

INNOVATIVE CONVOLUTIONAL NEURAL NETWORKS FOR OBJECT DETECTION IN AUTONOMOUS DRIVING SYSTEMS

PhD Thesis

Tolga TURAY

SCHOOL OF ENGINEERING
UNIVERSITY OF LEICESTER



Supervisor: Prof Tanya VLADIMIROVA

March 2023

Abstract

In this thesis, first, the results of the following novel studies are presented that reveal the limitations of using Convolutional Neural Networks (CNN) in object detection modules of Autonomous Driving Systems (ADSs): (i) a comprehensive investigation of the state-of-the-art of CNNs for image classification, object detection, and ADSs to determine the real-time constraints in a synergetic way; (ii) a trade-off analysis from a human vs machine and accuracy vs complexity perspective for novel CNN designs; and (iii) a novel analysis of commonly used convolution and operation types in CNNs as well as including a timing analysis aimed at assessing their impact on CNN performance. Based on the outcome of these studies, then a novel design approach is proposed, which fuses three strategies to build a new Sparse-Split-Parallelism (SSP) design framework. The SSP framework can be applied to existing block-based CNN models to lighten their computing budget, while maintaining the same performance. As the first strategy, a new type of sparse skip connections is introduced as a block-level strategy. At the module-level, a proportional channel split operation is employed, which characterizes the accuracy and model size trade-off. As the third layer-level strategy, an equal number of input and output channels in layers is selected so that the degree of parallelization is increased for faster inference.

To verify the efficacy of the proposed SSP framework, a comprehensive evaluation is conducted using three highly competitive visual recognition tasks employing ImageNet, MS COCO, and KITTI test datasets, including a real-time experiment as well. It is shown that as a result of the application of the SSP framework, the SSP-enabled CNNs for state-of-the-art CNNs, SSP-DenseNet, SSP-ResNet, and SSP-ShiftNet, outperform their ancestors featuring $\sim 3x$ lower number of parameters, $\sim 2x$ lower floating-point operations (FLOPs), and $\sim 2x$ faster inference performance, while attaining the same level of accuracies.

Acknowledgments

Sponsorship from the Ministry of National Education, Republic of Turkey, is gratefully acknowledged.

My deepest thanks and gratitude go to my supervisor, Prof Tanya VLADIMIROVA, who was an illuminating and supportive guide in this work.

I would also like to thank my dear wife, Asli TURAY, who was very patient during this work.

Table of Contents

Abstract.....	i
Acknowledgments	ii
Table of Contents.....	iii
List of Figures.....	vi
List of Tables	viii
List of Abbreviations	ix
Nomenclature.....	x
Publications.....	xi
CHAPTER 1 INTRODUCTION	
1.1 Overview of Convolutional Neural Networks	1
1.2 CNN Deployment in Automotive	2
1.3 Aims and Objectives	3
1.4 Research Contributions	4
1.5 Thesis Organization	5
CHAPTER 2 LITERATURE REVIEW	
2.1 Convolutional Neural Networks	7
2.1.1 CNN Architecture.....	7
2.1.2 Fully Convolutional Networks	11
2.1.3 Calculation of CNN Parameters	11
2.1.4 CNNs for Image Classification and Object Detection	13
2.1.5 Datasets.....	13
2.1.6 Libraries and Frameworks.....	17
2.2 Autonomous Driving Systems	20
2.2.1 Current Industry Activities	20
a) Taxonomy of Vehicle Automation	20
b) Current Status Quo in Industry.....	21
2.2.2 System Architectures and Components.....	23
a) Connectivity in ADS Architectures	23
b) Algorithmic Designs for ADS Architectures	23
c) Modules	25
2.2.3 ADSs Design Constraints	26
a) Performance and Predictability.....	28
b) Power.....	29

c) Storage	29
d) Trading-off Accuracy and Size	29
2.3 Summary	30
CHAPTER 3 CNN MODELS FOR IMAGE CLASSIFICATION	
3.1 Introduction	32
3.2 Heavyweight CNN Designs	34
3.3 Lightweight CNN Designs	40
3.4 Performance Analysis of Image Classification CNN Models.....	50
3.5 Analysis of Convolution and Operation Types	52
3.5.1 Outline of Experiment	52
3.5.2 Evaluation Results	53
3.6 Summary	56
CHAPTER 4 CNN MODELS FOR OBJECT DETECTION	
4.1 Traditional Detectors.....	57
4.2 Backbone Networks	58
4.3 Typical Baselines in CNN-based Detector Design	59
4.4 Two-stage CNN Detectors	60
4.5 One-stage CNN Detectors.....	65
4.6 Analysis of CNN-based Object Detectors.....	71
4.7 ADSs Related Works on Object Classification and Detection	72
4.7.1 Self-Driving Scenarios	73
4.7.2 Sensors.....	75
4.7.3 Detection Types.....	76
4.8 Summary	78
CHAPTER 5 DESIGN FRAMEWORK FOR LIGHTWEIGHT HIGH- PERFORMANCE CNN MODELS	
5.1 Introduction	79
5.2 SSP Framework Design Strategies.....	81
5.3 Prior Work.....	83
5.3.1 Compression Methods for Compactness	83
5.3.2 Compact Model Development.....	84
5.4 SSP Framework Design	85
5.4.1 Sparse Dense Connections	86
a) Related Work	86
b) Proposed Solution.....	88

c) Number of Modules	88
5.4.2 Proportional Channel Split	89
5.4.3 Degree of Parallelism	90
5.4.4 Framework Application.....	91
5.5 Evaluation Results.....	92
5.5.1 ImageNet Dataset Based Evaluation	92
a) SSP-Enabled Models	93
b) Training	95
c) Optimal Step Number	96
d) Experimental Results.....	97
e) Inference Time.....	99
5.5.2 CIFAR Datasets Based Evaluation.....	100
a) SSP-Enabled Models	100
b) Datasets and Training.....	101
c) Experimental Results.....	101
5.6 Discussion	102
5.6.1 SSP Framework: Advantages and Limitations.....	102
5.6.2 Sparse Skip Connections: A Powerful Novel Design Method.....	103
5.6.3 Effects of Skip Connections	104
5.6.4 Metric Analysis	104
5.7 Generalization Ability.....	106
5.7.1 MS COCO Dataset	106
5.7.2 KITTI Dataset.....	107
5.7.3 Real-Time Performance.....	108
5.8 Summary	110
CHAPTER 6 CONCLUSIONS AND FUTURE WORK	
6.1 Research Summary.....	113
6.2 Novelty Claims.....	115
6.3 Future Work	116
Appendix: CNN Design and Training Exploration	
References	

List of Figures

Figure 2.1: A different demonstration of AlexNet architecture.	8
Figure 2.2: The difference between image classification (a) and object detection (b) tasks.....	13
Figure 2.3: Trends in the selection of ML frameworks [71].	19
Figure 2.4: SAE recent updates on automation levels [74].	21
Figure 2.5: Algorithmic design types: (a) modular system; (b) end-to-end system.	24
Figure 2.6: Memory hierarchy in processors [149].	28
Figure 3.1: Frequently used different types of convolutions and operations.	33
Figure 3.2: Implementation of AlexNet with grouped convolutions on two GPUs [21].	35
Figure 3.3: Visualization of the layer-based features by deconvolution and unpooling [175]. ..	36
Figure 3.4: ResNet-50 architecture with bottleneck modules [22].	43
Figure 3.5: An example of dense block with five layer [24].	46
Figure 3.6: Graphical representation of the runtime timing analysis of the ResNet-50 model for the convolutions and operations in Figure 3.1. (a) ResNet-50 model with spatial filters; (b) and (c) - ResNet-50 model with grouped filters and shuffle operation, respectively; (d) ResNet-50 model with depth-wise and point-wise convolutions; (w) ResNet-50 model with shift operations.	55
Figure 4.1: Data-flow diagram of traditional detectors.	58
Figure 4.2: Functional diagrams of typical detector pipelines: (a) two-stage detector; (b) one-stage detector.	59
Figure 4.3: Timeline of traditional and CNN-based object detectors.	60
Figure 4.4: Architectural demonstrations of the three best-known R-CNN models R-CNN [220], Fast R-CNN [221], and Faster R-CNN [146].	62
Figure 4.5: Illustrations of the single-stage pipelines of the YOLO [103] and SSD [222] architectures.	66
Figure 4.6: CenterNet network [240].	69
Figure 5.1: Illustration of the amount of feature reuse for the third block of a trained DenseNet- 121.	82
Figure 5.2: An illustration of a SSP framework-applied CNN block comprising eight modules with three-step-away sparse skip connections.	86
Figure 5.3: Illustration of module design on SSP framework application for a block. (a) the proposed SSP module of SSP framework; (b) DenseNet bottleneck module [24]; (c) ResNet bottleneck module [22]; (d) ShiftNet CSC module [12]; (e) ShuffleNet Unit module [26]; (f) ShuffleNet-v2 Basic Unit module [25].	90
Figure 5.4: Results of the investigation of the most powerful step number in feature reuse. The numbers on the four lines represent the selected step number.	96

Figure 5.5: Error rates for ResNet-50, SSP-ResNet-50, ShuffleNet 1x ($g=3$), and SSP-ShuffleNet 1x ($g=3$) during training and validation, where the dotted curves correspond to the training phase.	98
Figure 5.6: Making trade-off between model size and accuracy with different ratios of channel split via ϵ hyperparameter.	99
Figure 5.7: Speedup performance comparisons on GPU (batch size=8) and CPU (batch size=1) platforms.	105
Figure 5.8: A detection instance with Faster RCNN on MS COCO.	107
Figure 5.9: A detection instance with Faster RCNN on KITTI.	108
Figure 5.10: Raspberry Pi 4 Model B (4 GB RAM) and Raspberry Pi Camera.	109
Figure 5.11: One sample each from real-time image classification experiment: (a) SSP-Resnet-50; (b) SSP-ShuffleNet 1x ($g=3$).	110

List of Tables

Table 2.1: Layer by layer parameters of Alex Net.	12
Table 2.2: Main features of training datasets for image classification and object detection CNNs.	15
Table 2.3: Commonly used libraries with python interface.	18
Table 2.4: Automation levels.	20
Table 2.5: Current situation of autonomous driving under experimentation.	22
Table 2.6: Comparison of design types.	24
Table 2.7: Advantages of accuracy and model size in CNNs.	30
Table 3.1: Performances of image classification algorithms on ImageNet.	51
Table 3.2: CIFAR-10&100 training of ResNet-50 with different convolution types.	54
Table 4.1: Performances of CNN-based object detectors.	72
Table 5.1: Implementation details of the SSP-enabled models.	92
Table 5.2: Top-1 and Top-5 error rates (single-crop/10-crop testing) on the ImageNet validation dataset with other parameters.	97
Table 5.3: Top-1 error rates (%) of SSP-enabled models on CIFAR-10&100 test sets without/with data augmentation.	102
Table 5.4: mAP (%) on MS COCO 2014 dataset.	106
Table 5.5: mAP (%) on KITTI dataset.	107
Table 5.6: Real-time inference performance (fps) comparison of the proposed models in this research.	110

List of Abbreviations

ADS	: Autonomous Driving System
ASIC	: Application-Specific Integrated Circuit
CNN	: Convolutional Neural Network
CPU	: Central Processing Unit
DBN	: Deep Belief Network
DNN	: Deep Neural Network
DL	: Deep Learning
FCL	: Fully Connected Layer
FCN	: Fully Convolutional Network
FLOP	: Floating-Point Operation
FPGA	: Field Programmable Gate Array
GPU	: Graphical Processing Unit
ML	: Machine Learning
RBM	: Restricted Boltzmann Machines
RNN	: Recurrent Neural Network

Nomenclature

B_C	: bias in convolutional layers
B_{CF}	: bias from convolutional to fully connected layer
B_{FF}	: bias from fully connected to fully connected layer
g	: growth rate
I or i	: input of a layer
O or o	: output of a layer
K	: kernel
I_D, O_D, K_D	: spatial height and width of input, output, and kernel
I_C, O_C, K_C	: channel number of input, output, and kernel
F	: filter number
K_P	: spatial height and width of pooling filter
P	: padding
S	: stride number
N	: neuron number
n	: number of examples from a dataset
m	: number of modules
s	: step number
$\mathcal{E}$	: channel division rate
μ	: learning rate

Publications

- 1-) T. Turay and T. Vladimirova, "Toward Performing Image Classification and Object Detection With Convolutional Neural Networks in Autonomous Driving Systems: A Survey," in *IEEE Access*, vol. 10, pp. 14076-14119, 2022, doi: 10.1109/ACCESS.2022.3147495.
- 2-) T. Turay and T. Vladimirova, "SSP Framework: A New Approach to Designing Lightweight Convolutional Neural Networks," in *IEEE Access*, vol. 10, pp. 102292-102307, 2022, doi: 10.1109/ACCESS.2022.3208924.

CHAPTER 1 INTRODUCTION

1.1 Overview of Convolutional Neural Networks

In recent years, Deep Learning (DL) techniques have been exhaustively utilised in a large variety of fields, and Convolutional Neural Networks (CNNs) are one of the most frequently used of them in solving computer vision tasks, as it enables the most accurate acquisitions. The concept behind CNNs drew inspiration from the structure of biotic visual systems. In the 1960s, it was theorized that the cats' visual cortex is based on sensitively to smaller sub-regions in the brain, called receptive fields [1]. Two decades later, Fukushima [2] introduced the first CNN model, based upon [1] and referred to as Neocognitron, which was simulated and implemented on a computer. This network, designed by using multi-layer artificial neuron connections for the transformation of images, is still considered as the source of inspiration for CNNs. Then, LeCun et al. [3], [4] created a CNN model, referred to as LeNet-5, which specifically classifies handwritten digits and can be trained to recognize patterns from raw pixels through using the backpropagation algorithm [5].

Despite all the innovative approaches, LeNet-5 [4] was ineffective in the implementation of complex problems, such as image classification, requiring large training data and powerful processors for computation. Advancements in hardware accelerators, such as Graphical Processing Units (GPUs), Field Programmable Gate Arrays (FPGAs), Application-Specific Integrated Circuits (ASICs), etc., made these devices a suitable implementation choice for Machine Learning (ML) techniques [6]. Becoming widespread in the early 2010s, CNN models along with efficient training methods were released and implemented on GPUs [7]–[9]. Following that revolution, many CNN algorithms have been proposed addressing the needs of different areas, with computer vision and natural language processing seen as major application fields [10]. Computer vision tasks to which CNNs have mostly been applied are image classification, object detection, face recognition, scene labelling, action recognition, human pose estimation, and document analysis [11]. Regarding CNN usage in industrial technologies, they are being employed in a variety of sectors, such as medical, automotive, aviation, agriculture, space, etc. [12].

1.2 CNN Deployment in Automotive

Along with recent technological advancements, consumer expectations from vehicles have evolved to extending their digital lifestyles into vehicle cabins and connecting with outside worlds. Concurrently, growing traffic congestion, rise in numbers of new drivers, and changing priorities have directed the focus on safety-based systems as well [13]. As a result, automakers have started to transition from horsepower to digital technology as a standard feature, enabling self-driving capabilities in the production of automobiles, which will be referred to as Autonomous Driving Systems (ADSs) [14] in the rest of this thesis.

In the near future, the next generation of ADSs promises an advanced level of self-driving experience. To enable this advance, the industry needs to combine multiple technologies into a single system, which has the capability of communicating, collaborating, and eventually performing human functions for almost all driving scenarios [14]. According to Intel [15], all these capabilities need numerous sensors, such as radar, lidar, GPS, and advanced camera technologies, capable of collecting the necessary information of ~ 1 GB/second. This data should be processed in real-time to make safe decisions for the vehicle on the road. Despite the advantages of lidars, such as high precision and 3D mapping, their extreme cost prevents their commercial deployments in the industry. For example, existing lidars in the market are as expensive as \$75k [16]. Thereby, some companies like Tesla [17] and Mobileye [18], [19] have announced that optical cameras and radars are their preferred choice of sensing devices in vision-based parts of ADSs. From this point of view, CNNs as an image-based technique have been effectively and widely utilised by automakers in object detector modules of automobiles.

Although there have been lots of advancements in object detection with CNNs for various applications, because of the stringent real-time limitations on both safe operational decision-making and latency of data processing, ADSs are notably challenging and still under experimentation. Therefore, each of the modules, dedicated to the main tasks, in the computational pipeline system of *modular* ADSs, needs improvements of their performances [14]. In addition, designing an *end-to-end* ADS, another algorithmic design type [14] specifying the system components and architecture like *modular* ADSs, remains an unsolved research topic [20]. As a result, efficient approaches to the trade-off between computational load and accuracy for ADS pipelines promise a bright future.

1.3 Aims and Objectives

At the initial glance of the areas outlined above, it quickly became clear that this rich research field deserves further investigations, analyses, and novel methodologies. To conduct that, before proposing efficient CNN design approaches, it is important to first establish what the object detection modules used in ADSs suffer from and how to tackle better limitations of ADSs, which is the strategy followed in this research. Therefore, the first aim of the research is to determine the current limitations in object detection modules of ADSs as well as explore the relative and comparative performances of frequently used convolution and operation types in novel CNNs designs. The expected outcome is revealing the research gap that CNN models with lower computational load have a strong potential to alleviate the current limitations of ADSs. Based on that this research aims to come up with an innovative design methodology to lighten the computing budget of state-of-the-art popular CNN models, while maintaining comparable accuracies so that they are applicable to object detection modules of ADSs.

To achieve the introduced aims above, our objectives are:

- The state-of-the-art of CNNs for image classification, object detection, and ADSs will be comprehensively investigated in the order of their historical evolution to determine the real-time constraints in a synergetic way.
- Using a novel way of parameter number calculation of a generic CNN, AlexNet [21], it will be revealed how convolution and operation types in CNNs impact the increase in the computational load in its entirety.
- A novel analysis of the convolution types and specially developed operations that are commonly used in CNNs for image classification will be carried out to establish the effect of these on the CNN performance. The evaluation will be based on a reference CNN model, ResNet [22], for a fair assessment, testing different types of filters on the ResNet's modules. A runtime timing analysis will also be carried out by decomposing the architectures into basic components. Experiments will be conducted using the CIFAR-10&100 [23] dataset on a Central Processing Unit (CPU) as well as two identical hardware accelerators (2xNVIDIA Pascal P100s GPUs).
- The dense skip connections in the DenseNet model [24] will be investigated with the aim of utilizing them on other state-of-the-art popular block-based CNNs as a

more efficient connectivity pattern. Alleviating the degradation problem and maximizing the information flow capability will also be examined, and how they affect accuracy increase and complexity decrease of CNN models.

- Several other promising CNN design strategies, such as, channel split [25], degree of parallelization [25], etc., will be studied. This will be followed by constructing an efficient design framework, which can be applied to every single block of existing block-based CNNs to lighten their computing budget while maintaining comparable accuracies.
- To fairly reveal its performance, the proposed design framework will be applied to the five most used CNN algorithms ShiftNet [12], ResNet [22], DenseNet [24], ShuffleNet-v2 [25], ShuffleNet [26], making use of almost all types of convolutions and operations, including the *shift* operation [12], as a more sophisticated operation instead of the spatial 2D convolution.

1.4 Research Contributions

The outcome of the conducted aims and objectives has enabled significant contributions to the research field, which are:

- Novel and clear guidelines for stakeholders to calculate parameter number of different CNN models.
- A practical overview and presentation of the CNN and ADS related datasets, classifying them from a novel perspective as domain-general and domain-specific, respectively.
- A most comprehensive and up-to-date investigation and data analysis of CNNs for image classification, object detection, and ADSs in a synergetic way from different aspects of the research field.
- The first study proving the relative and comparative performance of convolution and operation types on broad-ranging experiments, bringing deep insights and path-finding into design processes to creative CNN designers in a timesaving manner.
- The first attempt to bridge the academic research and industry practice on CNNs and ADSs by means of trade-off analyses conducted from a human vs machine and accuracy vs complexity perspective, targeting novel CNN designs dedicated to the ADSs pipeline.

- Proposal of a novel SSP (*Sparse-Split-Parallelism*) framework with wisely created three design principles to remedy the limitations of CNNs for use in ADSs. It can be holistically applied to the networks whose designs are formed with blocks and modules as in the ShiftNet [12], ResNet [22], DenseNet [24], ShuffleNet-v2 [25], ShuffleNet2 [26] models.
- It is proven that the proposed novel connectivity pattern, *sparse skip connections*, enables a better trade-off on information flow through the network compared to dense skip connections. Furthermore, the sparse skip connections pattern can be deployed in other existing or forthcoming CNN models subject to required adjustments.
- The SSP framework is comprehensively evaluated on the ImageNet [21] image classification task and the MS COCO 2014 [27] / KITTI [28] object detection tasks as well its real-time performance is examined on the Raspberry Pi 4 Model B 4GB. The SSP-enabled models (SSP-ShiftNet, SSP-ResNet, and SSP-DenseNet) explicitly surpass the original ShiftNet [12], ResNet [22], and DenseNet [24] with $\sim 3\times$ lower parameter numbers and $\sim 2\times$ lower FLOPs while maintaining equivalent accuracies, which distinctively shows the promising enhancements and the capability of SSP framework on model complexity decrease via the proposed design principles.

As a result, it is expected that these contributions would stimulate future work on CNN custom design and employment for object detection modules of ADSs.

1.5 Thesis Organization

This thesis is organised into 6 chapters. The following five chapters are outlined below:

Chapter 2 comprehensively overviews CNNs and ADSs. At first, the main components of CNNs in terms of their mathematical and architectural properties are introduced. While doing this, the chapter performs how to calculate the parameter number of a generic CNN, AlexNet [21], in an innovative concept. This provides a deeper understanding of which type of layers increase the model complexity more. As a result, it inspires new CNN designs to avoid inefficient design strategies. Then in the second, system architectures and industry status of ADSs are investigated. In the light of the investigations, a novel trade-off analysis from a human vs machine and accuracy vs complexity perspective is held. By means of these analyses, promising directions aimed at solving

problems of automakers in achieving real-time implementations are determined for novel CNN design approaches;

Chapter 3 first investigates the recent CNN models for image classification with review materials to identify the current limitations. In the meanwhile, the pros and cons of commonly used convolution and operation types are theoretically revealed. Following that, a novel empirical analysis of the introduced convolution and operation types to reveal their relative and comparative performances is implemented in an evolutionary manner by decomposing the designed architectures into basic components, which are analysed to find their individual execution times on compute-bound (CPU) and memory-bound (GPU) computation platforms during inference;

Like Chapter 3, Chapter 4 makes an investigation of the recent CNN models for object detection to identify the current limitations. The most recent ADSs related works concerned with CNN-based classification and detection is also investigated to bridge academic research and industry practice on CNNs and ADSs;

Chapter 5 remedies the determined limitations of ADSs, which are revealed in a synergetic way in Chapters 2-4. A novel approach is presented to design lightweight CNN models for object detection modules of ADSs. Three wisely produced strategies are fused to build a framework, which disburdens the computation complexity of existing popular CNN models;

Chapter 6 concludes the thesis.

The thesis makes use of material and results reported in the author's articles [29] and [30].

CHAPTER 2 LITERATURE REVIEW

The accumulation of knowledge on vehicle dynamics, the ground-breaking improvements in computer vision through the emergence of Deep Neural Networks (DNNs), and the advent of newly designed sensor modalities energized the R&D activities among researchers and developers of Autonomous Driving Systems (ADSs). From the first large-scaled automated driving competitions, DARPA Grand [31] and Urban [32] Challenges, to this day, numerous approaches have been proposed, and common system architectures have been established. In this period, substantial tasks in ADSs have been divided into subcategories, and explicit dominance of CNN models has been seen in a number of subcategories [33] aimed at reducing human interventions in driving. CNNs as a type of layered DNNs compose of artificial neurons and utilise several distinctive properties to provide high-level performance compared to other neural network types, such as local receptive fields, spatial subsampling, shared weights, etc. Even though these extraordinary advances, robust ADSs in urban environments have not been implemented yet [14], [34], [35].

This chapter provides background information for the research presented in Chapters 3-5. Section 2.1 introduces the essential components of a generic CNN algorithm, AlexNet [21], and the calculation of the parameter number of it. This calculation reveals the computationally costly layers and inspires novel CNN improvements. Section 2.1 also overviews: (i) the difference between image classification and object detection; (ii) the most frequently used datasets and libraries in the field from a different viewpoint: domain-general and domain-specific. Following that, section 2.2 presents the current industry status and system architectures of ADSs. Furthermore, in section 2.2, trade-off analyses from a human vs machine and accuracy vs complexity perspective are presented. Lastly, the chapter is concluded in section 2.3.

2.1 Convolutional Neural Networks

2.1.1 CNN Architecture

The input and output of each convolutional layer are known as feature maps. If the input is an image, each feature map is a 2D array containing colour channels and if a video, the feature map has to be a 3D array and 1D arrays for audio inputs. In every output stage along with the network, there are new features extracted from the pixels of its input,

and by the convolutional operations, more distinguishing features are detected in the later layers.

As an example of a typical CNN, the entire architecture of the popular CNN AlexNet [21], is illustrated in Figure 2.1, which includes five convolutional layers with padding operations, three max-pooling layers, and three FCLs. In Figure 2.1, I_D , O_D , K_D denote the spatial height and width dimension of the inputs, outputs, and kernels, while I_C , O_C , K_C are the channel numbers of those, respectively. In addition, F is the filter number, which is equal to the outputs O_C , and the kernel dimension in the pooling operation is symbolized by K_P . A notable term that will be used in the layer-based definitions is tensor, which in the simplest form is a generalization of matrices to n -dimensional space.

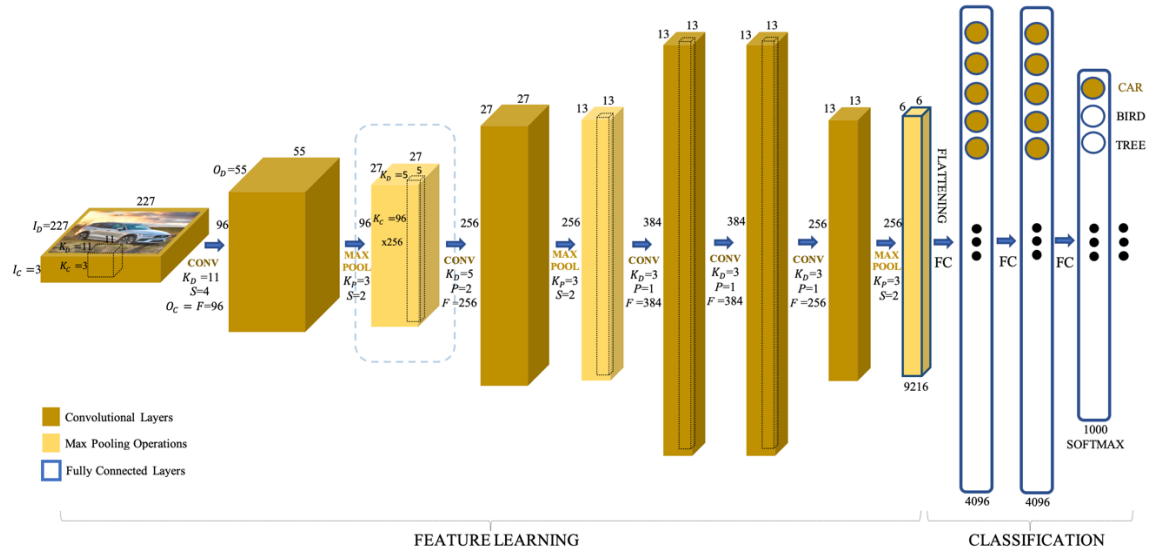


Figure 2.1: A different demonstration of AlexNet architecture.

The following subsections define all the architectural components of a CNN, based on the reference architecture AlexNet in Figure 2.1.

a) Input Layer

Input layers introduce input data in the network, which normally represents an image structured as a data array of pixel values. Before feeding the image into a designed model, the spatial and channel dimensions of the input have to be reshaped according to the model or the used deep learning library specifications.

b) Convolutional Layer

The core of a CNN is the convolutional layers. Learnable parameters in these layers form the kernels and the collection of the kernels in a layer is called a filter, which is

subjected to a convolution multiplication through the full spatial depth of the input. A kernel defines the field of view on the convolution operation, whereas, as a tensor sample, a filter is a concatenation of the total number of kernels in a layer channel-wise. For instance, a filter with a size of " $m \times n \times c$ " includes c kernels with " $m \times n$ " kernel dimension.

Via the convolution operation, a feature map is obtained with a new spatial dimension and channel number based on the input and filter dimensions. Based on the used kernel's parameter values, outputs are extracted with different features. In Figure 2.1, all the convolutional layers are composed of standard spatial 2D convolutions. Consider a spatial convolution, which takes a tensor $I \in \mathbb{R}^{I_D \times I_D \times I_C}$ as the input, and the filter of the convolution is the tensor $K \in \mathbb{R}^{K_D \times K_D \times I_C \times F}$. For simplicity, it is assumed the spatial dimensions of the input/output to be identical, which means that $O_D = I_D$, and the stride (i.e., the convolution kernel's step size) is 1. Then, the spatial 2D convolution outputs a tensor $O \in \mathbb{R}^{I_D \times I_D \times F}$, computed as

$$O_{k,l,n} = \sum_{i,j,m} K_{i,j,m,n} I_{k+\hat{i}, l+\hat{j}, m}, \quad (2-1)$$

where $\hat{i} = i - \lfloor I_D/2 \rfloor$ and $\hat{j} = j - \lfloor I_D/2 \rfloor$ denote the re-centred spatial indices; k, l and i, j are the indices over the spatial dimensions; whereas m, n provide indexation to the channels and filters.

c) Padding Operation

Padding is the placement of several extra pixel grids to the input's spatial plane to handle the output's spatial dimensions. In case of a demand for an equal dimension of input and output tensors, the padding operation could enable it. As seen in Figure 2.1, the output tensor's spatial dimensions are in the control of the padding operation. The effect of the padding operation in the output tensor's dimension ($O_D \times O_D$) is defined by

$$O_D = I_D + 2P - K_D + 1. \quad (2-2)$$

d) Stride Operation

The stride value indicates that the filter, which is the weight tensor for the convolution process, slides on the input tensor in increments of one or more-pixel steps. This is

another parameter that directly affects the output size. The dimension of the output tensor ($O_D \times O_D$) is defined by

$$O_D = \frac{I_D + 2P - K_D}{S} + 1, \quad (2-3)$$

where S is the stride number parameter.

e) Activation Functions

CNNs use various activation functions [36], such as Rectified Linear Units (ReLU), Sigmoid, TanH, etc., to build the feature map, obtained through the convolution operation. These functions enable the introduction of nonlinearities to the layers, which increases learnability. In particular, ReLU has been frequently preferred in CNNs on the score of enabling several times faster training in comparison to the other activation functions.

f) Pooling Operation

Pooling layers take small-sized rectangular blocks, defined by K_P , from the output feature map, and form subsamples from it to produce a single maximum, minimum, or average output from every block. After that, a new sampled feature map is formed, which is then used as an input for the next convolutional layer. By reducing the parameters of the feature map, the pooling layers allow to reduce the spatial size as well as the control of overfitting. As instance, in Figure 2.1, following the convolutional layers and the application of the activation functions, there are three maximum pooling layers, represented with light amber layers. Additionally, global average pooling is another pooling method, used in [37] as well. It does a pooling to the whole image at a time without any sliding.

g) Fully Connected Layer

The CNN architecture performs feature learning via the operations introduced above until the last convolutional and pooling layer output is derived. The final CNN stage consists of Fully Connected Layers (FCLs) and performs a high-level classification. It is placed after the last convolution or pooling operation has occurred. The flattening operation in Figure 2.1 provides the transition from convolutional later to FCL by converting the data to a 1-dimensional array. Essentially, in this stage of the architecture, a usual neural network process for the operation of classification takes place.

h) Output Layer

This is the final stage, and the predicted result for the input image is obtained via different cost functions. For instance, in Figure 2.1, Softmax multi-class classifier [38] is deployed. Thus, a cost is produced for a prediction by comparing the predicted result and the real data from the training set.

2.1.2 Fully Convolutional Networks

Fully Convolutional Networks (FCNs) [39] are neural networks that are composed of only convolutional layers without adding any FCLs. The difference from CNNs is that FCNs are fully convolutional networks, which do all the learning with convolution-based filters even for decision making in the last layer, whereas CNNs include FCLs at the end of its architecture like Figure 2.1. Another difference is that FCNs learn everything by using global information, whereas FCLs try to learn and make decisions only based on local spatial inputs. A few examples of this type of networks are included in Chapters 3-4.

2.1.3 Calculation of CNN Parameters

Definition of the output tensors' dimension and calculation of the number of parameters, memory accesses, and FLOPs in CNNs are crucial for understanding superior aspects of the different types of convolution filters discussed in the rest of this thesis. This also shows which type of design strategies increase these parameters more, which leads to inefficiencies in the real-time implementations. This section, therefore, introduces the widely used method to calculate the number of parameters for frequently used 2D convolutions, based on the AlexNet architecture [21] in Figure 2.1. The number of parameters, memory accesses, and FLOPs for different types of convolution kernels are also calculated in Chapter 3. Initially, output dimensions have to be examined regarding different layer types. Equation (2-3) is employable in convolutional and pooling layers with the difference that K_D needs to be replaced by the kernel size of the pooling operation, K_P . As for the FCLs, it is a length vector and thus, the output size is equal to the number of neurons in the particular layer.

To calculate the number of CNN parameters, it can be represented by the sum of the number of parameters in three different CNN operations: the convolutional layers, the transition from convolutional layer to FCL, and the transition from FCL to FCL. Pooling layers do not include any parameters, as they are aimed at the reduction of the dimension size of the outputs.

The number of parameters in the convolutional layers can be computed by

$$F \times K_C \times K_D^2 + B_C, \quad (2-4)$$

where B_C denotes the bias number in the convolutional layer.

In the transition from convolutional layer to FCL, the number of parameters is

$$N \times I_C \times I_D^2 + B_{CF}, \quad (2-5)$$

where N is the neuron number in the first FCL; I_C and I_D are the channel number and spatial dimension of the tensor previous to the first FCL; and B_{CF} represents the bias number in the layer.

In the transition from FCL to FCL, the number of parameters is

$$N_P \times N + B_{FF}, \quad (2-6)$$

where N_P and B_{FF} are the neuron number in the previous layer and biases respectively.

Table 2.1: Layer by layer parameters of Alex Net.

Layer Name	Tensor Size	Weights	Biases	Parameters
Input Image	227x227x3	0	0	0
Conv-1	55x55x96	34,848	96	34,944
MaxPool-1	27x27x96	0	0	0
Conv-2	27x27x256	614,400	256	614,656
MaxPool-2	13x13x256	0	0	0
Conv-3	13x13x384	884,736	384	885,120
Conv-4	13x13x384	1,327,104	384	1,327,488
Conv-5	13x13x256	884,736	256	884,992
MaxPool-3	6x6x256	0	0	0
FC-1	4096x1	37,748,736	4,096	37,752,832
FC-2	4096x1	16,777,216	4,096	16,781,312
FC-3	1000x1	4,096,000	1,000	4,097,000
Output	1000x1	0	0	0
Total				62,378,344

Consequently, the total number of parameters in AlexNet, calculated by means of equations (2-4)-(2-6) defined above, is 62,378,344, as detailed in Table 2.1.

2.1.4 CNNs for Image Classification and Object Detection

Before delving into the investigation of the recent studies of CNNs for classification and object detection to determine what they suffer from, it is essential to explain these two imaging tasks. In general, classifying an object in an image aims at assigning it to a certain category, while object detection involves localization and classification of all present objects. Figure 2.2 illustrates the outcome of the two operations, where as a result of the image classification task, the image in Figure 2.2 (a) is identified as a car; and as a result of object detection in Figure 2.2 (b), cars, persons, and traffic lights are detected, and their positions and sizes are established by drawing a bounding box around the objects of interest in the image.

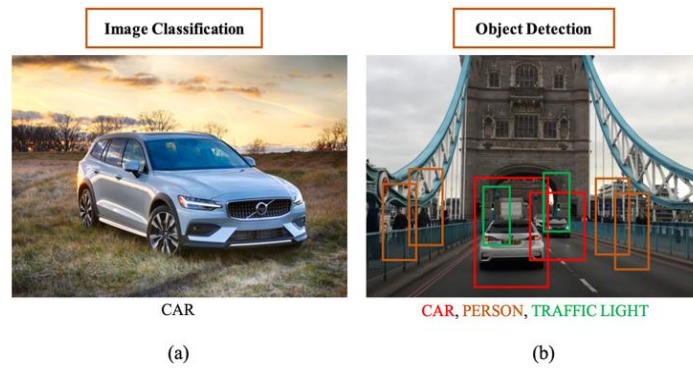


Figure 2.2: The difference between image classification (a) and object detection (b) tasks.

There is an obvious overlap between the two tasks, which requires that designs of high-performance object detection CNNs incorporate high-quality image classification properties in their architectures. In other words, an effective CNN architecture for object detection depends directly on the image classification quality of that CNN architecture.

In the sections that follows, frequently used datasets and libraries for the implementation of CNNs for image classification and object detection are introduced in a different manner, respectively.

2.1.5 Datasets

The learning capabilities of CNNs are obtained during training and heavily rely on the suitability and comprehensiveness of the available training datasets. In fact, features of datasets, such as collection scenarios, class numbers, resolution, etc., are crucial for CNN performance. In this section, the most widely cited datasets that are related to

the scope of this thesis are presented along with their distinctive characteristics. For clarity of presentation, the existing datasets are divided into two classes: domain-general and domain-specific. Domain-general datasets here represent datasets that can be used for the training of CNNs for image classification or object detection, which then could be utilised for a particular application domain. For instance, ResNet-50 [22] has been trained on a domain-general dataset, ImageNet [40], and deployed for image classification tasks in various domains, such as medical [41], agriculture [42], autonomous driving [43], and so on. Contrary to domain-general, domain-specific datasets are originally created for a particular domain with the aim to facilitate learning of specialised domain-related features. Main features of renowned examples of both classes are summarised in Table 2.2, where the names in the left-most column are shaded light grey for domain-general datasets and dark grey for domain-specific datasets.

There exist two prominent domain-general datasets for image classification, which are CIFAR-10 & 100 [23] and ImageNet [40]. As an intermediate level dataset for image classification, CIFAR-10 & 100, have been introduced by one of the creators of AlexNet [21], Alex Krizhevsky. Both consist of a 50k training set and 10k test set fixed size 32x32 colour images. In CIFAR-10, each class is represented by 6k images, whereas in CIFAR-100 there are 600 images per class. As for ImageNet [40], it is regarded as an advanced level dataset, which has been developed with around 14M labelled high-resolution images according to the WordNet hierarchy with 22k subcategories. There has been also held a CNN competition since 2010 based on this dataset, referred to as the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). Since the 2012 ILSVRC competition, the dataset has been kept unchanged and involves 1k subcategories, ~ 1.3 M training images, 50k images for validation, and 100k images for testing. In addition, the number of images for each class, termed ‘synset’ [40], ranges between 732 and 1300. The newly proposed CNNs for domain-general image classification in the academic community have been mostly trained on these two datasets. The main features of these datasets are summarised in Table 2.2.

Table 2.2: Main features of training datasets for image classification and object detection CNNs.

Domain-general & Domain-specific	~Data Size & Resolution ^a	~Number of Images & Classes ^b	Annotations ^c		Collection Sensors			Diversity on Collection Scenarios			Usage Scenarios	
			2D	3D	Camera	LIDAR	Radar	GPS (IMU/INS)	All Weather Conditions	Day / Night Time	Different Traffic Conditions	Image Classification / Object Detection ^d
CIFAR-10 & 100 [23]	162 MB 32x32 400 GB	60k 10 & 100 1.4M 1000	-	-	✓	-	-	-	-	-	✓	-
ImageNet [40]	224x224 ^e 2 GB	12k 20	-	-	✓	-	-	-	-	-	✓	-
Pascal VOC [44]	469x387 ^f	12k 20	✓	-	✓	-	-	-	-	-	✓	✓
ILSVRC 2014 [40]	482x415 37.57 GB	510k 200 328k 80	✓	-	✓	-	-	-	-	-	✓	-
MS COCO [27]	640x480	15k 7	✓	✓	✓	✓	-	✓	-	-	✓	✓
KITTI [28]	180 GB 1392x512	100k 10	✓	-	✓	-	-	✓	✓	✓	✓	-
BDD100K [46]	1280x720 63 GB	25k 30	✓	-	✓	-	-	✓	-	-	✓	✓
CityScapes [47]	1024x2048 30	12k 14	✓	✓	✓	✓	-	✓	-	-	✓	-
A2D2 [48]	2.3 TB 1920x1208	1.4M 23	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
nuScenes [49]	400 GB 1600x900	140k 28	✓	✓	✓	✓	-	✓	-	-	✓	✓
ApolloScape [50]	270 GB 3384x2710	250k	✓	-	✓	-	-	-	-	-	✓	-
Caltech [51]	11 GB 640x480	47k 30	✓	-	✓	-	-	✓	✓	✓	✓	-
EuroCity [55]	100 GB 1920x1024	29k 20	✓	✓	✓	✓	✓	✓	✓	✓	✓	-
Leddar PixSet [52]	1440x1080 23 TB	12M 3	✓	✓	✓	✓	-	✓	✓	✓	✓	✓
Oxford RobotCar [53]	1024x1024 336 GB	15k 11	✓	-	✓	✓	-	✓	-	-	✓	-
Waymo [54]	1280x1920 3 GB	1920x1200	✓	-	✓	✓	-	✓	-	-	✓	-
Udacity [56]	1920x1200	2k 8	-	✓	✓	✓	-	-	-	-	✓	✓
TorontoCity [57]	1280x640											

^a Parameters related to the total dataset. ^b Parameters related to the image classification or object detection parts of the datasets, if exist. ^c Annotations can vary from bounding box to segmentation mask, please, see the sources for details. ^d At least one is provided. ^e The common cropped resolution in usage by state-of-the-art networks. ^f Average image resolution.

Frequently used domain-general object detection datasets according to the open literature are Pascal VOC (Visual Object Classes) [44], MS COCO (Microsoft Common Objects in Context) [27], and ILSVRC 2014 [40]. Pascal VOC is a standardised dataset for object recognition, and popular challenge competitions based on it had been run from 2005 to 2012. Pascal VOC 2012 has 11,530 images with 27,450 annotations and 20 object categories. The currently most popular object detection dataset, MS COCO, is a large-scale dataset consisting of 328k images. It is suitable for multiple tasks, such as object detection, segmentation, keypoint detection, and captioning. Another widely used large-scale dataset in detector training is ILSVRC 2014, which is a part of the 2014 ILSVRC competition [40]. It comprises 450k training images with 200 object categories, 20k validation and 40k test images. The main features of the outlined datasets are summarised in Table 2.2.

While there is no doubt of the usefulness of ImageNet [40] and MS COCO [27], their images often represent single-object scenes, which are not representative enough of what is encountered in real-time self-driving situations. Besides that, these domain-general datasets lack the necessary content to support the learning capabilities of ADSs regarding specific self-driving scenarios. For example, adverse weather conditions could lead to poor CNN detection performance in ADSs, if weather-related information has not been present in the training data. In consequence, various domain-specific datasets have been created including different self-driving scenarios to facilitate CNN classification and detection learning capabilities for ADSs.

The most cited self-driving-specific datasets developed in recent years, main features of which can be found in Table 2.2, are briefly overviewed below:

(i) KITTI Vision Benchmark [28] is the most widely cited dataset [45] among researchers and developers since its publication year, 2012. It offers a relatively large amount of self-driving scenes with 2D, 3D, and bird’s eye-view object detection datasets together with annotations.

(ii) UC Berkeley DeepDrive, BDD100K [46] is a recent dataset with 100k annotated videos and 10 different tasks for evaluation of self-driving image recognition algorithms. The dataset comprises more than 1k hours of driving experience and 1M frames. It also provides a large context diversity for self-driving by being strengthened with all-weather, geographic, and environmental conditions in the day/night time.

(iii) CityScapes [47] is a large-scale dataset and focuses on the semantic understanding of urban street scenes with 25k annotated images and 30 classes.

(iv) A2D2 [48] has been released by Audi and includes 41k labelled data with 38 features and 390k unlabelled data. Its size is around 2.3 TB and is split by annotation types, such as semantic segmentation, 3D bounding boxes, etc.

(v) nuScenes [49] has been created by a full sensor suite mounting six 360° cameras, five radars, a 36-beam LIDAR, and a Global Positioning System and Inertial Measurement Unit (GPS-IMU). It contains over 1.4M images with a diversity of driving manoeuvres, traffic situations and unexpected behaviours.

(vi) ApolloScape [50] is a part of the Apollo project, which is an evolutive and ever-growing research project across all aspects of autonomous driving. The project website offers various simulation tools over 80k lidar point cloud, 100k street views, and 1000km trajectories for city traffic.

(vii) Caltech [51] is a pedestrian detection dataset with annotations.

(viii) Leddar PixSet [52] was released in 2021. It consists of 29k frames in 97 sequences with more than 1.3M 3D bounding box annotations, collected by a full Autonomous Vehicle (AV) sensor suite (LIDARs, cameras, radars, and GPS-IMU). What presents this dataset novel is the deployment of a flash LIDAR with the inclusion of the full-waveform data.

(ix) Oxford RobotCar [53] contains over 100 records of a consistent route during a year in Oxford, UK, which brings a long-term diversity approach in data collection. Its sensor suite comprises six cameras, LIDAR, and Global Positioning System and Inertial Navigation System (GPS-INS).

(x) Waymo [54] was collected by Waymo self-driving cars.

The domain-specific datasets EuroCity [55], Udacity [56], and TorontoCity [57] in Table 2.2, feature person instances, mountain views and large-scale urban views, respectively.

Apart from the above, more sophisticated types of training datasets are also available, for example, naturalistic driving datasets, such as NUDrive [58], euroFOT [59], etc., which focus on the human element in autonomous driving: the driver.

2.1.6 Libraries and Frameworks

Compared to other programming languages, Python-based machine learning libraries have been the go-to choice for researchers and developers as they offer a variety

of features and flexibility, which can increase productivity and quality of code writing, implementation, and integration.

The CNN coding process can be divided into two main stages: (i) data pre-processing and (ii) building of the CNN algorithm. Several general-purpose libraries can be employed at the first stage to insert and prepare datasets for the second stage, such as, the multi-dimensional array-processing library NumPy [60], scientific computing library SciPy [61], data frame processing library Pandas [62], etc. A few powerful drawing libraries, such as matplotlib [63], seaborn [64], plotly [65], are available as well.

As regards the second stage, there are a large number of frequently used libraries and frameworks. The most popular ones are summarised in Table 2.3, and some of them are outlined below:

(i) Google’s Tensorflow [66] is an open-source ML library, available since 2015, which has been widely used. It enables the development of highly computational ML tasks by using data flow graphs in which edges signify tensors. Through this, a single Application Programming Interface (API) can distribute the load between multiple nodes, such as GPUs, and CPUs.

Table 2.3: Commonly used libraries with python interface.

LIBRARIES	Written in	Interface	Operating Systems	Features
Tensorflow [66]	Python, C++, CUDA	Python, C/C++, Java, R, Julia, Swift	Windows, macOS, Linux, Android	OpenCL, CUDA, Pretrained Models, Parallel Execution (Multi Node), RNN ^a , CNN, RBM/DBN ^b , Dataflow Graphs
Theano [67]	Python	Python	Cross Platform	OpenMP, OpenCL, CUDA, Pretrained Models, Parallel Execution (Multi Node), RNN, CNN, RBM/DBN
Chainer [68]	Python	Python	Linux, macOS	CUDA, Pretrained Models, Parallel Execution (Multi Node), RNN, CNN
Keras [69]	Python	Python, R	Linux, macOS, Windows	OpenMP, OpenCL, CUDA, Pretrained Models, Parallel Execution (Multi Node), RNN, CNN, Fast Experimentation
PyTorch [70]	Python, C, C++, CUDA	Python, C++, Julia	Linux, macOS, Windows	OpenMP, OpenCL, CUDA, Pretrained Models, Parallel Execution (Multi Node), RNN, CNN, RBM/DBN
MXNet [71]	C++	Python, C++, Java, R, Go, Julia, Matlab, Perl, Clojure, JavaScript, Scala	Linux, macOS, Windows, AWS, Android, iOS, JavaScript	OpenMP, OpenCL, CUDA, Pretrained Models, Parallel Execution (Multi Node), RNN, CNN, RBM/DBN, Portability
Caffe [72]	C++	Python, C++, Matlab	Linux, macOS, Windows	OpenMP, OpenCL, CUDA, Pretrained Models, Parallel Execution (Multi Node), RNN, CNN

^a Recurrent Neural Network; ^b Restricted Boltzmann Machines/Deep Belief Network

(ii) Theano [67] is compatible with numerical computations and simplifies code writing in deep learning. It also provides tight integration with NumPy [60] resulting in quite accurate calculations.

(iii) Keras [69] is another widely used library that can be run on top of Theano or Tensorflow, as it is designed as a high-level API. As such, it offers a productive and user-friendly interface reducing developers' cognitive load.

(iv) PyTorch [70] has been a fast-growing library in recent years. It has been upgraded by the Facebook AI Research Lab (FAIR) with many outstanding novel features, such as scalability for distributed training [21] and cloud support, offering design simplicity from research to production. In addition, it smoothly integrates with the Python data science library NumPy. In fact, PyTorch has been ranked as the most-used framework in ML implementations, reported in the open literature since March of 2019 [73], as shown in Figure 2.3. However, Keras has still a better real-time performance than PyTorch.

(v) Caffe [72] is a research-based library and has been written in C++ with Python interface by Berkeley AI Research (BAIR) and community contributors. Its remarkable features are speed and switching between CPU and GPU by a single flag. It also offers seamless integration with GPU training in image-based datasets.

Other powerful Python-based libraries and frameworks, such as MXNet [71], Chainer [68], ScikitLearn [74], have also been commonly used.

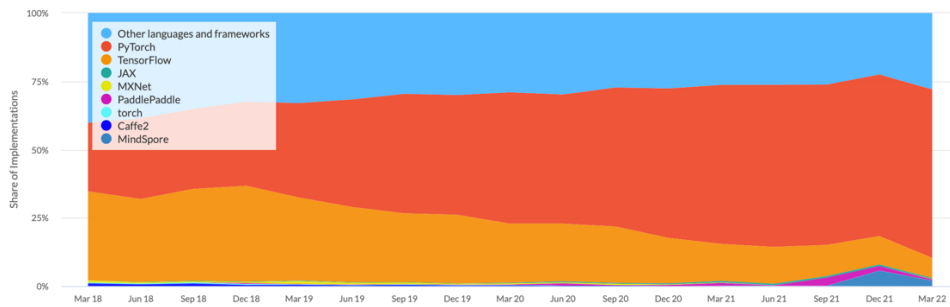


Figure 2.3: Trends in the selection of ML frameworks [73].

2.2 Autonomous Driving Systems

2.2.1 Current Industry Activities

a) Taxonomy of Vehicle Automation

In 2017, the US National Highway Traffic Safety Authority [75] released guidelines for ADSs, in which six levels of automation were specified [76]. These are summarised in Table 2.4 and are outlined below.

Table 2.4: Automation levels.

Levels of Automation	Steering	Acceleration /Deceleration	Lane Change	Certain Conditions	All Conditions	Human Driver
Level 0	-	-	-	-	-	✓
Level 1	✓	✓	-	✓	-	✓
Level 2	✓	✓	-	-	✓	✓
Level 3	✓	✓	✓	-	✓	✓
Level 4	✓	✓	✓	✓	-	-
Level 5	✓	✓	✓	-	✓	-

- Level 0: No automation

All driving tasks must be completed by a human driver, which also means zero autonomy.

- Level 1: Driver assistance

The vehicle is mainly controlled by a driver. Only under limited driving conditions, some driving assistance features, such as steering, acceleration/deceleration, are shared with the automated system.

- Level 2: Partial automation

The vehicle fully controls the steering and acceleration/deceleration even under limited conditions. However, drivers must remain focused on other driving tasks and monitor vehicle surroundings at all times.

- Level 3: Conditional automation

The human driver is not required to monitor the environment as the automated system takes on all driving tasks, including lane change as well. What the driver must still do is to be in a state of readiness to respond to requests arising from the automated system.

- Level 4: High automation

Under certain conditions, the vehicle is capable to handle all the driving tasks. A human driver might have an option to take part in the control.

- Level 5: Full automation

In this level of autonomy, vehicles are capable of performing all the driving tasks under all driving conditions. There is no task assigned to the human driver.

In brief, as seen from Table 2.4, whereas in levels 1 and 2 drivers are still responsible for a few tasks, in levels 3-5 the responsibility for all driving tasks, even lane change, is supposed to be handled automatically in any road in an urban environment (i.e., not under certain conditions). Figure 2.4 shows recent updates to the level definitions above by the Society of Automotive Engineers (SAE) [76].

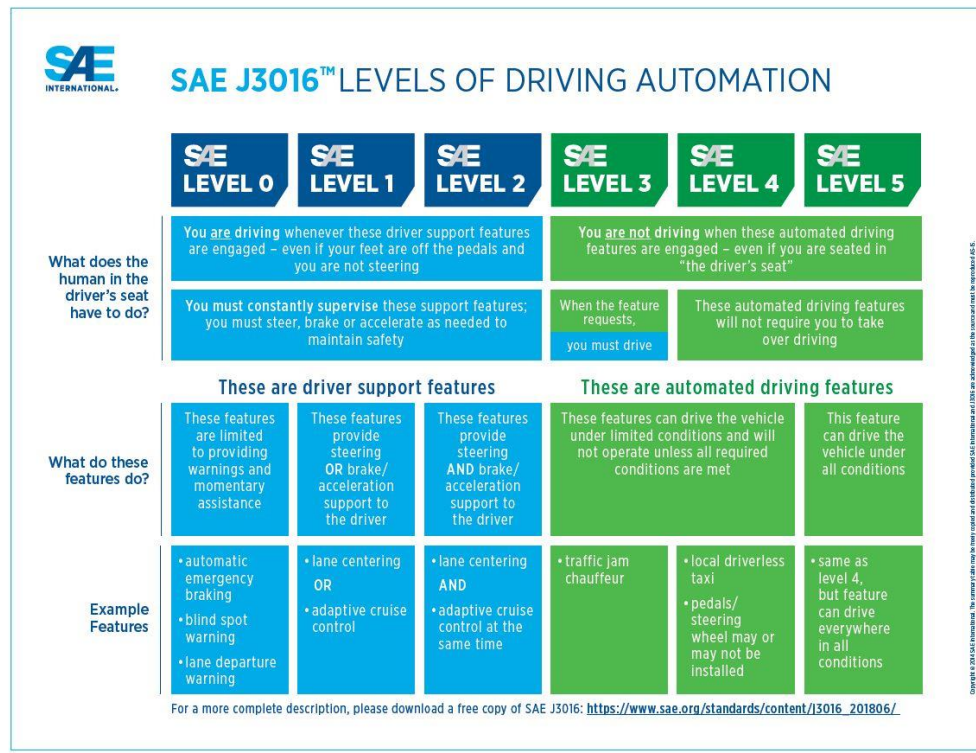


Figure 2.4: SAE recent updates on automation levels [76].

b) Current Status Quo in Industry

This subsection presents findings of a comprehensive investigation on the current status and activities of industry leaders in terms of the achieved level of automation as well as used sensors and computing platforms in the vehicles, in an attempt to establish where automakers stand. A summary of the findings is presented in Table 2.5.

It could be seen from Table 2.5 that even Tesla [17], Audi [77], and Mobileye [18], [19] which are leading automotive companies, have only achieved automation levels 2 or 3, in which the driver must be highly involved in completing the driving tasks. It

would appear that the production process relating to ADSs of many automakers is still under experimentation. In addition, some of them have imposed certain conditions on the use of their ADSs, i.e., driving in confined areas such as specific cities, particular highways, etc. Currently, levels 3-4 can be operated only in limited Operational Design Domains (ODDs) like highways, and Audi claims to be the first company which have produced a level-3 vehicle under the condition that it is driven on a restricted highway [78].

Table 2.5: Current situation of autonomous driving under experimentation.

Leading Automakers	Tesla [17]	Nvidia/Audi [77]	Waymo [79]	Mobileye [18], [19]	Navya [80]	Honda [81]
Automation	level 2	level 3 ^a	level 4 ^a	level 2	level 4 ^a	level 3 ^a
Platform	SoCs GPUs	SoCs GPUs	SoCs GPUs	SoCs FPGAs	SoCs GPUs	SoCs, GPUs, FPGAs
Sensor	camera radar	camera lidar, radar	camera lidar, radar	camera	camera lidar, radar	camera, lidar, radar

^a Under certain conditions.

Although, as shown in Table 2.5, Waymo [79] have reached level 4 driving under certain conditions, there has not been yet any production of a vehicle satisfying levels 3-5 driving on any road in an urban environment. More surprisingly, while one of the automakers, Honda [81], plans to achieve level 3 automation in any road condition by 2029, some others [82], such as, Toyota, Volvo, Ford, etc., seem to skip level 3 automation because of the safety and technology issues. Furthermore, the Toyota Research Institute [83] have stated that there is no one in the industry that has been even close to attaining level 5. Thus, the details in Table 2.5 show that there are still challenges in the production of highly automated vehicles, especially at levels 3-5, which motivates the research community to actively investigate this emerging area.

As for the used sensors, as seen from Table 2.5, Nvidia/Audi [77], Waymo [79], and Navya [80], who have achieved automation levels 3-4 differ from the others by using lidars, which enable high precision as sensing devices send light beams to surroundings. However, the reason why the industry tends to mount cameras instead of lidars is their overwhelming cost, as mentioned in section 1.2. There is also a general tendency for using a combination of SoCs and GPUs based high-performance computing platforms, as shown in Table 2.5, as these can support high-speed processing of large volumes of sensor data.

2.2.2 System Architectures and Components

System architectures of ADSs defining working order can be classified into two types by their connectivity status and algorithmic design. By connectivity status, architectures could be subdivided into two classes: ego-only, i.e., a single vehicle [84], [85] or connected vehicles, i.e., multi-agent systems [86]–[88]. Depending on their algorithmic design, ADS architectures could also be subdivided into two alternative classes, which are: modular [84], [85] or end-to-end driving systems [89]–[91].

a) Connectivity in ADS Architectures

The idea behind the connected systems is based on a communication network among the vehicles (agents). It is expected that this type of system would allow to advance autonomous driving in a more hierarchical way. Even though there is not any kind of operational system of this type at present, in academia, it is notably believed that this emerging technology will take a high proportion of the future ADSs [86]–[88]. Current ever-developing design approaches to connected systems are: Vehicular Ad hoc Network (VANETs) [86], Internet of Vehicles (IoV) [86], Information-Centric Networking (ICN) [86], and vehicle to everything (V2X) [92].

Contrary to connected systems depending on other vehicles on the roads, the ego-only system brings a different approach, whereby all the necessary automated driving operations are managed by a self-sufficient vehicle. At present, this approach is the most preferred system type [84], [85] as it may be that the enormous development challenges force the industry to focus on this system type for now.

b) Algorithmic Designs for ADS Architectures

Modular systems consist of separate linked components that form the pipeline of ADSs, including connections from the sensory inputs to the actuator outputs [33]. It is referred to as a mediated approach in [93]. Figure 2.5 (a) demonstrates a state-of-the-art modular system design, which is based on recent publications [20], [14], [94]. The individual modules in Figure 2.5 (a) are described below.

Unlike modular systems, end-to-end systems generate ego-motion information by directly using the input data provided by the sensors. Ego motion is defined as either the continuous operations on the steering wheel and driving pedals or a set of discrete actions, e.g., acceleration and turning left/right [14]. It is also referred to as a direct reception in

[93]. The pipeline of an end-to-end system is represented in Figure 2.5 (b). Neuroevolution [95], [96], direct supervised deep learning [91], [93], [97], and deep reinforcement learning [90], [98] are three different main approaches for these systems.

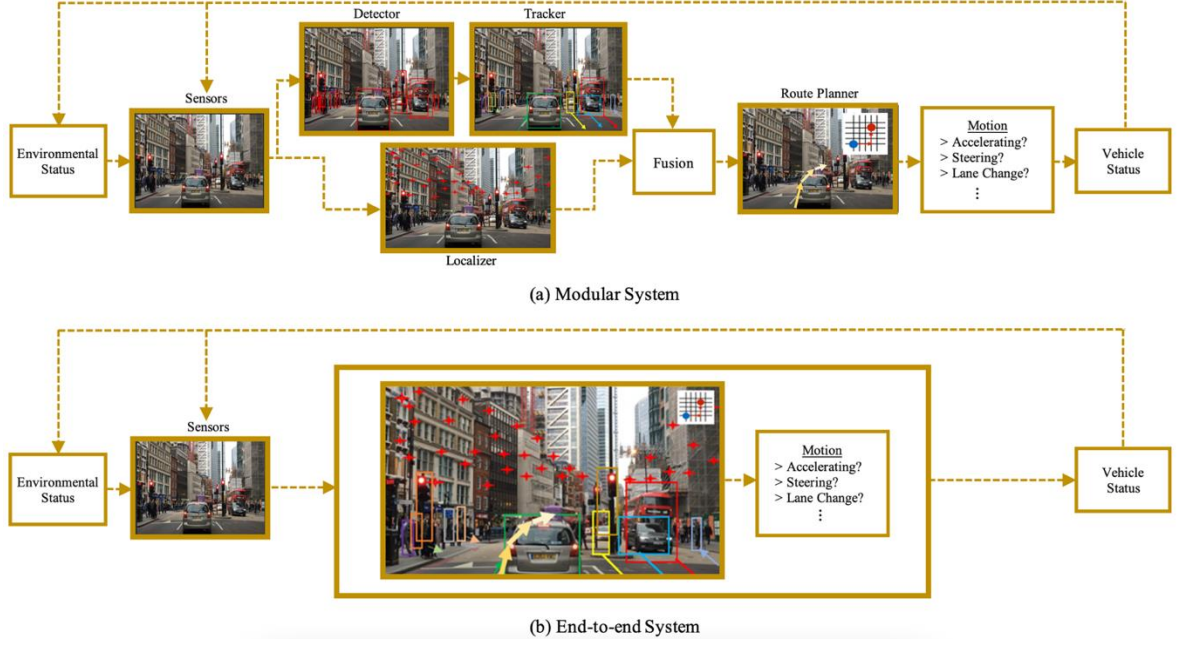


Figure 2.5: Algorithmic design types: (a) modular system; (b) end-to-end system.

Table 2.6 summarises the pros and cons of the two algorithmic designs. The key factor of why automakers prefer modular designs compared to end-to-end design is existing literature, which enables easier production even though there are still many shortcomings, primarily error propagations. End-to-end designs still need many efforts for near-future production. Particularly, hardcoded safety measures and interpretability remain crucial topics for these designs.

Table 2.6: Comparison of design types.

Type	Pros	Cons
Modular	> Convenience in solving the problems [99] > Existing literature for modules enabling transferable info [14] > Feasible integration with reliable architectures [100]	> Error propagation in the pipeline [33] > Complexity [101]
End-to-end	> Offline training option [14] > Better performance during operation [99]	> No real-world implementation yet [14] > Hard coded safety measures [102] > Interpretability [99]

c) Modules

A variety of sensors are used in the onboard data processing pipelines of cutting-edge ADSs. Commonly used types of sensors are exteroceptive sensors, e.g., ultrasonic, radar, lidar, and cameras, which are responsible for perceiving the vehicle surroundings. They are employed in both modular and algorithmic designs, as seen from Figure 2.5 (a) and (b). As previously stated in section 1.2, many leading companies, such as Tesla [17] and Mobileye [18], [19] have mainly focused on camera-based approaches. However, others have used lidar sensors as well, e.g., the Uber car (XC90) [103] deploys 20 cameras and 8 lidars, and VisLab’s BRAiVE [104] has 10 cameras and 5 lidars. The data-flow diagram for modular systems is shown in Figure 2.5 (a).

Data collected from sensors are utilised for two different purposes: (i) perception and (ii) localisation & mapping. The first requires detector and tracker modules, while the second requires a localizer module, as illustrated in Figure 2.5 (a). The perception modules perform the crucial task of perceiving the vehicle environment and extracting meaningful information from the objects. They also play a significant role in a vehicle’s navigation system. Perception has some core sub-tasks which are object detection [105], semantic segmentation [106], road and lane detection [107], [108], and object tracking [109]–[112]. Information extracted by the detector and tracker in the pipeline is combined with the localizer in the fusion module. Then, this information is used in the behaviour prediction module, also known as path planning.

The localization & mapping modules are responsible for finding the vehicle’s ego-position relative to a map or a reference frame taken in the surroundings [113]. By means of localization, the vehicle finds its precise relationship with all of the objects on the map. This task is crucial for ADSs similarly to for any other mobile robotic system [114]. This is because vehicles need to estimate their ego-position with a centimetre-level precision [14]. In the industry, mainly the following five localization techniques are used: SLAM [115], Absolute positioning sensors [116], Odometry/dead reckoning [117], GPS-IMU fusion [118], prior map-based [119]–[121], and many of them are lidar-based approaches. There have been suggestions that the future of localization may be shaped by camera-based approaches as their deployment is more cost-efficient [14].

Lastly, a significant part of the ADSs pipeline is responsible for behaviour prediction and planning, which are produced in a route planner module as shown on Figure

2.5 (a). Planning modules realise two sub-tasks, global planning and local planning. As the name suggests, a global planner finds a route from point of departure to point of arrival by using the road network, whereas local planning tries to perform a defined global plan with as lower as a possible error. In other words, ADSs strive to reach the final destination by finding trajectories avoiding obstacles and satisfying optimization criteria.

In addition, many modern cars are equipped with navigation systems, which plot a global route by utilising GPS or offline maps. Both, the academy community and industry have proposed a number of global planning methods, such as goal-directed path [122], separator based [123], hierarchical [124], bounded-hop [125]. In addition to these methods, some hybrid studies have been published as well [126], [127]. Recently, several methods on local planning have been proposed too, following different approaches, e.g., graph search [128], sampling-based [129], [130], curve interpolation [131], [132], numerical optimization [133]. Apart from these traditional methods, new approaches to planning have emerged based on DL and reinforcement learning [133], [134], which still have many shortcomings that need to be overcome such as interpretability [99], hard-coded safety measures [102], generalization, training data, etc.

In contrast to modular systems, end-to-end systems aim to produce a similar result in one step, as shown in Figure 2.5 (b). However, these systems are still an ongoing research topic and need a lot of development efforts.

2.2.3 ADSs Design Constraints

According to official information, 94% of the total number of road accidents are due to human errors [135]. Against these grim statistics, ADSs promise to reduce traffic accidents, driving-related stress, emissions, and many others [136]. However, ADSs have been the cause of accidents as well. For instance, Google's ADS [137] hit a bus when changing lanes, as the system did not accurately estimate the speed of the bus. As another example, Tesla's Autopilot [138] failed to detect a truck and it caused a fatal accident resulting in the death of the driver. There are many other such collisions with tragic ends [139], [140]. These unfortunate events show that there are still many issues that need to be addressed during the design process of such systems.

The purpose of this section is to discuss the main criteria that should be handled to realise the object detection algorithms designed for ADSs.

Design of ADSs is a trade-off decision process taking into account a number of requirements on different parts of the system, in terms of performance, predictability of

performance, power consumption, storage, thermal constraints, among others [20]. The object detection part of ADSs, for example, must satisfy real-time implementation requirements. In this section, ADSs design constraints will be analysed based on the open literature, after which a trade-off analysis between accuracy and model size will be discussed.

Mainly, four parameters that directly affect the performance of the vehicle are considered when designing a novel CNN for object classification or detection, namely accuracy, model size, inference speed, and efficiency. Increasing the accuracy level of CNNs for both classification [141], [142], [143] and detection [144], [145], [146] has been a mainstream research goal, a part of several works, such as [12], [147], on classification and [148], [105] on object detection. Moreover, what is another trend exemplified by [142], [143], [144], [145] is that the rises in the accuracy have been followed by rises in the model sizes as well. However, while the former is regarded as an added value, the latter is not a favourable result for real-time implementations.

Without a doubt, a novel CNN architecture featuring both low model size and a high accuracy level would bring many advantages to ADSs systems. Some of the benefits of a small model size are low power consumption (efficiency), low latency and high inference speed. It also facilitates over-the-air updates [149], online and distributed training [149], [150], FPGA-based implementation, use of low memory bandwidth [149] and it even leads to lower fuel consumption [20].

Some technical terms about latency and memory types of processors (CPUs) are clarified next. There are two types of CPU memory – internal (on-chip) and external (off-chip) where the former could be of two types: ROM (non-volatile) or RAM (volatile). When the processor needs to process data, it fetches it from an on-chip memory component that is embedded in the processor itself, as shown in Figure 2.6. The on-chip RAM is of an SRAM type and the off-chip memory is of a DRAM type. When the processor needs to process data not available in the on-chip memory, it fetches it from an off-chip memory component that is not a part of the processor itself.

The on-chip SRAM memory allows reducing the latency (delay time), as generally, it supports a single-cycle access time. SRAM performs as a cache memory for the off-chip DRAM as well. The off-chip DRAM has a much greater latency of 1000x cycles more compared to the on-chip SRAM [151] as illustrated in Figure 2.6. Due to its high cost, SRAM memory is preferred at low sizes. Although DRAMs are low-priced memories, they are quite power-hungry due to their internal design features [152].

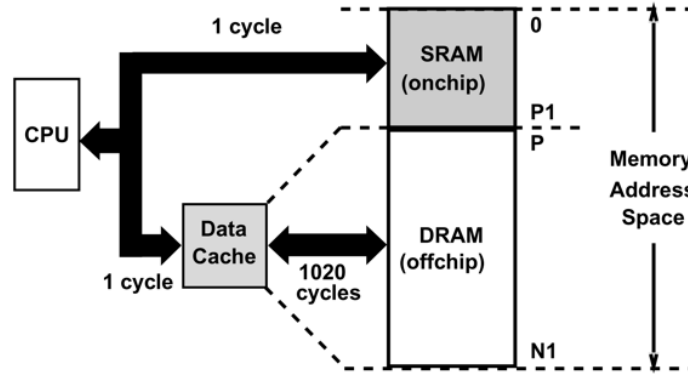


Figure 2.6: Memory hierarchy in processors [151].

a) Performance and Predictability

Being candidates for preventing car accidents, ADSs need the capability of understating their surroundings and reacting to them in an acceptably fast way. Despite the promise for such a potential, the real-time performance requirements for ADSs are still largely undefined.

According to Mody [153], the reaction time of ADSs is defined by two criteria, frame rate and processing latency. While frame rate denotes how fast the data coming from real-time sensors can be fed into ADSs, the processing latency is defined by how fast the system reaction to every piece of fed data (frame) is. Following that, Lin et al. [20] state that ADSs should be able to react or process current traffic events within the latency of 100ms and support a frequency of at least 10 fps in object detection, as well. These metrics are based on the actual real-time performances of the human driver [135], [154]–[157].

In [20], the reaction performance of human drivers is used to systematize the ADSs real-time performance. Even though the human being is a very complicated non-linear system, and its behaviour cannot be assessed easily [158], [159], it is shown in [160], [161] that human ability for image classification outperforms DNNs indisputably. Ideally, in operation, a 100% accuracy rate is expected from object detectors in finding objects of interest. However, such an accuracy, which is on par with human abilities, is yet to be achieved by ADSs, and it is a great challenge in the field. In summary, the only agreed so far target to be reached by ADSs is up to %100 accuracy rate, and the evaluation of human behaviours on detection accuracy is yet to be finalised.

Three performance targets of ADSs were defined so far: processing latency, frame rate, and accuracy. The predictability constraint is determined by the reliability and quickness of the ADSs reactions to the real-time traffic conditions. In general, they are defined

with both temporal aspects, like timing deadlines, and functional aspects, like making the correct operational decisions. Not being able to respond reliably within a specific deadline may endanger passengers and could result in fatal accidents. The performance of ADSs, therefore, needs to be exceedingly predictable to be adopted in real driving scenarios.

b) Power

As mentioned previously in section 1.2, Intel [15] has stated that fully developed ADSs will be required to process approximately 1GB of data per second of its real-time operation. This data needs to be processed very fast so that vehicles can react to their surroundings by keeping real-time performance constraints. Consequently, that results in huge power consumption as well. In addition, as stated in [20], a power-hungry system could cause a significant downgrade of the fuel efficiency of the vehicle, i.e., up to 11.5%.

c) Storage

In localization tasks of ADSs, tens of TBs data are required to store the prior maps [20], [119]–[121]. For instance, a prior map of the USA is 41 TB [162], which need to be stored while driving. The reasons why such prior maps are needed are the lack of precision and accessibility in GPS technology [163], [164].

Apart from the above, there are also many other constraints, such as thermal [20], privacy [15], hardware reliability, etc.

d) Trading-off Accuracy and Size

The low model size of CNN-based object detectors enables many advantages, but their accuracy is not so high. In view of this, here the trade-off between high accuracy and low model size is held.

The benefits of low model size CNN designs come from the feasibility of their implementation on embedded computing platforms. The high performance of GPUs in terms of processing latency is mostly due to the use of large on-chip memories, which have a very short access time. Nevertheless, despite their quite low latency, they have a high-power consumption. This is also confirmed by [20] with experimental results based on testing of a number of real-time implementations of CNN-based object detection, tracking, and extraction using CPUs, GPUs, and FPGAs. Moreover, [20] reveals that GPUs are outperformed by FPGAs in respect of energy consumption.

At the same time, processing latency of CNN FPGA implementations is restricted by the size of the on-chip memory, which often amounts to less than 10MB [149]. Examples of automotive-grade FPGA families are XA Spartan-3A (576KB) [165], XA Artix-

7 XA7A12T (720KB) [166], XA Spartan-7 XA7S75 (3.2MB) [167] for XILINX and Intel Cyclone IV E EP4CE1150 (3.9MB) [168], Intel MAX 10 10M50 (1,6MB) [169], Intel Cyclone 10 10CL055 (2,3MB) [170] for Intel. Even though these FPGAs support access to external off-chip memory, performance of CNN models would be limited by the amount of the on-chip SRAM memory.

As a result, there are many benefits of reducing the model size of CNN-based detectors as well as making them capable of state-of-the-art accuracy rates as seen in Table 2.7. Therefore, there is a need of improving low model size CNN designs by increasing their accuracy to a favourable level. Such developments would expedite the use of CNNs in the ADSs design process by allowing CNN deployment in low-power FPGA-based implementations despite the low on-chip memory resources.

Table 2.7: Advantages of accuracy and model size in CNNs.

High Accuracy	Low Model Size
> High Security [20]. > High Predictability [20]. > High real-time reaction time [105].	> Low power consumption (efficiency). > Low fuel consumption [20]. > Low latency and inference speed. > Capable of over-the-air updates [149]. > Feasible with FPGAs having limited on-chip memory. > Capable to work with low memory bandwidth. [149]. > Capable of online and distributed training [150].

2.3 Summary

The outcome of this comprehensive chapter has provided several contributions, which are: (i) the parameter number calculation of a generic CNN algorithm, AlexNet [21], for each layer type have been introduced in an authentic manner for interested CNN designers to demonstrate how convolution operations affect the computational load; (ii) a piece of comprehensive investigation from a novel viewpoint for the favourite datasets and libraries in the scope of the thesis has been given with functional table demonstrations, regarded as a useful handbook; (iii) the most recent ADSs system architectures and current industry practice have been documented, providing to complete the meaningful data collection from both academy and industry in a synergetic way for next chapters; (iv) the knowledge on what ADSs mostly suffer from has been provided by making trade-off analyses from

a human vs machine and accuracy vs complexity respectively, which determine promising directions aimed at solving the problems of automakers in achieving real-time implementations for new CNN design approaches. Consequently, by means of these studies, a sufficient theoretical background of CNNs and ADSs has been provided for the following chapters. Now, the chapter that follows attempts to determine the current limitations on the CNN-based image classifiers.

CHAPTER 3 CNN MODELS FOR IMAGE CLASSIFICATION

One of the primary objectives of image classification algorithms is to be deployed in systems aimed at solving various computer vision tasks. For instance, they can act as a backbone network, also known as a base network, in various object detection modules. In this respect, a distinctive classification network can play a crucial role in enhancing the performance of detection systems. Therefore, in this chapter, a comprehensive investigation of outstanding CNN models for image classification by means of review materials is carried out to reveal their current limitations.

The CNN models, published in the open literature, are grouped into two classes based on their model size: heavyweight or lightweight designs. They are considered in a chronological order showing the architectural influences inspired by previous designs. This is to facilitate understanding of the design and historical relations of previous architectures, which is essential in improving and designing new CNN models [171]. They are then analysed with a focus on their accuracy performance, parameter number, model size, and inference speed.

This chapter is structured as follows. Section 3.1 introduces accuracy metrics and frequently used different types of convolutions and operations of CNN models. In sections 3.2 and 3.3, the most prominent CNNs for image classification are investigated based on review materials and then in section 3.4, their performance is analysed to determine their current limitations. Section 3.5 reports the results of a multifaceted experimental analysis revealing the individual training and inference performance of the convolutions and operations in sections 3.1, 3.2, and 3.3, using a block-based CNN model, ResNet [22], as the reference network and taking into account the layer-based timing performance. Lastly, section 3.6 concludes the chapter by summarising its importance.

3.1 Introduction

In this section, introductory concepts, such as the Top-1&Top-5 accuracy terms and frequently used different types of convolutions and operations, are described, which are used in sections 3.2 and 3.3.

The image classification accuracy of CNN models on the ImageNet [40] dataset has been evaluated by two core metrics, Top-1 and Top-5, which have been employed as the primary training objectives in the field. The Top-1 accuracy means that the highest probability prediction of the CNN model for an image must be exactly the expected answer or otherwise it fails, while the Top-5 accuracy requires that the top five highest probability predictions of the model must include the expected answer or otherwise it fails. These metrics are frequently used in the performance explanations of the CNN models in the following sections.

There are certainly numerous qualities that affect the CNN design process. However, the types of convolutions and specially designed operations could be seen as one of the most significant design parameters in convolutional networks. In the investigations, it is, therefore, attempted to provide an evolutionary insight highlighting the progressive developments in the types of convolutions and operations. The focal point is mainly on the most popular convolutional processes shown in Figure 3.1, even though, numerous other types of convolutions and operations have been proposed, such as 3D [172], dilated [173], flattened [174], spatially separable [147], etc. The parameters I_D , I_C , K_D , K_C , O_D , O_C , F used in Figure 3.1 and the spatial 2D convolution in Figure 3.1 (a) are introduced in section 2.1. The convolutional processes, shown in Figure 3.1 (b) - (f), are theoretically described in sections 3.2 and 3.3 below, as follows: Figure 3.1 (b) and (c) are explained in section 3.3.7; Figure 3.1 (d) – in section 3.2.1; Figure 3.1 (e) – in section 3.3.12; and Figure 3.1 (f) – in section 3.3.10.

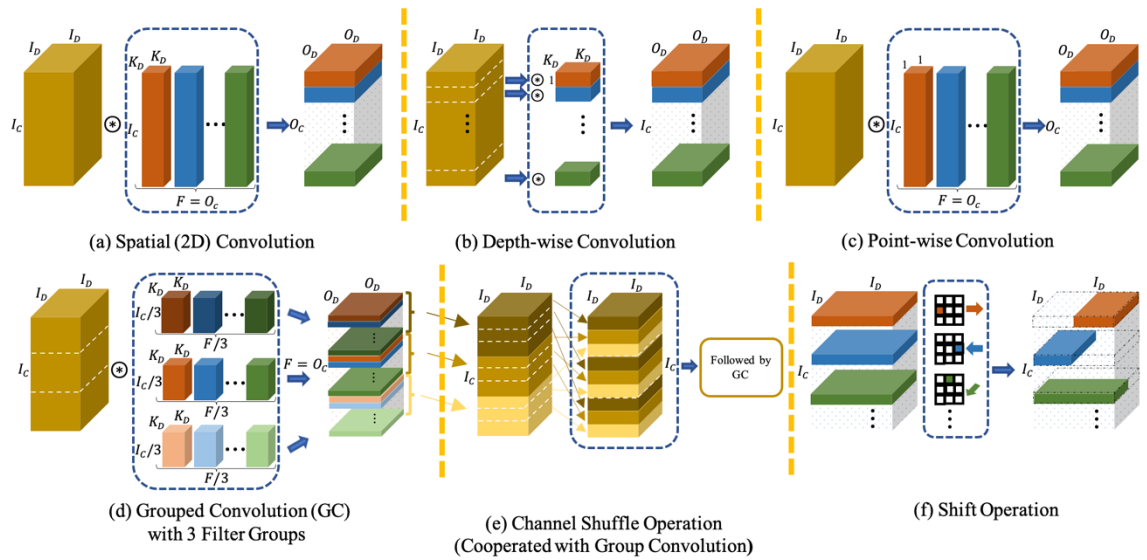


Figure 3.1: Frequently used different types of convolutions and operations.

3.2 Heavyweight CNN Designs

3.2.1 AlexNet

The concept of CNNs was introduced for the first time in 1998 by LeCun et al. [3], [4] and named LeNet-5. The model was developed using a multi-layer artificial neural network including two convolutional, two pooling, and three fully connected layers, and had 60k parameters. As the ReLU activation function hadn't been improved in these years, the TanH activation function was mostly used in the learning process, whereas the sigmoid function was only used in the output layer. The average pooling method was employed rather than max pooling in the pooling layers. The training process was slow taking many days due to the type of processors at the time, however, nowadays this has improved significantly with the use of new-generation GPUs.

LeNet-5 was classifying handwritten digits [175], MNIST (Modified National Institute of Standards and Technology), which is referred to as the ancestor of basic level image classification with 60k training and 10k test fixed-size images. By means of using a backpropagation algorithm [5], it could be trained to detect the patterns from raw pixels and to eliminate a separate feature detection. The method had a 0.95% accuracy rate performance in the dataset. Despite all these advantages, LeNet-5 was ineffective when applied to complex problems such as video classification, which needs a large set of training data and powerful processors for the computation. However, the novelty of the LeNet-5 design is enormous, as it has provided a standard 'template' for almost all subsequent CNNs.

Following the improvements on both compute-bound and memory-bound computing platforms, Krizhevsky et al. [21] upgraded the AlexNet architecture shown in Figure 2.1, and it achieved the best score in the ILSVRC competition based on ImageNet [40] dataset in 2012. Though it was only assembling three more convolutional layers into the LeNet design, there were four notable architectural contributions: (i) the first implementation of the ReLU activation function, enabling six times faster training than the TanH function; (ii) the first deployment of the overlapping pooling technique in the pooling layers; (iii) the application of a data augmentation technique [176]; and (iv) the introduction of the grouped convolution.

AlexNet was implemented on two Nvidia GPUs GTX 580 with 1.5GB virtual memory per node to allow efficient network training as shown in Figure 3.2. The design features two separate spatial 2D convolution paths grouping the convolution operation

with two filter groups, which represents model-parallelization across two GPU nodes, also known as a model-distributed training method. In addition, a compact and efficient version of AlexNet, named CaffeNet, was executed on a single node NVIDIA GeForce graphics card GTX 1080Ti.

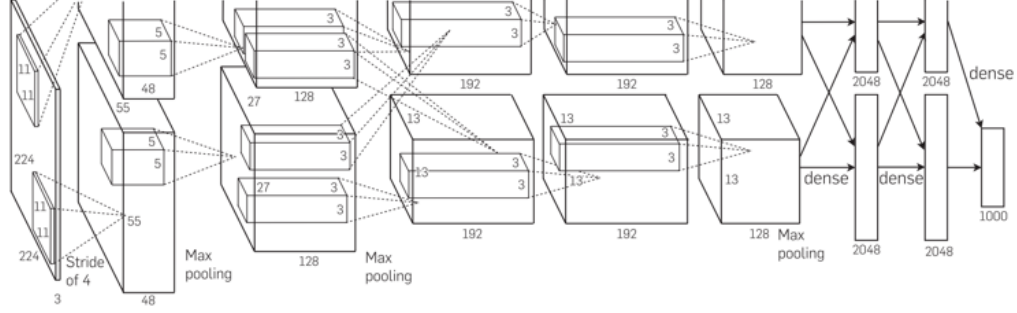


Figure 3.2: Implementation of AlexNet with grouped convolutions on two GPUs [21].

The 2D spatial and grouped convolutions are illustrated in Figure 3.1 (a) and (d), respectively. In a grouped convolution, the number of kernel channels K_C is equal to the number of the input channels I_C , and the number of filters is separated into an equal number of sub-groups, which are responsible for a conventional spatial 2D convolution with the divided group depths. Unlike the AlexNet design with two filter groups in Figure 3.2, the kernel channel number and filter number are divided into three sub-groups as seen in Figure 3.1 (d), whereby each filter group could be defined by the tensor $[K_D: K_D: K_C/3: F/3]$. As a result, each group creates $O_C/3=F/3$ output channels. Overall, three groups create $3 \times O_C/3=3 \times F/3$ output channels, equal to $O_C=F$.

The use of grouped convolutions in networks brings three essential benefits. The first one is making it possible to parallelise the CNN model, which allows executing the convolutions over multiple GPUs. The second benefit is that the network could be fed with more images per step compared to a single GPU implementation. Furthermore, each filter group can learn a unique representation of the fed data achieving more structured learning. The third benefit is the improved efficiency since the number of parameters decreases when the number of filter groups increases. This can be easily shown analytically. In a spatial convolution, the number of parameters is calculated by the expression $F \times K_C \times K_D^2$, whereas in a grouped convolution it could be expressed as $(F/3 \times K_C/3 \times K_D^2) \times 3$. Thus, it can be seen that, with three filters, a decrease in the parameter number by two thirds or 67% is achieved.

It can be observed from the above that grouped convolutions bring substantial advantages compared to standard spatial convolutions, representing another notable contribution of the AlexNet model.

3.2.2 ZFNet

One year later, Zeiler and Fergus [177] proposed ZFNet, the winner of the 2013 ILSVRC competition [40], which significantly reduced the classification error of AlexNet by visualizing the convolutional networks layer by layer and by adjusting layer hyper-parameters, such as filter sizes and strides.



Figure 3.3: Visualization of the layer-based features by deconvolution and unpooling [177].

Although there had been some observation and improvement of shallow layer features, it was the ZFNet authors who investigated deeper features of the pixel domain. Using the deconvolution and un-pooling techniques, they were able to visualize the convolutional networks layer by layer, as shown in Figure 3.3. By means of the deconvolution technique, they also analysed and rearranged several hyper-parameters of their algorithm, which led to reducing the error rate.

In deconvolution, two problems were determined in the first two layers. There was only high and low-frequency information in the first layer, as well as the scarce number of mid-frequencies was causing a chain effect in the learning of some mid-frequency

features. In the second layer, an aliasing problem, occurring in the event of a low level of the sampling frequency, was detected, which was due to the large stride in the convolution operation of the first layer. The researchers fixed the problems by reducing the filter size from 11 to 7 in the first layer and changing the stride step of the first layer to 2 instead of 4. As a result, a valuable performance increase was achieved.

3.2.3 VGGNet

VGGNet [178] is another heavyweight CNN, which was developed by VGG (Visual Geometry Group) from the University of Oxford. It was followed by many other improved models, such as VGG-11, VGG-11(LRN), VGG-13, VGG-16, VGG-16 (Conv1), and VGG-19, built on top of VGGNet. Unlike [21] and [177], which used 11x11 and 7x7 kernels respectively, it was based on the placing of 2x2 and 3x3 convolution kernels and is comprised of thirteen convolutional and three fully-connected layers by deploying the ReLU activation function. The aim was to go deeper into the layers in order to improve the accuracy. This work was highly important as it allowed to reduce the error rates in CNN image classification. However, that entailed a significant increase of the parameters and model size amounting to 138 million (M) and 575 megabytes (MB), respectively, where the model size specifies the required storage space for all parameters of the trained model. Nevertheless, it was the winner of [40] in localization tasks and the runner-up in general.

3.2.4 SENet

The investigated CNN designs so far have been focused on improving the spatial components to strengthen the representational effectiveness in each layer even though central blocks of CNNs might have been used to obtain informative features. From that point of view, Hu et al. [179] from Oxford University designed and proposed SENet as a novel architecture by enhancing channel-wise feature responses. Despite the fact that SENet had improved the accuracy level comparing to the former winners of the ImageNet competitions [40], it had also increased the computational costs slightly bringing the number of parameters to 146M. Its Top-5 error rate was tested as 3.7%, which secured its winning place in 2017 and ever since it has been the best performing classification CNN model.

3.2.5 NASNet

Zoph et al. [180] from Google Brain proposed NASNet. The followed method was initially tracing an architectural building block with a small dataset, CIFAR-10 [23].

That was then applied to a larger dataset, ImageNet [40]. A novel regularization technique, ScheduledDropPath, was also proposed that significantly improved the generalization of NASNet. Their architecture was composed of blocks and cells, called “normal” and “reduction” cell, whereby the former returned a feature map in the same dimension and the latter returned a feature map whose height and width were reduced by a factor of two. As a consequence, NASNet reached the same Top-1 accuracy 82.7% of SENet with a lower parameter number on the ImageNet dataset.

3.2.6 ResNeXt

Until the emergence of ResNeXt [181], the state-of-the-art image classification models had relied on supervised pre-training [182], and the ImageNet dataset had been widely used. In this study, Mahajan et al. presented a peerless pre-training scheme to predict hashtags for social media images. Through the application of their unique transfer learning technique, they demonstrated a remarkable performance of ResNeXt on ImageNet classification, and thus, it was listed as the winner of the 2018 competition [40].

3.2.7 EfficientNet-L2

In EfficientNet-L2 [183], a semi-supervised learning approach [184], named Noisy Student Training, was introduced under the assumption of an abundance of labelled data. The work draws on the machine learning method of self-training [185] and the data compression technique of distillation [186] (More information about machine learning methods and compression techniques is provided in Appendix A.4).

Firstly, the EfficientNet [187] model is trained with labelled images on ImageNet, and then, the model is used as a teacher in producing pseudo labels for 300M images. Next, a larger model of EfficientNet is trained as a student with a combination of labelled and pseudo images, produced in the first stage. Following that, the same process is applied iteratively by adding to the student different types of noise, such as dropout, stochastic depth, and data augmentation, to be able to learn better than the teacher. Thus, the performance of this model on the ImageNet [40] dataset proved to be explicitly outstanding in 2020 with a Top-1 accuracy of 85.5%.

3.2.8 FixEfficientNet-L2

FixEfficientNet-L2 [188] is an upgraded model of EfficientNet-L2 [183], reported by Touvron et al. from the Facebook AI Research group, and it utilises the augmentation technique proposed by the same researchers in [189], which is aimed at reducing the discrepancy between objects by using different resolutions at testing and training time. The

method has resulted in notable performance enhancements in different existing algorithms, for example achieving Top-1 accuracy in ResNet-50 [22] - up to 79.8%, ResNetXt-101 32x48d [181] - up to 86.4%, and EfficientNet-L2 [183] - up to 88.5%. On the other hand, EfficientNet-L2 and FixEfficientNet-L2 have been one of the weightiest heavyweight algorithms in terms of parameter number, amounting to 480M.

3.2.9 Vit-H/14

Dosovitskiy et al. [141] from Google Research have brought to life the idea of the Transformer architecture [190], which is mainly used in natural language processing tasks, for image classification tasks. Their model, Vit-H/14, has shown a striking result with an 88.6% Top-1 accuracy on the ImageNet [40] dataset. The model's accuracy, thus, is ranked as the 2020's highest in Table 3.1.

3.2.10 LambdaResNet200

LambdaResNet200 [191] is a quite recent CNN architecture, which was reported at the 2021 ICLR conference. It introduces novel lambda layers, which enable a 4.5x faster training on modern ML accelerators compared to EfficientNet-L2 [183]. Along with the training performance, its Top-1 accuracy performance on [40] has been listed as 84.3%, which is excellent as it keeps a low model parameter number.

3.2.11 Meta Pseudo Labels

Google AI [142] introduced a novel semi-supervised learning method, named Meta Pseudo Labels. The idea improves on the student-teacher concept in [183], whereby the teacher network generates pseudo labels by using unlabelled data [192] to teach a student. The difference is that the teacher is not fixed, and it is constantly adapted by means of feedback that comes through the student network performance on a labelled dataset [40]. That produces better pseudo labels compared with the method in [183]. In this way, EfficientNet-L2 and EfficientNet-B6-Wide [183], when trained with the method, achieved the highest Top-1 scores of 90.0% and 90.2% accuracy performance in 2021, respectively. However, the parameter number is also correlative with the results reported in [183] and [141], which are the highest among the algorithms introduced above.

3.2.12 Top- k DiffSortNets

In machine learning, the classification accuracy of the state-of-the-art models, particularly on ImageNet [40], has been determined by a core metric, Top- k , where k is a

positive integer. Based on this metric, Top-1 and Top-5 have been the core training objectives. In contrast to the conventional recipe, Petersen et al. [193] propose a relaxation on this assumption, which is that the proposed models are simultaneously optimized with multiple k values instead of a single k . The approach is based on a novel classification loss, differentiable Top- k cross-entropy, which leverages the training of the network by considering not only Top-1 classification predictions but also Top-2 and Top-5 predictions. The evaluation results prove that the approach produces better Top-1 (88.4%) and Top-5 (98.7%) accuracies with lower parameter numbers on Top- k DiffSortNets.

3.2.13 RevCol

In this study, a new design paradigm, Reversible Column Network (RevCol) [194], is proposed to improve CNN performance, which is also expected to inspire further novel CNN designs. The design paradigm assembles multiple subnetworks, *columns*, where reversible connections among them are employed. This connectivity pattern attributes a unique behaviour during forward propagation in that the original information (i.e., the input) can flow through the network without being compressed or distorted as exist in other CNNs. The conducted experiments on the models built by the proposed design paradigm achieve competitive accuracies, particularly on larger datasets, such as ImageNet [40]. Thus, the one, RevCol-H, achieves 90.0% Top-1 accuracy.

3.2.14 BASIC-L

In another recent study [143] by Google researchers, a novel optimization algorithm is formulated for deep neural network training. They present an effective and simple algorithm, Lion (EvoLved Sign Momentum), which provides much more memory efficiency compared to Adam [195], a type of adaptive optimizers, by only keeping the track of momentum. Lion is also different from adaptive optimizers in that its updates produce the same magnitude for each parameter. The performance of the proposed Lion optimizer on ImageNet [40] has been evaluated on an existing network, BASIC-L [196], and achieved the highest Top-1 (91.1%) and Top-5 (99.0%) accuracy ever (see Table 3.1).

3.3 Lightweight CNN Designs

3.3.1 Inception-v1 (GoogLeNet)

Inception-v1, also known as GoogLeNet, proposed by Szegedy et al. [37], is a distinctive and lightweight architecture, which is based on [197], [198]. The authors introduced the inception modules, which comprise convolutional kernels of different sizes

operating over the same input, and then stacking all the outputs from the different kernels. Using convolution kernels of different sizes, the architecture was able to capture all sorts of features, containing 22 layers with 6.8M parameters. It has been the first architecture ever using 1×1 convolution kernels at the middle of the network to reduce dimensionality, parameter numbers, and computational budget in layers, and at the same time increasing non-linearity. Global average pooling was only used at the end of the network instead of every layer. ReLU was the deployed activation function. By going deeper with these inception modules, it was ranked as the winner of the 2014 ImageNet competition [40].

Inception-v1 also introduced two auxiliary classifiers to improve the classification performance in the lower stages of the classification process and to rise the backpropagation gradient signal, as well as for additional regularization. The auxiliary networks are only employed for training and are not used in testing or at inference time.

3.3.2 Inception-v3

Following the publication of Inception-v1 [37], Szegedy et al. [199] designed a novel architecture called Inception-v3. The study first introduced the Inception-v2 model with a deep learning method, called batch normalization, which was created for normalizing the value distribution prior to the next layer, allowing to increase the accuracy performance and training speed. Keeping that novelty and introducing an additional factorization technique have led to the release of the third version, Inception-v3.

Inception-v2 had some flaws in the loss function, which was adding batch normalisation to the auxiliary classifiers in its layers, and in the optimization method (Appendix A.1 provides more details on optimization algorithms). Inception-v3 incorporated these weaknesses but avoided computational problems by using the factorization method, which factorises the $n \times n$ kernels to $1 \times n$ and $n \times 1$ asymmetric kernels. This led to a reduction in the parameter number of the architecture without compromising network efficiency. For instance, by using a 5×5 kernel in a layer, the number of parameters were equal to $5 \times 5 = 25$. However, by using two layers of 3×3 kernels, the number of parameters amounts to $3 \times 3 + 3 \times 3 = 18$. Thus, the number of parameters was reduced by 28%.

Other specific features in Inception-v3 were the use of only one auxiliary classifier, aimed for regularization and batch normalization, and employing efficient grid size-reduction techniques instead of using a pooling layer. Due to its novel efficient properties, the Inception-v3 architecture was able to achieve increased accuracy levels on [40], as detailed in Table 3.1.

3.3.3 ResNet

As it could be observed from the architectures discussed until now, the increase of the number of network layers leads to achieving better accuracy results. Taking this on board, the Microsoft researchers He et al. designed the ResNet model [22], including five networks with different layer numbers from varying 18 to 152.

A ResNet architecture comprising 50 layers is illustrated in Figure 3.4, where bottleneck modules comprise two 1x1 and one 3x3 spatial convolutions with varied channel numbers. There are residual (also known as identity/projection/shortcut) connections, allowing to connect sequential modules to increase feature reuse. The primary contributions of the residual connections were the prevention of the vanishing gradient problem in the first layers and the reduction on the degradation problem. By means of going deeper in the network, ResNet-50 was ranked as the winner of 2015 ILSVRC [40] with a 93.3% Top-5 accuracy performance.

3.3.4 Inception-v4

The high performances of the Inception models [37], [199] and the high impact on the performance of the residual connections in ResNet [22], have raised the question as to whether there is an advantage of combining the Inception models and the residual connections. Inspired by that idea, Szegedy et al. [200] proposed the Inception-v4 architecture, which is an empirical study demonstrating that residual connections considerably accelerate the training performance of Inception networks. Moreover, it was proved that the training process might be facilitated by deploying an activation function scaling technique. As a result, Inception-v4 achieved a 3.8% Top-5 error rate in ImageNet ILSVRC competition [40]. Inception-ResNet-v2 was another model proposed as part of the same study, which even outperformed Inception-v4.

3.3.5 Trimps-Soushen

The Trimps-Soushen model was the winner of ILSVRC 2016 [40], with a 2.99% Top-5 error rate. However, the creators of the Trimps-Soushen architecture did not publish any technical report or paper. They only released the testing results of their model on the ImageNet dataset in a presentation at the 2016 ECCV workshop. Nevertheless, the results have been approved and listed by ImageNet.

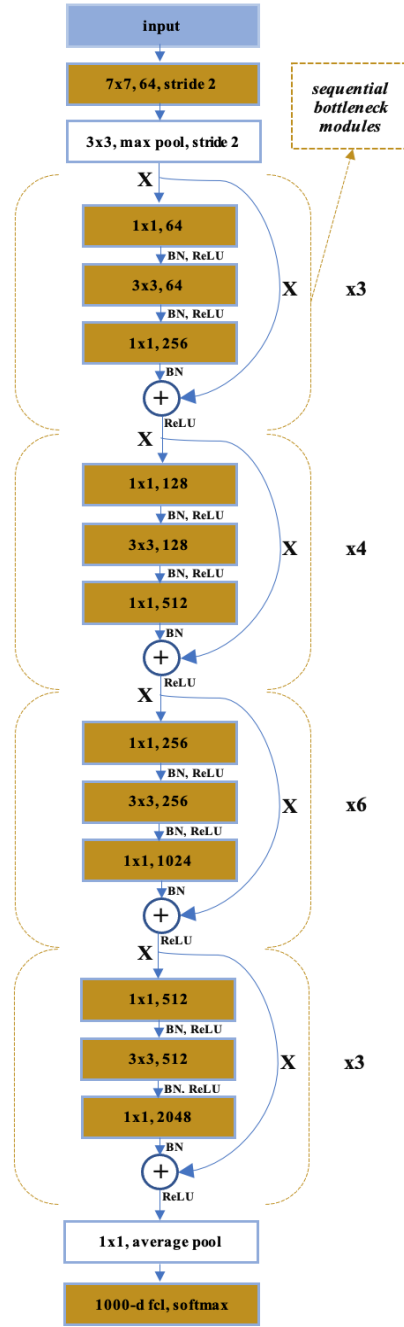


Figure 3.4: ResNet-50 architecture with bottleneck modules [22].

3.3.6 SqueezeNet

Iandola et al. [149] proposed a novel compact model, called SqueezeNet, based on the AlexNet architecture [21]. They achieved a remarkably low model size of 0.5MB, keeping the AlexNet accuracy levels. In order to obtain that result, 3 steps were followed. Initially, the 3x3 spatial 2D filters were mostly changed with 1x1 filters providing lower parameter numbers. Following that, squeeze layers were used to reduce the number of

input channels to 3x3 filters, and finally, for utilising large activation maps in the convolutional network, a late downsampling technique was applied.

As a result, the same level of accuracy as in [21] was achieved with x50 fewer parameters, and thus, SqueezeNet has been one of the most striking lightweight algorithms.

3.3.7 MobileNets

In 2017, Howard et al. [147] from Google Inc. introduced the MobileNets CNN, which was focused on reducing model size and complexity, targeting mobile and embedded vision applications. Depth-wise separable convolutions in Figure 3.1 (b) and (c) form the basis of MobileNets. Compared to a same spatial convolution structure, the parameter numbers were reduced by means of depth-wise separable convolutions. But, neither the model size of SqueezeNet [149] nor its parameter number of around 1.2M were outperformed by MobileNets. However, in terms of the accuracy performance, MobileNets has shown a distinct enhancement with 13.1% higher Top-1 accuracy than SqueezeNet.

For a simple representation of a spatial convolution like the one expressed by equation (2-1), it is supposed that the input/output spatial dimensions are identical, and the stride step is 1. Thus, the output of the depth-wise convolution, $\hat{O}_{k,l,m}$, in Figure 3.1 (b) can be computed by

$$\hat{O}_{k,l,m} = \sum_{i,j} \hat{K}_{i,j,m} I_{k+\hat{i}, l+\hat{j}, m} , \quad (3-1)$$

where $\hat{K} \in \mathbb{R}^{D_K \times D_K \times I_C}$ denotes a depth-wise convolution kernel.

Likewise, the point-wise convolution, $O_{k,l,n}$, in Figure 3.1 (c) is defined by

$$O_{k,l,n} = \sum_m \hat{K}_{m,n} \hat{O}_{k,l,m} , \quad (3-2)$$

where $\hat{K} \in \mathbb{R}^{1 \times 1 \times I_C \times F}$ is a point-wise convolution kernel.

Similar to grouped convolutions in section 3.2.1, the reason why depth-wise convolutions are selected is related to their efficiency, as they reduce considerably the parameter number and computational cost. This is because, in the standard spatial convolutions, the parameter number is defined by $F \times K_C \times K_D^2$, whereas the parameter number

in a depth-wise convolution followed by a point-wise convolution is calculated by $(K_C \times K_D^2) + (F \times 1 \times 1)$, which is quite less. As to the computational cost comparison, the FLOPs number is defined as $F \times K_C \times K_D^2 \times O_D^2$ in spatial convolutions and as $(K_C \times K_D^2 \times O_D^2) + (F \times K_C \times 1^2 \times O_D^2)$ in depth-wise separable convolutions. These two expressions can be evaluated by using identical dimensions. They could also be compared by the ratio of the number of FLOPs between the depth-wise separable convolution and the 2D convolution, which is defined by

$$\frac{1}{F} + \frac{1}{K_D^2} . \quad (3-3)$$

It is evident from the above expression that it has a fractional value, which shows that the parameter number and computational budget of depth-wise separable convolutions are significantly more efficient compared to spatial convolutions.

3.3.8 Xception

Xception is another model developed by Google [201], which was inspired by Inception-v3 [199], discussed in section 3.3.2 above. It employs depth-wise separable convolutions that generally consist of depth-wise and point-wise convolutions, as illustrated in Figure 3.1 (b) and (c). The Inception modules in the Inception-v3 were replaced with depth-wise separable convolutions instead of spatial 2D convolutions, which improved its performance. In spatial convolutions, whereas 1x1 convolution kernels provide for cross-feature map correlations, 3x3 and 5x5 convolution kernels support spatial correlations. In depth-wise separable convolutions, these correlations are provided without even using mid-level activations, and due to that Xception has outperformed Inception-v3.

3.3.9 DenseNet

Most of the proposed networks up to this point contain short connections between layers, where one step skip connections, referred to as identity mapping in ResNet, are used for feature reuse to prevent the vanishing gradient and the degradation of information problems. The vanishing gradient problem is substantially addressed by ResNet [22], which also reduces the degradation problem of networks to a reasonable level. At this point, DenseNet [24] handles the degradation problem further with novel dense skip connections instead of one step skip connections. In its connectivity pattern, the input of each

layer in a dense block is the concatenation of the outputs of all preceding layers. Figure 3.5 demonstrates an example of this feed-forward fashion, where H_i is defined as a nonlinear and composite function of BN, ReLU, and convolution operations and $x_1 - x_4$ contain four channels and called as growth rate in DenseNet. Thus, DenseNet provides three crucial breakthroughs: (i) alleviation of the vanishing gradient problem; (ii) substantially prevention of degradation by strengthening the feature propagation; and (iii) lower computational complexity. The experimental result of one of its models, DenseNet-169, is shown in Table 3.1.

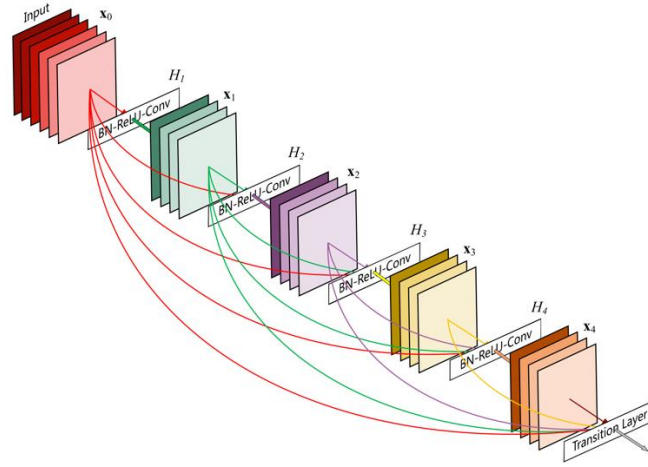


Figure 3.5: An example of dense block with five layer [24].

3.3.10 ShiftNet

The popularity and extensive usage of depth-wise separable convolutions in CNNs led to several further research works, including a remarkable approach, proposed by Wu et al. [12]. In theory, depth-wise separable convolutions seem to improve the parameter number and computational cost of CNNs. This could be demonstrated by the ratio between computational cost and memory access, which is expressed in spatial 2D convolutions as follows:

$$\frac{F \times K_C \times K_D^2 \times O_D^2}{I_D^2 \times (K_C + F) + K_D^2 \times K_C \times F} , \quad (3-4)$$

whereas the ratio for depth-wise separable convolution is the following:

$$\frac{K_C \times K_D^2 \times O_D^2}{I_D^2 \times (2K_C) + K_D^2 \times K_C} , \quad (3-5)$$

where the input/output dimensions are identical.

A lower ratio indicates that memory accesses take more time, causing several orders of magnitude slower operation and higher energy consumption per floating-point operation. Thus, this drawback prevents I/O-bound devices to perform to their maximum computational ability. Based upon this viewpoint, a novel convolutional operation, named shift, was proposed.

The shift operation, illustrated in Figure 3.1 (f), can be viewed as a special type of depth-wise convolutions, the output of which, $\tilde{O}_{k,l,m}$, could be expressed by

$$\tilde{O}_{k,l,m} = \sum_{i,j} \tilde{K}_{i,j,m} I_{k+\hat{i}, l+\hat{j}, m} , \quad (3-6)$$

where the kernel of the operation is a tensor $\tilde{K} \in \mathbb{R}^{K_D \times K_D \times I_C}$ and every kernel can be represented as follows:

$$\tilde{K}_{i,j,m} = \begin{cases} 1, & \text{if } i=i_m \text{ and } j=j_m , \\ 0, & \text{otherwise} . \end{cases} \quad (3-7)$$

In equation (3-6), the i_m and j_m indices depend on channels and assign arbitrarily one of the values to 1 in $\tilde{K}_{:, :, m} \in \mathbb{R}^{K_D \times K_D}$ called shift matrix. Thus, K_D^2 possible shift matrices exist and any of them corresponds to a shift direction.

The shift operation could be seen as a bundle of memory operations shifting the input tensor channels into certain directions by convolving with shift kernels. In contrast to spatial 2D and depth-wise separable convolutions, the shift operation does not bring any overhead in terms of parameter or FLOPs cost due to fusing a point-wise convolution after it. This allows the point-wise convolution to directly fetch the shifted data from the cache. Thus, the shift operation has allowed a lightweight CNN model to achieve a remarkable performance on both parameter number and accuracy in [40], as seen from Table 3.1.

3.3.11 MobileNet-V2

A year later, the MobileNet’s creators, Howard et al. [202], proposed a novel algorithm, MobileNetV2, improving the performance of MobileNets by revising the residual structure where skip connections were added between the bottleneck layers. Via those structural changes, MobileNet-V2 improved the Top-1 accuracy performance to 72.0 % and the parameter number to 3.4M in comparison with MobileNets. In addition, MobileNet-V2 was not only improved for image classification but was also designed for object detection tasks.

3.3.12 ShuffleNet

ShuffleNet [26] as a computation-efficient algorithm was designed for mobile devices. It is inspired by and linked to the grouped [21], and depth-wise [147], [201] types of convolutions. In a nutshell, shuffled grouped convolutions involve a grouped convolution combined with a channel shuffling.

As explained earlier in section 3.2.1, group convolutions significantly reduce the number of the total operations. Nevertheless, there is a drawback that each filter group could operate over a certain fixed portion of information from the previous layers, as shown in Figure 3.1 (d). As such, filter groups are limited in the learning of few features, which weakens the representation and information flow throughout different channel groups. Channel shuffling overcomes that problem by mixing up the information between channels. In Figure 3.1 (e), the feature map with three channels, $I \in \mathbb{R}^{I_D \times I_D \times I_C}$ obtained subsequently to the first grouped convolution via three filter groups in Figure 3.1 (d), is first divided into several subgroups, and then these subgroups are mixed up. Following the shuffling, the usual second grouped convolution is performed with the difference of strengthening the representation and information flow between channel groups.

In ShuffleNet, the point-wise grouped 1x1 convolution is also considered instead of the 3x3 convolutions employed in [147], [201]. The idea behind that was the lower computational efficiency of the 1x1 point-wise convolutions. The operation is identical to the grouped convolutions with a minor alteration of the used kernel size.

As a result, the model utilized grouped, depth-wise, and point-wise convolutions enabling a computation-efficient and lightweight algorithm while maintaining the accuracy level of 86.7 % in [40] as seen from Table 3.1. Thus, it has gained popularity in CNNs for mobile devices.

3.3.13 SqueezeNext

After the SqueezeNet model [149] reduced dramatically the parameter number of AlexNet [21] by keeping similar accuracy levels, Gholami et al. [203] introduced a novel family of CNN architecture, SqueezeNext, which enables the same accuracy with 112x fewer parameters. There are different types of SqueezeNext models with varied parameter numbers between 1.5 and 0.54M. One of them, the 1.0-SqNxt-44 model, has achieved a 5% better Top-1 and Top-5 performance compared to SqueezeNet with the same number of parameters.

Their design was based on the lower rank filters and compression of the redundant parameters of SqueezeNet, which do not affect the accuracy. Moreover, the architecture used a final bottleneck layer cooperating to reduce the input channel size of the last FCL in SqueezeNet. The application of these microarchitecture level strategies has allowed to achieve a considerable reduction of the parameter number (More details about micro/macro architecture design parameters are explained in Appendix A.2).

3.3.14 ColorNet

Another interesting finding has been explored by Gowda and Yuan [204]. They have focused on the importance of colour spaces of RGB images in datasets and shown that colour spaces, especially transformations of RGB images, are able to significantly improve classification accuracy. By using that idea, their architecture, named ColorNet, takes RGB images as input and converts the images into 7 different colour spaces. Following that, every colour space is used as an input to individual DenseNet modules [24]. By applying this method, the 84.6% Top-1 accuracy performance of ColorNet on ImageNet [40] was listed as the winner in 2019 on Table 3.1.

3.3.15 MobileViTv3

MobileViTv1 [205] is the first study proposing a novel approach exploiting vision transformers (ViT) to create lightweight CNN models for mobile vision tasks. Despite the competitive state-of-the-art results of MobileViTv1, it created image scale challenges and had complexity in the learning. At this point, MobileViTv3 [206] has addressed the problems of it by removing the fusion blocks and using linear complexity transformers [207]. As a result, MobileViTv3-1.0 has achieved state-of-the-art performance on Top-1 (79.8%) accuracy with the lowest model size (5.1M) ever, as seen on Table 3.1.

3.4 Performance Analysis of Image Classification CNN Models

The image classification performance of the investigated CNNs trained on ImageNet [40] dataset is summarised in Table 3.1, in which the names of the lightweight models are shaded in light grey colour, while the names of the heavyweight models – in dark grey colour. For each model, the following data are provided: literature source, year of introduction, Top-1 accuracy, Top-5 accuracy, number of parameters, number of FLOPs, model size in MB, and number of layers. An explicit observation is that the heavyweight models achieve better accuracy levels mostly by increasing the layer and parameter numbers without regard for the model size or energy consumption. For instance, ViT-H/14 [141], and BASIC-L [143], which have achieved the best ever Top-5 accuracy of 99.0%, have a high layer number of 132 and 126, and a very high parameter number of 632M and 2440M, respectively. Besides that, BASIC-L [143] has also achieved the highest Top-1 accuracy ever with 91.1%. Another, MPL-EfficientNet-L2 [142], which is a deeper CNN model than ViT-H/14 and BASIC-L, has obtained the second-best score on Top-5 accuracy of 98.8% and it has reached the second highest Top-1 accuracy of 90.2% by including 154 network layers and having 480M parameters. As a result, what stands out as the main trend in the heavyweight design history as seen in Table 3.1 is the continual growth in the parameter and FLOPs number.

On the other hand, the lightweight CNNs in Table 3.1 aim to reduce the model sizes (parameter number) by conceding their accuracy performance, for example ShiftNet [12], SqueezeNet [149], SqueezeNext [203], and MobileViTv3 [206]. In SqueezeNet, the layer number has been reduced to 19, the parameter number to 1.2M, and the model size to 0.5MB by utilising 1x1 2D spatial convolutions. SqueezeNet was implemented on an FPGA with 10MB of on-chip memory without using off-chip memory [208]. As the best parameter number performance in Table 3.1, ShiftNet has benefitted from a parameter-free shift operation in its layers achieving AlexNet accuracy [21] with only 0.8M parameters. As a quite recent study, MobileViTv3 [206] has achieved 79.8% Top-1 accuracy with only a 5.1M parameter number, which seems the best trade-off between model size and accuracy in Table 3.1. These breakthroughs in the aforementioned CNN developments are substantially enabled by the newly proposed convolution and operation types, which shows their importance and impact on enhancing performance. However, it is beyond dispute that the accuracy performances of the lightweight CNN designs are considerably lower compared to the heavyweight models.

Table 3.1: Performances of image classification algorithms on ImageNet.

Model	Year	Top-1 %	Top-5 %	Parameters ($\times 10^6$)	FLOPs ($\times 10^6$)	~Model Size ^a (MB)	Layers
LeNet-5 [4] ^b	1998	95.5	98.5	0.06	-	-	5
AlexNet [21]	2012	57.2	80.3	62	720	240	8
ZFNet [177]	2013	73.9	88.3	60	630	170	8
VGGNet [178]	2014	68.5	88.5	138	15500	575	19
Inception-v1 [37]	2014	77.4	93.3	6.8	1550	53	22
Inception-v3 [199]	2015	82.8	94.5	23.6	-	-	159
ResNet [22]	2015	79.3	94.7	23.7	3800	-	50
Inception-v4 [200]	2016	82.3	96.2	43	1650	83	144
Trimps-Soushen [40]	2016	82.3	97.1	19	-	-	89
SqueezeNet [149]	2016	57.5	80.3	1.2	352	0.5	19
Xception [201]	2017	79.0	94.5	22	1200	88	126
DenseNet [24]	2017	76.2	93.15	14.1	6700	-	169
ResNetXt-50 [209]	2017	77.7	93.8	22.2	4100	-	50
SENet [179]	2017	82.7	96.3	146	3870	210	201
MobileNets [147]	2017	70.6	89.9	4.2	569	16	29
ShiftNet [12]	2017	58.8	82.0	0.8	279	-	44
MobileNetV2 [202]	2018	72.0	90.1	3.4	300	300	114
NASNet Large [180]	2018	82.7	96.2	89	2380	343	32
ShuffleNet [26]	2018	67.6	86.7	2.5	140	75	50
SqueezeNext [203]	2018	60.3	83.5	0.9	310	-	112
ResNeXt [181]	2018	85.4	97.6	829	153000	458	101
ColorNet [204]	2019	84.6	96.1	19	250	26	-
EfficientNet-L2 [183]	2020	85.5	97.5	480	23500	-	154
FixEfficientNet-L2 [188]	2020	88.5	98.7	480	23500	-	154
ViT-H/14 [141]	2020	88.6	99.0	632	32120	534	132
LambdaResNet200 [191]	2021	84.3	96.3	42	34000	-	200
MPL-EfficientNet-B6-Wide [142]	2021	90.0	98.7	390	-	-	-
MPL-EfficientNet-L2 [142]	2021	90.2	98.8	480	-	-	154
Top- k DiffSortNets [193]	2022	88.4	98.7	89	1600	339	101
MobileViTv3-1.0 [205]	2022	79.8	93.5	5.1	4220	-	128
RevCol-H [194]	2023	90.0	96.8	2158	253700	-	-
BASIC-L [143]	2023	91.1	99.0	2440	287000	-	126

^a Model size means the required number of bytes to store all of the trained model's parameters; ^b Accuracy values of LeNet-5 for MNIST dataset [175].

The explained trends on both sides above bring some limitations. For instance, a major downside of the heavyweight CNN designs is that their large parameter numbers prevent them from being implemented on computational devices with limited off-chip memory, such as FPGAs, ASICs, SoCs, etc. Regarding the lightweight models, their accuracy performances in comparison with the heavyweight models do not meet the industry standards to be deployed in real-time because of safety issues.

3.5 Analysis of Convolution and Operation Types

Due to the use of CNNs for image classification in a wide range of applications, various studies have proposed techniques to reduce their high storage overhead and computational cost, resulting in a compact and accurate model design. As part of that, it has been recognised that carefully designed convolution kernels, which serve as a basic component of the network layers, as the ones are shown in Figure 3.1, can bring significant benefits. Some of them are accuracy-oriented, whereas others improve efficiency. In this section, the training and runtime performances of each of the convolution and specially developed operations in Figure 3.1 are examined and compared by employing them in the well-known CNN architecture ResNet [22], which is a block-based network easily enabling adaptations of different approaches on it.

As explained in section b), the spatial 2D convolution in Figure 3.1 (a) is frequently used especially in heavyweight models [21], [37], [178]. The rest of the convolution types in Figure 3.1 are proposed to enhance the performance of the spatial convolutions. For example, the grouped convolution [21] in Figure 3.1 (d) brings three essential benefits, which are: enabling of model-parallelization; more structured learning with unique representations of data; and computational efficiency by reducing multiply-add operation number. Furthermore, the shuffle operation [26] in Figure 3.1 (e) allows improving the accuracy of the grouped convolution kernels. On the other hand, the depth-wise separable convolutions in Figure 3.1 (b) and (c), which are used in designing lightweight architectures [147], [201] reduce the parameter and FLOPs numbers. However, despite their superiority in terms of lowering the computational cost, their fragmented memory footprints prevent efficient implementations in practice as shown by equations (3-4) and (3-5). To overcome that constraint, the shift operation [12] in Figure 3.1 (f) presents an alternative by cooperating with the point-wise convolutions in Figure 3.1 (c) and aggregating the spatial information for free.

3.5.1 Outline of Experiment

The approach is to train separately different convolutions and operations using the same reference architecture (ResNet-50 [22]). The six of them in Figure 3.1 are placed in the bottleneck module of the ResNet-50 architecture in Figure 3.4 for a fair comparison. As the model in Figure 3.4 is originally designed with spatial 2D convolutions, it forms the first experimental CNN with the required dimension adjustments arising from the used

dataset, and four more designs are constructed using the remaining convolution and operation types in Figure 3.1. While the second is shaped with two-grouped convolution kernels, the third design is created by three-grouped shuffle operations. As an exception in the fourth, 3x3 and 1x1 spatial convolutions in the original model are replaced by depth-wise and point-wise convolutional filters, respectively as they are usually deployed together [147], [201]. Regarding the final design, the 3x3 spatial convolutions are removed and shift operations are placed to form the fifth model. Then, the designed models are trained and tested, and then their accuracy and inference time on the CIFAR-10 and CIFAR-100 [23] datasets are analysed. Moreover, by decomposing the architectures into basic components, they are further analysed to find their individual execution times on compute-bound (CPU) and memory-bound (GPU) computation platforms during inference.

In the training, NVIDIA GPUs Tesla P100s, part of the ALICE High-Performance Computing Facility at the University of Leicester, are employed according to five different model requirements with a mini-batch size of 128 and a base learning rate of 0.1. The weight decay and momentum are 0.0001 and 0.9, respectively. The training for each architecture stops after 32k iterations. In addition, the learning rate decays with a factor of 10 after 16k and 24k iterations. The inference time of the trained models is tested on both a CPU (Intel Core i5-8500) and a GPU (Tesla P100) by decomposing it into proportional slices to the time taken by the basic components of the architecture.

3.5.2 Evaluation Results

The individual performances of the evaluated convolution types are presented in Table 3.2 and Figure 3.6.

As it could be seen from Table 3.2, different convolutions perform differently when deployed in the same CNN model. The grouped convolution and shuffle operation reduced the parameter and computational overheads compared to the spatial 2D kernels as well as enabled a better accuracy, which confirms the findings presented in the Alex Net 3.2.1 and ShuffleNet 3.3.12 sections above. Notably, as shown in Table 3.2, there is a sharp drop in the parameter and FLOPs number when the depth-wise separable and shift operations are being employed. Nonetheless, they achieve a similar accuracy performance compared to the others.

It should be mentioned that the accuracy levels of the evaluated models in Table 3.2 are lower than those reported in [22], [12], [26]. One reason for that is the smaller

number of iterations in this experiment, as the aim here is only to assess the relative performance of the different convolution types.

Table 3.2: CIFAR-10&100 training of ResNet-50 with different convolution types.

Convolution Types	Parameters ($\times 10^6$) / FLOPs ($\times 10^6$)	CIFAR-10 Accuracy (%)	CIFAR-100 Accuracy (%)	Inference (ms)
Spatial	22.7/3800	79.32	62.12	79
Grouped	16.1/1970	81.40	61.02	73
Shuffle	10.3/1330	83.13	62.03	61
Depth-wise	4.5/878	79.42	60.70	53
Shift	1.4/318	78.13	59.87	37

It can also be seen from Table 3.2 that the accuracy rates on the CIFAR-10 dataset are higher compared to CIFAR-100 for all convolution types, which is due to the difference in the class number, as detailed in section 2.1.5.

Figure 3.6 represents a summary of the timing analysis of the ResNet-50 [22] model during the inference stage for the evaluated convolutions and operations in Figure 3.1. For a clear demonstration, element-wise operations, such as ReLU, batch normalization, tensor addition, and concatenation, etc., are incorporated into a single word (El-wise.). Also, the pre-processing time for the data feeding is not represented here. Results are obtained under PyTorch and averaged from 30 runs. Figure 3.6 (a) shows evaluation results for the ResNet-50 model with spatial filters; (b) and (c) - with grouped filters and shuffle operation, respectively; while (d) includes depth-wise and point-wise convolutions; and (e) uses shift operations. The convolution time (Conv.) is depicted by blue colour for the CPU-based executions, and by orange colour for the GPU-based executions.

First of all, it could be seen from Figure 3.6 that the element-wise operation (El-wise.) in each ResNet-50 model takes more time in the GPU compared to the CPU during inference as they are more prone to compute-bound (CPU) computation platforms. The reason behind that is their memory access cost is heavy even if they have small FLOPs. As an expected outcome, the slices of the convolution (Conv.) operations in Figure 3.6 (b) and (c) are lower compared to (a) as grouped convolutions reduce computational overhead. It can also be observed from Figure 3.6 (c) that the slice of the shuffling operation (Shuffle) equals 12% of the runtime of the GPU, which means that its parallel hardware is limited by the shuffling operation. Nevertheless, the accuracy level and efficiency of the shuffling operation outperform spatial convolutions as per the results in Table 3.2.

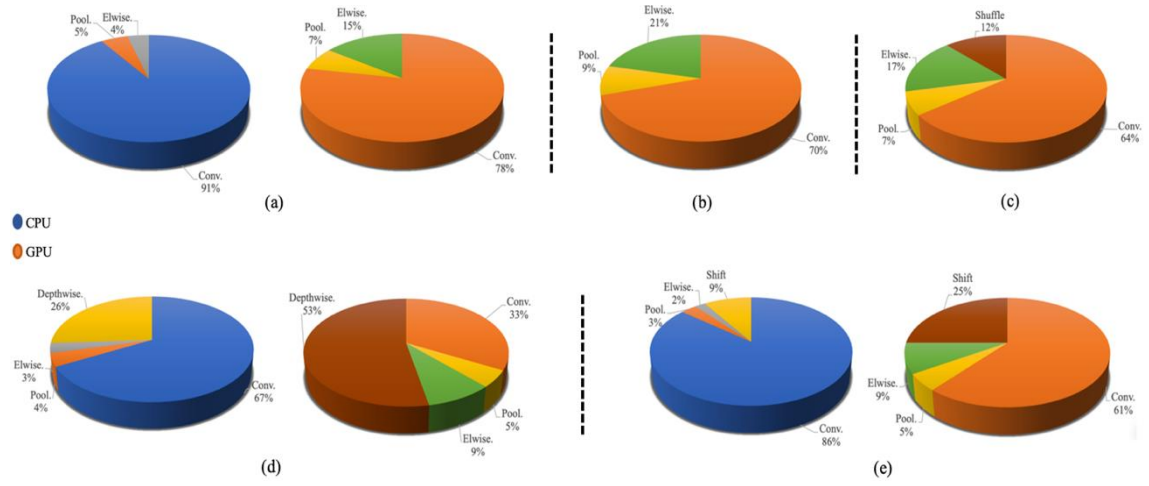


Figure 3.6: Graphical representation of the runtime timing analysis of the ResNet-50 model for the convolutions and operations in Figure 3.1. (a) ResNet-50 model with spatial filters; (b) and (c) - ResNet-50 model with grouped filters and shuffle operation, respectively; (d) ResNet-50 model with depth-wise and point-wise convolutions; (w) ResNet-50 model with shift operations.

The pie charts in Figure 3.6 (d), verify equations (3-4) and (3-5). Even though depth-wise convolution filters require fewer parameters and lower computational overhead in theory, unlike spatial convolutions, their memory access demand dominates computations and limits the performance of memory-bound computation platforms such as GPUs. As it can be seen in Figure 3.6 (d), the depth-wise convolution operations occupy a substantial part of the runtime in GPU amounting to 53%. This drawback also indicates higher energy consumption per floating-point operation, thereby leading to an inefficient computation. Lastly, Figure 3.6 (e) proves that the shift operation takes a step further compared to the depth-wise separable convolution in terms of performance. Despite its considerable runtime, caused by the large number of memory accesses instigating bottlenecks, the shift operation requires a lower number of parameters and FLOPs and occupies a smaller slice of the GPU runtime compared to the dept-wise convolution in Figure 3.6 (d), i.e. 25% versus 53%.

In conclusion, based on a fair comparison, Table 3.2 and Figure 3.6 practically confirm the introduced theoretical findings for the popular convolution types and reveal invisible timing aspects related to their runtime execution. Thus, this analysis inspires the novel CNN designs to be able to propose computation efficient models.

3.6 Summary

In this chapter, several contributions have been provided as follows. The state-of-the-art CNN models for image classification task have been investigated by publication dates from the line of vision: heavyweight or lightweight. Thus, current limitations on the recent work have been revealed to direct this research in the process of a novel CNN design. In addition, the development stages of the proposed convolution and operation types have been also analysed, and their mutual plus and cons have been revealed with equations and experiments in an evolutionary manner. For this, a novel approach has been developed decomposing the designed architectures into basic components, which are analysed to find their individual execution times on compute-bound (CPU) and memory-bound (GPU) computation platforms during inference. The experiments provide valuable insight while designing novel efficient architectures. Next, the following chapter investigates to determine the current limitations of object detection CNNs as done for image classifier CNNs in this chapter.

CHAPTER 4 CNN MODELS FOR OBJECT DETECTION

Object detection has been a long-standing theme in computer vision research. With regards to self-driving vehicles, object detection is an essential part of modules in the ADSs pipeline, such as scene understanding and object tracking. The aim is to identify the locations and sizes of the objects present in images taken by cameras at the viewing angle of the car. These could be both static objects, e.g., traffic lights, road signs, etc., and dynamic objects, e.g., vehicles, pedestrians, etc.

This chapter aims to probe the recent CNN-based object detection models, which frequently place an image classification network as a backbone network for feature extraction and these extractors are notably determinative of the performance of the detectors. As detectors are directly deployed in the ADSs pipeline, there is a strong correlative connection among classifiers, detectors, and ADSs. The objective of this chapter is to investigate the main streams of the recent object detection work and to produce meaningful data with a table demonstration. To bridge academic research and industry practice on CNNs and ADSs, this chapter also investigates the most recently published ADS-related works that are focused on CNN-based image classification and object detection. These are then used to determine the expectations of the ADSs from a CNN-based detector.

This chapter is structured as follows. Sections 4.1, 4.2, and 4.3 introduce traditional detectors, what a backbone network is and for what, and typical baselines, which specify the design kind and building blocks of detectors, respectively. Following that, sections 4.4 and 4.5 investigate representative state-of-the-art models for object detection in detail, dividing them into two categories: one-stage detectors and two-stage detector. By means of the produced content from sections 4.4 and 4.5, the main streams in the field are revealed by means of a table demonstration in section 4.6. Then, in section 4.7, the most recent ADS-related object classification and detection work is also investigated. Lastly, section 4.8 concludes the chapter by summarising the importance.

4.1 Traditional Detectors

Looking back over the last two decades of detection methods, until the early 2010s the limitations on computing resources, datasets, and the mostly theoretical nature of DNN development had led to employing traditional detectors, such as DPM [210], Selective Search [211], Oxford-MKL [212], HOG [213], NLPR-HOGLBP [214], SIFT [215],

VJ Det [216], Bag of Words [217], etc. A modular data-flow diagram and a historical publication timeline of these popular detectors are depicted in Figure 4.1 and Figure 4.3, respectively.



Figure 4.1: Data-flow diagram of traditional detectors.

The purpose of the Region Selector module in Figure 4.1 is to prepare sliding windows of different sizes and to slide them over the image from left to right and top to bottom by keeping a certain step size. Then, cropped image blocks are produced by the sliding windows and converted to images with uniform dimensions. In the Feature Extractor module, features are extracted from the images by the way of deploying different algorithms such as HOG [213], SIFT [215], etc. The last module in Figure 4.1 is a Classifier, which is used to identify the category of objects, extracted in the previous step via different algorithms like SVM [218] and Adaboost [219]. Despite their popularity, the traditional detectors were relatively mature and had multiple drawbacks, such as high window redundancy, computing complexity, etc.

Following the emergence of high-performance computing devices and inclusive datasets as well as the staggering performance of AlexNet [21], in the last decade, the attention in the field gradually turned to CNN-based solutions, as illustrated in Figure 4.3.

4.2 Backbone Networks

As briefly stated above, object detection models deploy a classification algorithm as a backbone or a base network that acts as a basic feature extractor. The CNN models for image classification, introduced in Chapter 3, such as ResNet [22], ShuffleNet [26], MobileNets [147], SqueezeNet [149], Xception [201], can be directly adopted or improved with new features to be used as a backbone for object detection tasks. In several publications [105], [220], [221], specific requirements have been stated for existing classification algorithms to perform better in a detection pipeline. Additionally, it is recognised that even if detection speed is a key factor, e.g., in real-time applications, high precision and accuracy hold at least an equal importance as well. That means that there is a need for a good trade-off between speed and accuracy [171] when selecting backbone

networks for detectors. The recently published high-performance CNN architectures for classification may help to solve the problem, as quality feature extraction raises up the whole detection performance. For example, He et al. [106] achieved a remarkable performance following this approach. Further details are included in the descriptions of the individual detection algorithms in sections 4.4 and 4.5 below.

4.3 Typical Baselines in CNN-based Detector Design

In CNN-based detector design, there are two commonly used baseline schemes: one-stage and two-stage detection pipelines. Among the two-stage detectors, the Region-based Convolutional Neural Network (R-CNN) series [148], [222], [223] are the most prevalent ones, whereas YOLO [105] and SSD [224] are the CNNs of choice for the one-stage detectors.

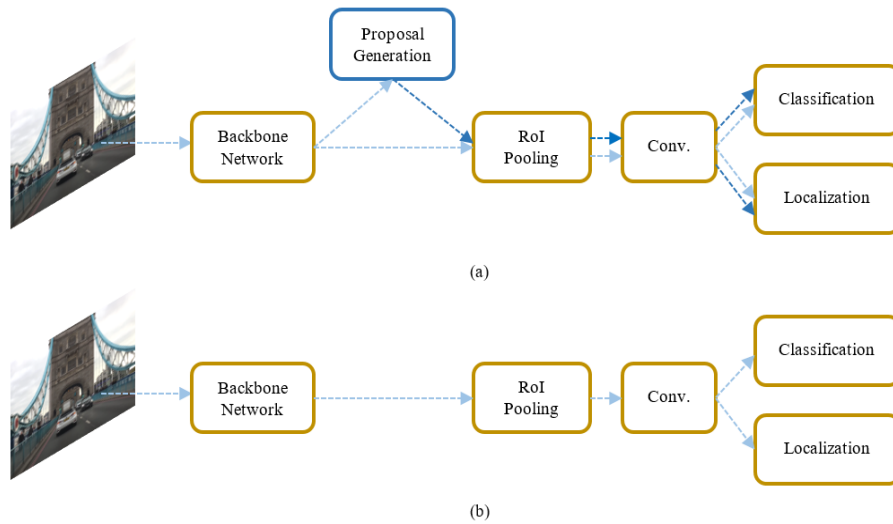


Figure 4.2: Functional diagrams of typical detector pipelines: (a) two-stage detector; (b) one-stage detector.

The functional diagrams of the two baselines are illustrated in Figure 4.2. The two-stage detector pipeline is shown in Figure 4.2 (a), where region proposal network block (Proposal Generation) feeds region proposals into the classifier and localization modules. These detectors differ from the one-stage detectors, depicted in Figure 4.2 (b), in the operation of the RoI (Region of Interest) pooling layer. Unlike the two-stage detectors, where the prediction of the bounding-boxes is carried out by the region proposal stage (Proposal Generation), in the one-stage detectors, it is implemented directly from the input images, which allows them to perform faster. This is why one-stage detectors

are more preferred in real-time applications despite their low accuracy performance compared to the two-stage ones. Weighing the merits of the two schemes, it could be said that the accuracy of object recognition and localization is higher in two-stage detectors, whereas the inference speed is better in one-stage detectors [11].

Figure 4.3 displays the entire timeline of both the traditional object detectors and the CNN-based designs of the last decade. It could be seen from Figure 4.3 that the two-stage CNN-based detectors preceded the development of the one-stage detectors. Therefore, in the following investigations of the existing object detectors with CNNs, first, the two-stage designs are introduced in section 4.4, and then the one-stage detectors - in section 4.5.

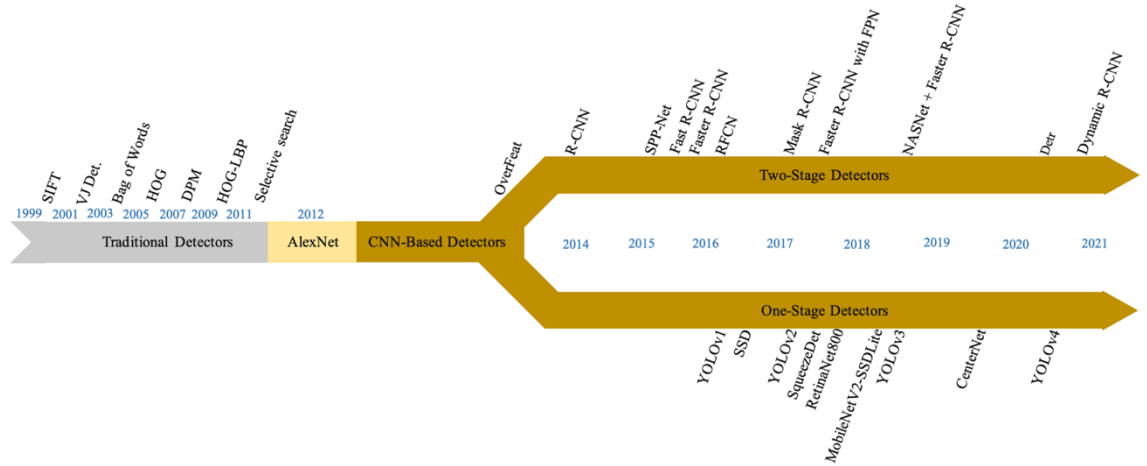


Figure 4.3: Timeline of traditional and CNN-based object detectors.

4.4 Two-stage CNN Detectors

4.4.1 R-CNN

In 2014, Girchick et al. [222] introduced a detection algorithm called Region-based Convolutional Neural Network (R-CNN), which featured a high object detection performance. Figure 4.4 illustrates the architectures of the three best-known R-CNN models: R-CNN [222], Fast R-CNN [223] and Faster R-CNN [148] during both the testing and the training phase. The diagrams related to the testing phase are depicted using only amber colour. The diagrams of the training phase are depicted using both amber and purple colour for the Fast R-CNN and the Faster R-CNNs in Figure 4.4 (b) and Figure 4.4 (c), respectively, which include multi-task loss functions.

As shown in Figure 4.4 (a), the R-CNN method starts by identifying category-independent $\sim 2k$ region proposals containing potential objects, and then, proceeds each

region to the backbone network AlexNet [21] to extract 4096-dimensional feature representations. Lastly, while an SVM is deployed for classification, fine adjustments of the bounding boxes are provided by a Bounding-Box regression and a greedy non-maximum suppression (NMS) method [225]. Using such a design, R-CNN achieved an improved mean average precision (mAP) of 58.5% on the Pascal VOC dataset [44]. Despite its accuracy performance, R-CNN had a few design problems. For instance, it was slow in processing the images, taking a second per image or 1 frame per second (fps), and required many GPU days to be trained, as $\sim 2k$ region proposals per image had to be classified in the network. The algorithm, therefore, could not be utilised in real-time. In order to solve these drawbacks, novel versions of R-CNN were proposed afterwards.

4.4.2 SPP-Net

He et al. [226] proposed a novel algorithm, SPP-Net, with a strategy, which is referred to as spatial pyramid pooling (SPP) eliminating the singly passing of $\sim 2k$ region proposals to the backbone network in R-CNN. SPP-Net firstly computes a convolutional feature map of the whole input image and following that, classifies each object proposal by taking advantage of the spatial pyramid pooling layer. Thus, SPP-Net accelerated the testing time of R-CNN by between 10 and 100 times. Another superior side of SPP-Net was that it could be used for increasing the performance of all CNN-based tasks, like image classification. ZF-Net [177] was used as a backbone to measure the mAP performance of SPP-Net on Pascal VOC [44] dataset and it achieved 60.9%, which was better than the R-CNN performance with a lower computational load. In spite of its advantages, SPP-Net also had several drawbacks of [222]. The training was a multi-stage pipeline including extracting features from the input image, which required fine-tuning of the network with log loss, using of SVMs, and providing bounding boxes for the regression stage. All those signalled new approaches in the field.

4.4.3 Fast R-CNN

In an attempt to resolve the problems of R-CNN [222] and SPP-Net [226], outlined in sections 4.4.1 and 4.4.2 above, Girshick [223], from Microsoft Research, proposed a novel algorithm called Fast R-CNN at the ICCV 2015 conference. By introducing several innovations in the Fast R-CNN design, the training and testing speed of R-CNN was improved, and its detection accuracy was increased as well. A novel RoI pooling layer was fitted to the network as the single-level spatial pyramid pooling layer in SPP-Net.

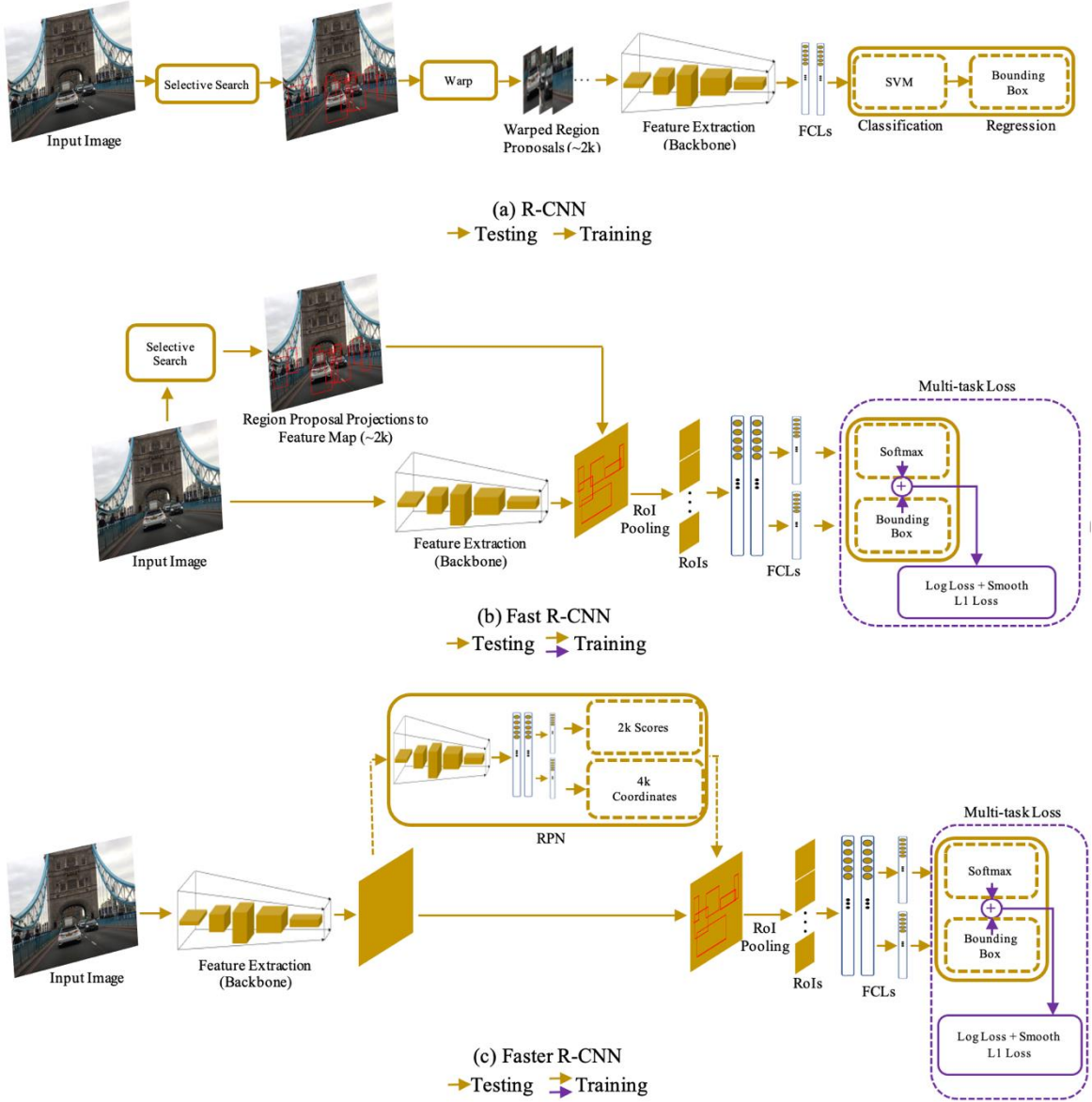


Figure 4.4: Architectural demonstrations of the three best-known R-CNN models R-CNN [222], Fast R-CNN [223], and Faster R-CNN [148].

Figure 4.4 (b) represents the architecture, where region proposals are first created by a Selective Search algorithm [211] and then, mapped onto the feature maps. Following that, the RoI Pooling layer prepares different feature regions as fixed-size feature vectors, which are then used as inputs in FCLs. Finally, whereas object categories are predicted by the softmax operation [227] with log loss [228] in contrast to R-CNN employing SVM [218], object locations are located by a bounding-box regression with a smooth L1 (absolute) loss [229]. Thus, its performance on the Pascal VOC 2007, 2010, and 2012 datasets on training with the backbone network VGG-16 [178] was measured as 70.0%,

68.8%, and 68.4% mAP, respectively. Fast R-CNN was trained on the VGG-16 base network 9x faster than R-CNN and was 213x faster during testing. As for SPP-Net, Fast R-CNN was 3x faster in training, 10x faster in testing, and more accurate when training on VGG-16. Besides improving training, testing, and accuracy performance, there were three other contributions which are: end-to-end training, no need for disk storage, and updates on all network layers.

4.4.4 Faster R-CNN

In R-CNN [222] and Fast R-CNN [223], region proposals were being generated by different blocks, rather than using only one convolutional neural network. That was time-consuming, i.e., ~ 2.3 s for making predictions and ~ 2 s for generation of 2k RoIs, representing a bottleneck in system performance. To solve the drawback, Ren et al. [148] proposed a novel Region Proposal Network (RPN), shown in Figure 4.4 (c), which was constructed by using CNN-based layers, where the region proposals are generated right after the backbone network. Unlike its ancestors, the number of RoIs is not a constant value and is defined by the size of the feature map. Thus, the region proposals were implemented on GPUs with nearly free of computation cost compared to previous baselines. The performance results of the algorithm are detailed in Table 4.1.

4.4.5 RFCN

Dai et al. [230] addressed the basic shortcoming of Faster R-CNN [148], which was related to the RoI layer being located between the backbone network and the object detection modules. Such a design caused a lack of translational invariance in the detector as the RoI was converting the multi-dimensional feature map to fixed size FCLs. To overcome the problem, they proposed position-sensitive score maps in the design of RFCN, based on a fully convolutional network. Thus, all learnable parameters were convolutional and sharable in the kernels, which was increasing the translational invariance. It achieved 80.5% mAP on the Pascal VOC [44] dataset with a ResNet101 [22] backbone network. Further details are represented in Table 4.1.

4.4.6 Mask R-CNN

Mask R-CNN [106] is a result of extending Faster R-CNN [148]. It has been primarily designed as a framework to perform a segmentation task in addition to object detection. The Mask network as a modernised version of FCNs [39] is placed in parallel to the detection pipeline for the generation of split masks for each RoI. Due to an alignment problem between the feature map and the original image in the RoI pooling layer, which

uses integer quantization, a RoIAlign layer based on a bilinear interpolation method was employed. This allows preserving the alignment at pixel-level in the exact spatial locations between the feature map and input image. It also enables to achieve a better detection accuracy. Thus, by deploying ResNeXt-101 [181] as a base network, a rise in performance was achieved compared to the previous detectors, as shown in Table 4.1.

4.4.7 FPN

In 2017, Lin et al. [220] proposed the Feature Pyramid Network (FPN) that fuses features and enhances the system detection performance. The algorithms, proposed until then, were detecting either a top-level feature or were performing an independent detection in feature layers. This disallows to combine classification and location information. However, FPN introduced a way of laterally connecting layers top-down and bottom-up, evoking a pyramidal operation. FPN was incorporated in Faster R-CNN [148] and achieved state-of-art results on the MS COCO dataset [27], as detailed in Table 4.1.

4.4.8 NASNet

NASNet [180] was not solely designed for the purpose of image classification. It was also targeted at object detection tasks, as the learned features from the classification process can be used in detectors. Thus, NASNet used the Faster R-CNN [148] framework for detection and obtained a remarkable accuracy of 43.1% mAP compared to previous baselines, such as MobileNetV2 [202] with 22.1% mAP on MS COCO [27] dataset.

4.4.9 Detr

Carion et al. [144] from the Facebook AI team developed a new CNN model, called DETR, which was designed as a solution to the direct set prediction problem for object detection. The model does not include anchor generation [231] and non-maximum suppression [225] as they prevent the use of prior information in the detection pipeline.

There are two main contributions. The first one is a set-based global loss, which enables powerful predictions through partite matching and the other one is a transformer encoder-decoder architecture. Via these strategies, the model has improved a bit further the accuracy rate of the two-stage detectors to 44.9% mAP with 10 fps in the COCO dataset [27].

4.4.10 Dynamic R-CNN

In 2020, Zhang et al. [232] published an advanced R-CNN algorithm. Their idea was based on addressing the inconsistencies between the dynamic training process and

the fixed network adjustments, which significantly affect performance. To tackle the problem, Dynamic R-CNN provides an automatic and dynamic adjustment on both the shape of the regression loss function, e.g., the parameters of Smooth L1 Loss [229], and the label assignment criteria, e.g., the IoU threshold [233], by using the region proposal statistics in training. Moreover, no extra cost has been added to the model. Thus, the model performance was improved considerably achieving 49.2% mAP on the COCO dataset.

4.4.11 Dynamic Sparse R-CNN

Following a recent powerful object detection baseline, Sparse R-CNN [234], Dynamic Sparse R-CNN [146] has been proposed to improve the former with two dynamic design methods. First, instead of a one-label assignment scheme, the Hungarian algorithm [235], a dynamic label assignment (DLA) is introduced to increase positive samples in the training. Second, a drawback of Sparse R-CNN in the inference process is that the learned features and proposal boxes for different samples remain fixed. Based on dynamic convolutions [236], dynamic proposal generation (DPG) is also introduced to dynamically derive better and multiple features and proposal boxes. Thereby, experiments have shown that Dynamic Sparse R-CNN with ResNet-50 [22] backbone network surpasses the object detection performance of Sparse R-CNN by 2.2% mAP (47.2% mAP) on the COCO dataset [27]. Dynamic Sparse R-CNN with ResNet-101 [22] backbone reaches the state-of-the-art for two-stage object detectors with 47.8% mAP on the COCO dataset (see Table 4.1).

4.5 One-stage CNN Detectors

4.5.1 YOLO

Even though the proposed RPN in Faster-RCNN [148] reduces the region proposal number and the overlaps between them, the repetitive computations due to inevitable overlaps still cause an essential bottleneck in the performance of these detectors. So as to deal with the problem, Redmon et al. [105] introduced a novel hybrid CNN-based architecture, named YOLO (You Only Look Once), which is illustrated in Figure 4.5 (a). It combines the region proposals and detection branches into one single stage in contrast to R-CNN [222], Fast R-CNN [223] and Faster R-CNN [148], as seen in Figure 4.5 (a), where the classification and bounding boxes regression are responsible to detect the object centres in the grid cells.

YOLO detectors first divide the input images into $S \times S$ grids and then, each grid is responsible to predict C class probabilities, B bounding boxes, and the confidence value for those boxes. Thus, the input is encoded as a tensor with $S \times S \times (5B + C)$ dimension. The end-to-end architecture is built by 24 convolutional and 2 FCLs. As a result, YOLOv1 has achieved a high processing performance of 45 fps, which satisfies real-time implementation requirements. Further details are presented in Table 4.1.

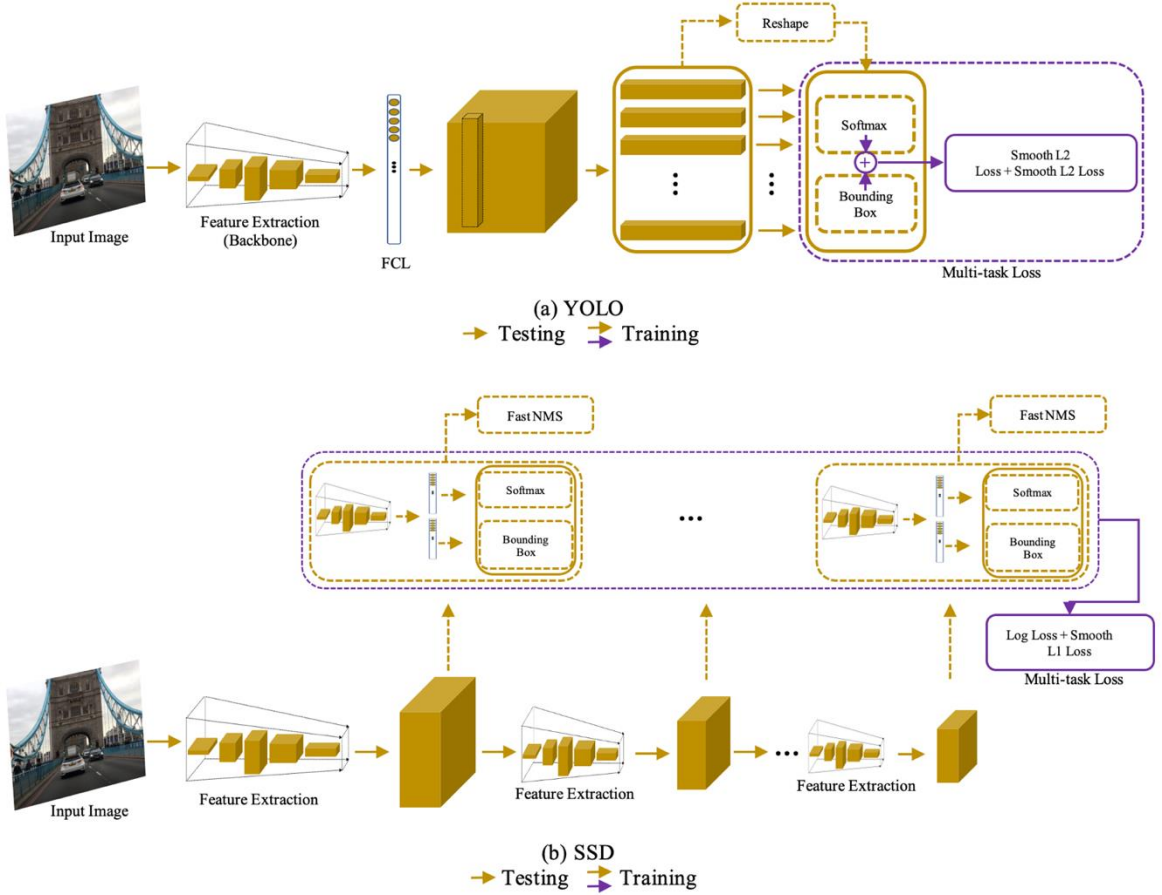


Figure 4.5: Illustrations of the single-stage pipelines of the YOLO [105] and SSD [224] architectures.

Despite its success, the YOLO model has also exhibited a few limitations. As a major problem, its detection ability on dense small objects is inefficient, since each grid predicts two bounding boxes from the same class at most. Another shortcoming is its generalization ability in case of different aspect ratios of objects on different images during testing. Lastly, the loss function is in a need of improvement, as it affects the detection score.

4.5.2 SSD

As discussed in sections 4.4 and 4.5.1 above, the R-CNN series [148], [222], [223] and YOLO [105] come with their superior and inferior features, trading-off between accuracy and speed. Liu et al. [224] proposed the Single Shot MultiBox Detector (SSD), which has advantages on both sides. In the SSD design, VGG-16 [178] is used as the backbone network, in which the fully connected sixth and seventh layers are replaced with convolutional layers as well as four more convolutional layers are placed at the end.

The entire design, which comprises six-stages, is hierarchical, and represents a single forward pass network. The reason for such a design is to provide hierarchical extraction of features, where each hierarchical layer provides object classification and bounding-box detection with different semantic information levels, as it could be seen from Figure 4.5 (b). Moreover, each stage applies a fast non-maximum suppression (NMS) technique [225], aimed at post-processing of redundant bounding boxes. The purpose is to eliminate overlaps in bounding boxes at every stage, and it also leads to reducing the amount of computation without compromising the accuracy. Thus, SSD512 outperforms Faster R-CNN on both accuracy and speed, and SSD300 shows a processing performance of 59 fps, which is higher than that of YOLO.

4.5.3 SqueezeDet

In 2017, another CNN model was proposed by the creators of SqueezeNet [149], named SqueezeDet [237], which is a fully convolutional neural network designed for object detection. In fact, it received inspiration from both YOLO [105] and SqueezeNet. SqueezeDet is aimed at satisfying real-time constraints of embedded deployment in terms of model size, energy efficiency, as well as a high level of accuracy.

Due to its single forward pass neural network pipeline, SqueezeDet architecture is effectively fast, small model sized, accurate, and energy efficient. By keeping the same accuracy as previous baselines, such as YOLO, it enables 8MB model size, which is 30x smaller compared to Fast RCNN + AlexNet [223]. The energy consumption of 1.4J per fps in NVIDIA Titan X GPU is 35x lower compared to [223], and, the inference processing speed of 57 fps, is 20x faster compared to [223]. It also requires fewer DRAM (Dynamic Random-Access Memory) accesses and enables the best average precision in all three difficulty levels of cyclist detection of the KITTI dataset [28], which is introduced in section 2.1.5.

4.5.4 YOLOV2 (YOLO9000)

As stated in section 4.5.1 above, YOLO has several limitations, causing lower localization and recall performances. To address the problems, YOLOv2 [238] model introduces a number of improvements on YOLO. First, it enhances the generalization capability by means of batch normalization [239], which speeds up the optimization process. The anchor idea [148], aimed at increasing the generalization capability in different aspect ratios, is also incorporated in YOLOV2, and thus, more scale and aspect ratio can be predicted by each grid cell. The second improvement is that it trains high-resolution classifiers to locate images with higher resolutions. Third, in order to increase the detection ability, K-means clustering algorithm [240] is deployed to automatically find the bounding boxes. Lastly, it handles the instability of the YOLO model [105] by limiting the ground truth offset in regard to the grid coordinates. By upgrading YOLO with these advancements and designing a novel base network, Darknet-19, based on VGG-16 [178], YOLOV2 has achieved a 78.6% mAP on the Pascal VOC [44] and a higher number of learned object categories.

4.5.5 MobileNetV2-SSDLite

MobileNetV2-SSDLite is based on the classification CNN MobileNetV2 [202], which was discussed in section 3.3.11 above. It includes an architecture for object detection using a modified version of the SSD (Single Shot Detector) model [224] that is a mobile variant of SSD, whereby the main idea comes from the separation of regular convolutions. All the spatial 2D convolutions in the SSD prediction layers are replaced with separable convolutions (i.e., depth-wise convolutions, followed by point-wise convolutions). As a result, its performance was even better than that of YOLOv2 [238] with 10x fewer parameter number and 20x less computational load.

4.5.6 YOLOv3

YOLOv3 [241] inherits the superior sides of the previous YOLOv1 and YOLOv2 models and handles their shortcoming balancing accuracy and speed. To actualize the aim, YOLOv3 combines residual networks [22], FPN [220], and binary cross-entropy/log loss [228]. As a result, the model can detect complex, multi-size, and multiple-category objects, achieving the performance of 33.0% mAP on the COCO dataset [27].

4.5.7 CenterNet

In 2019, Duan et al. [242] proposed an algorithm, CenterNet, whose motivation comes from the realisation that faults in the detection of bounding boxes are the main

problems in architectures, and if the cropped regions are looked into again, the accuracy performances could level up. CenterNet, therefore, focused on minimising that kind of errors. They built their framework being inspired by CornerNet [243], which is a typical one-stage key-point based detector. Unlike a pair of key-points in CornerNet, each object in CenterNet is detected as a triplet of key-points, which allows to improve precision and recall accuracy.

As seen from Figure 4.6, a convolutional backbone network is used to define a two-corner feature map and centre key-point features on the map by employing cascade corner and centre pooling modules, respectively. By using these feature maps, CenterNet detects the final bounding boxes. It outperformed state-of-the-art detectors, such as YOLO [105], Faster R-CNN [222], SSD [224], and CornerNet [243], on the COCO dataset [27], achieving an accuracy performance of 47.0% mAP.

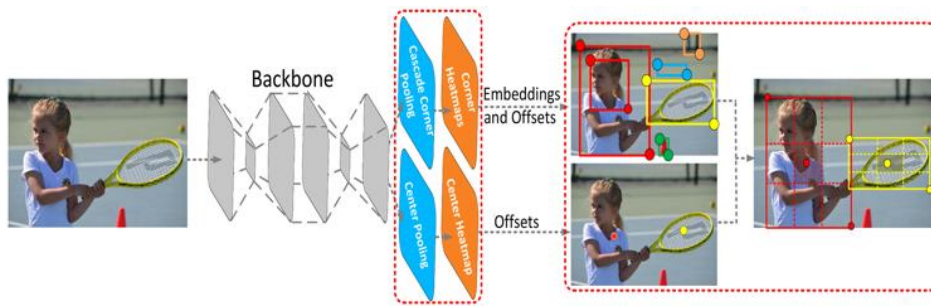


Figure 4.6: CenterNet network [242].

4.5.8 YOLOv4

Bochkovskiy et al. [244] introduced YOLOv4 at CVPR 2020. It incorporates several existing universal and unique strategies, such as DropBlock regularization, CIOU loss, Mosaic data augmentation, Mish activation, etc., into the architecture. CSPDarknet53 [245] is used as the backbone network as well as SPP [226] and PAN [246] are employed as neck layers. These are transition layers between the backbone and head blocks (classification and bounding boxes), which are inserted to improve detection accuracy. Consequently, its real-time processing speed and accuracy are quite good with 43.5% mAP at 65 fps, which is better than the previous YOLO version, YOLOv3 [241].

4.5.9 YOLOv5

In contrast to the other YOLO series, YOLOv5 [247] has not been published as a scientific paper and released by Ultralytics. It is the first version of YOLO implemented on PyTorch [70]. Similarly to YOLOv4 [244], CSPDarknet53 [245] is employed as the

backbone network. The major improvement of YOLOv5 is the integration of a *focus layer*, which is represented by a single layer and created by replacing the first three layers of YOLOv3 [241]. This novel layer reduces the number of layers and parameters as well as increases both forward and backward propagation speed without allowing any major impact on the mAP. As a result, YOLOv5 outperforms the previous version with 49% mAP at 113fps.

4.5.10 YOLOv6

YOLOv6 [248], written in PyTorch [70], is primarily dedicated to industrial applications with efficient, high-performance, and hardware-friendly design. YOLOv6 has introduced three significant advancements over the YOLOv5, which are: (i) hardware-friendly backbone, RepBlock for small models or CSPStackRep Block for large models, and hardware-friendly neck layers, Rep-PAN; (ii) an efficient decoupled head (classification and localization branches of object detection network); (iii) a more effective training strategy. By means of these characteristics, YOLOv6 has outperformed the previous versions with outstanding results, especially with 179 fps performance (see Table 4.1).

4.5.11 YOLOv7

The next version, YOLOv7 [249], has made two major improvements. First. On the architectural basis, the architecture of YOLOv7 is reformed by integrating the Extended Efficient Layer Aggregation Network (E-ELAN), which enables the model for better learning on more diverse features. To meet the needs of various inference speeds, YOLOv7 architecture is also built on a compound model scaling basis, which allows concatenating different application requirements by keeping the key attributes. In this way, YOLOv7 can scale its width, depth, and image resolutions. Then in the second, trainable bag-of-freebies, some of the tricks, are used to improve model accuracy without causing any training cost. In consequence, this version of YOLO surpasses all the previous versions in accuracy with 51.2% as shown in Table 4.1.

4.5.12 YOLOv8

As the latest version of YOLO series, in 2023, the release of YOLOv8 as done for YOLOv5 in 2021 has been confirmed by Ultralytics [145]. The company has not published any scientific paper yet. This new version promises new features and performance advancements over its predecessors. YOLO v8 comes with a new API making training and inference much easier on both compute-bound (CPU) and memory-bound (GPU) de-

vices and supporting previous YOLO versions. This makes it an ideal choice to take advantage of the latest technology while still being able to access previous YOLOs too. The outstanding innovations are: (i) a novel backbone network; (ii) a novel anchor-free detection head; (iii) a novel loss function. Thus, the latest version achieves the state-of-the-art performance with 53.9% mAP at 253 fps.

4.6 Analysis of CNN-based Object Detectors

The performance of the investigated leading CNN-based detectors is summarized in Table 4.1, in which the names of the two-stage models are shaded in light grey colour, while the names of the one-stage models – in dark grey colour. For each model, the following data are provided: literature source, backbone network, year of introduction, model size in MB, processing speed in frames per second (fps), implementation platform, and mean average precision (mAP) on the Pascal VOC [44] and MS COCO [27] pascal datasets.

It can be observed from Table 4.1 that the models that have performed best in terms of accuracy in their baseline class are Detr [144], Dynamic R-CNN [232], and Dynamic Sparse R-CNN [146] for two-stage detectors and mainly YOLOv7 [249] and YOLOv8 [145] series for one-stage detectors. It can also be seen from Table 4.1 that the model sizes of these detectors are quite high among models. Thus, there is a similar trend that heavyweight detectors have provided higher accuracies on the test datasets like the image classification models, discussed in Chapter 3.

As for the trend on low model size detectors, the winners are SqueezeDet [237] and MobileNetV2-SSDLite [202] having the outstandingly small size of around 8MB and 18MB, respectively, according to the literature. However, their accuracy levels are remarkably low for any real-time implementation, compared to other state-of-the-art detectors. So, lightweight CNN-based object detection solutions have shown the same tendency as classification CNNs.

It is not surprising that the evolution of the object detection CNN-based models and the CNNs for image classification have shown similar tendencies. This is because they have a close historical and architectural relationship.

Table 4.1: Performances of CNN-based object detectors.

<i>Model</i>	Backbone Network	Year	~Model Size (MB)	fps	Platform	mAP	mAP
						Pascal VOC	MS COCO
DPM [210]	-	2010	37	2.0	Intel Xeon	33.7%	-
R-CNN [222]	AlexNet	2014	-	-	Titan	58.5%	-
SPP-Net [226]	ZFNet	2015	143	2.6	Titan	60.9%	-
Fast R-CNN [223]	VGG-16	2015	240	0.5	Titan X	70.0%	19.7%
Faster R-CNN [148]	VGG-16	2015	490	0.7	Titan X	73.2%	21.9%
RFCN [230]	ResNet101	2016	435	9.0	Titan X	80.5%	29.9%
Mask R-CNN [106]	ResNeXt-101	2017	-	5.1	Tesla M40	-	39.8%
Faster R-CNN with FPN [220]	ResNet-50	2017	137	6.7	Tesla M40	-	33.9%
NASNet + Faster R-CNN [180]	NasNet	2018	-	-	-	-	43.1%
Detr [144]	ResNet-101	2020	720	10.0	Tesla V100	-	44.9%
Dynamic R-CNN [232]	ResNet-101	2020	550	13.9	RTX 2080TI	-	46.9%
Dynamic Sparse R-CNN [146]	ResNet-101	2022	590	14.0	A100	-	47.8%
YOLOv1 [105]	-	2016	-	45	Titan X	66.4%	-
SSD [224]	VGG-16	2016	-	59	Titan X	76.8%	28.8%
YOLOv2 [238]	DarkNet-19	2017	106	40	Titan X	78.6%	21.6%
SqueezeDet [237] ^a	SqueezeNet	2017	8	57.2	Titan X	76.7%	-
RetinaNet800 [243]	ResNet-101	2017	500	13.7	Tesla M40	-	39.1%
MobileNetV2-SSDLite [202]	MobileNetV2	2018	18	5.0	CPU	-	22.1%
YOLOv3 [241]	DarkNet-53	2018	320	45.5	Tesla M40	-	33.0%
CenterNet+ [242]	Hourglass	2019	512	3.7	Tesla P100	-	47.0%
YOLOv4 [244]	CSPDarknet-53	2020	416	65.0	Tesla V100	-	43.5%
YOLOv5 [247]	CSPDarknet-53	2020	367	113	-	-	49.0%
YOLOv6 [248]	CSPStackRep	2022	296	179	Tesla T4	-	49.5%
YOLOv7 [249]	E-ELAN	2022	418	110	Tesla V100	-	51.2%
YOLOv8 [145]	P5	2023	-	283	A100	-	53.9%
Dave-2 System [250]	-	2016	-	30.0	NVIDIA DRIVE PX	-	-

^a Accuracy value was obtained on KITTI dataset [28].

4.7 ADSs Related Works on Object Classification and Detection

This research is ultimately targeted at the perception part of ADSs, which is one of the subsystems of self-driving cars among others. It is responsible for understanding scenes of the driving environment that include objects from multiple different categories. Instead of the traditional single-label image classification, which handles input images as containing one category of an object per image, multi-label image classification [251], which is performed by image classification CNNs (i.e., the introduced backbone networks in Chapter 3), is employed in the object detection module of ADSs. More clearly, by means of anchor boxes [223], each single input image in object detection is subdivided into small images. Then, single-label image classification is performed in each of them.

This process is called multi-label image classification or object classification as an advanced type of single-label image classification.

The investigated image classification and object detection algorithms in Table 3.1 and Table 4.1 are specifically designed for general-purpose datasets, such as ImageNet [40], Pascal VOC [44], MS COCO [27], which can be utilised in different fields, as discussed in section 2.1.5. To enhance the object classification and detection capabilities of ADSs, a diverse set of naturalistic driving scenarios have to be also addressed, e.g., all weather conditions, day/night time, pedestrians, traffic lights, cyclists, traffic density, etc., which are not properly represented in domain-general datasets. Thus, it is necessary to use the domain-specific datasets (exemplified in section 2.1.5).

In this section, recent ADS-related works on object classification and detection are also investigated by analysing them in terms of dataset collection scenarios, sensors, and detection types, which completes the meaningful data collection of this research from every angle in a synergetic way.

4.7.1 Self-Driving Scenarios

Understanding the complexity of naturalistic road scenes is vitally important when there are diverse self-driving scenarios, such as day/night time and weather conditions. In fact, handling diverse driving conditions represents the most challenging research part of ADS design. Hence, enhancements in object classification accuracy of such scenes can be seen as a foremost solution to overcoming the difficulties.

In this direction, Li et al. [252] proposed a deep adaptive neural network for multi-label image classification, ML-ANet, which enhances the accuracy performance in cross-domain adaptations. Aiming at an effective knowledge transfer between similar but different domains, the proposed approach exploits the technique of transfer learning from a fully labelled to a limited or unlabelled domain. In order to achieve this, the domain discrepancies are reduced by distributing feature maps of source and target domains via multiple-kernel variants of maximum mean discrepancies (MK-MMD) loss. The experimental work is primarily focused on domain changes arising from the conversion between clear and hazy weather conditions. It is shown that the proposed multi-label classifier network outperforms existing state-of-the-art methods [253]–[255] on three commonly used domain-specific datasets: KITTI [28], Cityscapes [47], and Foggy Cityscapes [256]. The ML-ANet neural network is also capable of adapting to round the clock illuminations

in diverse weather conditions. In addition, it offers a reduced development cycle, as there is no need of fully labelled training data.

Three models aimed at a detection of pedestrians in hazy weather, which are based on the YOLO CNNs (sections 4.5.1 and 4.5.4), are proposed in [257]. In one of the models, named MNPrioriBoxes-Yolo, the detection precision is increased by employing a novel weighted combination layer. The network features a high speed and low computational cost due to the use of depth-wise separable convolutions and linear bottlenecks. In addition, a modified priori boxes method [148] is employed to enhance precision and detection speed. Evaluation results based on a purpose-built hazy weather pedestrian dataset, which was created using six different augmentation techniques, have shown that the proposed model outperforms state-of-the-art methods in terms of accuracy and speed.

Fog is also an adverse weather condition that can occur in the driving environment. The object detection performance of Faster R-CNN [148] in four levels of foggy weather: clear (no fog), light, medium, heavy is analysed in [258]. Other works could be found in [259]–[261].

A large number of works have been addressing rainy and snowy weather conditions. The performance of the Faster R-CNN [148] and YOLO-v3 [241] detectors under clear and rainy conditions is analysed in [262]. The impact on performance of mitigating the effect of rain is investigated using various techniques, such as image translation [263], domain adaptation [264], and deraining (i.e., raindrop removal) [265]. The BDD100K dataset [46] is used in the experiments, as it offers image tagging for weather types. Thus, each image weather condition, such as foggy, rainy, and so on, is labelled. The evaluation results show that the mitigating techniques have a positive effect on the rainy weather detection performance of the employed CNNs.

Regarding snowy weather, Bernuth et al. [266] apply an adverse weather augmentation approach to reuse existing, well-organised, and labelled datasets, KITTI [28] and Cityscapes [47]. The approach produces lifelike and physically correct images to be added to existing ready-to-use datasets created with real-world images. Rendering the scenes with snowflakes for realistic images is provided by using OpenGL. Then, these generated images are evaluated in the proposed object detection algorithm to verify their efficiency. Other recently published works regarding rainy and snowy weather are [261] and [267].

Driving at night-time under low-light conditions causes a lack of details in the driver’s field of vision and increases accident risks. Thus, many research works have been

targeted at finding possible solutions. Li et al. [268] proposed an image enhancement network, LE-net based on CNN, which specifically addresses the exceedingly low-light conditions at night-time, such as rural areas without street lighting. Then, the study follows four sequential steps: (i) building a novel pipeline to generate low-light images from existing daytime images, extracted from BDD100K [46]; (ii) generation of naturalistic image pairs via the proposed pipeline for model development; (iii) training and validation of the designed network with the generated low-light images; (iv) testing of the network, LE-net, on real night surroundings with variable low-lights. It is shown that on the the BDD100K dataset, the proposed network outperforms other models [269]–[271]. The EuroCity [55] dataset is used as a second evaluation metric, whereby a better performance is also observed in comparison to other studies [270], [271], [272].

To enhance the accuracy performance of object classification tasks at night, a dataset is created in [273] by combining two well-known datasets, MMSP [274] and BDD100K [46]. Following that, the created dataset samples are subclassified into daytime and night-time, and similar samples in the dataset are removed to prevent overfitting. The evaluation results have shown an improved night-time object classification performance. Other works addressing night-time object classification and detection are [275] and [261].

4.7.2 Sensors

The works investigated in the section above mostly use camera-based image data, which are susceptible to level changes in the lighting conditions due to different seasons, intemperate weather, and shifting shadows. Changes in illumination could lead to failure of algorithms, affecting badly the quality of perception. Alternative sensors for perception tasks in ADSs are lidar and radar, however lidar struggles with foggy and snowy weather [276], while radar lacks sufficient resolution for perception tasks [277].

Sensor fusion is currently employed to improve precision and prevent any single point of failure [278]. Furthermore, dynamic conditions in camera-only systems [279] and low lighting conditions in thermal infrared imaging [280] are dealt with by data processing techniques that extract lighting invariant features [281] and assess feature quality [282]. Despite that, perception quality remains a central issue and prevents the ADSs from becoming prevalent in vehicles.

In addition to illumination changes, another issue is the image space for camera-based perception, as the scale of the image scene is unknown in advance. In fact, making

use of scale information in dynamic tasks, such as obstacle avoidance is feasible by means of a single camera [283], although multi-view or stereo systems are more preferable in terms of robustness [284]. However, they lead to a considerable amount of computational load to an already complicated perception pipeline. As a relatively new and alternative sensing method for 3D perception, 3D lidars exist to solve the scale problem as they depend far less on illumination changes and intemperate weather. Nevertheless, their own-ing cost remains as a noteworthy problem as stated in section 1.2.

In view of the above, numerous recent works have been concerned with enhancing the perception quality of self-driving scenes on different collection scenarios and improv-ing the object classification and detection performances in real-time. For instance, Gao et al. [285] have fused lidar and vision data in an object classification task. Point clouds of lidar data and RGB images from the KITTI [28] dataset are used in the study. They ini-tially upsample and convert the point cloud data into depth feature maps at pixel-level, and the RGB images are also converted to depth feature maps. Following that, the inte-grated depth and RGB data are fed into a CNN. Thus, experiments on the KITTI [28] dataset exhibit superior object classification accuracy compared to using only depth or RGB data. Moreover, acceleration in feature learning and convergence has been achieved using lidar data.

Another fusion-based system [265], which employs a cheaper 4-beam lidar, rather than an expensive 64-beam one, and a stereo camera for 3D object detection, handles the cost problem of lidars. Due to a significant enhancement of the depth estimation tech-nique, the proposed method has shown an improved 3D object detection performance on the KITTI dataset [28].

In [286], the performance of point clouds of lidar data on 3D object detection is addressed by simulating clear weather to foggy weather compared to other sensor types. Details about more sophisticated recently published works can be found in VoxelNet [287], SECOND [288], IPOD [289], PointPillars [290], PointRCNN [291], F-PointNet [292].

4.7.3 Detection Types

So far, the investigated ADS related works have focused on the overall content of the self-driving scenes, or in other words, the big picture. On the other hand, enhance-ments of object classification and detection of specific features of the scene, such as pe-destrian, cyclist, traffic sign and light, pavement marking, are also quite important, e.g.,

from a safety point of view. Henceforth, these are called sub-classification and detection types.

In a recent study on pedestrian detection, Boyuan and Muqing [293] placed a novel Spatial Pyramid Pooling (SPP) network to YOLOv4 [244] detector for an improvement of the detection accuracy, where a Mish activation function [294] is used instead of Leaky ReLU. Then, the anchor parameters of YOLOv4 were optimized with a K-means clustering algorithm. As a result, an excellent pedestrian detection accuracy of 84.7% mAP was obtained with 36.6 fps in real-time, on the Caltech [51] dataset (introduced in Table 2.2). In [295], thermal imaging, enabling day/night time and illumination-independent data collection, was used to implement a robust pedestrian and cyclist detection using the Faster R-CNN [148]. Details of other related approaches are available in [296], [297], and [298].

Detection of traffic signalization, comprising elementarily traffic signs, traffic lights, and road surface markings in the surrounding of self-driving cars allows the ADSs to make correct decisions according to the traffic rules. There are several most recent studies regarding this research topic. Henschri and Mtibaa [299] propose a two-stage approach for traffic sign detection. In the first stage, only the shape of the signs, which are circular or triangular, are detected and classified by using HOG [213] features and support vector machines (SVM). Then, the second stage by means of a CNN model attempts to classify these detected shapes into their own subclasses. Finally, the approach is tested on the GTSDb benchmark [300] with improved results.

Another significant object for ADSs in road scenes is traffic lights. An SSD [224] detector is exploited for an adaptation study to detect traffic lights and small objects in [301]. The Inception-v3 CNN (section 3.3.2) is employed as the base network instead of the originally enlisted VGG CNN (section 3.2.3) to increase speed and accuracy. The study adapts a prior box generation allowing smaller strides in the latter network layers, which enables detections of smaller object. Non-maximum suppression (NMS) is also adapted to prevent multiple detections for a single object. Finally, an additional block is inserted to classify the states of the traffic lights (i.e., red, amber, or green). The model showed a good performance on the DriveU [302] traffic light dataset.

Road surface markings as an important component of traffic signalization have been also studied in recent years. Ye et al. [303] have addressed partly distorted and worn road markings by means of a two-stage model based on YOLO-v2 [238]. The first stage

is responsible for the detection of initial road markings with coordinates and class confidences of bounding boxes by using the YOLO-v2 CNN (section 4.5.4). In the second stage, a novel, lightweight, and transformation-invariant classification network, RM-Net, is used for road markings detection, which is able to tackle the distortions and surface wears. An annotated road marking detection dataset for public use is developed, consisting of $\sim 12k$ high-resolution images, grouped into 13 classes, which are collected under various weather conditions in day/night time. The model achieved 86.5% mAP on this dataset, outperforming other existing frameworks.

Details of other most recently performed works could be found in [304] and [305] for traffic sign detection, [306] and [307] for traffic light detection, [43] and [308] for road surface marking detection.

4.8 Summary

It is infeasible to identify and reach a decision on which limitations ADSs suffer from by only looking at the image classification algorithms, as done in Chapter 3. A novel CNN design, which addresses the drawbacks of ADSs, requires knowledge of both image classifiers and object detectors. Therefore, in this chapter, recent studies and main streams of object detection methods have been investigated in a similar manner to the previous chapter. Thus, CNN-based image classification and detection, ADSs system architectures, and current industry practices have been investigated to complete the meaningful data collection from both academy and industry in a synergetic way.

In particular, what Table 2.7, Table 3.1, and Table 4.1 represent and the analyses highlight that CNN models that feature low model sizes and achieve satisfactory accuracy have a promising future. In addition, it is strongly emphasized by the investigations that with the necessary real-time hardware capabilities and successful resolution of design constraints, CNNs would enable safer and more efficient self-driving cars and a rise in their popularity. As a result, the investigations up to this point have prominently promoted this research on the proposing of novel lightweight CNN models for ADSs pipelines, which may lead to notable improvements corresponding to what object detection modules of ADSs suffer from. The following chapter introduces a novel framework which reduces the computational budget of existing block-based CNNs while keeping their accuracies.

CHAPTER 5 DESIGN FRAMEWORK FOR LIGHT- WEIGHT HIGH-PERFORMANCE CNN MODELS

As it has been shown in Chapters 3 and 4, the most accurate recently proposed CNN models [22], [179], [181], [188], [191] usually include hundreds of convolutional layers with thousands of channels, which bring a huge computational overhead of billions of floating-point operations (FLOPs). This results in a large CNN model size, which is problematic in real-time autonomous applications [149], as they employ resource constrained embedded computing platforms based on semiconductor devices with limited on-chip memory, such as multi- and many-core CPUs, GPUs, FPGAs, and ASICs. This chapter addresses the aforementioned issues by proposing an advanced design framework, aimed at generating lightweight CNN designs capable of high performance.

The rest of the chapter is structured as follows. Section 5.1 introduces the proposed novel Sparse-Split-Parallelism (SSP) framework. In section 5.2, a brief overview of the design strategies underpinning the SSP framework and the incentives behind each of them is given, and then prior work is presented in section 5.3. Technical details of the framework are given in section 5.4. Evaluation of the image classification performance of CNN models, derived through application of the framework, is reported in section 5.5, employing various types of convolutions and operations to reveal the resilience of the framework under different conditions. Important aspects of the SSP framework are discussed in section 5.6, and its generalization ability regarding object detection and real-time performance are analysed in section 5.7. Finally, section 5.8 concludes the chapter.

5.1 Introduction

The exceptional image processing performance of Convolutional Neural Networks (CNNs) has made them a preferred technique for visual recognition tasks, such as image classification [22], object detection [148], style transfer [12], etc.. This has motivated their numerous applications across varied industries, for example robotics [309], self-driving cars [310], medical [311], satellites [312], aviation [313], power systems [314], etc.

Efficient utilization of on-chip SRAM memory offers 1000x cycles lower latency compared to off-chip DRAM memory [151] as also stated in section 2.2.3. Thus, a lightweight CNN model that could use the on-chip memory space of embedded platforms has a powerful potential for a faster execution. Moreover, it has been shown that FPGAs outperform the other counterparts in terms of processing latency and power consumption [20], which is crucial in many autonomous applications, and in particular in Autonomous Driving Systems (ADSs) [29], [15]. At the same time, on-chip memory of automotive-grade FPGAs is limited, often amounting to less than 10MB [165], [166], [168], [169], which shows the pressing need for high-performance reduced size CNN models. Additional advantages of smaller CNN models are that they lend themselves well to online and distributed training [149], [150], over-the-air updates [12], [149], and reduced memory bandwidth platforms [20].

The topology of the majority of the existing CNNs can be represented using three components: blocks, modules, and layers. A module represents a few wisely stacked adjacent convolutional layers like *bottlenecks* in ResNet [22], whereas a block is comprised of several identical and sequentially mounted modules, like *dense* blocks in DenseNet [24]. Ultimately, a sequential combination of blocks forms the final network model, also referred to as architecture.

Over the past few years, there have been two main work streams to lighten CNNs in terms of model size, computation, and inference time, which are: (i) compressing the size of pre-trained models [183], [315]; (ii) designing new lightweight models from scratch [26], [316]. This chapter presents a different design framework that reduces the computational complexity and inference time of existing block-based CNN models [12], [22], [24], [25], [26] without compromising their accuracy. Contrary to the existing approaches, it is applicable to any block-based models. In order to achieve these properties, the framework is structured from three novel purpose-designed strategies, applied at a block, module, and layer level. These comprise the following new techniques:

- at a block level, a novel *sparse skip connectivity* pattern is implemented, which is applicable to blocks of all block-based CNNs;
- at a module level, a novel *proportional channel split* operation is realised; and
- at a layer level, the idea of *degree of parallelism* [25] is employed.

Hence the proposed framework is referred to as SSP framework, where SSP stands for *Sparse-Split-Parallelism*. The default characteristic and design domain of the SSP framework targets to be applied to every single separate block of block-based CNNs with

the three novel design strategies on block, module, and layer level, thereby enabling SSP framework-applied blocks and holistically transforming the selected block-based networks to SSP-enabled networks.

5.2 SSP Framework Design Strategies

The block-level design strategy of the proposed SSP framework is based on the novel concept of ‘*sparse skip connections*’ between modules within a block. It could be viewed as a novel more efficient alternative to the method of densely connected skip connections [24], where the input of each module within a block is the concatenation of the outputs of all the preceding modules. The proposed method removes all redundant dense skip connections and employs only the strongest ones, which improves the model efficiency. It is established that the three-step skip connection pattern provides an optimal information flow.

The motivations behind this novel concept are explained by the following.

The dense connectivity pattern [24] brings the best-possible information and gradient flow across the blocks and barricades the problem of degradation [25]. However, due to its stringent connection density, it is not applicable to all existing or newly designed CNNs. In addition, its excessive feed-forward dataflow cumulatively increases the number of the feature maps at the inputs of sequentially connected modules and escalates the computational budget. As a result of that, it requires a higher memory bandwidth and training/inference time, especially on memory-bound computing platforms (e.g., GPUs) [317]. Furthermore, the efficiency of feature reuse within a block with such a dense connectivity decays exponentially based on the distance between two modules as experimentally shown in DenseNet [24] and ShuffleNet-v2 [25]. So, the proposed method in this research extends the existing concept of dense skip connections proposing a novel more efficient scheme, referred to as *sparse skip connections*.

The-proposed method is completely different from recent works [317], [318] that have also developed approaches to improving the efficiency of dense skip connections. This is because, it is the first work that makes skip connections employable in the majority of the block-based CNN models rather than only in DenseNet [24].

As a proof of concept, the proposed ‘*sparse skip connections*’ method was validated experimentally based on the DenseNet-121 CNN [24]. First, the model was trained using the ImageNet [21] dataset. Then, for each source module m within a block, the

average absolute L1 norm of the weights assigned to its connections with the target module t was computed.

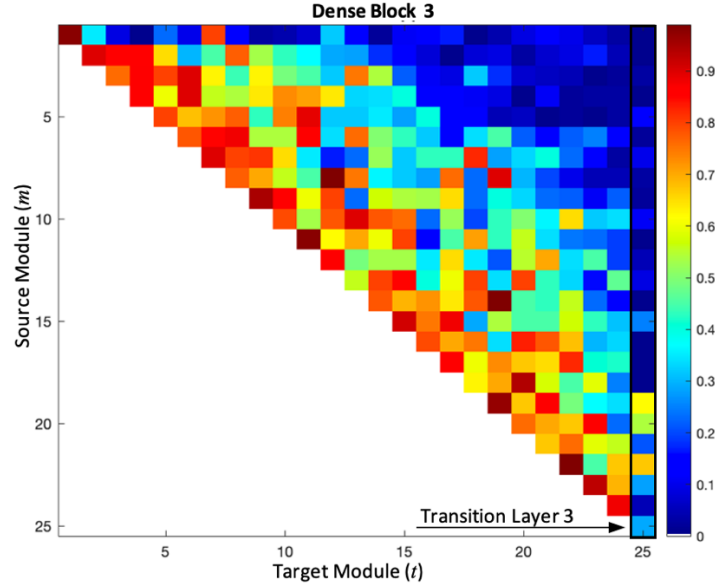


Figure 5.1: Illustration of the amount of feature reuse for the third block of a trained DenseNet-121.

Figure 5.1 illustrates the heat-map for the third block of the DenseNet-121 model, and similar heat-maps are observed for the remaining blocks as well. The colour of each pixel (m, t) encodes the average absolute L1 norm (normalised by the number of the input feature-maps) of the filter weights for each module. The weights connect the module m to the module t within the block where the rightmost column framed by a black rectangle corresponds to the third transition layer of DenseNet-121. The top row, corresponding to $m=1$, represents the weights connected to the output feature-maps of the second transition layer. Any pixel (m, t) in Figure 5.1 acts as a surrogate to show the dependency between the modules m and t . The scale of the colour bar on the right-hand side of the figure indicates the feature reuse intensity (i.e., nearer to the top the higher the feature reuse). For instance, a fully dark red pixel in any position (m, t) indicates that there is a fully effective feature reuse between the modules.

An apparent observation from Figure 5.1 is that feature reuse intensity is stronger, especially in connections skipping four steps at the most. Beyond this distance, there starts a gradually rising decay. A similar behaviour has also been demonstrated in ShuffleNet-v2 and DenseNet ($k=12$) with 40 layers, on the CIFAR dataset [23].

The motivations behind the second and third design strategies of the proposed SSP framework, originating from the ShuffleNet-v2 model [25], are to further enhance its efficiency.

In the module-level design strategy, a novel ‘*proportional channel split*’ operation with concatenation is applied to the modules, which differs from the one introduced in ShuffleNet-v2 [25]. This operation primarily enables a capability for making a trade-off between accuracy and model size.

As for the layer-level design strategy, the idea of the *degree of parallelism* [12], [25] is directly applied to the models designed with the proposed method. In theory, FLOPs (multiply-adds) can be seen as a direct metric for the computational budget. Contrary to that, the number of memory access operations [12] is a more decisive factor in practice. At this point, ShuffleNet-v2 proposes a general enhancement for this dilemma, in which the equal number of input and output channels in a convolutional layer ensures the lower bound of memory access number.

Thus, the *Sparse-Split-Parallelism* (SSP) framework is built to be applied to every single block of block-based CNNs by means of the introduced three design strategies above.

5.3 Prior Work

This section provides an overview of the research work on the lightening of CNNs in terms of computation, model size, and practical inference time, while preserving accuracy, over the last decade. The proposed approaches are grouped into two subcategories from the point of view of whether a pre-trained model is provided.

5.3.1 Compression Methods for Compactness

Mainly there are four different methods to compress a provided pre-trained network into a compact one: (i) *pruning* [315], [319]–[321] is focused on turning weight tensors into sparse ones by removing insignificant members (i.e., weights) in them; (ii) *tensor decomposition* [322]–[324] utilizes the spatial or channel redundancies of weight matrices and pursues their lower rank approximations; (iii) *quantization* [325]–[327] is based on the adaptation of low bit representations of each weight instead of floating point ones; (iv) *knowledge distillation* [183], [328]–[330] transfers the knowledge from complex models, referred to as *teacher*, to compact models, referred to as *student*.

All the above methods can compress to a considerable extent provided pre-trained models into lightweight ones without any loss of accuracy.

5.3.2 Compact Model Development

Compact model development by design strategies aims to create a computationally efficient CNN model, focusing on particular architectural components, such as spatial 2D convolutions.

For example, in ResNet [22], a *bottleneck* module is introduced and exploits two 1x1 convolution layers before and after a 3x3 convolution layer, which reduces the input and output channels. SqueezeNet [149] adopts a computationally efficient *fire* module, where concatenation is used as an element-wise operation instead of summation. Then, recent models MobileNet [147], ResNeXt [181], and Xception [201] deploy *depth-wise separable* (i.e., the combination of depth-wise and point-wise convolution) and *group* convolutions as alternative to spatial 2D convolutions.

However, in practice, depth-wise convolutions cannot be efficiently implemented on memory-bound hardware due to their arithmetic intensity ratio (FLOPs to memory accesses), which is significantly low, as detailed in [12], [25], [316]. The third design strategy of the proposed SSP framework, which is borrowed from [25], addresses this dilemma by enabling smaller memory accesses in each layer.

In another work, ShuffleNet [26] integrates depth-wise and point-wise group convolutions along with a newly proposed channel shuffling operation, upgrading the accuracy without causing a higher number of parameters. Lastly, a quite recent study [12] puts spatial convolutions entirely aside and introduces a FLOPs/parameter-free *shift* operation as an alternative. It moves each channel of the input tensor to different directions to ensure spatial and channel information communication by working co-ordinately with a subsequent point-wise convolution. In addition, the redundancies of the shift operation are resolved in a few follow-up studies [316], [331].

The insertion of *shortcut* connections among layers or modules has been another attractive field of interest, and its theoretical background [332], [333] has been established a long time ago. Highway Networks [334] is the first practical study introducing shortcut connections (by-pass) with gating functions [335] to efficiently train deeper networks. However, the gates are data dependent as well as having parameters. To remedy the discrepancy, ResNets [22] presents identity shortcuts (identity mapping) bringing breakthrough performance with parameter-free paths. Moreover, unlike by-pass connections

[334], identity shortcuts attend and stabilise the learning process at every stage and are never closed. A further analysis, Stochastic Depth [336], randomly drops the redundant layers of ResNets during training to allow a better performance. A few popular variants of ResNets are also proposed, such as ResNeXt [181].

Despite the pioneer performance of ResNet, its representation capability (i.e., information flow) is still limited due to the summation operation at the output of each layer. Another simple yet impressive approach, DenseNet [24], creates *dense short paths (skip connections)* from any early layer to every later layer by using element-wise concatenation operation in contrast to ResNet. Such a connectivity pattern enables efficient feature reuse. However, it brings many more *shortcut* connections compared to ResNet. In addition, dense connectivity with concatenation also requires a higher GPU memory and training time. These substantial shortcomings are fully addressed by implementing the novel concept of *sparse skip connections*, as part of the first design strategy of the proposed SSP framework.

There exist several other solutions. Zhang et. al. [317] propose *dense summation* as a fusion of dense concatenation [24] and identity summation [22], which is applied by means of dense weighted normalized shortcuts. Suchlike summation technique is also applied in another study [337] to reduce model complexity of DenseNet.

Lastly, two other existing works exploit dense connectivity. The first one, CondenseNet [318], introduces a novel module, *learned group convolution*, which gradually removes superfluous skip connections for feature reuse in the training process. The second one [338] employs a recurrent neural network (RNN) as an agent to automatically carry out pruning of redundant dense skip connections, creating a sparse skip connection map.

Although, the techniques reviewed above, enhance the performance of DenseNet, the proposed design approach in this research differs from all of them. This is because, it is introduced for the first time a novel *sparse skip* connectivity pattern, based on manually defined *n*-step-away paths, which can be applied to various state-of-art models, as detailed in section 5.4.

5.4 SSP Framework Design

Overarching approach to lightweight CNN design in the proposed SSP framework is to retain as much as possible the feature map diversity in the information flow, where

presence of label-related information is expected. In this section the technical details of the three design strategies, summarised in section 5.2, are presented in detail in subsections 5.4.1, 5.4.2, and 5.4.3. The application of the proposed framework to the blocks of selected popular block-based CNN models [12], [22], [24], [25], [26] is described in subsection 5.4.4.

5.4.1 Sparse Dense Connections

a) Related Work

The design process of a new CNN model hugely depends on the completion of the convergence without difficulties. Consistent with the solutions [239], [339] to the notorious vanishing gradient problem, a degradation problem emerges when increasing the depth of the network, where the accuracy initially gets saturated, and immediately after that a sharp degradation follows. This is caused by the lack or distortion of the information (source imagery) flow towards the end of deeper models. In ResNet [22] this is largely addressed by an identity mapping approach (i.e., residual networks are used instead of plain networks, simply stacking layers).

Let us consider a single block of a block-based CNN, where i_m and o_m denote the image tensors (feature maps) related to the inputs and outputs of each module m of a particular block; o_0 is equal to i_1 ; and the spatial dimensions of the input and output of each module are equal. Figure 5.2 shows an example of such a block with the proposed *three-step-away* skip connections, which comprises eight modules and each module combines l stacked layers. The transition layer represents the connection point between adjacent blocks on the block-based CNN. ‘*Three-step-away*’ in this context denotes the number of modules that are bypassed by the skip connections. Figure 6.2 will also be an illustration of a SSP framework-applied block.

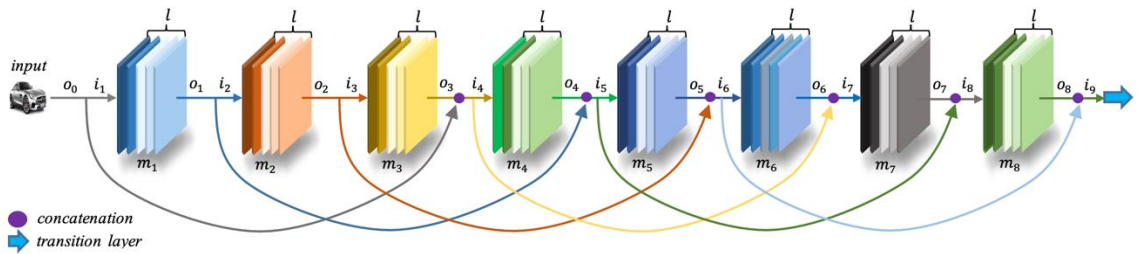


Figure 5.2: An illustration of a SSP framework-applied CNN block comprising eight modules with three-step-away sparse skip connections.

Let us now specify a non-linear transformation function $F_m(i_m)$, which maps the input of module m and produces its output $o_m = F_m(i_m)$, which is then fed to the input of the next module. The function comprises a combination of different operations (e.g., ReLU, Batch Normalization (BN), down-sampling, convolutional layers, etc.). It is referred to as underlying mapping in ResNet [22].

In ResNet, it is proposed to insert shortcut (one-step-away skip) connections between adjacent modules only, forming a simpler connectivity pattern than the one illustrated in Figure 5.2. These connections are also referred to as identity mapping. In this case, the input of module $m + 1$, i_{m+1} , is calculated as follows:

$$i_{m+1} = F_m(i_m) + i_m = o_m + i_m \quad (5-1)$$

where “+” denotes an element-wise summation.

Thus, the map function, captured by equation (5-1), represents the rise in the information flow, which serves the purpose of hampering distortions. In addition, this formulation does not bring extra parameters or FLOPs except the element-wise addition. As a result, ResNet achieves a notable alleviation of the degradation problem by smoothing the optimization, as gradients can flow through the identity mapping.

Then in DenseNet [24] a drawback of ResNet is addressed, which is caused by the element-wise summation impeding the information flow. For this purpose, a sophisticated connectivity pattern is introduced, in which the input i_m of each module is formed by concatenating the outputs o_m of each of the preceding modules, as follows:

$$o_m = F_m(i_m) = F_m([o_0, o_1, o_2, \dots, o_{m-1}]) \quad (5-2)$$

where $i_m = [o_0, o_1, o_2, \dots, o_{m-1}]$ symbolizes the concatenation of all its elements, from o_0 to o_{m-1} .

The map function, represented by equation (5-2), yields feature reuse in each module and boosts the information flow across the entire network. Despite that, the approach brings redundant connections as shown in Figure 5.1 and leads to a higher GPU memory demand, as stated in section 5.2. Another handicap is that dense skip connections involve numerous concatenation operations, even though they are parameter/FLOPs free. A recent study [25] proves that the element-wise operations occupy ~20% of the runtime, especially on GPUs.

b) Proposed Solution

To address the limitations of dense skip connections, outlined in subsection 5.4.1.a) above, a new approach is proposed, which is based on the selection of the most powerful skip connections to enable more efficient feature reuse in CNN models. In order to achieve that, a novel map function is formulated as follows:

$$o_m = F_m(i_m) = F_m([o_{m-1}, i_{m-s}]) \quad (5-3)$$

where $i_m = [o_{m-1}, i_{m-s}]$ is the concatenation of its elements; o_m denotes the output of m^{th} module; s is the step number. An illustration of it with three-step-away skip connections is demonstrated on Figure 5.2, which demonstrates the application of SS framework to a sample block.

As a result, the proposed novel connectivity pattern, captured by (5-3), gives rise to the novel design technique of *sparse skip* connections, which is behind the first design strategy of the proposed SSP framework. It could be applied to any single block of block-based CNN model, such as ResNet or DenseNet with a different number of steps depending on the number of the modules in a block.

c) Number of Modules

To be able to separately apply sparse skip connections to every single block of an existing block-based CNN architecture, the number of modules per block in the architecture has to be re-evaluated, as it needs to correspond to the selected step number. For that, during the application process, a piecewise function is defined to calculate the optimal number of modules as follows:

$$m = \begin{cases} m, & m/s \in \mathbb{Z} \\ \lfloor m \rfloor \text{ or } \lceil m \rceil, & m/s \notin \mathbb{Z} \end{cases} \quad (5-4)$$

where m and s denote the number of the modules and the number of steps in a block, respectively; $\lfloor m \rfloor$ and $\lceil m \rceil$ are equal to $m - a$ and $m + s - a$, respectively, and $a = m \bmod s$.

Which one to choose when applying the SSP framework to a selected block of CNN, $\lfloor m \rfloor$ or $\lceil m \rceil$, is a matter of making a trade-off between accuracy and model size, the latter being reduced when the lower one is being used.

Following these definitions, the efficiency ratio of the proposed connectivity pattern in terms of the skip connection number compared to DenseNet can be calculated as $[(m^2+m) / (4m-4)] \times 100\%$.

In addition, it is designed that each module in a block outputs g feature maps, which is called as *growth rate*, like applied on DenseNet [24]. Therefore, the proposed approach offers a modest channel number growth throughout the input feature maps of the modules, which has an essential role in the parameter/FLOPs reduction. Suppose that there exist m modules, and growth rate is g in one of the DenseNet block. The number of channels at the output of the last module after the concatenation operation is accumulated as g_0+gm , where g_0 is the channel number of i_1 . When the sparse skip connection approach is applied to this DenseNet block with identical hyperparameters, the last module produces $g_0+(m/s)g$ channels after the concatenation operation, which is notably narrower compared to the counterpart. As a result, the proposed approach in this research can be viewed as a special (sparsified) case of dense skip connections.

5.4.2 Proportional Channel Split

A well-tuned trade-off between accuracy and model size requires carefully optimized and synchronically working module design strategies. As part of the module-level design strategy of the SSP framework to be applied on a selected block, the channel split operation introduced in ShuffleNet-v2 [25] is deployed and it enables both local (module-based) information flow and a compound element-wise operation structure providing lower inference time. In contrast to the half-and-half application of this operation in [25], this research advances it further, creating a novel operation referred to as *proportional channel split*.

The proportional channel split operation is accomplished with a hyperparameter ($\mathcal{E}$), characterizing channel division rate, which is capable of achieving a trade-off between accuracy and FLOPs/parameter numbers. Figure 5.3 (a) shows the proposed novel SSP module of SSP framework on each block, which contains a holder space for mapping original modules of existing architectures. In Figure 5.3 (b), Figure 5.3 (c), Figure 5.3 (d), Figure 5.3 (e), and Figure 5.3 (f), the module designs of the five reference CNN models are illustrated, and g and f denote the number of filters in Figure 5.3. In the proposed novel SSP module for SSP framework application, after the first 1×1 convolution reducing the input channel number to the growth rate, the proportionally split branches are concatenated after completing any other operations. In a similar spirit to ResNet [22], this

novel operation also acts as identity mapping for the modules. In section 5.5, the value of the hyperparameter ($\mathcal{E}$) is defined in the model descriptions and analysed further for changing values.

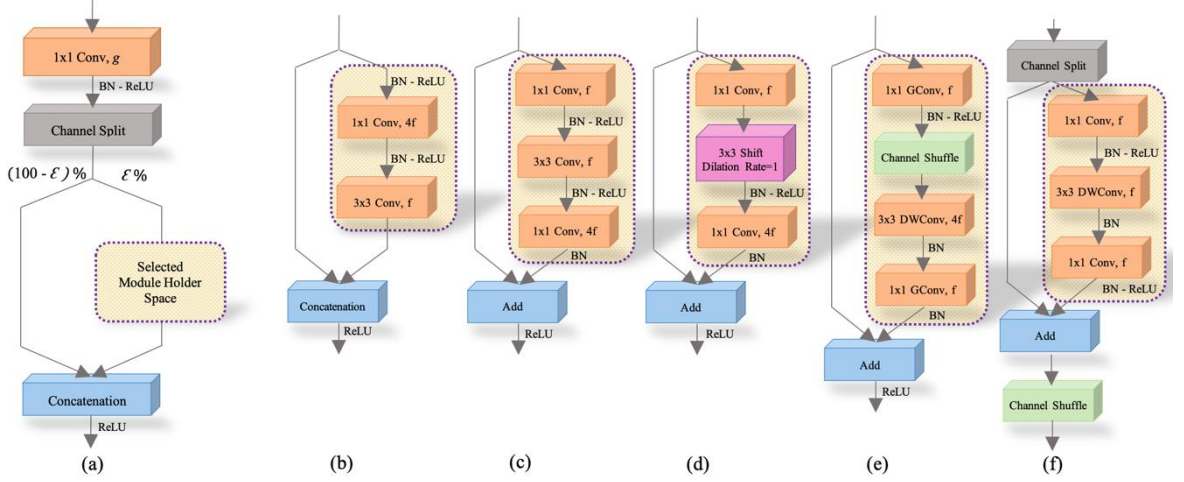


Figure 5.3: Illustration of module design on SSP framework application for a block. (a) the proposed SSP module of SSP framework; (b) DenseNet bottleneck module [24]; (c) ResNet bottleneck module [22]; (d) ShiftNet CSC module [12]; (e) ShuffleNet Unit module [26]; (f) ShuffleNet-v2 Basic Unit module [25].

5.4.3 Degree of Parallelism

The layer-level design strategy of the proposed SSP framework for each block is partly inspired by the *degree of parallelism* concept in ShuffleNet-v2 [25], which characterizes the high number of costly external memory accesses, which consume 1000x more energy than a single arithmetic operation [12]. A good example of that are depth-wise separable convolutions, offering a fewer number of parameters and FLOPs compared to spatial convolutions. However, their memory access cost is notably higher, which brings a slower inference time and a higher energy consumption.

To address the issue, the unequal filter number effect in sequential convolutional layers, as shown in Figure 5.3 (c), where three sequential convolutional layers have filter numbers $[f f 4f]$, was analysed further. The study [25] revealed that inequalities in the filter numbers increase the number of memory accesses. Therefore, balanced convolutions, i.e., equal filter numbers or channel widths, were made use of inside the module of ShuffleNet-v2, as seen in Figure 5.3 (f), where the filter number of each convolutional layer $[f f f]$ is equal. This idea has been applied inside the modules of a block in this research, and it shapes the layer-level design strategy on the SSP framework application to a block.

5.4.4 Framework Application

The SSP framework, discussed in subsections 5.4.1, 5.4.2, and 5.4.3 above, can be separately applied to any single block of block-based CNNs with similar architectures to DenseNet [24], as well as ResNet [22], ShiftNet [12], ShuffleNet [26], and ShuffleNet-v2 [25]. These five models involve almost all different types of convolutions and operations frequently used in many other state-of-the-art CNN models. The real power of the framework is revealed by the experimental results in section 5.5.

Here the application of the framework using the proposed three strategies is described on the example of DenseNet-12 blocks, one of the versions of the DenseNet CNN model [24]. It has four sequential blocks comprising [6 12 24 16] modules respectively. Each module is identical and has the design illustrated in Figure 5.3 (b).

Starting with the block-level strategy of the SSP framework as demonstrated in Figure 5.2, equation (5-4) is initially applied to redefine the number of modules for each single SSP-enabled block, so that the sparse skip connections can be applicable with any selected step number without any overflow. Thus, the sparse skip connections are defined and separately adapted to each single block of DenseNet-121.

Then, following the module-level strategy on every single block, the right path of the bottleneck module of DenseNet-121 is incorporated into the module holder space, indicated by a dashed line rectangle with a yellow background in the SSP module, shown in Figure 5.3 (a). The inspiration behind the module-level design came from a similar technique, which was utilised in ShuffleNet [26], for the purpose of assessing the performance of the *shuffle* operation using the feature learning layers of the ResNet *bottleneck* [22] and VGG [178] modules.

Lastly, the layer-level strategy is applied, whereby the number of channels is readjusted in the output of each layer in the selected module of each block to enhance the *degree of parallelism*. For example, following the layer-level design, the output channels for the two adjacent layers Figure 5.3 (b), which are [4f f], are readjusted to [f f] in the each block of SSP-enabled model, respectively.

As a result, the application of three design strategies of SSP framework, which is dedicated to be applied on every single block of an existing network, is completed. The newly generated network is referred to as SSP-DenseNet-121, as a novel SSP-enabled model. There exist several extra details while applying the SSP framework, which are discussed in section 5.5.

5.5 Evaluation Results

As part of the evaluation process, first a range of SSP-enabled models are constructed by applying the SSP framework to popular state-of-the-art CNN models. This is followed by defining the optimal number of steps, based on assessing the framework’s ability with different step numbers. The framework is then deployed to the model improvement with the optimal number of steps as a default, and finally the performance of the models on several different applications is evaluated.

5.5.1 ImageNet Dataset Based Evaluation

To start with, end-to-end trainable models with three-step-away skip connections are constructed by applying the SSP framework to five existing CNN models, DenseNet, ResNet, ShiftNet, ShuffleNet and ShuffleNet-v2, which are evaluated using the ImageNet [21] dataset.

Table 5.1: Implementation details of the SSP-enabled models.

		SSP- DenseNet-121	SSP- ResNet-50	SSP- Shift50	SSP- ShuffleNet 1x (g=3)	SSP- ShuffleNet v2 1x
Layers	Output Size	Kernel Size / Output Channels / Stride / xRepeat ^a				
Conv.	112x112	7x7 / 64 / s=2 / -	7x7 / 64 / s=2 / -	7x7 / 64 / s=2 / -	3x3 / 24 / s=2 / -	3x3 / 24 / s=2 / -
Pooling	56x56	3x3 ^b / 64 / s=2 / -	3x3 ^b / 64 / s=2 / -	3x3 ^b / 64 / s=2 / -	-	-
SSP	56x56	- / 192 / - / x6	- / 128 / - / x3	- / 128 / - / x3	-	-
Transition	56x56	1x1 / 192 / - / -	-	-	-	-
	28x28	2x2 ^c / 192 / s=2 / -	2x2 ^c / 128 / s=2 / -	2x2 ^c / 128 / s=2 / -	5x5 ^b / 24 / s=4 / -	5x5 ^b / 24 / s=4 / -
SSP	28x28	- / 448 / - / x12	- / 256 / - / x3	- / 256 / - / x3	- / 84 / - / x3	- / 140 / - / x3
Transition	28x28	1x1 / 448 / - / -	-	-	-	-
	14x14	2x2 ^c / 448 / s=2 / -	2x2 ^c / 256 / s=2 / -	2x2 ^c / 256 / s=2 / -	2x2 ^c / 84 / s=2 / -	2x2 ^c / 140 / s=2 / -
SSP	14x14	- / 960 / - / x24	- / 768 / - / x6	- / 768 / - / x6	- / 444 / - / x9	- / 836 / - / x9
Transition	14x14	1x1 / 960 / - / -	-	-	-	-
	7x7	2x2 ^c / 960 / s=2 / -	2x2 ^c / 768 / s=2 / -	2x2 ^c / 768 / s=2 / -	2x2 ^c / 444 / s=2 / -	2x2 ^c / 836 / s=2 / -
SSP	7x7	- / - / 1280 / x15	- / 1280 / - / x3	- / 1280 / - / x3	- / 684 / - / x3	- / 1300 / - / x3
Conv.	7x7	1x1 / 1024 / - / -	1x1 / 1024 / - / -	1x1 / 1024 / - / -	1x1 / 1024 / - / -	1x1 / 1024 / - / -
Pooling	1x1	7x7 ^d / 1024 / - / -	7x7 ^d / 1024 / - / -	7x7 ^d / 1024 / - / -	7x7 ^d / 1024 / - / -	7x7 ^d / 1024 / - / -
FCL ^e		1000	1000	1000	1000	1000

^a Module number in each SSP block. ^b 2D max pooling. ^c 2D average pooling. ^d 2D Adaptive average pooling. ^e Fully Connected Layer

Table 5.1 shows the implementation details of the SSP-enabled models, SSP-DenseNet, SSP-ResNet, SSP-ShiftNet, SSP-ShuffleNet and SSP-ShuffleNet-v2. In Table 5.1, the rows named SSP in the leftmost column, represent the blocks to which the proposed approach, SSP framework, is applied to in the selected models. In all basic models except of DenseNet, the original model operations for batch normalization and nonlinearity placements are followed. The inside of the modules differs from the original Dense-

Net by using the sequence of Conv-BN-ReLU instead of BN-ReLU-Conv. In the transition layers, a BN-ReLU operation is placed after the 1x1 convolution, which was the only placement of BN before it in the original model. Regarding SSP-DenseNet, SSP-ResNet, and SSP-ShiftNet in Table 5.1, the first convolutional and pooling layers are kept completely the same. As another quite efficient and powerful strategy from ShuffleNet-v2 [25], the last convolutional layer of it is adopted for all other models to regularize the output channel numbers after the last SSP framework-applied block in Table 5.1.

a) SSP-Enabled Models

The resultant SSP-enabled models are described below:

- **SSP-DenseNet:** DenseNet-121 and DenseNet-169 [24] are selected to be improved with the SSP framework. At block level, they have four blocks with [6 12 24 16] and [6 12 32 32] modules, respectively. After applying these module numbers to (5-4), based on three-step away skip connections for each separate block, the new module numbers are [6 12 24 15] and [6 12 30 30] respectively, where the floor values $\lfloor m \rfloor$ are selected for lower model size. In the original DenseNet models, there are *transition* layers between adjacent blocks, which enable reduction of the excessive output channel number (to half in most of the models) as well as the output size of the preceding block. Similar transition layers are placed in all the proposed models in this research as seen in Table 5.1, where an 1x1 convolution and average pooling layers are placed for these tasks.

There is no need to reduce the output channel size in the SSP-based models as the three-step connections technique brings quite modest rises in the output channel numbers of each block compared to the original DenseNet models. Such an approach also serves the *degree of parallelism* strategy by removing three extra convolutional layers having unbalanced channel widths. In Table 5.1 similar *transition* layers are also used in other models but with only downsampling without any convolutional layers, as there is no need to reduce their already modest channel growth.

At module level, the constant value for the growth rate (g) is 64 instead of the value 32 used in all original DenseNet models. As it could be seen in Figure 5.3 (a), the entry 1x1 convolutional layer produces $g = 64$ channels, which is then split according to the selected $\mathcal{E}$. In the SSP-DenseNet and all other models, the split ratio $\mathcal{E}$ is adjusted to three quarters (i.e., 75%). Thus, three-quarters of channels flow

through the underlying mapping (right branch) while the rest (a quarter) go through the identity mapping (left branch) to join in the concatenation.

At layer level, as described previously in section 5.4.4, the underlying mapping of DenseNet, shown in Figure 5.3 (b), is incorporated into the SSP module mapping space, shown in the right branch of Figure 5.3 (a). In addition, the order of the filter numbers (f) is changed from $[4f\ f]$ to $[f\ f]$, and all the channel numbers are adjusted to $f=48$, which is 75% of 64 (g).

Thus, by implementing the parameter values in Table 5.1 and transferring structural elements, as illustrated in Figure 5.3 (a) & Figure 5.3 (b), the novel SSP-enabled models, SSP-DenseNet-121 and SSP-DenseNet-169 are created.

- **SSP-ResNet:** ResNets [22] employs two types of *bottleneck* modules with identity or projection shortcuts. Figure 5.3 (c) illustrates the identity one and projection shortcuts are placed in case of downsampling of feature map sizes. In the SSP-enabled models, SSP-ResNet-50 and SSP-ResNet-101, it is avoided to use projection shortcuts, as the transition layers are exploited, such as in DenseNet (details are given in Table 5.1). A similar structure also exists in the remaining SSP-enabled models, where the issue is addressed in a similar manner as well. The original module numbers of ResNet-50 are transformed from $[3\ 4\ 6\ 3]$ to $[3\ 3\ 6\ 3]$ via equation (5-4), where a gradually rising growth rate g $[64\ 128\ 256\ 512]$ is used in each block with split $\mathcal{E}$ (75%), respectively. Finally, as regards to the third SSP strategy on each block, the ratios of the bottleneck layers' channel widths $[f\ f\ 4f]$ are transformed to $[f\ f\ f]$.

- **SSP-ShiftNet:** To test the framework ability under various conditions, it is also applied to ShiftNet [12] introducing a more sophisticated approach, whereby the *shift* operation is substituted by spatial convolutions. Apart from the *shift* operation at the middle of the CSC module as demonstrated in Figure 5.3 (d), all other settings are identical to SSP-ResNet. In the implemented experiments in this research, the shift operation is captured as a PyTorch customized C++/CUDA extension.

- **SSP-ShuffleNet and SSP-ShuffleNet-v2:** Unlike the previous models, ShuffleNet [26] and ShuffleNet-v2 [25] involve three blocks with the same sequence of module numbers $[4\ 8\ 4]$, which is then changed to $[3\ 9\ 3]$ in the SSP-based models by means of equation (5-4). The composition of the original modules can be seen in Figure 5.3 (e) and Figure 5.3 (f), where the shuffling operation allows a cross-group information interchange.

The SSP framework is applied to each block of ShuffleNet 1x ($g=3$), ShuffleNet 1x ($g=8$), Shufflenet-v2 0.5x and Shufflenet-v2 1x, where g represents group number and 0.5x and 1x represent model complexity. Differently from the original models, both employ identical first convolutional layer and a max-pooling layer, which is placed in first transition layer, as shown in Table 5.1. Like the previous models, the channel width ratio of ShuffleNet is converted to the $[f\ f\ f]$ instead of $[f\ f\ 4f]$. However, ShuffleNet v2 keeps it the same as it was already $[f\ f\ f]$, as shown in Figure 5.3 (f). A gradually rising growth rate g is also applied to both models with $[60\ 120\ 240]$ and $[116\ 232\ 464]$. As a small detail, ShuffleNet differently from the SSP-enabled models does not employ a group convolution in the first module of the first block as its input channel number is rather small, i.e., 24.

As a result of applying the proposed SSP framework to each block of the selected basic models, ten novel SSP-enabled models are built, as detailed in Table 5.1. For the interested stakeholders in the reimplementation of the SSP-enabled models, Figure 5.3 and Table 5.1 as well as the descriptions of the SSP-enabled models above provide all the guidance that is required for the SSP framework application to each separate block.

b) Training

As the primary target of the experimental work, the SSP-enabled variants of the reference models are evaluated performing image classification with the 1000-class ImageNet [21] dataset, which consists of 1.28 million training, 50k validation, and 100k test images. The image number in each class, termed also as ‘synset’ [21], ranges between 732 and 1300.

In the training process, the frequently used scheme [22], [24], [340], [341] is employed for data augmentation and weight initialization. The proposed models are trained for 90 epochs on NVIDIA Tesla P100 GPUs, part of the ALICE High-Performance Computing Facility at the University of Leicester. The Stochastic Gradient Descent (SGD) optimization algorithm with Nesterov momentum extension [342] is used to determine the parameters of the SSP-enabled models with a batch size of 512. The initial learning rate is 0.1 decaying by 10 at every 30 epochs. Weight decay and Nesterov momentum are selected as $1e-4$ and 0.9, respectively. To prevent overfitting, dropout strategies of the selected original models [12], [22], [24]–[26] are followed. Mixed-precision [343] is used for a more efficient GPU memory usage, as the naive implementation of the SSP-enabled models may tend to memory inefficiency. Classification errors are evaluated using the

validation dataset in a similar manner as in [24], [341]. Standard single-crop and 10-crop testing are applied in the comparisons with other studies.

c) Optimal Step Number

In this subsection, the optimal step number is defined, which most powerfully enhances feature reuse across the network. The guidelines provided for the creation of the SSP-enabled models in section 5.4 and subsection 5.5.1.a) above allow to construct novel SSP-enabled models with a different number of steps in sparse skip connections.

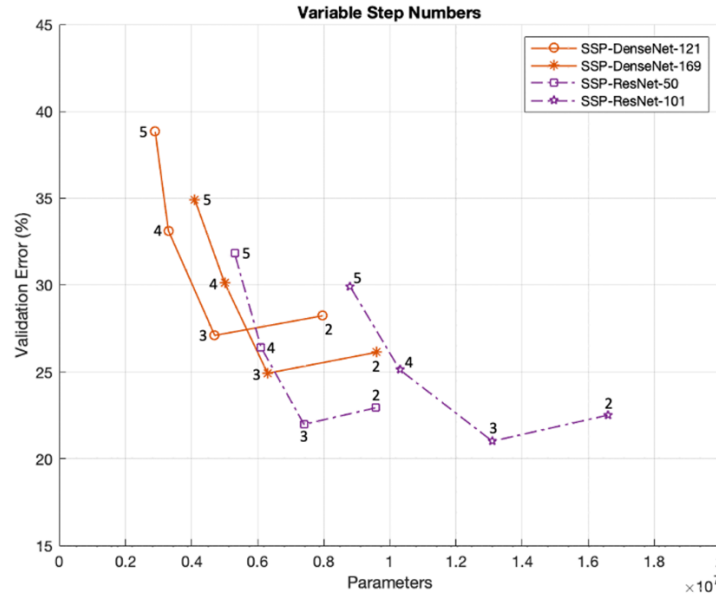


Figure 5.4: Results of the investigation of the most powerful step number in feature reuse. The numbers on the four lines represent the selected step number.

An investigation was carried out with SSP-DenseNet-121 & SSP-DenseNet-169 and SSP-ResNet-50 & SSP-ResNet-101 using step numbers in the interval of two to five, where the most efficient skip connections exist, as depicted in Figure 5.1. The experimental results are illustrated in Figure 5.4, where the numbers on the lines represent the applied step number for each model. A surprising result, as seen in Figure 5.4, is that the amount of feature reuse after step number three decays exponentially and leads to an excessive accuracy loss in the models. Another interesting outcome is that two-step-away connections show a higher error compared to three-step-away ones. The reason behind this is that models with two-step-away connections severely cut the feature reuse fetched from the preceding modules and limit the information flow to a narrower space.

The more surprising aspect in Figure 5.4 is the symmetry between all models. As a result, three-step-away skip connections experimentally show the best Top-1 validation

accuracy performance in feature reuse. This is significant, because it proves that performance gains from three-step-away skip connections are model-independent, and result in encoding more information and strengthening feature reuse. These results are also supported by [24], [25], and [338].

d) Experimental Results

Following the investigation outlined above, all models are trained on the ImageNet dataset with 3-step-away paths.

The full outcome of the evaluation process with the ImageNet dataset is summarised in Table 5.2. It could be observed from Table 5.2 that after the application of the SSP framework, the novel models consistently outperform their counterparts in achieving a better trade-off between model complexity and accuracy. The SSP-enabled models achieve $\sim 3\times$ lower parameters and $\sim 2\times$ lower FLOPs with negligible accuracy loss on DenseNet, ResNet, and ShiftNet. In addition, they perform with a $\sim 3\%$ lower error, having roughly the same number of parameters on ShuffleNet, and with a $\sim 6\%$ lower error, having a slightly higher number of parameters on ShuffleNet-v2. The reason why the SSP-enabled model suffers from increase of the computational load on ShuffleNet-v2 is that the topology of the network is quite compact by default. Moreover, the SSP framework is already inspired by the architecture of ShuffleNet-v2 [25], as part of the second and third design strategies (see sections 5.4.2 and 5.4.3).

Table 5.2: Top-1 and Top-5 error rates (single-crop/10-crop testing) on the ImageNet validation dataset with other parameters.

Model	Depth	$\sim$ Parameters ($\times 10^6$)	$\sim$ FLOPs ($\times 10^9$)	Top-1 (%)	Top-5 (%)	Speed (Batch size ^a / Second) GPU / CPU
DenseNet-121	121	8.0	5.5	25.02 / 23.61	7.71 / 6.66	121 / 27
SSP-DenseNet-121	177	4.7	2.9	27.10 / 25.08	9.32 / 7.43	187 / 35
DenseNet-169	169	14.1	6.7	23.80 / 22.08	6.85 / 5.92	97 / 21
SSP-DenseNet-169	240	6.3	3.1	24.92 / 23.21	7.23 / 6.83	159 / 31
ResNet-50	50	23.7	3.8	20.74 / 22.85	5.25 / 6.71	94 / 13
SSP-ResNet-50	48	7.4	1.6	21.98 / 22.97	6.21 / 7.03	151 / 22
ResNet-101	101	44.5	7.6	19.87 / 21.75	4.60 / 6.05	56 / 6
SSP-ResNet-101	93	13.1	3.1	21.01 / 21.91	4.93 / 6.01	101 / 11
Shift50	50	21.9	3.1	20.03 / 21.03	4.85 / 5.53	105 / 12
SSP-Shift50	48	6.3	1.5	21.67 / 22.94	5.91 / 6.03	161 / 21
Shift101	101	40.8	7.3	17.93 / 18.02	4.02 / 4.97	61 / 7
SSP-Shift101	93	9.6	2.8	19.78 / 20.03	4.97 / 5.42	123 / 16
ShuffleNet 1x (g=3)	50	1.8	0.14	32.60 / 35.01	12.67 / 13.89	221 / 24
SSP-ShuffleNet 1x (g=3)	51	1.7	0.13	30.07 / 32.18	10.32 / 11.87	227 / 27
ShuffleNet 1x (g=8)	50	2.5	0.14	32.40 / 34.86	13.33 / 15.91	203 / 23
SSP-ShuffleNet 1x (g=8)	51	2.3	0.13	30.52 / 32.11	12.05 / 13.41	217 / 26
ShuffleNet v2 0.5x	51	1.4	0.04	39.70 / 40.52	15.87 / 15.93	438 / 63
SSP-ShuffleNet v2 0.5x	51	2.0	0.07	35.97 / 36.07	12.46 / 12.77	351 / 55
ShuffleNet v2 1x	51	2.3	0.15	30.60 / 31.30	12.32 / 12.97	352 / 29
SSP-ShuffleNet v2 1x	51	4.6	0.27	23.95 / 24.83	6.87 / 6.89	247 / 24

Note: Unprovided accuracy, parameter number, and FLOPs of the basic models are calculated, and their training strategies are implemented for fair assessment. ^a Batch size is 8 for GPU and 1 for CPU.

Figure 5.5 illustrates the optimization process for ResNet-50, SSP-ResNet-50, ShuffleNet 1x, and SSP-ShuffleNet 1x, depicting the error rates as a function of the epochs both during training and validation. It could clearly be seen from Figure 5.5 that the SSP-enabled models achieve convergence faster compared with their ancestors.

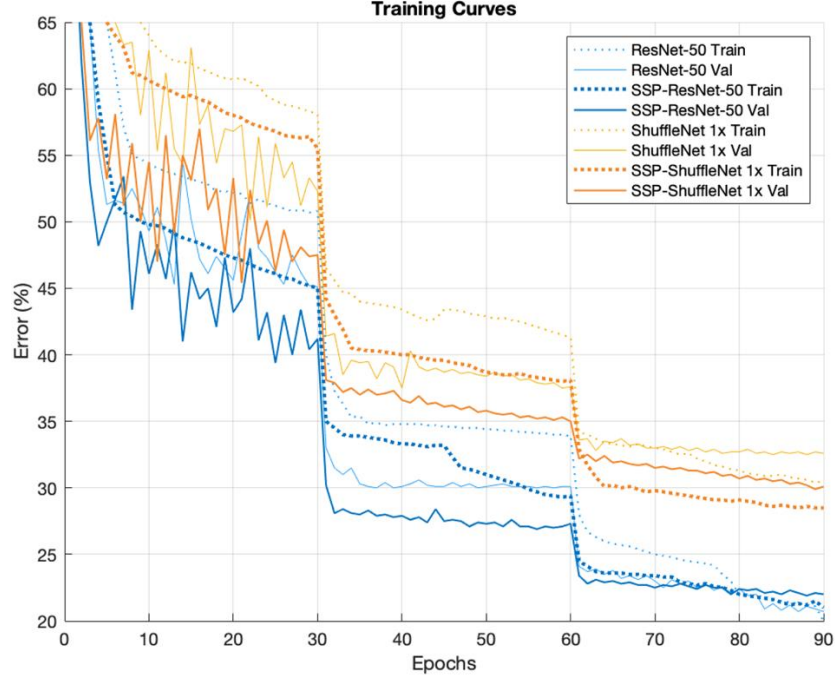


Figure 5.5: Error rates for ResNet-50, SSP-ResNet-50, ShuffleNet 1x ($g=3$), and SSP-ShuffleNet 1x ($g=3$) during training and validation, where the dotted curves correspond to the training phase.

Another innovative feature of the SSP framework is that it can customize the SSP-enabled models by controlling the desired complexity via the scale factor $\mathcal{E}$, which can be applied simply without affecting the organization of the modules or network. It characterizes the split ratio of the growth rate g of the modules. In this way, the SSP framework gains the ability to make a trade-off between model size and accuracy.

Figure 5.6 shows the performance of SSP-DenseNet-121 and SSP-ShiftNet50 for various $\mathcal{E}$ hyperparameters, where the numbers on the curves represent the $\mathcal{E}$ values. An explicit observation from Figure 5.6 is that accuracy degrades when $\mathcal{E}$ ranges from 75 to 100. This is due to the reduction in the channel size of identical shortcuts, which hampers the feature reuse in module-wise design leading to accuracy loss. The most striking outcome from Figure 5.6 is that the models are quite sensitive to the hyperparameter. In particular, after half-and-half split, (i.e., when $\mathcal{E}$ equals to 50%) the accuracy rapidly goes down. This is because feature extraction is strongly impeded due to the smaller channel

sizes in the layers. This shows the powerful nature of the proposed hyperparameter concept, which could be exploited further to enhance the performance of the SSP framework. Overall, the optimal $\mathcal{E}$ value ranges in the interval of 65-85 for the models.

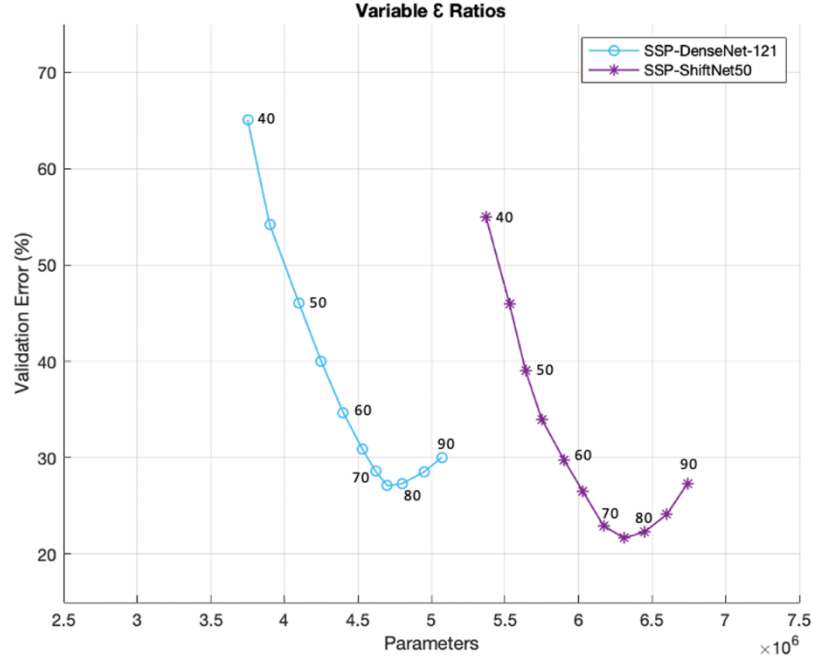


Figure 5.6: Making trade-off between model size and accuracy with different ratios of channel split via $\mathcal{E}$ hyperparameter.

e) Inference Time

Based on the experimental results up to now, it could be expected that the SSP-enabled models would show similar performance in terms of the inference time as well. However, it should be noted that the FLOPs (M-Adds) metric only takes account the convolutional parts. Even though they consume the most time, a considerable amount of time is also occupied by the other operations, such as data shuffle, data I/O, element-wise operations (summation, concatenation, ReLU, etc.). Therefore, FLOPs is an indirect metric and is not a convincing enough measurement of actual runtime [12], [25].

The inference time of the models are further analysed as a direct metric speed or latency. The last column in Table 5.2 gives the speed of the basic and SSP-enabled models on both compute-bound (CPU) and memory-bound (GPU) computation platforms. A higher value of speed (lower latency) in Table 5.2 represents a better actual runtime performance. Unsurprisingly, there is no proportional relationship between model complexity (FLOPs) and speed. For example, while the SSP framework reduces the model complexity of ResNet-101 more than 2x, the speedup is lower than 2x. That supports the

considerations introduced above. Nevertheless, a similar improvement in inference time is also observed with speedup as a direct metric (*speed*), which strongly emphasizes the capability of the SSP framework. A deeper analysis of this column in Table 5.2 is discussed in section 5.6.

5.5.2 CIFAR Datasets Based Evaluation

The applicability and effectiveness of the SSP framework is further investigated on a few basic models, DenseNet-40 & DenseNet-100, ResNet-20&ResNet-56 and ShiftResNet-20&ShiftResNet-56, originally designed for the CIFAR10&CIFAR100 classification task [23].

There are several design features that are common with the ImageNet models, discussed in section 5.5.1. The same adaptation procedure is followed with the difference that there are *three* SSP framework-applied blocks and *two* transition layers in the selected basic models for the CIFAR dataset. The channel split ratio $\mathcal{E}$ is applied as 75% in all models and the number of modules in the blocks are re-calculated via the application of equation (5-4) as in the ImageNet models.

In contrast to the ImageNet models in Table 5.1, max pooling is not used, since there is no need to reduce the output size before the first SSP framework-applied block. This is done in the transition layers by gradually reducing it to 16x16 and 8x8 via average pooling. In addition, the final pooling operation in the CIFAR models is average pooling.

a) SSP-Enabled Models

The resultant SSP-enabled models are described below:

- **SSP-DenseNet:** The DenseNet-40 & DenseNet-100 [24] models (growth rate g is 12) are selected for the tests. The growth rate g is increased to 48 across the network as the connectivity pattern allows us to use more. The strategies on BN and ReLU placement are identical to the DenseNet ImageNet models. Before the first SSP-framework-applied block, the filter number of the first convolutional layer is selected to be 48. In a fashion similar to SSP-DenseNet for ImageNet, a 1x1 convolutional layers are used in transitions to compress channel numbers. In SSP-DenseNet-40, compression is not applied in the transition layers. As for SSP-DenseNet-100, it is reduced half at every transition layer.
- **SSP-ResNet:** The ResNet-20 & ResNet-56 [22] basic models are used. In transitions, only the output sizes are reduced. The filter number of the first convolution is

16 in both models. A gradually rising growth rate g [16 32 64] in the three SSP framework-applied blocks is applied as in the ImageNet models.

- **SSP-ShiftNet:** The ShiftResNet-20 & ShiftResNet-56 [12] models are selected to prove the capability of the framework under changing conditions as ShiftResNet deploys the shift operation instead of the generic spatial convolution. CSC [12] modules are placed into the SSP modules as shown in Figure 5.3 (a) and Figure 5.3 (d). All other hyperparameters are the same as in the SSP-ResNet models.

Following the design details above, six different SSP-enabled novel models are created to be tested on the CIFAR task.

b) Datasets and Training

Both datasets comprise 50k training and 10k test set with 32x32 colour images. In CIFAR-10&CIFAR-100, each class is represented by 6k and 600 images respectively. The implementation for CIFAR-10 & CIFAR-100 employs the scheme in [12], [22], [24]. In training, 5k of the training images are held out as a validation set.

The models are trained with the same optimization algorithm as used in the ImageNet models by employing a batch size of 128 on NVIDIA Tesla P100 GPUs, part of the ALICE High-Performance Computing Facility at the University of Leicester. The initial learning rate is 0.1, which is then divided by 10 at 32k and 48k iterations and terminated at 64k iterations. Weight decay and Nesterov momentum are selected as $1e-4$ and 0.9 respectively. Weight initialization, BN, and a simple method of data augmentation are adopted from [22], [239], and [340] respectively. To prevent overfitting, dropout strategies of the selected original models [12], [22], [24]–[26] are followed. Finally, the models are evaluated on the test sets without and with data augmentation.

c) Experimental Results

The outcome of the evaluation of the SSP-enabled models, using the CIFAR datasets, is summarised in Table 5.3. It could be seen from the data in Table 5.3 that the SSP framework can significantly reduce the model complexity of the existing models in terms of number of parameters. For instance, SSP decreases the parameter number of DenseNet100 $\sim 4.7x$ and of ShiftResNet20 $\sim 3.5x$. In addition, compared to the ImageNet models in section 5.5.1 above, CIFAR models achieve a higher complexity reduction. However, the original CIFAR models are consistently more accurate than the SSP-enabled ones.

Table 5.3: Top-1 error rates (%) of SSP-enabled models on CIFAR-10&100 test sets without/with data augmentation.

Model	Depth	Parameters ($\times 10^6$)	CIFAR-10	CIFAR-100
DenseNet40 (k=12)	40	1.0	7.00 / 5.24	27.55 / 24.42
SSP-DenseNet40	40	0.6	7.89 / 6.54	28.65 / 25.03
DenseNet100 (k=12)	100	7.0	5.77 / 4.10	23.79 / 20.20
SSP-DenseNet100-C	94	1.5	7.32 / 6.95	26.43 / 24.22
ResNet20	20	0.27	9.31 / 8.75	31.44 / 31.02
SSP-ResNet20	20	0.10	10.9 / 10.1	35.0 / 34.87
ResNet56	56	0.85	7.83 / 6.97	27.23 / 25.9
SSP-ResNet56	56	0.4	8.12 / 7.79	30.12 / 28.81
ShiftResNet20	20	0.25	8.30 / 7.41	30.87 / 30.13
SSP-ShiftResNet20	20	0.07	9.75 / 8.31	32.33 / 33.23
ShiftResNet56	56	0.76	6.76 / 5.23	27.03 / 25.15
SSP-ShiftResNet56	56	0.28	8.27 / 7.33	29.3 / 28.01

Note: Unprovided values are substituted with calculated ones. Dropout is used when obtaining results without data augmentation.

5.6 Discussion

Fundamentally, the resultant SSP-enabled models are quite similar to their ancestors. This is because the proposed map function (5-3), underpinning the block-level strategy, is related to the previously proposed formulations (5-1) and (5-2). Moreover, it is strengthened by the other two design strategies. In fact, although these design strategies bring along substantial architectural changes to the original CNNs, they, nevertheless, preserve their original technical integrity and suitability for real space practical applications. In the rest of this section, the proposed framework is analysed from different aspects.

5.6.1 SSP Framework: Advantages and Limitations

As mentioned in section 5.3, this is the first research work that proposes a design framework, which can be holistically applied to CNNs, whose designs are formed from blocks and modules [12], [22], [24]–[26]. To fairly assess its performance, the framework has been applied to the five most widely used CNN models that include almost all types of convolutions and operations, and even a more sophisticated operation like the *shift* [3] instead of spatial 2D convolutions.

A unique feature of the proposed approach, which distinguishes it from the compression methods, introduced in section 5.3.1, is that there is no requirement for the CNNs, to which it is applied, to be pre-trained models. This is because the SSP framework creates from scratch the novel models when applied to them. That brings freedom and flexibility in the model selection, as the design process is not contingent upon pre-trained models.

Furthermore, unlike recent studies [318], [338], which propose methods to remove redundant skip connections only in DenseNet [24], this is the first study that employs the technique of skip connections in other block-based CNNs. Thus, one of the main achievements of this study is making skip connections available to other CNNs with the aim to improve feature reuse and increase accuracy.

The experimental results, presented in Table 5.2 and Table 5.3, which are based on representative block-based CNN models, have explicitly confirmed the size-reduction ability of the SSP framework. Besides that, the proposed method is also very competitive when compared to compression methods. For instance, according to a recent study [344], in which a *pruning* compression method is used, the parameter number of ResNet-50 [22] is reduced from 23.7M to 14.8M preserving the original model accuracy, while the proposed method in this research achieves that with only 7.4M parameters, i.e., 50% less, as seen in Table 5.2. In another compression study, using *tensor decomposition* [345], the original ResNet-20 accuracy is achieved with a x2.3 compression rate, while the proposed method in this research achieves the same with a higher rate of x2.7, as seen in Table 5.3. The SSP framework strongly competes with other compression methods too, i.e., *quantization* and *knowledge distillation* [346]–[348].

On the other hand, contrary to compression methods, there are two main limitations of the approach in its present form: (i) It can only be applied to block-based CNNs. For instance, the GoogLeNet series CNNs [200], [239] which widely adopt multi-path module structure, also known as *fragmentation*, would require a different approach tailored to their architectural specifics; (ii) It includes a lengthy manual fine-tuning process to find the optimum values of the parameters (module number (m), growth rate (g), channel split rate ($\mathcal{E}$), and channel numbers (f)), taking into account the design properties of the selected block-based model and parameter guidelines.

5.6.2 Sparse Skip Connections: A Powerful Novel Design Method

Previously, a method of dense skip connections existed, which was proposed in the DenseNet model. The *sparse skip connections* pattern that was developed as part of this study is a different novel method, which is applicable to any block-based CNN architecture. It was also discovered and confirmed by extensive analysis that module outputs at three steps back, or three-step-away skip connections, prove to form the strongest skip connections pattern.

Other sparse skip connections patterns, e.g., two or more powerful skip connections with different step numbers can also be deployed in CNN models. The reason why only three-step-away skip connections are adopted in the proposed SSP framework is to prevent overgrowth of channel numbers. Direct result of that is that the newly created SSP-enabled models have a lower model size than their ancestors. Moreover, by applying sparse skip connections with a combination of different step numbers, this novel method can also be used for increasing the accuracy in any block-based CNN models.

Additionally, the SSP framework design philosophy allows to selectively apply its constituent strategies. For example, existing CNNs or new ones to be designed from scratch, can only exploit the block-level design strategy to reap the benefits of improving feature reuse. This could lead to the proposed *sparse skip connections* method becoming an innovative emerging trend in CNN custom design on its own.

5.6.3 Effects of Skip Connections

As previously stated, dense skip connections with element-wise concatenation operations result in a higher GPU memory consumption [317], [318], [338]. One interesting side-effect of this drawback is witnessed in the inference time analysis of the models as seen in the last column of Table 5.2. The speed ratio of DenseNet-121&DenseNet-169 between GPU vs. CPU per image is the lowest compared to SSP-DenseNet and all the others in Table 5.2. On the other hand, apart from SSP-DenseNet, the other SSP-enabled models have a lower ratio compared to their own ancestors. This finding indicates that the ratios of GPU vs. CPU speeds of the SSP-enabled models depend on the density of the skip connections. By flagging up this outcome, designers are informed that skip connections could be affecting the inference speed, if special measures are not taken to prevent that.

5.6.4 Metric Analysis

The speed data in the last column of Table 5.2 reveal another interesting finding, i.e., that improvements on the FLOPs performance do not reflect linearly the speed of GPU and CPU platforms, namely speedups differ. To further analyse that, the inference performance of ShuffleNet, ShuffleNet-v2, and their SPP-enabled versions are tested. The four models are used and their hyperparameters are toggled to create models with a similar number of FLOPs. Figure 5.7 illustrates the results for both the GPUs and CPUs. The same implementation methodology as in Table 5.2 is applied.

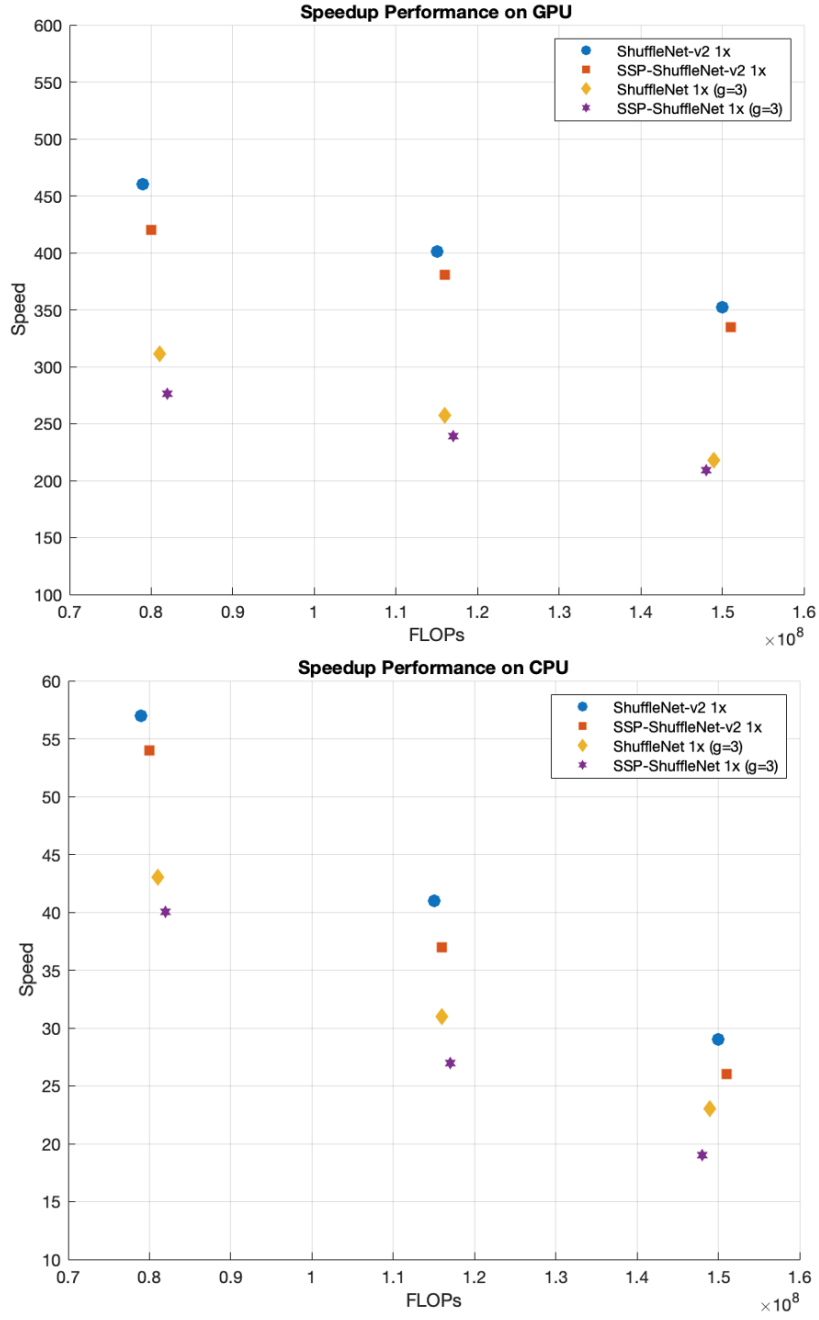


Figure 5.7: Speedup performance comparisons on GPU (batch size=8) and CPU (batch size=1) platforms.

A few more thought-provoking findings could be observed from Figure 5.7. It seems that SSP-enabled models are slower than their ancestors under similar FLOPs. These results are somehow in line with the findings in section 5.6.3 above. Furthermore, CPUs are proportionately more responsive to FLOPs changes. One reason for that is that matrix multiplication is slower on GPUs as found in [349]. These findings show that there is a need of further investigation towards formulation of conceptual design strategies on speedup.

5.7 Generalization Ability

5.7.1 MS COCO Dataset

Generalization ability over various datasets and vision tasks is a prerequisite for a competent network. By now, in sections 5.5.1 and 5.5.2, SSP-enabled models are evaluated on two datasets for image classification tasks leading to several breakthroughs.

Let us now further analyse the SSP framework on the MS COCO 2014 [27] dataset with more than 200k images and 80 categories for domain-general object detection tasks. The Faster R-CNN [148] is selected for the implementation of the object detector due to being frequently used [22], [26], [147]. Two CNN models as the backbone network [29], ResNet-50 [22] and ShuffleNet 1x (g=3) [26], are chosen and replaced with the derived models in this research, SSP-ResNet-50 and SSP-ShuffleNet 1x (g=3). They are pre-trained on ImageNet [21], and then fine-tuned on the MS COCO dataset with 5 epochs for a faster performance validation. The experimental results on 300x resolution, presented in Table 5.4, show that the SSP-enabled models achieve considerably competitive mean average precision (mAP) despite their lower model sizes (see Table 5.2).

Table 5.4: mAP (%) on MS COCO 2014 dataset.

Backbone	mAP.5	mAP.75	mAP [.5, .95]
ResNet-50	51.3	33.6	31.4
SSP-ResNet-50	49.6	31.2	28.9
ShuffleNet 1x (g=3)	29.3	16.8	14.5
SSP-ShuffleNet 1x (g=3)	28.9	17.1	14.2

In Figure 5.8, a detection instance, which includes bounding boxes with the predicted labels and probabilities, on a selected sample image from the MS COCO validation dataset is displayed. This detection instance is attained by Faster RCNN detector, where SSP-ResNet-50 is placed as the backbone network (feature extractor). False detections also exist in this instance as mAP is not sufficiently high (49.6% (ideally expected to be 100%) as seen in Table 5.4), leading to mispredictions (i.e., a suitcase at the right-most is misclassified as a backpack and a trolley at the left-most is misclassified as a bicycle).

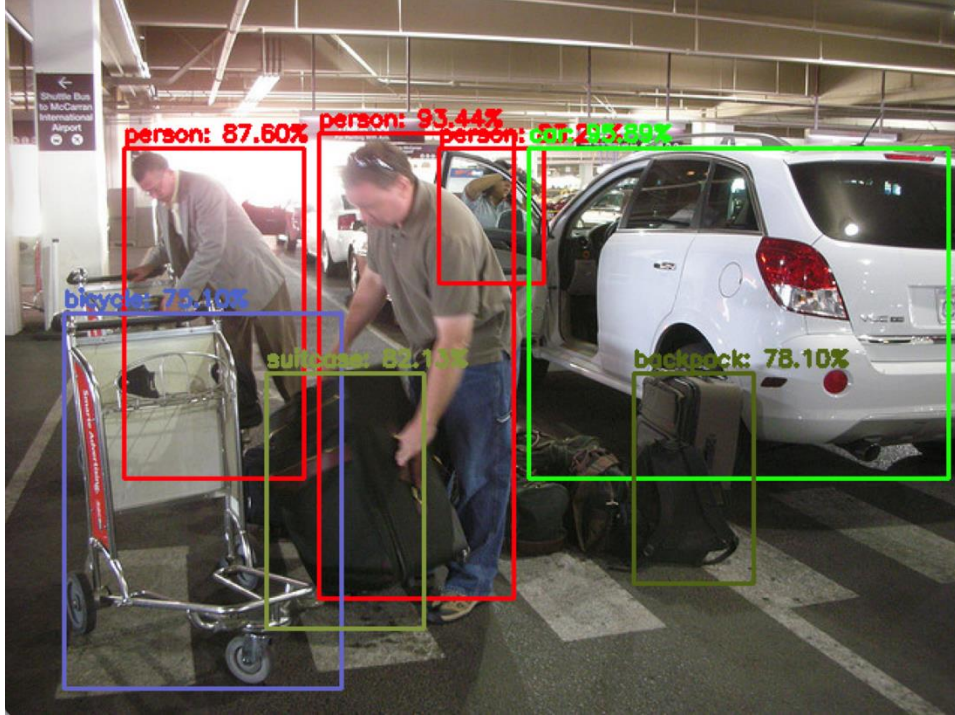


Figure 5.8: A detection instance with Faster RCNN on MS COCO.

5.7.2 KITTI Dataset

In the preceding subsection, the object detection ability of the proposed models in this thesis is evaluated on a domain-general dataset, MS COCO 2014 [27]. Now, the performance of the proposed models is evaluated on a domain-specific dataset, KITTI object detection [28], which is based on autonomous driving scenes with 15k training/test images, comprising 80k labelled objects for 7 different categories. The outcome of this evaluation gains importance to reveal the performance of the proposed models on autonomous driving scenes, which are the fundamental focal point of this research with real-time applications. Table 5.5, which represents the outcome of the KITTI evaluation, proves the ability of the proposed models in autonomous driving scenes as well. Compared to the computationally higher size existing models (ResNet-50 and ShuffleNet 1x (g=3)), SSP-enabled models achieve nearly equal accuracies with much lower model sizes (see Table 5.2).

Table 5.5: mAP (%) on KITTI dataset.

Backbone	mAP.5	mAP.75	mAP [.5, .95]
ResNet-50	75.9	48.5	47.1
SSP-ResNet-50	76.1	47.2	46.6
ShuffleNet 1x (g=3)	43.5	24.7	21.5
SSP-ShuffleNet 1x (g=3)	42.9	23.9	20.9

Similarly to Figure 5.8, Figure 5.9 also illustrates a detection instance, which includes bounding boxes with the predicted labels and probabilities, on a selected sample image from the KITTI validation dataset this time. This detection instance is also attained by Faster RCNN detector, where SSP-ResNet-50 is placed as the backbone network (feature extractor). False detections do not exist at this instance as we have higher mAP at this dataset compared to MS COCO 2014, as seen in Table 5.5. This is primarily because of the small number of object categories in this dataset, 80 (MS COCO) vs. 7 (KITTI).

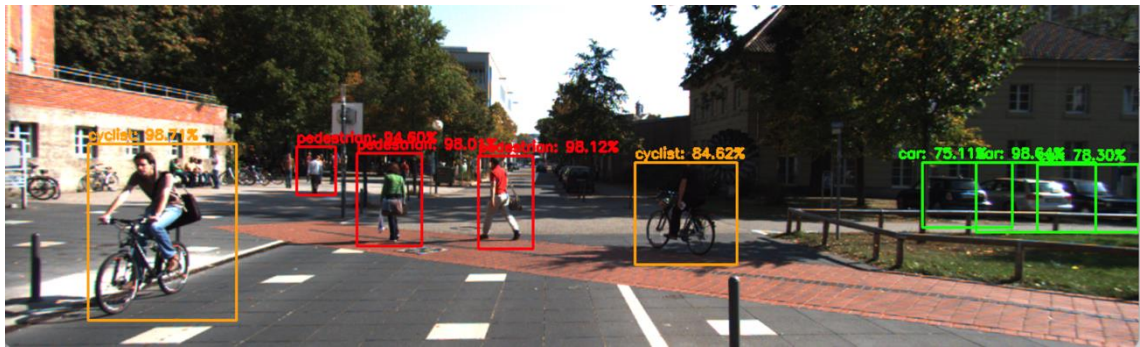


Figure 5.9: A detection instance with Faster RCNN on KITTI.

All these detection performances explicitly signify the emerging capacity of the proposed models for the ADSs as similar accuracies are achieved with the much lower computational load. That is quite crucial for the detection module of the ADS pipeline as allowing FPGA-based deployments (see section 2.2.3).

5.7.3 Real-Time Performance

As a future work, a multi-directional, real-time, FPGA-based, ADS platform research with real road scenes by employing the proposed models in this research will be in place with several other novel approaches. The reason behind this postponement is the timing constraints because of the time-consuming training processes of the proposed models with the limited GPU resources. Even so, to prove the real-time performance of the SSP framework with a preliminary study, a remarkable real-time test is carried out in this section, which also illuminates the strong potential of the planned future work.

By now, as non-real-time (or batch processing) experiments, inference performances of the proposed models have been experienced from different aspects on compute-bound (CPU) and memory-bound (GPU) computation platforms. But now, a real-time image classification task on ImageNet [21] with the proposed models is implemented

to compare with an attractive competition, which is an officially executed experiment by PyTorch [350].

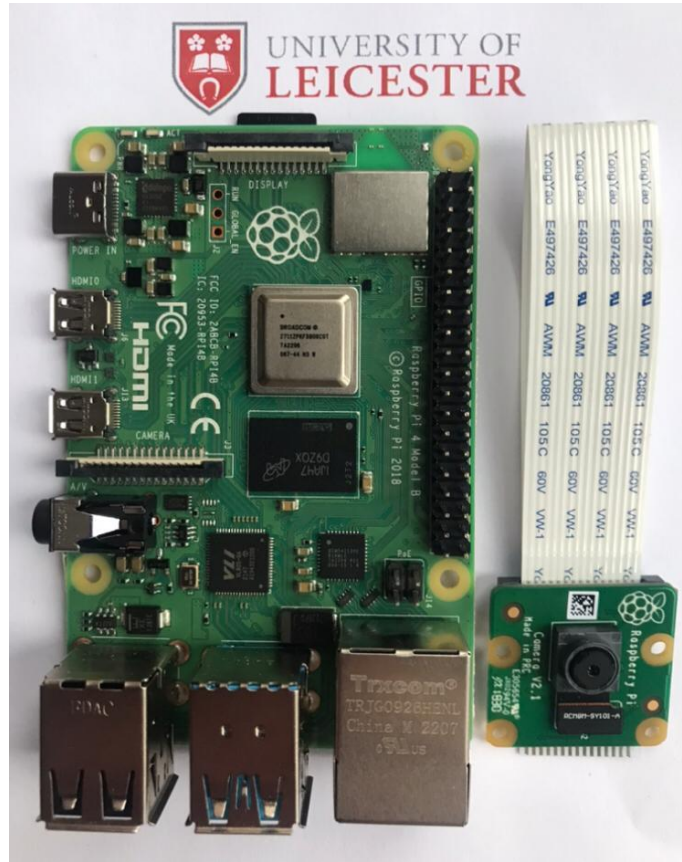


Figure 5.10: Raspberry Pi 4 Model B (4 GB RAM) and Raspberry Pi Camera.

The experiment [350] by PyTorch setups a real-time image classification platform with Raspberry Pi 4 Model 4GB (ARM-based architecture with 4GB RAM) and Raspberry Pi Camera Module, seen in Figure 5.10, as well as the other standard accessories (power supplier, SD card, etc.). In the experiment, the camera module captures the frames from the real-time environment and the deployed image classifier algorithm classifies the captured frames in real-time according to the ImageNet [21] classes. PyTorch deploys ResNet-50 and ShuffleNet 1x (g=3) models and their real-time performances are inserted into first and third lines of Table 5.6 from PyTorch website [350]. In this research, the same setup for this experiment is prepared to compare the real time inference performance the proposed models with the experiment by PyTorch. As seen in the second and fourth lines of Table 5.6, the proposed models (SSP-ResNet-50 and SSP-ShuffleNet 1x (g=3)) in this research achieve higher frame per second (fps) performance with much lower model sizes. Figure 5.11 displays one image classification frame sample each from the

experiment for both SSP-ResNet-50 and SSP-ShuffleNet 1x (g=3) models. As a result, this experiment revalidates the promising performance of the SSP framework, demonstrated in the previous experiments, but this time from the point of view of the critically important real-time response.

Table 5.6: Real-time inference performance (fps) comparison of the proposed models in this research.

Backbone	fps	Total Time (ms/frame)	Model Time (ms/frame)
ResNet-50	4.3	233.9	225.2
SSP-ResNet-50	9.1	109.7	103.5
ShuffleNet 1x (g=3)	24.4	41.1	37.7
SSP-ShuffleNet 1x (g=3)	25.8	38.8	34.1

Moreover, this experiment reveals a significant consequence in terms of the mobile processing aspect. Because 95% of the leading mobile processors are ARM-based devices today [351]. As the Raspberry Pi 4 Model 4GB platform is an ARM-based device, this experiment demonstrates the capability of SSP-enabled models on mobile platforms as well. What is, even more, is that all these experiments hint at the potential success of the planned ADS platform research in advance as a logical outcome.



Figure 5.11: One sample each from real-time image classification experiment: (a) SSP-Resnet-50; (b) SSP-ShuffleNet 1x (g=3).

5.8 Summary

In this chapter, a novel design framework has been introduced, which can be applied to existing CNN models to reduce their model size while retaining similar accuracies. A precondition of the framework is that the selected models should have a block-

based design structure, for example, as in DenseNet, ResNet, ShiftNet, ShuffleNet, and ShuffleNet-v2. The ultimate goal of this research is to facilitate more accurate and lower model size CNN implementations for ADSs, targeting FPGA platforms.

The proposed SSP framework has been based on three carefully designed strategies that are applied at a block, module, and layer level, resulting into unique new capabilities as described below.

The first strategy is motivated by the observation that redundant skip connections cause a higher memory bandwidth on embedded platforms, especially on GPUs. To remedy that, n -step-away skip connections have been introduced in the blocks as a novel sparse skip connectivity pattern. It has been confirmed that three-step-away connections provide the best feature reuse in the blocks. Thus, the novel sparse skip connections prevent a severe rise in channel numbers and keep a satisfactory feature reuse performance on the networks.

In the second strategy, a novel SSP module has been built that incorporates module designs of targeted CNN models by placing their underlying mapping in a dedicated holder space. Thus, novel SSP-enabled models are created, and the performance of the targeted CNN models are enhanced through the application of the framework. Another important feature of the SSP module is the novel proportional channel split operation, which allows to control the channel proportion for the underlying and identity mapping. This has been achieved by means of a special hyperparameter $\mathcal{E}$, which is also used in the SSP framework for managing a desired trade-off between accuracy and model size.

In the third strategy, the idea of degree of parallelism from ShuffleNet-v2 has been adopted and incorporated in the SSP framework. As a result, the channel numbers in the layers are made equal, which decreases memory access cost in the models and yields a better inference time performance.

Comprehensive experimental results have validated the effectiveness of the framework. Compared to their ancestors the SSP-enabled models achieve a substantial reduction of the computational complexity by a significant margin. The models have been evaluated on image classification using the ImageNet [21] and CIFAR-10&CIFAR-100 datasets, as well as on object detection with the MS COCO [27] and KITTI [28] dataset. They surpass the original DenseNet, ResNet, and ShiftNet with $\sim 3\times$ lower parameter numbers and $\sim 2\times$ lower FLOPs while maintaining equivalent accuracies. In addition, SSP-enabled models have reached absolute 2% lower Top-1 error on ShuffleNet with

quite similar computational loads and absolute 6% lower Top-1 error on ShuffleNet-v2 with a slightly higher computational budget.

Speedup performance has been also examined on both compute-bound (CPU) and memory-bound (GPU) platforms, showing promising results. With respect to object detection performance, the framework has achieved a similar accuracy with a lower computational budget in the Faster R-CNN [148] detector compared to ResNet.

Finally, the real-time performance of the framework has been evaluated at an embedded platform and promising results have been even obtained at even these experiments.

The key strength of this research has been in bringing novel qualities to CNN design by making use of existing models. It can be expected that the proposed approach would stimulate future work on platform-aware CNN custom design for advancing many areas, such as autonomous driving, robotics, earth observation satellites, to name a few.

CHAPTER 6 CONCLUSIONS AND FUTURE WORK

This chapter concludes the thesis by summarising the research in section 6.1, declaring the novelty claims in section 6.2, and introducing future work in section 6.3.

6.1 Research Summary

Many crucial improvements have been reported in the development of vehicular sensors for performing different complex tasks including object detection, localization, tracking, and activity recognition for numerous applications. By means of these advancements, the sensing and computing processes of Autonomous Driving Systems (ADSs) have been improved. Despite the breakthroughs, ADSs still demand further attention of industry and academia due to their stringent real-time limitations, also especially because of that, in the near future, it is expected that ADSs with full automation may need to process the necessary information of ~ 1 GB/second. From this viewpoint, the focus of this thesis was first set out to investigate and analyse the present work to determine the current limitations in the field, and then secondly to propose a novel solution to alleviate them.

In Chapter 2, the following research contributions are detailed: (i) the first attempt to establish a novel type of parameter number calculation of a generic CNN, AlexNet [21], with detailed explanations in its entirety; (ii) the first study to practically demonstrate the CNN&ADS-related datasets by classifying them from a novel perspective: domain-general or domain-specific; (iii) the first study making trade-off analyses from a human vs machine and accuracy vs complexity for ADSs as well as investigating the current ADS architectures and industry practices to complete the meaningful data collection from both academy and industry in a synergetic way.

In Chapter 3 and 4, the main contributions are: (i) considerably extending the knowledge of the latest CNN models for image classification and object detection by covering the most up-to-date developments; and (ii) a novel timing analysis aimed at assessing the impact of commonly used convolution types on CNN performance has been measured running a reference CNN model on a desktop computer and two powerful GPUs. This novel experiment has established a new insight into the performance of each convolution type in terms of training time, inference time, and layer-level decomposition.

The contributions of Chapters 2-4 have helped to specify promising directions aimed at solving the problems of automakers in achieving real-time implementations for a novel CNN design. Prior to this research, there were no such large-scale data or analyses in the field. In particular, it has been confirmed that CNN models that feature low model sizes and achieve satisfactory accuracy have a promising future, which is addressed with a novel solution in Chapter 5. Consequently, based on the work in Chapters 2-4, a strong message could be conveyed to stakeholders that with the necessary real-time hardware capabilities and successful resolution of design constraints, CNNs would enable safer and more efficient self-driving cars and a rise in their popularity.

In Chapter 5, a novel method has been developed to lighten the computation complexity of the several popular block-based CNN models, which are commonly placed as backbone networks for feature extraction on the frequently used object detectors. For image classification tasks on ImageNet and CIFAR-10&100, the proposed novel design SSP framework, has been applied to each block of the selected block-based models and there have been obtained notable breakthroughs in reducing their model size and inference time performances while retaining their accuracies. Furthermore, similar results have been also seen on the domain-general MS COCO and domain-specific KITTI datasets for object detection. In addition to these experiments, the real-time performance of the framework has been also tested on an embedded platform to see if coherent results are provided and the results have proven similar achievements to the previous experiments. In conclusion, experimentally shown results in this chapter have enabled novel performance results in the field (see Table 5.2, Table 5.3, Table 5.4, Table 5.5, and Table 5.6). It is shown that as a result of the application of the SSP framework, the SSP-enabled CNNs for state-of-the-art CNNs, SSP-DenseNet, SSP-ResNet, and SSP-ShiftNet, out-perform their ancestors featuring $\sim 3\times$ lower number of parameters, $\sim 2\times$ lower floating-point operations (FLOPs), and $\sim 2\times$ faster inference performance, while attaining the same level of accuracies. In this way the ability of the proposed framework to achieve the targeted aims has been proven, demonstrating its performance for both nonreal-time and real-time scenarios.

In addition to the novel performance results of SSP framework in the explained experiments above, it also comes with several important contributions to the CNN foreground by means of its novel approach and three design strategies, which are: (i) the first research work that proposes a design framework, which can be holistically applied to

CNNs, whose designs are formed from blocks and modules [12], [22], [24]–[26]; (ii) unlike recent studies [318], [338], which propose methods to remove redundant skip connections only in DenseNet [24], this is the first approach that employs the technique of skip connections in other block-based CNNs. Thus, one of the main achievements of this study is making skip connections available to other CNNs with the aim to improve feature reuse and increase accuracy; (iii) previously, a method of dense skip connections existed, which was proposed in the DenseNet [24] model. The sparse skip connections pattern that was developed as part of this study is a totally novel method, which is applicable to any block-based CNN architecture. It was also discovered and confirmed by extensive analysis that module outputs at three steps back, or three-step-away skip connections, prove to form the strongest skip connections pattern; (iv) in contrast to the half-and-half application of channel split operation in [25], this research advances it further, creating a novel operation referred to as *proportional channel split*, which is a more efficient one. Taken together, these contributions suggest a significant role of the SSP framework in enabling design of smaller-size CNN models by keeping similar accuracies.

6.2 Novelty Claims

The most significant novelty claims, arising from this research, which extend the state-of-the-art, are declared below:

- This research brings a new insight into the *relative and comparative performance of convolution and operation types* in terms of their inference, accuracy, and layer-level performance by means of a novel timing analysis. This contribution significantly extends the current state-of-the-art by revealing for the first time the relative and comparative performances of the convolution and operation types, which is very significant in CNN design processes.
- This research advances further the half-and-half application of the channel split operation, proposed in recent works, creating a novel more efficient operation, referred to as *proportional channel split*, which expands the scope of the channel split operation to other applications.
- One of the main novel contributions of this research is that for the first-time dense skip connections are made available to other block-based CNNs by means of a novel

connectivity pattern, *sparse skip connections*, which is very powerful on feature reuse and improving accuracy. Moreover, this establishes the ground for other approaches to be able to employ skip connections in existing or forthcoming CNNs.

- This is the first research work that proposes a design framework for lightweight high-performance CNN models, referred to as *SSP framework*, which can be holistically applied to CNNs, whose designs are formed from blocks and modules. This approach differs principally from the existing compression techniques and extends the current state-of-the-art with a new design approach to reduce the model size of existing CNNs, preserving their performance. Its capability has been verified with the means of a novel comprehensive numerical experiment.

The CNN design community will also benefit from the following novel contributions, resulting from the research:

- As a result of this research a novel type of guidelines have been established for the calculation of the parameter number of a generic CNN that clearly show the effects of each convolutional layer in CNN model, which did not exist previously.
- The research has also extended considerably the current knowledge on image classification, object detection, and ADSs by covering in the most comprehensive way the up-to-date developments on the latest CNNs, as well as producing trade-off analyses from different perspectives. Prior to this research, there were no such large-scale data analyses in the field.

6.3 Future Work

As the limitations of a study naturally may point to the potential future works, which remedy the deficiencies, the future works are determined by these limitations in this section, which are: (i) as explained in section 5.6, SSP framework can only be applied to block-based CNNs as a remarkable limitation. Therefore, future work should be undertaken to explore a slightly different approach tailored to the architectural specifics of other CNNs; (ii) another limitation of applying SSP framework is that it includes a lengthy manual fine-tuning process to find the optimum values of the parameters (module number (m), growth rate (g), channel split rate ($\mathcal{E}$), and channel numbers (f)), taking into account the design properties of the selected block-based model and parameter guidelines. Further modelling work will have to be conducted in order to determine the underlying causes of this drawback; (iii) the current design philosophy of the SSP framework obliges

researchers to apply it to only block-based CNN models. However, it also allows to selectively apply its constituent strategies. For instance, existing CNNs or new ones to be designed from scratch can only exploit the block-level design strategy (*sparse skip connections*) to reap the benefits of improving feature reuse. Considerably more work could lead to the proposed *sparse skip connections* method becoming an innovative emerging trend in CNN custom design on its own; (iv) as a major limitation, the current research was unable to evaluate the novel SSP-enabled models on a multi-directional, real-time, FPGA-based, ADS platform research with real road scenes. That's why the initial plan for future work following this research is to analyse the real-time performance of the proposed networks in depth with several other novel approaches. Thus, this also could raise new research directions.

Apart from the above, there also exist several prominent future work directions for object detection and ADSs from the managed investigations in Chapter 2-4,, which are: (i) utilizing one-stage and two-stage detectors by combining their superior sides is a big challenge enabling higher accuracy and lower latency together in the detectors; (ii) due to the necessity of a large amount of data in deep learning, as a non-CNN based method, GAN-based detectors [352] are expected to become a promising and ever-increasing area; (iii) connected systems promise a huge potential in improving the level of autonomy, but they require communication among road users, which is a complex and challenging problem that is yet to be addressed; (iv) human-machine interaction is an attractive area, required in designs at automation level 3 or upper levels. However, mutual understanding between cars and human requires further research work; (v) establishing ADSs requirements for real-time applications is another crucial area. As discussed in Chapter 2, despite the fact that human behaviours are not easy to be evaluated, investigations of parallels between humans and machines have already been undertaken towards formulating commonly held ADSs regulations in real-time applications [20].

In conclusion, hope that the proposed CNN models with advantages and limitations could bring profits when placed into the object detection modules of the ADSs and thus, they can help rise in their popularity with safety and efficiency.

Appendix: CNN Design and Training Exploration

Chapters 3-4 have highlighted that it is by innovating and perfecting the existing design approaches and techniques that researchers have been advancing so effectively the CNN state-of-the-art. The investigation of the literature reveals that the success of CNN models depends on the design strategies in regard to the CNN micro/macro architecture parameters as well as optimization algorithms, activation functions, compression techniques, learning types, normalization [353], regularization [354], loss functions [355], data augmentation [356], RoI feature extractors [148], Region proposal algorithms [222], etc. Therefore, this chapter reviews some of the most important areas of design space exploration for classification and object detection CNNs.

A.1 Optimization

Optimization algorithms have played a significant role in the training of machine learning models in recent years. In particular, iterative or coordinate descent methods have been instrumental in increasing model performance. However, the optimization field has gradually faced more challenges with the growth in model complexity and data size, as reported in [171]. A summary of the algorithms, employed in CNNs, is presented below, dividing them into three main categories: first-order, high-order, and heuristic derivative-free.

A.1.1 First-Order Algorithms

Gradient descent algorithms are the most used algorithms in first-order optimization methods. They depend on the first-order derivatives of the loss functions, represented in Jacobian matrices. In its simplest form, the weight matrix θ is defined by

$$\theta = \theta - \mu \cdot \nabla J(\theta) \quad (\text{A-1})$$

where μ is the learning rate and $\nabla J(\theta)$ is the Jacobian matrix of the cost function $J(\theta)$.

Three gradient descent variants are used depending on the number of examples (n) from a dataset, which the algorithm uses to compute the gradient objective in each iteration. The first variant, Batch gradient descent, uses all n examples in every iteration and is calculated with

$$\theta = \theta - \mu \cdot \nabla_{\theta} J(\theta; x^{(i:i+n)}; y^{(i:i+n)}), \quad (\text{A-2})$$

where x represents the input features of the i^{th} training example and y is the real value for the i^{th} example. Its drawback is a possible slowness, as it calculates the gradients for the whole dataset.

The second one is Stochastic gradient descent (SGD) that uses only one example in each iteration. Its limitation is that SGD performs updates frequently with a high variance that produces intense fluctuations in the objective function. Its function is

$$\theta = \theta - \mu \cdot \nabla_{\theta} J(\theta; x^{(i)}; y^{(i)}) \quad (\text{A-3})$$

The last variation is Mini-batch gradient descent that uses $z=n/x$ examples, where x is selected according to the application, in each iteration. The method, which is defined by the expression below, enhances performance, as it reduces the variance effects and computational load.

$$\theta = \theta - \mu \cdot \nabla_{\theta} J(\theta; x^{(i)}; y^{(i)}) \quad (\text{A-4})$$

Among these three algorithms, the SGD method [357], [358] and its variants have been widely used in the last years, as a representative of first-order optimization methods. It is crucial to state that the SGD term can be interchangeably used when mini-batches are used as well. It is also quite significant to pay attention to the application scope and characteristics of these methods while selecting them for application in DNNs.

There is a long list of gradient descent algorithms that fall in the category of first-order optimization methods. Methods have pros and cons towards each other, and they are:

- Adaptive moment estimation (Adam) [195]
- Nesterov Accelerated Gradient Descent (NAG) [359]
- AdaGrad [360]
- AdaDelta [361]
- RMSProp [362]
- Adaptive moment estimation (Adam) [195]

- Stochastic Average Gradient (SAG) [363]
- Stochastic Variance Reduction Gradient Since (SVRG) [364].

A.1.2 High-Order Algorithms with Hessian Matrix

Compared to the first-order methods, the high-order optimization methods [365]–[367] converge most correctly to the local or global minimum as the curvature information of these methods is more powerful. In spite of the attractive features, they face multiple challenges. Second-order algorithms present problems with the computation and storage of the inverse matrices of the Hessian matrices, which are used in the high-order optimization to hold the second or higher-order derivatives, in each iteration of the training process.

Newton’s methods have been applied to overcome the problem [368], [369]. Following that the Stochastic quasi-Newton method and its improved versions [370]–[372] has been proposed, to allow the use of high-order methods on large-scale datasets. Several other methods have been proposed too, which are:

- Conjugate Gradient [373]
- Newton’s Method [374]
- Quasi-Newton Method [375]
- Stochastic Quasi-Newton Method [376]
- Hessian Free Method [368]
- Sub-sampled Hessian Free Method [377]
- Natural Gradient [378].

A.1.3 Heuristic Derivative-Free Algorithms

The heuristic derivative-free algorithms in [372], [379] have been proposed to address the case when the derivative information of the target functions may not exist or be hard to calculate. They are based on two root ideas. While the first one is a heuristic investigation with empirical rules [380], the second one is trying to fit the function with samples [381].

A.2 CNN Micro/Macro Architecture

Microarchitecture parameters of CNNs are related to layer-based adjustments, such as filter dimensions, pooling operation types, etc. There is a variety of those parameters among CNN architectures. For instance, while VGGNet [178] employs 3x3 filters, Inception-v1 [37] makes use of different sizes of filters, such as 1x1, 3x3, and 5x5. In addition, as CNNs tend to go deeper, manual adjustments of kernel types and input/output dimensions become impractical in individual layers. To address the problem, higher-level pre-designed blocks or modules, which involve different sizes and types of kernels, bypass connections, activation functions, etc., have been proposed to lighten designers' effort. An example of modules is the inception modules, which were used in the design of the Inception-v1 CNN, also known as GoogLeNet, where many of them are combined with ad-hoc layers to construct a complete network.

Whereas the properties of individual layers and modules represent microarchitecture, CNN macro architecture concerns the system-level organization in its entirety or the end-to-end algorithm pipeline. The selection of depth, i.e., the number of layers or modules, and the choice of connection types across multiple layers or blocks can be listed as the two primary areas of the CNN macro architecture. In VGGNet, increasing the number of layers and modules gives a better accuracy performance at the system level. An example of connection types between the layers is the FCLs that are visualised in Figure 2.1 for AlexNet [21]. In addition, several macro architecture strategies were proposed in ResNet-50 [22] that advanced the state-of-the-art in its publication year. Several other CNN designs [24], [148], and [105], which are introduced before, have gained success with innovative macro architecture-based strategies as well.

A.3 Compression

Although Deep Neural Networks (DNNs) have obtained great success in various computer vision tasks, many existing designs are computationally intensive and require a large amount of memory. This prevents their implementation on hardware with limited memory like FPGAs and uses in applications that run on strict latency constraints. For these reasons, compression of a pre-trained algorithm into a lightweight one without ceding from the model performance is a frequent technique to enable real-time deployments of the models. Four approaches exist, which are pruning, tensor decomposition, quantization, and knowledge distillation.

A.3.1 Pruning

Pruning reduces the redundant parameters, which have no effect on the performance. Thus, weight matrices are turned into sparse ones. Parameter pruning is quite robust over various settings in the layer and can support training from scratch or pre-trained models. Additionally, it helps with the over-fitting problem. More information on pruning could be found in [319], [320], [382].

A.3.2 Quantization

Quantization is to adopt low-bit number representations instead of floating-point ones for every weight parameter. Currently, the main research tendency on quantization is model binarization. For instance, Binarized neural networks [383] quantize the weight values to 1-bit representations based on BinnaryConnect as well as changing the activation values to 1 bit. Thus, it reduces memory usage and simplifies many multiply-add operations into bitwise operations XNOR-Count. Another example is XNOR-Net [327], which achieves 32x compression, providing a 52x speed rise. Several other techniques [325], [384]–[388] have been proposed as well.

A.3.3 Knowledge Distillation

The idea behind the knowledge distillation approach is related to a type of migration learning, in which a complex architecture, called teacher model, is used to train a compact algorithm, called student model, by means of distilling knowledge from the teacher. The knowledge distillation approach is used in both classification and detection algorithms. However, the technique is quite sensitive to the range of applications and there is no pre-trained model option. An example of distillation is EfficientNet-L2 [183], in which it is applied quite effectively. Improved versions of the technique can be found in [328]–[330].

A.3.4 Tensor Decomposition

The aim of tensor decomposition is to exploit channel or spatial-wise redundancies of convolution filters and to seek the lower rank approximations of them. Further details are reported in [322], [323].

A.4 Machine Learning Methods

Here it is briefly touched upon the current machine learning methods, as choosing the right machine learning approach in ADSs pipelines may allow improving their performance [183], [188].

Based upon the addressed problem and data type, machine learning approaches can be broadly divided into three groups: primary approaches, hybrid approaches, and other types. Primary approaches comprise supervised [389], unsupervised [390], and reinforcement learning [391]. Hybrid approaches, the use of which has become quite prevalent in the last few years [184], could be subdivided into three groups: semi-supervised learning [184], self-supervised learning [392], and self-taught learning [393].

Numerous other types of machine learning methods have also been developed, which are:

- active [394];
- online [150];
- transfer [395];
- federated [396];
- adversarial [397];
- multi-task [398];
- multi-modal [399].

A.5 Summary

In this chapter, more information on optimization algorithms, CNN macro/micro-architecture parameters, compression techniques, and ML methods has been provided. For interested readers, additional references are also inserted in the explanations.

References

- [1] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *J. Physiol.*, vol. 160, no. 1, pp. 106–154, 1962, doi: 10.1113/jphysiol.1962.sp006837.
- [2] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biol. Cybern.*, vol. 36, no. 4, pp. 193–202, 1980, doi: 10.1007/BF00344251.
- [3] Y. LeCun *et al.*, "Handwritten Digit Recognition with a Back-Propagation Network," *Dermatologic Surg.*, vol. 39, no. 1, p. 149, 1998, doi: 10.1111/dsu.12130.
- [4] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-Based Learning Applied to Document Recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998, [Online]. Available: <http://ieeexplore.ieee.org/document/726791/#full-text-section>.
- [5] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *International 1989 Joint Conference on Neural Networks*, pp. 593–605.
- [6] D. Steinkraus, I. Buck, and P. Y. Simard, "Using GPUs for machine learning algorithms," in *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*, vol. 2, pp. 1115–1120, doi: 10.1109/ICDAR.2005.251.
- [7] K. Chellapilla, S. Puri, and P. Simard, "High Performance Convolutional Neural Networks for Document Processing," [Online]. Available: <https://hal.inria.fr/inria-00112631>.
- [8] G. E. Hinton, S. Osindero, and Y. Teh, "A Fast Learning Algorithm for Deep Belief Nets," *Neural Comput.*, vol. 18, no. 17, pp. 1527–1554, 2006, doi: 10.1162/neco.2006.18.7.1527.
- [9] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," *Adv. Neural Inf. Process. Syst.*, vol. 19, no. 1, pp. 153–160, 2007, doi: 10.7551/mitpress/7503.003.0024.
- [10] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," *Artif. Intell. Rev.*, vol. 53, no. 8, pp. 5455–5516, 2020, doi: 10.1007/s10462-020-09825-6.
- [11] Y. Xiao *et al.*, *A review of object detection based on deep learning*, vol. 79, no.

- 33–34. Springer, 2020.
- [12] B. Wu *et al.*, “Shift: A Zero FLOP, Zero Parameter Alternative to Spatial Convolutions,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2018, pp. 9127–91355, doi: 10.1109/CVPR.2018.00951.
 - [13] C. Badue *et al.*, “Self-driving cars: A survey,” *Expert Syst. Appl.*, vol. 165, p. 113816, Mar. 2021, doi: 10.1016/J.ESWA.2020.113816.
 - [14] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, “A Survey of Autonomous Driving: Common Practices and Emerging Technologies,” *IEEE Access*, vol. 8, pp. 58443–58469, 2020, doi: 10.1109/ACCESS.2020.2983149.
 - [15] Intel, “Technology and Computing Requirements for Self-Driving Cars,” *Intel Corporation*, 2014. <https://www.intel.com> (accessed Jul. 15, 2021).
 - [16] Velodyne Lidar, “HDL-64E Durable Surround Lidar Sensor,” 2021. <https://velodynelidar.com/products/hdl-64e/> (accessed Jul. 12, 2021).
 - [17] Forbes, “Why Is Tesla’s Full Self-Driving Only Level 2 Autonomous?,” 2023. <https://www.forbes.com/sites/jamesmorris/2021/03/13/why-is-teslas-full-self-driving-only-level-2-autonomous/?sh=4d2b67ce6a32>.
 - [18] Mobileye, “True Redundancy,” 2021. <https://www.mobileye.com/true-redundancy/> (accessed Jul. 13, 2021).
 - [19] Mobileye, “Understanding L2+ in Five Questions,” 2023. <https://www.mobileye.com/blog/understanding-l2-in-five-questions/>.
 - [20] S.-C. Lin, Y. Zhang, C.-H. Hsu, M. S. M. E. Haque, L. Tang, and J. Mars, “The architectural implications of autonomous driving: Constraints and acceleration,” in *ACM SINGPLAN Notices*, 2018, vol. 53, no. 2, pp. 751–766, doi: 10.1145/3173162.3173191.
 - [21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” *Commun. ACM*, vol. 60, no. 6, pp. 84–90, 2012, doi: DOI:10.1145/3065386.
 - [22] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” in *Computer Vision and Pattern Recognition (CVPR)*, 2016, vol. 19, no. 2, pp. 770–778, doi: 10.1016/0141-0229(95)00188-3.
 - [23] A. Krizhevsky, “Learning Multiple Layers of Features from Tiny Images,” 2009. [Online]. Available: <http://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf>.

- [24] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2261–2269, doi: 10.1109/CVPR.2017.243.
- [25] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, “ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design,” pp. 122–138, Jul. 2018, Accessed: Aug. 23, 2021. [Online]. Available: <https://arxiv.org/abs/1807.11164v1>.
- [26] X. Zhang, X. Zhou, M. Lin, and J. Sun, “ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices,” in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6848–6856, doi: 10.1109/CVPR.2018.00716.
- [27] T. Y. Lin *et al.*, “Microsoft COCO: Common objects in context,” in *European Conference on Computer Vision (ECCV)*, 2014, vol. 8693, no. 5, pp. 740–755, doi: 10.1007/978-3-319-10602-1_48.
- [28] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for autonomous driving? The KITTI Vision Benchmark Suite,” in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3354–3361, doi: 10.1109/CVPR.2012.6248074.
- [29] T. Turay and T. Vladimirova, “Toward Performing Image Classification and Object Detection With Convolutional Neural Networks in Autonomous Driving Systems: A Survey,” *IEEE Access*, vol. 10, pp. 14076–14119, 2022, doi: 10.1109/ACCESS.2022.3147495.
- [30] T. Turay and T. Vladimirova, “SSP Framework: A New Approach to Designing Lightweight Convolutional Neural Networks,” *IEEE Access*, vol. 10, pp. 102292–102307, 2022, doi: 10.1109/ACCESS.2022.3208924.
- [31] M. Buehler, K. Iagnemma, and S. Singh, *The 2005 DARPA Grand Challenge: The Great Robot Race*, 1st ed. Springer Publishing Company, Incorporated, 2007.
- [32] M. Buehler, K. Iagnemma, and S. Singh, *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic*, 1st ed. Springer Publishing Company, Incorporated, 2009.
- [33] R. McAllister *et al.*, “Concrete problems for autonomous vehicle safety: Advantages of Bayesian deep learning,” in *IJCAI International Joint Conference on Artificial Intelligence*, 2017, vol. 0, pp. 4745–4753, doi: 10.24963/ijcai.2017/661.
- [34] S. Hecker, D. Dai, and L. Van Gool, “End-to-End Learning of Driving Models with Surround-View Cameras and Route Planners,” *Lect. Notes Comput. Sci.*, vol.

- 11211 LNCS, pp. 449–468, Mar. 2018, Accessed: Oct. 18, 2021. [Online]. Available: <https://arxiv.org/abs/1803.10158v2>.
- [35] M. A. Khan *et al.*, “Level-5 Autonomous Driving—Are We There Yet? A Review of Research Literature,” *ACM Comput. Surv.*, vol. 55, no. 2, Jan. 2022, doi: 10.1145/3485767.
 - [36] D. Pedamonti, “Comparison of non-linear activation functions for deep neural networks on MNIST classification task,” Apr. 2018, doi: 10.48550/arxiv.1804.02763.
 - [37] C. Szegedy *et al.*, “Going deeper with convolutions,” in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 7, no. 12, pp. 1–9, doi: 10.1089/pop.2014.0089.
 - [38] R. Duncan, “Individual Choice Behavior: A Theoretical Analysis,” *J. R. Stat. Soc.*, vol. 123, no. 4, pp. 486–488, 1960, doi: 10.2307/2343282.
 - [39] E. Shelhamer, J. Long, and T. Darrell, “Fully Convolutional Networks for Semantic Segmentation,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 4, pp. 640–651, 2017, doi: 10.1109/TPAMI.2016.2572683.
 - [40] O. Russakovsky *et al.*, “ImageNet Large Scale Visual Recognition Challenge,” *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, 2015, doi: 10.1007/s11263-015-0816-y.
 - [41] S. Rajpal, N. Lakhyani, A. K. Singh, R. Kohli, and N. Kumar, “Using handpicked features in conjunction with ResNet-50 for improved detection of COVID-19 from chest X-ray images,” *Chaos, solitons and fractals*, vol. 145, p. 110749, 2021.
 - [42] M. O. Ramkumar, S. Sarah Catharin, V. Ramachandran, and A. Sakthikumar, “Cercospora Identification in Spinach Leaves Through Resnet-50 Based Image Processing,” in *Journal of physics. Conference series*, 2021, vol. 1717, no. 1.
 - [43] T. M. Hoang, S. H. Nam, and K. R. Park, “Enhanced detection and recognition of road markings based on adaptive region of interest and deep learning,” *IEEE Access*, vol. 7, pp. 109817–109832, 2019, doi: 10.1109/ACCESS.2019.2933598.
 - [44] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, “The pascal visual object classes (VOC) challenge,” *Int. J. Comput. Vis.*, vol. 88, no. 2, pp. 303–338, 2009, doi: 10.1007/s11263-009-0275-4.
 - [45] H. Yin and C. Berger, “When to use what data set for your self-driving car algorithm: An overview of publicly available driving datasets,” in *IEEE Conference on Intelligent Transportation Systems (ITSC)*, 2018, vol. 2018-March,

- pp. 1–8, doi: 10.1109/ITSC.2017.8317828.
- [46] F. Yu *et al.*, “BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2020, pp. 2633–2642, doi: 10.1109/CVPR42600.2020.00271.
 - [47] M. Cordts *et al.*, “The Cityscapes Dataset for Semantic Urban Scene Understanding,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Apr. 2016, vol. 2016-Decem, pp. 3213–3223, Accessed: Oct. 06, 2021. [Online]. Available: <https://arxiv.org/abs/1604.01685v2>.
 - [48] J. Geyer *et al.*, “A2D2: Audi Autonomous Driving Dataset,” 2020, Accessed: Oct. 06, 2021. [Online]. Available: <http://www.a2d2.audi>.
 - [49] H. Caesar *et al.*, “nuScenes: A multimodal dataset for autonomous driving,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 11618–11628, Mar. 2019, Accessed: Oct. 06, 2021. [Online]. Available: <https://arxiv.org/abs/1903.11027v5>.
 - [50] X. Huang, P. Wang, X. Cheng, D. Zhou, Q. Geng, and R. Yang, “The ApolloScape Open Dataset for Autonomous Driving and its Application,” *IEEE Trans. Pattern Anal. Mach. Intell.*, Accessed: Oct. 06, 2021. [Online]. Available: <https://arxiv.org/pdf/1803.06184.pdf>.
 - [51] P. Dollár, C. Wojek, B. Schiele, and P. Perona, “Pedestrian Detection: A Benchmark,” Accessed: Oct. 19, 2021. [Online]. Available: www.vision.caltech.edu/Image.
 - [52] J.-L. Déziel *et al.*, “PixSet : An Opportunity for 3D Computer Vision to Go Beyond Point Clouds With a Full-Waveform LiDAR Dataset,” Feb. 2021, Accessed: Oct. 07, 2021. [Online]. Available: <https://arxiv.org/abs/2102.12010v2>.
 - [53] W. Maddern, G. Pascoe, C. Linegar, and P. Newman, “1 Year , 1000km : The Oxford RobotCar Dataset,” *Int. J. Robot. Res.*, vol. 36, no. 1, pp. 1–15, 2017, doi: 10.1177/0278364916679498.
 - [54] P. Sun *et al.*, “Scalability in Perception for Autonomous Driving: Waymo Open Dataset,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Dec. 2019, pp. 2443–2451, Accessed: Oct. 19, 2021. [Online]. Available: <https://arxiv.org/abs/1912.04838v7>.
 - [55] M. Braun, S. Krebs, F. Flohr, and D. M. Gavrila, “The EuroCity Persons Dataset:

A Novel Benchmark for Object Detection.”

- [56] Udacity, “Udacity dataset.” <https://github.com/udacity/self-driving-car/tree/master/datasets> (accessed Dec. 23, 2021).
- [57] S. Wang *et al.*, “TorontoCity: Seeing the World with a Million Eyes,” *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2017-Octob, pp. 3028–3036, Dec. 2016, Accessed: Oct. 07, 2021. [Online]. Available: <https://arxiv.org/abs/1612.00423v1>.
- [58] K. Takeda, J. H. L. Hansen, P. Boyraz, L. Malta, C. Miyajima, and H. Abut, “International Large-Scale Vehicle Corpora for Research on Driver Behavior on the Road,” *IEEE Trans. Intell. Transp. Syst.*, vol. 12, no. 4, pp. 1609–1623, 2011, doi: 10.1109/TITS.2011.2167680.
- [59] M. Benmimoun, A. Pütz, A. Zlocki, and L. Eckstein, “euroFOT: Field Operational Test and Impact Assessment of Advanced Driver Assistance Systems: Final Results,” vol. 197, 2013, pp. 537–547.
- [60] C. R. Harris *et al.*, “Array programming with NumPy,” *Nature*, vol. 585, no. 7825, pp. 357–362, 2020, doi: 10.1038/s41586-020-2649-2.
- [61] W. McKinney, “Data structures for statistical computing in python,” in *Proceedings of the 9th Python in Science Conference*, 2015, pp. 56–61, doi: 10.25080/Majora-92bf1922-00a.
- [62] W. McKinney and PyData Development Team, “pandas: powerful Python data analysis toolkit,” 2019. [Online]. Available: <https://pandas.pydata.org/docs/pandas.pdf>.
- [63] J. D. Hunter, “Matplotlib: A 2D graphics environment,” *Comput. Sci. Eng.*, vol. 9, no. 3, pp. 90–95, 2007, doi: 10.1109/MCSE.2007.55.
- [64] M. L. Waskom, “Seaborn: Statistical data visualization,” *J. Open Source Softw.*, vol. 6, no. 60, p. 3021, 2021, doi: 10.21105/joss.03021.
- [65] Plotly Technologies Inc., “Collaborative data science,” *Plotly Technol. Inc.*, 2015, Accessed: Jul. 15, 2021. [Online]. Available: <https://plot.ly>.
- [66] M. Abadi *et al.*, “TensorFlow: A System for Large-Scale Machine Learning,” in *12th USENIX Symposium on Operating Systems Design and Implementation*, 1983, vol. 101, pp. 582–598, doi: 10.1016/0076-6879(83)01039-3.
- [67] T. T. D. The Theano Development Team *et al.*, “Theano: A Python framework for fast computation of mathematical expressions,” May 2016, Accessed: Jul. 26, 2021. [Online]. Available: <http://arxiv.org/abs/1605.02688>.
- [68] S. Tokui *et al.*, “Chainer: A Deep Learning Framework for Accelerating the

- Research Cycle,” in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 2002–2011, doi: 10.1145/3292500.3330756.
- [69] F. Chollet, “Keras,” *GitHub Repos.*, 2015, [Online]. Available: <https://github.com/fchollet/keras>.
 - [70] A. Paszke *et al.*, “PyTorch: An Imperative Style, High-Performance Deep Learning Library,” *Adv. Neural Inf. Process. Syst.* 32, pp. 8024–8035, 2019, [Online]. Available: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.
 - [71] T. Chen *et al.*, “MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems,” Dec. 2015, Accessed: Jul. 26, 2021. [Online]. Available: <http://arxiv.org/abs/1512.01274>.
 - [72] Y. Jia *et al.*, “Caffe: Convolutional architecture for fast feature embedding,” in *MM 2014 - Proceedings of the 2014 ACM Conference on Multimedia*, Nov., pp. 675–678, doi: 10.1145/2647868.2654889.
 - [73] Paperswithcode.com, “Papers With Code: Trends,” *paperswithcode*, 2021. <https://paperswithcode.com/trends> (accessed Jul. 12, 2021).
 - [74] F. Pedregosa *et al.*, “Scikit-learn: Machine Learning in Python,” *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011, doi: 10.5555/1953048.2078195.
 - [75] NHTSA, “Federal Automated Vehicles Policy: Accelerating the Next Revolution In Roadway Safety,” *U.S. Dep. Transp.*, no. September, p. 116, 2016, [Online]. Available: https://www.transportation.gov/sites/dot.gov/files/docs/AV_policy_guidance_PDF.pdf.
 - [76] Jennifer Shuttleworth, “SAE J3016 automated-driving graphic,” *SAE International*, 2019. <https://www.sae.org/news/2019/01/sae-updates-j3016-automated-driving-graphic> (accessed Jul. 29, 2021).
 - [77] NVIDIA, “Audi and NVIDIA | NVIDIA,” 2021. <https://www.nvidia.com/en-gb/self-driving-cars/partners/audi/> (accessed Jul. 27, 2021).
 - [78] P. E. Ross, “The Audi A8: the World’s First Production Car to Achieve Level 3 Autonomy - IEEE Spectrum,” Jul. 11, 2017. <https://spectrum.ieee.org/cars-that-think/transportation/self-driving/the-audi-a8-the-worlds-first-production-car-to-achieve-level-3-autonomy> (accessed Jul. 27, 2021).
 - [79] E. Ackerman, “What Full Autonomy Means for the Waymo Driver,” *IEEE Spectrum*, 2021. <https://spectrum.ieee.org/full-autonomy-waymo-driver> (accessed

Dec. 01, 2022).

- [80] Navya, “Navya reaches a new milestone in autonomous mobility with the first fully autonomous level 4 operation on a restricted site,” 2021. <https://navya.tech/en/navya-reaches-a-new-milestone-in-autonomous-mobility-with-the-first-fully-autonomous-level-4-operation-on-a-closed-site/> (accessed Jul. 27, 2022).
- [81] “Honda to develop advanced level 3 self-driving technology by 2029,” *Reuters*, 2022. <https://www.reuters.com/business/autos-transportation/honda-develop-advanced-level-3-self-driving-technology-by-2029-2022-11-30/> (accessed Nov. 30, 2022).
- [82] M. Rechtin, “Toyota Might Skip Level 3 Autonomy.” <https://www.motortrend.com/news/toyota-might-skip-level-3-autonomy/> (accessed Oct. 10, 2022).
- [83] E. Ackerman, “Toyota’s Gill Pratt on Self-Driving Cars and the Reality of Full Autonomy - IEEE Spectrum,” *IEEE Spectrum*, Jan. 23, 2017. <https://spectrum.ieee.org/cars-that-think/transportation/self-driving/toyota-gill-pratt-on-the-reality-of-full-autonomy> (accessed Jul. 27, 2021).
- [84] J. Levinson *et al.*, “Towards fully autonomous driving: Systems and algorithms,” in *IEEE Intelligent Vehicles Symposium*, 2011, pp. 163–168, doi: 10.1109/IVS.2011.5940562.
- [85] C. Urmson *et al.*, “Autonomous Driving in Urban Environments: Boss and the Urban Challenge,” *J. F. Robot.*, vol. 33, no. 1, pp. 1–17, 2014, doi: 10.1002/rob.
- [86] E. K. Lee, M. Gerla, G. Pau, U. Lee, and J. H. Lim, “Internet of Vehicles: From intelligent grid to autonomous cars and vehicular fogs,” *Int. J. Distrib. Sens. Networks*, vol. 12, no. 9, 2016, doi: 10.1177/1550147716665500.
- [87] M. Gerla, E. K. Lee, G. Pau, and U. Lee, “Internet of Vehicles: From Intelligent Grid to Autonomous Cars and Vehicular Clouds,” *Arch. Phys. Med. Rehabil.*, vol. 46, pp. 198–199, 1965.
- [88] M. Amadeo, C. Campolo, and A. Molinaro, “Information-centric networking for connected vehicles: A survey and future perspectives,” *IEEE Commun. Mag.*, vol. 54, no. 2, pp. 98–104, 2016, doi: 10.1109/MCOM.2016.7402268.
- [89] H. Xu, Y. Gao, F. Yu, and T. Darrell, “End-to-end learning of driving models from large-scale video datasets,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 3530–3538, doi: 10.1109/CVPR.2017.376.

- [90] B. R. Kiran *et al.*, “Deep Reinforcement Learning for Autonomous Driving: A Survey,” *IEEE Trans. Intell. Transp. Syst.*, pp. 1–18, 2021, doi: 10.1109/TITS.2021.3054625.
- [91] L. Chen, Q. Wang, X. Lu, D. Cao, and F. Y. Wang, “Learning Driving Models from Parallel End-to-End Driving Data Set,” in *Proceedings of the IEEE*, 2020, vol. 108, no. 2, pp. 262–273, doi: 10.1109/JPROC.2019.2952735.
- [92] S. Behere and M. Törngren, “A functional architecture for autonomous driving,” in *2015 First International Workshop on Automotive Software Architecture (WASA)*, May 2015, pp. 3–10, doi: 10.1145/2752489.2752491.
- [93] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, “DeepDriving: Learning affordance for direct perception in autonomous driving,” in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 2722–2730, doi: 10.1109/ICCV.2015.312.
- [94] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda, and T. Hamada, “An open approach to autonomous vehicles,” *IEEE Micro*, vol. 35, no. 6, pp. 60–68, 2015, doi: 10.1109/MM.2015.133.
- [95] S. Baluja, “Evolution of an artificial neural network based autonomous land vehicle controller,” *IEEE Trans. Syst. Man, Cybern. Part B Cybern.*, vol. 26, no. 3, pp. 450–463, 1996, doi: 10.1109/3477.499795.
- [96] J. Koutník, G. Cuccu, J. Schmidhuber, and F. Gomez, “Evolving large-scale neural networks for vision-based reinforcement learning,” in *GECCO 2013 - Proceedings of the 2013 Genetic and Evolutionary Computation Conference*, 2013, pp. 1061–1068, doi: 10.1145/2463372.2463509.
- [97] Y. LeCun, U. Muller, J. Ben, E. Cosatto, and B. Flepp, “Off-road obstacle avoidance through end-to-end learning,” *Adv. Neural Inf. Process. Syst.*, pp. 739–746, 2005.
- [98] A. Kendall *et al.*, “Learning to drive in a day,” in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 8248–8254, doi: 10.1109/ICRA.2019.8793742.
- [99] L. Chi and Y. Mu, “Learning end-to-end autonomous steering model from spatial and temporal visual cues,” in *Proceedings of the Workshop on Visual Analysis in Smart and Connected Communities*, Oct. 2017, pp. 9–16, doi: 10.1145/3132734.3132737.
- [100] S. J. Anderson, S. B. Karumanchi, and K. Iagnemma, “Constraint-based planning

- and control for safe, semi-autonomous operation of vehicles,” in *2012 IEEE Intelligent Vehicles Symposium*, 2012, pp. 383–388, doi: 10.1109/IVS.2012.6232153.
- [101] Y. Tian, K. Pei, S. Jana, and B. Ray, “DeepTest: Automated Testing of Deep-Neural-Network-driven Autonomous Cars,” 2017, [Online]. Available: <http://arxiv.org/abs/1708.08559>.
- [102] M. Wood *et al.*, “Safety First For Automated Driving,” 2019. [Online]. Available: <https://www.daimler.com/documents/innovation/other/safety-first-for-automated-driving.pdf>.
- [103] H. Somerville, P. Lienert, and A. Sage, “Uber’s use of fewer safety sensors prompts questions after Arizona crash,” *Reuters*. <https://www.reuters.com/article/us-uber-selfdriving-sensors-insight/ubers-use-of-fewer-safety-sensors-prompts-questions-after-arizona-crash-idUSKBN1H337Q> (accessed May 10, 2021).
- [104] A. Broggi *et al.*, “Extensive tests of autonomous driving technologies,” *IEEE Trans. Intell. Transp. Syst.*, vol. 14, no. 3, pp. 1403–1415, 2013, doi: 10.1109/TITS.2013.2262331.
- [105] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Dec. 2016, pp. 779–788, doi: 10.1109/CVPR.2016.91.
- [106] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask R-CNN,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 2, pp. 386–397, 2017, [Online]. Available: <https://arxiv.org/abs/1703.06870>.
- [107] A. Bar Hillel, R. Lerner, D. Levi, and G. Raz, “Recent progress in road and lane detection: A survey,” *Mach. Vis. Appl.*, vol. 25, no. 3, pp. 727–745, 2014, doi: 10.1007/s00138-011-0404-2.
- [108] C. Fernández, D. Fernández-Llorca, and M. A. Sotelo, “A Hybrid Vision-Map Method for Urban Road Detection,” *J. Adv. Transp.*, vol. 2017, 2017, doi: 10.1155/2017/7090549.
- [109] M.-P. Dubuisson and A. K. Jain, “A modified Hausdorff distance for object matching,” in *Proceedings of 12th International Conference on Pattern Recognition*, Dec. 2002, pp. 566–568, doi: 10.1109/ICPR.1994.576361.
- [110] S. Hwang, N. Kim, Y. Choi, S. Lee, and I. S. Kweon, “Fast multiple objects

- detection and tracking fusing color camera and 3D LIDAR for intelligent vehicles,” in *2016 13th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, 2016, pp. 234–239, doi: 10.1109/URAI.2016.7625744.
- [111] T. N. Nguyen, B. Michaelis, A. Al-Hamadi, M. Tornow, and M. M. Meinecke, “Stereo-camera-based urban environment perception using occupancy grid and object tracking,” *IEEE Trans. Intell. Transp. Syst.*, vol. 13, no. 1, pp. 154–165, 2012, doi: 10.1109/TITS.2011.2165705.
- [112] J. Ziegler, P. Bender, T. Dang, and C. Stiller, “Trajectory planning for Bertha - A local, continuous method,” in *2014 IEEE Intelligent Vehicles Symposium Proceedings*, 2014, pp. 450–457, doi: 10.1109/IVS.2014.6856581.
- [113] S. Kuutti, S. Fallah, K. Katsaros, M. Dianati, F. Mccullough, and A. Mouzakitis, “A Survey of the State-of-the-Art Localization Techniques and Their Potentials for Autonomous Vehicle Applications,” *IEEE Internet Things J.*, vol. 5, no. 2, pp. 829–846, 2018, doi: 10.1109/JIOT.2018.2812300.
- [114] G. Bresson, Z. Alsayed, L. Yu, and S. Glaser, “Simultaneous Localization and Mapping: A Survey of Current Trends in Autonomous Driving,” *IEEE Trans. Intell. Veh.*, vol. 2, no. 3, pp. 194–220, 2017, doi: 10.1109/TIV.2017.2749181.
- [115] T. Bailey and H. Durrant-Whyte, “Simultaneous localization and mapping (SLAM): Part II,” *IEEE Robot. Autom. Mag.*, vol. 13, no. 3, pp. 108–117, 2006, doi: 10.1109/MRA.2006.1678144.
- [116] M. A. Al Khedher, “Hybrid GPS-GSM Localization of Automobile Tracking System,” *Int. J. Comput. Sci. Inf. Technol.*, vol. 3, no. 6, pp. 75–85, 2011, doi: 10.5121/ijcsit.2011.3606.
- [117] K. S. Chong and L. Kleeman, “Accurate odometry and error modelling for a mobile robot,” in *Proceedings of International Conference on Robotics and Automation*, 1997, vol. 4, pp. 2783–2788, doi: 10.1109/robot.1997.606708.
- [118] C. P. Urmson *et al.*, “High Speed Navigation of Unrehearsed Terrain: Red Team Technology for Grand Challenge 2004,” Pittsburgh, PA, 2004. [Online]. Available: <http://www.blackwell-synergy.com/doi/abs/10.1111/j.1365-2486.2006.01252.x>.
- [119] A. Hata and D. Wolf, “Road marking detection using LIDAR reflective intensity data and its application to vehicle localization,” in *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, 2014, pp. 584–589, doi:

10.1109/ITSC.2014.6957753.

- [120] T. Ort, L. Paull, and D. Rus, “Autonomous Vehicle Navigation in Rural Environments Without Detailed Prior Maps,” in *Proceedings - IEEE International Conference on Robotics and Automation*, 2018, pp. 2040–2047, doi: 10.1109/ICRA.2018.8460519.
- [121] J. Levinson and S. Thrun, “Robust vehicle localization in urban environments using probabilistic maps,” in *Proceedings - IEEE International Conference on Robotics and Automation*, 2010, pp. 4372–4378, doi: 10.1109/ROBOT.2010.5509700.
- [122] P. E. Hart, N. J. Nilsson, and B. Raphael, “A Formal Basis for the Heuristic Determination of Minimum Cost Paths,” *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, 1968, doi: 10.1109/TSSC.1968.300136.
- [123] D. Van Vliet, “Improved shortest path algorithms for transport networks,” *Transp. Res.*, vol. 12, no. 1, pp. 7–20, 1978, doi: 10.1016/0041-1647(78)90102-8.
- [124] R. Geisberger, P. Sanders, D. Schultes, and C. Vetter, “Exact routing in large road networks using contraction hierarchies,” *Transp. Sci.*, vol. 46, no. 3, pp. 388–404, 2012, [Online]. Available: <http://dx.doi.org/10.1287/TRSC.1110.0401>.
- [125] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick, “Reachability and distance queries via 2-hop labels,” *SIAM J. Comput.*, vol. 32, no. 5, pp. 1338–1355, 2003, doi: 10.1137/S0097539702403098.
- [126] R. Bauer, D. Delling, P. Sanders, D. Schieferdecker, D. Schultes, and D. Wagner, “Combining hierarchical and goal-directed speed-up techniques for dijkstra’s algorithm,” *ACM J. Exp. Algorithmics*, vol. 15, pp. 1–26, 2010, doi: 10.1145/1671970.1671976.
- [127] A. Uçar, Y. Demir, and C. Güzeliş, “Object recognition and detection with deep learning for autonomous driving applications,” *Simulation*, vol. 93, no. 9, pp. 759–769, 2017, doi: 10.1177/0037549717709932.
- [128] M. Pivtoraiko and A. Kelly, “Efficient constrained path planning via search in state lattices,” in *Proceedings of 8th International Symposium on Artificial Intelligence, Robotics and Automation in Space (iSAIRAS '05)*, 2005, no. 603, pp. 249–255.
- [129] J. Barraquand and J. C. Latombe, “Robot Motion Planning: A Distributed Representation Approach,” *Int. J. Rob. Res.*, vol. 10, no. 6, pp. 628–649, 1991, doi: 10.1177/027836499101000604.
- [130] S. M. LaValle and J. J. Kuffner, “Randomized kinodynamic planning,” in

- Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, 1999, vol. 1, pp. 473–479, doi: 10.1109/ROBOT.1999.770022.
- [131] H. Fuji, J. Xiang, Y. Tazaki, B. Levedahl, and T. Suzuki, “Trajectory planning for automated parking using multi-resolution state roadmap considering non-holonomic constraints,” *IEEE Intell. Veh. Symp. Proc.*, vol. 407, pp. 407–413, 2014, doi: 10.1109/IVS.2014.6856433.
 - [132] J. P. Rastelli, R. Lattarulo, and F. Nashashibi, “Dynamic trajectory generation using continuous-curvature algorithms for door to door assistance vehicles,” in *2014 IEEE Intelligent Vehicles Symposium Proceedings*, 2014, pp. 510–515, doi: 10.1109/IVS.2014.6856526.
 - [133] J. Ren, K. A. McIsaac, and R. V. Patel, “Modified Newton’s method applied to potential field-based navigation for mobile robots,” *IEEE Trans. Robot.*, vol. 22, no. 2, pp. 384–391, 2006, doi: 10.1109/TRO.2006.870668.
 - [134] L. Caltagirone, M. Bellone, L. Svensson, and M. Wahde, “LIDAR-based driving path generation using fully convolutional neural networks,” in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, 2017, pp. 1–6, doi: 10.1109/ITSC.2017.8317618.
 - [135] D. V McGehee, E. N. Mazzae, and G. H. S. Baldwin, “Driver Reaction Time in Crash Avoidance Research: Validation of a Driving Simulator Study on a Test Track,” in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 2000, vol. 44, no. 20, pp. 3–323, doi: 10.1177/154193120004402026.
 - [136] T. J. Crayton and B. M. Meier, “Autonomous vehicles: Developing a public health research agenda to frame the future of transportation policy,” *J. Transp. Heal.*, vol. 6, pp. 245–252, 2017, doi: 10.1016/j.jth.2017.04.004.
 - [137] A. Tilley, “Google’s Self-Driving Car Caused Its First Accident,” *Forbes*, 2016. <https://www.forbes.com/sites/aarontilley/2016/02/29/googles-self-driving-car-caused-its-first-accident/?sh=3b9b1b2f538d> (accessed May 12, 2021).
 - [138] E. Fernandez, “Who Is Responsible In A Crash With A Self-Driving Car?,” *Forbes*, 2018. <https://www.forbes.com/sites/fernandezelizabeth/2020/02/06/who-is-responsible-in-a-crash-with-a-self-driving-car/?sh=7b5dbd974b2b> (accessed May 12, 2021).
 - [139] D. Lavrinc, “This is how bad self-driving cars suck in rain,” *Jalopnik*, 2014. <https://jalopnik.com/this-is-how-bad-self-driving-cars-suck-in-the-rain->

- 1666268433 (accessed May 09, 2021).
- [140] “Uber’s self-driving operator charged over fatal crash,” *BBC*, 2020. <https://www.bbc.co.uk/news/technology-54175359> (accessed Jul. 29, 2021).
 - [141] A. Dosovitskiy *et al.*, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale,” Oct. 2020, Accessed: Jul. 26, 2021. [Online]. Available: <http://arxiv.org/abs/2010.11929>.
 - [142] H. Pham, Q. Xie, Z. Dai, and Q. V. Le, “Meta Pseudo Labels,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 11557–11568, [Online]. Available: <https://arxiv.org/abs/2003.10580>.
 - [143] X. Chen *et al.*, “Symbolic Discovery of Optimization Algorithms,” Feb. 2023, doi: 10.48550/arxiv.2302.06675.
 - [144] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, “End-to-End Object Detection with Transformers,” in *Computer Vision – ECCV 2020*, 2020, pp. 213–229, [Online]. Available: <http://arxiv.org/abs/2005.12872>.
 - [145] G. Jocher, A. Chaurasia, and J. Qiu, “YOLO by Ultralytics.” 2023, doi: 10.5281/zenodo.3908559.
 - [146] Q. Hong, F. Liu, D. Li, J. Liu, L. Tian, and Y. Shan, “Dynamic Sparse R-CNN,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2022-June, pp. 4713–4722, May 2022, doi: 10.48550/arxiv.2205.02101.
 - [147] A. G. Howard *et al.*, “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,” Apr. 2017, Accessed: Jul. 26, 2021. [Online]. Available: <http://arxiv.org/abs/1704.04861>.
 - [148] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, 2017, doi: 10.1109/TPAMI.2016.2577031.
 - [149] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, “SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and 0.5MB model size,” Feb. 2016, Accessed: Jul. 26, 2021. [Online]. Available: <http://arxiv.org/abs/1602.07360>.
 - [150] V. Losing, B. Hammer, and H. Wersing, “Incremental on-line learning: A review and comparison of state of the art algorithms,” *Neurocomputing*, vol. 275, pp. 1261–1274, 2018, doi: 10.1016/j.neucom.2017.06.084.
 - [151] P. R. Panda, N. D. Dutt, and A. Nicolau, “On-chip vs. off-chip memory: the data partitioning problem in embedded processor-based systems,” *ACM Trans. Des.*

- Autom. Electron. Syst.*, vol. 5, no. 3, pp. 682–704, 2000, doi: 10.1145/348019.348570.
- [152] S. Han *et al.*, “EIE: Efficient Inference Engine on Compressed Deep Neural Network,” in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, Aug. 2016, pp. 243–254, doi: 10.1109/ISCA.2016.30.
- [153] Mihir Mody, “ADAS Front Camera: Demystifying Resolution and Frame-Rate,” *EETimes*, 2016. <https://www.eetimes.com/adas-front-camera-demystifying-resolution-and-frame-rate/#> (accessed Mar. 10, 2021).
- [154] G. Johansson and K. Rumar, “Drivers’ Brake Reaction Times,” *Hum. Factors*, vol. 13, no. 1, pp. 23–27, 1971, doi: 10.1177/001872087101300104.
- [155] A. Newell and S. K. Card, “The Prospects for Psychological Science in Human-Computer Interaction,” *Hum.-Comput. Interact.*, vol. 1, no. 3, pp. 209–242, Sep. 1985, doi: 10.1207/s15327051hci0103_1.
- [156] S. Thorpe, D. Fize, and C. Marlot, “Speed of processing in the human visual system,” *Nature*, vol. 381, no. 6582, pp. 520–522, 1996, doi: 10.1038/381520a0.
- [157] Mobileye, “Mobileye C2-270 Essentials,” 2017. [Online]. Available: https://manualzz.com/doc/23526946/mobileye-c2-270-essentials?__cf_chl_jschl_tk__=pmd_b91f7eab6b66ebe7b1bf5a63e43ccefe471565b0-1627407908-0-gqNtZGzNAfjcnBszQii.
- [158] R. Rajalingham, E. B. Issa, P. Bashivan, K. Kar, K. Schmidt, and J. J. DiCarlo, “Large-scale, high-resolution comparison of the core visual object recognition behavior of humans, monkeys, and state-of-the-art deep artificial neural networks,” *J. Neurosci.*, vol. 38, no. 33, pp. 7255–7269, 2018, doi: 10.1523/JNEUROSCI.0388-18.2018.
- [159] R. M. Cichy, A. Khosla, D. Pantazis, A. Torralba, and A. Oliva, “Comparison of deep neural networks to spatio-temporal cortical dynamics of human visual object recognition reveals hierarchical correspondence,” *Sci. Rep.*, vol. 6, pp. 1–13, 2016, doi: 10.1038/srep27755.
- [160] L. E. van Dyck and W. R. Gruber, “Seeing eye-to-eye? A comparison of object recognition performance in humans and deep convolutional neural networks under image manipulation,” Jul. 2020, Accessed: Jul. 27, 2021. [Online]. Available: <https://arxiv.org/abs/2007.06294v2>.
- [161] S. Dodge and L. Karam, “A study and comparison of human and deep learning

- recognition performance under visual distortions,” in *2017 26th International Conference on Computer Communications and Networks (ICCCN)*, 2017, pp. 1–7, doi: 10.1109/ICCCN.2017.8038465.
- [162] U.S. Department of Transportation, “Highway Statistics 2015,” 2015. [Online]. Available: <https://www.fhwa.dot.gov/policyinformation/statistics/2015/>.
- [163] J. Levinson, M. Montemerlo, and S. Thrun, “Map-based precision vehicle localization in urban environments,” *Robot. Sci. Syst.*, vol. 3, pp. 121–128, 2008, doi: 10.15607/rss.2007.iii.016.
- [164] C. Berger and B. Rumpe, “Autonomous driving—5 years after the urban challenge: The anticipatory vehicle as a cyber-physical system,” in *Proceedings of the 10th Workshop on Automotive Software Engineering (ASE 2012)*, 2012, vol. 208, pp. 789–798.
- [165] Xilinx, “XA Spartan-3A Automotive FPGA Family Data Sheet,” 2021.
- [166] Xilinx, “XA Artix-7 FPGAs Data Sheet: Overview,” 2017.
- [167] Xilinx, “XA Spartan-7 Automotive FPGA Data Sheet: Overview,” 2019.
- [168] “Cyclone IV FPGA,” *Intel*. <https://www.intel.com> (accessed Jul. 29, 2021).
- [169] “Intel Max 10 FPGA,” *Intel*. <https://www.intel.com> (accessed Jul. 29, 2021).
- [170] “Intel Cyclone 10 LP FPGA,” *Intel*. <https://www.intel.com> (accessed Jul. 29, 2021).
- [171] L. Jiao *et al.*, “A survey of deep learning-based object detection,” *IEEE Access*, vol. 7, no. 3, pp. 128837–128868, 2019, doi: 10.1109/ACCESS.2019.2939201.
- [172] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, “Learning spatiotemporal features with 3D convolutional networks,” in *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 4489–4497, doi: 10.1109/ICCV.2015.510.
- [173] F. Yu and V. Koltun, “Multi-Scale Context Aggregation by Dilated Convolutions,” Nov. 2015, Accessed: Jul. 27, 2021. [Online]. Available: <https://arxiv.org/abs/1511.07122v3>.
- [174] J. Jin, A. Dundar, and E. Culurciello, “Flattened Convolutional Neural Networks for Feedforward Acceleration,” Dec. 2014, Accessed: Jul. 26, 2021. [Online]. Available: <http://arxiv.org/abs/1412.5474>.
- [175] Y. LeCun, C. Cortes, and C. Burges, “MNIST handwritten digit database,” *ATT Labs*, vol. 2, 2010, [Online]. Available: <http://yann.lecun.com/exdb/mnist>.
- [176] C. Shorten and T. M. Khoshgoftaar, “A survey on Image Data Augmentation for

- Deep Learning,” *J. Big Data*, vol. 6, no. 1, pp. 1–48, 2019, doi: 10.1186/s40537-019-0197-0.
- [177] M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional networks,” in *Computer Vision – ECCV 2014*, vol. 8689, no. 1, pp. 818–833, doi: 10.1007/978-3-319-10590-1_53.
- [178] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” in *ICLR 2015*, vol. 75, no. 6, pp. 398–406, doi: 10.2146/ajhp170251.
- [179] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, “Squeeze-and-Excitation Networks,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 8, pp. 2011–2023, 2020, doi: 10.1109/TPAMI.2019.2913372.
- [180] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, “Learning Transferable Architectures for Scalable Image Recognition,” in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8697–8710, doi: 10.1109/CVPR.2018.00907.
- [181] D. Mahajan *et al.*, “Exploring the Limits of Weakly Supervised Pretraining,” in *Lecture Notes in Computer Science*, 2018, vol. 11206, pp. 185–201, doi: 10.1007/978-3-030-01216-8_12.
- [182] F. Zhuang *et al.*, “A Comprehensive Survey on Transfer Learning,” *Proc. IEEE*, vol. 109, no. 1, pp. 43–76, 2020, doi: 10.1109/JPROC.2020.3004555.
- [183] Q. Xie, M.-T. Luong, E. Hovy, and Q. V. Le, “Self-Training With Noisy Student Improves ImageNet Classification,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2020, pp. 10684–10695, doi: 10.1109/cvpr42600.2020.01070.
- [184] J. E. Van Engelen and H. H. Hoos, “A survey on semi-supervised learning,” *Mach. Learn.*, vol. 109, no. 2, pp. 373–440, 2020, doi: 10.1007/s10994-019-05855-6.
- [185] H. H. Mao, “A Survey on Self-supervised Pre-training for Sequential Transfer Learning in Neural Networks,” Jul. 2020, Accessed: Jul. 26, 2021. [Online]. Available: <http://arxiv.org/abs/2007.00800>.
- [186] G. Hinton, O. Vinyals, and J. Dean, “Distilling the Knowledge in a Neural Network,” Mar. 2015, Accessed: Jul. 26, 2021. [Online]. Available: <https://arxiv.org/abs/1503.02531>.
- [187] M. Tan and Q. V. Le, “EfficientNet: Rethinking model scaling for convolutional neural networks,” in *36th International Conference on Machine Learning, ICML*

- 2019, pp. 10691–10700, [Online]. Available: <https://arxiv.org/abs/1905.11946>.
- [188] H. Touvron, A. Vedaldi, M. Douze, and H. Jégou, “Fixing the train-test resolution discrepancy: FixEfficientNet,” Mar. 2020, Accessed: Jul. 26, 2021. [Online]. Available: <https://arxiv.org/abs/2003.08237>.
 - [189] H. Touvron, A. Vedaldi, M. Douze, and H. Jégou, “Fixing the train-test resolution discrepancy,” *Adv. Neural Inf. Process. Syst.*, vol. 32, pp. 1–14, 2019.
 - [190] A. Vaswani *et al.*, “Attention is all you need,” in *31st Conference on Neural Information Processing Systems (NIPS 2017)*, 2017, pp. 5999–6009, [Online]. Available: <https://arxiv.org/abs/1706.03762>.
 - [191] I. Bello, “LambdaNetworks: Modeling Long-Range Interactions without Attention,” in *ICLR 2021*, p. 16, [Online]. Available: <https://arxiv.org/pdf/2102.08602.pdf>.
 - [192] C. Sun, A. Shrivastava, S. Singh, and A. Gupta, “Revisiting Unreasonable Effectiveness of Data in Deep Learning Era,” in *2017 International Conference on Computer Vision (ICCV)*, pp. 843–852, doi: 10.1109/ICCV.2017.97.
 - [193] F. Petersen, H. Kuehne, C. Borgelt, and O. Deussen, “Differentiable Top-k Classification Learning,” Jun. 2022, doi: 10.48550/arxiv.2206.07290.
 - [194] Y. Cai *et al.*, “Reversible Column Networks,” Dec. 2022, doi: 10.48550/arxiv.2212.11696.
 - [195] K. Gregor, I. Danihelka, A. Graves, D. Rezende, and D. Wierstra, “DRAW: A Recurrent Neural Network For Image Generation,” in *Proceedings of the 32nd International Conference on Machine Learning*, 2015, vol. 37, pp. 1462–1471, [Online]. Available: <http://proceedings.mlr.press/v37/gregor15.html>.
 - [196] H. Pham *et al.*, “Combined Scaling for Open-Vocabulary Image Classification,” Nov. 2021, doi: 10.48550/arxiv.2111.10050.
 - [197] M. Lin, Q. Chen, and S. Yan, “Network In Network,” Dec. 2013, Accessed: Jul. 26, 2021. [Online]. Available: <https://arxiv.org/abs/1312.4400>.
 - [198] S. Arora, A. Bhaskara, R. Ge, and T. Ma, “Provable bounds for learning some deep representations,” in *31st International Conference on Machine Learning, ICML 2014*, vol. 1, pp. 883–891.
 - [199] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the Inception Architecture for Computer Vision,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2818–2826, doi: 10.1109/CVPR.2016.308.

- [200] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, “Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning,” in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*, 2017, pp. 4278–4284, [Online]. Available: <http://arxiv.org/abs/1602.07261>.
- [201] F. Chollet, “Xception: Deep learning with depthwise separable convolutions,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1800–1807, doi: 10.1109/CVPR.2017.195.
- [202] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. C. Chen, “MobileNetV2: Inverted Residuals and Linear Bottlenecks,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Dec. 2018, pp. 4510–4520, doi: 10.1109/CVPR.2018.00474.
- [203] A. Gholami *et al.*, “SqueezeNext: Hardware-aware neural network design,” *2018 IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Work.*, pp. 1719–1728, 2018, doi: 10.1109/CVPRW.2018.00215.
- [204] S. N. Gowda and C. Yuan, “ColorNet: Investigating the Importance of Color Spaces for Image Classification,” in *Computer Vision – ACCV 2018*, 2019, vol. 11364 LNCS, pp. 581–596, doi: 10.1007/978-3-030-20870-7_36.
- [205] S. Mehta and M. R. Apple, “MobileViT: Light-weight, General-purpose, and Mobile-friendly Vision Transformer,” Oct. 2021, doi: 10.48550/arxiv.2110.02178.
- [206] S. N. Wadekar and A. Chaurasia, “MobileViTv3: Mobile-Friendly Vision Transformer with Simple and Effective Fusion of Local, Global and Input Features,” Sep. 2022, doi: 10.48550/arxiv.2209.15159.
- [207] J. Lu *et al.*, “SOFT: Softmax-free Transformer with Linear Complexity,” Oct. 2021, doi: 10.48550/arxiv.2110.11945.
- [208] D. Gschwend, “ZynqNet: An FPGA-Accelerated Embedded Convolutional Neural Network,” 2020.
- [209] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, “Aggregated residual transformations for deep neural networks,” in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, Nov. 2017, pp. 5987–5995, doi: 10.1109/CVPR.2017.634.
- [210] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, “Object detection with discriminatively trained part-based models,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 9, pp. 1627–1645, 2010, doi: 10.1109/TPAMI.2009.167.

- [211] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders, "Selective Search for Object Recognition," *Int. J. Comput. Vis.* 2013 1042, vol. 104, no. 2, pp. 154–171, Apr. 2013, doi: 10.1007/S11263-013-0620-5.
- [212] A. Vedaldi, V. Gulshan, M. Varma, and A. Zisserman, "Multiple Kernels for object detection," in *2009 IEEE 12th International Conference on Computer Vision*, 2009, pp. 606–613, doi: 10.1109/ICCV.2009.5459183.
- [213] 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)*, 2005, vol. 1, pp. 886–893, doi: 10.1109/CVPR.2005.177.
- [214] Y. Yu, J. Zhang, Y. Huang, S. Zheng, W. Ren, and C. Wang, "Object detection by context and boosted HOG-LBP," 2010.
- [215] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vis.*, vol. 60, no. 2, pp. 91–110, 2004, doi: 10.1023/B:VISI.0000029664.99615.94.
- [216] P. Viola and M. Jones, "Robust real-time face detection," in *Proceedings of the IEEE International Conference on Computer Vision*, 2001, vol. 2, p. 747, doi: 10.1109/ICCV.2001.937709.
- [217] N. M. Ali, S. W. Jun, M. S. Karis, M. M. Ghazaly, and M. S. M. Aras, "Object classification and recognition using Bag-of-Words (BoW) model," in *Proceeding - 2016 IEEE 12th International Colloquium on Signal Processing and its Applications, CSPA 2016*, Jul. 2016, pp. 216–220, doi: 10.1109/CSPA.2016.7515834.
- [218] I. Steinwart and A. Christmann, *Support Vector Machines*. New York, USA: Springer, 2008.
- [219] Y. Freund and R. E. Schapire, "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting," *J. Comput. Syst. Sci.*, vol. 55, pp. 119–139, 1997.
- [220] T. Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 936–944, doi: 10.1109/CVPR.2017.106.
- [221] Z. Li, C. Peng, G. Yu, X. Zhang, Y. Deng, and J. Sun, "DetNet: A Backbone network for Object Detection," Apr. 2018, Accessed: Jul. 26, 2021. [Online]. Available: <https://arxiv.org/abs/1804.06215v2>.

- [222] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 580–587, doi: 10.1109/CVPR.2014.81.
- [223] R. Girshick, “Fast R-CNN,” in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1440–1448, doi: 10.1109/ICCV.2015.169.
- [224] W. Liu *et al.*, “SSD: Single Shot Multibox Detector,” in *Computer Vision – ECCV 2016*, 2016, vol. 9905, pp. 21–37, doi: 10.1007/978-3-319-46448-0_2.
- [225] A. Neubeck and L. Van Gool, “Efficient non-maximum suppression,” in *Proceedings - International Conference on Pattern Recognition*, 2006, vol. 3, pp. 850–855, doi: 10.1109/ICPR.2006.479.
- [226] K. He, X. Zhang, S. Ren, and J. Sun, “Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 9, pp. 1904–1916, 2015, doi: 10.1109/TPAMI.2015.2389824.
- [227] X. Qi, T. Wang, and J. Liu, “Comparison of Support Vector Machine and Softmax Classifiers in Computer Vision,” in *Proceedings - 2017 2nd International Conference on Mechanical, Control and Computer Engineering, ICMCCE 2017*, Jan. 2018, vol. 2018-January, pp. 151–155, doi: 10.1109/ICMCCE.2017.49.
- [228] P.-T. de Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein, “A Tutorial on the Cross-Entropy Method,” *Ann. Oper. Res. 2005 1341*, vol. 134, no. 1, pp. 19–67, Jan. 2005, doi: 10.1007/S10479-005-5724-Z.
- [229] K. Janocha and W. M. Czarnecki, “On Loss Functions for Deep Neural Networks in Classification,” *Schedae Informaticae*, vol. 25, pp. 49–59, Feb. 2017, Accessed: Jul. 28, 2021. [Online]. Available: <https://arxiv.org/abs/1702.05659v1>.
- [230] Y. Li, K. He, J. Sun, and others, “R-FCN: Object detection via region-based fully convolutional networks,” *Adv. Neural Inf. Process. Syst.*, pp. 379–387, 2016, [Online]. Available: <http://papers.nips.cc/paper/6465-r-fcn-object-detection-via-region-based-fully-convolutional-networks.pdf>.
- [231] S. Li, L. Yang, J. Huang, X. S. Hua, and L. Zhang, “Dynamic anchor feature selection for single-shot object detection,” *2019 IEEE/CVF Int. Conf. Comput. Vis.*, vol. 2019-Octob, pp. 6608–6617, Oct. 2019, doi: 10.1109/ICCV.2019.00671.
- [232] H. Zhang, H. Chang, B. Ma, N. Wang, and X. Chen, “Dynamic R-CNN: Towards High Quality Object Detection via Dynamic Training,” pp. 1–19, 2020, [Online].

Available: <http://arxiv.org/abs/2004.06002>.

- [233] R. Padilla, S. L. Netto, and E. A. B. Da Silva, “A Survey on Performance Metrics for Object-Detection Algorithms,” *2020 Int. Conf. Syst. Signals Image Process.*, vol. 2020-July, pp. 237–242, Jul. 2020, doi: 10.1109/IWSSIP48289.2020.9145130.
- [234] P. Sun *et al.*, “Sparse R-CNN: End-to-End Object Detection with Learnable Proposals,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 14449–14458, Nov. 2020, doi: 10.48550/arxiv.2011.12450.
- [235] H. W. Kuhn, “The Hungarian method for the assignment problem,” *Nav. Res. Logist.*, vol. 52, no. 1, pp. 7–21, Feb. 2005, doi: 10.1002/NAV.20053.
- [236] Y. Chen, X. Dai, M. Liu, D. Chen, L. Yuan, and Z. Liu, “Dynamic Convolution: Attention over Convolution Kernels,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 11027–11036, Dec. 2019, doi: 10.48550/arxiv.1912.03458.
- [237] B. Wu, F. Iandola, P. H. Jin, and K. Keutzer, “SqueezeDet: Unified, Small, Low Power Fully Convolutional Neural Networks for Real-Time Object Detection for Autonomous Driving,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2017, pp. 446–454, doi: 10.1109/CVPRW.2017.60.
- [238] J. Redmon and A. Farhadi, “YOLO9000: Better, faster, stronger,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 6517–6525, doi: 10.1109/CVPR.2017.690.
- [239] S. Ioffe and C. Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” *32nd Int. Conf. Mach. Learn. ICML 2015*, vol. 1, pp. 448–456, Feb. 2015, Accessed: Jul. 29, 2021. [Online]. Available: <https://arxiv.org/abs/1502.03167v3>.
- [240] K. Wagstaff, C. Cardie, S. Rogers, and S. Schrödl, “Constrained K-Means Clustering with Background Knowledge,” in *Proceedings of the Eighteenth International Conference on Machine Learning*, 2001, pp. 577–584.
- [241] J. Redmon and A. Farhadi, “YOLOv3: An Incremental Improvement,” Apr. 2018, Accessed: Jul. 26, 2021. [Online]. Available: <https://arxiv.org/abs/1804.02767v1>.
- [242] K. Duan, S. Bai, L. Xie, H. Qi, Q. Huang, and Q. Tian, “CenterNet: Keypoint triplets for object detection,” in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 6568–6577, doi: 10.1109/ICCV.2019.00667.

- [243] H. Law and J. Deng, "CornerNet: Detecting Objects as Paired Keypoints," *Int. J. Comput. Vis.*, vol. 128, no. 3, pp. 642–656, 2020, doi: 10.1007/s11263-019-01204-1.
- [244] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection," Apr. 2020, Accessed: Jul. 27, 2021. [Online]. Available: <https://arxiv.org/abs/2004.10934v1>.
- [245] C. Y. Wang, H. Y. Mark Liao, Y. H. Wu, P. Y. Chen, J. W. Hsieh, and I. H. Yeh, "CSPNet: A new backbone that can enhance learning capability of CNN," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2020, pp. 1571–1580, doi: 10.1109/CVPRW50498.2020.00203.
- [246] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, "Path Aggregation Network for Instance Segmentation," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8759–8768, doi: 10.1109/CVPR.2018.00913.
- [247] G. Jocher, "YOLOv5 by Ultralytics." 2020, doi: 10.5281/zenodo.3908559.
- [248] C. Li *et al.*, "YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications," Sep. 2022, doi: 10.48550/arxiv.2209.02976.
- [249] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," Jul. 2022, doi: 10.48550/arxiv.2207.02696.
- [250] M. Bojarski *et al.*, "End to End Learning for Self-Driving Cars," Apr. 2016, Accessed: Jul. 26, 2021. [Online]. Available: <https://arxiv.org/abs/1604.07316v1>.
- [251] L. Chen, W. Zhan, W. Tian, Y. He, and Q. Zou, "Deep Integration: A Multi-Label Architecture for Road Scene Recognition," *IEEE Trans. Image Process.*, vol. 28, no. 10, pp. 4883–4898, 2019, doi: 10.1109/TIP.2019.2913079.
- [252] G. Li, Z. Ji, Y. Chang, S. Li, X. Qu, and D. Cao, "ML-ANet: A Transfer Learning Approach Using Adaptation Network for Multi-label Image Classification in Autonomous Driving," *Chinese J. Mech. Eng.*, vol. 34, no. 1, pp. 1–11, Aug. 2021, doi: 10.1186/S10033-021-00598-9.
- [253] L. Song *et al.*, "A Deep Multi-Modal CNN for Multi-Instance Multi-Label Image Classification," *IEEE Trans. Image Process.*, vol. 27, no. 12, pp. 6025–6038, 2018, doi: 10.1109/TIP.2018.2864920.
- [254] J. Wang, Y. Chen, S. Hao, W. Feng, and Z. Shen, "Balanced Distribution Adaptation for Transfer Learning," *Proc. - IEEE Int. Conf. Data Mining, ICDM*, vol. 2017-Novem, pp. 1129–1134, Jul. 2018, Accessed: Oct. 18, 2021. [Online].

Available: <https://arxiv.org/abs/1807.00516v1>.

- [255] G. Li *et al.*, “Drivers’ visual scanning behavior at signalized and unsignalized intersections: A naturalistic driving study in China,” *J. Safety Res.*, vol. 71, pp. 219–229, 2019, doi: <https://doi.org/10.1016/j.jsr.2019.09.012>.
- [256] C. Sakaridis, D. Dai, and L. Van Gool, “Semantic Foggy Scene Understanding with Synthetic Data,” *Int. J. Comput. Vis.*, vol. 126, no. 9, pp. 973–992, Aug. 2017, doi: [10.1007/s11263-018-1072-8](https://doi.org/10.1007/s11263-018-1072-8).
- [257] G. Li, Y. Yang, and X. Qu, “Deep Learning Approaches on Pedestrian Detection in Hazy Weather,” *IEEE Trans. Ind. Electron.*, vol. 67, no. 10, pp. 8889–8899, 2020, doi: [10.1109/TIE.2019.2945295](https://doi.org/10.1109/TIE.2019.2945295).
- [258] Z. Liu, Y. He, C. Wang, and R. Song, “Analysis of the Influence of Foggy Weather Environment on the Detection Effect of Machine Vision Obstacles,” *Sensors* 2020, vol. 20, no. 2, p. 349, Jan. 2020, doi: [10.3390/S20020349](https://doi.org/10.3390/S20020349).
- [259] Y. He and Z. Liu, “A Feature Fusion Method to Improve the Driving Obstacle Detection under Foggy Weather,” *IEEE Trans. Transp. Electr.*, 2021, doi: [10.1109/TTE.2021.3080690](https://doi.org/10.1109/TTE.2021.3080690).
- [260] S. C. Huang, T. H. Le, and D. W. Jaw, “DSNet: Joint Semantic Learning for Object Detection in Inclement Weather Conditions,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 8, pp. 2623–2633, Aug. 2021, doi: [10.1109/TPAMI.2020.2977911](https://doi.org/10.1109/TPAMI.2020.2977911).
- [261] M. Haris and A. Glowacz, “Road object detection: A comparative study of deep learning-based algorithms,” *Electronics*, vol. 10, no. 16, 2021, doi: [10.3390/electronics10161932](https://doi.org/10.3390/electronics10161932).
- [262] M. Hnewa and H. Radha, “Object Detection Under Rainy Conditions for Autonomous Vehicles: A Review of State-of-the-Art and Emerging Techniques,” *IEEE Signal Process. Mag.*, vol. 38, no. 1, pp. 53–67, Jun. 2020, doi: [10.1109/msp.2020.2984801](https://doi.org/10.1109/msp.2020.2984801).
- [263] M.-Y. Liu, T. Breuel, and J. Kautz, “Unsupervised Image-to-Image Translation Networks,” *Adv. Neural Inf. Process. Syst.*, vol. 2017-Decem, pp. 701–709, Mar. 2017, Accessed: Oct. 19, 2021. [Online]. Available: <https://arxiv.org/abs/1703.00848v6>.
- [264] Y. Chen, W. Li, C. Sakaridis, D. Dai, and L. Van Gool, “Domain Adaptive Faster R-CNN for Object Detection in the Wild,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 3339–3348, Mar. 2018, Accessed: Oct. 19,

2021. [Online]. Available: <https://arxiv.org/abs/1803.03243v1>.
- [265] N. A. M. Mai, P. Duthon, L. Khoudour, A. Crouzil, and S. A. Velastin, "Sparse LiDAR and Stereo Fusion (SLS-Fusion) for Depth Estimation and 3D Object Detection," Mar. 2021, Accessed: Oct. 16, 2021. [Online]. Available: <https://arxiv.org/abs/2103.03977v3>.
- [266] A. Von Bernuth, G. Volk, and O. Bringmann, "Simulating Photo-realistic Snow and Fog on Existing Images for Enhanced CNN Training and Evaluation," *2019 IEEE Intell. Transp. Syst. Conf. ITSC 2019*, pp. 41–46, Oct. 2019, doi: 10.1109/ITSC.2019.8917367.
- [267] X. Z. Chen, C. M. Chang, C. W. Yu, and Y. L. Chen, "A real-time vehicle detection system under various bad weather conditions based on a deep learning model without retraining," *Sensors (Switzerland)*, vol. 20, no. 20, pp. 1–22, 2020, doi: 10.3390/s20205731.
- [268] G. Li, Y. Yang, X. Qu, D. Cao, and K. Li, "A deep learning based image enhancement approach for autonomous driving at night," *Knowledge-Based Syst.*, vol. 213, p. 106617, 2021, doi: 10.1016/j.knosys.2020.106617.
- [269] Q. Chen, J. Xu, and V. Koltun, "Fast Image Processing with Fully-Convolutional Networks," in *Proceedings of the IEEE International Conference on Computer Vision*, Sep. 2017, vol. 2017-Octob, pp. 2516–2525, Accessed: Oct. 19, 2021. [Online]. Available: <https://arxiv.org/abs/1709.00643v1>.
- [270] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," *Lect. Notes Comput. Sci.*, vol. 9351, pp. 234–241, May 2015, Accessed: Oct. 19, 2021. [Online]. Available: <https://arxiv.org/abs/1505.04597v1>.
- [271] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation," in *Computer Vision -- ECCV 2018*, 2018, pp. 833–851.
- [272] D. J. Jobson, Z. Rahman, and G. A. Woodell, "Properties and performance of a center/surround retinex," *IEEE Trans. Image Process.*, vol. 6, no. 3, pp. 451–462, 1997, doi: 10.1109/83.557356.
- [273] C.-E. Wu, Y.-M. Chan, C.-H. Chen, W.-C. Chen, and C.-S. Chen, "IMMVP: An Efficient Daytime and Nighttime On-Road Object Detector," in *2019 IEEE 21st International Workshop on Multimedia Signal Processing (MMSP)*, 2009, pp. 1–5.

- [274] J.-I. Guo *et al.*, “Summary Embedded Deep Learning Object Detection Model Competition,” in *2019 IEEE 21st International Workshop on Multimedia Signal Processing (MMSP)*, 2019, pp. 1–5, doi: 10.1109/MMSP.2019.8901733.
- [275] A. Bhandari, A. Kafle, P. Dhakal, P. R. Joshi, and D. B. Kshatri, “Image Enhancement and Object Recognition for Night Vision Surveillance,” Jun. 2020, Accessed: Oct. 19, 2021. [Online]. Available: <https://arxiv.org/abs/2006.05787v1>.
- [276] R. H. Rasshofer and K. Gresser, “Automotive radar and lidar systems for next generation driver assistance functions,” *Adv. radio Sci.*, vol. 3, pp. 205–209, 2005.
- [277] J. Wei, J. M. Snider, J. Kim, J. M. Dolan, R. Rajkumar, and B. Litkouhi, “Towards a viable autonomous driving research platform,” in *IEEE Intelligent Vehicles Symposium, Proceedings*, 2013, pp. 763–770, doi: 10.1109/IVS.2013.6629559.
- [278] P. Radecki, M. Campbell, and K. Matzen, “All Weather Perception: Joint Data Association, Tracking, and Classification for Autonomous Ground Vehicles,” May 2016, Accessed: Oct. 19, 2021. [Online]. Available: <https://arxiv.org/abs/1605.02196v1>.
- [279] N. Carlevaris-Bianco and R. M. Eustice, “Learning visual feature descriptors for dynamic lighting conditions,” in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 2769–2776.
- [280] P. Hurney, P. Waldron, F. Morgan, E. Jones, and M. Glavin, “Review of pedestrian detection techniques in automotive far-infrared video,” *IET Intell. Transp. Syst.*, vol. 9, no. 8, pp. 824–832, 2015.
- [281] V. Peretroukhin, W. Vega-Brown, N. Roy, and J. Kelly, “PROBE-GK: Predictive Robust Estimation using Generalized Kernels,” 2017.
- [282] W. Maddern, A. D. Stewart, C. Mcmanus, B. Upcroft, W. Churchill, and P. Newman, “Illumination Invariant Imaging: Applications in Robust Vision-based Localisation, Mapping and Classification for Autonomous Vehicles.”
- [283] X. Ma, Z. Wang, H. Li, P. Zhang, X. Fan, and W. Ouyang, “Accurate Monocular Object Detection via Color-Embedded 3D Reconstruction for Autonomous Driving,” *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2019-Octob, pp. 6850–6859, Mar. 2019, Accessed: Oct. 19, 2021. [Online]. Available: <https://arxiv.org/abs/1903.11444v4>.
- [284] X. Cheng, P. Wang, and R. Yang, “Learning Depth with Convolutional Spatial Propagation Network,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 10, pp. 2361–2379, 2020.

- [285] H. Gao, B. Cheng, J. Wang, K. Li, J. Zhao, and D. Li, "Object Classification Using CNN-Based Fusion of Vision and LIDAR in Autonomous Vehicle Environment," *IEEE Trans. Ind. Informatics*, vol. 14, no. 9, pp. 4224–4230, Sep. 2018, doi: 10.1109/TII.2018.2822828.
- [286] M. Hahner, C. Sakaridis, D. Dai, L. Van Gool, L. V. Be, and E. Zürich, "Fog Simulation on Real LiDAR Point Clouds for 3D Object Detection in Adverse Weather," *Comput. Vis. Pattern Recognit.*, 2021, Accessed: Oct. 16, 2021. [Online]. Available: www.trace.ethz.ch/lidar.
- [287] Y. Zhou and O. Tuzel, "VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4490–4499.
- [288] Y. Yan, Y. Mao, and B. Li, "Second: Sparsely embedded convolutional detection," *Sensors (Basel)*, vol. 18, no. 10, p. 3337, 2018.
- [289] Z. Yang, Y. Sun, S. Liu, X. Shen, and J. Jia, "IPOD: Intensive Point-based Object Detector for Point Cloud," Dec. 2018, Accessed: Oct. 19, 2021. [Online]. Available: <https://arxiv.org/abs/1812.05276v1>.
- [290] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019, pp. 12689–12697.
- [291] S. Shi, X. Wang, and H. Li, "PointRCNN: 3D object proposal generation and detection from point cloud," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019, pp. 770–779.
- [292] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, "Frustum PointNets for 3D Object Detection from RGB-D Data," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2018, pp. 918–927.
- [293] B. Wen and M. Wu, "Study on Pedestrian Detection Based on an Improved YOLOv4 Algorithm," *2020 IEEE 6th Int. Conf. Comput. Commun. ICC3 2020*, pp. 1198–1202, Dec. 2020, doi: 10.1109/ICC351575.2020.9344983.
- [294] D. Misra, "Mish: A Self Regularized Non-Monotonic Activation Function," Aug. 2019, Accessed: Dec. 24, 2021. [Online]. Available: <http://arxiv.org/abs/1908.08681>.
- [295] N. Annapareddy, E. Sahin, S. Abraham, M. M. Islam, M. Depiro, and T. Iqbal, "A

- Robust Pedestrian and Cyclist Detection Method Using Thermal Images,” *2021 IEEE Syst. Inf. Eng. Des. Symp. SIEDS 2021*, Apr. 2021, doi: 10.1109/SIEDS52267.2021.9483730.
- [296] Z. Ahmed, R. Iniyavan, and M. M. P., “Enhanced Vulnerable Pedestrian Detection using Deep Learning,” in *2019 International Conference on Communication and Signal Processing (ICCSP)*, 2019, pp. 971–974, doi: 10.1109/ICCSP.2019.8697978.
- [297] T. T. Feng and H. Y. Ge, “Pedestrian detection based on attention mechanism and feature enhancement with SSD,” in *2020 5th International Conference on Communication, Image and Signal Processing (CCISP)*, 2020, pp. 145–148, doi: 10.1109/CCISP51026.2020.9273507.
- [298] M. García-Venegas, D. A. Mercado-Ravell, L. A. Pinedo-Sánchez, and C. A. Carballo-Monsivais, “On the safety of vulnerable road users by cyclist detection and tracking,” *Mach. Vis. Appl.*, vol. 32, no. 5, p. 109, 2021, doi: 10.1007/s00138-021-01231-4.
- [299] A. Hechri and A. Mtibaa, “Two-stage traffic sign detection and recognition based on SVM and convolutional neural networks,” *IET Image Process.*, vol. 14, no. 5, pp. 939–946, Apr. 2020, doi: 10.1049/IET-IPR.2019.0634.
- [300] S. B. Wali, M. A. Hannan, A. Hussain, and S. A. Samad, “An Automatic Traffic Sign Detection and Recognition System Based on Colour Segmentation, Shape Matching, and SVM,” *Math. Probl. Eng.*, vol. 2015, p. 250461, 2015, doi: 10.1155/2015/250461.
- [301] J. Muller and K. Dietmayer, “Detecting Traffic Lights by Single Shot Detection,” *IEEE Conf. Intell. Transp. Syst. Proceedings, ITSC*, vol. 2018-Novem, pp. 266–273, Dec. 2018, doi: 10.1109/ITSC.2018.8569683.
- [302] A. Fregin, J. Muller, U. Krebel, and K. Diermayer, “The DriveU Traffic Light Dataset: Introduction and Comparison with Existing Datasets,” in *Proceedings - IEEE International Conference on Robotics and Automation*, Sep. 2018, pp. 3376–3383, doi: 10.1109/ICRA.2018.8460737.
- [303] X. Y. Ye, D. S. Hong, H. H. Chen, P. Y. Hsiao, and L. C. Fu, “A two-stage real-time YOLOv2-based road marking detector with lightweight spatial transformation-invariant classification,” *Image Vis. Comput.*, vol. 102, p. 103978, Oct. 2020, doi: 10.1016/J.IMAVIS.2020.103978.
- [304] Y. Wu, Z. Li, Y. Chen, K. Nai, and J. Yuan, “Real-time traffic sign detection and

- classification towards real traffic scene,” *Multimed. Tools Appl.* 2020 7925, vol. 79, no. 25, pp. 18201–18219, Mar. 2020, doi: 10.1007/S11042-020-08722-Y.
- [305] Y. C. Chiu, H. Y. Lin, and W. L. Tai, “Implementation and Evaluation of CNN Based Traffic Sign Detection with Different Resolutions,” *Proc. - 2019 Int. Symp. Intell. Signal Process. Commun. Syst. ISPACS 2019*, Dec. 2019, doi: 10.1109/ISPACS48206.2019.8986319.
- [306] M. Weber, M. Huber, and J. M. Zollner, “HDTLR: A CNN based Hierarchical Detector for Traffic Lights,” *IEEE Conf. Intell. Transp. Syst. Proceedings, ITSC*, vol. 2018-November, pp. 255–260, Dec. 2018, doi: 10.1109/ITSC.2018.8569794.
- [307] Z. Ouyang, J. Niu, Y. Liu, and M. Guizani, “Deep CNN-Based real-time traffic light detector for self-driving vehicles,” *IEEE Trans. Mob. Comput.*, vol. 19, no. 2, pp. 300–313, Feb. 2020, doi: 10.1109/TMC.2019.2892451.
- [308] C. Gu, X. Wu, H. Ma, and L. Yang, “An End-to-End Practical System for Road Marking Detection,” in *Image and Graphics*, 2019, pp. 85–93.
- [309] J. Richter-Klug, P. Mania, G. Kazhoyan, M. Beetz, and U. Frese, “Improving object pose estimation by fusion with a multimodal prior – Utilizing uncertainty-based CNN pipelines for robotics,” *IEEE Robot. Autom. Lett.*, vol. 7, no. 2, pp. 2282–2288, Apr. 2022, doi: 10.1109/LRA.2022.3140450.
- [310] K. Khnissi, C. Ben Jabeur, and H. Seddik, “Implementation of a Compact Traffic Signs Recognition System Using a New Squeezed YOLO,” *Int. J. Intell. Transp. Syst. Res.*, vol. 20, no. 2, pp. 466–482, Aug. 2022, doi: <https://doi.org/10.1007/s13177-022-00304-6>.
- [311] O. Barukab, F. Ali, W. Alghamdi, Y. Bassam, and S. Afzal Khan, “DBP-CNN: Deep learning-based prediction of DNA-binding proteins by coupling discrete cosine transform with two-dimensional convolutional neural network,” *Expert Syst. Appl.*, vol. 197, p. 116729, Jul. 2022, doi: 10.1016/J.ESWA.2022.116729.
- [312] H. Sun, L. Li, M. Xu, Q. Li, and Z. Huang, “Using Minimum Component and CNN for Satellite Remote Sensing Image Cloud Detection,” *IEEE Geosci. Remote Sens. Lett.*, vol. 18, no. 12, pp. 2162–2166, Dec. 2021, doi: 10.1109/LGRS.2020.3014358.
- [313] J. Zhang *et al.*, “Recognition of void defects in airport runways using ground-penetrating radar and shallow CNN,” *Autom. Constr.*, vol. 138, p. 104260, Jun. 2022, doi: 10.1016/J.AUTCON.2022.104260.
- [314] A. Agga, A. Abbou, M. Labbadi, Y. El Houm, and I. H. Ou Ali, “CNN-LSTM: An

- efficient hybrid deep learning architecture for predicting short-term photovoltaic power production,” *Electr. Power Syst. Res.*, vol. 208, p. 107908, Jul. 2022, doi: 10.1016/J.EPSR.2022.107908.
- [315] A. S. Gaikwad and M. El-Sharkawy, “Pruning convolution neural network (squeezenet) using taylor expansion-based criterion,” in *2018 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*, 2018, pp. 1–5, doi: 10.1109/ISSPIT.2018.8705095.
- [316] W. Chen, D. Xie, Y. Zhang, and S. Pu, “All you need is a few shifts: Designing efficient convolutional neural networks for image classification,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2019-June, pp. 7234–7243, 2019, doi: 10.1109/CVPR.2019.00741.
- [317] C. Zhang *et al.*, “ResNet or DenseNet? Introducing Dense Shortcuts to ResNet,” pp. 3549–3558, Oct. 2020, Accessed: Aug. 11, 2021. [Online]. Available: <https://arxiv.org/abs/2010.12496v1>.
- [318] G. Huang, S. Liu, L. van der Maaten, and K. Q. Weinberger, “CondenseNet: An Efficient DenseNet using Learned Group Convolutions,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 2752–2761, Nov. 2017, Accessed: Aug. 23, 2021. [Online]. Available: <https://arxiv.org/abs/1711.09224v2>.
- [319] S. Srinivas and R. V. Babu, “Data-free parameter pruning for Deep Neural Networks,” Jul. 2015, Accessed: Jul. 27, 2021. [Online]. Available: <https://arxiv.org/abs/1507.06149v1>.
- [320] S. Han, J. Pool, J. Tran, and W. J. Dally, “Learning both weights and connections for efficient neural networks,” in *Advances in Neural Information Processing Systems*, 2015, pp. 1135–1143, [Online]. Available: <https://arxiv.org/abs/1506.02626>.
- [321] W. Chen, Y. Zhang, D. Xie, and S. Pu, “A Layer Decomposition-Recomposition Framework for Neuron Pruning towards Accurate Lightweight Networks,” Accessed: Feb. 15, 2022. [Online]. Available: www.aaai.org.
- [322] C. Tai, T. Xiao, Y. Zhang, X. Wang, and E. Weinan, “Convolutional neural networks with low-rank regularization,” in *4th International Conference on Learning Representations (ICLR 2016)*, 2016, vol. 1, no. 2014, pp. 1–11, [Online]. Available: <https://arxiv.org/abs/1511.06067>.
- [323] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, “Speeding-up convolutional neural networks using fine-tuned CP-decomposition,” in *3rd*

- International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015, pp. 1–11.
- [324] W. Wen, C. Xu, C. Wu, Y. Wang, Y. Chen, and H. Li, “Coordinating Filters for Faster Deep Neural Networks,” in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 658–666, doi: 10.1109/ICCV.2017.78.
 - [325] Y. Gong, L. Liu, M. Yang, and L. Bourdev, “Compressing Deep Convolutional Networks using Vector Quantization,” Dec. 2014, Accessed: Jul. 27, 2021. [Online]. Available: <https://arxiv.org/abs/1412.6115v1>.
 - [326] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, “Quantized neural networks: Training neural networks with low precision weights and activations,” *J. Mach. Learn. Res.*, vol. 18, pp. 1–30, 2018.
 - [327] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, “XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks,” in *Computer Vision -- ECCV 2016*, 2016, pp. 525–542, doi: https://doi.org/10.1007/978-3-319-46493-0_32.
 - [328] B. B. Sau and V. N. Balasubramanian, “Deep Model Compression: Distilling Knowledge from Noisy Teachers,” Oct. 2016, Accessed: Jul. 29, 2021. [Online]. Available: <https://arxiv.org/abs/1610.09650v2>.
 - [329] P. Luo, Z. Zhu, Z. Liu, X. Wang, and X. Tang, “Face Model Compression by Distilling Knowledge from Neurons,” in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 2016, pp. 3560–3566.
 - [330] S. Zagoruyko and N. Komodakis, “Paying More Attention to Attention: Improving the Performance of Convolutional Neural Networks via Attention Transfer,” Dec. 2016, Accessed: Jul. 27, 2021. [Online]. Available: <https://arxiv.org/abs/1612.03928v3>.
 - [331] Y. Jeon and J. Kim, “Constructing fast network through deconstruction of convolution,” *Adv. Neural Inf. Process. Syst.*, vol. 2018-Decem, pp. 5951–5961, 2018.
 - [332] C. (Chris M. Bishop, *Neural networks for pattern recognition*. Oxford: Oxford University Press, 1995.
 - [333] B. D. Ripley, *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996.
 - [334] R. K. Srivastava, K. Greff, and J. Schmidhuber, “Training Very Deep Networks,” in *Advances in Neural Information Processing Systems*, 2015, vol. 28, [Online].

Available:

<https://proceedings.neurips.cc/paper/2015/file/215a71a12769b056c3c32e7299f1c5ed-Paper.pdf>.

- [335] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.
- [336] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, “Deep Networks with Stochastic Depth,” in *Computer Vision -- ECCV 2016*, 2016, pp. 646–661.
- [337] D. Yu, J. Yang, Y. Zhang, and S. Yu, “Additive DenseNet: Dense connections based on simple addition operations,” *J. Intell. Fuzzy Syst.*, vol. 40, no. 3, pp. 5015–5025, 2021, doi: 10.3233/JIFS-201758.
- [338] T. Li, W. Jiao, L. N. Wang, and G. Zhong, “Automatic DenseNet sparsification,” *IEEE Access*, vol. 8, pp. 62561–62571, 2020, doi: 10.1109/ACCESS.2020.2984130.
- [339] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, “Efficient BackProp,” in *Neural Networks: Tricks of the Trade: Second Edition*, G. Montavon, G. B. Orr, and K.-R. Müller, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 9–48.
- [340] K. He, X. Zhang, S. Ren, and J. Sun, “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification,” in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1026–1034, doi: 10.1109/ICCV.2015.123.
- [341] K. He, X. Zhang, S. Ren, and J. Sun, “Identity Mappings in Deep Residual Networks,” in *Computer Vision -- ECCV 2016*, 2016, pp. 630–645.
- [342] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, “On the importance of initialization and momentum in deep learning,” in *Proceedings of the 30th International Conference on Machine Learning*, 2013, vol. 28, no. 3, pp. 1139–1147, [Online]. Available: <https://proceedings.mlr.press/v28/sutskever13.html>.
- [343] S. Narang *et al.*, “Mixed precision training,” 2018, doi: 10.48550/arxiv.1710.03740.
- [344] X. Ma, G. Yuan, S. Lin, Z. Li, H. Sun, and Y. Wang, “ResNet Can Be Pruned 60x: Introducing Network Purification and Unused Path Removal (P-RM) after Weight Pruning,” Apr. 2019, Accessed: Jun. 25, 2022. [Online]. Available: <http://arxiv.org/abs/1905.00136>.
- [345] M. Yin, Y. Sui, S. Liao, and B. Yuan, “Towards Efficient Tensor Decomposition-

- Based DNN Model Compression with Optimization Framework,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 10669–10678, Jul. 2021, doi: 10.48550/arxiv.2107.12422.
- [346] S. Young, Z. Wang, D. Taubman, and B. Girod, “Transform Quantization for CNN Compression,” *IEEE Trans. Pattern Anal. Mach. Intell.*, 2021, doi: 10.1109/TPAMI.2021.3084839.
- [347] J. Kim, Y. Bhalgat, J. Lee, C. Patel, and N. Kwak, “QKD: Quantization-aware Knowledge Distillation,” 2019, doi: 10.48550/arxiv.1911.12491.
- [348] S. Lin, R. Ji, C. Chen, D. Tao, and J. Luo, “Holistic CNN Compression via Low-Rank Decomposition with Knowledge Transfer,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 12, pp. 2889–2905, Dec. 2019, doi: 10.1109/TPAMI.2018.2873305.
- [349] Y. He, X. Zhang, and J. Sun, “Channel Pruning for Accelerating Very Deep Neural Networks,” in *Proceedings of the IEEE International Conference on Computer Vision*, Dec. 2017, vol. 2017-October, pp. 1398–1406, doi: 10.1109/ICCV.2017.155.
- [350] T. Rice, “Real Time Inference on Raspberry Pi 4 (30 fps!),” *PyTorch*, 2022. https://pytorch.org/tutorials/intermediate/realtime_rpi.html (accessed Jun. 01, 2022).
- [351] Arm, “Smartphones.” <https://www.arm.com/products/smartphones> (accessed Jun. 01, 2022).
- [352] X. Sun *et al.*, “High spectral response of self-driven GaN-based detectors by controlling the contact barrier height,” *Sci. Rep.*, vol. 5, no. 1, p. 16819, 2015, doi: 10.1038/srep16819.
- [353] J. Sun, X. Cao, H. Liang, W. Huang, Z. Chen, and Z. Li, “New Interpretations of Normalization Methods in Deep Learning,” *AAAI 2020 - 34th AAAI Conf. Artif. Intell.*, pp. 5875–5882, Jun. 2020, Accessed: Jul. 27, 2021. [Online]. Available: <https://arxiv.org/abs/2006.09104v1>.
- [354] R. Moradi, R. Berangi, and B. Minaei, “A survey of regularization strategies for deep models,” *Artif. Intell. Rev.*, vol. 53, no. 6, pp. 3947–3986, 2020, doi: 10.1007/s10462-019-09784-7.
- [355] Q. Wang, Y. Ma, K. Zhao, and Y. Tian, “A Comprehensive Survey of Loss Functions in Machine Learning,” *Ann. Data Sci.*, no. 0123456789, 2020, doi: 10.1007/s40745-020-00253-5.

- [356] D. Yu, H. Zhang, W. Chen, J. Yin, and T.-Y. Liu, “How Does Data Augmentation Affect Privacy in Machine Learning?,” Jul. 2020, Accessed: Jul. 27, 2021. [Online]. Available: <https://arxiv.org/abs/2007.10567v3>.
- [357] H. Robbins and S. Monro, “A Stochastic Approximation Method,” *Ann. Math. Stat.*, vol. 22, no. 3, pp. 400–407, 1951, doi: 10.1214/aoms/1177729586.
- [358] P. Jain, P. Netrapalli, S. M. Kakade, R. Kidambi, and A. Sidford, “Parallelizing stochastic gradient descent for least squares regression: Mini-batching, averaging, and model misspecification,” *J. Mach. Learn. Res.*, vol. 18, pp. 1–42, 2018.
- [359] Y. Nesterov, “A method for unconstrained convex minimization problem with the rate of convergence $o(1/k^2)$,” 1983.
- [360] J. C. Duchi, P. L. Bartlett, and M. J. Wainwright, “Randomized smoothing for (parallel) stochastic optimization,” in *Proceedings of the IEEE Conference on Decision and Control*, 2012, vol. 12, pp. 5442–5444, doi: 10.1109/CDC.2012.6426698.
- [361] M. D. Zeiler, “ADADELTA: An Adaptive Learning Rate Method,” Dec. 2012, Accessed: Jul. 27, 2021. [Online]. Available: <https://arxiv.org/abs/1212.5701v1>.
- [362] Tijmen Tieleman and G. Hinton, “Divide the gradient by a running average of its recent magnitude,” *Coursera Neural networks Mach. Learn.*, vol. 4, no. 2, pp. 26–31, 2012.
- [363] N. Le Roux, M. Schmidt, and F. Bach, “A stochastic gradient method with an exponential convergence rate for finite training sets,” in *Advances in Neural Information Processing Systems*, 2012, vol. 4, no. 2, pp. 2663–2671.
- [364] R. Johnson and T. Zhang, “Accelerating stochastic gradient descent using predictive variance reduction,” in *Advances in Neural Information Processing Systems*, 2013, vol. 1, no. 3, pp. 1–9.
- [365] D. F. Shanno, “Conditioning of Quasi-Newton Methods for Function Minimization,” *Math. Comput.*, vol. 24, no. 111, pp. 647–656, 1970.
- [366] L. Devroye, J. King, and C. McDiarmid, “Random Hyperplane Search Trees,” *SIAM J. Comput.*, vol. 38, no. 6, pp. 2411–2425, 2009.
- [367] J. Pajarinen, H. L. Thai, R. Akrou, J. Peters, and G. Neumann, “Compatible natural gradient policy search,” *Mach. Learn.*, vol. 108, no. 8–9, pp. 1443–1466, 2019, doi: 10.1007/s10994-019-05807-0.
- [368] J. Martens, “Deep learning via Hessian-free optimization,” *ICML 2010 - Proceedings, 27th Int. Conf. Mach. Learn.*, pp. 735–742, 2010.

- [369] J. E. Dennis and Jorge J. Moree, “Quasi-Newton Methods , Motivation and Theory,” *Soc. Ind. Appl. Math.*, vol. 19, no. 1, pp. 46–89, 2010, [Online]. Available: <https://www.jstor.org/stable/2029325>.
- [370] F. Roosta-Khorasani and M. W. Mahoney, “Sub-Sampled Newton Methods II: Local Convergence Rates,” 2016, [Online]. Available: <http://arxiv.org/abs/1601.04738>.
- [371] P. Xu, J. Yang, F. Roosta-Khorasani, C. Ré, and M. W. Mahoney, “Sub-sampled Newton methods with non-uniform sampling,” in *Advances in Neural Information Processing Systems*, 2016, pp. 3008–3016.
- [372] R. Bollapragada, R. H. Byrd, and J. Nocedal, “Exact and inexact subsampled Newton methods for optimization,” *IMA J. Numer. Anal.*, vol. 39, no. 2, pp. 545–548, 2019, doi: 10.1093/imanum/dry009.
- [373] M. R. Hestenes and E. Stiefel, “Methods of conjugate gradients for solving linear systems,” *J. Res. Natl. Bur. Stand. (1934).*, vol. 49, no. 6, p. 409, 1952, doi: 10.6028/jres.049.044.
- [374] S. Akram and Q. ul Ann, “Newton Raphson Method,” *Int. J. Sci. Eng. Res.*, vol. 6, no. 7, pp. 1748–1752, 2015.
- [375] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. New York, NY, USA: Springer New York, 2006.
- [376] L. Bottou, F. E. Curtis, and J. Nocedal, “Optimization Methods for Large-Scale Machine Learning,” <https://doi.org/10.1137/16M1080173>, vol. 60, no. 2, pp. 223–311, May 2018, doi: 10.1137/16M1080173.
- [377] R. H. Byrd, G. M. Chin, W. Neveitt, and N. Jorge, “On the use of stochastic hessian information in optimization methods for machine learning,” *SIAM J. Optim.*, vol. 21, no. 3, pp. 977–995, 2011, doi: 10.1137/10079923X.
- [378] S. I. Amari, “Natural Gradient Works Efficiently in Learning,” *Neural Comput.*, vol. 10, no. 2, pp. 251–276, Feb. 1998, doi: 10.1162/089976698300017746.
- [379] L. M. Rios and N. V. Sahinidis, “Derivative-free optimization: A review of algorithms and comparison of software implementations,” *J. Glob. Optim.*, vol. 56, no. 3, pp. 1247–1293, 2013, doi: 10.1007/s10898-012-9951-y.
- [380] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, “Optimization by simulated annealing,” *Science (80-.)*, vol. 220, no. 4598, pp. 671–680, 1983, doi: 10.1126/science.220.4598.671.
- [381] R. Fletcher and E. S. de la Maza, “Nonlinear programming and nonsmooth

- optimization by successive linear programming,” *Mathematical Programming*, vol. 43, no. 1–3, pp. 235–256, 1989, doi: 10.1007/BF01582292.
- [382] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, “Compressing neural networks with the hashing trick,” in *Proceedings of the 32nd International Conference on International Conference on Machine Learning*, 2015, vol. 37, pp. 2275–2284, [Online]. Available: <https://arxiv.org/abs/1504.04788>.
- [383] X. Wu, Y. Wu, and Y. Zhao, “Binarized Neural Networks on the ImageNet Classification Task,” Apr. 2016, Accessed: Jul. 29, 2021. [Online]. Available: <https://arxiv.org/abs/1604.03058v5>.
- [384] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, “Quantized convolutional neural networks for mobile devices,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 4820–4828, doi: 10.1109/CVPR.2016.521.
- [385] V. Vanhoucke, A. Senior, and M. Mao, “Improving the speed of neural networks on CPUs,” in *Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011*, 2011, pp. 1–8.
- [386] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, “Deep learning with limited numerical precision,” in *In Proceedings of the 32nd International Conference on International Conference on Machine Learning*, 2015, vol. 37, pp. 1737–1746.
- [387] S. Han, H. Mao, and W. J. Dally, “Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding,” Oct. 2015, Accessed: Jul. 27, 2021. [Online]. Available: <https://arxiv.org/abs/1510.00149v5>.
- [388] Y. Choi, M. El-Khamy, and J. Lee, “Towards the Limit of Network Quantization,” *5th Int. Conf. Learn. Represent. ICLR 2017 - Conf. Track Proc.*, Dec. 2016, Accessed: Jul. 27, 2021. [Online]. Available: <https://arxiv.org/abs/1612.01543v2>.
- [389] S. B. Kotsiantis, “Supervised machine learning: a review of classification techniques,” *Informatica*, vol. 31, no. 3, pp. 249–268, 2007.
- [390] Y. Bengio, A. Courville, and P. Vincent, “Representation learning: A review and new perspectives,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1798–1828, 2013, doi: 10.1109/TPAMI.2013.50.
- [391] V. François-lavet *et al.*, “An Introduction to Deep Reinforcement Learning,” *Found. trends Mach. Learn.*, vol. 2, no. 3, pp. 1–140, 2018, doi: 10.1561/22000000071.Vincent.
- [392] L. Jing and Y. Tian, “Self-supervised Visual Feature Learning with Deep Neural

- Networks: A Survey,” *IEEE Trans. Pattern Anal. Mach. Intell.*, pp. 1–1, May 2020, doi: 10.1109/TPAMI.2020.2992393.
- [393] K. Huang, Z. Xu, I. King, M. R. Lyu, and C. Campbell, “Supervised Self-taught Learning: Actively transferring knowledge from unlabeled data,” in *2009 International Joint Conference on Neural Networks*, 2009, pp. 1272–1277, doi: 10.1109/IJCNN.2009.5178647.
- [394] B. Settles, “Active Learning Literature Survey,” *Mater. Lett.*, vol. 65, no. 5, pp. 854–856, 2011, doi: 10.1016/j.matlet.2010.11.072.
- [395] S. J. Pan and Q. Yang, “A survey on transfer learning,” *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, 2010, doi: 10.1109/TKDE.2009.191.
- [396] K. Bonawitz *et al.*, “Towards Federated Learning at Scale: System Design,” Feb. 2019, Accessed: Jul. 27, 2021. [Online]. Available: <https://arxiv.org/abs/1902.01046v2>.
- [397] D. Lowd and C. Meek, “Adversarial learning,” in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2005, pp. 641–647, doi: 10.1145/1081870.1081950.
- [398] Y. Zhang and Q. Yang, “A Survey on Multi-Task Learning,” *IEEE Trans. Knowl. Data Eng.*, 2021, doi: 10.1109/TKDE.2021.3070203.
- [399] T. Baltrusaitis, C. Ahuja, and L. P. Morency, “Multimodal Machine Learning: A Survey and Taxonomy,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 2, pp. 423–443, Feb. 2019, doi: 10.1109/TPAMI.2018.2798607.