a BEV projection technique of the point cloud, where we are able to encode the $x - y$ axes of the 3D space into the spatial dimensions of an image, while having the vertical axis discretized into the channels' dimension. The image nature of point cloud's BEV representation enables the use of the advanced and efficient 2D CNN-based object detectors. As in both PIXOR [46] and PIXOR++ [48], point-cloud's BEV projection is used to simplify the detection task as well as any post-processing steps, since all the target objects are expected to have non-overlapping areas, while also having similar size ratios from the top-view perspective.

In order to construct a BEV projection of a raw 3D point-cloud, we first define the dimensions of the 3D physical space as $L \times W \times H$, denoting the length, width, and height of the 3D physical space, respectively. Then, we define the discretization factors based-on which we will construct our BEV representation. In the process of discretizing the 3D space, we aim to divide each dimension of the 3D space into equally space cuboids, which will then be represented by image pixels. The discretization factors are defined as $d_L \times d_W \times d_H$, indicating the factors used to encode the length, width, and height of the 3D physical space, respectively. The final output resolution of the BEV representation of a point cloud is defined as $\frac{L}{d_L} \times \frac{W}{d_W} \times \frac{H}{d_H}$. The value of each position $\{i, j, k\}$ in the final BEV representation $O_{\frac{L}{d_L} \times \frac{W}{d_W} \times \frac{H}{d_H}}$ is a boolean, which signifies the presence or absence of 3D LiDAR points in the corresponding cuboid in the 3D physical space.

The size of the BEV representation is a crucial factor for the inference time, thus, the goal is to decrease its spatial dimensions, while maintaining a high quality representation of the 3D space. In our method, we define the dimensions of the 3D

physical space $L \times W \times H$, as $[0,70] \times [-40,40] \times [-1,3]$, respectively. The measurements of the physical space are in meters. Further, we drop all 3D points that are outside the camera's field-of-view (FoV) from the raw LiDAR point cloud before applying the projection step. We define the discretization factors d_L, d_W, and d_H as 0.15625, 0.15625, and 0.125, respectively, which leads to producing a BEV image with spatial dimensions of 448×512, along with 32 channels encoding height information. Since this encoding is a representation of the occupancy of the point cloud from the top-view, it is normalized by default in the range 0–1. The BEV projection process is demonstrated in Figure 4.1.

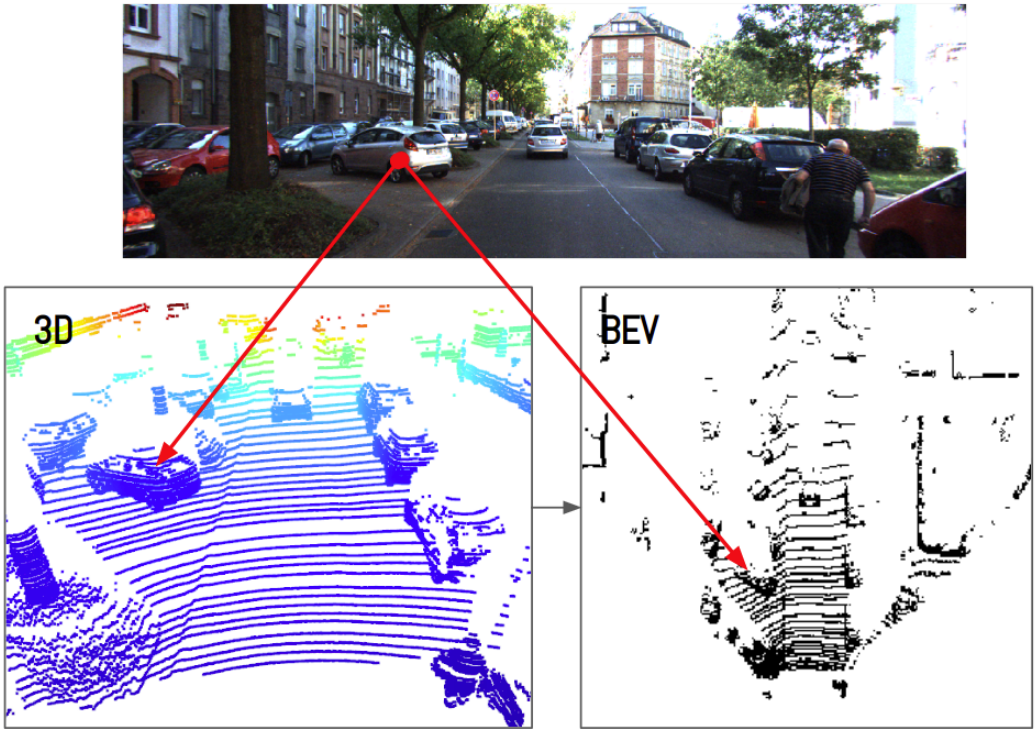


FIGURE 4.1: LiDAR point clouds' 3D-to-BEV projection with FoV filtering.

Frontal View Representation. We enrich the input of our sensor fusion framework with another projected view from the LiDAR point cloud, namely the *frontal view* projection. The frontal view is the same view of the point cloud as perceived from the camera perspective. We leverage the available and accurate camera calibration parameters provided in the KITTI [6] dataset to construct such view, so that we can early fuse it with RGB camera images.

We can obtain three different frontal view feature from the sparse LiDAR point

clouds: *intensity*, *depth*, and *height* maps. The intensity maps are constructed using the reflectance values provided within the raw point cloud data. Both depth and height maps are created by calculating a ratio between the depth or height value of a 3D point and the maximum value in that corresponding dimension. The spatial dimensions of all three feature maps match the spatial dimensions of the RGB camera image, which is 1242×375. All three feature are then concatenated in the channels' dimension, to form a frontal view feature image with size $1242 \times 375 \times 3$. The frontal view encoding is normalized in the range 0–1. The frontal view projection process is demonstrated in Figure 4.2.

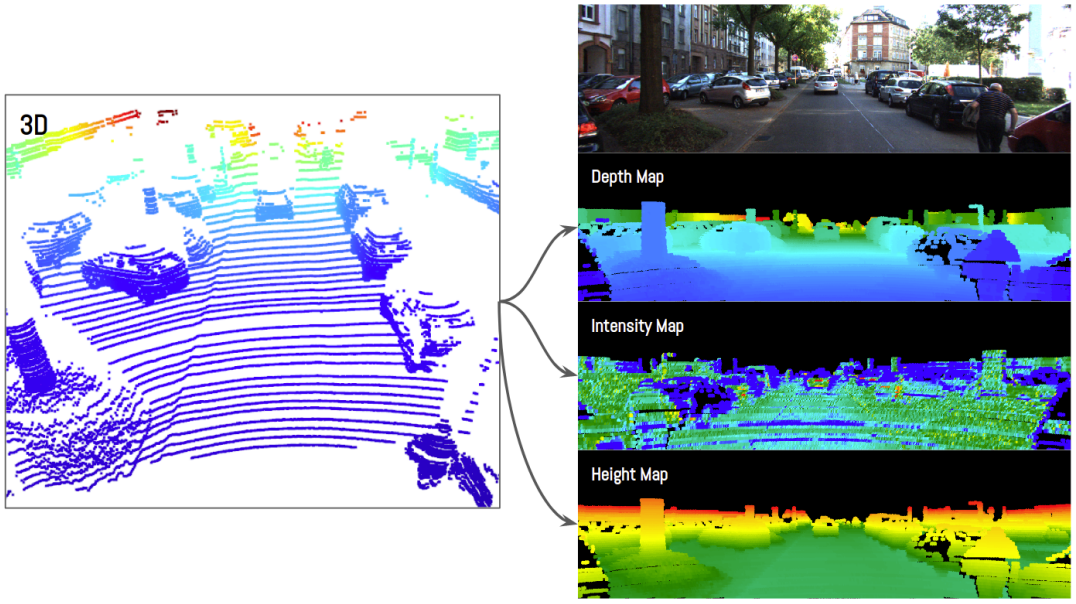


FIGURE 4.2: LiDAR point clouds' 3D-to-FV projection, and extracting features as: depth, height, and intensity.

4.1.2 Output Encodings

In this subsection, we describe our method's target outputs, based on which we train our supervised deep learning-based sensor fusion framework. We generally rely on BEV to decode the final bounding box detections, which simplifies the learning task, as mentioned in the previous subsection.

Bird's Eye View Objectness Maps

For a given input LiDAR sweep, we build the target output by first projecting the corresponding frame's groudtruth 3D bounding boxes to the BEV perspective. Then, we construct a BEV boolean heatmap, where the pixel value of 1 denotes the presence of an object (foreground), and the pixel value of 0 denotes the absence of objects (background). Furthermore, and inspired by [46], we ignore the boundary pixels of the groundtruth bounding box, and treat these pixels as background. We do that by generating a zoomed-out version of the groundtruth box, by a factor of $0.6\times$, and we call this process *subsampling*. The constructed heatmap - i.e. objectness map, will serve as the groundtruth for our supervised deep learning model.

Due to the compact nature of the input BEV representation, we use an overall downsampling factor of 2 to construct the target objectness maps, so that we can avoid the problem of objects having a very small area in the output heatmap, which can make them hard to learn and detect. The size of the objectness map is $224 \times 256 \times 1$. Figure 4.3 demonstrates a visualization of the objectness maps, with and without the removal of boundary pixels.

Frontal View Objectness Maps

To construct an objectness map for the frontal view stream, we apply similar steps to the ones mentioned in the encoding process of BEV objectness maps, the only difference is that we rely on the groundtruth 2D bounding boxes. To stabilize the training process, we also apply subsampling, by denoting an object's presence with a circular shape contained within the initial rectangular 2D bounding box. Frontal view objectness maps are constructed as boolean heatmaps, where the value is 1 in pixels where there exist an object, and 0 otherwise. We use a downsampling of 4 for the frontal view branch, thus, we obtain a target range view objectness map of size $310 \times 94 \times 1$. Figure 4.4 shows the target frontal view objectness map used to train the frontal view network stream.

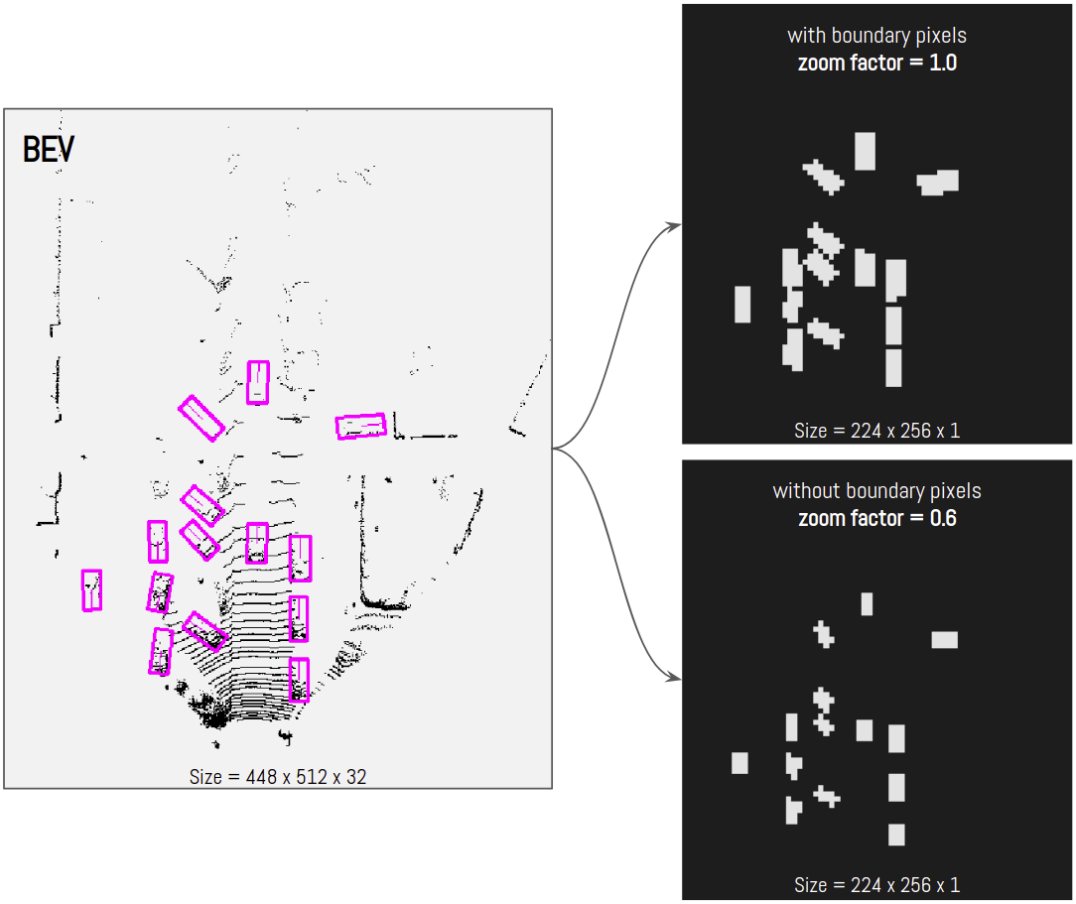


FIGURE 4.3: BEV objectness maps that are used as target output for the deep learning model.

Bird's Eye View Regression Maps

The target regression map's spatial dimensions matches the BEV objectness map's, which is 224×256. We parameterize every object as an oriented bounding box, represented by: object's center coordinates $\{x_c, y_c, z_c\}$, object's size $\{w, l, h\}$, and object's heading angle $\{\theta\}$. Instead of directly regressing real-valued center position of the target objects, the goal is to learn offset values between the predicted and groundtruth bounding boxes' centers. Thus, the box's spatial position in BEV is encoded as (d_x, d_y). The box's height is encoded as the distance between the ground (given the sensor's position from the ground) and the box's height value - i.e. the altitude. The object's size is normalized by applying the logarithmic operation, resulting in $(\log(w), \log(l), \log(h))$. To encode the heading angle, we follow both [46], [54], by factorizing the box's orientation into $(\cos(\theta), \sin(\theta))$.

The target regression map contains eight channels, one for each regression value:

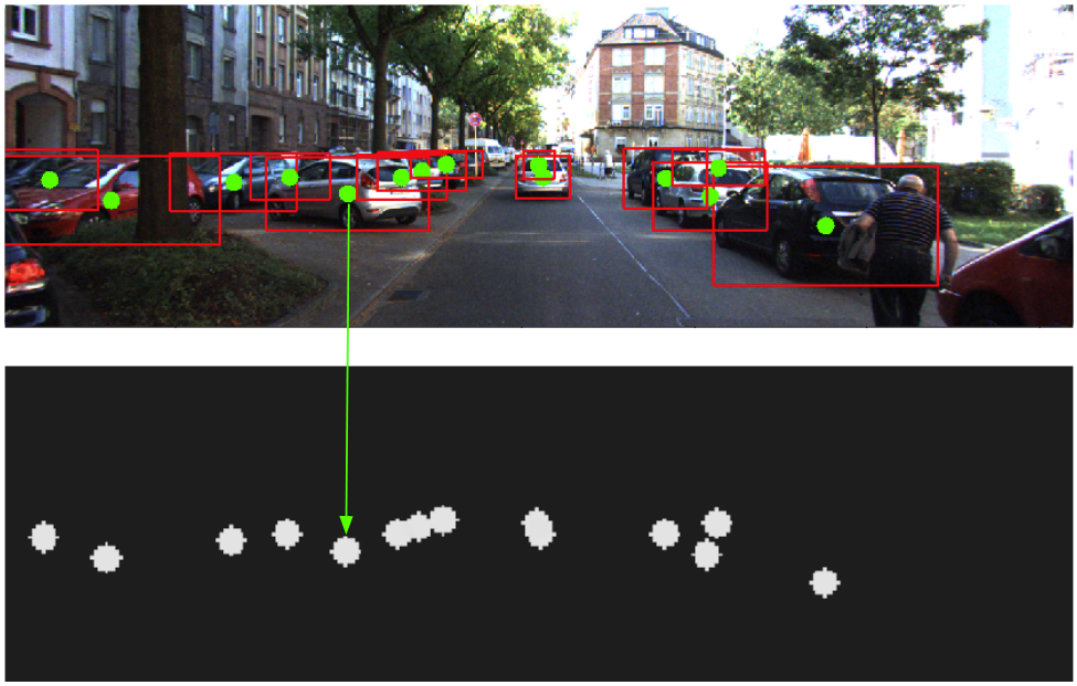


FIGURE 4.4: Frontal view objectness map that is used as target output for the frontal view stream.

$\{ d_x, d_y, alt, \log(w), \log(l), \log(h), \cos(\theta), \sin(\theta) \}$. Each of the regression values is standardized based on the mean and standard deviations obtained from the whole training set. The target BEV regression map's size is: $224 \times 256 \times 8$.

4.2 Training

Our end-to-end sensor fusion network is trained using the RMSProp optimization algorithm, which aims to minimize the defined objective functions discussed in 4.2.1. The learning rate schedule is an exponential decay with a warm-up period. We have implemented distributed multi-GPU training to boost the model training time. In order to avoid overfitting on the small-sized KITTI dataset, we have implemented multiple data augmentation techniques that boost the performance of the sensor fusion framework.

4.2.1 Objective Functions

In order to train our end-to-end multi-modal fusion framework, we adopt a multi-task loss, consisting of three components: frontal view classification loss, BEV classification loss, and BEV regression loss, as in Equation 4.1.

$$\mathcal{L}_{multitask} = \mathcal{L}_{FV(cls)} + \mathcal{L}_{BEV(cls)} + \mathcal{L}_{BEV(reg)} \tag{4.1}$$

Classification Task

In order to deal with the severe class imbalance between background and foreground in both tasks, frontal view and BEV classification, we employ a focal-loss [13], previously explained in Equation 2.14. Furthermore, we take into consideration the subsampling mask to ignore object's boundary pixels, as mentioned in 4.1.2 and 4.1.2. Finally, we normalize the resulting loss by the number of pixels with the value 1 in the groundtruth objectness map. The final focal loss definition is in 4.2.

$$\begin{aligned} p &= \begin{cases} \hat{y}, & \text{if } y = 1 \\ 1 - \hat{y}, & \text{otherwise} \end{cases} \\ \mathbf{FL}(p) &= -\tfrac{1}{N}(\alpha(1-p)^{\gamma}\log(p)) \\ &\text{where } \alpha \text{ is a weight factor, } \gamma \text{ is a modulating factor} \\ &N \text{ is the number of positive pixels in the GT objectness map} \end{aligned} \tag{4.2}$$

Regression Task

To accurately estimate the bonding box information, we adopt a smooth-L1 loss, as defined in Equation 2.17. The regression loss is computed only in the position of positive pixels of the corresponding groundtruth objecntess map.

4.2.2 Data Augmentation

Due to the small size of the KITTI dataset [6], deep learning architecture tend to overfit, and the goal of generalization becomes harder, which leads to low detection accuracy on the test set. In order to give our end-to-end framework the ability

to generalize, we apply data augmentation on KITTI's frames, which significantly improves the performance of our model when evaluated on our validation set.

Since our framework deals with multi-modal sensory information as input, data augmentation needs to be applied simultaneously to both modalities without introducing any form of inconsistency. We integrated six different augmentation strategies in our training pipeline. All six strategies were applied on both LiDAR point clouds, as well as monocular RGB images, as in Figure 4.5. The types of augmentation are described as follows:

- **Horizontal Flipping.** We perform a flipping operation on the horizontal axis of the input image, as well as the point cloud.
- **Dropout.** We perform this augmentation technique in point clouds, by dropping 3D points based on a probability. There are two types of dropout:
  - **Per-box Dropout.** Per-box dropout is done by dropping 3D points from within a groundtruth bounding box.
  - **Global Dropout.** Global dropout is done by dropping 3D points from the input raw point cloud, regardless the point is within a bounding box or not.
- **Scaling.** We apply scaling on both the raw point cloud as well as the RGB image. The scale factor is in the range $\{0.9, 1.0\}$, which enables us to apply zoom-in and -out operations to the frame, resulting in new augmented version of the input modalities.
- **Translate.** We start by applying a translation in the 3D physical space. We translate the vertical axis with value in the range of $\{-3, 3\}$ meters, and the horizontal axis in the range of $\{-5, 5\}$ meters. Then, we use the random translation values and project them to the camera view to be able to apply the same translation on the image. Translated pixels are then padded with 'same' padding strategy.

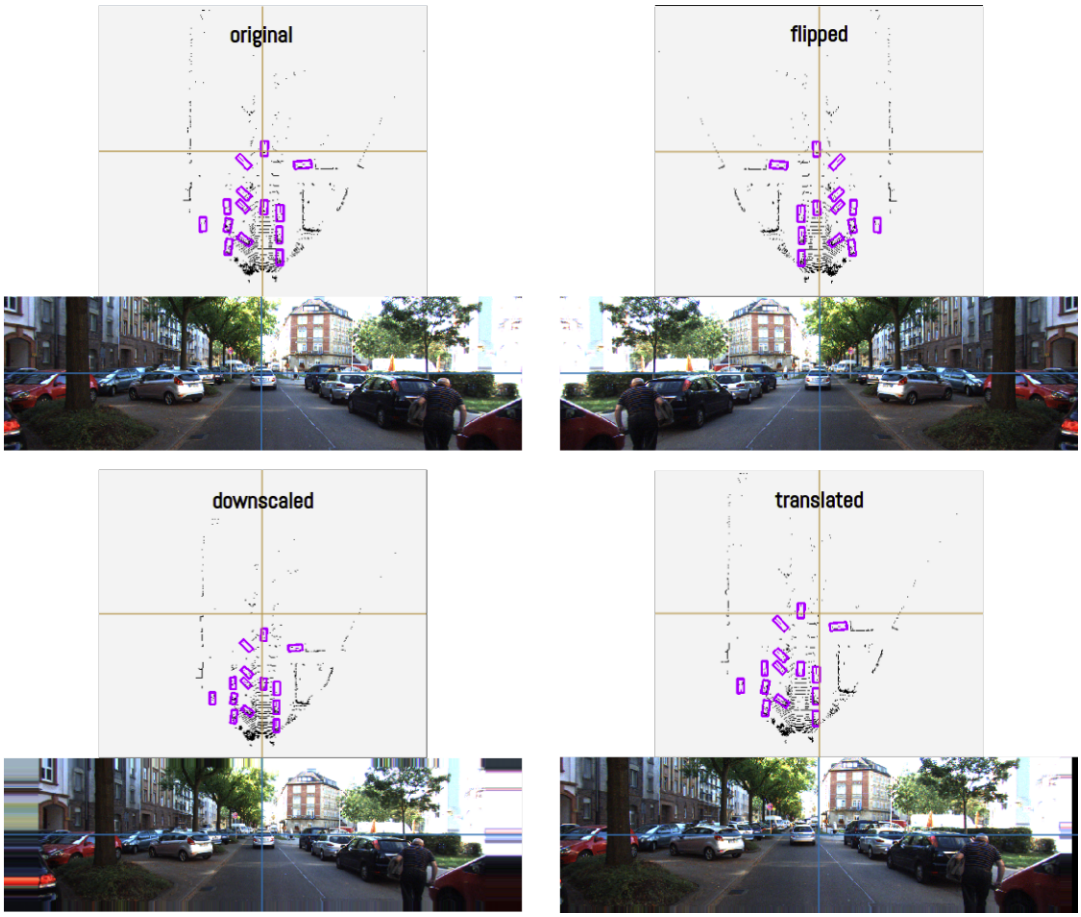


FIGURE 4.5: Demonstration of 3 multi-modal data augmentation techniques.

4.3 Inference

Preprocessing. Our end-to-end sensor fusion framework is frame-based. First, we obtain a frame from KITTI dataset, which contains: a LiDAR sweep and a monocular camera image, along with the calibration parameters. We start by cropping the LiDAR sweep according to the camera's FoV, then we project the remaining 3D point into two views: the BEV and the frontal view representations. We then normalize all the input data to be in the range 0–1.

Forward Pass. We feed each input modality to the corresponding CNN stream to produce the BEV objectness and regression maps.

Decoding. BEV objectness and regression maps are used together in the bounding box decoding step. The decoded information are denormalized for each box. Then, the decoded bounding boxes are filtered using a confidence thershold.

Post-processing. Finally, non-maximum suppression (NMS) is used to remove

duplicate bounding box detections by applying a thersholded Intersection-over-Union (IoU) operation. After NMS, we get a list of final bounding box detections. Thanks to our bounding box parametrization, the end-to-end framework is able to produce accurate bounding boxes that can be used in either 2D, 3D, or BEV object detection tasks.

Chapter 5

Fusion Network Architecture

Within our proposed end-to-end fusion framework, we propose a two-stream network architecture: one stream for each input view. The streams act as feature extractors for the input modalities – i.e. backbone networks. We employ an intermediate fusion strategy between both stream. Finally, detected bounding boxes are obtained through detection heads.

5.1 Backbone Networks

The backbone network architecture must be robust in terms of feature extraction capability, and at the same time, it must be efficient from a computational complexity standpoint.

5.1.1 Frontal View Stream

The frontal view stream is a feature extractor CNN, that is used to process the input modalities obtained from the frontal view (the monocular RGB image in addition to the features extracted from the point cloud frontal view projection). The frontal view backbone network expects the two inputs to be of the same size: $1242 \times 375 \times 3$.

Early Fusion. First, we fuse the two input modalities using a learnable fusion operation, called Multi-modal Factorized Bilinear Pooling [57], and is explained in 3.5.1. By employing this early fusion technique, we aim to transform the multi-modal inputs into a more robust representation of the frontal view, that would then be used for further processing and feature extraction.

Factorized Bilinear Pooling. This learnable fusion technique was introduced in [57] as a fusion technique for temporal activity detection in videos. We adapt this fusion mechanism to the object detection task. This learnable fusion mechanism enables more interactions between the input modalities, which would result in a high quality fused output compared to fixed arithmetic operations (e.g. element-wise addition). First, both input modalities are projected into a higher dimensional space using fully-connected layer, which were replaced by a 1x1-Conv in our fully-convolutional architecture. Then, both projected feature maps are fused using element-wise multiplication, followed by dropout and sum pooling operations. The output feature map is then concatenated with the resulting feature map of an element-wise addition operation between both original input modalities. Finally, power and l_2 normalization are applied to produce the final fused feature map. A more detailed discussion on the multi-modal factorized bilinear pooling design can be found in Subsection 3.5.1.

CNN Design. We propose a three-block ResNet-based [12] feature extractor network, with a total downsampling factor of 8. The convolutional blocks contain $\{2, 4, 4\}$ layers each, respectively. The residual block's design follows the *pre-activated residual blocks* [24] in Figure 2.12b. We define a fixed number of filters per block, and this number is doubled right after a downsampling operation is applied. The number of filters in the three residual blocks is $\{32, 64, 128\}$, respectively. We perform downsampling by applying strided convolutions in the first convolutional layer of a residual block. ReLU [11] activations are used in all convolutional layers, along with 'same' padding strategy.

Feature Maps. In order to combine multi-scale feature maps from all three residual blocks, we employ an FPN-like [36] architecture. We resize the feature maps of both residual Blocks 1 and 3 to the same size of the output feature map of Block 2, which yields three feature maps with the same spatial dimension: 310×93. We apply a 1×1 convolution with 64 channels after each bilinear resize operation to mitigate any resulting checkerboard effect. Then, we combine all three feature maps by a *concatenation* operation in the channels dimension, resulting in a feature map of shape: $310 \times 93 \times 192$. Which means that the final downsampling factor of the frontal view stream is 4.

Output Map. The combined feature maps using FPN are then fed into a single channel convolution operation to perform the supervised objectness task from the frontal view, as discussed in 4.1.2.

An illustrated overview of the frontal view backbone network architecture can be found in Figure 5.1. The frontal view stream is pre-trained on the objectness task separately. Then, it is incorporated within the end-to-end detection framework for the 3D object detection task.

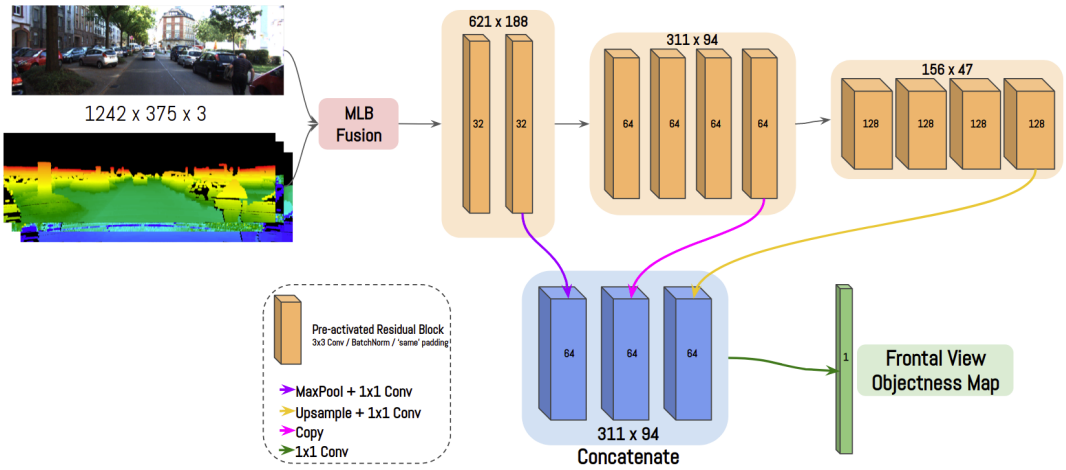


FIGURE 5.1: Architectural overview of the frontal view stream.

5.1.2 Bird's Eye View Stream

The BEV stream is a feature extractor CNN, used to process the BEV projection of the raw LiADR point cloud. We employ a CNN-based backbone network that expects an input of size: $448 \times 512 \times 32$.

CNN Design. Similar to 5.1.1, we employ a three-block ResNet-based [12] network, each block contains $\{4, 6, 6\}$ pre-activated residual convolutional layers [24], respectively. The number of filters per block is $\{32, 64, 128\}$, respectively. ReLU activations are used for non-linearity. Downsampling is performed using strided convolutions. The overall downsampling factor of the BEV stream is 8.

Feature Maps. We employ an FPN-like [36] architecture to combine multi-scale BEV feature maps. The process is the same as discussed in 5.1.1. The size of the resulting feature map from the BEV stream is: $224 \times 256 \times 192$. The multi-scale feature map combination step results in an overall downsampling factor of 2.

An illustrated overview of the BEV stream architecture can be found in Figure 5.2.

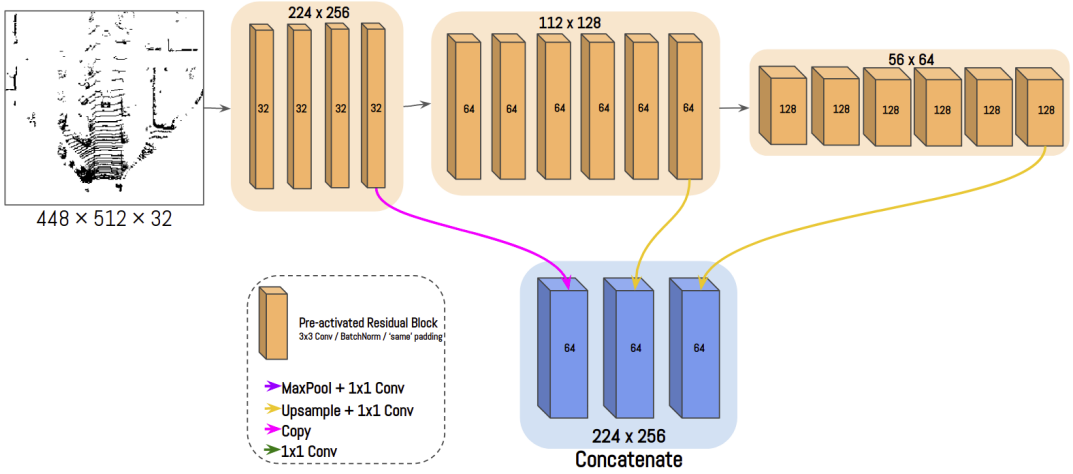


FIGURE 5.2: Architectural overview of the BEV stream.

5.1.3 Fusion Module

In our proposed sensor fusion framework, we aim to fuse camera and LiDAR sensors by extracting features from the RGB frontal view images, then, transforming these features to the BEV so that they can be easily fused with features maps extracted directly from the BEV stream. Transforming feature maps from one view to another is not trivial, and this is where we use Continuous Convolutions, initially proposed in [55].

Continuous Convolution. We employ the continuous convolution process to transform FV feature maps to BEV. The feature space transformation steps are described as follows:

1. **K-Nearest Neighbours.** We start by observing the target BEV feature map $\mathcal{F}^{bev}$, which we will fuse with a transformed version of FV feature map ($\mathcal{F}^{fv}$). First, we iterate through $\mathcal{F}^{bev}$'s spatial dimensions; we unproject each $\mathcal{F}^{bev}_{i,j}$ pixel to the 3D space using the camera-LiDAR calibration parameters. Then, we search for the unprojected point's nearest neighbour ($K = 1$) in the raw 3D point cloud. This nearest neighbour is described as a point in the Euclidean space $\{x_{(i,j)}, y_{(i,j)}, z_{(i,j)}\}$.

2. **Feature Matching.** The obtained nearest 3D point $\{x_{(i,j)}, y_{(i,j)}, z_{(i,j)}\}$ is then projected back to the image plane, which results in a 2D point: $\{l, m\}$. This 2D point is used to query the FV feature maps in that same 2D position to obtain the corresponding feature vector. The feature vector is extracted, and can be denoted as $\mathcal{F}^{fv}_{l,m}$.

3. **Feature Transformation**. The extracted FV feature vector $\mathcal{F}^{fv}_{l,m}$ is fed through a series of fully connected layers, aiming to reconstruct the BEV feature vector position $\{i, j\}$ in the BEV feature map: $\mathcal{F}^{bev}_{i,j}$.

4. **Feature Map Reconstruction.** By applying the step 1 through 3 for each index $\{i, j\}$ in the BEV feature map's spatial dimension, we are able to reconstruct a new feature map that has the same spatial dimensions as the BEV feature map, and the transformed values are learned using MLPs based on the feature vectors obtained from the FV feature maps.

The continuous convolution process is applied to the multi-scale feature map obtained from the frontal view stream. Three different target feature maps with three different scales are reconstructed after this stage. The complete process is illustrated in Figure 5.3.

Intermediate Fusion. The outcome of the continuous convolution process is then fused with each of the three multi-scale BEV feature maps obtained from the BEV stream 5.2, before applying the concatenation operation. The fusion of the BEV feature maps with the transformed FV feature maps is done by applying the Multi-modal Factorized Bilinear Pooling [57].

Output Map. The fused, multi-scale feature maps are then concatenated in the channels dimensions, then passed to the final BEV detection heads, discussed in 5.2.

5.2 Detection Heads

The fused feature maps from both CNN streams are fed into CNN-based header networks that are responsible for encoding the final detections. As mentioned in 4.1.2, the final bounding boxes are predicted from the top view to simplify the detection task.

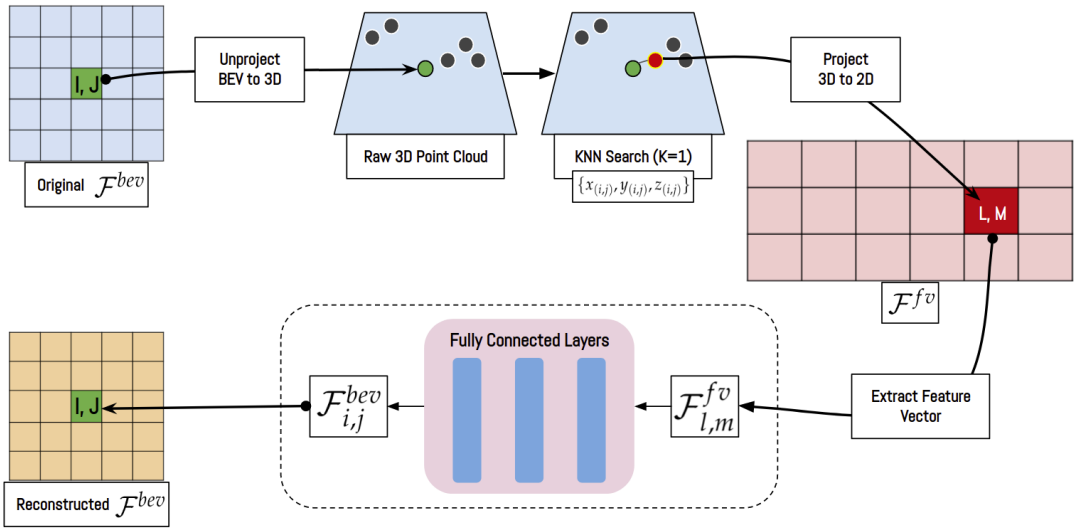


FIGURE 5.3: Overview of the Continuous Convolution process [55]. Starting with the original BEV feature map $\mathcal{F}^{bev}$, we unprojected it to 3D in order to find its nearest neighbouring 3D LiDAR point. Then, using the calibration parameters, we project the nearest point into the frontal view (FV) to extract the corresponding feature vector. The extracted feature vector is fed into a series of MLPs in order to be reconstructed in the BEV.

5.2.1 Classification Head

After combining the fused, multi-scale feature maps, a sigmoid-activated 1×1 convolution operation is used to generate a class-specific objectness map (probability of the presence of an object in BEV), which is directly learnt by the help of the BEV objectness maps during the supervised learning process. The size of this objectness map is: $224 \times 256 \times 1$.

5.2.2 Regression Head

Bounding box information are encoded in a regression map of size $224 \times 256 \times 8$, as discussed in 4.1.2. We train our sensor fusion architecture to generate oriented 3D bounding boxes, thus, we encode the following information in the regression map: size, position, and heading angle. For each pixel in the target regression map, the model is trained to estimate eight values:

- The normalized size in the 3 different dimensions as: $(\log(w), \log(h), \log(l))$.
- The offset value from the center of the estimated bounding box: (d_x, d_y). We also estimate the altitude between the ground and the center of the box (alt).

- The factorized orientation for the estimated bounding box as: $(\cos(\theta), \sin(\theta))$.

Each of the eight regression parameters is estimated using dedicated, linearly-activated 1×1 convolution operation. The complete sensor fusion framework architecture is demonstrated in Figure 5.4.

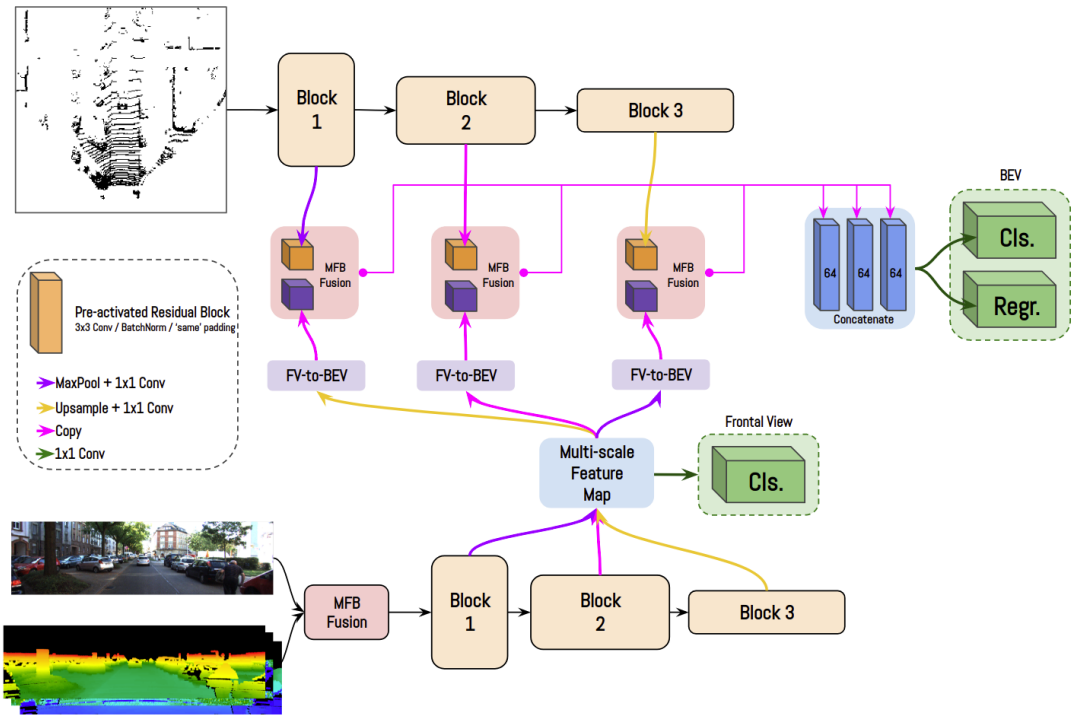


FIGURE 5.4: Overall sensor fusion framework architecture.

5.3 Conclusion

In this chapter, we have explained our sensor fusion architecture in detail. The most important components in our design are: (1) the use of pre-activated residual convolutional blocks [24], (2) employing MFB [57] fusion mechanism for both early and intermediate fusion, and (3) adding an additional classification head for the frontal view stream to increase the quality of the feature maps before applying intermediate fusion.

In the next chapter, we will be experimenting our sensor fusion framework and analyzing the impact of our design decisions on the detection accuracy by evaluating on the KITTI [6] dataset.

Chapter 6

Experimentation

6.1 Experimental Setting

In this chapter, we present various experiments that we have performed to study and analyze our sensor fusion framework, and to conclude which are the most important factors that contribute in building such an end-to-end fusion system. We provide a detailed experimental setting, along with experiments in different scopes, such as: training and optimization 6.2, network architecture 6.3, fusion mechanisms 6.5.

We perform our experimentation on KITTI [6] validation set, which comprises around 50% of the publically available training data, and we are following a train/val split from [39]. Using the validation set gives us the flexibility to study and analyze our propose sensor fusion framework, compared to the official test set, which is only available on the KITTI evaluation server for benchmarking. All the details of the experimental setting are shown in Table 6.1. We evaluate our models based on AP at IoU $= 0.5$, and we train and evaluate on the "Car" class. We train the deep learning models for 60 epochs using RMSProp optimization algorithm with an exponentially decaying learning rate schedule, starting with an initial learning rate of 0.0005 for the first 30 epochs with a decay factor of 0.8 every epoch, followed by a decaying learning rate of 0.0003 for the followings 30 epochs. We report evaluation results after evaluating the checkpoints for the last 15 epochs to capture the best performing model. We train the models on 2 GPUs in parallel, each GPU is processing a batch size of 2. For the classification task, we are using a focal loss with $\alpha = 0.75$ and $\beta = 1.0$. For the regression task, we are using a smooth L1 loss with $\sigma = 3.0$. We use a subsampling factor of 0.7 in the construction of the objectness maps, unless stated

otherwise.

We are following the same details mentioned in the Methodology chapter 5 for the input/output shapes and the input/output encodings. Our deep learning models consist of pre-activated residual convolutional blocks, unless stated otherwise. Regarding the fusion between feature maps, MFB [57] learnable fusion is used, unless stated otherwise. The *k*-NN process required for the continuous convolution (feature map transformation) has been performed before the training process. Data augmentation is being performed in an online manner (during training). We are using 2 NVIDIA GeForce GTX 1080Ti in parallel for training. We are using TensorFlow [63] software for building the training pipelines, model creation, and the optimization procedure, along with their visualization software, TensorBoard.

Target Class (train/val)	Car
Number of Training Samples	3712
Number of Training Annotations	14357
Number of Validation Samples	3769
Number of Validation Annotations	14385
Optimizer	RMSProp
Conv. Block	Pre-activated Residual Block [24]
Activation	ReLU [11]
Normalization	BatchNorm [16]
Training GPUs	**2**–NVIDIA GeForce GTX 1080Ti
Batch Size	2 (per GPU)
Training Framework	TensorFlow-GPU v2.2.0rc2
Visualization Framework	TensorBoard v2.2.2
OS	Ubuntu 18.04
CUDA	10.1
CuDNN	7.6
Evaluation GPU	NVIDIAGeForce GTX TITAN X
Evaluation Code	Official KITTI C++ Eval. Kit [64]

TABLE 6.1: Experimental Setting.

6.2 Training & Optimization

6.2.1 Data Augmentation

Performing data augmentation during the training process is critical to reduce overfitting, since the variety of driving scenarios in KITTI is very limited. We were able to implemented data augmentation strategies that are applicable to both LiDAR and

Camera inputs in the same input frame. Each data augmentation technique is performed on both input modalities with the help of the provided calibration parameters. Augmentations are applied based on specific probabilities, as shown in Table 6.2.

Augmentation	Probability
Translation	0.3
Scaling	0.3
Dropout	0.1
Flipping	0.5

TABLE 6.2: Probabilities attached to each augmentation technique.

In Table 6.3, we are comparing the detection accuracy (AP) for both cases: with and without applying data augmentations. The models with and without data augmentations are trained for 54 and 55 epochs, respectively. We observe a minimum of 7.51% increase in AP for the *easy* cases, and a maximum of 10.73% increase for the *hard* cases in the BEV detection task. Clearly, the model that was trained without applying data augmentation has overfit on the training data, which reduced its ability to generalize on the validation data.

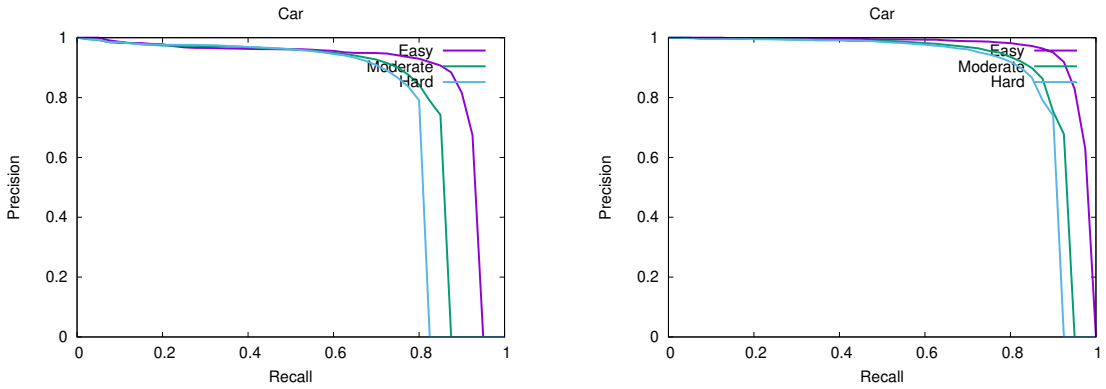
Figure 6.1 shows the difference between precision-recall plots for BEV $AP_{0.5}$ for *easy*, *moderate*, and *hard* cases.

6.2.2 Multi-task Training

We observed that multi-task training can boost detection accuracy. In addition to optimizing for the BEV learning tasks (BEV objectness and BEV regression maps),

Data Augmentation	Easy	Moderate	Hard
	BEV Detection		
without	87.65%	80.30%	76.05%
with	**95.16%** (+7.51)	**89.12%** (+8.82)	**86.78%** (+10.73)
	3D Detection		
without	83.48%	73.27%	69.35%
with	**91.75%** (+8.27)	**82.94%** (+9.67)	**80.42%** (+11.07)
	2D Detection		
without	67.29%	69.88%	69.76%
with	**87.26%** (+19.97)	**83.13%** (+13.25)	**81.17%** (+11.41)

TABLE 6.3: BEV, 3D, and 2D detection accuracies with $AP_{0.5}$ (%) comparing the trained architecture with and without applying data augmentation techniques.



(A) Without Data Augmentation.

(B) With Data Augmentation.

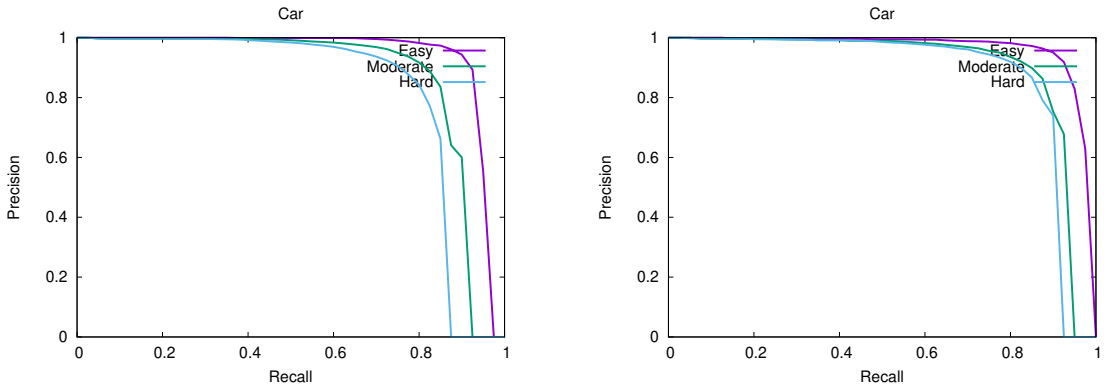
FIGURE 6.1: Precision-recall plots comparing the impact of data augmentations on BEV AP for the 'Car' class.

we also leverage the power of the frontal view stream by training it to generate 2D frontal view objectness maps. This modification helps increase the quality of the frontal view feature maps before fusing them with the BEV feature maps using the learnable fusion mechanism, which yields better overall detection accuracy. The impact of such modifications are shown in Table 6.4. We got a minimum of 2.07% increase for *easy* cases, and a maximum of 5.33% increase for *hard* cases in BEV object detection task. The additional supervision task for generating 2D objectness maps had a huge impact on 3D detection as well, with 9.59% increase in AP in *moderate* cases, and 11.86% increase in *hard* cases.

Learning Tasks	Easy	Moderate	Hard
	BEV Detection		
BEV Obj. + BEV Regr.	93.09%	86.26%	81.45%
BEV Obj. + BEV Regr. + FV Obj.	**95.16%** (+2.07)	**89.12%** (+2.86)	**86.78%** (+5.33)
	3D Detection		
BEV Obj. + BEV Regr.	85.87%	73.35%	68.56%
BEV Obj. + BEV Regr. + FV Obj.	**91.75%** (+5.88)	**82.94%** (+9.59)	**80.42%** (+11.86)
	2D Detection		
BEV Obj. + BEV Regr.	85.81%	80.33%	76.07%
BEV Obj. + BEV Regr. + FV Obj.	**87.26%** (+1.45)	**83.13%** (+2.8)	**81.17%** (+5.1)

TABLE 6.4: Detection accuracy AP (%) for BEV, 3D, and 2D object detection tasks comparing trained sensor fusion models optimizing for different supervised learning tasks (multi-task learning).

Figure 6.2 shows the effect of multi-task training on the precision-recall plots for BEV AP detection.



(A) Without Multi-task Training.

(B) With Multi-task Training.

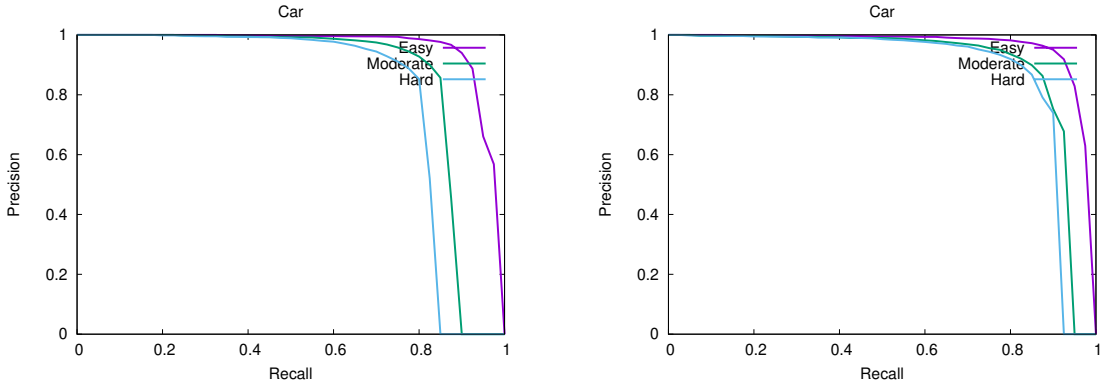
FIGURE 6.2: Precision-recall plots comparing the impact of multi-task training on BEV AP for the 'Car' class.

6.2.3 Dropping Boundary Pixels

Another important detail that we have implemented to support the supervised training process is dropping boundary pixels – i.e. subsampling, of the groundtruth bounding boxes based on which the focal loss is being calculated. This modification significantly impacts the AP (%) for *moderate* and *hard* cases. To evaluate the impact of this modification, we train two versions of our model, one with subsampling factor equal to 1.0 (no subsampling), and we apply a subsampling factor of 0.7 to the other model. Table 6.5 shows the improvements we were able to achieve in BEV, 3D, and 2D detection accuracy on KITTI validation set. Figure 6.3 shows the precision-recall curves on BEV AP. It is worth mentioning that we have used a subm-sampling factor of 0.7 after a series of pre-experimentation of the best subsampling factor in the set of $\{0.5, 0.6, 0.7, 0.8\}$. The main goal behind using a subsampling factor during training is to make the classification targets harder to learn, which will result in improved generalization of our sensor fusion framework. That being said, it is important to figure out a subsampling factor that will enable generalization without making it impossible for the learnable parameters to reach a global optima.

Subsampling Factor	Easy	Moderate	Hard
	BEV Detection		
1.0	94.73%	84.59%	79.47%
0.7	**95.16%** (+0.43)	**89.12%** (+4.53)	**86.78%** (+7.31)
	3D Detection		
1.0	**92.35%**	78.19%	73.07%
0.7	91.75% (-0.60)	**82.94%** (+4.75)	**80.42%** (+7.35)
	2D Detection		
1.0	85.56%	82.9%	79.08%
0.7	**87.26%** (+1.70)	**83.13%** (+0.23)	**81.17%** (+2.09)

TABLE 6.5: Comparing our sensor fusion architecture with different subsampling factor used for dropping boundary pixels in the target objectness maps during training. We provide evaluation results on BEV, 3D, and 2D evaluation.



(A) Subsampling Factor = 1.0

(B) Subsampling Factor = 0.7

FIGURE 6.3: Precision-recall plots comparing the impact of dropping boundary pixels of groundtruth boxes when computing loss function.

6.3 Network Architecture

6.3.1 Input Modalities

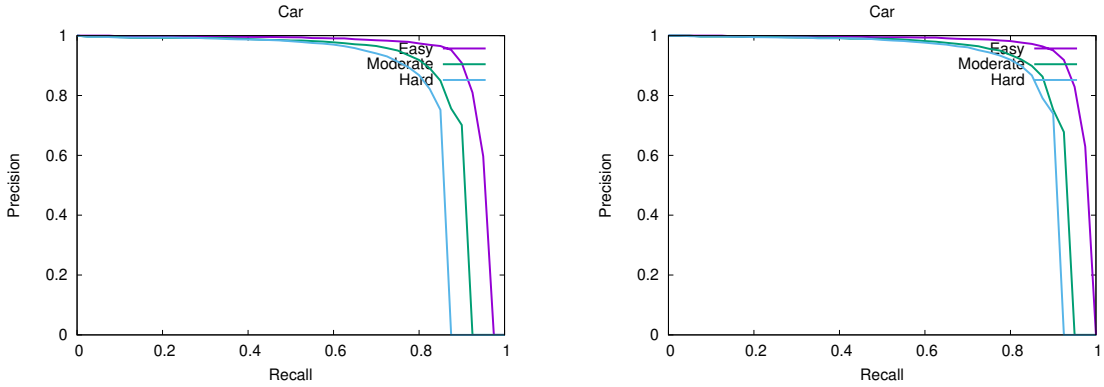
We have analyzed the impact of adding projected LiDAR frontal view features such as intensity, height, and depth maps, as an additional input to the frontal view stream. We observed significant improvements in AP (%) in all three categories (*easy*, *moderate*, and *hard*), as well as in all three detection tasks (BEV, 3D, and 2D). Quantitative results are shown in Table 6.6, and the precision-recall curves are visualized in Figure 6.4.

Without using the frontal view LiDAR features, we would only have an RGB image as input to the FV stream, which has color and texture information, but lacks depth, intensity, and height information, which all are of a high importance when it

comes to the task of 3D and BEV object detection.

Input Modalities	Easy	Moderate	Hard
	BEV Detection		
LiDAR BEV + RGB	92.45%	86.48%	81.77%
LiDAR BEV + RGB **+ LIDAR FV**	**95.16%** (+2.71)	**89.12%** (+2.64)	**86.78%** (+5.01)
	3D Detection		
LiDAR BEV + RGB	88.01%	77.04%	72.33%
LiDAR BEV + RGB **+ LIDAR FV**	**91.75%** (+3.74)	**82.94%** (+5.90)	**80.42%** (+8.09)
	2D Detection		
LiDAR BEV + RGB	82.8%	80.71%	78.5%
LiDAR BEV + RGB **+ LIDAR FV**	**87.26%** (+4.46)	**83.13%** (+2.42)	**81.17%** (+2.67)

TABLE 6.6: Comparing our sensor fusion architecture with different input modalities, showing the impact of using LiDAR's frontal view projections.



(A) Without LiDAR FV Features.

(B) With LiDAR FV Features.

FIGURE 6.4: Precision-recall plots comparing the impact of using LiDAR frontal view features as additional input to the frontal view stream.

6.3.2 CNN Design

We also include experiments with several types on convolutional blocks to analyze the impact of such design decisions on our fusion framework's detection accuracy. In Table 6.7, we show that it is crucial to use robust convolutional blocks in the feature extraction process to generate high quality feature maps that are ready for fusion and which can also increase the detection accuracy of the deep learning model. We compared three types of convolutional blocks: (1) separable convolutions, which where used in MobileNets [28], (2) standard convolution, and (3) pre-activated residual convolutions [24]. Huge improvements where achieved in the BEV detection task by using the pre-activated residual convolutional blocks, with a 12.11% increase in

hard cases compared to standard convolution, and a 22.08% increase in *hard* cases compared to separable convolutions.

The main reason behind such improvements is the robustness of the pre-activated residual convolutions. The pre-activated residual convolutional enables a direct linear connection, not just within a convolutional block, but throughout the whole network architecture, which enables more interaction between different feature maps within the multi-stream sensor fusion framework. It is clear that more interaction between feature maps in a multi-stream, multi-modal, and multi-view CNN architecture is of a huge significance. Without such interactability between different components of the proposed architecture, the objective function would be much harder to optimize, which would have a direct impact on the overall detection accuracy.

Conv. Type	Easy	Moderate	Hard
	BEV Detection		
Separable Conv.	77.92%	71.03%	64.7%
Standard Conv.	85.47% (+7.55)	78.96% (+7.93)	74.67% (+9.97)
Pre-activated Residual Conv.	**95.16%** (+9.69)	**89.12%** (+10.16)	**86.78%** (+12.11)
Overall Improvement (AP%)	(+17.24)	(+18.09)	(+22.08)
	3D Detection		
Separable Conv.	65.77%	54.47%	49.35%
Standard Conv.	74.25% (+8.48)	60.29% (+5.82)	59.06% (+9.71)
Pre-activated Residual Conv.	**91.75%** (+17.50)	**82.94%** (+22.65)	**80.42%** (+21.36)
Overall Improvement (AP%)	(+25.98)	(+28.47)	(+31.07)
	2D Detection		
Separable Conv.	64.38%	65.29%	64.68%
Standard Conv.	63.83% (-0.55)	65.56% (+0.27)	65.24% (+0.56)
Pre-activated Residual Conv.	**87.26%** (+23.43)	**83.13%** (+17.57)	**81.17%** (+15.93)
Overall Improvement (AP%)	(+22.88)	(+17.84)	(+16.49)

TABLE 6.7: Comparing BEV AP using different types of convolutional blocks.

6.3.3 Inference Speed

Another aspect to the choice of the convolution type is its impact on the inference speed and the number of learnable parameters – i.e. the model size. The details of such CNN design choices is presented in Table 6.8. While separable convolutions yield the least number of parameters $563K$ and the fastest inference speed $86ms$, however, the trade-off between speed and performance (moderate AP%) is the reason why we still choose pre-activated residual convolutions over separable convolutions as the main building blocks to our feature extraction networks. In these experiments, we measure the inference time on a single NVIDIA GeForce GTX TITAN X.

Conv. Type	Number of Parameters	Inference Speed (ms)	BEV AP (%) Moderate
Separable Conv.	**563K**	**85**	71.03%
Standard Conv.	2.16M	90	78.96%
Pre-activated Residual Conv.	4M	103	**89.12%**

TABLE 6.8: Comparing Inference Speed for sensor fusion architectures with different types of convolutional blocks.

6.4 Fusion Mechanisms

One of the most critical components inside our sensor fusion framework is the intermediate fusion strategy, which fuses feature maps from both BEV and FV streams. In our experiments, we show the importance of adopting learnable fusion modules, such as MFB [57], in object detection tasks used within autonomous driving system. After making sure that both BEV and FV streams generate high quality multi-scale feature maps, the fusion stage could either be a bottleneck or a booster for the final CNN detection header network. In Table 6.9, we perform experiments with several fusion types: (1) element-wise addition, which is widely adopted in previous methods like [55], (2) element-wise multiplication, which has not been used yet in

sensor fusion frameworks, and (3) a learnable fusion module, called MFB [57], imported from the field of temporal activity detection in videos, and adapted to object detection domain.

We show that element-wise addition yields the worst BEV detection accuracy. Element-wise multiplication fusion technique improves BEV AP with 6.44%, 4.97%, and 2.84%, in *easy*, *moderate*, and *hard* cases, respectively. The reason behind a better performance for element-wise multiplication is that it is designed to solidify the high-valued features to be passed to the subsequent layers, and to diminish any low-valued features from going on to the next layers. We can think of it as a ReLU [11] layer, but for a multi-modal input.

Both element-wise addition and multiplication fusion models are outperformed by a similar architecture that employs a learnable fusion mechanism, the Multi-modal Factorized Bilinear Pooling (MFB) [57]. MFB introduces significant improvements over element-wise multiplication in BEV AP%, with an 2.92% increase in *easy* cases, 3.38% increase in *moderate* cases, and a very important 6.16% increase in *hard* cases. Similar significant improvements can also be observed in both 3D and 2D detection AP%. The most important advantage of a learnable fusion mechanism when compared to a fixed mathematical operation (e.g. addition or multiplication), is that it is tuned and tweaked based on the objective function, to produce the best fusion function that will contribute to the lowest training error.

Intermediate Fusion Type	Easy	Moderate	Hard
	BEV Detection		
Element-wise Addition	85.8%	80.77%	77.78%
Element-wise Multiplication	92.24%	85.74%	80.62%
Multi-modal Factorized Bilinear Pooling	**95.16%**	**89.12%**	**86.78%**
	3D Detection		
Element-wise Addition	68.61%	59.75%	57.29%
Element-wise Multiplication	87.14%	76.64%	71.38%
Multi-modal Factorized Bilinear Pooling	**91.75%**	**82.94%**	**80.42%**
	2D Detection		
Element-wise Addition	71.12%	69.14	68.93%
Element-wise Multiplication	84.32%	79.71%	77.37%
Multi-modal Factorized Bilinear Pooling	**87.26%**	**83.13%**	**81.17%**

TABLE 6.9: Comparing our sensor fusion architecture with different fusion techniques, showing the importance of using learnable fusion modules to boost detection accuracies.

Sensors	Views	Easy	Moderate	Hard
		BEV Detection		
LiDAR only	BEV	92.53%	87.62%	83.03%
LiDAR + Camera (ours)	BEV + FV	**95.16%**	**89.12%**	**86.78%**
		3D Detection		
LiDAR only	BEV	85.88%	74.85%	70.01%
LiDAR + Camera (ours)	BEV + FV	**91.75%**	**82.94%**	**80.42%**
		2D Detection		
LiDAR only	BEV	79.02%	77.14%	75.18%
LiDAR + Camera (ours)	BEV + FV	**87.26%**	**83.13%**	**81.17%**

TABLE 6.10: Comparing the detection accuracy of our sensor fusion framework to LiDAR-only architectures, like [46].

6.5 Sensor Fusion vs. LiDAR-only

In Table 6.10, we show how our multi-modal multi-view framework is able to outperform a LiDAR-only method. We have chosen to compare our framework's performance to our own implementation of PIXOR [46]. The reason behind choosing PIXOR is the architectural similarities between their LiDAR backbone network, which operates on top-view images, and our proposed bird's eye view stream. Other LiDAR-only methods such as VoxelNet [43] or PointPillars [44] use a different encoding to construct the top-view input (e.g. voxelization or dividing the 3D space into pillars). However, PIXOR [46] use a similar discretization technique to construct an occupancy grid, which is similar to ours, hence the choice of comparison. As reported in 6.10, our multi-modal sensor fusion framework has a higher detection accuracy (AP_{40} at IoU 0.5) in all detection tasks, and under all difficulty levels, compared to our implementation of PIXOR [46].

6.6 Qualitative Results

In this section, we show scenarios captured from KITTI [6] dataset after using our sensor fusion network to generate oriented bounding box detections for the 'Car' class. All the visualizations have the similar setting:

- *Top left image:* 2D bounding boxes on top of a frontal view RGB image.
- *Bottom left image:* projected 3D bounding boxes on top a frontal view RGB image.

- *Right image:* projected BEV bounding boxes on top of a BEV projection of the LiDAR input point cloud.
- All the visualized detections have been filtered based on a confidence threshold equals to 80%. This filtering thershold is applied right after NMS.
- All samples used to generate the bounding boxes are obtained from the validation set.
- Groundtruth boxes are drawn in *green*, while Predictions are drawn in *red*.

Figure 6.5 shows a car that has not been detected due extreme occlusion for both sensor modalities, Camera and LiDAR. Occlusion is one of the most challenging problems in object detection in perception systems. One potential solution would be integrating the detection module with another tracking module that is able to track cars and estimate their new position in the frame where they are fully occluded based on previous frames.

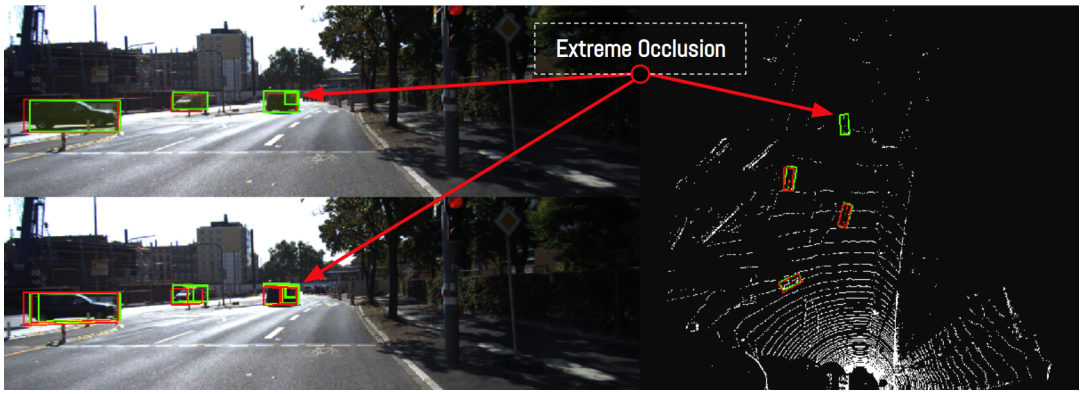


FIGURE 6.5: A scenario from KITTI [6] where a car is extremely occluded with respect to both Camera and Lidar. GT: green, PRED: red.

Figure 6.6 shows a rare and challenging scenario where our fusion network fails to detect any cars. We face this issue due to the lack of driving scenarios in KITTI, which is one of the reasons we suggest in our Future Work 7.2 section to explore other large-scale data sets such as NuScenes [38] or Lyft [56]. Figure 6.7 shows another failure case due to the cars' small size (in pixels), and being partially occluded by pedestrians.

Figures 6.8 and 6.9 are showing the capability of our sensor fusion network to accurately estimated oriented bounding boxes for moving and non-moving cars for BEV, 3D, and 2D views, in a simultaneous manner.

FIGURE 6.6: A rare scenario from KITTI [6] that is tricky to our fusion network and yeilds a failure case. GT: green, PRED: red.

FIGURE 6.7: A challenging scenario from KITTI [6] that leads to failure for our fusion model due to small-sized objects and partial occlusion. GT: green, PRED: red.

Figures 6.9, 6.10, 6.11, and 6.12 are showing a fundamental issue with KITTI [6] dataset. In all four visualizations, we observe that many actual cars have not been annotated by the dataset, and our sensor fusion network is able to capture these cars. We are able to detect 4, 2, 1, and 1, unannotated cars in Figures 6.10, 6.11, 6.12, and 6.9, respectively. This issue with the initial inaccurate annotation leads to a drop in the AP metric, as these actual cars are not considered as groundtruth, and are being treated as *false positives* during the evaluation process, while these detections should be counted as *true positives*. Accurate annotation of 3D bounding boxes is a challenging and a highly demanding task. That is why it would be beneficial to explore recently created datasets such as NuScenes [38] and Lyft [56] where these problem have been put into consideration during the process of building the datasets.

FIGURE 6.8: A scenario from KITTI [6] shows our network's high detection accuracy for moving cars. GT: green, PRED: red.

FIGURE 6.9: A scenario from KITTI [6] that shows our network's high detection accuracy for non-moving cars. GT: green, PRED: red.

6.7 Discussion

The use of IoU 0.5 **for the Car class.** In our experiments, we have used KITTI's [6] official evaluation metric, the average precision with 40 recall points AP_{40}, with an acceptable intersection over union (IoU) threshold of 0.5. We believe that besides KITTI being the most well-established benchmark for object detection, it still has multiple challenges and downsides, including the number of available training examples, and the variety of the driving scenarios. We attempt to tackle these challenges by applying multiple multi-modal label-preserving data augmentation techniques, however, to get a high detection accuracy using an IoU threshold of 0.7 for the Car class, we recommend using the whole training set (around 7k examples) along with applying augmentations before submitting the test results to KITTI's evaluation server. That being said, using an IoU threshold of 0.5 is reasonable for studying and analysing the behaviour of the sensor fusion framework as it has less constraints on the exact localization task, and focuses more on the detection task.

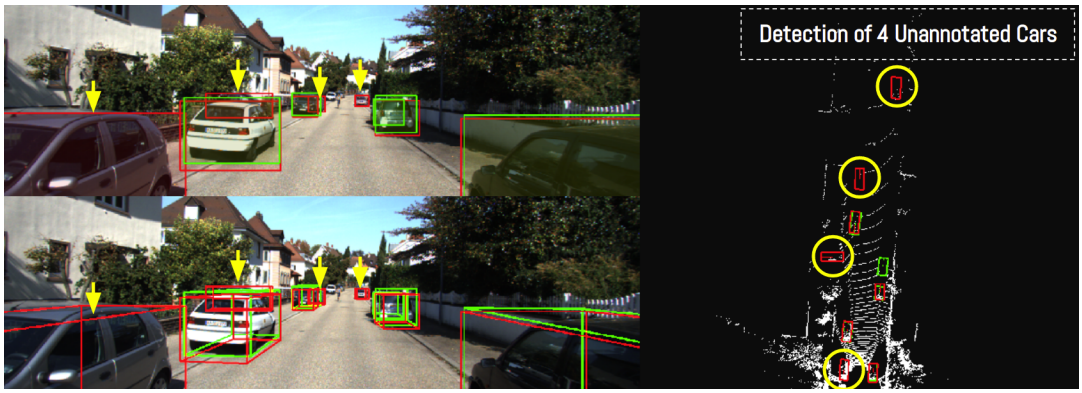


FIGURE 6.10: A scenario from KITTI [6] where 4 unannotated cars were correctly detected by our fusion network. GT: green, PRED: red.

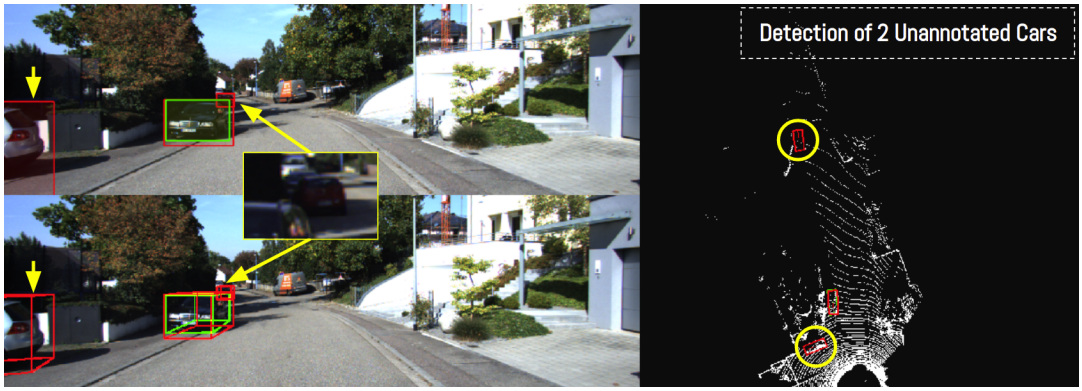


FIGURE 6.11: Another scenario from KITTI [6] where 2 unannotated cars were correctly detected by our fusion network. GT: green, PRED: red.

Comparing with SOTA sensor fusion architectures. In 2019, the authors of [40] have undercovered a critical glitch in KITTI's official evaluation metric, which was an average precision with 11 recall points starting at 0 (AP_{11}), as it took only one correct detection per difficulty level to get an AP of 9.09%, and that is why they proposed to use an average precision with 40 recall points, while neglecting the precision at recall$=0$. The aforementioned modification results in more fair comparison between different detection architectures.

Since we have reviewed multiple sensor fusion architectures in Chapter 3, such as MV3D [47], AVOD [54], or PointFusion [53], we aimed to obtain their performance on KITTI's validation set. However, most of these methods were published before 2019, when the glitchy AP_{11} was used. In order to still be able to compare our proposed framework with other methods published before 2019, we have re-evaluated our framework's performance using AP_{11}, and compared it only with other methods who reported their detection accuracy on KITTI's validation at IoU 0.5. Despite

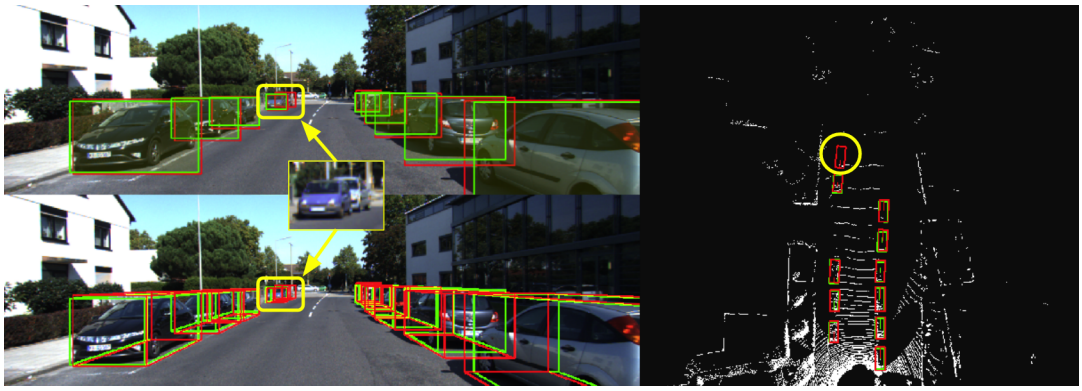


FIGURE 6.12: A scenario from KITTI [6] that shows our network's high detection accuracy, while capturing unannotated cars. GT: green, PRED: red.

knowing the unfair outcome of such a comparison, we aimed to provide a complete set of evaluations, as well as denoting important aspects of such challenges to be considered in the future by other fellow researchers.

In Table 6.11, we compare our proposed sensor fusion framework with other methods which reported their detection accuracies on KITTI's validation set, which was proposed in [60]. As shown in the table, our method outperforms other uni-modal architectures like Mono3D [50], 3DOP [60], and VeloFCN [61]. The performance of MV3D [47] outperforms ours, however, it is worth noting that MV3D is a two-stage architecture, while ours is a single-stage, end-to-end architecture.

		BEV Detection (AP_{11})		
Method	Modalities	Easy	Moderate	Hard
Mono3D [50]	Monocular Camera	30.50%	22.39%	19.16%
3DOP [60]	Stereo Vision	55.04%	41.25%	34.55%
VeloFCN [61]	LiDAR	79.68%	63.82%	62.80%
MV3D [47]	LiDAR + Camera	96.34%	89.39%	88.67%
Ours	LiDAR + Camera	89.82%	87.77%	86.56%

TABLE 6.11: Comparing our proposed sensor fusion framework with other methods who provided their performance results on KITTI's validation set at IoU 0.5.

	Inference Speed (ms)	BEV Detection (AP_{11})		
MV3D [47] (BEV + FV)	240	95.74%	88.57%	88.13%
MV3D [47] (BEV + FV + IMG)	360	96.34%	89.39%	88.67%
Ours (BEV + FV + IMG)	103	89.82%	87.77%	86.56%

TABLE 6.12: Comparison between our proposed framework and MV3D [47] that highlights the inference efficiency, as well as the detection accuracy in BEV.

Being a single-stage object detector, our proposed sensor fusion framework is able to produce high detection accuracy, while being more than 3x times faster than the equivalent MV3D two-stage architecture (which has the same input modalities), as shown in Table 6.12.

6.8 Summary of our Contributions

We would like to summarize our contributions into four main pillars.

Multi-task learning. Previous work which tackled the task of 3D object detection aimed to enhance the training process by learning other auxiliary tasks, such as HD maps' estimation [48], depth completion [65], or ground estimation [65]. While training sensor fusion framework on multiple tasks simultaneously is of a significant importance, however, we showed in our work that leveraging the multi-stream architecture can have the same impact of the detection accuracy, by adding an learnable target output from the stream that is not used for generating final detections. We have done that by adding a frontal view classification output to the frontal view stream, while our final detections are still being generated from the top-view, which means that adding this extra term to the objective function increases generalization and enhances the quality of the feature maps in the frontal view stream, which results in having better outcomes from the fusion mechanism.

Using frontal view LiDAR features. Normally, the frontal view stream would only have RGB image as input, as in MV3D [47]. However, we proposed to early fuse RGB with frontal view features extracted from the LiDAR sensor. RGB-only input would help by providing the network with color and texture information, but it still lacks information about depth, intensity, and height, which all three are important for robust 3D and BEV object detection and localization.

Using pre-activated residual convolutions. A major advantage of using pre-activate residual convolutional blocks is to enable forward and backward propagation of information throughout the whole architecture, not just within individual blocks, which is critical in multi-stream, multi-view, sensor fusion frameworks, as it enables more interaction between feature maps in both multi-view streams.

Employing robust fusion mechanisms. In our work, we proposed feature maps' level fusion with a goal to enable more interaction between feature maps in both CNN streams. We also proposed changing a fusion technique that has been widely used in the literature, which is the element-wise addition, by element-wise multiplication, which resulted in high detection accuracy. Our analysis suggests that it solidifies high-valued features and enable them to propagate to further layers, while diminishing low-valued features in both modalities and stopping them from progressing more in the fusion network. Moreover, we are the first work to integrate bilinear pooling in sensor fusion architectures that tackles the challenge of 2D, 3D, and BEV object detection and localization. We introduce an efficient way to implement it in single-stage detectors to maintain a fully-convolutional architecture.

Finally, it is worth emphasizing on the importance of these design decision in other object detection or sensor fusion architectures, and that these contributions are not limited to the perception algorithms of autonomous vehicles.

6.9 Conclusion

In this chapter, we have presented extensive experimental analysis on different design aspects of the sensor fusion framework, while providing strong arguments for each of our design decision, including: the chosen input modalities and their representations, the chosen fusion mechanisms, the use of multi-task learning, as well as the architectural design of our end-to-end, single-stage, fully convolutional architecture.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this thesis, we have provided a comprehensive review of deep-learning based sensor fusion architecture that address the object detection and localization task. We have proposed a Camera-LiDAR multi-view sensor fusion framework that operates on both bird's eye view (BEV) and frontal view (FV). We show the impact of performing early fusion in the FV stream, along with feature maps' level fusion – i.e. intermediate fusion, for both streams. We also studied the effect of critical procedures applied during training, including data augmentation and multi-task learning. We support our analysis with a comparison between various convolutional blocks and their contribution to the final detection accuracy, which solidifies our choice of using pre-activated residual blocks [24]. We supplement such analysis with inference speed and model size details.

We show the importance of choosing a robust fusion mechanism. We propose exchanging the Element-wise Addition fusion operation, which has been used in the literature, with Element-wise Multiplication, and we quantitatively show the impact of such modification on KITTI's [6] BEV AP. Moreover, we adapt a fusion mechanism from the temporal activity detection field to the object detection field, namely Multi-modal Factorized Bilinear Fusion (MFB) [57], and show how fusion-based deep learning architectures can benefit from learnable fusion techniques. Our findings paves a new path for developing more robust learnable fusion mechanisms, which will boost sensor fusion frameworks, and leverage modern automotive sensors' setups with multi-modal data streams to the highest extent.

After performing the aforementioned analysis and experimentation, we can conclude that employing robust sensor fusion mechanisms along with the right design decisions can be as good, or even outperform single sensor based methods. Most of the previous work in the field of object detection for autonomous vehicles have focused solely on LiDAR only methods due to its high localization accuracy. In this work, we show that such methods can be enhanced with rich information from other sensors as well. Sensor fusion can be beneficial in the case of a sensor failure or an unexpected noise, and would increase the reliability of the perception module in an autonomous vehicle. Having another input modality can overcome these challenges.

In some cases, the camera sensor might be affected by severe light conditions, in which case, the LiDAR will be considered as the source of truth for the detection algorithm. To overcome that, the camera stream can also be trained on augmented input images with various pixel-level manipulations that would mimic different light variations (e.g. sun, snow, rain). Sever occlusion can still be a problem even when both sensors are used and fully utilized. However, the effect of the occlusion in consecutive frames can be mitigated by using a tracking algorithm, or operating on consecutive LiDAR sweeps. Finally, advanced automotive can have complex sensor kits with tens of different types of sensors in-place, and finding a way to combine and learn from all of these different modalities, instead of relying on a single sensor, is an important challenge that has to be more investigated by researchers in the future.

7.2 Future Work

Based on our comprehensive analysis and experimentation, our suggestions for future work can be in four directions: (1) the underlying design of the deep learning architecture, (2) the fusion mechanisms, (3) extending the current work to other LiDAR-based methods, and (4) extending to other datasets.

Underlying design of the deep learning architecture. In our proposed method, we employ CNN-based feature extractor networks, that are the defacto architecture

for computer vision tasks in the past 10 years. However, new work has been emerging that is using Transformers [66] as feature extractors, and this new paradigm shift is looking for replacing CNNs with Transformers, after a huge success in the NLP field. We suggest implementing a robust vision transformers (ViT) as backbone feature extractors for the multi-view inputs modalities to the sensor fusion framework.

Another direction would be to work on improving the inference speed of our sensor fusion architecture to make it usable in real-time applications, and on embedded systems as well.

Fusion mechanisms. There is a huge room for further development of more robust learnable fusion mechanisms, which could be started from MFB [57], which we have employed in our sensor fusion framework.

Extending to other LiDAR-based methods. In our proposed method, the design and training procedure of our framework are inspired by [46], [48], [55]. Currently, there are novel methods tackling the object detection task based only on LiDAR inputs. It would be interesting to investigate if fusing Camera input with such methods would result in more accurate detectors.

Extending to other datasets. In our work, we have used a well-established benchamrking dataset, namely KITTI [6]. However, using the KITTI dataset for training has multiple disadvantages, such as: the small number of frames with limited driving scenarios and conditions. Extending our work to large-scale datasets, such as: NuScenes [38], or Lyft [56] datasets would be extremely beneficial, as it is going to further solidify the concept that sensor fusion aims to overcome challenges with individual sensors. Moreover, such large-scale datasets employ more sensors to collect data, such as 5 or more cameras (KITTI provides only stereo images), LiDAR sweeps with faster frequency, and radar point clouds. Also, these datasets contain a large set of classes, compared to KITTI's three classes (cars, pedestrians, and cyclists).

Appendix A

Link of Scripts for all our experimentations

We provide the links for our code repository which contains training and evaluation pipelines, as well as our checkpoints for best performing models that were mentioned in the experimentation chapter (Chapter 6).

1. **GitHub repository:** Corresponding guidelines are provided within the repository's README.

2. **Google Drive checkpoints' folder:** Each checkpoint is saved in two files: *.h5* and *.json*. Scripts for loading these checkpoints are also provided within the folder.

Bibliography

[1] N. H. T. S. Administration, "Critical reasons for crashes investigated in the national motor vehicle crash causation survey," 2018. [Online]. Available: https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812506.

[2] Intel, "Accelerating the future: The economic impact of the emerging passenger economy," 2017. [Online]. Available: https://newsroom.intel.com/newsroom/wp-content/uploads/sites/11/2017/05/passenger-economy.pdf.

[3] *Autonomous Driving with Deep Learning: A Survey of State-of-Art Technologies*, Jun. 2020.

[4] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio, "Object recognition with gradient-based learning," Springer-Verlag, 1999, ISBN: 3540667229.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., vol. 25, Curran Associates, Inc., 2012, pp. 1097–1105.

[6] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[7] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, Sep. 2014. DOI: 10.1007/s11263-015-0816-y.

[8] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[9] T. Chen, H. Chen, and R.-w. Liu, "Approximation capability in by multilayer feedforward networks and related problems," *Neural Networks, IEEE Transactions on*, vol. 6, pp. 25 –30, Feb. 1995. DOI: 10.1109/72.363453.

[10] S. Sonoda and N. Murata, "Neural network with unbounded activation functions is universal approximator," *Applied and Computational Harmonic Analysis*, vol. 43, no. 2, pp. 233 –268, 2017, ISSN: 1063-5203. DOI: https://doi.org/10.1016/j.acha.2015.12.005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1063520315001748.

[11] V. Nair and G. Hinton, "Rectified linear units improve restricted boltzmann machines vinod nair," vol. 27, Jun. 2010, pp. 807–814.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[13] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, "Focal loss for dense object detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PP, pp. 1–1, Jul. 2018. DOI: 10.1109/TPAMI.2018.2858826.

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

[15] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html.

[16] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning*, F. Bach and D. Blei, Eds., ser. Proceedings of Machine Learning Research, vol. 37, Lille, France: PMLR, 2015, pp. 448–456. [Online]. Available: http://proceedings.mlr.press/v37/ioffe15.html.

[17] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How does batch normalization help optimization?" In *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and

R. Garnett, Eds., vol. 31, Curran Associates, Inc., 2018, pp. 2483–2493. [Online]. Available: https://proceedings.neurips.cc/paper/2018/file/905056c1ac1dad141560467e0a99e1cf-Paper.pdf.

[18] Y. Wu and K. He, "Group normalization," in *ECCV*, 2018.

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

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

[21] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9. DOI: 10.1109/CVPR.2015.7298594.

[22] M. Lin, Q. Chen, and S. Yan, "Network in network," *CoRR*, vol. abs/1312.4400, 2014.

[23] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2818–2826. DOI: 10.1109/CVPR.2016.308.

[24] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," vol. 9908, Oct. 2016, pp. 630–645, ISBN: 978-3-319-46492-3. DOI: 10.1007/978-3-319-46493-0_38.

[25] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *Proceedings of the British Machine Vision Conference (BMVC)*, E. R. H. Richard C. Wilson and W. A. P. Smith, Eds., BMVA Press, 2016, pp. 87.1–87.12, ISBN: 1-901725-59-6. DOI: 10.5244/C.30.87. [Online]. Available: https://dx.doi.org/10.5244/C.30.87.

[26] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *2017 IEEE Conference on Computer Vision*

and Pattern Recognition (CVPR), 2017, pp. 5987–5995. DOI: 10.1109/CVPR.2017.634.

[27] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," *AAAI Conference on Artificial Intelligence*, Feb. 2016.

[28] A. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," Apr. 2017.

[29] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 580–587. DOI: 10.1109/CVPR.2014.81.

[30] R. Girshick, "Fast r-cnn," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1440–1448. DOI: 10.1109/ICCV.2015.169.

[31] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28, Curran Associates, Inc., 2015, pp. 91–99.

[32] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788. DOI: 10.1109/CVPR.2016.91.

[33] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. Berg, "Ssd: Single shot multibox detector," vol. 9905, Oct. 2016, pp. 21–37, ISBN: 978-3-319-46447-3. DOI: 10.1007/978-3-319-46448-0_2.

[34] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 6517–6525. DOI: 10.1109/CVPR.2017.690.

[35] M. Everingham, L. Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, pp. 303–338, 2009.

[36] T. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 936–944. DOI: 10.1109/CVPR.2017.106.

[37] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds., Cham: Springer International Publishing, 2014, pp. 740–755, ISBN: 978-3-319-10602-1.

[38] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "Nuscenes: A multimodal dataset for autonomous driving," *arXiv preprint arXiv:1903.11027*, 2019.

[39] X. Chen, K. Kundu, Y. Zhu, H. Ma, S. Fidler, and R. Urtasun, "3d object proposals using stereo imagery for accurate object class detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PP, Aug. 2016. DOI: 10.1109/TPAMI.2017.2706685.

[40] A. Simonelli, S. R. Bulò, L. Porzi, M. Lopez-Antequera, and P. Kontschieder, "Disentangling monocular 3d object detection," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 1991–1999. DOI: 10.1109/ICCV.2019.00208.

[41] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 77–85. DOI: 10.1109/CVPR.2017.16.

[42] C. R. Qi, L. Yi, H. Su, and L. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," in *NIPS*, 2017.

[43] Y. Zhou and O. Tuzel, "Voxelnet: End-to-end learning for point cloud based 3d object detection," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4490–4499, 2018.

[44] A. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12 689–12 697, 2019.

[45] Y. Yan, Y. Mao, and B. Li, "Second: Sparsely embedded convolutional detection," *Sensors (Basel, Switzerland)*, vol. 18, 2018.

[46] B. Yang, W. Luo, and R. Urtasun, "Pixor: Real-time 3d object detection from point clouds," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 7652–7660. DOI: 10.1109/CVPR.2018.00798.

[47] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3d object detection network for autonomous driving," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 6526–6534. DOI: 10.1109/CVPR.2017.691.

[48] B. Yang, W. Luo, and R. Urtasun, "Pixor: Real-time 3d object detection from point clouds," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 7652–7660. DOI: 10.1109/CVPR.2018.00798.

[49] J. Zhou, X. Lu, X. Tan, Z. Shao, S. Ding, and L. Ma, "Fvnet: 3d front-view proposal generation for real-time object detection from point clouds," *2019 12th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, pp. 1–8, 2019.

[50] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, and R. Urtasun, "Monocular 3d object detection for autonomous driving," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2147–2156. DOI: 10.1109/CVPR.2016.236.

[51] X. Weng and K. Kitani, "Monocular 3d object detection with pseudo-lidar point cloud," *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pp. 857–866, 2019.

[52] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, "Frustum pointnets for 3d object detection from rgb-d data," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 918–927. DOI: 10.1109/CVPR.2018.00102.

[53] D. Xu, D. Anguelov, and A. Jain, "Pointfusion: Deep sensor fusion for 3d bounding box estimation," Nov. 2017.

[54] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. L. Waslander, "Joint 3d proposal generation and object detection from view aggregation," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 1–8. DOI: 10.1109/IROS.2018.8594049.

[55] M. Liang, B. Yang, S. Wang, and R. Urtasun, "Deep continuous fusion for multi-sensor 3d object detection," in *Computer Vision – ECCV 2018*, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds., Cham: Springer International Publishing, 2018, pp. 663–678, ISBN: 978-3-030-01270-0.

[56] R. Kesten, M. Usman, J. Houston, T. Pandya, K. Nadhamuni, A. Ferreira, M. Yuan, B. Low, A. Jain, P. Ondruska, S. Omari, S. Shah, A. Kulkarni, A. Kazakova, C. Tao, L. Platinsky, W. Jiang, and V. Shet, *Lyft level 5 perception dataset 2020*, https://level5.lyft.com/dataset/, 2019.

[57] M. A. Rahman and R. Laganière, "Mid-level fusion for end-to-end temporal activity detection in untrimmed video," in *BMVC*, 2020.

[58] J.-H. Kim and J. K. J.-W. H. B.-T. Z. Kyoung-Woon On Woosang Lim, "Hadamard product for low-rank bilinear pooling," *ArXiv*, 2017.

[59] Z. Yu, J. Yu, J. Fan, and D. Tao, "Multi-modal factorized bilinear pooling with co-attention learning for visual question answering," Oct. 2017, pp. 1839–1848. DOI: 10.1109/ICCV.2017.202.

[60] X. Chen, K. Kundu, Y. Zhu, A. G. Berneshawi, H. Ma, S. Fidler, and R. Urtasun, "3d object proposals for accurate object class detection," in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28, Curran Associates, Inc., 2015. [Online]. Available: https://proceedings.neurips.cc/paper/2015/file/6da37dd3139aa4d9aa55b8d237ec5d4a-Paper.pdf.

[61] B. Li, T. Zhang, and T. Xia, "Vehicle detection from 3d lidar using fully convolutional network," *ArXiv*, vol. abs/1608.07916, 2016.

[62] S. Wang, S. Suo, W.-C. Ma, A. Pokrovsky, and R. Urtasun, "Deep parametric continuous convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[63] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: http://tensorflow.org/.

[64] Carlos Guindel, *Eval$_k$itti*. [Online]. Available: https://github.com/cguindel/eval_kitti.

[65] M. Liang, B. Yang, Y. Chen, R. Hu, and R. Urtasun, *Multi-task multi-sensor fusion for 3d object detection*, Dec. 2020.

[66] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=YicbFdNTTy.