Camera Based Deep Learning Algorithms with Transfer Learning in Object Perception

Camera Based Deep Learning Algorithms with Transfer Learning in Object Perception

By

YUJIE HU, B.ENG.

A Thesis Submitted to the School of Graduate Studies

in Partial Fulfillment of the Requirements for the Degree
Master of Applied Science

McMaster University

© Copyright by Yujie Hu, September 2021

Master of Applied Science (2021)
McMaster University (Mechanical Engineering)
Hamilton, Ontario

TITLE:	Camera Based Deep Learning Algorithms with Transfer Learning in Object Perception
AUTHOR:	Yujie Hu B.Eng. (McMaster University)
SUPERVISORS:	Dr. Saeid Habibi, Dr. Ryan Ahmed
NUMBER OF PAGES:	xiv, 150

Abstract

The perception system is the key for autonomous vehicles to sense and understand the surrounding environment. As the cheapest and most mature sensor, monocular cameras create a rich and accurate visual representation of the world. The objective of this thesis is to investigate if camera-based deep learning models with transfer learning technique can achieve 2D object detection, License Plate Detection and Recognition (LPDR), and highway lane detection in real time. The You Only Look Once version 3 (YOLOv3) algorithm with and without transfer learning is applied on the Karlsruhe Institute of Technology and Toyota Technological Institute (KITTI) dataset for cars, cyclists, and pedestrians detection. This application shows that objects could be detected in real time and the transfer learning boosts the detection performance. The Convolutional Recurrent Neural Network (CRNN) algorithm with a pre-trained model is applied on multiple License Plate (LP) datasets for real-time LP recognition. The optimized model is then used to recognize Ontario LPs and achieves high accuracy. The Efficient Residual Factorized ConvNet (ERFNet) algorithm with transfer learning and a cubic spline model are modified and implemented on the TuSimple dataset for lane segmentation and interpolation. The detection performance and speed are comparable with other state-of-the-art algorithms.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisors Dr. Saeid Habibi and Dr. Ryan Ahmed. Dr. Saeid Habibi provided me with an opportunity to pursue my Master's degree in the autonomous driving research field. This journey opened a whole new world for me to automotive AI technology. His technical support and patient guidance were essential to the completion of this thesis. I also appreciate Dr. Ryan Ahmed for his valuable advice from modern industries. His contributions of time and ideas made my Master experience more productive.

My sincere thanks also go to Cam Fisher, Zeina Tawakol, Nicole McLean, and my colleagues. Without their cooperation in the CMHT lab or remotely, my overseas studies and work would not be motivated and smooth as it is now.

Last but not the least, I would like to thank my families and T.Q for their endless love and support during this difficult time of the COVID-19 pandemic.

Contents

List of Figures

List of Tables

Declaration

I hereby declare that I am the sole author of this thesis work, with advice and guidance provided by supervisors Dr. Saeid Habibi and Dr. Ryan Ahmed. This is a true copy of the thesis, including any required final revisions, as accepted by the examiners. And I understand that my thesis may be made electronically available to the public by McMaster University Libraries Institutional Repository: MacSphere.

Chapter 1

Introduction

This chapter briefly introduces the background of autonomous vehicles. The research motivation and objective are described, followed by sensors setup on the lab vehicle. Finally, the thesis contributions in seven chapters are outlined.

1.1 Background Overview

A fully realized autonomous vehicle is a vehicle capable of sensing the surrounding environment and operating without human involvement. This concept has been under active research and development since the 1980s and has experienced accelerated growth in both academia and industry worldwide. One of the key considerations for the autonomous driving technology is to reduce car accidents in daily life by developing a vehicle that can drive by itself in all environments. The reason is that around 94% of traffic accidents are due to human errors [1]. After decades of development, this technology not only could potentially prevent road accidents but also aid in reducing pollutant emissions. Moreover, it can render driving time more useful and improve traffic mobility. Furthermore, new industries and opportunities are emerging such as logistics and shipping, and notably Mobility as a Service (MaaS). About $800 billion per annum in social benefits are expected by 2050 if driverless cars can be manufactured and applied to markets [1].

The autonomous driving system blueprint is shown in Figure 1.1.1. In terms of connectivity, self-driving cars can be ego-only systems or connected systems. An ego-only system is an "ego-car" that refers to an independent autonomous vehicle. A connected system is also called "vehicle to everything" (V2X), including Vehicle to Vehicle (V2V), Vehicle to Infrastructure (V2I), and Vehicle to Device (V2D). Due to the complexity of system design and security concerns, no connected systems are practical so far. Thus, ego-only systems are the mainstream application. In terms of algorithm design, two types of design are as follows: modular systems mainly consist of perception, localization and mapping, planning and decision making, and vehicle control; end-to-end driving systems work from perception to vehicle control directly. End-to-end approaches have not yet been fully implemented due to safety concerns.

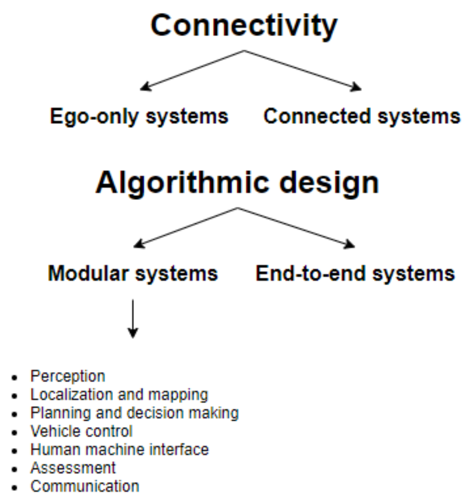


Figure 1.1.1: Autonomous driving system architecture [1]

1.2 Research Motivation and Objective

In the autonomous driving system architecture, the indispensable component is the perception which works as the first stage of autonomous driving. It is intended to act as a human's sense, and to collect and process diverse information from the surrounding environment by using different types of sensors. The accuracy of perception significantly affects the performance of vehicle operations. For example, one Tesla crash in 2016 was caused by false recognition of a white truck being classified as sky in perception. Commonly used sensors for perception are cameras, Light Detection And Ranging (LiDARs), and radars.

Optical cameras collect colorful information and are passive sensors that do not actively rely on emitted signals and use ambient light without interfering with other sensors. With advanced computer vision technologies, optical cameras are the most commonly used devices in autonomous vehicles with the smallest size and lowest cost. However, in monocular implementations, they have no depth information and are affected by weather and illumination conditions. Other types of cameras could be used for complementary capabilities, for instance, infrared (IR) cameras can detect pedestrians in low light condition.

As further complements to cameras, LiDARs utilize laser emission and reflection to generate cloud points of surrounding objects with depth information. They are robust to poor illumination conditions but are affected by severe weather conditions. They have high accuracy but large size and high cost. Their size and price have been decreasing in recent years, making them tangible sensors for autonomous operability. Radars employ radio waves to capture depth information and are stable under different lighting and weather conditions. They have lower accuracy than LiDARs.

With the evolution of Artificial Intelligence (AI) technology, deep learning models have been widely used in computer vision and natural language processing fields to generate outputs with high accuracy and fast speed. In addition, transfer learning technique could improve deep learning algorithms performance by sharing related information from a pre-trained model in one application domain to another application.

Accordingly, the objective of this thesis is to investigate if camera-based deep learning algorithms with transfer learning technique can achieve high performance on three perception tasks in real time:

(i) 2D object detection

(ii) license plate detection and recognition

(iii) highway lane detection

1.3 Lab Vehicle Sensors Setup

Sensors for perception research are installed and operated on a lab vehicle in the Centre for Mechatronics and Hybrid Technologies (CMHT) which focuses on advanced automotive technology in its research on autonomous driving, connected vehicles, electrification, and shared mobility solutions.

On the CMHT lab vehicle, as shown in Figure 1.3.1, optical cameras (CalmCar), IR camera (FLIR A65), LiDAR (Velodyne HDL-32E), and radar (Delphi ESR) are installed to work independently and corporately by sensor fusion. For this thesis research, only the monocular camera (highlighted by a red box in the figure) inside the front windshield will be used for image data collection.

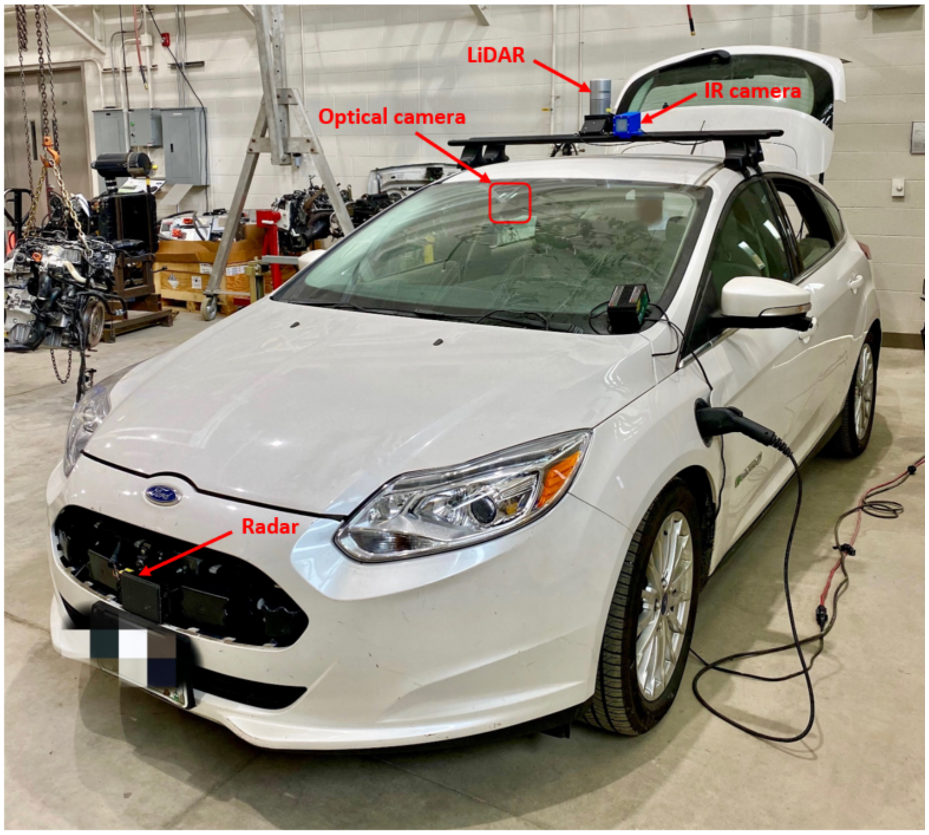


Figure 1.3.1: Sensors on CMHT lab vehicle

1.4 Thesis Contributions and Outline

Further to the background overview of autonomous driving that is presented in this Chapter, a more detailed discussion of research literature is provided in Chapter 2. Chapter 2 reviews autonomous driving, perception system, and camera types. Chapter 3 provides a literature review of deep learning neural network basics, and three perception applications: image-based object detection, License Plate Detection and Recognition (LPDR), and lane detection. The You Only Look Once version 3

(YOLOv3) algorithm is applied on the Karlsruhe Institute of Technology and Toyota Technological Institute (KITTI) dataset for the object detection in Chapter 4. The mechanism of YOLOv3 and the data mining of KITTI are explained in details. The detection results of Keras and PyTorch version YOLOv3 are evaluated and compared. Multiple experiments are designed to investigate the impacts of transfer learning and severe object occlusion/truncation on the detection performance. In Chapter 5, images are filtered from multiple datasets for the LPDR research. PyTorch YOLOv3 is used for detection and three models are explored for recognition: the Tesseract Optical Character Recognition (OCR) engine, the Convolutional Recurrent Neural Network (CRNN) with transfer learning, and the attention OCR without transfer learning. The CRNN method achieves the best performance so multiple experiments are then designed to optimize the algorithm. License Plates (LPs) are collected by a monocular camera on the CMHT lab vehicle and used for Ontario LP recognition testing via the optimized model. The Efficient Residual Factorized ConvNet (ERFNet) algorithm with transfer learning and a cubic spline model are analyzed and optimized for highway lane detection on TuSimple dataset in Chapter 6. The results are discussed and compared with other state-of-the-art algorithms. Finally, Chapter 7 concludes the research efforts in this thesis and discusses some future work.

Chapter 2

Autonomous Vehicle Perception and Cameras

This chapter provides a literature review in three aspects: autonomous driving, perception hardware, and camera types used in autonomous vehicles.

2.1 Autonomous Driving

The Society of Automotive Engineers (SAE) generated six levels of driving automation: level 0 (no automation), level 1 (driver assistance), level 2 (partial automation), level 3 (conditional automation), level 4 (high automation), and level 5 (fully autonomous). The current technology has achieved level 3. For example, Honda launched the world's first certified level 3 self-driving technology in 2021.

The benefits of autonomous vehicles include reduction in traffic accidents, transportation costs via ride sharing, and urban CO_2 emissions due to less traffic congestion; increase in human activity and parking space. On the other hand, the challenges consist of artificial intelligence technology development, weather and lighting conditions, traffic laws in different regions, internet security (malicious hacking and customers' information leakage) and software failure.

2.1.1 Systems of Autonomous Vehicles

According to the review paper from Badue et al. [2], a self-driving car has two main components: the perception system and the decision-making system. The former is to perceive the surrounding environment with multiple sensors and estimate the state of the car and its surrounding. The latter is to navigate and control the vehicle's motion. The perception system includes localization, obstacle mapping, road mapping, dynamic objects tracking, traffic sign detection and recognition. The decision-making system focuses on route planning, motion planning, and control.

The function of each task in two systems are briefly summarized. In perception, localization is used to estimate the position and orientation of the ego-car; obstacle mapping is to produce a map that represents the surrounding environment and can provide guidance for driving in free spaces; road mapping provides information of roads and lanes; dynamic objects tracking is implemented to track moving things to avoid collisions; traffic sign detection and recognition are able to detect and recognize human-defined traffic signs in order to follow traffic laws. In decision making, route planning is designed to compute an optimized path between start and end points; motion planning is applied to predict path or trajectory from the current to next states of vehicles to understand and decide the driving actions; and control is to control and actuate vehicles automatically.

2.1.2 Computer Vision in Autonomous Vehicles

One important reason why autonomous vehicles have difficulty in achieving level 5 autonomous driving is that most existing computer vision technologies are not very accurate and are prone to errors in perception. Robust perception from visual information is required to reach human-level reliability. Janai et al. [3] provide an overview of computer vision researches in autonomous driving. Four applied fields of computer vision techniques are briefly introduced here: object detection, semantic segmentation, reconstruction, motion and pose estimation.

Object detection is to classify and localize objects in the environment, which is necessary to avoid accidents. The challenges include the variety of objects' appearance, occlusion and truncation, distant objects, weather and illumination conditions. Current computer vision researches involve 2D object detection from images by optical or thermal cameras; 3D object detection from 3D point clouds by LiDARs, or from 2D images of cameras by extruding 2D predicted bounding boxes to 3D boxes via depth estimation, or from both LiDAR and camera data via sensor fusion.

Semantic segmentation is to classify each pixel into a class, which is essential for autonomous vehicles to understand the surroundings, such as identifying lanes from background. The challenges come from the scene's complexity, the number and size of segments, small or occluded objects, varying weather and illumination conditions. Object segmentation classifies individual object in pixel level so that information of object pose and shape can be collected. Road segmentation segments roads or lanes so that drivable space can be recognized and vehicles could be kept between lanes.

Reconstruction reconstructs 3D models from 2D images to understand the scene better, and it also benefits the map building of the environment. Stereo reconstruction technology uses stereo cameras to extract 3D information from 2D images for the depth estimation. However, the technology could fail in shiny and reflective regions. Multi-view 3D reconstruction technologies are applied for complete 3D scene reconstruction through images captured from multiple (more than two) viewpoints.

Motion and pose estimation have many research directions as follows. Optical flow leverages the 2D motion of brightness patterns between two images from monocular cameras to estimate objects' motions. It is affected by illumination conditions and occlusions. Scene flow integrates optical flow and depth information to estimate 3D motions using stereo cameras. Ego-motion estimation is to estimate the motion and pose of the car itself by cameras or LiDARs. The estimation has difficulties in rush hours or large turns of vehicles. Simultaneous Localization And Mapping (SLAM) simultaneously estimates the location of vehicle and builds the surrounding map. The main challenge is the large-scale environment mapping in real time.

2.1.3 Datasets for Autonomous Vehicles

The development of self-driving vehicles needs extensive data collected on roads, therefore, datasets with ground truth annotations perform a key role for algorithms' training and evaluation. Due to the extreme time consumption of data collection and pixel-level labeling of images, some synthetic datasets are created, but they may lack information compared to the real-world data. Some real-world datasets for autonomous vehicles have been produced and are summarized below.

For objection detection and semantic segmentation, ImageNet [4], PASCAL Visual Object Classes (VOC) [5], Microsoft Common Objects in Context (COCO) [6], and Cityscapes [7] are popular datasets. For 3D reconstruction, the Middlebury stereo benchmark [8], the Middlebury Multi-View Stereo (MVS) benchmark [9] and the Technical University of Denmark (TUD) MVS dataset [10] are commonly applied. For motion estimation, the Middlebury flow benchmark [11] and a dataset with 160 diverse real-world sequences of scenes [12] are built for optical flow algorithms. For tracking, the Performance Evaluation of Tracking and Surveillance (PETS) [13] and Multiple Object Tracking (MOT) [14] benchmarks are common evaluation tools. The Karlsruhe Institute of Technology and Toyota Technological Institute (KITTI) Vision Benchmark [15] is created for all of the above tasks to develop challenging real world computer vision benchmarks for autonomous driving. In addition, Yin and Berger [16] present an overview of 27 existing driving datasets in terms of generation time, size, traffic condition, used sensors, data format, and provided information.

2.2 Perception

As the first stage of intelligence vehicles, the perception is required to perceive and understand the surrounding environment in real time accurately. The most important activities in perception are vehicle detection and tracking, road and lane detection, traffic sign recognition, and scene understanding. Current challenges are due to computer vision technical level, real-time requirements, and environment complexity, such as severe occlusion of objects, diversities of lane and road appearances, weather and lighting conditions.

2.2.1 Sensors

The sensors in autonomous vehicles consist of internal and external types. Internal sensors measure information of the car itself such as engine temperature and battery charge. External sensors are also called exteroceptive sensors and measure items of interest that surround the car such as lanes, vehicles and pedestrians. Rosique et al. [17] have completed a detailed review of sensors. External sensors will be discussed in this section in terms of their properties and applications.

Cameras are commonly classified into visible, infrared (IR), and Time-of-Flight (ToF) categories. Monocular cameras are the most common and the cheapest optical cameras with single lens and produce 2D color images. Stereo cameras use two lenses to capture depth information and generate 3D images. Omnidirectional cameras can provide panoramic views to maximize information about the surrounding environment. Aforementioned visible cameras, are highly affected by lighting, rain, snow, or fog conditions. Thermal cameras measure the infrared energy of objects and create images using infrared radiation. They are not affected by weather or lighting conditions. ToF cameras are active sensors and employ time-of-flight techniques to estimate objects' depth by measuring Near-Infrared (NIR) light signals' travel time.

LiDARs emit laser waves to compute the distance to objects. A 2D LiDAR with a single laser beam only collects 2D data; a 3D LiDAR with multiple laser beams can receive 3D data. Moreover, 3D LiDARs operate with 360 degrees Horizontal Field of View (HFoV) and 20-45 degrees Vertical Field of View (VFoV), and they can be used in object detection and 3D mapping. Unlike traditional scanning LiDARs, solid-state LiDARs are built entirely on a silicon chip with less cost and do not require moving parts, so they improve the robustness to vibration. Also, in order to change directions, traditional LiDARs have to be physically moved, re-calibrated, and re-mounted back onto vehicles; whereas solid-state LiDARs' directional focus can be adjusted via a technique called optical phased array. The phased array is a row of transmitters that change directions of laser beams by adjusting relative phases of signals between transmitters. A 2D optical phased array can be designed by passing the emitted light along one axis through a grating array and guiding the light in different directions along another axis [18]. LiDARs are robust to lighting conditions, and less affected by various weather conditions. However, typical LiDARs cannot capture the objects' subtle textures, and their resolutions become sparse with distant objects. Flash LiDARs can produce detailed object information by illuminating the entire field of view with a diverging laser beam in a single pulse. Furthermore, Frequency Modulated Continuous Wave (FMCW) LiDARs can estimate the velocity of objects by splitting lasers into two portions, only sending one portion to target objects, and measuring the frequency difference between returned portion and retained portion.

Radars emit radio waves to calculate the range, angle, or velocity of objects by the Doppler effect. They have multiple applications: Blind Spot Detection (BSD), Lane Change Assistant (LCA), Rear Cross Traffic Alert (RCTA), Forward Cross Traffic Alert (FCTA). Radars work well in all weather conditions; but are limited by their lack of accuracy due to low resolution, restricted FoV, and errors due to bouncing of signals. Ultrasonic sensors are used to measure short distances to obstacles, and commonly found in parking assist systems. They have low cost and are robust in dusty or humid environments. However, they produce errors by having blind zones. Global Positioning Systems (GPS) provide the absolute location information of vehicles using the Global Navigation Satellite System (GNSS) network and are applied in localization and path planning.

Camera, 3D LiDAR, radar, and ultrasonic are evaluated according to FoV, operation range, accuracy, frame rate, resolution, color perception, sensor size, weather affections, maintenance, visibility, price by Rosique et al. [17] in Table 2.2.2.1. The qualification scores 0-3 refer to none, low, medium, and high. According to the table, different types of sensors can be selected depending on customized applications.

Table 2.2.2.1: Summary of sensors' qualification [17]

	Ultrasonic	Radar	3D LiDAR		Camera		
			Rotating	Solid state	VIS	IR	ToF
FoV	1	2	3	2	3	3	2
Range	1	3	3	3	2	3	2
Accuracy	1	2	3	3	3	2	2
Frame rate	2	2	2	2	2	3	3
Resolution	1	1	2	2	3	1	1
Colour perception	0	0	1	2	3	1	1
Size	1	1	2	1	1	1	1
Weather affections	1	1	2	2	3	1	3
Maintenance	2	1	2	1	2	2	2
Visibility	2	1	3	2	2	2	2
Price	1	2	3	1	1	3	2

2.2.2 Computing Devices

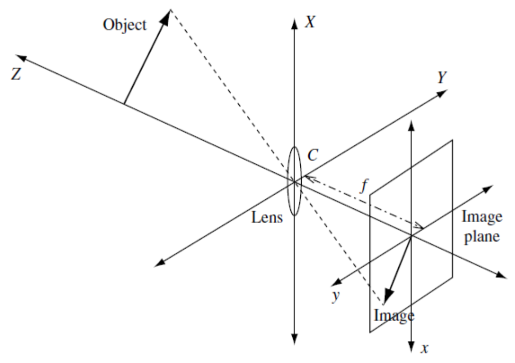
In addition to perception sensors, real-time computing platforms are also critically important. Graphics Processing Units (GPUs) are designed for graphical processing tasks such as the NVIDIA GTX 1080 Ti and RTX 2080 Ti. Also, several parallel programming platforms such as Compute Unified Device Architecture (CUDA) [19] and Open Computing Language (OpenCL) [20] have been developed for GPUs. A Field-Programmable Gate Array (FPGA) [21] is an electronic circuit that builds reconfigurable digital circuits. It is more energy-efficient than CPUs or GPUs. Moreover, Tensor processing units (TPUs) by Google [22] is specially designed for deep learning algorithms.

2.3 Camera Types in Autonomous Vehicles

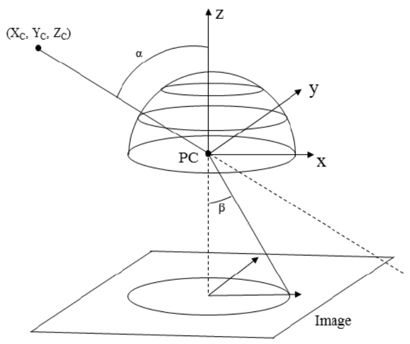
Cameras have diverse applications in each stage of autonomous driving. The Charge-Coupled Device (CCD) [23] and the Complementary Metal Oxide Semiconductor (CMOS) [23] are current dominant technologies for image sensing. In this section, the mechanisms and utilization of these camera technologies are explained.

2.3.1 2D Cameras

Monocular cameras imitate one eye in human vision and collect data at high resolution via a single image sensor. They are the most mature sensor technology with the lowest cost and smallest size, compared to other exteroceptive sensors. Two frequently used models are perspective and fisheye lens cameras. Perspective cameras convert spatial points from objects onto a 2D plane through perspective projection (Figure 2.3.1.1a). Fisheye lens cameras are designed to increase the field of view by deflecting light and distorting images (Figure 2.3.1.1b). Monocular cameras are commonly implemented in computer vision applications.



(a) Perspective camera projection [24]



(b) Fisheye camera projection [25]

Figure 2.3.1.1: Monocular camera models

IR cameras detect the infrared radiation of objects and focus the infrared energy onto a senor chip that contains pixels, and convert energy to electronic signal at each pixel. After computation, a color map of object temperature is generated as a thermal image. The main types are cooled and uncooled infrared detectors. Cooled IR cameras use a cryogenically-cooled system to cool to 0 degrees C or lower, in order to hide their own infrared emissions. Uncooled IR cameras have no cooling unit and operate at ambient temperature. Cooled IR cameras are more sensitive to temperature and produce higher image quality, but are more expensive and heavier than uncooled IR cameras. As a complementation to optical cameras, IR cameras can better detect pedestrians and animals in darkness or fog weather.

Omnidirectional cameras create a 360 degree Field of View (FoV) in the horizontal plane. Some types of omnidirectional cameras are described as follows: dioptric cameras [26] combine different shape of lenses; catadioptric cameras [26] use one standard camera with different mirrors (e.g. parabolic, hyperbolic) for the reflection of light; polydioptric cameras [26] overlap FoV from several cameras to generate panoramic images. Omnidirectional cameras can be used for traffic monitoring around an ego-car and mapping.

2.3.2 3D Cameras

3D cameras can measure the depth of objects and generate 3D information about the surrounding environment. Applications include 3D object detection, motion tracking, 3D reconstruction, Simultaneous Localization And Mapping (SLAM).

Stereo cameras act like a pair of human eyes with two separated image sensors. This type of cameras calculates the depth of an object by finding the disparity (pixel location difference) between the left and right cameras' images for the same point. According to Figure 2.3.2.1, O_l and O_r are optical centers of the two cameras, P_l and P_r are target points in two images, T is the baseline of stereo vision system, and f is the focal length of the cameras. Then, $\frac{x_l}{f} = \frac{X}{Z}$ and $\frac{x_r}{f} = \frac{X-T}{Z}$, and thus, $Z\ (depth) = \frac{f \times T}{x_l - x_r} = \frac{f \times T}{d}$ where $d\ (disparity) = x_l - x_r$.

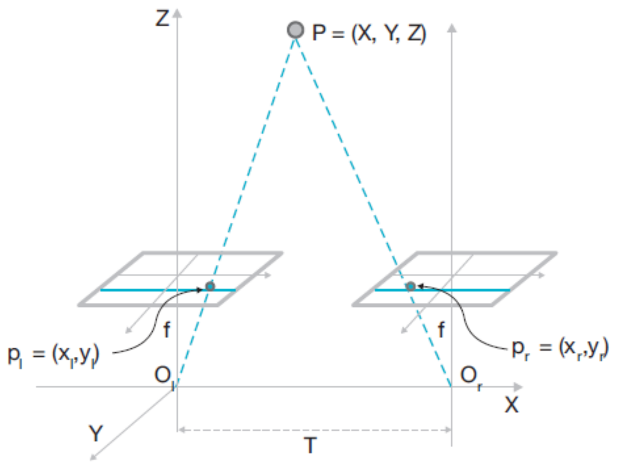
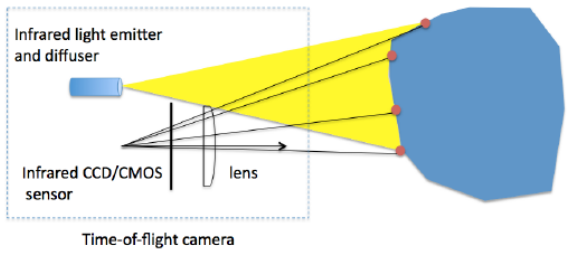
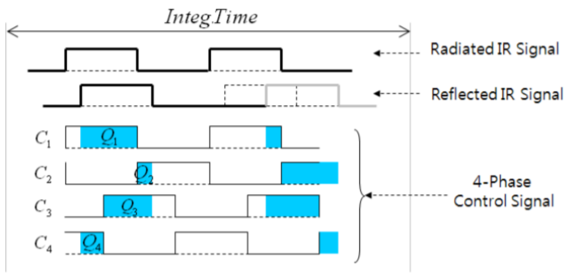


Figure 2.3.2.1: Depth of stereo vision [27]

Optical Time of Flight (ToF) methods are divided into direct ToF (dToF) and indirect ToF (iToF). The dToF measures time directly, such as LiDAR (pulsed light). The iToF computes time indirectly by measuring the phase shift between emitted and reflected signals, such as Continuous-Wave modulated light ToF (CW-ToF) camera in Figure 2.3.2.2a. For the depth calculation of iToF, as shown in Figure 2.3.2.2b, $d\ (distance) = \frac{c \times t}{2 \times \pi} \times arctan(\frac{q_3 - q_4}{q_1 - q_2})$, where c refers to the speed of light, t refers to the length of the signal, and q_1 to q_4 are the amount of electric charge for each phase.



(a) CW-ToF camera [28]



(b) Depth measurement [29]

Figure 2.3.2.2: ToF camera and depth estimation

A RGB-D ("D" refers to "depth" or "distance") camera uses structured light technology to predict the depth of objects. It consists of one RGB camera, one IR projector, and one IR camera. For example, in Figure 2.3.2.3, the RGB camera (camera 1) is responsible for collecting color images, the IR emitter projects a predefined pattern of light, and the IR camera (camera 2, also called depth camera) receives data of the deformed pattern on the surface of the object. Depth is computed via the disparity between patterns before and after deformation.

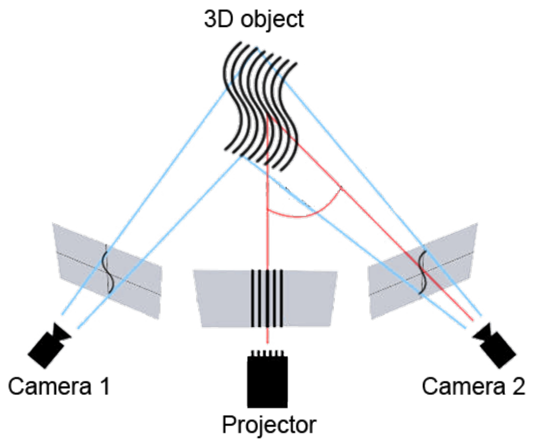


Figure 2.3.2.3: RGB-D camera configuration [30]

2.3.3 Event Cameras

Different from standard frame-based cameras, event cameras (also called neuromorphic vision sensors) are frameless and operate each pixel independently and asynchronously. They only record pixel-level brightness changes as sparse "events" (ON or OFF) due to movement and do not store intensity information. One event has three outputs: the pixel coordinate (x, y), the timestamp of event t, and the polarity of event p (+ for brightness increase, - for brightness decrease). Event cameras do not suffer from motion blur, latency, and low dynamic range. Figure 2.3.3.1 illustrates that when a disk rotates with high speed, a frame-based camera has motion blur issue, but an event-based camera works well. There is increasing research on the use of event-based vision [31] for object detection and tracking, object and motion segmentation, optical flow estimation, 3D reconstruction, and SLAM.

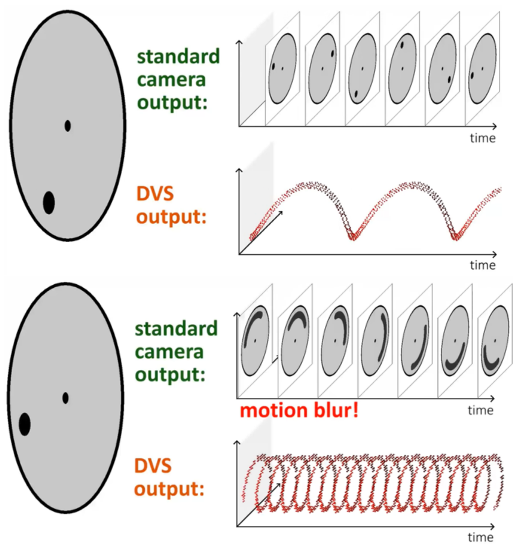


Figure 2.3.3.1: Disk slow and fast rotation captured by standard and event cameras [32]

Chapter 3

Deep Learning and Perception Applications

This chapter provides a literature review of deep learning neural network basics, and three perception applications: image-based object detection algorithms, license plate detection and recognition methods, and lane detection approaches.

3.1 Deep Learning Neural Network Basics

Artificial Intelligence (AI) is a research field that enables computer systems to mimic human intelligence processes. Machine learning is a subfield of AI that automates data analysis via various computer algorithms. Deep learning is a method of machine learning that extracts information from inputs via deep neural networks. With the rapid development of deep learning, diverse neural network types are emerging. In this research, three types of neural networks are reviewed: Artificial Neural Network

(ANN), Convolution Neural Network (CNN), and Recurrent Neural Network (RNN). ANN mimics the biological neural networks in human brains. CNN and RNN are more complex neural networks and extensively used in Computer Vision (CV) and Natural Language Processing (NLP) fields, respectively. In this thesis, CNNs are used in all projects, and RNNs are applied in the license plate recognition research.

3.1.1 Artificial Neural Network (ANN)

As Figure 3.1.1.1 is shown, an ANN consists of one input layer, multiple hidden layers, and one output layer. Any layer between input and output layers is known as a hidden layer. ANN is also referred to as Feed-Forward Neural Network (FFNN), because of the way information processes through the network from its input to its output layer in a sequential forward direction. Inside each layer, the circle unit is called a neuron. A common strategy in ANN (referred to as fully-connected or dense) is to connect a neuron in a current layer to all neurons in the next layer.

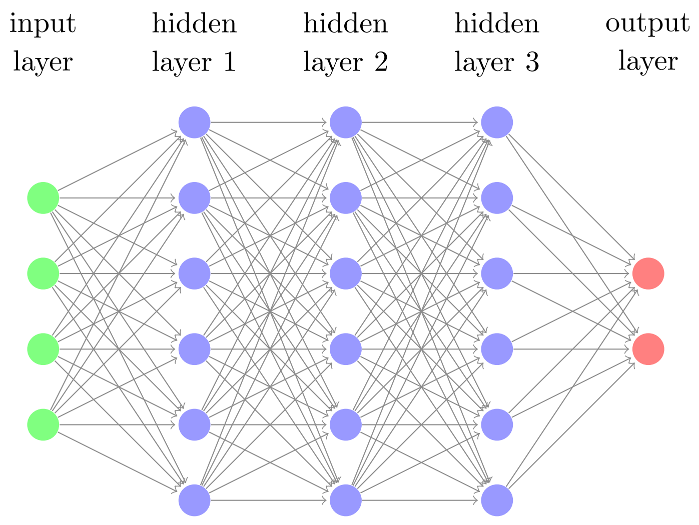


Figure 3.1.1.1: Deep artificial neural network

Figure 3.1.1.2 represents the structure of a single neuron. It works as a computational function that combines the sum of all weighted inputs and a bias term: $\sum_i w_i x_i + b$. Inside the function, x, w, and b denote to input signal, weight, and bias. The weight parameter w controls the influence of an input signal on the output, and the bias constant b is applied to offset the neuron computation result. An activation function f is utilised to control the transformation of the neuron information before being passed to the next layer.

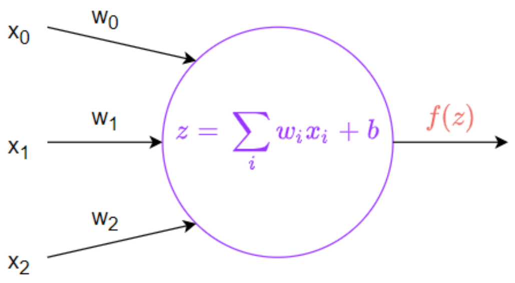


Figure 3.1.1.2: Neuron mechanism

3.1.2 Convolution Neural Network (CNN)

ANN works well in tabular and text data, but for image data, it has a limited capability and effectiveness. Use of ANN in image processing requires high dimensionality as the intensity of each pixel in an image is used as an input to the network. In addition, ANN loses the spatial relation in adjacent pixels after flatting of an image to a column of pixels as per the above. To maintain the spatial structure, CNN is designed to process the image data using convolutional layers (denoted by CONV), pooling layers (denoted by POOL), and fully connected layers (denoted by FC).

In a CONV, as an example, Figure 3.1.2.1 shows how a 3x3 kernel (also named filter or window) is used to scan a 6x6 input image and convolution operations are applied to generate a 4x4 output which is called feature map (also named activation map). The kernel starts at the top left of the input and moves one step each time from left-to-right and top-to-bottom.

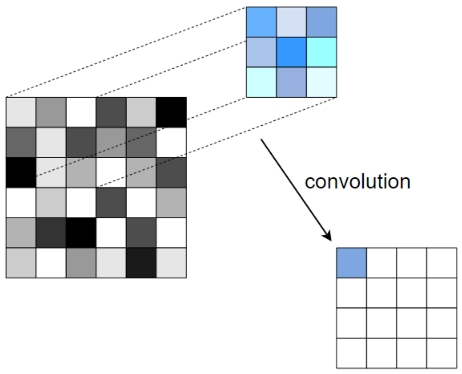
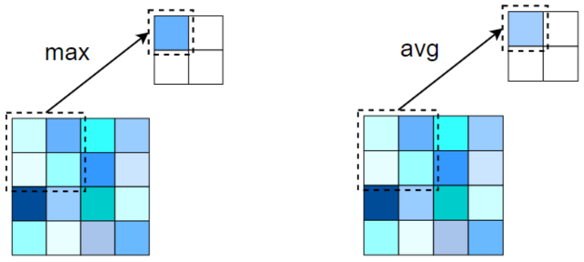


Figure 3.1.2.1: Convolutional layer

The POOL is designed to compress the spatial information and down-sample the size of feature maps. It is typically applied after a CONV. The commonly used types are max pooling and average pooling. For the max pooling in Figure 3.1.2.2a, a 2x2 filter (dashed box) scans a 4x4 feature map and the max value from the region covered by the filter is selected as an output. The ultimately generated 2x2 feature map contains the most prominent features from the previous 4x4 feature map. The process is the same for average pooling, except that the average value is calculated.



(a) Max pooling (b) Average pooling

Figure 3.1.2.2: Pooling layer

The FC is an added structure in CNN depending on applications, and it is usually implemented in the end of CNN as output layer. In Figure 3.1.2.3, a 2x2 feature map is flatten to a 4x1 layer and all neurons are fully connected with next layer neurons.

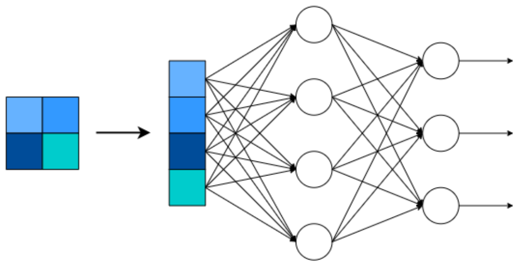


Figure 3.1.2.3: Fully connected layer

3.1.3 Recurrent Neural Network (RNN)

Both ANN and CNN have fixed input and output lengths, and they do not memorize previous processed information. However, sequential data (e.g. sentences) has arbitrary lengths, and output prediction needs historical information. Thus, RNN is designed to have memory and to solve this problem by feeding previous information back into a model via recurrent connections.

RNN processes sequential data in a time series. In Figure 3.1.3.1, the expression to the left of the equal sign represents a RNN in time step t. The input X_t is fed into the RNN as a vector by encoding the input text using a word embedding algorithm such as Word2Vec [33]. The generated hidden state h_t is a "memory" of the RNN and is sent back into the RNN as an additional input for next time step operation. The h_t is equal to $f(UX_t + Wh_{t-1} + b)$. U is a shared weight for input X across all time steps, and W is also a shared weight for hidden state h. Besides, b is a bias term. And the function f is an activation function.

By unfolding the expression to the right of the equal sign in the figure, it clearly demonstrates that the hidden states h_0, h_1, h_2, ..., h_{t-1} are passed to the RNN at time steps 1, 2, 3, ..., t as previous information to predict future information.

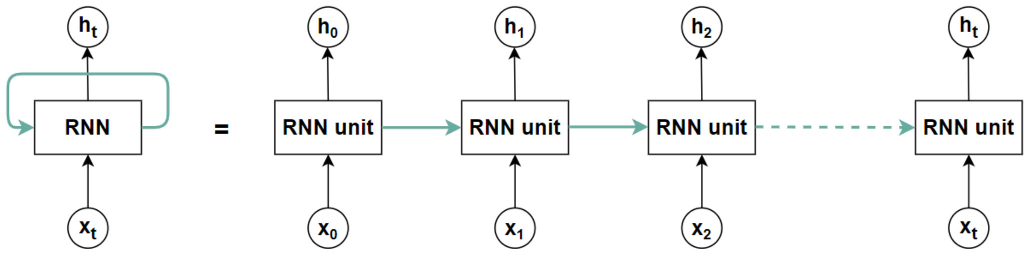


Figure 3.1.3.1: RNN expansion

3.1.4 Activation Functions

As mentioned in ANN, a neuron cell computation $z = \sum_i w_i x_i + b$ is sent as an input to an activation function $f(z)$ to reshape the neuron output. Four commonly used activation functions are introduced here.

The first one is the sigmoid function [34] $sigmoid(z) = \frac{1}{1+e^{-z}}$ and it is also called the logistic function. As shown in Figure 3.1.4.1, it returns values in the range of 0 to 1, so the function is often used for binary classification. The second one is the softmax function [35] $softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$, $i = 1, ..., K$. K in the function refers to the number of classes in a data. The softmax function normalizes the output to the range of 0 to 1, and all $softmax(z_i)$ sum to 1. Thus, it is widely used for multi-class classification. The third one is the tanh function [36] $tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$ that outputs values in the range of -1 to 1. The function is also plotted in the figure below. Many RNNs use the tanh function because it can generate both positive and negative outputs that could increase or decrease values of hidden states in the

]

process of sequential information propagation. The last one is the Rectified Linear Unit (ReLU) function [37] $ReLU(z) = max(0, z)$. In Figure 3.1.4.1, ReLU outputs zero when z is negative and z itself when z is positive. It is simple and fast for matrix computation operations, and commonly used in both CNN and RNN applications.

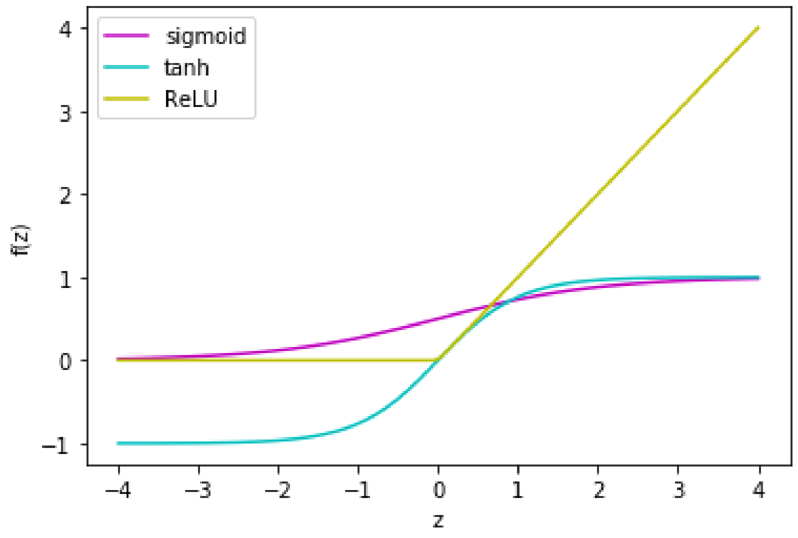


Figure 3.1.4.1: Activation functions

3.2 Image-based Object Detection

Image-based object detection is applied for 2D object classification and localization. Traditional models first search regions which potentially contain objects using sliding windows, and then extract features by hand-engineered feature descriptors such as Histogram of Oriented Gradients (HOG) [38]. They finally distinguish objects by classifiers, such as Supported Vector Machines (SVM) [39] or Deformable Part-based Model (DPM) [40]. Traditional methods are inefficient, time-consuming, and have limited learning capacity. Today, with the significant evolution of deep learning algorithms, robust deep object detection networks have been widely implemented which can learn complex features more efficiently. The overall architecture of object detectors includes: input (images), backbone (network for feature extraction), neck (optional part for feature map collection from different stages), and head (class and bounding box predictions for objects). The latest detectors are categorized into two groups according to the type of head: anchor-based and anchor-free as follows.

3.2.1 Anchor-based Detectors

Anchors (also named priors) are boxes with predefined widths and heights based on the sizes of objects in a dataset. Instead of randomly predicting the dimensions of bounding boxes, anchors can be used as references to accelerate bounding boxes regression. Anchor-based detectors dominate the object detection field, and produce the top detection performances in common benchmarks. They overlay a large amount of predefined anchor boxes with varying aspect ratios, scales, and spatial locations on an image. The classes of objects are predicted and the anchors with the highest scores are selected for the final bounding boxes prediction around objects.

Two-stage algorithms are region proposal based, the first stage is to identify the Regions of Interest (RoIs), and the second stage is to classify and localize objects. A RoI is a region in an image that may contain an object and it is represented as a rectangular box. One-stage algorithms are regression based, they avoid using RoIs and map features directly for object classification and localization. Two-stage detectors achieve better precision due to multiple times anchors' refinement. However, they result in longer inference time and more complex architectures. One-stage detectors refine anchors once in order to enable real-time and more straightforward structures. In order to be competitive in accuracy as two-stage detectors, one-stage detectors rely on more dense anchor boxes in an image.

The following two-stage algorithms are introduced: Region-based Convolutional Neural Network (R-CNN) family and Region-based Fully Convolutional Network (R-FCN, 2016). R-CNN family includes R-CNN (2014), fast R-CNN (2015), and faster R-CNN (2015). Anchors were first used in faster R-CNN, but R-CNN and fast R-CNN will be mentioned as the family members. Besides, mask R-CNN (2017) was invented for instance segmentation and mesh R-CNN (2019) was proposed for 3D object mesh generation from a 2D image. They are out of the object detection scope and will not be reviewed here. The following one-stage algorithms are discussed: You Only Look Once (YOLO) family, Single Shot MultiBox Detector (SSD, 2016), and RetinaNet (2017). YOLO family contains YOLO (2016), YOLOv2 (2017), and YOLOv3 (2018). YOLO does not use anchors, but it will be explained in the YOLO family.

R-CNN [41] creates RoIs using the selective search method [42] and sends each RoI to individual CNN for object classification and bounding box regression. N RoIs would need N CNNs for feature extractions. Fast R-CNN [43] accelerates the image processing speed significantly by generating RoIs and feature maps in parallel via the selective search method and one CNN, then projecting RoI boxes onto feature maps and utilizing the projected regions of feature maps for object detection. Compared with fast R-CNN, faster R-CNN [44] replaces the selective search method by a CNN named Region Proposal Network (RPN) for RoIs generation. The RPN takes feature maps from the first CNN as inputs and uses predefined anchors to predict RoIs. R-FCN [45] improves faster R-CNN by replacing fully connected layers with fully convolutional layers in the architecture to retain the spatial information.

YOLO [46] is one of the earliest real-time algorithms. It divides an input image into S x S (S = 7) grids and each grid cell predicts one object by generating bounding box coordinates, object existence confidence, and class probabilities. The architecture of YOLO is a variant of GoogleLeNet [47] and contains 24 convolutional layers followed by 2 fully connected layers. YOLO has high localization errors because it initially random guesses bounding boxes in the training process. In addition, YOLO has poor performance in small object detection and it fails to detect crowded objects because of the rule that one grid cell focuses on one object.

SSD [48] starts to use manually designed anchors to improve object localization and leverage multi-scale feature maps to improve small object detection. In a CNN, convolutional layers in a forward pass reduce spatial dimension and resolution of feature maps. YOLO detects objects using the final layer feature map (Figure 3.2.1.1b: single feature map), the small spatial dimension and low resolution of the feature map result in poor performance in small object detection. SSD predicts objects using multiple layers' feature maps (Figure 3.2.1.1c: pyramidal feature hierarchy) and this strategy can detect small objects better. For the algorithm architecture, SSD uses

VGG-16 [49] as the backbone for feature extraction and multiple convolutional layers as the head for object classification and localization.

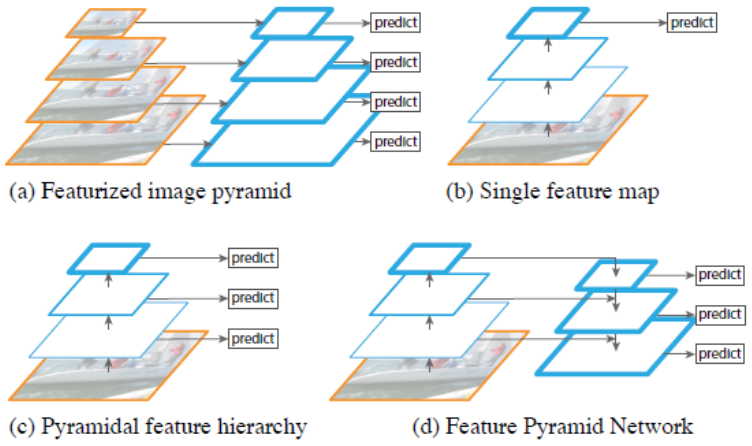
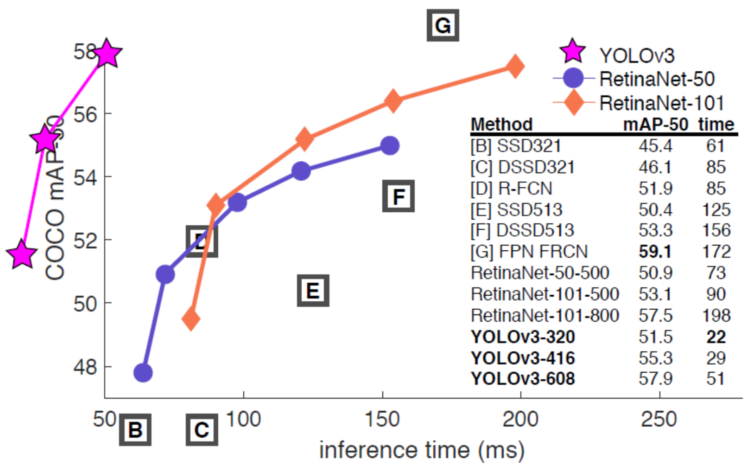


Figure 3.2.1.1: Feature pyramids models [50]

YOLOv2 [51] is the second version of YOLO and improves both object detection accuracy and prediction speed. In terms of accuracy improvement, some methods are adopted as follows: the batch normalization method normalizes outputs of activation functions per layer in the network to stabilize the training process; images with higher resolution are used for the network training; 5 anchors are predefined for object localization and their shapes are decided by the K-means clustering technique; the algorithm head consists of convolutional layers rather than dense layers; similar to the pyramidal feature hierarchy method in SSD, YOLOv2 applies a method named passthrough that reshapes an earlier layer's feature map and concatenates it with the final layer's feature map to achieve multi-scale object detection; multi-scale training is attempted by randomly choosing input image size every ten batches and resizing the training network. In terms of speed improvement, YOLOv2 customizes a light-weighted backbone named DarkNet-19 that is composed of 19 convolutional layers and it reduces the floating point operations when running the neural network.

RetinaNet [52] has two key innovations: Featurized Pyramid Network (FPN) and focal loss. The FPN (Figure 3.2.1.1d) is a backbone to detect objects at different scales, which is more accurate than the pyramidal feature hierarchy in SSD. Its structure combines low-resolution and semantically strong features with high-resolution and semantically weak features via bottom-up and top-down pathways and lateral connections, thus, it has rich semantics at all levels. The focal loss is a type of loss functions that compute errors between predicted and true object classes and locations. It is designed to solve the issue of extreme class imbalance by adding more weights for classes which have a small number of objects or are hard to be detected, and assigning less weights for classes which have more objects or are easier to be detected. RetinaNet achieves higher accuracy but lower speed than YOLOv2.

YOLOv3 [53] has some updates based on YOLOv2 and more details will be described in Chapter 4. The main changes are as follows: 9 anchors are used rather than 5 anchors; the backbone replaces DarkNet-19 by DarkNet-53 that consists of 53 convolutional layers; due to the deep neural network performance degradation problem, ResNet-alike structure is added into the backbone to improve its performance; FPN-alike structure is applied into the head for three-scale object detection (small, medium, large). Overall, YOLOv3 performs better than previous versions and SSD, similar to RetinaNet but several times faster, as shown in Figure 3.2.1.2.



Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

Figure 3.2.1.2: Algorithms performance in COCO dataset [53]

3.2.2 Anchor-free Detectors

Anchor boxes have multiple disadvantages. First, all predefined anchors are used to predict bounding box for an object, but only one anchor is finally responsible for the bounding box optimization and generation, so the process decelerates the training speed. Next, anchors' design may result in numerous parameters that take a long time and much complexity for training. Furthermore, anchors' aspect ratios and sizes depend on datasets, so they have to be re-designed if datasets are different.

Anchor-free detectors directly detect objects without predefined anchor boxes. They predict bounding boxes around objects using keypoints or centers. Keypoints are points that can determine a box, such as two corners of a box. Centers refer to the centers of objects. The keypoint-based methods detect objects using keypoints and then predict the boundary of objects by grouped points. The center-based methods predict the center of an object and the width and height of a bounding box, or the distance from the center to four sides of a bounding box. So far, most anchor-free detectors are one-stage algorithms and some of them are briefly introduced below.

CornerNet [54] detects an object as a pair of keypoints (the top-left corner and bottom-right corner), then groups them to form the final predicted bounding box. CenterNet [55] improves the precision and recall of CornerNet by detecting each object as a triplet (top-left, center, bottom-right). YOLO is a center-based method, it predicts centers of objects in the grid cells and widths and heights of bounding boxes. Fully Convolutional One-Stage (FCOS) [56] predicts centers of objects and four distances (center-to-left, center-to-right, center-to-top, center-to-bottom) to boundaries of bounding boxes. Overall, most current anchor-free detectors may not achieve real-time inference speed (no speed information mentioned in papers).

3.2.3 Datasets and Evaluation Metrics

For generic object detection, four popular datasets are summarized: PASCAL VOC [5], ImageNet [4], MS COCO [6] and Open Images [57]. Liu et al. [58] compared these datasets as reproduced in Table 3.2.3.1.

PASCAL VOC is a benchmark dataset for object detection with standardized evaluation metric and annual competitions. Its object categories include aeroplane, bicycle, bird, boat, bottle, bus, car, cat, chair, cow, dining table, dog, horse, motorbike, person, potted plant, sheep, sofa, train, TV/monitor. ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is derived from ImageNet and is much more extensive than PASCAL VOC in terms of the number of images and objects. ImageNet1000, a subset of ImageNet with 1.2 million images and 1000 object categories, provides a benchmark for the ILSVRC image classification challenge. MS COCO contains complex scenes in a natural environment, and objects are labeled as instance segmentation to provide more accurate evaluation. The COCO object detection challenge includes two tasks: object detection using bounding box or pixel-level instance segmentation. The Open Image Challenge Object Detection (OICOD) is derived from Open Images, and is one of the largest publicly available object detection datasets. Its annotation contains bounding boxes and segmentation masks.

Table 3.2.3.1: Dataset examples for object detection [58]

Name	Total images	Categories	Images per category	Objects per image	Image size	Started year
PASCAL VOC	11,540	20	303-4087	2.4	470x380	2005
ImageNet	14 millions+	21,841	-	1.5	500x400	2009
MS COCO	328,000+	91	-	7.3	640x680	2014
Open Images	9 millions+	6000+	-	8.3	varied	2017

Three frequently used criteria are precision, recall, and F1 score. One commonly used metric is Average Precision (AP) for each class of objects. Mean Average Precision (mAP) is computed to represent the performance over all object categories.

All parameters related to the object detection evaluation are explained below:

- Intersection over Union (IoU): a number that measures the overlap of a predicted and ground truth box (Figure 3.2.3.1), $IoU = \frac{Area\ of\ Overlap}{Area\ of\ Union}$.
- Confidence threshold β: a number that indicates the confidence that an object is detected.
- True Positive (TP): correct detection ($IoU \geq threshold$).
- True Negative (TN): correct identification of non-object regions.
- False Positive (FP): miss-classification or incorrect localization ($IoU \leq threshold$).
- False Negative (FN): missed detection.

- Precision: the ratio of correct detection and all predictions, $precision = \frac{TP}{TP + FP}$.
- Recall: the ratio of correct detection and all ground truths, $recall = \frac{TP}{TP + FN}$.
- F1 score: the harmonic mean of the precision and recall in order to balance them, $F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$.
- Average Precision (AP): the area under the precision-recall curve, $AP = \int_0^1 p(r)dr$.
- mean Average Precision (mAP): the average of all APs of objects' classes.

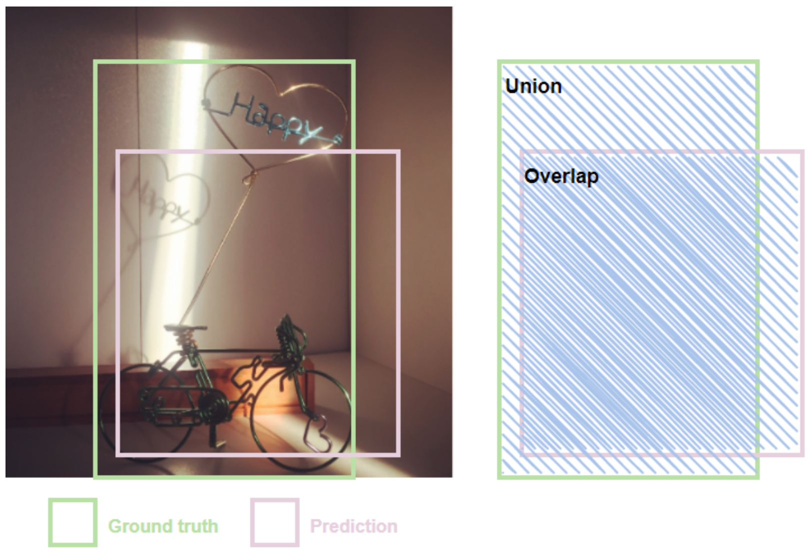


Figure 3.2.3.1: IoU

Precision and recall are sensitive to the confidence threshold and IoU. Confidence threshold β determines how likely a bounding box contains an object. Predicted boxes with scores below the value of β will be ignored, otherwise they will be kept for evaluation later. The selection of β is a trade-off between false positive and false negative. IoU measures how well the predicted box matches the ground truth box. The remaining predicted boxes with IoU values beyond the threshold are set as true positive, otherwise they are false positive.

3.2.4 Deep Learning Libraries and Transfer Learning

TensorFlow was the first and most used open-source library for deep learning, but its low-level API is difficult for users to create deep learning models directly. In recent years, Keras and PyTorch libraries have become more popular due to their simpler usage interfaces than TensorFlow.

Keras is a high-level API on top of TensorFlow, Theano, and CNTK. It is the most user-friendly framework to learn and operate for beginners. PyTorch stands between Keras and TensorFlow. It has more flexibility and control than Keras, and it has less complicated programming than TensorFlow. For building architectures of algorithms, Keras is more accessible to setup because it directly provides sequential and functional API to define neural network layers; functions have to be defined by users in PyTorch classes, but the overall structure is cleaner and more elegant. For training, a model is simply trained using "model.fit()" with built-in parameters in Keras; but multiple steps need to be implemented in PyTorch: the gradients initialization, the forward pass, the backward pass, the loss computation, and the weights update. For computing speed, Keras is written in python and supports CPU by default, it could enable GPU if Tensorflow-GPU is installed; PyTorch is written in python, C++, CUDA and supports GPU for every tensor and NumPy (stands for Numerical Python, a python library worked for arrays and matrices) variable, so PyTorch is faster than Keras when running code.

In deep learning neural networks, the beginning layers extract general features such as edges and the final layers extract specific features. It would be more beneficial and efficient if the information of general features from one application can be shared with another that is different but in a related application. Therefore, transfer learning technique is proposed that at first trains a network with a dataset from one application and saves the pre-trained weights, then transfers the learned features to another application by training the same algorithm with another dataset and using the pre-trained weights as the initial weights. Transfer learning works well if the transferred features have high similarity between the base and target datasets.

Network training benefits from transfer learning when the dataset size is not large, the overfitting issue and training time can be significantly reduced. How to best use transfer learning depends on the target data size and the similarity between base and target datasets. According to Table 3.2.4.1, if a large target data is similar to the base data, all network layers are fine-tuned; if a large target data is different from the base data, the model is trained without transfer learning; if a small target data is similar to the base data, higher layers of the model are fine-tuned; if a small target data is different from the base data, lower layers of the network are fine-tuned.

Table 3.2.4.1: Transfer learning implementation

Data similarity $\longrightarrow$		
Data size $\uparrow$	Train model from scratch	Fine tune the whole pre-trained model
	Fine tune the lower layers	Fine tune the higher layers

3.3 License Plate Detection and Recognition

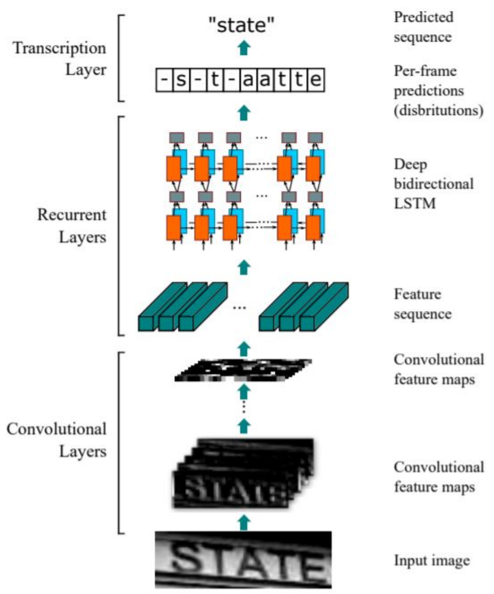
License Plate Detection and Recognition (LPDR) combines object detection technology for plate detection and Optical Character Recognition (OCR) technology for plate recognition. LPDR can be used for vehicle tracking in autonomous driving. Its challenges mainly come from two parts: plate and environment variations. Plates have different sizes, colors, fonts in different countries and regions; plates' characters could be standardized or customized with different length and formats; plates' qualities (clear or blur) are varying due to vehicles' motion, camera resolution, distance from the ego-car; plates can also be occluded, tilted, installed with frame and painted with other patterns. Environment challenges include bad weather (e.g. rainy, snowy) and illumination conditions (e.g. cloudy, cars' headlights).

3.3.1 Deep Learning Algorithms

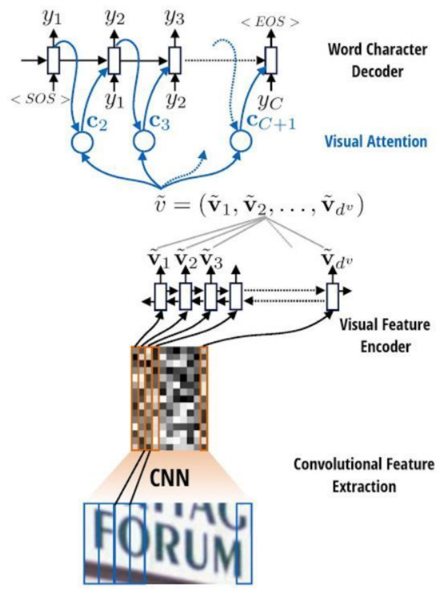
Deep learning methods contain two main stages: (i) License Plate (LP) detection; and (ii) recognition. The LP detection can be achieved using object detection algorithms, which have been briefly summarized in the previous section. Thus, only deep learning algorithms for image text recognition will be introduced here.

Optical Character Recognition (OCR) was primarily invented to scan and recognize text in images or printed documents using traditional computer vision techniques. It has developed maturely since the 20^{th} century, but is still challenging in an unconstrained environment, including illumination, different fonts, geometrical distortion, object motion in captured images. Therefore, deep learning algorithms are applied to this field to improve the unstructured text recognition.

Most algorithms are segmentation-free, because the performance of character segmentation is sensitive to illumination, shadows, complex background, and noise. Two famous text recognition algorithms are Convolutional Recurrent Neural Network (CRNN) with Connectionist Temporal Classification (CTC) [59] and attention OCR [60]. As shown in Figure 3.3.1.1a, in the architecture of CRNN with CTC, images are fed into a CNN to extract features, and then the feature sequence from the CNN is fed into a RNN to calculate the probability of each character class, finally CTC is used to decode and generate the output of text. In Figure 3.3.1.1b, attention OCR also has a CNN for feature extraction and RNN for feature encoding. Instead of using CTC for decoding, a visual attention model is applied as a decoder to predict the output. More details of these two algorithms will be explained in Chapter 5.



(a) CRNN architecture [59]


y_1 y_2 y_3 $< EOS >$
$< SOS >$ y_1 y_2 y_C
c_2 c_3 c_{C+1}
Word Character Decoder
Visual Attention
$\tilde{v} = (\tilde{v}_1, \tilde{v}_2, \ldots, \tilde{v}_{d^v})$
$\tilde{v}_1$ $\tilde{v}_2$ $\tilde{v}_3$ $\tilde{v}_{d^v}$
Visual Feature Encoder
CNN
Convolutional Feature Extraction
FORUM

(b) Attention OCR architecture [60]

Figure 3.3.1.1: Text recognition algorithms architectures

3.3.2 Datasets

Most publicly available LP datasets are collected from traffic monitoring (captured images from the front-top of cars), toll station (captured images from the rear-top of cars) or parking lots (captured images close to the stationary cars). Most of them have no annotations of LPs' locations (bounding boxes) on vehicles and numbers (text labeling). There are very limited published datasets for autonomous driving scenarios. One reason is that significant time is required to collect a large number of distinct LPs from moving cars on roads. Another reason is that some jurisdictions have privacy laws about gathering and using personal information including LP numbers.

Only a few datasets, whose LPs are applicable for autonomous driving research and consist of English letters and digits, are summarized here. Application Oriented License Plate (AOLP) [61] is a Taiwanese dataset that was created in 2013. It contains images for access control, road patrol, and traffic law enforcement applications. And images are collected in various locations/time/traffic/weather conditions. Only the traffic law enforcement subset is suitable to be used here and it includes total 757 images with various resolutions. Open Automatic License Plate Recognition (OpenALPR) [62] was generated in Europe, Brazil and United States in 2016. Only parts of US license plates are appropriate for the autonomous driving scenario, and it contains 222 images with various resolutions. Federal University of Paraná - Automatic License Plate Recognition (UFPR-ALPR) [63] is a Brazilian dataset that was published in 2018. All images are captured by moving cars with 1920 x 1080 resolution. It has a total of 150 images for buses, cars, or motorcycles at different locations.

3.4 Lane Detection

Lane detection is a computer vision research application in the Advanced Driver-Assistance Systems (ADAS) and autonomous driving. Lane markings are classified and their locations (coordinates) are inferred in the detection process. It is helpful in operations involving lane keeping, departure warning, lane changing, and trajectory planning decision. Real-time lane detection is vital for full autonomous driving. At present, cameras are the dominant type of sensors used for lane detection, because the visual cues from cameras are the same as the human eye's capture, and it has the lowest cost compared with other sensors. However, vision-based algorithms still have various challenges, related to low illumination, bad weather, variability of road surface conditions, shapes/types/quality of lane markings, shadows, and severe occlusion by driven vehicles.

3.4.1 Deep Learning Algorithms

Deep learning algorithms can learn high-level features, handle various conditions, or even work in real time. Most algorithms are in two stages, including lane segmentation and lane fitting. The fitting stage is primarily affected by the segmentation stage. Few methods are one-stage in order to achieve end-to-end learning. Two-stage algorithms are summarized into three categories: segmentation-based CNN, Generative Adversarial Network (GAN), a combination of CNN and RNN. The architectures of one-stage algorithms are mainly regression-based CNNs. Some popular models will be discussed below, some of them are designed for specific problems.

Segmentation is a computer vision research topic that partitions each image into multiple segments to understand the surrounding environment better. It is divided into two types: semantic segmentation and instance segmentation. Semantic segmentation groups pixels into defined object classes and marks objects in the same class using the same color. Instance segmentation segments objects individually and assigns a unique color for every single object. It is a combination of object detection and semantic segmentation. Both two types of segmentation have applications in lane detection research.

Segmentation-based CNN structures can be a fully convolutional network (e.g. Spatial CNN (SCNN) [64] and Vanishing Point Guided Network (VPGNet) [65]), an encoder-decoder (e.g. LaneNet [66]), and a base CNN with attention mechanism (e.g. Self Attention Distillation (SAD) [67]). To address the occlusion issue, SCNN reinforces the spatial information of scenes via inter layer propagation. It learns features by fully convolutional layers along four directions (downward, upward, rightward, leftward) in input images to generate semantic segmentation masks and then cubic splines are applied for lane fitting. VPGNet focuses on the challenge of weather scenes (i.e. rain and night). It predicts vanishing points of lanes so that these points can guide the convergence of lane patterns. Point sampling, clustering, and lane regression techniques are exploited in curve fitting for different lane types. In LaneNet, the encoder extracts high-level features from images and the decoder predicts lane instance segmentation. Subsequently, in order to alleviate the problem of road surface slope change, a second neural network named HNet is trained to dynamically estimate the transformation matrix coefficients for the "bird-eye view" of images and lanes are fitted via 3rd order polynomials on the ortho view of images before re-projecting them back to the original images. SAD adds activation-based attention maps into a base CNN model to refine the extracted features quality. It improves the semantic segmentation performance without increasing the inference time of lane detection. Finally, cubic splines are also used for lane fitting like SCNN.

GAN [68] contains a Generator (G) and a Discriminator (D). G learns to generate plausible data and deceive D. D learns to distinguish the fake data from real data (ground truth) and penalize the difference between them. In lane detection, GAN uses its generator to predict lanes, and its discriminator to evaluate the detection performance (e.g. Embedding Loss GAN (EL-GAN) [69]). EL-GAN is designed to improve the performance of conventional segmentation-based CNNs. CNNs classify lanes by calculating class probabilities for each single pixel independently in an image, thus, the detected lanes may not achieve high qualities of thinness, smoothness, and consistency. Instead, EL-GAN trains its network with embedding loss to minimize the difference between predicted lane segments and ground truth labels (thin polylines connected lane points). Its predicted lanes are thinner and smoother than that by segmentation-based CNNs. Presently, no published code for GAN algorithms can be found and used online.

CNN and RNN can work together as independent networks or can be fused as a whole architecture. Li et al. [70] pre-processed an input image as a sequence of image parts, next, a CNN was applied for feature extraction from each part, then a RNN was used to predict lanes from the features along time series. Most deep learning methods work on single images in their training process. The lack of information from single frames results in partial or false direction detection. Zou et al. [71] proposed an algorithm that trains one current image with multiple previous sequential images. The architecture is a “sandwich” structure: the encoder of a CNN extracts features from multiple continuous frames, next, the feature maps are fed into a RNN as a time-series sequence for analysis, and the output is fed into the decoder of the CNN for lane segmentation. No curve fitting models are implemented in these two algorithms.

Regression-based CNNs directly regress lanes at the end of network, such as CNN with regression [72] and differentiable least-squares fitting [73]. Most two-stage methods are based on semantic segmentation and are not effective because if the generated lane segments are fragmented, the lane fitting performance will be degraded. Thus, CNN with regression was proposed as an end-to-end network to improve lane detection's robustness. The network architecture has an encoder part for feature extraction followed by fully connected layer branches for lane points coordinates and lane classes prediction. However, there is no detailed description on how the outputs are generated in the algorithm and no open-source code has been provided online. The differentiable least-squares fitting algorithm generates weight maps for each lane via the Efficient Residual Factorized ConvNet (ERFNet) [74], then all weighted pixel coordinates are fed into a least-squares fitting model to output parameters (a, b, c) of the best interpolating curve (parabolic curve $y = ax^2 + bx + c$) per lane. An incomplete code was published for this model [73].

3.4.2 Datasets and Evaluation Metrics

Some small-scale lane detection datasets are published for applications such as the Cambridge-driving Labeled Video Database (CamVid) [75] with 367 training, 101 validation, and 233 testing images; Caltech Lanes Dataset [76] with total 1225 frames; and KITTI-road [77] with 289/290 training/testing images. However, they are not suitable for deep neural network learning. Here, three popular large-scale annotated datasets with evaluation metrics will be concisely summarized as follows: TuSimple

[78], CULane [64], and BDD100K [79]. In addition, unlike object detection annotation using bounding boxes, lanes are labeled as pixel coordinates at equally spaced intervals. Figure 3.4.2.1 visualizes labeled lane points by green color circles and equally spaced intervals by horizontal red color lines.

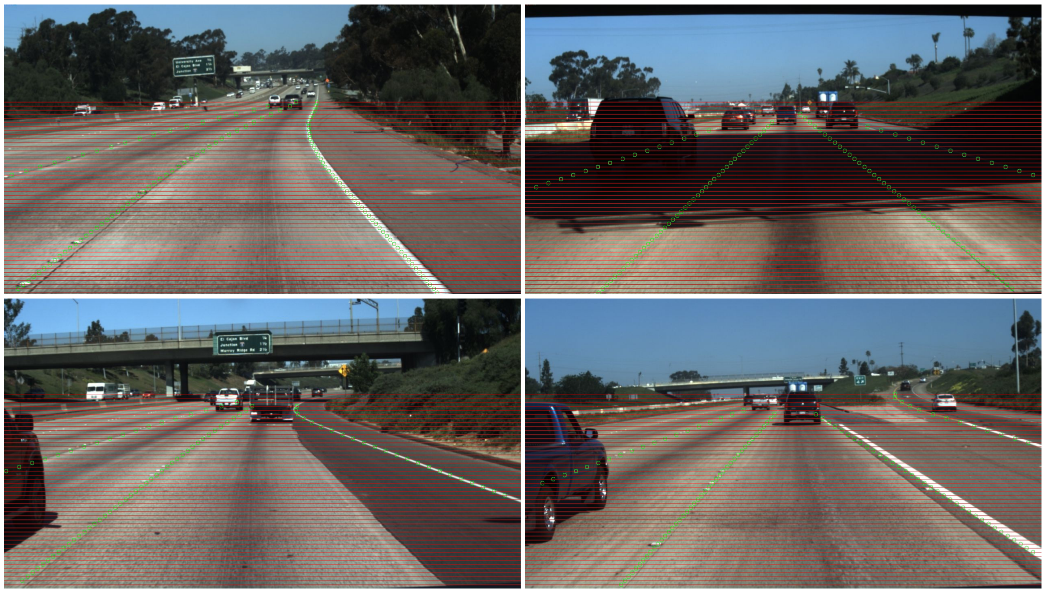


Figure 3.4.2.1: Lanes annotation [78]

TuSimple lane dataset has been collected on US highways in the San Diego area at different daytimes, consisting of 3626 training and 2782 testing images with 1280 x 720 resolution. There are no severe weather conditions and heavy occlusion. Most frames contain between 2 and 4 lanes, the rest of frames have 5 lanes. The evaluation metrics include accuracy, False Positive (FP) rate, and False Negative (FN) rate. Accuracy is the ratio of the number of correct points and the number of ground truth points per image ($acc = \sum_{im} \frac{C_{im}}{S_{im}}$). FP rate is equal to the number of wrongly predicted lanes divided by the total number of predicted lanes ($FP = \frac{F_{pred}}{N_{pred}}$). FN rate is the division of the number of missed lanes in prediction and the total number of ground truth lanes ($FN = \frac{M_{pred}}{N_{gt}}$).

CULane dataset has been collected by six vehicles driven in urban, rural, and highway areas of Beijing, China. It accommodates 88880 training, 9675 validation, and 34680 testing 1640 x 590 resolution images. It has nine environment scenarios: normal, crowded, night, no line, shadow, arrow, dazzle light, curve, and crossroad. There are no more than 4 lanes in each frame. It uses precision, recall, and F1 score as evaluation metrics. Under an assumption that lane width is 30 pixels (no explanation about the reason of lane width value setting in the paper), the Intersection-over-Union (IoU) between ground truth and prediction is calculated as the threshold. If IoU is equal or larger than 0.5, the predicted lane becomes True Positive (TP), otherwise, the prediction becomes False Positive (FP). A missed lane is viewed as False Negative (FN). Hence, precision $= \frac{TP}{TP+FP}$, recall $= \frac{TP}{TP+FN}$, and F1 $= 2 \times \frac{precision \times recall}{precision + recall}$.

BDD100K dataset was gathered from New York, Berkeley, San Francisco, Bay Area in the United States under various weather conditions and at different times. It labeled not only lane markings, but also road objects, drivable areas, and full-frame segmentation. Compared with previous two datasets' annotation, BDD100K is much more complex as it classifies lanes by solid or dashed, double or single, white, yellow, or other colors. It has a total of 100K images with 1280 x 720 resolution and 70K of the data is used for training. In this dataset, some images contain more than 5 lanes. BDD100K does not have its own lane detection benchmark.

Chapter 4

Object Detection

2D object detection in autonomous driving is used for classifying objects categories and localizing their positions in each video frame over time. Objects' information helps autonomous navigation to avoid traffic collisions. As reviewed in Chapter 3 Section 2.1, the top-performing deep learning algorithms include two-stage models (e.g. R-CNN family) and one-stage models (e.g. YOLO family). In comparing these two types of algorithms, one-stage models have lower accuracy but are faster in speed to achieve real-time inference than two-stage models. Therefore, one-stage methods are more applicable to the autonomous driving domain.

As one of the most prominent real-time object detection algorithms, YOLOv3 is implemented on the KITTI dataset for car, pedestrian, and cyclist classification and localization. Also, the results of two different algorithm versions based on Keras and PyTorch deep learning frameworks with NVIDIA GeForce RTX 2080 Ti GPU support are compared. Moreover, multiple experiments are designed to investigate the impacts of transfer learning technique and severe object occlusion/truncation condition on the object detection performance.

4.1 YOLOv3 Algorithm

YOLOv3 is the deep learning algorithm used in this 2D object detection project. In this section, the details of the original YOLOv3 architecture are explained, including the algorithm backbone for feature extraction and the algorithm head for object classification and localization. In addition, the loss function that measures the detection error between predictions and ground truths is discussed. Furthermore, how the final bounding box for an object is filtered out from all predictions is analyzed.

YOLOv3 is an anchor-based algorithm and designed for multi-scale object detection. It makes detection at three scales for small, medium, large objects. And each scale uses three bounding boxes in different shapes for prediction. Thus, YOLOv3 creates a total of 9 anchors as references using the K-mean clustering method and predicts bounding boxes for objects by adjusting anchors' attributes (box center, width, height) in the training process. As shown in Figure 4.1.1, the main components of YOLOv3 architecture are the feature extractor (grey region) and the feature detector (yellow, purple, blue regions). The extractor (also named backbone) has 75 layers and the detector (also named head) has 31 layers; hence, there are a total of 106 layers in the model.

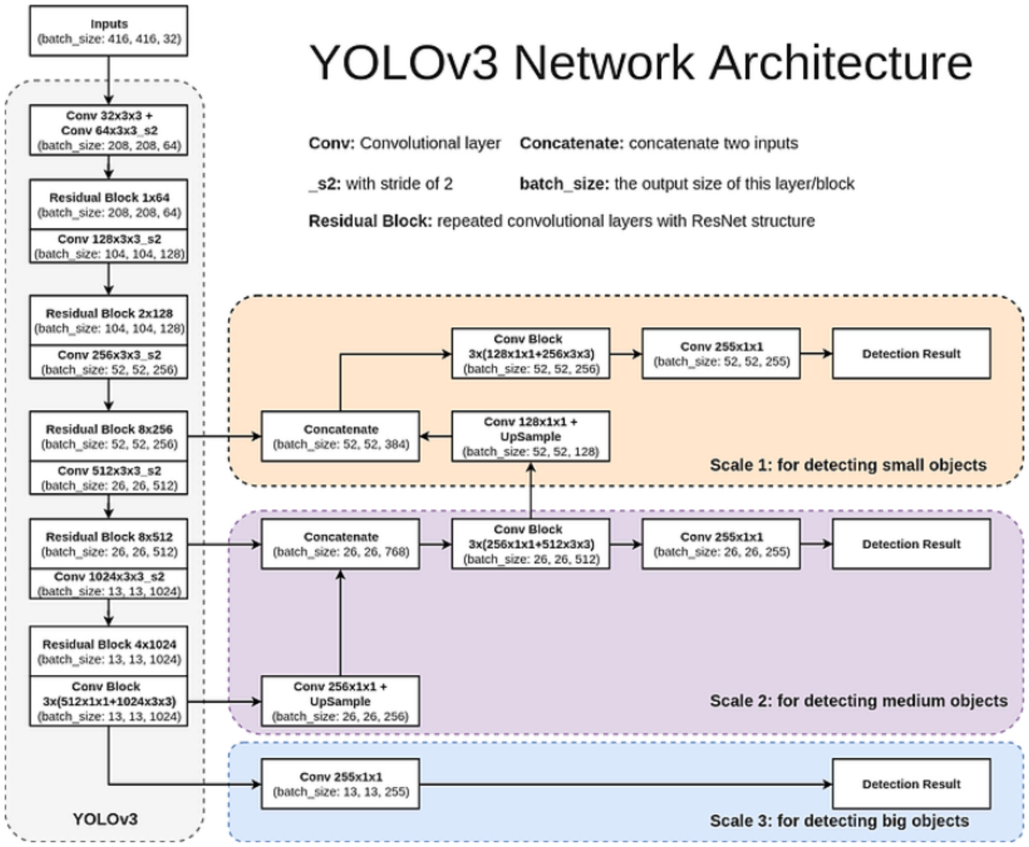
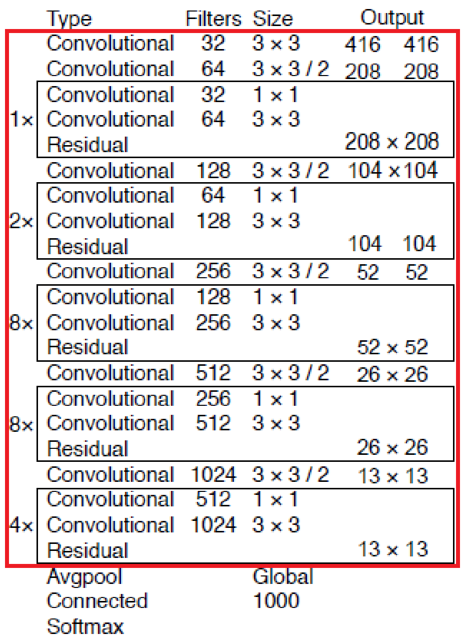

YOLOv3 Network Architecture
Conv: Convolutional layer
Concatenate: concatenate two inputs
_s2: with stride of 2
batch_size: the output size of this layer/block
Residual Block: repeated convolutional layers with ResNet structure
Inputs (batch_size: 416, 416, 32)
Conv 32x3x3 + Conv 64x3x3_s2 (batch_size: 208, 208, 64)
Residual Block 1x64 (batch_size: 208, 208, 64)
Conv 128x3x3_s2 (batch_size: 104, 104, 128)
Residual Block 2x128 (batch_size: 104, 104, 128)
Conv 256x3x3_s2 (batch_size: 52, 52, 256)
Residual Block 8x256 (batch_size: 52, 52, 256)
Conv 512x3x3_s2 (batch_size: 26, 26, 512)
Residual Block 8x512 (batch_size: 26, 26, 512)
Conv 1024x3x3_s2 (batch_size: 13, 13, 1024)
Residual Block 4x1024 (batch_size: 13, 13, 1024)
Conv Block 3x(512x1x1+1024x3x3) (batch_size: 13, 13, 1024)
YOLOv3
Conv Block 3x(128x1x1+256x3x3) (batch_size: 52, 52, 256)
Conv 255x1x1 (batch_size: 52, 52, 255)
Detection Result
Concatenate (batch_size: 52, 52, 384)
Conv 128x1x1 + UpSample (batch_size: 52, 52, 128)
Scale 1: for detecting small objects
Concatenate (batch_size: 26, 26, 768)
Conv Block 3x(256x1x1+512x3x3) (batch_size: 26, 26, 512)
Conv 255x1x1 (batch_size: 26, 26, 255)
Detection Result
Conv 256x1x1 + UpSample (batch_size: 26, 26, 256)
Scale 2: for detecting medium objects
Conv 255x1x1 (batch_size: 13, 13, 255)
Detection Result
Scale 3: for detecting big objects

Figure 4.1.1: YOLOv3 network architecture [80]

DarkNet-53 is the backbone of YOLOv3 and "53" refers to its 53 convolutional layers. Its architecture is shown in Figure 4.1.2. Except the final average pool layer, fully connected layer, and softmax layer, all rest layers (highlighted in the figure) are used in YOLOv3 to extract image features. The main components are convolutional blocks (only contains convolutional layers) and residual blocks (contains convolutional layers with residual connections). With more convolutional layers, the deeper neural network accuracy starts to degrade significantly. Thus, the residual connections are applied to solve the deeper network degradation problem. To illustrate its mechanism, a basic residual block is presented in Figure 4.1.3. A curve arrow connects the beginning and the end of a network. This residual connection (also named skip connection) output called identity mapping x, which is the same as the input of the network. After passing all layers, the output in the end of the network called residual mapping $\mathcal{F}(x)$. The network learns from both residual mapping $\mathcal{F}(x)$ and identity mapping x and the final output becomes $\mathcal{F}(x) + x$. If the network achieves its optimal result and continues to go deeper, it is hard to learn and the residual mapping may reduce to zero, then the identity mapping will maintain the performance of the deep network. In this way, DarkNet-53 can have much more layers to extract features than DarkNet-19 in YOLOv2.



	Type	Filters	Size	Output
	Convolutional	32	3 × 3	416 416
	Convolutional	64	3 × 3 / 2	208 208
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			208 × 208
	Convolutional	128	3 × 3 / 2	104 × 104
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			104 104
	Convolutional	256	3 × 3 / 2	52 52
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			52 × 52
	Convolutional	512	3 × 3 / 2	26 × 26
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			26 × 26
	Convolutional	1024	3 × 3 / 2	13 × 13
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			13 × 13
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 4.1.2: DarkNet-53 architecture [53]

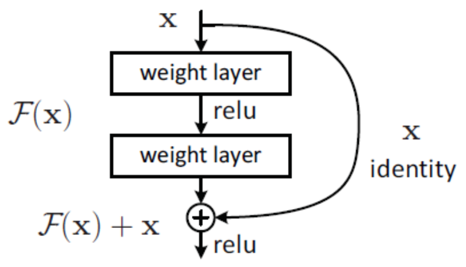


Figure 4.1.3: Residual block basic structure [81]

The multi-scale detector in YOLOv3 leverages an FPN-alike structure, as discussed in Chapter 3 Section 2.1, to detect objects at three scales (small, medium, big). According to Figure 4.1.1, feature maps from later layers are up-sampled and concatenated with feature maps from earlier layers in the backbone, next, they are fed into the detector for small and medium object detection. The concatenation operation is to stack (combine) feature maps with the same size. The purpose of feature map concatenation is as followed: feature maps become smaller but more detailed when layers are deeper, large feature maps can "see" small objects but small feature maps are hard to "see" them because small object features are lost in the down-sampling process; hence, feature map concatenation between shallower and deeper layers can improve small object detection performance. Each of the three detection branches in the detector outputs a tensor that contains information about object class and location. The tensor has a dimension N x N x [B x (4 + 1 + C)]. Each attribute in the tensor dimension is interpreted below and visualized in Figure 4.1.4:

- N x N is the amount of grid cells in one output feature map, i.e. feature map scale size of each detection branch output.
- B is the number of predicted bounding boxes per grid cell and is set as 3 in YOLOv3, namely, YOLOv3 generates 3 bounding boxes in each grid cell of a feature map for prediction.
- Dimension 4 refers to the bounding box offsets [t_x, t_y, t_w, t_h] against the corresponding anchor box: t_x and t_y are box centroid location offsets, t_w and t_h are box width and height offsets.
- Dimension 1 refers to the objectness score p_o, which infers how likely an object center is inside a grid.
- C is the number of class probabilities (p_1, p_2, ..., p_c), i.e. C classes in a dataset.

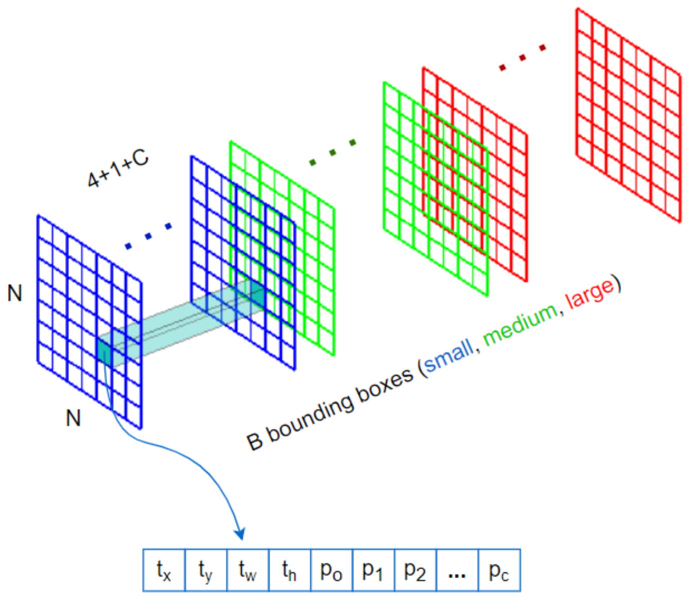

4+1+C
N
N
B bounding boxes (small, medium, large)
t_x t_y t_w t_h p_o p_1 p_2 ... p_c

Figure 4.1.4: YOLOv3 output attributes

As illustrated above, YOLOv3 detector output predicts three bounding boxes with three predefined anchors for each grid cell in a feature map, but finally only one bounding box with its anchor is selected for the object location prediction and box offset calculation. The strategy is that all bounding boxes with object confidence scores that are less than the confidence threshold are removed; then in the remaining bounding boxes, only one box, whose anchor has the largest IoU with the ground truth box, is retained.

In neural networks, the output errors called losses. A loss function (also named cost function or error function) is designed to quantify losses between predictions and ground truths. The loss function of YOLOv3 includes three main sub-functions: localization loss function, classification loss function, and objectness loss function. The localization loss function measures a bounding box center coordinates (x, y) loss, width and height loss. The classification loss function computes an object class prediction loss. The objectness loss function calculates an object existence prediction loss. Redmon et al. [53] only mentioned that the Squared Error (SE) loss function is used to calculate a box center (x, y) and width and height losses, and the Binary Cross-Entropy (BCE) loss function is applied to compute the objectness and classification losses. There is no loss function formula published in the paper. Afterward, diverse variants of the loss function of YOLOv3 are designed and written into programs to improve the algorithm detection performance.

The SE loss function calculates squared difference between predicted (i.e. offsets $[t_x, t_y, t_w, t_h]$ between predicted bounding box and anchor) and actual (i.e. offsets $[t_{x'}, t_{y'}, t_{w'}, t_{h'}]$ between ground truth and anchor) values. The BCE loss function, also named binary log loss function, is used for binary classification tasks (i.e. objectness: if a bounding box contains object or not; and, classification: if an object belongs to a class or not). Before using BCE, the network outputs would be passed into a sigmoid function to convert their values into probabilities in the range of 0 and 1. Then, the prediction probabilities are passed into the BCE loss function $BCEloss = -(y * log(x) + (1 - y) * log(1 - x))$ to compute losses between prediction x and ground truth y. The x refers to the predicted objectness probability p_o or class probabilities p_1, p_2, ..., p_c. Its value is between 0 and 1. The y refers to the true value of objectness or class. Its value is 0 or 1. The BCE loss curves for ground truth y at 0 and 1 are visualized in Figure 4.1.5. To conclude, the loss function is created to measure errors in object detection, and objects could be localized and classified accurately by optimizing the loss function (i.e. minimizing the loss value).

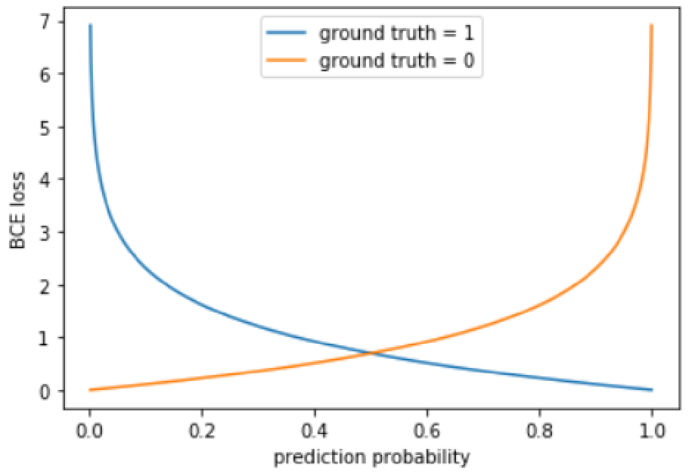


Figure 4.1.5: Binary Cross-Entropy loss curves

YOLOv3 prediction in each grid cell results in multiple detections on the same object in an image, and these bounding boxes are highly overlapped. Thus, the Non-Maximal Suppression (NMS) is used in inference to remove (suppress) redundant predictions per class according to the predicted boxes' overlapping conditions. The NMS is a method that selects a single entity out of many overlapping entities. Its basic logic is as follows: bounding boxes are retained if their objectness confidences are larger than the confidence threshold, so bounding boxes that are unlikely containing objects are removed; next, for each class (e.g. person), the box with the highest confidence is set as a reference box (e.g. Figure 4.1.6a); then, IoUs are computed between the reference box and all other boxes, and boxes with IoUs that are larger than the IoU threshold are removed (e.g. Figure 4.1.6b). All steps above filter out bounding boxes that have lower confidence and too much overlap. After that, the box with the second highest confidence is selected and the same steps are repeated (e.g. Figure 4.1.6c-d). The process will stop until all boxes are "scanned" and no more boxes can be suppressed. The NMS also has different variants in the present research field, here, the original NMS technique is applied in YOLOv3 algorithm.

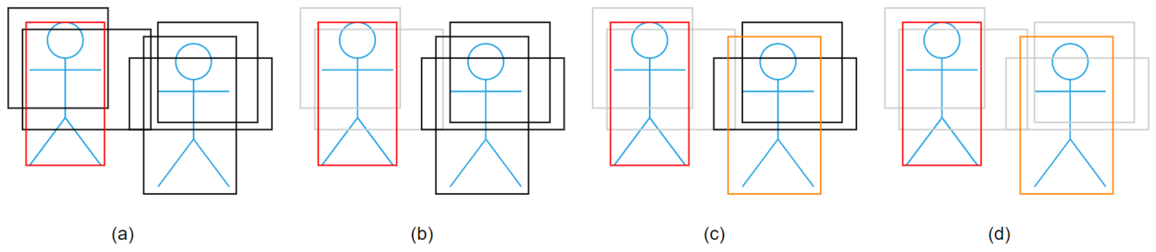


Figure 4.1.6: Non-Maximal Suppression example

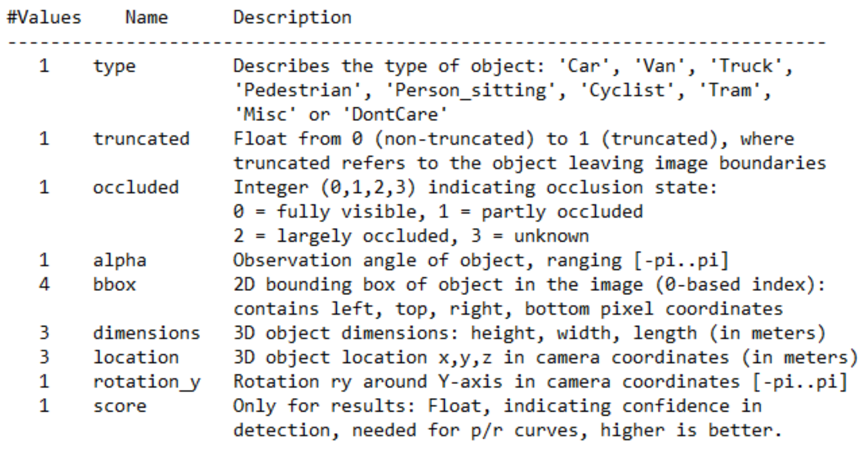
4.2 KITTI Data Mining

KITTI object detection dataset [15] is used for the YOLOv3 algorithm training and testing in this research. It was created in 2012 for autonomous driving research. Its 2D object data consists of 7481 training images and 7518 testing images with total 80256 labeled objects. However, only objects in the training images are annotated; thus, only the training data with labels are used here.

The object labelling has three levels of difficulties in terms of bounding box size (box size around an object), occlusion (an object is occluded by other things), and truncation (an object is only captured partially in an image). The KITTI Vision Benchmark Suite quantifies the difficulty levels as follows:

- Easy: the minimum bounding box height is 40 pixels, the maximum occlusion level is fully visible, and the maximum truncation is 15%.
- Moderate: the minimum bounding box height is 25 pixels, the maximum occlusion level is partly occluded, and the maximum truncation is 30%.
- Hard: the minimum bounding box height is 25 pixels, the maximum occlusion level is difficult to see, and the maximum truncation is 50%.

The labelling information description for each object is provided in Figure 4.2.1 and comprises of the following: class, truncation percentage, occlusion level, 2D bounding box coordinates, angle alpha, 3D object dimension, 3D object location, rotation (tilt forward/backward), and score. For 2D object detection, only the first four elements of annotation information will be used in this project. Other labels are created for object orientation estimation, 3D object detection, and bird's eye view. Class refers to object category and is used for object classification task; truncation percentage reflects how many percent of object body is not inside an image and occlusion level indicates the condition of object occlusion, these two labels are used to determine the detection difficulty level; and, 2D bounding box defines the top-left and bottom-right corners pixel coordinates in an image and it is used for object localization. Classes and 2D boxes were labeled by KITTI customized labelling tool, while the truncation percentage and occlusion level were labelled manually by people.



```
#Values    Name      Description
----------------------------------------------------------------------------
   1    type         Describes the type of object: 'Car', 'Van', 'Truck',
                     'Pedestrian', 'Person_sitting', 'Cyclist', 'Tram',
                     'Misc' or 'DontCare'
   1    truncated    Float from 0 (non-truncated) to 1 (truncated), where
                     truncated refers to the object leaving image boundaries
   1    occluded     Integer (0,1,2,3) indicating occlusion state:
                     0 = fully visible, 1 = partly occluded
                     2 = largely occluded, 3 = unknown
   1    alpha        Observation angle of object, ranging [-pi..pi]
   4    bbox         2D bounding box of object in the image (0-based index):
                     contains left, top, right, bottom pixel coordinates
   3    dimensions   3D object dimensions: height, width, length (in meters)
   3    location     3D object location x,y,z in camera coordinates (in meters)
   1    rotation_y   Rotation ry around Y-axis in camera coordinates [-pi..pi]
   1    score        Only for results: Float, indicating confidence in
                     detection, needed for p/r curves, higher is better.
```

Figure 4.2.1: KITTI data format description [15]

KITTI data has 8 classes of objects: car, van, truck, pedestrian, person sitting, cyclist, tram, and DontCare. KITTI designed a benchmark that only evaluates detectors' performance on car, pedestrian, and cyclist detection. Therefore, in this project, only car, pedestrian, and cyclist are considered, and car/van/truck are combined as car, pedestrian/person sitting are combined as pedestrian, and tram/DontCare are ignored. Finally, the KITTI classes are consolidated into three categories as car, pedestrian, and cyclist.

In addition, due to much more cars than other classes' objects, images that only have cars inside are removed to reduce the unbalance issue. After that, around 2500 images are retained. 90% of data is allocated for training and validation, and 10% of data is assigned for testing. In addition, the training and validation data is split using a ratio of 80% : 20%.

4.3 Experiments and Evaluation

In this section, the YOLOv3 algorithm variants that are written using Keras library [82] and PyTorch library [83] are implemented and their mechanisms are discussed. For each YOLOv3 version, multiple experiments are designed to investigate how the transfer learning technique, severe object occlusion and truncation conditions affect the object detection performance. In terms of the transfer training experiments, YOLOv3 are trained with and without transfer training. In terms of the severe object occlusion and truncation experiments, YOLOv3 are trained using KITTI data with and without its hard level. Finally, all experiments' testing results are evaluated and compared. Furthermore, PASCAL VOC criteria [5] is selected for KITTI 2D object detection performance evaluation. It contains the following metrics: precision, recall, F1, F2, Average Precision (AP) per class, and mean Average Precision (mAP) for all classes with IoU threshold at 0.5. Their definitions have been discussed in Chapter 3 Section 2.3.

4.3.1 Keras YOLOv3

Keras YOLOv3 is attempted first and three experiments are designed as follows. When discussing the training process of Keras YOLOv3, the related parameters of these experiments are mentioned.

(i) Training without transfer learning using KITTI data with its hard level

(ii) Training with transfer learning using KITTI data with its hard level

(iii) Training with transfer learning using KITTI data without its hard level

As an anchor-based algorithm, YOLOv3 needs to predefine nine anchors before training. All anchor widths and heights are computed for the KITTI dataset using the K-means clustering. The K-means clustering is a method that identifies k number of centroids in a data and partitions each data point to its nearest cluster. By using this method, all labelled object bounding boxes in KITTI can be partitioned into nine clusters and the centroids (the mean of clusters) become YOLOv3 anchors. The anchors generation flow chart is illustrated in Figure 4.3.1.1.

At the beginning, object bounding box widths and heights are computed using the labelled top-left and bottom-right pixel coordinates in KITTI data. A box ($width$, $height$) can be regarded as a 2D point. In all data points, nine points are randomly picked as initial centroids for clusters. Next, for each data point, distances between the point and centroids are calculated and the point is assigned to its nearest centroid's cluster. A distance is computed as 1 minus IoU between a bounding box and a centroid bounding box. Afterwards, each centroid is updated as the mean per cluster. The above steps are iterated until centroids have no change. The final centroids [($width_1$, $height_1$), ..., ($width_9$, $height_9$)] are defined as anchors for YOLOv3.

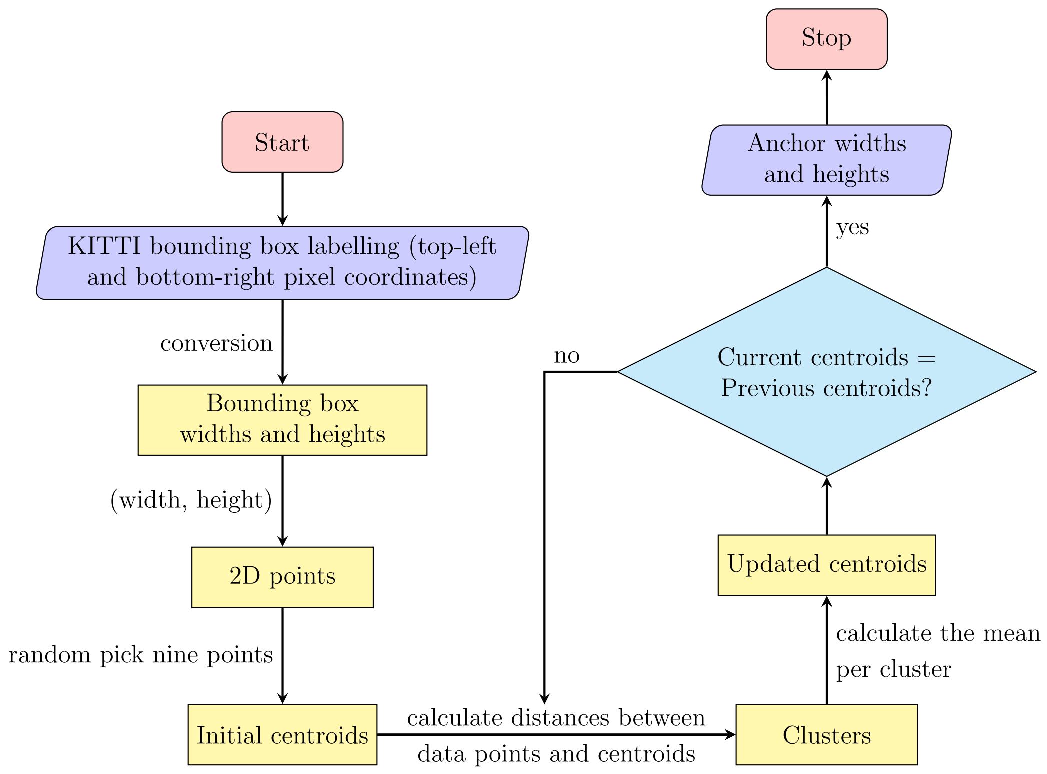

Start
KITTI bounding box labelling (top-left and bottom-right pixel coordinates)
conversion
Bounding box widths and heights
(width, height)
2D points
random pick nine points
Initial centroids
calculate distances between data points and centroids
Clusters
calculate the mean per cluster
Updated centroids
Current centroids = Previous centroids?
no
yes
Anchor widths and heights
Stop

Figure 4.3.1.1: Flow chart of anchors generation

One example of anchors generation iterations is created for visualization in Figure 4.3.1.2. The horizontal axis represents the labeled bounding box width, and the vertical axis represents the labeled bounding box height. The top, middle, bottom sub-figures are plotted for the initial, middle, and final iteration of centroids generation. In each sub-figure, the major blue points are KITTI data and nine colorful points are the centroids of clusters. For KITTI data without its hard level, the generated anchors (*width*, *height*) are [(14, 34), (23, 60), (33, 25), (39, 93), (59, 38), (75, 163), (102, 55), (162, 94), (277, 178)]. For KITTI data with its hard level, the computed anchors are [(12, 30), (19, 50), (30, 72), (32, 24), (46, 111), (59, 37), (76, 167), (116, 66), (223, 144)].

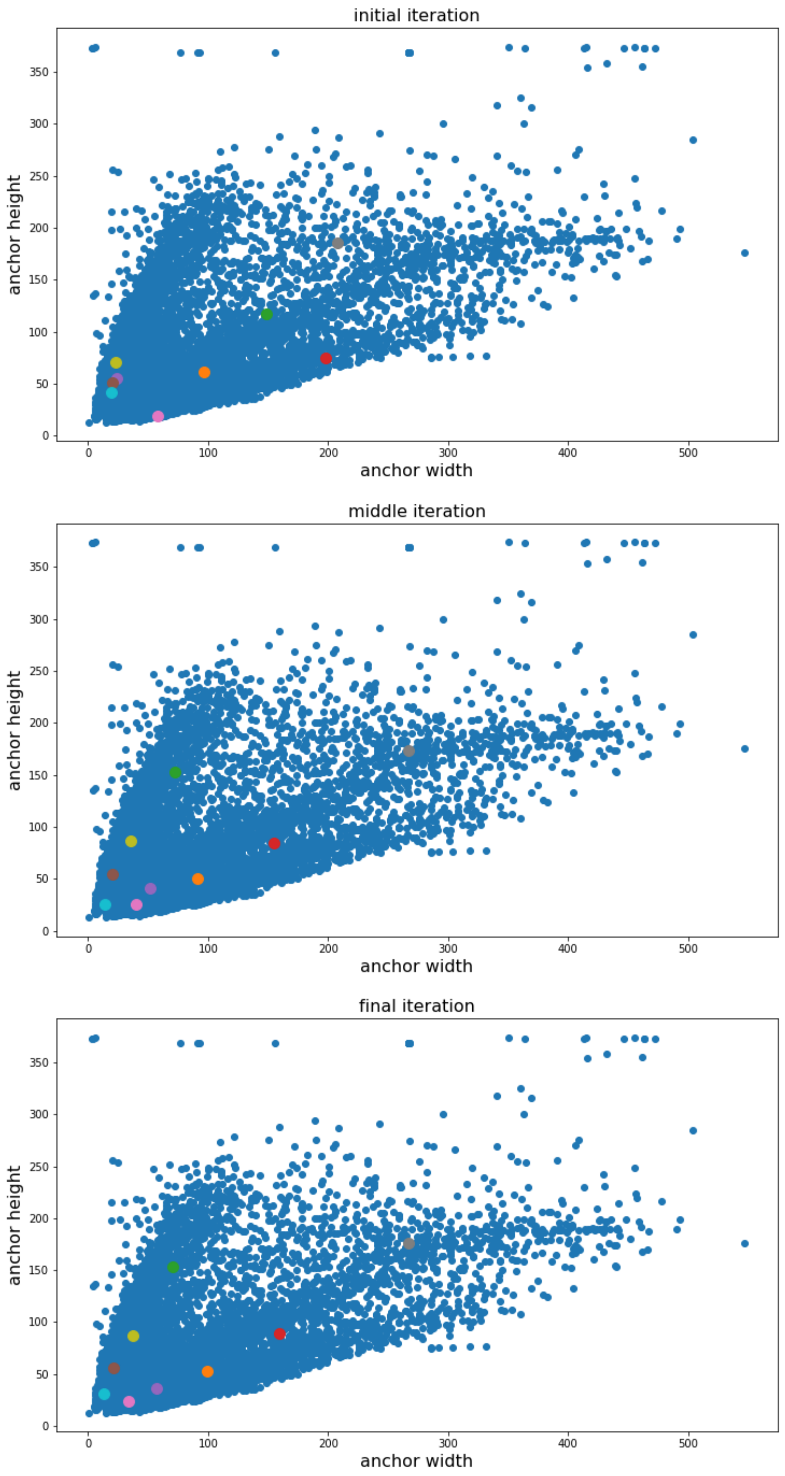
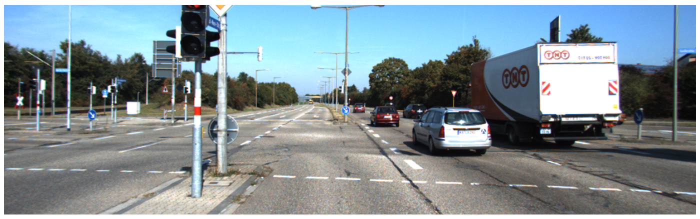


Figure 4.3.1.2: YOLOv3 anchors computation by K-means clustering in KITTI data

The input image size for YOLOv3 also needs to be determined before training. An image size is inversely proportional to the algorithm inference speed. To achieve real time operation, the inference speed has to be equal or faster than 30 frames per second, namely, the inference time is equal or less than around 33 milliseconds. In Figure 3.2.1.2, YOLOv3 compared three input image sizes in terms of accuracy and time. The larger image size produces higher accuracy but requires a longer prediction time. Under the premise of real-time inference, the 416 x 416 image size generates the highest accuracy. Therefore, in this project, the network input image is resized to 416 x 416 without changing the original width and height ratio. Besides, grey boundaries are added around a resized image to maintain the designed input size. The coordinate shifts dx and dy are also added to the labelled boxes because the grey boundaries change image pixel locations. For example, in Figure 4.3.1.3, the KITTI original image size is 1242 x 375 and the resized image size is 416 x 416.


(a) KITTI original image

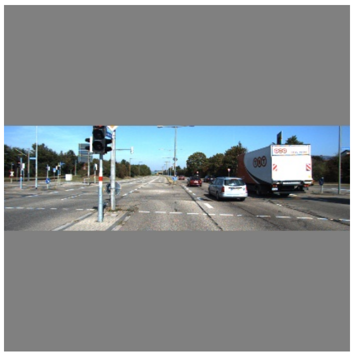

(b) KITTI resized image

Figure 4.3.1.3: KITTI image adjustment for YOLOv3 input

Besides anchors and input image size, hyperparameters are set before starting the algorithm training. Here are two types of hyperparameters. The model related hyperparameters are variables who determine a neural network structure, such as network architecture and loss function. The training related hyperparameters are variables who determine how a neural network is trained, such as batch size and number of epochs. The model related hyperparameters are kept as default values in Keras YOLOv3, but the training related hyperparameters are adjusted by trial and error to achieve an optimized result.

For Keras YOLOv3 training strategy, some terminologies are explained first. “Freeze layers” means layers are not learning, their weights and biases do not update in the training process. “Batch size” refers to the number of images that are fed into the neural network simultaneously. The larger the batch size, the more features would be learned each time at the expense of higher computational power. Thus, the batch size setting depends on the computer work load (i.e. GPU quality and quantity). “Number of epochs” indicates how many times that all images in a dataset are learned by a network. The number of epochs used in training depends on the model loss convergence performance.

In this research, for training without transfer learning, all neural network layers are unfrozen, the batch size and number of epochs are set at 8 and 80. For training with transfer learning, DarkNet-53 layers are frozen and the detector layers are unfrozen from epoch 1 to 15 with a batch size of 16; after the training loss becomes stable, all backbone layers are unfrozen for fine-tuning from epoch 16 to 80 with a batch size of 8. Early-stopping is set to stop the training process by monitoring the validation loss when its fluctuation remains within 0.1 in six epochs. Therefore, the program does not have to run 80 epochs if the learning of the network has converged.

The loss function of Keras YOLOv3 includes Binary Cross-Entropy (BCE) functions for bounding box center (x, y), objectness, classification losses computation, and Squared Error (SE) function for box width and height losses computation. To optimize the loss function (minimize losses), various optimization methods are proposed for deep learning, and they use a parameter called learning rate to control the speed of optimization process. The learning rate determines the step size of weight adjustment at each iteration while moving toward a reduction in the loss function.

Keras YOLOv3 uses an adaptive gradient descent optimization method named Adam [84]. Its adaptive learning rate can adjust itself in the training process instead of keeping constant, so it can accelerate the training loss convergence. As shown in Figure 4.3.1.4, the horizontal axis w is the weight of a neural net and the vertical axis J(w) is a loss function. The learning steps (arrows) get shorter when the loss function value approaches to a minimum. This means that the learning rate (speed) decreases in the learning process.

For training without transfer learning, the initial learning rate is set as 1e-3. For training with transfer learning, the initial learning rate is set as 1e-3 when the backbone is frozen, and reduced to 1e-4 when the backbone is unfrozen. Moreover, a learning rate scheduler is designed as follows: the learning rate is reduced by a factor of 10 via monitoring the validation loss fluctuates within 0.0001 in three epochs. It can help to keep the network learning and to improve the loss convergence.

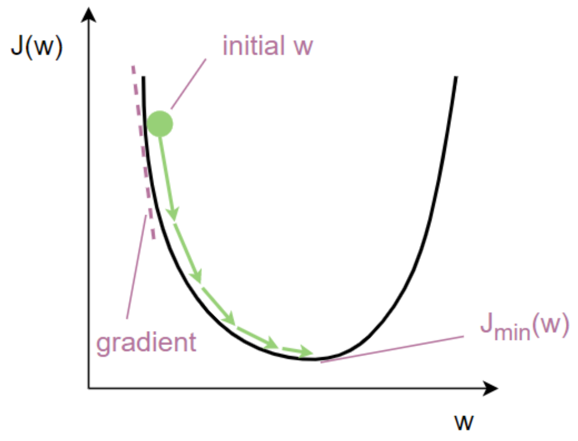


Figure 4.3.1.4: Adaptive learning rate

Online data augmentation, which represents image transformations in batches, is adopted in the training data. The purpose is to increase the amount of data by adding modifications on the existing data. It saves a lot of time and work by avoiding collecting new data and annotating them. Also, it acts as a regularizer to reduce overfitting in the training process. In Keras YOLOv3, augmentation methods applied on input images include resizing with random ratio, shifting, flipping, adding color space distortion, and adding salt-and-pepper noise.

The resizing is to change an image size. The shifting is to translate an image in any direction. The flipping is to flip an image left and right. The color space distortion is to change an image's HSV values. HSV refers to Hue, Saturation, and Value, they are the color properties of an image. The salt-and-pepper noise is a form of noise seen on an image. All image transformation hyperparameter values are kept as default. For each augmentation method, Keras YOLOv3 programming code generates 50% probability to apply it on input images in the training process.

4.3.2 PyTorch YOLOv3

Given the experimental results from Keras YOLOv3, the transfer learning technique reinforces the performance and effectiveness of object detection. Moving forward, only two cases will be considered, namely:

(i) Training with transfer learning using KITTI data with its hard level

(ii) Training with transfer learning using KITTI data without its hard level

PyTorch YOLOv3 has the same anchors and input image size as Keras YOLOv3. Rather than manually altering the training related hyperparameters' values in Keras YOLOv3, a simplified variant of genetic algorithm is applied in PyTorch YOLOv3 to automatically optimize hyperparameters in a certain range of values. As the flowchart shown in Figure 4.3.2.1, the three main components of the optimization algorithm are fitness, selection, and mutation (yellow blocks). In the fitness process, the fitness score is computed as $0.0precision + 0.01recall + 0.99mAP + 0.0F1$. The coefficient values are kept as default, and the evaluation metric values are generated by running YOLOv3. In the selection process, the default method is to select a single parent that has the highest fitness score from previous generations to create off-springs and to use them in the next iteration. In the mutation process, the mutation probability is set to 10%. For each hyperparameter in the parent, if the mutation happens, its value is multiplied by a random number. The number of iterations is the termination criterion and when all iterations are completed, the hyperparameters that generate the highest fitness score would be the optimized hyperparameters.

The genetic algorithm optimization requires long GPU hours with hundreds of generations to produce good results. However, due to the expensive computation and large time consumption of the algorithm, the number of iterations is set as 40. In addition, the mutation probability is increased from 10% to 50% to accelerate the search (mutation) process and generate an acceptable result within 40 iterations. The final hyperparameter values in PyTorch YOLOv3 for the two experiments are provided in Appendix A.

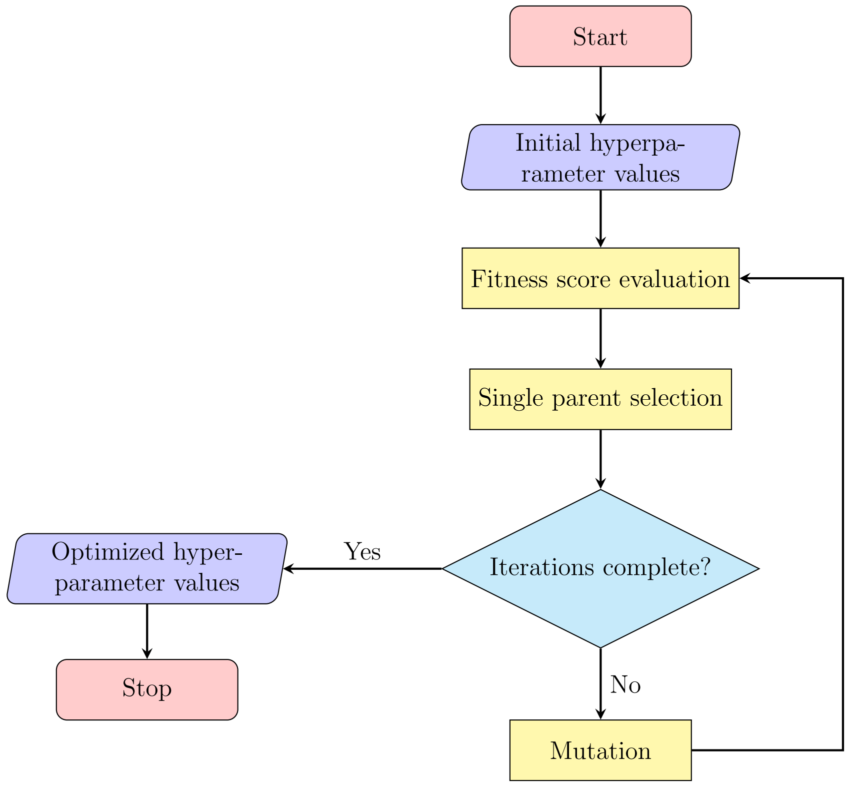


Figure 4.3.2.1: Genetic algorithm for hyperparameters optimization

Unlike the training strategy of Keras YOLOv3, all layers in PyTorch YOLOv3 are unfrozen for fine-tuning at the beginning of model training with transfer learning. The batch size and number of epochs are also customized as 8 and 80. The default optimizer is still the Adam algorithm, but the learning rate scheduler changes to reduce the learning rate by a factor of 10 when the number of epochs reaches 40% (i.e. epoch 32) and 90% (i.e. epoch 72) of the total number of epochs (i.e. epoch 80).

In the loss function of PyTorch YOLOv3, as discussed in Chapter 4 Section 1, the objectness and classification losses are still calculated using Binary Cross-Entropy (BCE) loss functions. One hyperparameter called positive weight is added into a function $BCEloss = -(\textbf{positive weight} * y * log(x) + (1 - y) * log(1 - x))$ to reduce the sample imbalance issue. The positive weight is a weight of positive examples and is equal to $\frac{negative\ examples}{positive\ examples}$ per class. If the weight is larger than 1, more negative examples than positive examples are in dataset and the network tends to predict an object as negative example to gain higher accuracy. By adding the weight into the loss function, the False Negatives (FNs) will be reduced and the recall will be improved. Contrarily, if the weight is less than 1, there are fewer negative examples than positive examples and the network tends to predict positive examples. After considering the weight, the False Positives (FPs) will be decreased and the precision will be enhanced.

For the bounding box center (x, y), width and height losses computation, PyTorch YOLOv3 replaces the Squared Error (SE) loss function in the original YOLOv3 by the Generalized Intersection over Union (GIoU) loss function [85]. Before introducing the GIoU loss function, the IoU loss function is illustrated first. IoU measures how well the predicted bounding box fits the ground truth box, and it is equal to $\frac{\mathcal{I}}{\mathcal{U}}$, where $\mathcal{I}$ and $\mathcal{U}$ refer to the intersection and union areas. Accordingly, the IoU loss function $\mathcal{L}_{IoU} = 1 - IoU$. As mentioned in Chapter 4 Section 1, the SE loss function calculates the squared difference between predictions and ground truths. Compared the two loss functions, the IoU loss function can represent losses better than the SE loss function. As shown in Figure 4.3.2.2, green rectangles represent three same ground truth boxes and black rectangles represent three different predicted bounding boxes. The L2 norms (the square root of the squared difference) between predictions and ground truths are [8.41, 8.41, 8.41], so the SE loss could be derived as [70.73, 70.73, 70.73]. It indicates that the SE loss function generates the same detection losses for different predictions. The IoUs between predictions and ground truths are [0.26, 0.49, 0.65], so the IoU loss can be computed as [0.74, 0.51, 0.35]. It shows that the IoU loss function produces different detection losses for different predictions.

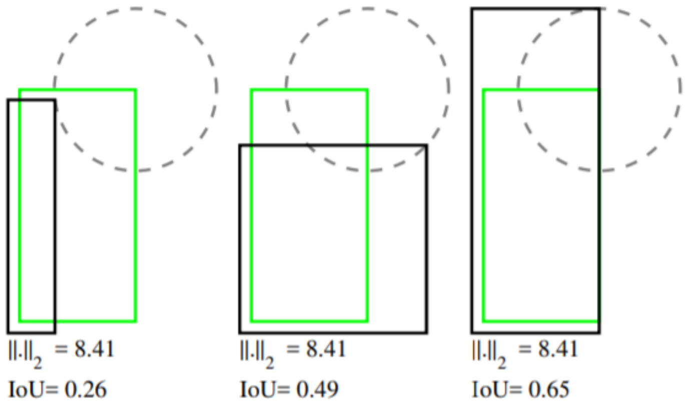


Figure 4.3.2.2: L2 vs IoU [85]

However, the IoU loss function has a disadvantage: it cannot represent the prediction performance difference if there is no overlap between the bounding box and ground truth, as shown in Figure 4.3.2.3. In this example, the blue box is ground truth of an object, yellow and green boxes are predicted bounding boxes. Here, the green box prediction is better than the yellow box because it is closer to the ground truth. However, IoU has no difference in both cases because there is no intersection between the prediction and target. GIoU is designed for addressing the weakness of IoU. Its value is equal to $IoU - \frac{A^c - \mathcal{U}}{A^c}$, where A^c refers to the smallest convex hull that encloses both the bounding box and the ground truth. A^c is the region that a red box covered for each case in the figure. Therefore, the GIoU loss function $\mathcal{L}_{GIoU} = 1 - GIoU$. PyTorch YOLOv3 selects the GIoU loss function for its bounding box localization regression.

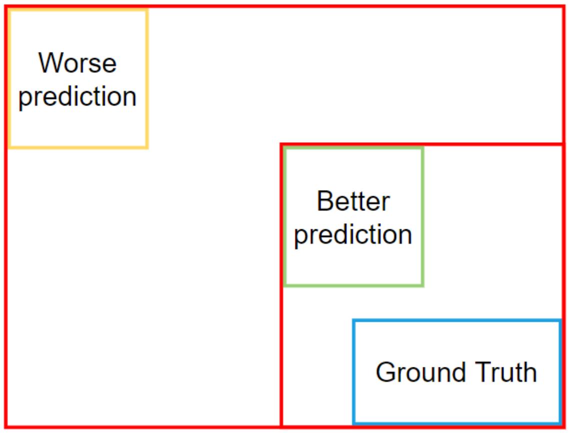


Figure 4.3.2.3: IoU metric weakness

For the online data augmentation, PyTorch YOLOv3 programming code also generates 50% probability to apply a method on input images in the training process. The augmentation methods include resizing, shifting, flipping (left-right), rotating, shearing, and adding color space distortion. An additional data augmentation technique named mosaic is added. It combines four random images into one composite image in the training process to improve the model's learning ability by increasing features at one time and to reduce the requirement for larger batch sizes. The training batch 0 is displayed as an example in Figure 4.3.2.4 (8 composite images and each of them contains 4 sub-images).

After all discussions of PyTorch YOLOv3, the hyperparameters related to the loss function, learning rate, data augmentation, IoU are listed in Table 4.3.2.1. And their values are represented in Appendix A. In addition, PyTorch YOLOv3 visualizes the training and validation results in one figure. Figures for the two experiments are also represented in Appendix A.

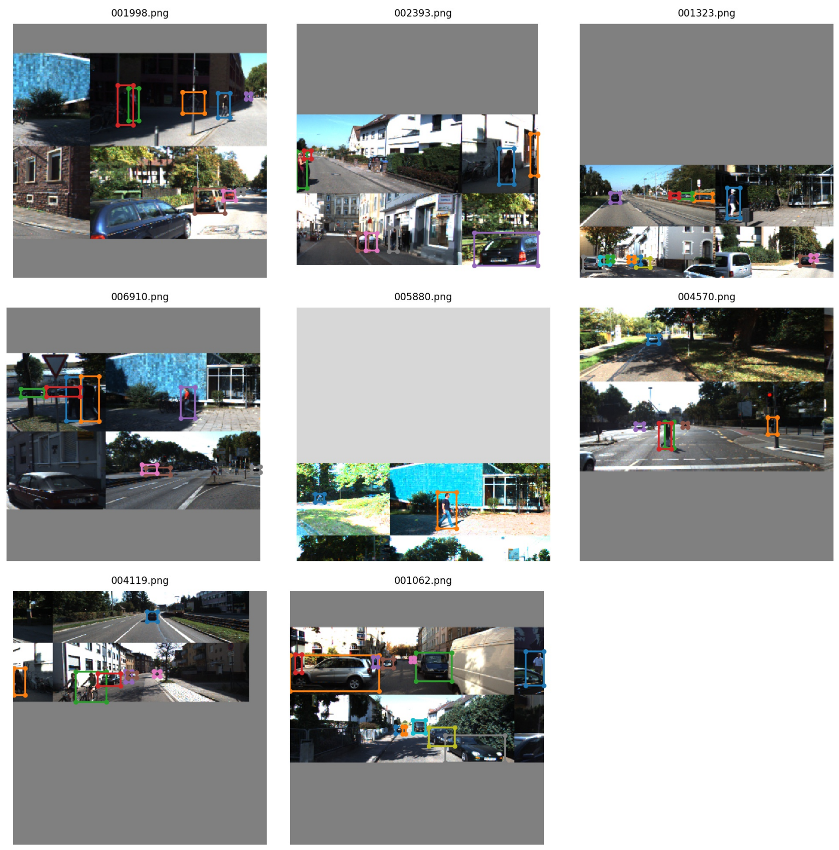


Figure 4.3.2.4: Mosaic in the training batch 0

Table 4.3.2.1: PyTorch YOLOv3 hyperparameters

	Hyperparameters	**Definitions**
Loss function related	giou	GIoU loss weight
	cls	Classification loss weight
	cls_pw	Positive weight for data imbalance in classification loss
	obj	Objectness loss weight
	obj_pw	Positive weight for data imbalance in objectness loss
Learning rate related	lr_0	Initial learning rate
Data augmentation related	hsv_h	Image hue
	hsv_s	Image saturation
	hsv_v	Image value
	degrees	Image rotation degree
	translations	Image shifting distance
	scale	Image resizing scale
	shear	Image shear degree
IoU related	iou_t	IoU threshold

4.3.3 Comparisons and Discussions

In the inference process, the confidence/IoU thresholds in the Non-Maximal Suppression (NMS) are manually fine-tuned at 0.2/0.4 for Keras YOLOv3 and 0.1/0.3 for PyTorch YOLOv3 to produce good bounding box predictions. In addition, the prediction speed with one NVIDIA GeForce RTX 2080 Ti GPU is about 28 FPS for Keras YOLOv3 and 27 FPS for PyTorch YOLOv3, both of them are able to approach real-time operation (approximately 30 FPS). Common cameras capture videos at 30 FPS, so an algorithm is real-time if its prediction speed is synchronous with or faster than a camera recording speed. The testing results of the three experiments in Keras YOLOv3 are performed in Table 4.3.3.1 and that of the two experiments in PyTorch YOLOv3 are represented in Table 4.3.3.2. Two tables include precision, recall, F1 score, F2 score for car, cyclist, and pedestrian classes. Moreover, the Average Precision (AP) per class is plotted in Figures 4.3.3.1 and 4.3.3.2.

Table 4.3.3.1: Keras YOLOv3 testing result

Experiments	Precision			Recall			F1 score			F2 score		
	Car	Cyclist	Pedestrian	Car	Cyclist	Pedestrian	Car	Cyclist	Pedestrian	Car	Cyclist	Pedestrian
Train from scratch [1]	78%	60%	59%	79%	62%	57%	78%	61%	58%	79%	62%	57%
Transfer learning [1]	85%	69%	67%	82%	69%	58%	84%	69%	62%	83%	69%	60%
Transfer learning [2]	83%	70%	73%	83%	72%	70%	83%	71%	72%	83%	71%	71%

Note: [1] refers to KITTI data with the hard level and [2] refers to KITTI data without the hard level.

Table 4.3.3.2: PyTorch YOLOv3 testing result

Experiments	Precision			Recall			F1 score			F2 score		
	Car	Cyclist	Pedestrian	Car	Cyclist	Pedestrian	Car	Cyclist	Pedestrian	Car	Cyclist	Pedestrian
Transfer learning [1]	86%	79%	81%	76%	72%	67%	81%	75%	73%	78%	73%	69%
Transfer learning [2]	85%	84%	86%	91%	81%	78%	88%	83%	82%	90%	82%	79%

From Table 4.3.3.1, training with transfer learning from a pre-trained model on the COCO dataset has distinctly better performance than training without transfer learning (train from scratch). Besides, prediction without severe occlusion and truncation (without the hard level data) has a better result than that with heavy occlusion and truncation (with the hard level data), especially in the recall metric (less missing detection). Furthermore, the detection performance of object classes in descending order is car, cyclist, pedestrian. It can be caused by the size of objects (better detection for larger object), the shape of objects (e.g. people sitting, walking), and the dataset class unbalance issue (more cars than others). When the YOLOv3 algorithm learns features from the KITTI data, it prefers to classify an object as car to achieve a higher accuracy because cars are more than cyclists and pedestrians.

Comparing results pertaining to transfer learning experiments between Table 4.3.3.1 and Table 4.3.3.2, PyTorch YOLOv3 has a significantly better detection performance compared to Keras YOLOv3. It may be due to three main modifications in PyTorch YOLOv3, namely: initial hyperparameters optimization by using a genetic algorithm rather than by manual adjustments; the SE loss function is replaced by the GIoU loss function for the bounding box localization loss computation; and, mosaic technique is used in data augmentation to improve object classification, particularly small objects.

Additionally, the confidence and IoU thresholds in the NMS are set for redundant bounding boxes filtration, and their values affect the precision ($\frac{TP}{TP + FP}$) and recall ($\frac{TP}{TP + FN}$) by experimental observations. For the confidence threshold, its increasing results in precision increasing and recall decreasing, vise versa. The reason is that more predicted bounding boxes are removed with higher confidence threshold, this results in more TP and less FP, but also more FN, vise versa. For the IoU threshold, its effect depends on objects overlapping conditions in each class. Generally, the IoU threshold increasing generates more FP and this results in lower precision, vise versa. If there are crowded objects from one class (e.g. pedestrians congregation) in an image, lower IoU threshold may cause the deletion of TPs and this results in the reduction of both precision and recall.

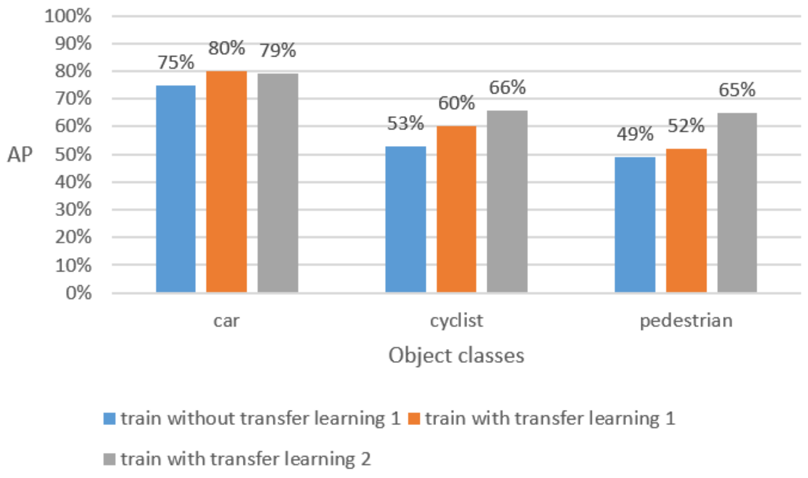


Figure 4.3.3.1: Keras YOLOv3 Average Precision (AP)

Note: [1] refers to KITTI data with the hard level and [2] refers to KITTI data without the hard level.

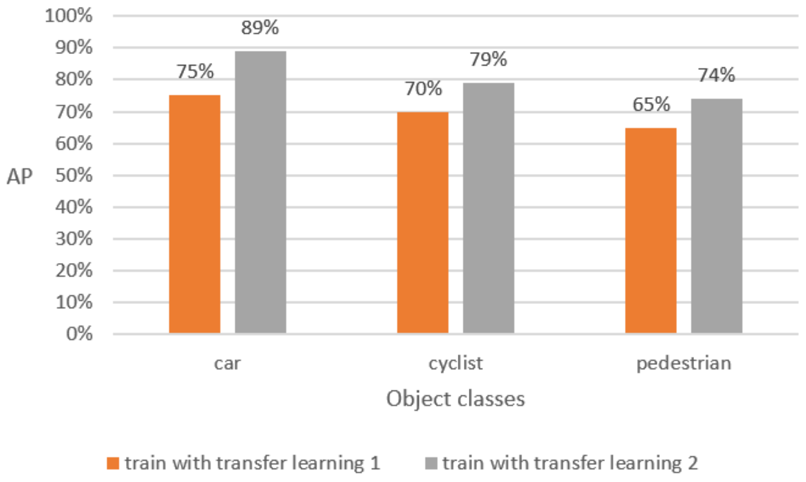


Figure 4.3.3.2: PyTorch YOLOv3 Average Precision (AP)

The values of AP per object class in Figures 4.3.3.1 and 4.3.3.2 visualize the object detection performances of experiments and also support the same analysis results from the two tables above, namely: transfer learning boosts the detection performance; severe object occlusion and truncation conditions worsen the object detection; PyTorch YOLOv3 has better detection performance than Keras YOLOv3; and, KITTI data bias results in YOLOv3 detection bias (higher AP of cars than that of cyclists and pedestrians). In addition, small objects are harder to be detected than moderate or large objects. Some testing images with predicted bounding boxes, ground truth boxes, missing or misclassification boxes are provided in Appendix A.

In conclusion, it should be noted that anchors computation (size and quantity) for custom datasets via the K-means algorithm before neural network training affects the final localization accuracy of a model. The K-means algorithm is very sensitive to the initial picked centroids. Furthermore, all object detection metrics used here are based on IoU and its reliability still needs to be considered. As mentioned earlier, GIoU is designed to compensate a drawback of IoU.

Chapter 5

License Plate Detection and Recognition

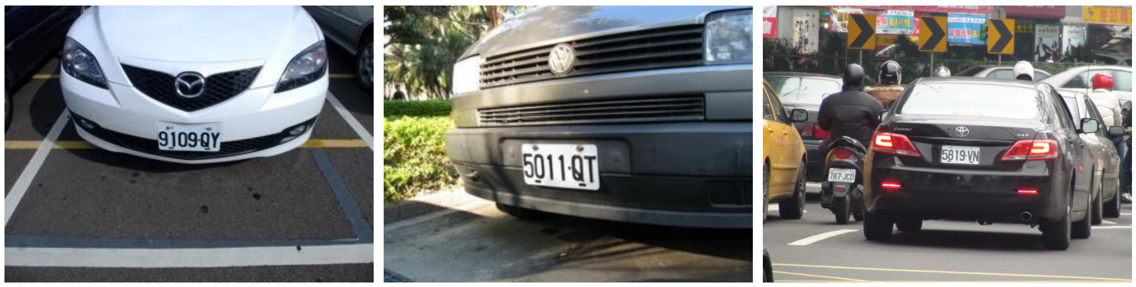
In the License Plate Detection and Recognition (LPDR) research, license plates are detected and plate numbers are recognized as texts. It is valuable for traffic-related tasks, such as traffic control, vehicle tracking, parking automation and security. For the detection stage, the real-time object detection technique in Chapter 4 is also used here to localize LPs. For the recognition stage, as reviewed in Chapter 3 Section 3.1, deep learning methods for image text recognition are mainly Connectionist Temporal Classification (CTC)-based and attention-based. Therefore, both types of algorithms are applied.

LPDR research does not have a large published dataset. This research therefore has to rely on multiple small datasets with bounding box and plate number labelling. In conjunction with these, the YOLOv3 algorithm is implemented to detect LPs. After the license plate detection, three models for the license plate recognition are investigated and compared, namely: Tesseract OCR engine, CRNN, and attention OCR. The CRNN algorithm achieves the best performance and is optimized by data augmentation, input image resampling methods, input image size adjustments, and by adding a classifier. Finally, standard license plates in Ontario, Canada are collected in urban areas using a camera installed on the lab vehicle, and the optimized model is tested on the dataset. As a goal in this research, within a range of 10 meters, the Ontario LP recognition performance is acceptable, high, or utmost when the accuracy achieves above 90%, 95%, or 99%.

5.1 Multiple Datasets

Multiple datasets include Application Oriented License Plate (AOLP) [61], Open Automatic License Plate Recognition (OpenALPR) [62], Federal University of Paraná - Automatic License Plate Recognition (UFPR-ALPR) [63], and Stanford Cars [86]. Most of them were introduced in Chapter 3 Section 3.2, and more details for each dataset are provided in this section.

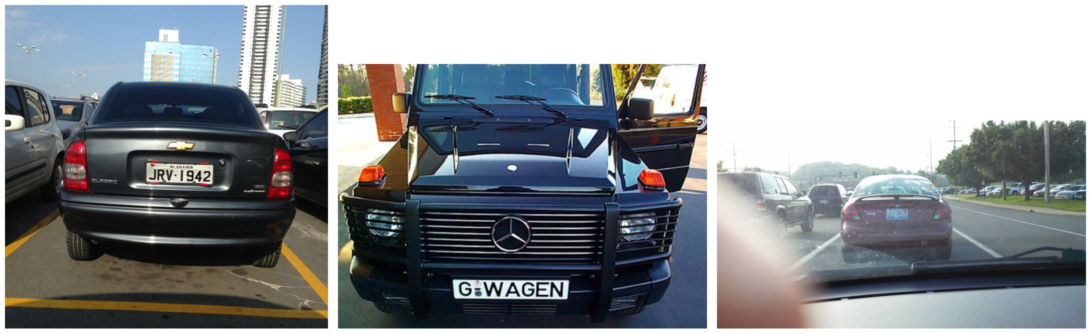
AOLP was collected from Taiwan in 2013 and it has three types of images for different applications: access control, road patrol, and traffic law enforcement. Figure 5.1.1 gives a sample image for each scenario. Access control is to capture a vehicle when it enters a location by fixing a camera facing the location. Road patrol is to capture stationary cars from close distance via a camera on a patrolling vehicle. It is used for parking security or lost vehicle searching. Traffic law enforcement is to capture driving vehicles if they exceed the speed limits, and the camera is installed on a car. By comparing the three scenarios, only the traffic law enforcement data is suitable for the LPDR research because it captures driving vehicles on roads. Therefore, images in this scenario are used here. It contains a total of 757 images taken at different times, most of them have a resolution of 640 x 480 pixels, and the rest of the images have a resolution of 320 x 240 pixels.



(a) Access control (b) Road patrol (c) Traffic law enforcement

Figure 5.1.1: AOLP scenarios

OpenALPR was generated from Brazil, Europe, and United States in 2016. As shown in Figure 5.1.2, both Brazil and Europe LPs are captured from stationary vehicles in parking lots. Some US LPs are captured when the ego-car is driven on roads, and they also have a similar size compared to Canadian LPs. Therefore, only images that were collected from the US are applied in the LPDR research. There are 222 images with different resolutions such as 1920 x 1080 pixels, 1280 x 720 pixels, and 720 x 480 pixels.



(a) Brazil LP (b) Europe LP (c) US LP

Figure 5.1.2: OpenALPR samples

UFPR-ALPR was created from Brazil in 2018. Its images were captured in different urban areas by cameras installed on driven cars, and all of them have a resolution of 1920 x 1080 pixels. In the dataset, 150 vehicles are tracked at 30 FPS recording speed and 30 images (in 1s video) are generated for each vehicle, so it contains a total of 4500 images. For the LPDR research, it is not necessary to use 30 almost repetitive images for one plate number recognition. Thus, only one image per vehicle is extracted from the dataset. In addition, UFPR-ALPR collected three types of vehicles: cars, buses, and motorcycles. For instance, Figure 5.1.3 presents how the license plate frame looks like for each vehicle type. License plates of motorcycles (square-like shape with two rows of words) are not only different from that of cars and buses (rectangular shape with one row of words), but also have significant dissimilarity from LPs in Ontario, Canada (rectangular shape with one row of words). Hence, motorcycle images are filtered out, and 120 images are retained for use.

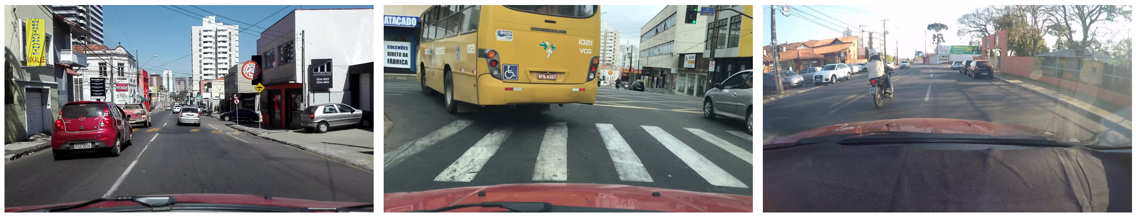

(a) Car (b) Bus (c) Motorcycle

Figure 5.1.3: UFPR-ALPR vehicles

Stanford Cars dataset was originally designed for car brand classification and it contains 16185 images of 196 classes of cars with various resolutions. The dataset labeled the bounding boxes and brand types for cars. In order to increase the data size for the license plate recognition research, some images which have clear license plates are extracted from this dataset. The reason why selecting the Stanford Cars dataset is that no Canadian LP dataset has to date been published online, US LPs are similar to Canadian LPs and can be used for a model training. A total of 93 images are selected and their car plates are manually labelled. One car example is shown in Figure 5.1.4.

Figure 5.1.4: Stanford Cars dataset sample

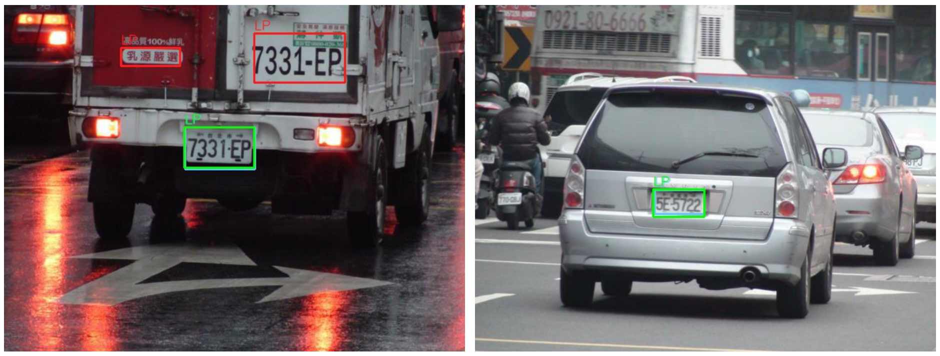
A total of 1192 images are selected from the four published datasets (757 from AOLP, 222 from OpenALPR, 120 from UFPR-ALPR, 93 from Stanford Cars). All images' annotation format is unitized, comprising image size, license plate text, and license plate bounding box pixel coordinates (min x, min y, max x, max y).

5.2 License Plate Detection

The license plate detection stage has the same training process as 2D object detection and the PyTorch YOLOv3 algorithm with transfer learning is used as described in Chapter 4 Section 3.2. The data is split into 90% and 10% for training/validation and testing. Furthermore, the training/validation part consists of 80% training images and 20% validation images. So the number of training/validation/testing images are 858/215/119. Some minor changes of the detection model are made specific to license plates. In terms of annotation, only one class (i.e. LP) is labeled in this training. Nine predefined anchors for YOLOv3 are re-computed using the training images, the generated anchor widths and heights are [(45,24), (60,32), (73,36), (81,43), (92,50), (107,59), (113,36), (132,68), (188,86)]. For data augmentation, the flip (left-right) operation is removed because LP rectangular shape feature does not change after flipping.

The evaluation metrics obtained in testing are as follows: 83% precision, 87% recall, 85% F1 score, 86% F2 score, and 86% Average Precision (AP). Examples of False Positive (FP) and False Negative (FN) are presented in Figure 5.2.1. Green boxes refer to correct detection and red boxes refer to incorrect predictions. The left sub-figure shows that YOLOv3 detector predicts logos and text prints on the back of a truck as license plates. Furthermore, the right sub-figure shows that one license plate (middle left in figure) fails to be detected mainly because it is far away from the ego-car.

All considered, the limited amount of labelled data is not large enough to fine-tune the YOLOv3 model to achieve a better detection performance. In addition, small object detection (e.g. LPs) is still a challenge for deep learning algorithms. Moreover, logo prints on vehicle surfaces and traffic occlusion also significantly impact the detection result.



(a) False Positive (FP) (b) False Negative (FN)

Figure 5.2.1: License plate detection FP and FN examples

5.3 License Plate Recognition

In the license plate recognition stage, all license plate numbers are combinations of alphabets and digits, so the recognition process is to classify each character on a plate into one of 36 classes: A to Z (26 alphabets), and 0 to 9 (10 digits). To prepare the recognition data, LPs are cropped from images using their bounding box coordinates in the annotation files. Most images have one LP inside, and few images have two LPs inside, resulting in a total of 1226 LPs extracted for use. The training/validation/testing data percentage distribution is set as 75%/15%/10%, in other words, 920 images for training, 200 images for validation, and 106 images for testing.

In this section, three methods are explored: Tesseract OCR engine, CRNN, and attention OCR. Each method is explained and the testing results are presented in subsection 5.3.1. Next, four experiments are designed to optimize the algorithm prediction in subsection 5.3.2: data augmentation, input image scaling methods investigation, input image size adjustment, and a classifier for prediction post-processing. To apply the optimized model on LPs in Ontario, some standard license plates are captured in the Greater Toronto Area (GTA) using a Logitech Brio webcam installed on the Centre for Mechatronics and Hybrid Technologies (CMHT) lab vehicle. The camera setup, data collection and pre-processing, and testing result are discussed in subsection 5.3.3.

5.3.1 Models Comparison

5.3.1.1 Tesseract OCR Engine

As an Optical Character Recognition (OCR) engine, Tesseract [87] was originally a doctoral project in HP Lab between 1985 and 1994. After testing at the University of Nevada, Las Vegas (UNLV) in 1995, it was released as an open-source model in 2005. Since 2006 until now, Tesseract has been developed by Google. The OCR engine is not a deep learning algorithm, the reason for investigating it is that Tesseract is considered as one of the most accurate open-source OCR engines. The basic architecture of Tesseract OCR engine is presented as a flow chart in Figure 5.3.1.1.1.

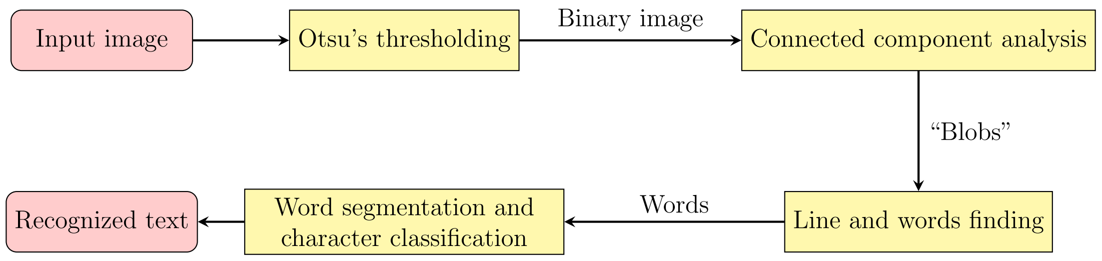


Figure 5.3.1.1.1: Flow chart of Tesseract OCR engine architecture

At the beginning, an input gray-scale image (pixel values are between 0 and 255) is converted to a binary image (pixel values are either 0 or 255) via Otsu's thresholding. Thresholding is to polarize image pixel values by assigning 0 (black color) to pixels whose intensity values are below a threshold and assigning 255 (white color) to pixels whose intensity values are above the threshold. Otsu's method assumes image pixel values can be classified into two clusters, one cluster approaches 0 and another towards 255. So the generated histogram would also have two peaks. The method is to find the optimal threshold value iterating over the numbers from 0 to 255 by minimizing the sum of the intra-class variance per cluster, or equivalently, by maximizing the inter-class variance among two clusters.

Figure 5.3.1.1.2 visualizes the Otsu's thresholding. The x axis represents pixel intensity values (0-255) and the left y axis represents the number of pixels at each intensity value. The red curve displays the inter-class variance and the right y axis shows its value at each pixel intensity. The black vertical line refers to the threshold, it iterates (horizontally moves) from 0 to 255, and its left and right sides are assigned to two clusters. The optimal threshold is located at the maximum inter-class variance. The inter-class variance equation is $w_{class0}\ w_{class255}\ (\mu_{class0} - \mu_{class255})^2$, where w represents $\frac{\sum pixel\ count\ in\ a\ class}{the\ total\ number\ of\ pixels}$ and μ refers to $\frac{\sum(pixel\ count)(pixel\ intensity)}{\sum pixel\ count\ in\ a\ class}$.

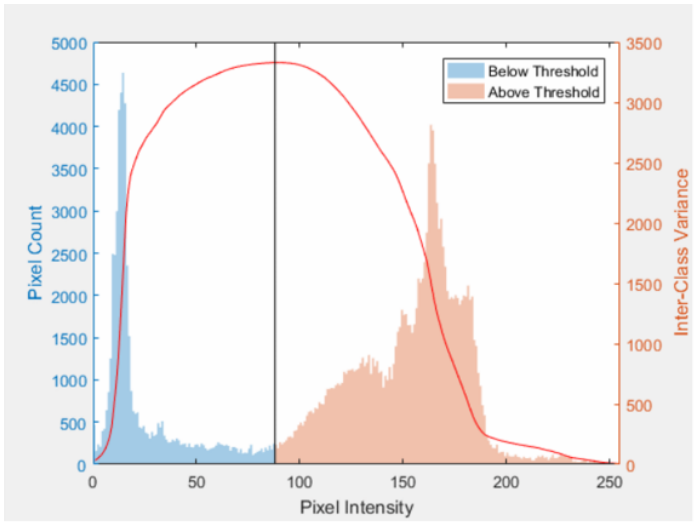


Figure 5.3.1.1.2: Otsu's thresholding visualization [88]

After generating a binary image, Connected Component Analysis (CCA) is applied to detect connected regions inside the image. Connected regions are described as "blobs", and each "blob" is assigned with an unique label. Figure 5.3.1.1.3 is drawn as an example. In (a), an 8x8 binary image contains background (white area) and foreground (black area). For background pixels, they are assigned with 0 as their labels. For foreground pixels, their labels are set as question marks temporarily and will be updated later. In (b), pixels are scanned from left-to-right and top-to-bottom. When reaching a foreground pixel, its label is determined by its left and top pixel labels. There are three conditions: first, if both left and top labels are 0, the pixel is assigned with a non-zero label whose value is one larger than the maximum label in the image; second, if one of the left or top labels is 0 and another is a non-zero label, then the pixel is set as the non-zero label; third, if both left and top labels are non-zero labels, the pixel is assigned as min(two non-zero labels). In (c), after all foreground pixels are scanned, the larger one between the two non-zero labels in the third condition in (b) would be reduced to min(two non-zero labels). In (d), two "blobs" are formed according to pixel labels and they are plotted with red and green colors in the image.

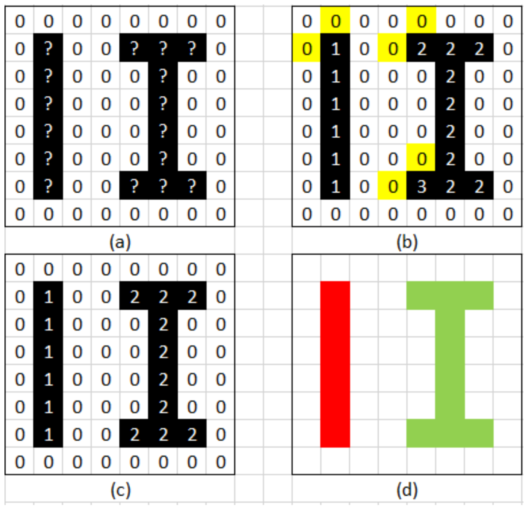


Figure 5.3.1.1.3: Connected Component Analysis

Next, line and words finding techniques are used to fit a baseline for text and separate words according to white spaces between blobs along the baseline direction. The paper of Tesseract OCR [87] does not describe the details of their techniques, but the basic idea is as follows: for line finding, noise blobs are filtered out first if their heights are smaller than a fraction of the median height of all blobs in the image and a baseline under text blobs is constructed by a least median of squares fit; for word finding, white spaces are estimated between blobs for words separation.

The output words are segmented as characters by a trained chopper, and characters are approximated as polygons by polygonal approximation which represents boundaries of characters by straight line segments. For instance, Figure 5.3.1.1.4 shows a chopped word "mountains" and polygonal featured characters. For each character, the similarity between its polygonal feature and prototypes of all classes are then computed. Finally, each character is classified into a class with the highest similarity. There are no details about the chopper training and feature similarity calculation process in the paper.

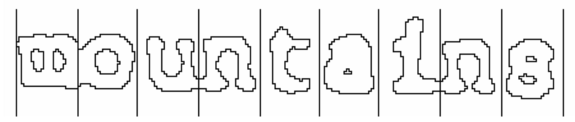


Figure 5.3.1.1.4: Word segmentation [87]

To implement Tesseract OCR on the license plate recognition, a python library named pytesseract (Python-tesseract) is used. It is a wrapper of Google's Tesseract-OCR Engine, LPs can be read and printed directly by calling a function "pytesseract.image_to_string()" without training mode. Therefore, 106 testing images are used here to evaluate the model accuracy. The testing result is that only 10 out of 106 LPs are recognized correctly, namely, the recognition accuracy is about 9.43%. In summary, Tesseract OCR engine generates low accuracy for scene text whose images are captured in both outdoor environment and motion condition with lower quality compared with printed text such as PDF documents.

5.3.1.2 CRNN

Convolutional Recurrent Neural Network (CRNN) with Connectionist Temporal Classification (CTC) loss [59] is a deep learning algorithm for image text recognition. As Chapter 3 Section 3.1 mentioned, it contains three components in order: a CNN for image feature extraction; a RNN for characters prediction per time step in the feature sequence output from the CNN; a CTC for final text decoding and generation from the RNN predicted character class probabilities. The mechanism of them will be explained below. In addition, input images are gray-scale, their heights are resized to 32 pixels and widths are resized to W pixels according to image width/height ratios.

The CNN is designed as a combination of seven convolutional layers, four max pooling layers and one fully connected layer. Convolutional layers generate feature maps, max pooling layers down-sample feature map sizes, and the dense layer converts feature maps to a feature sequence. Each input image is fed into the CNN with width W and height 32. After convolutional and max pooling layers, 512 feature maps are generated, the width and height of feature maps are reduced from (W, 32) to (W/4 - 1, 1). So the output shape becomes (W/4 - 1) x 1 x 512.

The dense layer works as shown in Figure 5.3.1.2.1. It "slices" all feature maps (bottom part in the figure) along the width direction, and turns them to columns arranged in alignment from left to right (top part in the figure). Each column contains all image features at one time step. Therefore, feature maps output with shape (W/4 - 1) x 1 x 512 is converted to a feature sequence with shape (W/4 - 1) x 512.

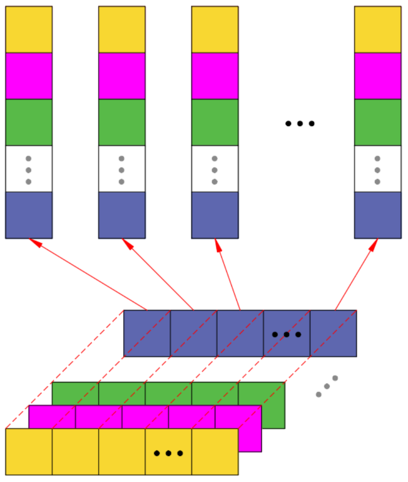
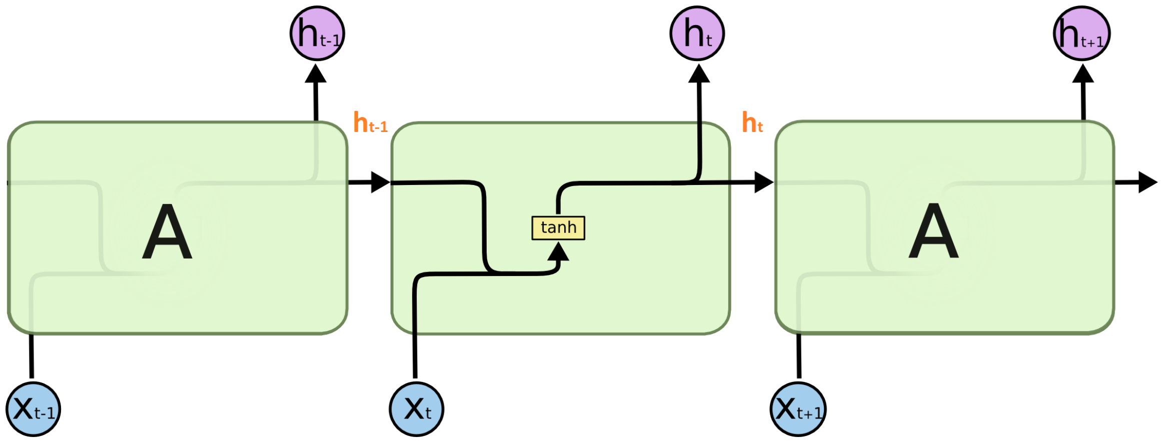


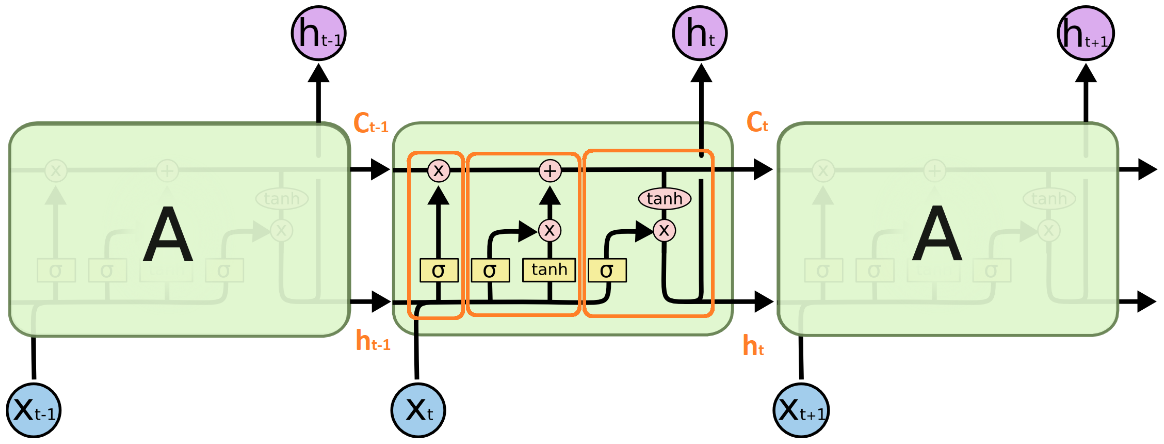
Figure 5.3.1.2.1: Feature maps to sequence

The RNN includes two bi-directional LSTMs. Long Short Term Memory (LSTM) [89] is a type of RNN, the structures of original RNN and LSTM are shown in Figure 5.3.1.2.2. The original RNN was reviewed in Chapter 3 Section 1.3, its current hidden state h_t is computed using a single activation function (e.g. tanh in the figure) with the information of the last hidden state h_{t-1} and current input X_t. If a data sequence is long, the original RNN becomes unable to learn to connect the information between current hidden state and very early hidden states. This problem called vanishing gradient and results in the memory degradation.

LSTM is designed to address the problem of original RNN and improve information connections during long time intervals. The core of LSTM is the cell state C (the top horizontal line in Figure 5.3.1.2.2b). It transports information all the way though entire network units. To remove or add information to the cell state C, three gates are designed in LSTM architecture: forget gate, input gate, and output gate. They are highlighted by orange boxes in the figure. The forget gate (left) removes old information out from C; the input gate (middle) adds new information into C; and the output gate (right) computes current hidden state h_t using the information of last hidden state h_{t-1}, current input X_t, and current cell state C_t. To control the amount of information flowed, each gate has a sigmoid function σ that produces a number in a range of 0 to 1. As shown in the figure, all sigmoid outputs are multiplied by information that pass in or out the cell state C. As an example of sigmoid outputs, zero would stop all information flow, one would allow all information flow, and 0.5 would accept half information flow.



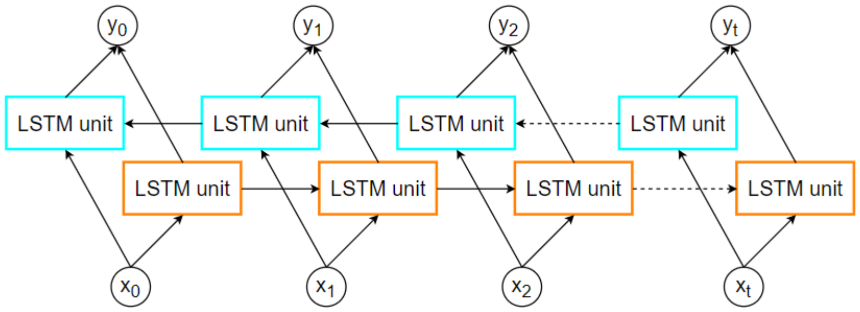
(a) Original RNN



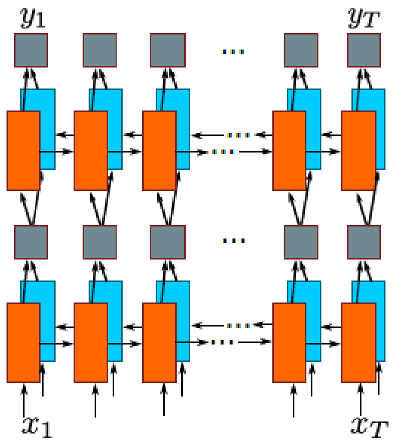
(b) LSTM

Figure 5.3.1.2.2: Structures of original RNN and LSTM [90]

A bi-directional LSTM is visualized in Figure 5.3.1.2.3a. It combines two independent LSTMs in parallel. The input sequence is fed into both LSTMs, then the information flows in opposite directions, and outputs from two LSTMs are concatenated as the final output. This kind of architecture allows network to learn and predict current information from both past and future, rather than only previous information. The two bi-directional LSTMs structure in CRNN can be seen in Figure 5.3.1.2.3b. They are connected in series followed by a dense layer and output probabilities for 36 (A to Z and 0 to 9) + 1 (blank character ϵ) = 37 classes at each time step. If there is no character at a time step, the model uses the blank character class as its prediction. The output sequence shape becomes (W/4 - 1) x 37.



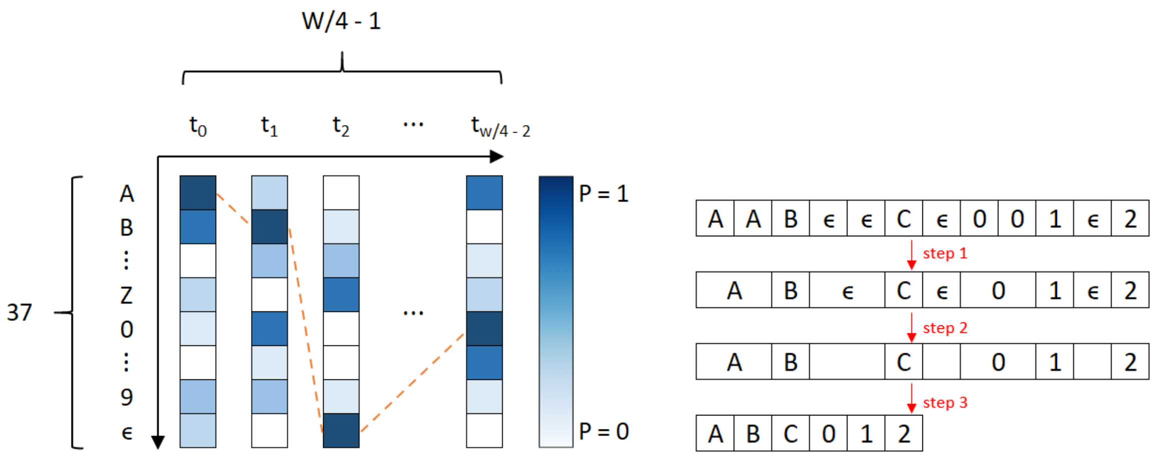
(a) One bi-directional LSTM



(b) Two bi-directional LSTMs [59]

Figure 5.3.1.2.3: LSTM architecture in CRNN

Connectionist Temporal Classification (CTC) [91] is a method to generate and optimize text output via its decoder and loss function. For the decoder, the most simple and fastest approach is best path decoding. As shown in Figure 5.3.1.2.4a, blue colored grids represent predicted probabilities of characters per time step from the RNN and the darker the color, the larger the probability. The best path decoding is to pick the character with the largest probability per time step, and connect all picked characters as a label sequence (e.g. orange dashed lines in the figure). After decoding, text output is generated in three steps which are illustrated as an example in Figure 5.3.1.2.4b: firstly, adjacent repeated characters are merged; next, blank characters ϵ are removed; finally, all rest characters are connected as the output text.



(a) CTC decoding (b) CTC text generation

Figure 5.3.1.2.4: CTC decoder

To optimize the CRNN recognition result, the probability $P = \prod_{t=0}^{t=\frac{W}{4}-2} P_t$ from the decoder in CTC needs to be maximized. Therefore, the CTC loss function is designed as $-log(P)$ and the negative log probability is minimized in the algorithm training process.

PyTorch version CRNN algorithm code [92] is open-source, it allows transfer learning and also provides a pre-trained model from a synthetic word dataset (Synth) released by Jaderberg et al. [93]. This dataset contains 9 million images covering 90K English words, and its sample images are shown in Figure 5.3.1.2.5. According to the figure, the license plate dataset is not similar to the synthetic dataset, and therefore, the CRNN algorithm with the pre-trained weights will be fine-tuned using the license plate data.

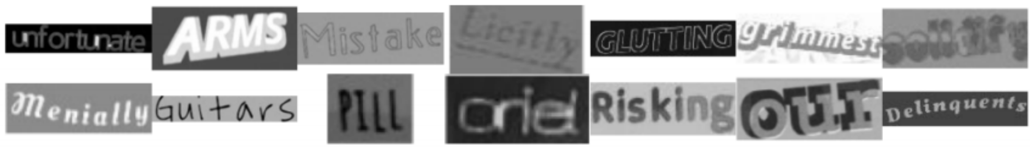


Figure 5.3.1.2.5: Synth dataset [93]

In the training process, with only one NVIDIA GeForce RTX 2080 Ti GPU computation power, the original batch size in the code is reduced from 64 to 32, and the number of epochs is manually adjusted from 25 to 60. Also, the initial learning rate lr is reduced from 0.01 to 0.001 for slower fine-tuning. A learning rate scheduler is added to decrease the learning speed by a factor of 10 when the training process completed 80% and 90% of the number of epochs. It helps the loss convergence in the end.

In the testing process, LPs are predicted by the CRNN algorithm at about 72 FPS, so the prediction speed satisfies the real-time recognition. 93 out of 106 testing LPs are identified correctly, and thus, the model achieves 87.74% accuracy. Compared with Tesseract OCR engine, CRNN with transfer learning has a much higher recognition accuracy for scene text recognition.

5.3.1.3 Attention OCR

Attention OCR [60] is also a deep learning algorithm for image text recognition. It combines a CNN and a sequence-to-sequence (seq2seq) model with attention. Its CNN has the same mechanism and architecture as that in CRNN. Its seq2seq model with attention encodes the feature sequence from the CNN and decodes information to final text outputs directly. The mechanism and architecture of seq2seq model and attention are explained below. Moreover, input images have the same formats as that in CRNN.

A seq2seq model is a model that translates a sequence of items (e.g. Chinese) to another sequence of items (e.g. English). An encoder-decoder framework is a standard structure for a seq2seq model, as shown in Figure 5.3.1.3.1. Both encoder and decoder are composed of RNNs, and output could have different length from input. The encoder processes an input sequence and its final hidden state is sent to the decoder. The final hidden state vector is also named context vector. Next, the decoder uses that state vector to predict an output sequence.

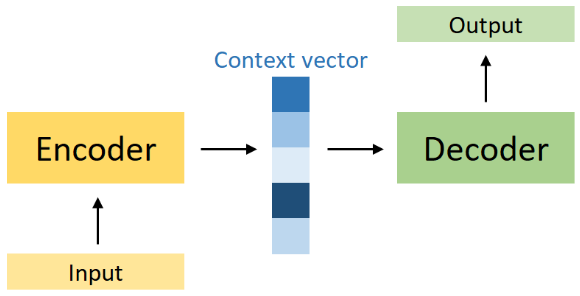


Figure 5.3.1.3.1: Encoder-decoder structure

Note that the context vector has a fixed size in a model, so it is difficult to store the entire input information into one fixed-length vector if the sequence is long and some information are lost in the encoding process. Also, information at all time steps in the encoder may have different degrees of connections with information per time step in the decoder, but the context vector cannot represent all relations in the decoding process. To address these problems, attention is created and implemented in the encoder-decoder framework.

Attention is a technique that mimics human visual attention to correlative information in a sequence. It allows a model to focus on information from certain encoding hidden states at each decoding time step by assigning different weights to all encoding hidden states. The focused hidden states are assigned larger weights. After all, the weighted encoding hidden states are used in the decoding process. Therefore, attention utilizes information in all hidden states rather than the final hidden state in the encoder. In the meantime, at each decoding time step, attention provides correlation between each encoder state and that decoder state. Figure 5.3.1.3.2 gives a clearer visualization about the mechanism of attention.

The encoder generates its hidden states h_i in a time interval [0, t], and the decoder produces its hidden states s_j in a time interval [0, k]. At the beginning, for a decoder state s_j, the "relevance" between each h and s_j is calculated using the attention function $score(h_i, s_j) = w\ tanh(w_h h_i + w_s s_j),\ i = 0, ..., t$, which proposed by Bahdanau et al. [94]. w, w_h, w_s are learnable weights in the function. Next, scores are scaled to a range of [0, 1] via the softmax function. The scaled scores are the attention weigths w. After that, the attention vector c_j is computed as the sum of element-wise product between attention weights and encoder states $c_j = \sum_{i=0}^{t} w_i h_i$. Finally, the attention vector c_j is concatenated with the decoder state s_j into one vector to generate text output. One disadvantage of attention is that it is heavily time-consuming, because at each decoding time step, one attention vector is computed for that decoder state.

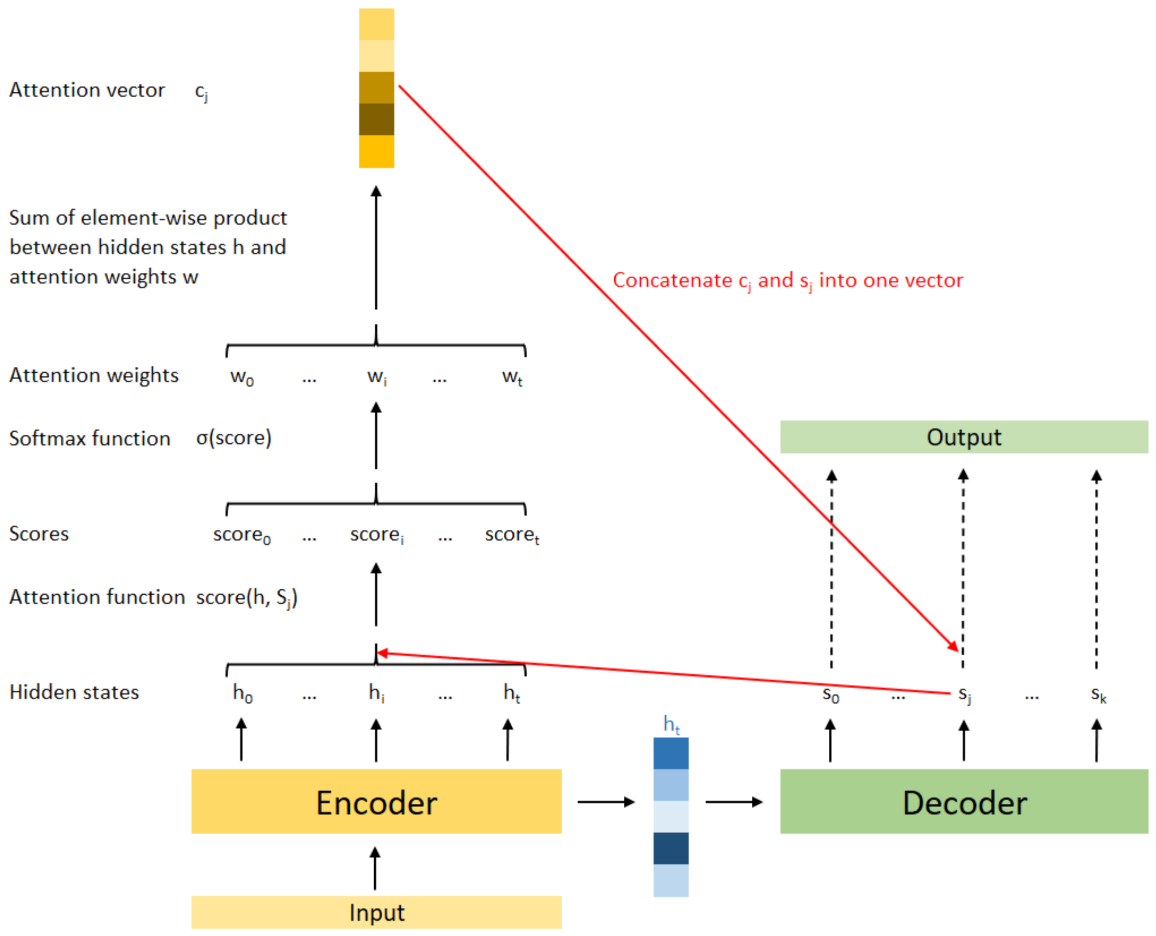

Attention vector c_j
Sum of element-wise product between hidden states h and attention weights w
Attention weights w_0 ... w_i ... w_t
Softmax function σ(score)
Scores $score_0$ ... $score_i$ ... $score_t$
Attention function score(h, S_j)
Hidden states h_0 ... h_i ... h_t
Concatenate c_j and s_j into one vector
Output
s_0 ... s_j ... s_k
h_t
Encoder
Decoder
Input

Figure 5.3.1.3.2: Attention mechanism

After explaining the mechanism of seq2seq model and attention, the architecture of seq2seq model with attention in attention OCR is shown in Figure 5.3.1.3.3. The encoder contains one bi-directional LSTM, and the decoder consists of two uni-directional LSTMs. Similar to CRNN, the decoder of attention OCR is followed by a fully connected layer, it produces probabilities for 36 (A to Z and 0 to 9) classes and the class with the highest probability is selected as the final output at each time step. The log loss is also used as the algorithm loss function.

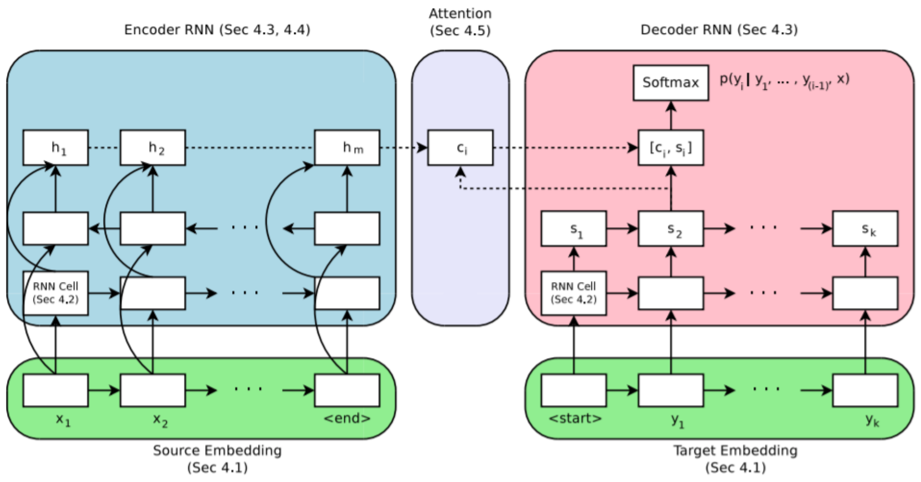


Figure 5.3.1.3.3: Encoder-decoder architecture with attention [95]

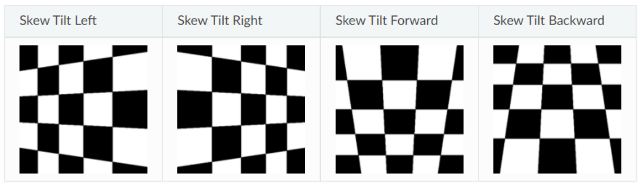
Python built-in model aocr is a Tensorflow version attention OCR. It is easy to implement but does not provide any pre-trained weights from large image text datasets. Thus, the network will be trained without transfer learning for the license plate recognition. Furthermore, in the training process, most hyperparameters are kept at their default values, except for manually adjusting the batch size and the number of epochs to 32 and 80 to achieve an optimized training result.

In the testing process, LPs are real-time predicted by attention OCR at about 39 FPS, but the speed is slower than that of CRNN. 75 out of 106 testing LPs are recognized correctly, in other words, the recognition accuracy is around 70.75%. By comparing all three algorithms: Tesseract OCR engine, CRNN with transfer learning, and attention OCR without transfer learning; CRNN with transfer learning has the highest accuracy and the fastest prediction speed. Therefore, in next subsection, four experiments are implemented to improve the CRNN model recognition performance.

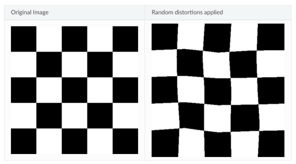
5.3.2 Optimization Experiments

Four experiments are designed for the CRNN recognition optimization in this research: the first strategy is to augment data via image perspective skewing and distortion, the second method is to attempt five image scaling filters, the third approach is to adjust input image size, and the fourth way is to add a post-processing classifier. The objective and details of experiments will be described below.

By observing the misidentified LPs from the testing result in Section 3.1.2, the model misclassifies characters which have high similarity in a part of their shapes. For example, {[D, 0], [B, 8], [M, N]} have little differences in the left or right side of their contours; and {[Z, 2], [I, 1], [B, R]} pairs are a bit different in the top or bottom side. Thus, perspective skewing is applied in data augmentation to amplify the right/left/top/bottom features of characters, as shown in Figure 5.3.2.1a. In addition, image distortion is also used to reinforce the neural network learning ability, as seen in Figure 5.3.2.1b.



(a) Perspective skewing



(b) Distortion

Figure 5.3.2.1: Image augmentation approaches [96]

Some samples of augmented LPs are presented in Figure 5.3.2.2. (a) and (d) are corresponding images before and after skewing right, the left side of character D is zoomed in. (b) and (e) are images before and after skewing forward, the top sides of characters Z and 2 are emphasized. (c) and (f) are images before and after distortion, it is helpful for recognition if a license plate is old and its plate label sticker has many wrinkles.

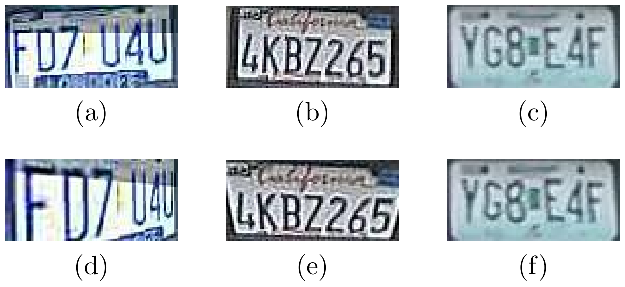


Figure 5.3.2.2: Image augmentation samples

By applying data augmentation, the prediction accuracy is increased from 87.74% (93 out of 106 LPs are recognized) to 95.28% (101 out of 106 LPs are recognized). This experiment significantly improves the LP recognition performance of the CRNN model.

LP images need to be scaled to a resolution of W x 32 pixels before feeding into the deep learning network, and image scaling technique has influence on images' quality. Because down-sampling removes pixels from the original images and up-sampling adds pixels to original images. The original resampling method in the model is bilinear. It is investigated and compared to other four methods (nearest, bicubic, box, lanczos) in the second experiment to examine which image scaling technique could generate the highest LP recognition accuracy.

The nearest, bilinear, and bicubic methods use adjacent pixels in an original image to interpolate pixels in a target image. In Figure 5.3.2.3, the top three plots clearly represent how these methods work in 1D dimension before extending to 2D dimensions. The black and yellow/green/red/blue points refer to a target point and neighbouring points. The 1D nearest-neighbour mode determines the target point value according the nearest neighbouring point value; the linear mode decides the target point value by linear interpolation between two neighbouring points; and the cubic mode generates the target point by cubic interpolation within four neighbouring points. In the bottom three plots (2D dimensions), all colorful points can be regarded as pixel center points, and their values refer to pixel values. Thus, for the pixel generation in target images, 2D nearest-neighbour and bilinear modes leverage 2x2 surrounding pixels, and bicubic mode utilizes 4x4 neighbouring pixels.

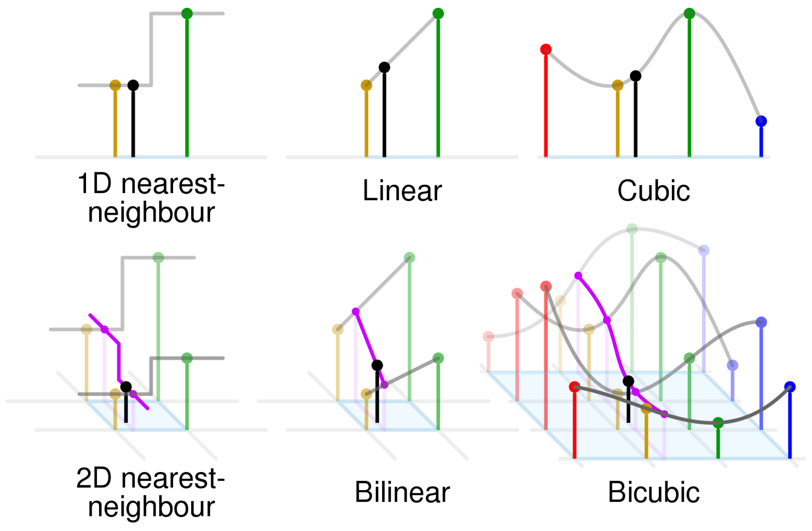


Figure 5.3.2.3: Nearest vs Bilinear vs Bicubic interpolation [97]

The box resampling model computes a target pixel value as the average of pixels' values inside a window, and the window size (sw, sh) is equal to (the scaling ratio of an image width w, the scaling ratio of the image height h). The lanczos resampling approach is to interpolate the whole image pixels in 2D dimensions using sinc function [98], and pixels in a target image are then sampled from the interpolated surface.

After training and testing the CRNN model with five image scaling methods respectively, the LP recognition accuracy are listed in Table 5.3.2.1. By comparing the testing result, the best image resampling method in this experiment is the original bilinear model, hence, here is no improvement for the CRNN recognition performance.

Table 5.3.2.1: LP recognition accuracy of image resampling techniques

Resampling filters	Nearest	Bilinear	Bicubic	Box	Lanczos
LP recognition accuracy	84.91%	**95.28%**	94.34%	87.74%	93.40%

In the input image scaling process, images in each batch have to retain the same size before entering the network. Their heights are fixed at 32 pixels, but their unified widths W need to be determined using batch images. The original method in CRNN algorithm is to calculate the ratio of width and height per image in a batch, then pick the maximum ratio and multiply it with 32 to produce the image width W for all images in that batch. This method may have a problem that some images would be excessively "stretched" horizontally and characters onside would also be misshaped. Therefore, the third experiment is designed to enhance the original method.

The idea is as follows: the python function random.random() is used to generate a random number between 0.0 and 1.0, if the returned random number is less than 0.5, the original method in the algorithm is applied to define the image width W for a batch; if the random number is larger or equal to 0.5, then pick the high median ratio and multiply it with 32 to produce the image width W for a batch.

After the input image size adjustment, the CRNN recognition accuracy of this experiment is increased from 95.28% (101 out of 106 LPs are recognized) to 97.17% (103 out of 106 LPs are recognized). It indicates that image resizing affects the shape of characters on the license plates and that further impacts the recognition behaviour.

By analyzing the third experiment result, 3 out of 106 LPs are misidentified in the prediction. The errors are caused by confusion between alphabets and integers (B and 8, I and 1). Thus, adding a post-processing classifier to rectify misclassifications between uppercase letters and numbers is the fourth experiment. The classifier is implemented in the next subsection for the Ontario license plate recognition. As the standard Ontario LPs have a fixed length of 7, the first four positions are alphabets, and the last three positions are integers. If an alphabet (integer) is predicted at a position of integers (alphabets), the classifier could replace it by a comparable integer (alphabet). Table 5.3.2.2 includes all confused letters and numbers that appeared in this research and the classifier works on the basis of this table.

Table 5.3.2.2: Confused alphabets and integers in LP recognition

Alphabets	A	B	D	G	I	S	T	Z
Integers	4	8	0	6	1	5	7	2

5.3.3 Ontario License Plate Recognition

For Ontario license plate recognition with the optimized CRNN model, the camera device setup is briefly described. Next, the LPs collection and pre-processing are depicted. Afterward, the deep learning algorithm is re-trained and the testing results are discussed.

5.3.3.1 Camera Setup

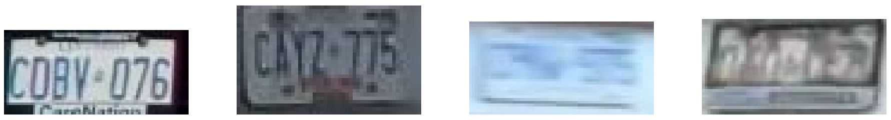
As mentioned in Chapter 1, the CalmCar optical camera is installed on the top of the front windshield inside the lab vehicle, and it is primally used for vehicle perception tasks such as object detection. The camera has a resolution of 1280 x 720 pixels and records videos at 30 FPS. However, when testing the hardware on roads, its resolution is low in capturing plate numbers in both urban and highway scenes.

Consequently, a Logitech Brio webcam (Figure 5.3.3.1.1) is used in this research. It is installed in the middle of the vehicle hood and connected to a laptop for video recording. The webcam setting is 1920 x 1080 resolution with 30 FPS recording speed. After driving trials, the auto focus, auto exposure, and Back Light Compensation (BLC) functions are disabled. The reasons are that the auto focus mode could result in focusing on certain non-LP regions and blurring surrounding views; the auto exposure and BLC modes bring too much light so the captured images become over-exposed. Therefore, the device focus and exposure values are manually adjusted depending on the driving environment at each time. Moreover, the webcam works reasonably in local areas, but fails to clearly capture LPs on highways (farther distance between two cars and higher car speed).

Figure 5.3.3.1.1: Logitech Brio webcam [99]

5.3.3.2 Ontario License Plates Collection and Pre-processing

The Ontario LPs are collected at different times of days in the GTA streets. Most weather conditions are good. LPs from different vehicles are manually cropped from recorded videos, and the total number of collected LPs are about 230. By recognizing with human eyes (easy or take some efforts), there are 198 LPs that could be read and they are regarded as “clear” LPs; the rest are either blur or low-quality, and they are regarded as “unclear” LPs. Blurring may have been caused by the webcam shutter speed and exposure settings, distance from the ego-car, or relative motion between vehicles. Low quality may be due to plate surface damage, dust and dirt, light reflection by plate plastic shell, characters’ partial block by plate frame, or environment illumination. Some examples are presented in Figure 5.3.3.2.1, (a) is easy to be read, (b) can be recognized but take some efforts, (c) is blur, and (d) has low quality.



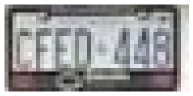
(a) clear (easy) (b) clear (hard) (c) blur (d) low quality

Figure 5.3.3.2.1: Ontario LP samples

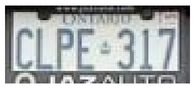
The texts of 198 LPs are manually labeled. In the annotation process, it is worth noting that some separator patterns are between the last alphabet and the first integer in Ontario LPs, such as crown, trillium, and flower shapes. Thus, the optimized CRNN model needs to be fine-tuned using Ontario LPs and the training/testing data split ratio is set as about 60% (117 LPs) and 40% (81 LPs).

5.3.3.3 Results and Discussions

For the testing, the prediction accuracy is 93.83% (76 out of 81 LPs are recognized) before adding the classifier, and that is increased to 97.53% (79 out of 81 LPs are recognized) after adding the classifier. Therefore, the Ontario LP recognition meets the goal setting of high performance. The recognition of two LPs has errors that misidentify E as F, and these plate images are shown in Figure 5.3.3.3.1. In (a), the bottom line of E is partially blocked by the thick plate frame. In (b), the error may be caused by inefficient training of the model with small sized data.



(a) Error 1



(b) Error 2

Figure 5.3.3.3.1: Ontario LP recognition errors

From the license plate recognition application on Ontario LPs, it turns out that the recognition accuracy is not only determined by a deep learning algorithm, but also camera device properties and its setup. The camera settings have significant impacts on license plate quality in the data collection process, and image quality further affects the model recognition performance. To gather better quality LPs in both urban and highway regions, a 4K (3840 x 2160 pixels) resolution is needed. Also, higher frame rates (greater than 30 FPS) can be used for video recording to examine if the number of blur images are reduced. In addition, the manual settings of camera's focus and exposure in the driving process lead to challenges for the collected data quality optimization.

Chapter 6

Lane Detection

Lane detection is to extract lanes information (e.g. coordinates, types) from roads using a camera and assist vehicle trajectory decisions such as lane keeping and lane changing. Real-time lane detection is very important to both autonomous driving and Advanced Driver-Assistance Systems (ADAS). As presented in Chapter 3 Section 4, segmentation-based deep learning algorithms are the most commonly used models at present and some of them have complete open-source codes available online that can be used and implemented in different applications. In addition, some popular datasets are reviewed such as TuSimple (collected on highways in San Diego, US), CULane (collected in urban areas, rural areas, highways from Beijing, China), and BDD100K (collected in urban areas from multiple cities, US).

This research focuses on highway lane detection. Therefore, a real-time algorithm [100], which consists of a segmentation-based CNN and a curve interpolation model, is selected and applied on TuSimple highway lane dataset [78]. Also, transfer learning is used with pre-trained weights from CULane dataset [64]. One NVIDIA GeForce

RTX 2080 Ti GPU is used for training, validation and testing. In this chapter, the algorithm is explained and optimized by modifying the loss function in the CNN and removing outliers before interpolating lane curves. TuSimple data pre-processing is illustrated. The lane detection results of all experiments are evaluated and discussed.

6.1 Algorithm Explanation

In this section, the architecture of the applied deep learning algorithm is explained. It includes a segmentation-based CNN and a curve interpolation model. The CNN is a variant of ERFNet [74] for the lane segmentation, and cubic spline curves are used for the lane interpolation.

6.1.1 Segmentation-based CNN

Before illustrating the details of the CNN, a visualization of how segmentation works is represented with an example. In Figure 6.1.1.1, the input image contains 2 classes of objects: cat and blanket. The "cat" class is labeled as 1 and "blanket" class is labeled as 2. By feeding the input image into a segmentation-based CNN, it generates two feature maps for classes prediction. In each class feature map, a binary classification is calculated to recognize if pixels belong to the class or not (i.e. 1 or 0). After that, all 1 in each map are replaced by the labelling of the class, and all feature maps are overlaid together to generate one segmentation mask on top of the input image. In the segmentation mask, pixels for each class are assigned with a specific color, in this example, red color represents cat and blue color represents blanket.

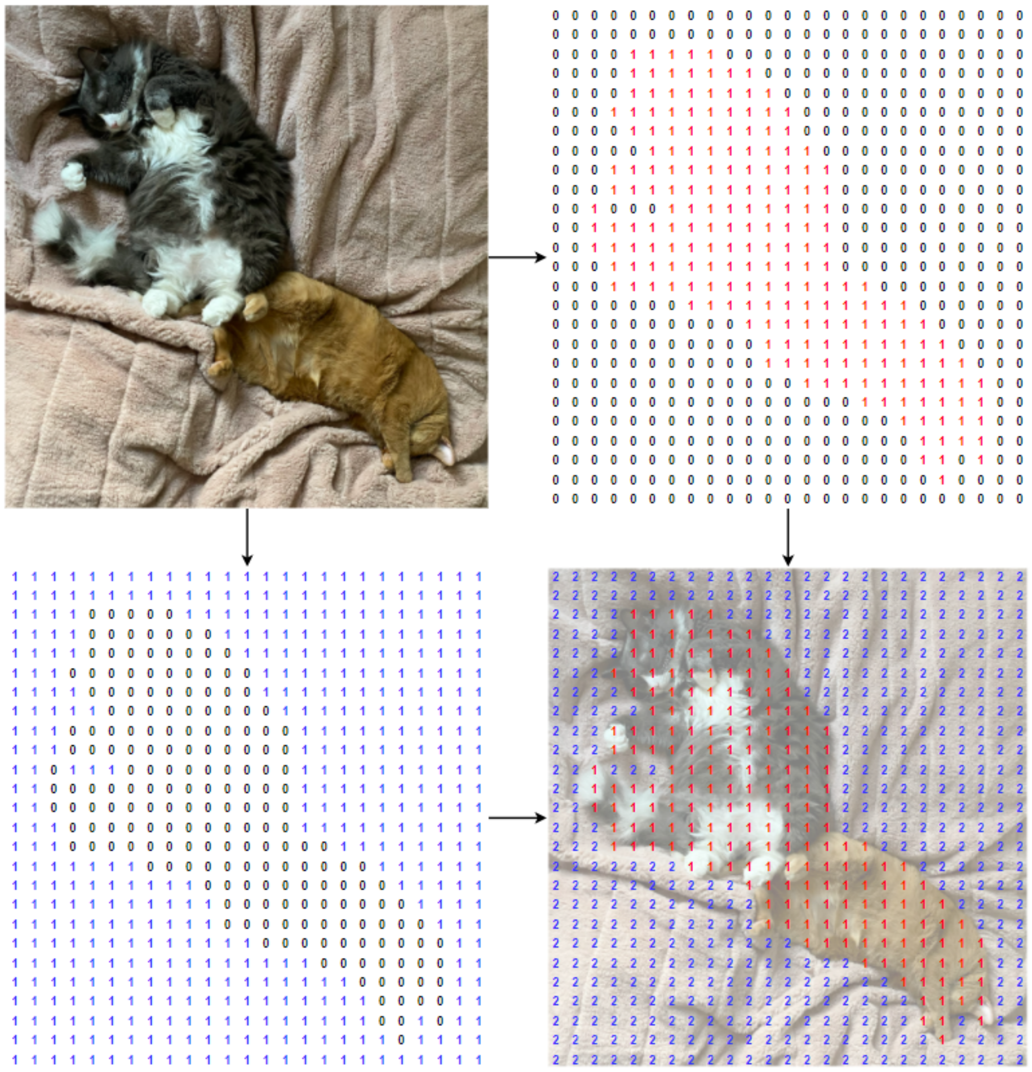


Figure 6.1.1.1: Segmentation visualization example

The CNN is a variant of Efficient Residual Factorized ConvNet (ERFNet) [74]. ERFNet is a real-time segmentation CNN with an encoder-decoder architecture which is presented in Table 6.1.1.1. In brief, the encoder learns more detailed features from input images via generating lower resolution feature maps, but it loses spatial information during the down-sampling process. Then the decoder recovers spatial information by up-sampling and outputs full resolution segmentation maps.

Table 6.1.1.1: ERFNet architecture [74]

	Layer	**Type**	**Number of feature maps**	**Output resolution**
Encoder	1	Downsampler block	16	W/2 x H/2
	2	Downsampler block	64	W/4 x H/4
	3-7	5 x Non-bt-1D	64	W/4 x H/4
	8	Downsampler block	128	W/8 x H/8
	9	Non-bt-1D (dilated 2)	128	W/8 x H/8
	10	Non-bt-1D (dilated 4)	128	W/8 x H/8
	11	Non-bt-1D (dilated 8)	128	W/8 x H/8
	12	Non-bt-1D (dilated 16)	128	W/8 x H/8
	13	Non-bt-1D (dilated 2)	128	W/8 x H/8
	14	Non-bt-1D (dilated 4)	128	W/8 x H/8
	15	Non-bt-1D (dilated 8)	128	W/8 x H/8
	16	Non-bt-1D (dilated 16)	128	W/8 x H/8
Decoder	17	Deconvolution (up-sampling)	64	W/4 x H/4
	18-19	2 x Non-bt-1D	64	W/4 x H/4
	20	Deconvolution (up-sampling)	16	W/2 x H/2
	21-22	2 x Non-bt-1D	16	W/2 x H/2
	23	Deconvolution (up-sampling)	C	W x H

From Table 6.1.1.1, the encoder has two types of layers: downsampler and non-bottleneck 1D; and the decoder also has two types of layers: deconvolution and non-bottleneck 1D. A downsampler block concatenates one convolutional layer and one max pooling layer to learn more features and reduce feature map size. A non-bottleneck 1D (non-bt 1D) block is a type of residual block, which was discussed in Chapter 4 Section 1, and it is designed to reduce the computation time and the number of parameters. A non-bt 1D includes four convolutional layers with one skip connection to deepen the network and produce higher accuracy. In some non-bt 1D blocks, dilated convolutions are used to collect richer information. The difference between standard and dilated convolutions are that a standard convolution extracts adjacent pixel information (dilation rate = 1) but a dilated convolution extracts pixel information with gaps (dilation rate = n means skipping n-1 pixels). For instance, pixels in yellow color are selected by a 3x3 kernel with dilation rate = 1, 2, 4 are plotted in Figure 6.1.1.2 top, middle, and bottom. The corresponding pixel gaps are 0, 1, 3. A deconvolution (also named transpose convolution) layer is designed to do an opposite operation as a typical convolution layer. It up-samples feature map size and reduces feature map numbers.

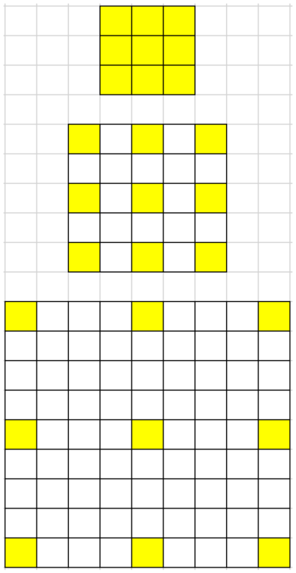


Figure 6.1.1.2: Dilated convolutions with kernel size 3 x 3

In ERFNet, the encoder generates 128 feature maps and reduces their width and height from the input image size (W, H) to (W/8, H/8) using downsampler and non-bottleneck 1D blocks. And the decoder reduces the number of feature maps from 128 to C (number of classes) and recovers output maps' size from (W/8, H/8) to (W, H) via deconvolution layers and non-bottleneck 1D blocks. In the lane detection case, one lane is defined as one class, so C classes represent C - 1 lanes and one background (all things except lanes in an image).

The CNN modifies ERFNet by adding one additional lane existence branch after the encoder of ERFNet and parallel to the decoder of ERFNet, as shown in Figure 6.1.1.3. The lane existence part combines two convolutional, one max pooling, and two linear (fully connected) layers to compute the existence probability (confidence) per lane and classify a lane existence as 0 or 1 with a probability threshold 0.5.

The purpose of adding the lane existence branch is to reduce computation time in the lane curve interpolation stage. For example, if a dataset contains a maximum of 4 lanes in each of its images, the decoder of ERFNet will generate 4 lane feature maps and each of them may contain one lane or nothing. Without the lane existence branch, every feature map needs to be checked by the lane curve interpolation model for lane points extraction. It is time consuming if the dataset size is very large. Therefore, by adding the lane existence part, only feature maps whose lane existence classification is 1 will be passed into the next curve interpolation step.

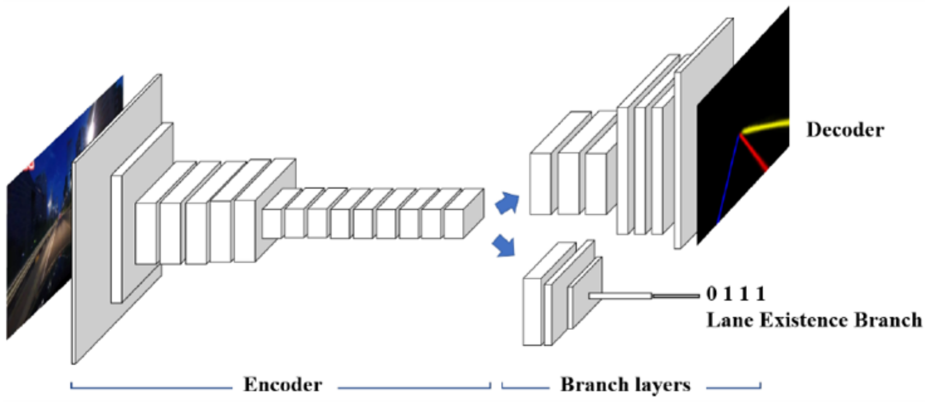


Figure 6.1.1.3: ERFNet variant [101]

6.1.2 Curve Interpolation

After the segmentation-based CNN, lane points' coordinates are extracted from segmentation feature maps and interpolated using cubic spline curves in the interpolation model of the algorithm. The cubic spline function S is as follows:

$$S(x) = \begin{cases} C_1(x), & x_0 \leq x \leq x_1 \\ ... & \\ C_i(x), & x_{i-1} < x \leq x_i \\ ... & \\ C_n(x), & x_{n-1} < x \leq x_n \end{cases}$$

where x_i (also named knot) represents a predefined point on S, $i = 0, ..., n$; each cubic function between two knots is $C_i = a_i + b_i x + c_i x^2 + d_i x^3$ $(d_i \neq 0)$, $i = 1, ..., n$. Except two end knots on S, each knot connects two adjacent cubic functions and satisfies boundary conditions $C_i(x_i) = C_{i+1}(x_i)$, $C_i^{'}(x_i) = C_{i+1}^{'}(x_i)$, and $C_i^{''}(x_i) = C_{i+1}^{''}(x_i)$, $i = 1, ..., n-1$. The boundary conditions indicate that two adjacent cubic functions at their connect point have the same value, slope (1^{st} derivative), and rate of change of slope (2^{nd} derivative).

In order to obtain the lane curve equations and visualize the lanes on roads, the cubic spline end condition for lane end points needs to be determined. There are three common end types for a spline function S: natural, clamped, and not-a-knot.

In the natural end condition, the change of the slope of S at its start and end is constrained as 0: $S_{start}^{''} = 0$, $S_{end}^{''} = 0$. It means that S turns into straight lines at the two ends. In the clamped end condition, the slope of S in both ends is specified as a constant: $S_{start}^{'} = A$, $S_{end}^{'} = B$. In the not-a-knot end condition, instead of adding constrains like natural and clamped, the number of degrees of freedom is reduced by removing the second knot (after the start) and the second last knot (before the end) on S: $S'''_{start} = S'''_{start+1}$, $S'''_{end-1} = S'''_{end}$. From this end condition plus previous boundary conditions, it can be derived that C_1 is the same as C_2, and C_{n-1} is the same as C_n. So the second and the second last knots can be "ignored" in the coefficients computation of S and they become "not-a-knot".

Figure 6.1.2.1 is a visualization example for cubic spline curve interpolation with natural, clamped, and not-a-knot end conditions. By observing all lane plotting forms in this research, the natural end type generates the best interpolation performance. Therefore, cubic spline with the natural end condition is selected for the lane interpolation model.

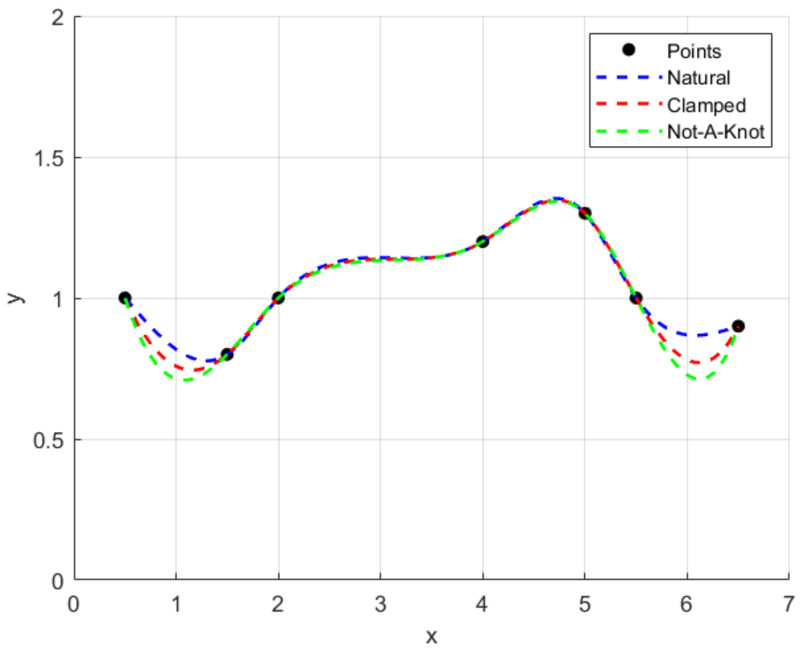


Figure 6.1.2.1: Cubic spline boundary conditions

6.2 TuSimple Data with Transfer Learning

TuSimple dataset is used in this project and transfer learning is also applied with a pre-trained model from CULane dataset. Both datasets have been introduced in Chapter 3 Section 4.2. In this section, the difference between the two datasets in terms of data collection environments, scenes, annotations, and how TuSimple data is pre-processed are discussed.

6.2.1 CULane Pre-trained Model

The pre-trained model of the algorithm is generated using CULane, which has a much larger dataset and more data collection environment scenarios in urban, rural and highway areas, compared with TuSimple. In Figure 6.2.1.1, a total of nine scenes with their proportions and example images are shown: normal, crowded, night, no line, shadow, arrow, dazzle light, curve, and crossroad.

In terms of data annotation, the origin (x_0, y_0) is the left-top of an image, x axis is along the horizontal direction from left to right, and y axis is along the vertical direction from top to bottom. CULane labels lane coordinates (x, y) consecutively without considering occluded locations. In addition, the end points of all lanes in one image are labeled at the same y value. In other words, all lane end points in one image are labeled equally far from the ego-car. The additional lane existence (0 or 1) labelling for CULane is based on the relative position between lanes and the ego-car (second to left, left, right, second to right).

The reason why some annotation features of CULane are described here is that they affect the fine-tuning of the deep learning algorithm, lane detection and evaluation on TuSimple. These will be discussed in the "Evaluation and discussions" section.

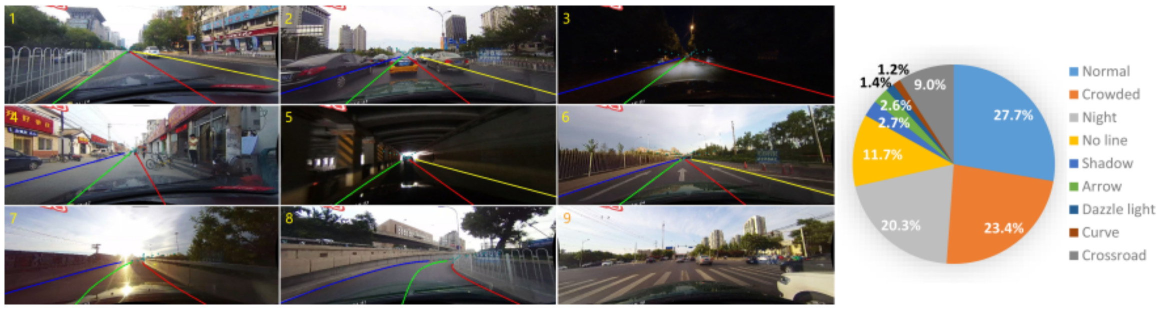


Figure 6.2.1.1: CULane scenes example [64]

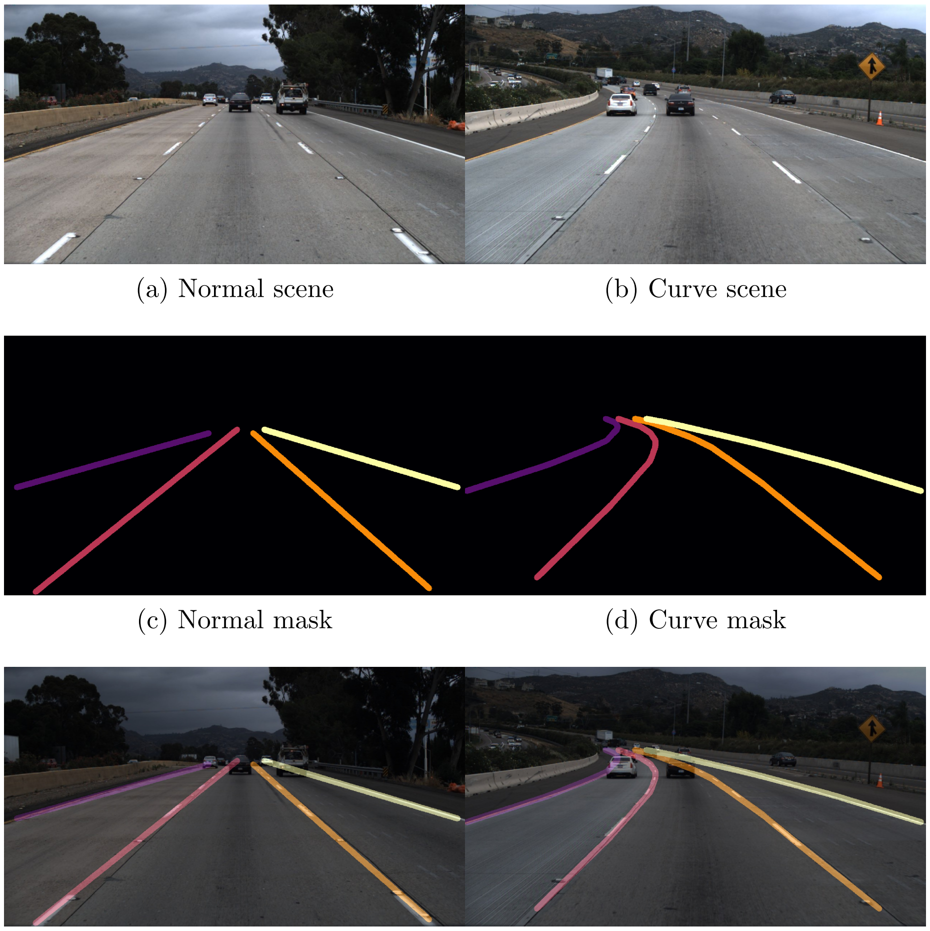
6.2.2 TuSimple Pre-processing

Compared with CULane, TuSimple has a much smaller data size and is collected only on US highways in day times. The dominant scene types are normal (Figure 6.2.2.1a) and curve (Figure 6.2.2.1b). A few of image scene types are shadowed when the ego-car is driven under bridges or in crowded conditions during rush hours. Thus, the pre-trained model needs to be fine-tuned using TuSimple data.

For data annotation, TuSimple also labels the origin (x_0, y_0) at the left-top of an image, x axis is from left to right horizontally, and y axis is from top to bottom vertically. TuSimple considers the occlusion of lanes by vehicles and does not label coordinates if a part of a lane is occluded. In addition, lane end points in one image are labeled at different y values depending on how far lanes can be visualized per image, rather than selecting a fixed y value as the end labeled point for all lanes per image in CULane. Moreover, as a part of this research, the lane existence annotation is manually added for TuSimple based on the absolute position of lanes from left to right on roads, rather than the relative position between lanes and the ego-car.

The ground truth masks of TuSimple are annotated for the error computation of predicted segmentation masks from the decoder in the algorithm. For example, the ground truth masks of normal and curve scenes are created in Figures 6.2.2.1c and 6.2.2.1d. To generate ground truth masks, lanes are plotted with different colors and non-lane areas (i.e. background) are plotted with black color. Firstly, the original image's width (W) and height (H) information are extracted and one zero matrix is created with shape (W x H x 3), where 3 refers to red, green, and blue channels. Black color is encoded as (0, 0, 0) in the RGB scheme. Thus, this zero matrix represents a black image. Next, lane point coordinates are extracted from annotation files and plotted with different colors. Finally, the matrix data is displayed as an image. Figures 6.2.2.1e and 6.2.2.1f show the overlays between original images and ground truth masks.

The number of lanes per image in CULane is less than or equal to 4. Most images in TuSimple have at most 4 lanes, except a very few images that have 5 lanes. Thus, images which are annotated and have at most 4 lanes in TuSimple will be filtered out for use in this research. The number of training/validation/testing images are set as 2858/358/172. In addition, TuSimple images are resized from 1280 x 720 pixels to 1640 x 590 pixels as CULane.



(a) Normal scene (b) Curve scene

(c) Normal mask (d) Curve mask

(e) Normal overlay (f) Curve overlay

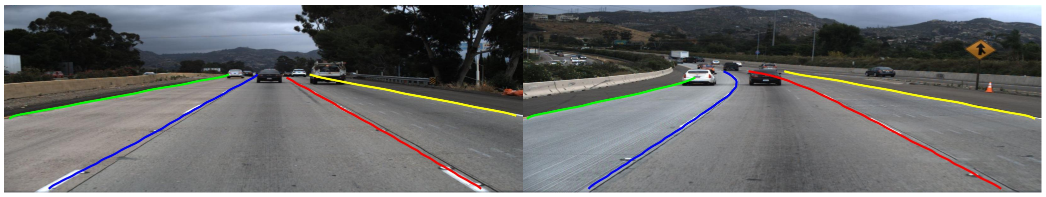
Figure 6.2.2.1: TuSimple data scenes and annotations

6.3 Experiments and Optimization

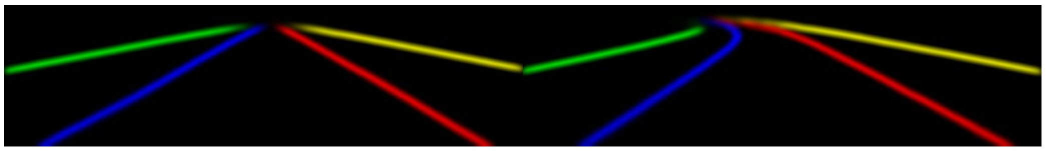
In this section, a total three of experiments are designed: the first is to fine tune the original model using TuSimple data; the second is to optimize the lane segmentation by changing the original algorithm's loss function; and, the third is to optimize the lane interpolation by removing outliers from predicted lane points before interpolating lanes. The optimization experiments are discussed in subsections 6.3.1 and 6.3.2.

In the first experiment, all hyperparameters in the deep learning algorithm are kept as default, except adjusting the initial learning rate lr and adding a learning rate scheduler. The hyperparameter definition and types have been summarized in Chapter 4 Section 3.1. The initial lr is set to 0.01 to accelerate the algorithm learning speed, and the lr scheduler adjusts the learning speed by reducing lr by a factor of 10 when the number of epochs reaches at 20%, 50%, 90% of the total number of epochs. These values are manually selected by observing the loss and accuracy convergence performance in the training process.

Figure 6.3.1 visualizes the lane segmentation prediction and interpolation results of the first experiment in normal and curve scenes. The lane segmentation masks are outputs of the segmentation-based CNN of the algorithm, and lane point coordinates are extracted from the masks according to pixel values. After that, lane points are interpolated as cubic splines onto the original lane images.



(a) Normal scene interpolation (b) Curve scene interpolation



(c) Normal mask prediction (d) Curve mask prediction

Figure 6.3.1: Lane segmentation predictions and interpolations

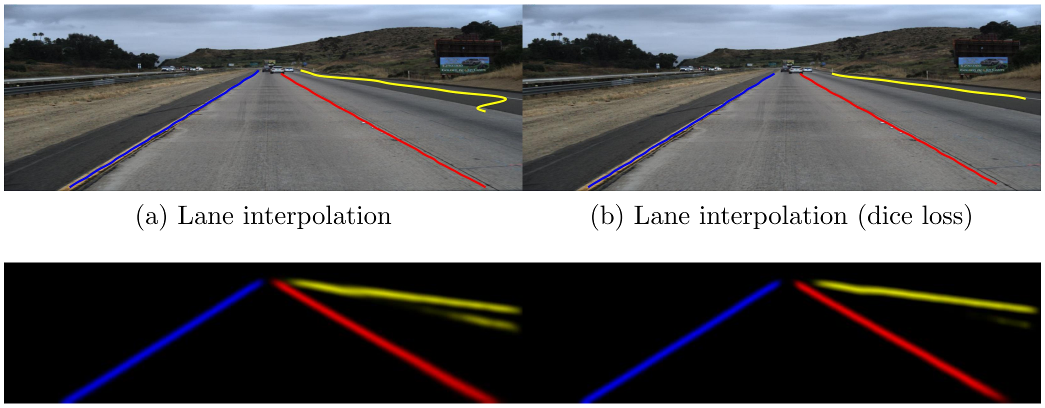
6.3.1 Loss Function Optimization

The original loss function for lane segmentation in the algorithm is the Negative Log-Likelihood (NLL) loss. It is used for classification tasks with multiple classes. NLL loss calculates the negative log of softmax of the predicted output $NLLloss = -log(\sigma(y_i))$. The softmax function $\sigma(y_i)$ re-scales output to the range of [0, 1]. As a type of distribution-based losses, NLL loss measures classification for each pixel, so it has a disadvantage when classes are extremely unbalanced. In case of the lane detection, as shown in Figure 6.3.1, the pixel amount of background class (black color area) is much larger than that of lane classes (colored lines), so the class imbalance exists in this research.

Therefore, region-based losses are considered in this case, which only maximize the overlap between predicted and ground truth lanes. One popular loss function named tversky loss is selected to be used here. It is equal to $\frac{TP}{TP\ +\ \alpha \times FP\ +\ \beta \times FN}$. [TP, FP, FN] refer to [True Positive, False Positive, False Negative], which were introduced in Chapter 3 Section 2.3. The hyperparameters α and β are used to penalize FP and FN, and $\alpha + \beta$ is equal to 1. When $\alpha = \beta = 0.5$, the tversky loss becomes equivalent to another loss function named dice loss, which gives equal weights for FP and FN.

To optimize lane segmentation, the original NLL loss is combined with the tversky loss as a compound loss, and 9 sub-experiments are conducted with [α, β] at [0.1, 0.9], [0.2, 0.8], [0.3, 0.7], [0.4, 0.6], [0.5, 0.5], [0.6, 0.4], [0.7, 0.3], [0.8, 0.2], [0.9, 0.1]. Different combinations of α and β give different weights for FP and FN in the tversky loss, so they relate to the trade-off between precision and recall. The testing result performs the best when [α, β] is [0.5, 0.5] (i.e. the tversky loss becomes the dice loss).

As an example, in Figure 6.3.1.1, before changing the algorithm loss function, the right lane (shown in yellow color) in the predicted segmentation mask has a significant pixel classification error and a part of pixels in background are classified as lane. The error from lane segmentation also reduces the accuracy of lane interpolation, because incorrect lane points are extracted from the predicted lane mask for interpolation. After applying the compound loss, the right lane is segmented from background better, and the lane interpolation performance is also improved.

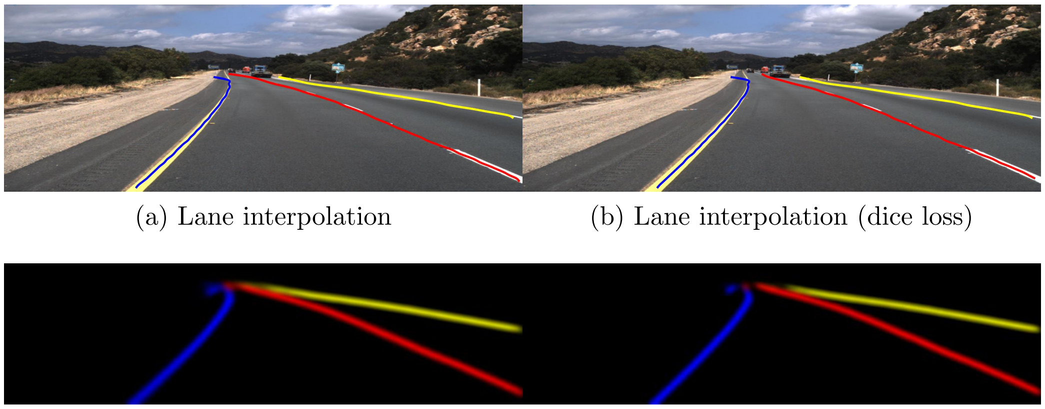

(a) Lane interpolation (b) Lane interpolation (dice loss)

(c) Lane predicted mask (d) Lane predicted mask (dice loss)

Figure 6.3.1.1: Before and after adding the dice loss

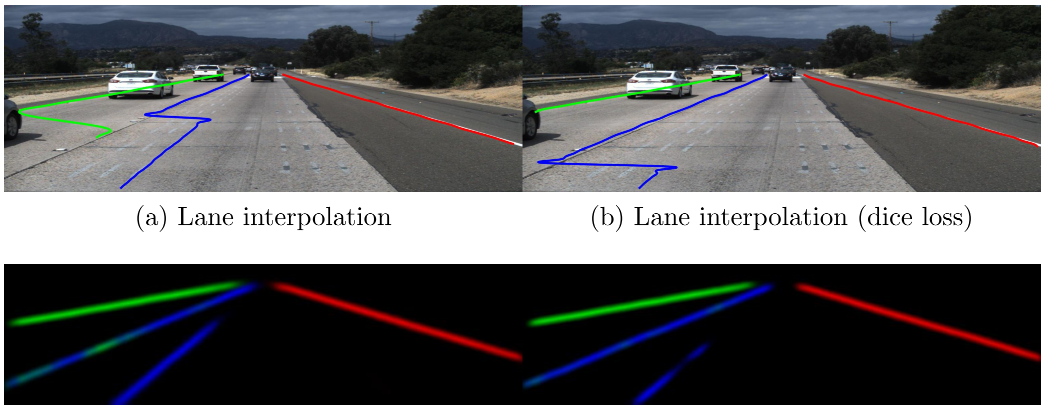
Even though the compound loss has been applied, two notable issues can still be observed. The first problem is that in some curve lanes, lane ends segmentation do not perform well. For instance, in Figure 6.3.1.2, after combining the dice loss with the NLL loss, the end of the middle lane (red color) has better segmentation, but the end of the left lane (blue color) still diverges from the correct trajectory. The next problem is that when some textures are painted on roads, such as former and unused lanes, the algorithm may miss-classify them as part of lanes. For example, in Figure 6.3.1.3, the left lane (green color) segmentation is improved by the compound loss, but the middle lane (blue color) still contains miss-segments due to white dashed textures on the road.

To alleviate these issues, points outside of the lane end in problem one and points that belong to road surface textures in problem two need to be removed in the lane post-processing after segmentation. They can be regarded as "outliers" in lane points. Thus, a method that removes outliers is added before the lane curve interpolation stage as follows.


(a) Lane interpolation (b) Lane interpolation (dice loss)

(c) Lane predicted mask (d) Lane predicted mask (dice loss)

Figure 6.3.1.2: Problem 1: lane ends segmentation


(a) Lane interpolation (b) Lane interpolation (dice loss)

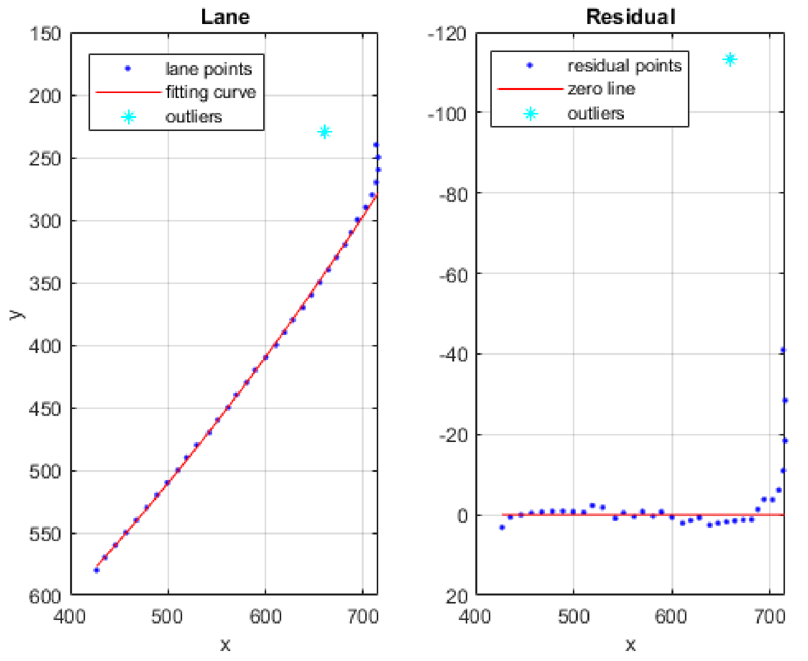
(c) Lane predicted mask (d) Lane predicted mask (dice loss)

Figure 6.3.1.3: Problem 2: textures on roads

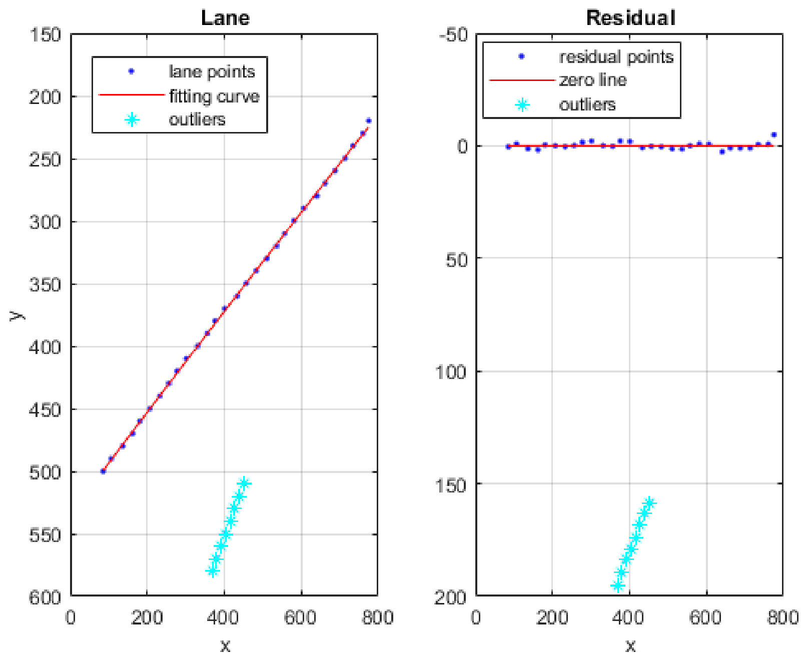
6.3.2 Outliers Optimization

The strategy is to fit a curve on lane points without being significantly effected by outliers, and then remove points which far away from the fitted curve. At the beginning, a quadratic polynomial $ax^2 + bx + c$ is selected for lane points fitting because TuSimple lanes are mainly straight or curved. Note that the least square fitting technique is sensitive to outliers, as these extreme points generate much larger square residuals in the error calculation and impact the coefficients of fitted curve. A robust fitting method called "bisquare weights" is applied to reduce the effect of outliers by minimizing a weighted sum of square residuals. To be specific, points nearer to the fitted curve are assigned more weights and points farther from the fitted curve are assigned less weights. Note that outliers can also result in excessive curvature in curve fitting, for instance, a straight lane could be "bent" by outliers in fitting process. To avoid this problem, the maximum absolute value of the coefficient a for the second degree term x^2 is constrained. Finally, the residual per point is calculated and points are regarded as outliers if their residuals are larger than a predefined threshold.

By observing lanes curvature and fitting performance in all testing data, the maximum absolute value of the coefficient a for the second degree term x^2 in the quadratic polynomial is manually set as 5, and the predefined residual threshold is also manually set as 60. Figure 6.3.2.1 represents the lane points, curve fitting, residual distribution, and outliers for the blue colored lanes in problem 1 and 2 as discussed in previous subsection. After removing the outliers using this method, subsequent lane interpolations in problem 1 and 2 are improved, as shown in Figures 6.3.2.2 and 6.3.2.3.



(a) The left lane in problem 1



(b) The middle lane in problem 2

Figure 6.3.2.1: Lane points fitting and residuals

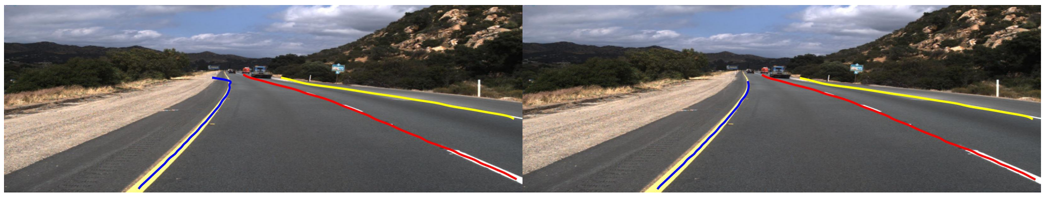

(a) Lane interpolation after adding dice loss (b) Lane interpolation after removing outliers

Figure 6.3.2.2: Lane detection problem 1 before and after removing outliers

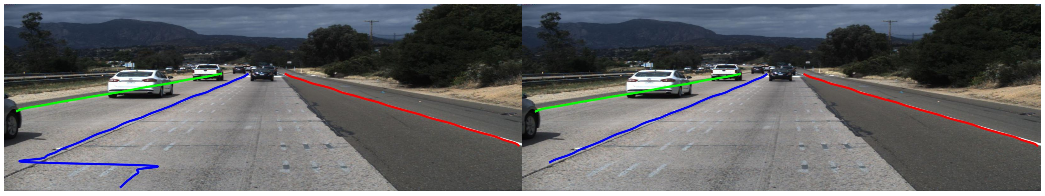

(a) Lane interpolation after adding dice loss (b) Lane interpolation after removing outliers

Figure 6.3.2.3: Lane detection problem 2 before and after removing outliers

6.4 Evaluation and Discussions

In the testing process, lanes are segmented at around 80 FPS with one NVIDIA GeForce RTX 2080 Ti GPU and therefore the algorithm could be implemented in real-time lane detection.

A lane detection evaluation benchmark designed for CULane is used to assess the lane detection performance for TuSimple. It contains precision, recall, and F1 score. All experimental results are presented in Table 6.4.1. For comparison, Table 6.4.2 lists testing results from other published algorithms as follows.

Spatial CNN (SCNN) [64], as reviewed in Chapter 3 Section 4.1, learns features along an input image width (rightward/leftward) and height (downward/upward) to increase the spatial information of lanes and outputs lane segmentation masks. It then interpolates lane points using cubic splines. Efficient Neural Network - Self

Attention Distillation (ENet-SAD) [67] adds attention to ENet to enforce the network learning ability for segmentation. It also uses cubic splines for lane interpolation. ResNet-34 [102] divides an input image into grid cells, trains the ResNet-34 for feature extraction and a classifier for lane location prediction in each cell. ResNet-18 [102] is a light weight version based on the same method. PolyLaneNet [103] regards lanes as polynomial curves and lane detection as a polynomial regression problem, it directly predicts the polynomial coefficients for lanes.

Table 6.4.1: TuSimple lane detection testing results

Experiments	**Precision**	**Recall**	**F1 score**
Original model fine-tuning	92.08%	89.77%	**90.91%**
Tversky loss with α 0.1 and β 0.9	91.77%	89.77%	90.75%
Tversky loss with α 0.2 and β 0.8	91.93%	89.93%	90.92%
Tversky loss with α 0.3 and β 0.7	91.79%	90.10%	90.94%
Tversky loss with α 0.4 and β 0.6	91.30%	89.77%	90.52%
Tversky loss with α 0.5 and β 0.5 (dice loss)	92.12%	90.27%	**91.19%**
Tversky loss with α 0.6 and β 0.4	91.94%	89.93%	90.92%
Tversky loss with α 0.7 and β 0.3	92.40%	89.77%	91.06%
Tversky loss with α 0.8 and β 0.2	91.75%	89.60%	90.66%
Tversky loss with α 0.9 and β 0.1	91.79%	90.10%	90.94%
Outliers removing	92.75%	90.10%	**91.40%**

According to Table 6.4.1, the compound loss (NLL + tversky) improves the lane segmentation from ERFNet by increasing F1 score from 90.91% to maximum 91.19% when [α, β] is [0.5, 0.5]. And removing outliers from predicted lanes points improves the lane interpolation by increasing F1 score from 91.19% to 91.40%.

Table 6.4.2: State-of-the-art results on TuSimple with available source code [104]

Algorithms	**F1 score**	**FPS**
SCNN (2018)	95.97%	7.5
ENet-SAD (2019)	95.92%	75.0
ResNet-18 (2020)	87.87%	312.5
ResNet-34 (2020)	88.02%	169.5
PolyLaneNet (2020)	90.62%	115.0

Comparing the F1 score and the prediction speed FPS with the state-of-the-art algorithms in Table 6.4.2, the lane detection result (91.40% F1 score and 80 FPS) obtained in this research are comparable. ENet-SAD has the best performance when considering both accuracy and speed.

The challenges and limitations of the deep learning algorithm associated with this research should be noted. In terms of FP, TuSimple ground truth does not label occluded lane points but occluded lane points are predicted in lane detection. This may affect IoU values and further affect FP. The difference between lane end points in ground truth and prediction also affects IoU calculation and FP. Moreover, road boundaries or boundaries between different colored road surfaces result in lane misclassification and FP increasing. In terms of FN, the pre-trained model from CULane only labels a maximum of two lanes on two sides of the ego-car. It would miss one lane prediction if a car is driven on the rightmost (leftmost) of a 4-lane highway and its left (right) side has three lanes. Thus, the lane existence labelling method is changed from the relative position between lanes and the ego-car to the absolute position of lanes on roads in TuSimple pre-processing. But the pre-trained model is hard to be fine-tuned and this still results in lanes missing and FN increasing. In addition, unclear painted lanes and heavily occluded lanes also increase FN.

Chapter 7

Conclusion and Future Work

This chapter provides the conclusions on this research about camera-based deep learning algorithms with transfer learning for 2D object detection, license plate detection and recognition, and highway lane detection. In addition, future research directions are discussed and recommended.

7.1 Conclusion

As overviewed, autonomous vehicles are inevitable elements of our future world. Their benefits include the reduction in traffic crashes and congestion, and in turn the reduction of fuel consumption and carbon emissions. They also provide greater flexibility for elderly and people with disabilities. To sense and understand the surrounding environment as the first stage of autonomous driving, the perception system uses different sensors independently or cooperatively for information collection. Monocular cameras are commonly used and the cheapest and the smallest sensors with the most mature technology that mimics one eye of human vision to collect data. The most important computer vision tasks in perception include object detection/tracking, license plate recognition, and lane/road detection. Deep learning and transfer learning techniques are increasingly being used with advanced sensors to improve the perception accuracy in real time. Therefore, this thesis focuses on applying and optimizing camera-based deep learning algorithms with transfer learning in relation to: 2D object detection, license plate detection and recognition, and highway lane detection.

For the 2D object detection, the YOLOv3 algorithm variants that are written using Keras and PyTorch libraries are implemented on the KITTI dataset for car, cyclist, and pedestrian detection. In terms of the detection speed, Keras and PyTorch YOLOv3 achieve about 28 and 27 FPS using one NVIDIA GeForce RTX 2080 Ti GPU. Therefore, they are able to approach real-time operation (approximately 30 FPS). In terms of the detection performance, PyTorch YOLOv3 outperforms Keras YOLOv3. It may be due to three techniques that added in the PyTorch YOLOv3 architecture: initial hyperparameters optimization using a genetic algorithm instead of manually adjustment by trial and error; GIoU loss replaces SE loss in the loss function to improve bounding box localization; and, the mosaic technique in data augmentation is applied to reinforce object classification. In addition, the algorithm training with transfer learning has distinctly better performance than that without transfer learning. It represents that the transfer learning technique boosts the object detection performance. Also, the detection without severe object occlusion and truncation is better than that with heavy occlusion and truncation. It reflects that severe occlusion and truncation conditions significantly affect the detection performance. Furthermore, KITTI data bias, namely, more cars than cyclists and pedestrians in images, results in better prediction of cars than that of other object classes.

For the license plate detection and recognition, multiple datasets (AOLP, OpenALPR, UFPR-ALPR, Stanford Cars) are pre-processed for use. In the detection stage, the PyTorch YOLOv3 model is implemented and the detection achieves 86% average precision. Small object detection (e.g. license plates far from the ego-car) is still a challenge for deep learning algorithms. In addition, logo prints on vehicle surfaces and traffic occlusion also impact the detection performance. In the recognition stage, the Tesseract OCR engine, CRNN (with transfer learning), and attention OCR (without transfer learning) algorithms are explored and compared: their recognition [accuracy, speed] are [9.43%, NA], [87.74%, 72 FPS], and [70.75%, 39 FPS], respectively. The CRNN model produces the best recognition accuracy in real-time operation. It is optimized by augmenting images via perspective skewing and distortion (accuracy is increased to 95.28%), adjusting input image resized width (accuracy is increased to 97.17%), and adding a post-processing classifier to switch confused alphabets and integers in License Plates (LPs). By applying the optimized model on Ontario LP recognition, the accuracy achieves 97.53%. In Ontario LP collection process, the monocular camera properties and its setup have significant impacts on LPs quality, and in turn affect the final LP recognition performance.

For the highway lane detection, a variant of the ERFNet algorithm with a cubic spline interpolation model is applied on TuSimple dataset for the highway lane segmentation and interpolation. The pre-trained weights from CULane dataset is used for transfer learning. The lane segments are predicted at about 80 FPS and F1 score achieves 90.91%. To improve the lane segmentation, the original NLL loss function is replaced by a compound loss function of NLL and tversky losses, and F1 score is increased to 91.19%. To improve the lane interpolation, an outlier removing strategy that filters out incorrect segmented lane points is applied before interpolating lanes, and F1 score is increased to 91.40%. The detection performance and speed of the optimized model are comparable with other state-of-the-art algorithms. According to the observations, unclear lane painting, lane occlusions by vehicles, and road surface textures lead to challenges for the lane prediction. Furthermore, the annotation difference between TuSimple and CULane datasets results in the difficulty of fine-tuning the pre-trained weights in training process.

Overall, camera sensors play a vital role in the perception system of autonomous vehicles. And camera-based deep learning algorithms with transfer learning work well in real time for the 2D object detection, license plate detection and recognition, and highway lane detection tasks in autonomous driving research. Current challenges include deficiencies of image datasets and algorithms design, complex road environment, and harsh weather/illumination conditions.

7.2 Future Work

For the 2D objection detection, three components are considered. The first one is the data bias in the dataset which contains much more cars than cyclists and pedestrians. Objects in the minority classes need to be collected more to improve the detection performance. The second one is the K-means algorithm for anchors generation. K-mean is sensitive to the initial selection of anchor sizes (widths and heights), and it produces different anchors every time it runs. The anchor sizes can be calculated in future by averaging multiple runs to increase anchors' robustness. The third one is the genetic algorithm for hyperparameters optimization. It requires a large number of iterations and high computation power for training. Alternative strategies or parallel computing implements that are more efficient for optimization should be considered.

For the license plate recognition, the CRNN model with transfer learning has achieved a high recognition accuracy. One that should be further investigated is the decoding approach of CTC. The greedy search is a simple and fast method that picks only one class with the highest probability per time step in the decoding process. However, the most likely class may not be the ground truth class. Thus, other methods such as beam search and prefix beam search [105] can be implemented and compared with the greedy search in terms of the LP recognition accuracy and prediction speed. Besides the deep learning algorithm, larger Ontario license plate dataset is needed most to assure the reliability of recognition accuracy. To create a large dataset in both urban and highway regions, better camera settings are required. The recommend industrial camera properties are 4K (3840 x 2160 pixels) resolution and at least 30 FPS recording speed.

For the highway lane detection, the major cause of errors is the deep learning algorithm architecture deficiency. The ERFNet segmentation is an encoder-decoder structure, its encoder learns and produces lower resolution feature maps while down-sampling input images. Only feature maps from the last layer in the encoder is sent to the decoder for full resolution segmentation maps generation by up-sampling. The final encoding features are not able to store all information of input images, so part of the information is lost in the encoding process. To address this problem, attention techniques should be inserted. In addition, highway exit lanes have much worse detection performance than normal lanes. Highway exits rarely appear in the dataset, and some of them are not annotated. Therefore, highway exits could be collected and labeled more to strengthen the detection capability. Moreover, current lane detection is based on image training, but lanes are continuous objects in time series. In future, video-based training would be worth investigating because videos contain the temporal information for lanes.

Furthermore, as an emerging research direction in computer vision, $transformer$ is briefly mentioned here. The transformer neural network was initially invented by Google researchers in 2017 [106]. Its architecture could purely rely on attention mechanisms without recurrence or convolutions. Transformers have been successfully applied for natural language processing and are an order of magnitude faster than RNN. This technique in computer vision research has also produced some remarkable outcomes since 2020. For example, the DEtection TRansformer (DETR) proposed by Facebook AI in 2020 [107] successfully combined CNN and transformer for object detection and segmentation applications. The model can pay attention to the extremities of objects to accurately localize bounding boxes. Extending to the LPDR and lane detection research, this ability of the transformer can also be leveraged to focus on license plate character contours and highway lane boundaries to optimize the training results. Therefore, the transformer technique is recommended as a new research direction for camera-based perception tasks.

Finally, AI technique safety is crucial for the safety-critical perception in autonomous vehicles. Methods that mitigate AI safety issues need to be considered as future work to ensure that perception models are robust and reliable as human sensing. For instance, to reduce algorithm processing latency between object detection and lane detection tasks, the two deep learning algorithms can share the same backbone for feature extraction.

Appendix A

Object Detection Supplementary

A.1 PyTorch YOLOv3 Hyperparameters

Figures A.1.1 and A.1.2 show PyTorch YOLOv3 hyperparameters optimization generations in the two experiments. The horizontal axis represents the value of hyperparameters and the vertical axis represents the fitness score of all iterations. The value of each hyperparameter is determined at the highest fitness score. The hyperparameters used in this object detection research are listed in Table 4.3.2.1.

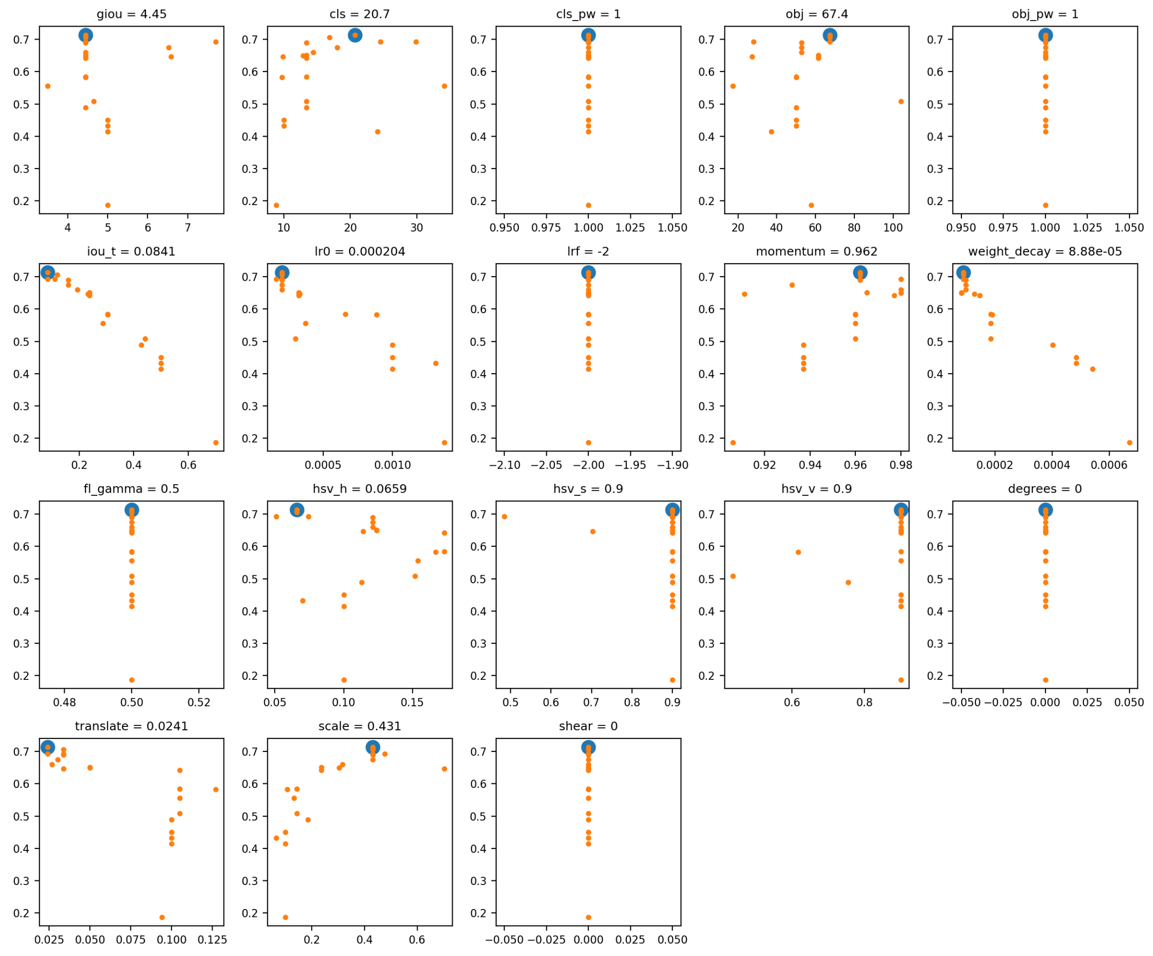


Figure A.1.1: Hyperparameters optimization in experiment 1

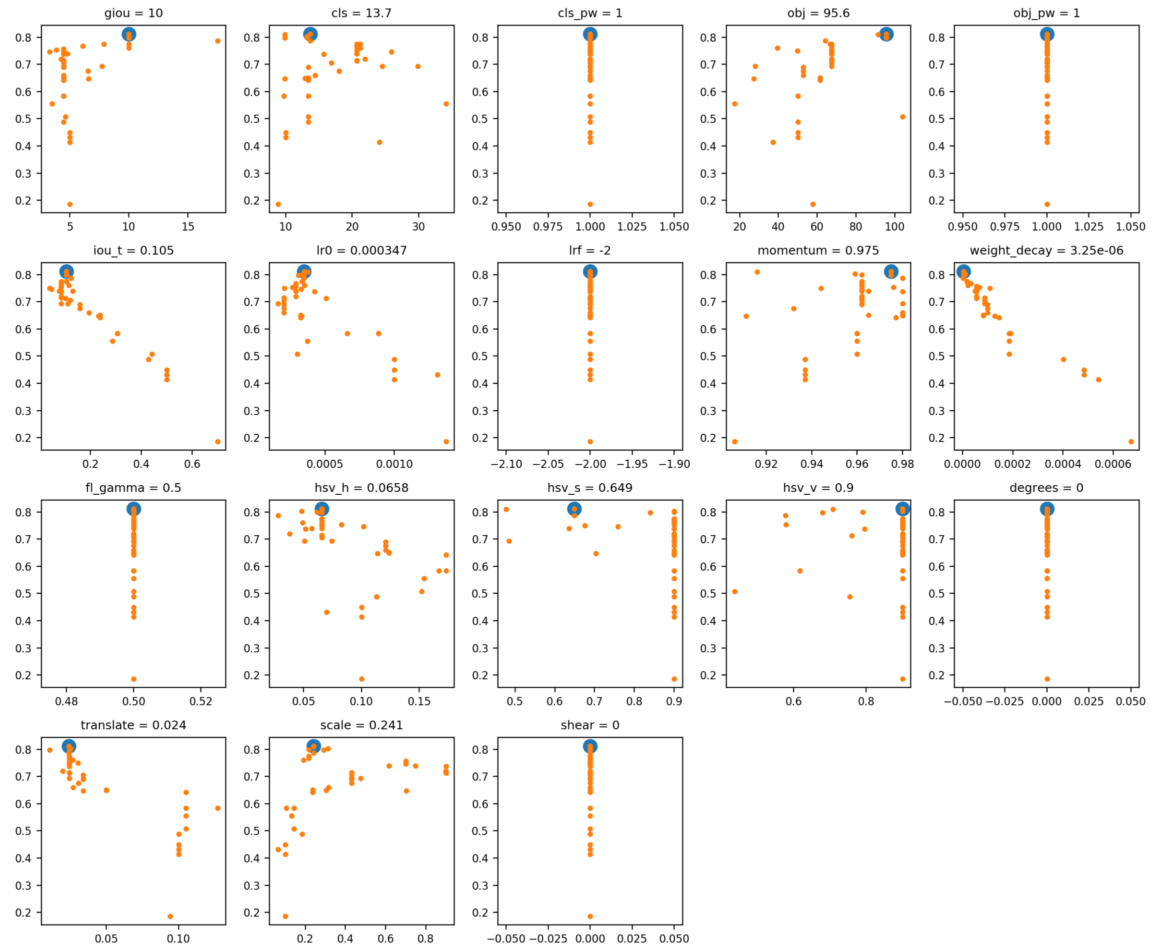
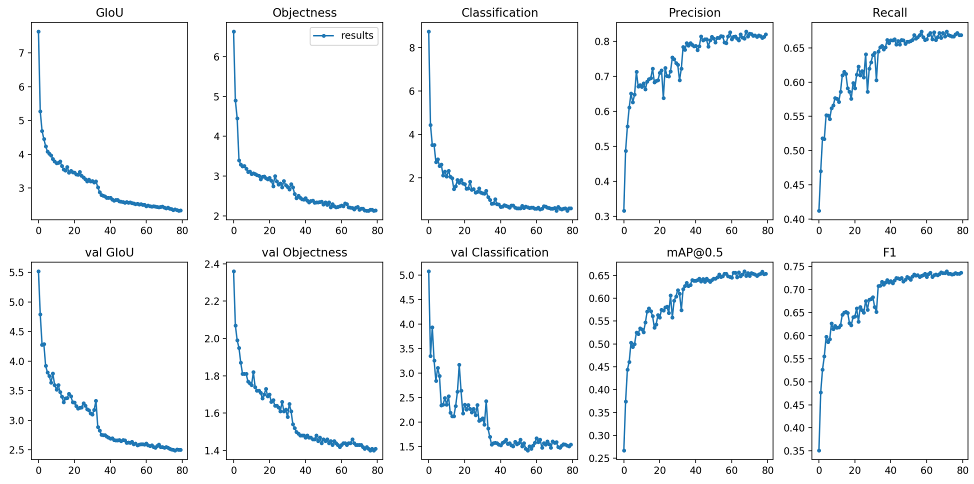


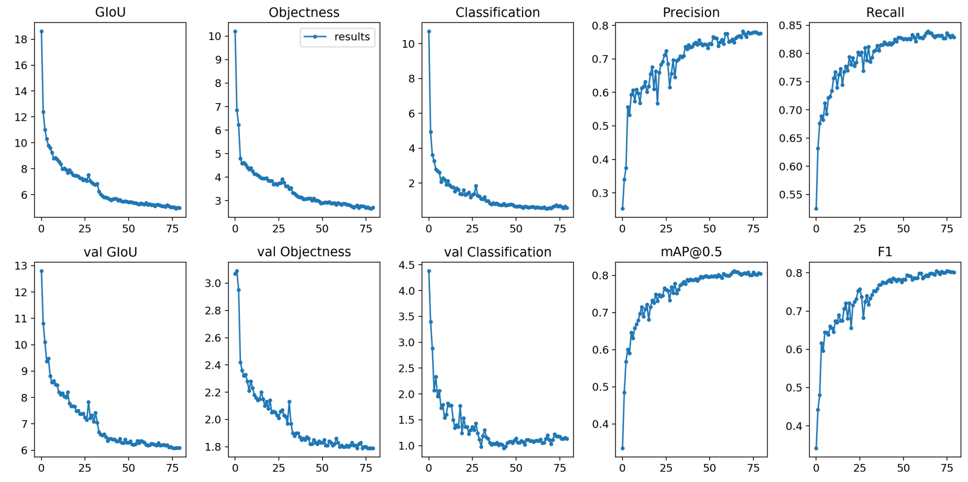
Figure A.1.2: Hyperparameters optimization in experiment 2

A.2 PyTorch YOLOv3 Training and Validation

Figure A.2.1 plots the training (top row) and validation (bottom row) results vs epochs for the two experiments. GIoU, Objectness, Classification refer to the localization loss, objectness loss, and classification loss for both training and validation; Precision, Recall, mAP, F1 are evaluation metrics on validation data.



(a) Experiment 1

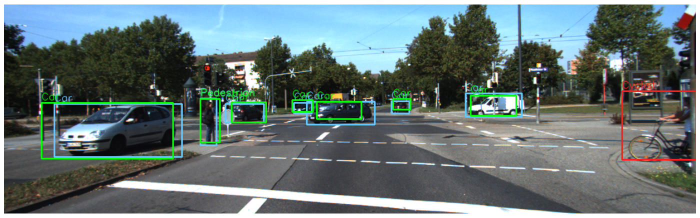


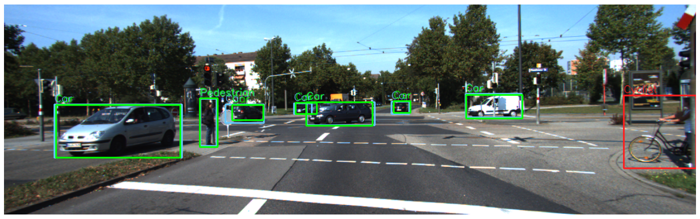
(b) Experiment 2

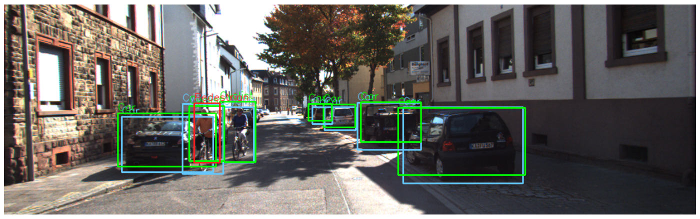
Figure A.2.1: Training and validation results

A.3 Keras YOLOv3 vs PyTorch YOLOv3

Two testing samples between Keras and PyTorch YOLOv3 are visualized in Figure A.3.1. The green, blue, red boxes refer to predicted bounding boxes, ground truth boxes, and missing/misidentified boxes.


(a) Keras YOLOv3 sample 1


(b) PyTorch YOLOv3 sample 1


(c) Keras YOLOv3 sample 2

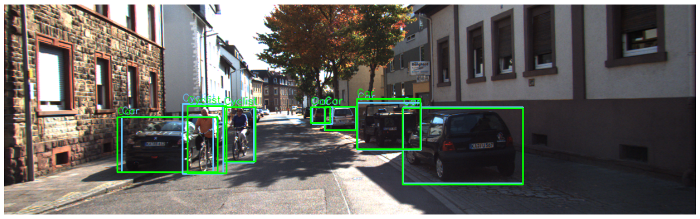

(d) PyTorch YOLOv3 sample 2

Figure A.3.1: Testing results visualization between Keras and PyTorch YOLOv3

References

[1] Ekim Yurtsever et al. “A Survey of Autonomous Driving: Common Practices and Emerging Technologies”. In: *IEEE Access* 8 (2020), pp. 58443–58469.

[2] C. Badue et al. “Self-Driving Cars: A Survey”. In: *ArXiv* abs/1901.04407 (2021).

[3] J. Janai et al. “Computer Vision for Autonomous Vehicles: Problems, Datasets and State-of-the-Art”. In: *ArXiv* abs/1704.05519 (2020).

[4] Jia Deng et al. “ImageNet: A large-scale hierarchical image database”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255.

[5] M. Everingham et al. “The Pascal Visual Object Classes (VOC) Challenge”. In: *International Journal of Computer Vision* 88.2 (June 2010), pp. 303–338.

[6] Tsung-Yi Lin et al. “Microsoft COCO: Common Objects in Context”. In: *ECCV*. 2014.

[7] Marius Cordts et al. “The Cityscapes Dataset for Semantic Urban Scene Understanding”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 3213–3223.

[8] Daniel Scharstein, Richard Szeliski, and Ramin Zabih. “A taxonomy and evaluation of dense two-frame stereo correspondence algorithms”. In: *International Journal of Computer Vision* 47 (2002), pp. 7–42.

[9] S.M. Seitz et al. “A Comparison and Evaluation of Multi-View Stereo Reconstruction Algorithms”. In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’06)*. Vol. 1. 2006, pp. 519–528.

[10] Rasmus Jensen et al. “Large scale multi-view stereopsis evaluation”. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2014, pp. 406–413.

[11] S. Baker et al. “A Database and Evaluation Methodology for Optical Flow”. In: *International Journal of Computer Vision* 92.1 (Mar. 2011), pp. 1–31.

[12] Joel Janai et al. “Slow Flow: Exploiting High-Speed Cameras for Accurate and Diverse Optical Flow Reference Data”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 1406–1416.

[13] J. Ferryman and A. Shahrokni. “PETS2009: Dataset and challenge”. In: *2009 Twelfth IEEE International Workshop on Performance Evaluation of Tracking and Surveillance*. 2009, pp. 1–6.

[14] A. Milan et al. “MOT16: A Benchmark for Multi-Object Tracking”. In: *ArXiv* abs/1603.00831 (2016).

[15] Andreas Geiger, Philip Lenz, and Raquel Urtasun. "Are we ready for autonomous driving? The KITTI vision benchmark suite". In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. 2012, pp. 3354–3361.

[16] Hang Yin and Christian Berger. "When to use what data set for your self-driving car algorithm: An overview of publicly available driving datasets". In: *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*. 2017, pp. 1–8.

[17] F. Rosique et al. "A Systematic Review of Perception System and Simulators for Autonomous Vehicles Research". In: *Sensors (Basel, Switzerland)* 19 (2019).

[18] Timothy B. Lee. *How the lidar-on-a-chip technology GM just bought probably works*. 2017. URL: `https://arstechnica.com/cars/2017/10/a-deep-dive-into-the-tech-behind-gms-new-lidar-on-a-chip-company/`.

[19] Michael Garland et al. "Parallel Computing Experiences with CUDA". In: *IEEE Micro* 28.4 (2008), pp. 13–27.

[20] John E. Stone, David Gohara, and Guochun Shi. "OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems". In: *Computing in Science Engineering* 12.3 (2010), pp. 66–73.

[21] Karl Pauwels et al. "A Comparison of FPGA and GPU for Real-Time Phase-Based Optical Flow, Stereo, and Local Image Features". In: *IEEE Transactions on Computers* 61.7 (2012), pp. 999–1012.

[22] Norman P. Jouppi et al. “In-datacenter performance analysis of a tensor processing unit”. In: *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. 2017, pp. 1–12.

[23] Teledyne DALSA. *CCD vs CMOS*. URL: `https://www.teledynedalsa.com/en/learn/knowledge-center/ccd-vs-cmos/`.

[24] Mahesh Ramachandran, Ashok Veeraraghavan, and Rama Chellappa. “CHAPTER 5 - Video Stabilization and Mosaicing”. In: *The Essential Guide to Video Processing*. Ed. by Al Bovik. Boston: Academic Press, 2009, pp. 109–140. ISBN: 978-0-12-374456-2.

[25] José Marcato Junior, M. Moraes, and Antonio Maria Garcia Tommaselli. “EXPERIMENTAL ASSESSMENT OF TECHNIQUES FOR FISHEYE CAMERA CALIBRATION”. In: 2015.

[26] Davide Scaramuzza. “Omnidirectional Camera”. In: *Computer Vision: A Reference Guide*. Ed. by Katsushi Ikeuchi. Boston, MA: Springer US, 2014, pp. 552–560. ISBN: 978-0-387-31439-6.

[27] Aish Dubey. “Stereo vision — Facing the challenges and seeing the opportunities for ADAS applications”. In: 2016.

[28] R. Horaud et al. “An overview of depth cameras and range scanners based on time-of-flight technologies”. In: *Machine Vision and Applications* 27 (2016), pp. 1005–1020.

[29] Miles Hansard et al. “Characterization of Time-of-Flight Data”. In: *Time-of-Flight Cameras: Principles, Methods and Applications*. London: Springer London, 2013, pp. 1–28. ISBN: 978-1-4471-4658-2.

[30] Open Source Imaging. *Structured-light 3D scanner*. URL: https://www.opensourceimaging.org/project/structured-light-3d-scanner/.

[31] Guillermo Gallego et al. "Event-based Vision: A Survey". In: *IEEE transactions on pattern analysis and machine intelligence* PP (2020).

[32] UZH Robotics and Perception Group. *Event-based, 6-DOF Pose Tracking for High-Speed Maneuvers using a Dynamic Vision Sensor [Video]*. URL: https://www.youtube.com/watch?v=LauQ6LWTkxM&t=30s.

[33] Jay Alammar. *The Illustrated Word2vec*. 2019. URL: https://jalammar.github.io/illustrated-word2vec/.

[34] Wikipedia contributors. *Sigmoid function*. [Online; accessed 01-July-2021]. URL: https://en.wikipedia.org/wiki/Sigmoid_function.

[35] Wikipedia contributors. *Softmax function*. [Online; accessed 24-June-2021]. URL: https://en.wikipedia.org/wiki/Softmax_function.

[36] Wikipedia contributors. *Activation function*. [Online; accessed 04-June-2021]. URL: https://en.wikipedia.org/wiki/Activation_function.

[37] Wikipedia contributors. *Rectifier (neural networks)*. [Online; accessed 22-July-2021]. URL: https://en.wikipedia.org/wiki/Rectifier_(neural_networks).

[38] N. Dalal and B. Triggs. "Histograms of oriented gradients for human detection". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 1. 2005, 886–893 vol. 1.

[39] Corinna Cortes and Vladimir Vapnik. “Support-Vector Networks”. In: *Machine Learning*. 1995, pp. 273–297.

[40] Pedro F. Felzenszwalb et al. “Object Detection with Discriminatively Trained Part-Based Models”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32.9 (2010), pp. 1627–1645.

[41] Ross B. Girshick et al. “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation”. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition* (2014), pp. 580–587.

[42] J. R. R. Uijlings et al. “Selective Search for Object Recognition”. In: *International Journal of Computer Vision* 104.2 (2013), pp. 154–171.

[43] Ross B. Girshick. “Fast R-CNN”. In: *2015 IEEE International Conference on Computer Vision (ICCV)* (2015), pp. 1440–1448.

[44] Shaoqing Ren et al. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39 (2015), pp. 1137–1149.

[45] Jifeng Dai et al. “R-FCN: Object Detection via Region-based Fully Convolutional Networks”. In: *ArXiv* abs/1605.06409 (2016).

[46] Joseph Redmon et al. “You Only Look Once: Unified, Real-Time Object Detection”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 779–788.

[47] Christian Szegedy et al. “Going deeper with convolutions”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 1–9.

[48] W. Liu et al. “SSD: Single Shot MultiBox Detector”. In: *ECCV*. 2016.

[49] K. Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *CoRR* abs/1409.1556 (2015).

[50] Tsung-Yi Lin et al. “Feature Pyramid Networks for Object Detection”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), pp. 936–944.

[51] Joseph Redmon and Ali Farhadi. “YOLO9000: Better, Faster, Stronger”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), pp. 6517–6525.

[52] Tsung-Yi Lin et al. “Focal Loss for Dense Object Detection”. In: *2017 IEEE International Conference on Computer Vision (ICCV)* (2017), pp. 2999–3007.

[53] Joseph Redmon and Ali Farhadi. “YOLOv3: An Incremental Improvement”. In: *ArXiv* abs/1804.02767 (2018).

[54] Hei Law and Jia Deng. “CornerNet: Detecting Objects as Paired Keypoints”. In: *ArXiv* abs/1808.01244 (2018).

[55] Kaiwen Duan et al. “CenterNet: Keypoint Triplets for Object Detection”. In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (2019), pp. 6568–6577.

[56] Zhi Tian et al. "FCOS: Fully Convolutional One-Stage Object Detection". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (2019), pp. 9626–9635.

[57] A. Kuznetsova et al. "The Open Images Dataset V4". In: *International Journal of Computer Vision* 128 (2020), pp. 1956–1981.

[58] Li Liu et al. "Deep Learning for Generic Object Detection: A Survey". In: *International Journal of Computer Vision* 128 (2019), pp. 261–318.

[59] Baoguang Shi, X. Bai, and C. Yao. "An End-to-End Trainable Neural Network for Image-Based Sequence Recognition and Its Application to Scene Text Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39 (2017), pp. 2298–2304.

[60] Y. Deng et al. "Image-to-Markup Generation with Coarse-to-Fine Attention". In: *ICML*. 2017.

[61] Gee-Sern Hsu, Jiun-Chang Chen, and Yu-Zu Chung. "Application-Oriented License Plate Recognition". In: *IEEE Transactions on Vehicular Technology* 62.2 (2013), pp. 552–561.

[62] OpenALPR Inc. *OpenALPR datasets*. 2016. URL: `https://github.com/openalpr/benchmarks/tree/master/endtoend`.

[63] Rayson Laroca et al. "A Robust Real-Time Automatic License Plate Recognition Based on the YOLO Detector". In: *2018 International Joint Conference on Neural Networks (IJCNN)* (2018), pp. 1–10.

[64] Xingang Pan et al. “Spatial As Deep: Spatial CNN for Traffic Scene Understanding”. In: *AAAI*. 2018.

[65] Seokju Lee et al. “VPGNet: Vanishing Point Guided Network for Lane and Road Marking Detection and Recognition”. In: *2017 IEEE International Conference on Computer Vision (ICCV)* (2017), pp. 1965–1973.

[66] D. Neven et al. “Towards End-to-End Lane Detection: an Instance Segmentation Approach”. In: *2018 IEEE Intelligent Vehicles Symposium (IV)* (2018), pp. 286–291.

[67] Yuenan Hou et al. “Learning Lightweight Lane Detection CNNs by Self Attention Distillation”. In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (2019), pp. 1013–1021.

[68] I. Goodfellow et al. “Generative Adversarial Networks”. In: *ArXiv* abs/1406.2661 (2014).

[69] M. Ghafoorian et al. “EL-GAN: Embedding Loss Driven Generative Adversarial Networks for Lane Detection”. In: *ArXiv* abs/1806.05525 (2018).

[70] Jun Li et al. “Deep Neural Network for Structural Prediction and Lane Detection in Traffic Scene”. In: *IEEE Transactions on Neural Networks and Learning Systems* 28.3 (2017), pp. 690–703.

[71] Qin Zou et al. “Robust Lane Detection From Continuous Driving Scenes Using Deep Neural Networks”. In: *IEEE Transactions on Vehicular Technology* 69 (2020), pp. 41–54.

[72] S. Chougule et al. “Reliable Multilane Detection and Classification by Utilizing CNN as a Regression Network”. In: *ECCV Workshops*. 2018.

[73] Bert De Brabandere et al. “End-to-end Lane Detection through Differentiable Least-Squares Fitting”. In: *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)* (2019), pp. 905–913.

[74] Eduardo Romera et al. “ERFNet: Efficient Residual Factorized ConvNet for Real-Time Semantic Segmentation”. In: *IEEE Transactions on Intelligent Transportation Systems* 19.1 (2018), pp. 263–272.

[75] G. Brostow, J. Fauqueur, and R. Cipolla. “Semantic object classes in video: A high-definition ground truth database”. In: *Pattern Recognit. Lett.* 30 (2009), pp. 88–97.

[76] Mohamed Aly. “Real time detection of lane markers in urban streets”. In: *2008 IEEE Intelligent Vehicles Symposium*. 2008, pp. 7–12.

[77] Jannik Fritsch, Tobias Kühnl, and Andreas Geiger. “A new performance measure and evaluation benchmark for road detection algorithms”. In: *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*. 2013, pp. 1693–1700.

[78] TuSimple. *TuSimple Lane Detection Challenge*. 2017. URL: `https://github.com/TuSimple/tusimple-benchmark/tree/master/doc/lane_detection`.

[79] F. Yu et al. “BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning”. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020), pp. 2633–2642.

[80] DMP. *A Closer Look at YOLOv3*. 2018. URL: https://blog.dmprof.com/post/a-closer-look-at-yolov3/.

[81] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 770–778.

[82] qqwweee. *keras-yolo3*. 2018. URL: https://github.com/qqwweee/keras-yolo3.

[83] Ultralytics. *YOLOv3 in PyTorch*. 2018. URL: https://github.com/ultralytics/yolov3.

[84] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980 (2015).

[85] Hamid Rezatofighi et al. “Generalized Intersection over Union”. In: (June 2019).

[86] Jonathan Krause et al. “3D Object Representations for Fine-Grained Categorization”. In: *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*. Sydney, Australia, 2013.

[87] R. Smith. “An Overview of the Tesseract OCR Engine”. In: *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*. Vol. 2. 2007, pp. 629–633.

[88] Wikipedia contributors. *Otsu’s method*. [Online; last edited 30-April-2021]. URL: https://en.wikipedia.org/wiki/Otsu%27s_method.

[89] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

[90] Christopher Olah. *Understanding LSTM Networks*. 2015. URL: `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

[91] A. Graves et al. “Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks”. In: *Proceedings of the 23rd international conference on Machine learning* (2006).

[92] Jieru Mei. *Convolutional Recurrent Neural Network in Pytorch*. 2017. URL: `https://github.com/meijieru/crnn.pytorch`.

[93] Max Jaderberg et al. “Synthetic Data and Artificial Neural Networks for Natural Scene Text Recognition”. In: *ArXiv* abs/1406.2227 (2014).

[94] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. “Neural Machine Translation by Jointly Learning to Align and Translate”. In: *CoRR* abs/1409.0473 (2015).

[95] D. Britz et al. “Massive Exploration of Neural Machine Translation Architectures”. In: *ArXiv* abs/1703.03906 (2017).

[96] Marcus Bloice, C. Stocker, and Andreas Holzinger. “Augmentor: An Image Augmentation Library for Machine Learning”. In: *ArXiv* abs/1708.04680 (2017).

[97] Wikipedia contributors. *Nearest-neighbor interpolation*. [Online; last edited 03-February-2017]. URL: `https://en.wikipedia.org/wiki/Nearest-neighbor_interpolation`.

[98] Wikipedia contributors. *Sinc function*. [Online; last edited 01-July-2021]. URL: `https://en.wikipedia.org/wiki/Sinc_function`.

[99] Logitech. *BRIO ULTRA HD PRO BUSINESS WEBCAM*. URL: `https://www.logitech.com/en-ca/products/webcams/brio-4k-hdr-webcam.960-001105.html`.

[100] Yuenan Hou. *Codes-for-Lane-Detection/ERFNet-CULane-PyTorch*. 2019. URL: `https://github.com/cardwing/Codes-for-Lane-Detection/tree/master/ERFNet-CULane-PyTorch`.

[101] Tong Liu et al. "Lane Detection in Low-light Conditions Using an Efficient Data Enhancement: Light Conditions Style Transfer". In: *2020 IEEE Intelligent Vehicles Symposium (IV)* (2020), pp. 1394–1399.

[102] Zequn Qin, Huanyu Wang, and Xi Li. "Ultra Fast Structure-aware Deep Lane Detection". In: *ECCV*. 2020.

[103] Lucas Tabelini et al. "PolyLaneNet: Lane Estimation via Deep Polynomial Regression". In: *2020 25th International Conference on Pattern Recognition (ICPR)* (2021), pp. 6150–6156.

[104] Lucas Tabelini et al. "Keep your Eyes on the Lane: Real-time Attention-guided Lane Detection". In: *arXiv: Computer Vision and Pattern Recognition* (2020).

[105] Andrew L. Maas et al. "First-Pass Large Vocabulary Continuous Speech Recognition using Bi-Directional Recurrent DNNs". In: *ArXiv* abs/1408.2873 (2014).

[106] Ashish Vaswani et al. “Attention is All you Need”. In: *ArXiv* abs/1706.03762 (2017).

[107] Nicolas Carion et al. “End-to-End Object Detection with Transformers”. In: *ArXiv* abs/2005.12872 (2020).